Coping with Syntactic Ambiguity or How to Put the Block in the Box on the Table[1]

Kenneth Church
Ramesh Patil

Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139

Sentences are far more ambiguous than one might have thought. There may be hundreds, perhaps thousands, of syntactic parse trees for certain very natural sentences of English. This fact has been a major problem confronting natural language processing, especially when a large percentage of the syntactic parse trees are enumerated during semantic/pragmatic processing. In this paper we propose some methods for dealing with syntactic ambiguity in ways that exploit certain regularities among alternative parse trees. These regularities will be expressed as *linear combinations* of ATN networks, and also as sums and products of formal power series. We believe that such encoding of ambiguity will enhance processing, whether syntactic and semantic constraints are processed separately in sequence or interleaved together.

Most parsers find the set of parse trees by starting with the empty set and adding to it each time they find a new possibility. We make the observation that in certain situations it would be much more efficient to work in the other direction, starting from the universal set (i.e, the set of all binary trees) and ruling trees out when the parser decides that they cannot be parses. Ruling-out is easier when the set of parse trees is closer to the universal set and ruling-in is easier when the set of parse trees is closer to the empty set. Ruling-out is particularly suited for *"every way ambiguous"* constructions such as prepositional phrases that have just as many parse trees as there are binary trees over the terminal elements. Since every tree is a parse, the parser doesn't have to rule any of them out.

In some sense, this is a formalization of an idea that has been in the literature for some time. That is, it has been noticed for a long time that these sorts of very ambiguous constructions are very difficult for most parsing algorithms, but (apparently) not for people. This observation has led some researchers to hypothesize additional parsing mechanisms, such as pseudo-attachment (Church 1980, pp. 65-71)[2] and permanent predictable ambiguity (Sager 1973), so that the parser could "attach all ways" in a single step. However, these mechanisms have always lacked a precise interpretation; we will present a much more formal way of coping with "every way ambiguous" grammars, defined in terms of *Catalan numbers* (Knuth 1975, pp. 388-389, 531-533).

1. Ambiguity is a Practical Problem

Sentences are far more ambiguous than one might have thought. Our experience with the EQSP parser (Martin, Church, and Patil 1981) indicates that there may be hundreds, perhaps thousands, of syntactic parse trees for certain very natural sentences of English. For example, consider the following sentence with two prepositional phrases:

[1] This research was supported (in part) by the National Institutes of Health Grant No. 1 P01 LM 03374-02 from the National Library of Medicine, and by the Defense Advanced Research Projects Agency (DOD) monitored by the Office of Naval Research under Contract No. N00014-75-C-0661.

[2] The idea of pseudo-attachment was first proposed by Marcus (private communication), though Marcus does not accept the formulation in Church 1980.

0362-613X/82/030139-11$03.00

(1) Put the block in the box on the table.

which has two interpretations:

(2a) Put the block[in the box on the table]
(2b) Put [the block in the box] on the table.

These syntactic ambiguities grow "combinatorially" with the number of prepositional phrases. For example, when a third PP is added to the sentence above, there are five interpretations:

(3a) Put the block [[in the box on the table] in the kitchen].
(3b) Put the block [in the box [on the table in the kitchen]].
(3c) Put [[the block in the box] on the table] in the kitchen.
(3d) Put [the block [in the box on the table]] in the kitchen.
(3e) Put [the block in the box] [on the table in the kitchen].

When a fourth PP is added, there are fourteen trees, and so on. This sort of combinatoric ambiguity has been a major problem confronting natural language processing. In this paper we propose some methods for dealing with syntactic ambiguity in ways that take advantage of regularities among the alternative parse trees.

In particular, we observe that enumerating the parse trees as above fails to capture the important generalization that prepositional phrases are "every way ambiguous," or more precisely, the set of parse trees over *i* PPs is the same as the set of binary trees that can be constructed over *i* terminal elements. Notice, for example, that there are two possible binary trees over three elements,

(4a) [... block ... [... box ... table ...]]
(4b) [[... block ... box ...] ... table ...]

corresponding to (2a) and (2b), respectively, and that there are five binary trees over four elements corresponding to (3a)–(3c), respectively.

PPs, adjuncts, conjuncts, noun-noun modification, stack relative clauses, and other "every way ambiguous" constructions will be treated as primitive objects. They can be combined in various ways to produce composite constructions, such as lexical ambiguity, which may also be very ambiguous but not necessarily "every way ambiguous." Lexical ambiguity, for example, will be analyzed as the sum of its senses, or in flow graph terminology (Oppenheim and Schafer 1975) as a parallel connection of its senses. Structural ambiguity, on the other hand, will be analyzed as the product of its components, or in flow graph terminology as a series connection.

2. Formal Power Series

This section will make the linear systems analogy more precise by relating context-free grammars to formal power series (polynominals). Formal power series are a well-known device in the formal language literature (e.g., Salomaa 1973) for developing the algebraic properties of context-free grammars. We introduce them here to establish a formal basis for our upcoming discussion of processing issues.

The power series for grammar (5a) is (5b).

(5a) NP → John | NP and NP
(5b) NP = John + John and John
+ 2John and John and John
+ 5John and John and John and John
+ 14John and John and John and John and John + ...

Each term consists of a *sentence* generated by the grammar and an *ambiguity coefficient*[3] which counts how many ways the sentence can be generated. For example, the sentence "John" has one parse tree

(6a) [John] *1 tree*

because the zero-th coefficient of the power series is one. Similarly, the sentence "John and John" also has one tree because its coefficient is one,

(6b) [John and John] *1 tree*

and "John and John and John" has two because its coefficient is two,

(6c) [[John and John] and John], [John and [John and John]] *2 trees*

and "John and John and John and John" has five,

(6d) [John and [[John and John] and John]], *5 trees*
[John and [John and [John and John]]],
[[[John and John], and John] and John],
[[John and [John and John]] and John],
[[John and John] and [John and John]]

and so on. The reader can verify for himself that "John and John and John and John and John" has fourteen trees.

Note that the power series encapsulates the ambiguity response of the system (grammar) to all possible input sentences. In this way, the power series is analogous to the impulse response in electrical engineering, which encapsulates the response of the system (circuit) to all possible input frequencies. (Ambiguity coefficients bear a strong resemblance to frequency coefficients in Fourier analysis.) All of these transformed representation systems (e.g., power series, impulse response, and Fourier series) provide a complete description of the system with no loss of information[4] (and no heuristic approximations, for example, search strategies (Kaplan 1972)). Trans-

[3] The formal language literature (Harrison 1978, Salomaa 1973) uses the term *support* instead of ambiguity coefficient.

forms are often very useful because they provide a different point of view. Certain observations are more easily seen in the transform space than in the original space, and vice versa.

This paper will discuss several ways to generate the power series. Initially let us consider successive approximation. Of all the techniques to be presented here, successive approximations most closely resembles the approach taken by most current chart parsers including EQSP (Martin, Church, and Patil 1981). The alternative approaches take advantage of certain regularities in the power series in order to produce the same results more efficiently.

Successive approximation works as follows. First we translate grammar (5a) into the equation:

(7) $NP = John + NP \cdot and \cdot NP$

where "+" connects two ways of generating an NP and "·" concatenates two parts of an NP. In some sense, we want to "solve" this equation for NP. This can be accomplished by refining successive approximations. An initial approximation NP_0 is formed by taking NP to be the empty language,

(8a) $NP_0 = 0$

Then we form the next approximation by substituting the previous approximation into equation (7), and simplifying according to the usual rules of algebra (e.g., assuming distributivity, associativity,[5] identity element, and zero element).

(8b) $NP_1 = John + NP_0 \cdot and \cdot NP_0 = John + 0 \cdot and \cdot 0 = John$

We continue refining the approximation in this way.

(8c) $NP_2 = John + NP_1 \cdot and \cdot NP_1 = John + John\ and\ John$

(8d) $NP_3 = John + NP_2\ and\ NP_2$
= John + (John + John and John) · and · (John + John and John)
= John + John and John
+ John and John and John
+ John and John and John
+ John and John and John and John
= John + John and John
+ 2 John and John and John
+ John and John and John and John
⋮

Eventually, we have NP expressed as an infinitely long polynominal (5b) above. This expression can be simplified by introducing a notation for exponentiation. Let x^i be an abbreviation for multiplying $x \cdot x \cdot \ldots \cdot x$, i times.

(9) NP = John + John and John
+ 2 John $(\text{and John})^2$
+ 5 John $(\text{and John})^3$
+ 14 John $(\text{and John})^4$
+ ...

Note that parentheses are interpreted differently in algebraic equations than in context-free rules. In context-free rules, parentheses denote optionality, where in equations they denote precedence relations among algebraic operations.

3. Catalan Numbers

Ambiguity coefficients take on an important practical significance when we can model them directly without resorting to successive approximation as above. This can result in substantial time and space savings in certain special cases where there are much more efficient ways to compute the coefficients than successive approximation (chart parsing). Equation (9) is such a special case; the coefficients follow a well-known combinatoric series called the *Catalan Numbers* (Knuth 1975, pp. 388-389, 531-533).[6] This section will describe Catalan numbers and their relation to parsing.

The first few Catalan numbers are 1, 1, 2, 5, 14, 42, 132, 469, 1430, 4862. They are generated by the closed form expression:[7]

(10) $$Cat_n = \binom{2n}{n} - \binom{2n}{n-1}$$

This formula can be explained in terms of parenthesized expressions, which are equivalent to trees. Cat_n is the number of ways to parenthesize a formula of length *n*. There are two conditions on parenthesization: (a) there must be the same number of open and close parentheses, and (b) they must be properly nested so that an open parenthesis precedes its matching close parenthesis. The first term counts the number of

[4] This needs a qualification. It is true that the power series provides a complete description of the ambiguity response to any input sentence. However, the power series representation may be losing some information that would be useful for parsing. In particular, there might be some cases where it is impossible to recover the parse trees exactly, as we will see, though this may not be too serious a problem for many practical applications. That is, it is often possible to recover most (if not all) of the structure, which may be adequate for many applications.

[5] The careful reader may correctly object to this assumption. We include it here for expository convenience, as it greatly simplifies the derivations though it should be noted that many of the results could be derived without the assumption. Furthermore, this assumption is valid for counting ambiguity. That is, $|A \cdot B| * |C| = |A| * |B \cdot C|$, where A, B, and C are sets of trees and $|A|$ denotes the number of members of A, and * is integer multiplication.

[6] This fact was first pointed out to us by V. Pratt. We suspect that it is a generally well-known result in the formal language community, though its origin is unclear.

[7] $\binom{a}{b}$ is known as a *binominal coefficient.* It is equivalent to $\{a!/[b!(a-b)!]\}$, where a! is equal to the product of all integers between 1 and a. Binomial coefficients are very common in combinatorics where they are interpreted as the number of ways to pick b objects out of a set of a objects.

sequences of $2n$ parentheses, such that there are the same number of opens and closes. The second term subtracts cases violating condition (b). This explanation is elaborated in Knuth 1975, p. 531.

It is very useful to know that the ambiguity coefficients are Catalan numbers because this observation enables us to replace equation (9) with (11), where Cat_i denotes the ith Catalan number. (All summations range from 0 to ∞ unless noted otherwise.)

(11) $NP = \sum_i Cat_i \; John \; (and \; John)^i$

The ith Catalan number is the number of binary trees that can be constructed over i phrases. This theoretical model correctly predicts our practical experience with EQSP. EQSP found exactly the Catalan number of parse trees for each sentence in the following sequence.

1	It was the number.
1	It was the number of products.
2	It was the number of products of products.
5	It was the number of products of products of products.
14	It was the number of products of products of products of products.
⋮	

These predictions continue to hold with as many as nine prepositional phrases (4862 parse trees).

4. Table Lookup

We could improve EQSP's performance on PPs if we could find a more efficient way to compute Catalan numbers than chart parsing, the method currently employed by EQSP. Let us propose two alternatives: table lookup and evaluating expression (10) directly. Both are very efficient over practical ranges of n, say no more than 20 phrases or so.[8] In both cases, the ambiguity of a sentence in grammar (5a) can be determined by counting the number of occurrences of "and John" and then retrieving the Catalan of that number. These approaches both take linear time (over practical ranges of n),[9] whereas chart parsing requires cubic time to parse sentences in these grammars, a significant improvement.

So far we have shown how to compute in linear time the number of ambiguous interpretations of a sentence in an "every way ambiguous" grammar. However, we are really interested in finding parse trees, not just the number of ambiguous interpretations. We could extend the table lookup algorithm to find trees rather than ambiguity coefficients, by modifying the table to store trees instead of numbers. For parsing purposes, Cat_i can be thought of as a pointer to the ith entry of the table. So, for a sentence in grammar (5a), for example, the machine could count the number of occurrences of "and John" and then retrieve the table entry for that number.

index	*trees*
0	{[John]}
1	{[John and John]}
2	{[[John and John] and John], [John and [John and John]]}
	⋮

The table would be more general if it did not specify the lexical items at the leaves. Let us replace the table above with

index	*trees*
0	{[x]}
1	{[x x]}
2	{[[x x] x], [x [x x]]}
	⋮

and assume the machine can bind the x's to the appropriate lexical items.

There is a real problem with this table lookup machine. The parse trees may not be exactly correct because the power series computation assumed that multiplication was associative, which is an appropriate assumption for computing ambiguity, but inappropriate for constructing trees. For example, we observed that prepositional phrases and conjunction are both "every way ambiguous" grammars because their ambiguity coefficients are Catalan numbers. However, it is not the case that they generate exactly the same parse trees.

Nevertheless we present the table lookup pseudo-parser here because it seems to be a speculative new approach with considerable promise. It is often more efficient than a real parser, and the trees that it finds may be just as useful as the correct one for many practical purposes. For example, many speech recognition projects employ a parser to filter out syntactically inappropriate hypotheses. However, a full parser is not really necessary for this task; a recognizer such as this table lookup pseudo-parser may be perfectly adequate for this task. Furthermore, it is often possible to recover the correct trees from the output of the pseudo-parser. In particular, the difference between prepositional phrases and conjunction could be accounted for by modifying the interpretation of the PP category label, so that the trees would be interpreted correctly even though they are not exactly correct.

[8] The table lookup scheme ought to have a way to handle the theoretical possibility that there are an unlimited number of prepositional phrases. The table lookup routine will employ a more traditional parsing algorithm (e.g., Earley's algorithm) when the number of phrases in the input sentence is not stored in the table.

[9] The linear time result depends on the assumption that table lookup (or closed form computation) can be performed in constant time. This may be a fair assumption over practical ranges of n, but it is not true in general.

The table lookup approach works for primitive grammars. The next two sections show how to decompose composite grammars into series and parallel combinations of primitive grammars.

(12a) $G = G_1 \cdot G_2$ *series*
(12b) $G = G_1 + G_2$ *parallel*

5. Parallel Decomposition

Parallel decomposition can be very useful for dealing with lexical ambiguity, as in

(13) ...to total with products near profits...

where "total" can be taken as a noun or as a verb, as in:

(14a) The accountant brought the daily sales to total with products near profits organized according to the new law. *noun*

(14b) The daily sales were ready for the accountant to total with products near profits organized according to the new law. *verb*

The analysis of these sentences makes use of the additivity property of linear systems. That is, each case, (14a) and (14b), is treated separately, and then the results are added together. Assuming "total" is a noun, there are three prepositional phrases contributing Cat_3 bracketings, and assuming it is a verb, there are two prepositional phrases for Cat_2 ambiguities. Combining the two cases produces $Cat_3 + Cat_2 = 5 + 2 = 7$ parses. Adding another prepositional phrase yields $Cat_4 + Cat_3 = 14 + 5 = 19$ parses. (EQSP behaved as predicted in both cases.)

This behavior is generalized by the following power series:

(15) $\begin{Bmatrix} P\ N \\ to\ V \end{Bmatrix} \sum_i (Cat_{i+1} + Cat_i)(P\ N)^i$

which is the sum of the two cases:

(16a) $\sum_{i>0} Cat_i (P\ N)^i = P\ N \sum_i Cat_{i+i} (P\ N)^i$ *noun*

(16b) $to\ V \sum_i Cat_i (P\ N)^i$ *verb*

This observation can be incorporated into the table lookup pseudo-parser outlined above. Recall that Cat_i is interpreted as the i^{th} index in a table containing all binary trees dominating i leaves. Similarly, $Cat_i + Cat_{i+1}$ will be interpreted as an instruction to "append" the i^{th} entry and $i+1^{th}$ entry of the table.[10]

```
(17) (ADD-TREES
          (CAT-TABLE i)
          (CAT-TABLE(+ i 1)))
```

Let us consider a system where syntactic processing strictly precedes semantic and pragmatic processing. In such a system, how could we incorporate semantic and pragmatic heuristics once we have already parsed the input sentence and found that it was the sum of two Catalans? The parser can simply subtract the inappropriate interpretations. If the oracle says that "total" is a verb, then (16a) would be subtracted from the combined sum, and if the oracle says that "total" is a noun, then (16b) would be subtracted.

On the other hand, our analysis is also useful in a system that interleaves syntactic processing with semantic and pragmatic processing. Suppose that we had a semantic routine that could disambiguate "total," but only at a very high cost in execution time. We need a way to estimate the usefulness of executing the semantic routine so that we don't spend the time if it is not likely to pay off. The analysis above provides a very simple way to estimate the benefit of disambiguating "total." If it turns out to be a verb, then (16a) trees have been ruled out, and if it turns out to be a noun, then (16b) trees have been ruled out. We prefer our declarative algebraic approach over procedural heuristic search strategies (e.g., Kaplan 1972) because we do not have to specify the order of evaluation. We can delay the binding of decisions until the most opportune moment.

6. Series Decomposition

Suppose we have a non-terminal S that is a series combination of two other non-terminals, NP and VP. By inspection, the power series of S is:

(18) $S = NP \cdot VP$

This result is easily verified when there is an unmistakable dividing point between the subject and the predicate. For example, the verb "is" separates the PPs in the subject from those in the predicate in (19a), but not in (19b).

(19a) The number of products over sales of ... *is* near the number of sales under ... *clearly divided*

(19b) *Is* the number of products over sales of ... near the number of sales under ...? *not clearly divided*

In (19a), the total number of parse trees is the product of the number of ways of parsing the subject times the number of ways of parsing the predicate. Both the subject and the predicate produce a Catalan number of parses, and hence the result is the product of two Catalan numbers, which was verified by EQSP (Martin, Church, and Patil 1981, p. 53). This result can be formalized in terms of the power series:

(20) $\left(N \sum_i Cat_i (P\ N)^i\right)\left(is \sum_j Cat_j (P\ N)^j\right)$

which is formed by taking the product of the two subcases:

(21a) $N \sum_i Cat_i (P\ N)^i$ *subject*

(21b) $is \sum_j Cat_j (P\ N)^j$ *predicate*

[10] This can be implemented efficiently, given an appropriate representation of sets of trees.

The power series says that the ambiguity of a particular sentence is the product of Cat_i and Cat_j, where *i* is the number of PPs before "is" and *j* is the number after "is." This could be incorporated in the table lookup parser as an instruction to "multiply" the ith entry in the table by the jth entry. Multiplication is a cross-product operation; L × R generates the set of binary trees whose left sub-tree $\underline{l}$ is from $\underline{L}$ and whose right sub-tree $\underline{r}$ is from $\underline{R}$.

(22) $L \times R = \{(l, r) \mid l \in L \ \& \ r \in R\}$

This is a formal definition. For practical purposes, it may be more useful for the parser to output the list in the factored form:

```
(23) (MULTIPLY-TREES
          (CAT-TABLE i)
          (CAT-TABLE j))
```

which is much more concise than a list of trees. It is possible, for example, that semantic processing can take advantage of factoring, capturing a semantic generalization that holds across all subjects or all predicates. Imagine, for example, that there is a semantic agreement constraint between predicates and arguments. For example, subjects and predicates might have to agree on the feature ±human. Suppose that we were given sentences where this constraint was violated by all ambiguous interpretations of the sentence. In this case, it would be more efficient to employ a feature vector scheme (Dostert and Thompson 1971) which propagates the features in factored form. That is, it computes a feature vector for the union of all possible subjects, and a vector for the union of all possible VPs, and then compares (intersects) these vectors to check if there are any interpretations that meet the constraint. A system such as this, which keeps the parses in factored form, is much more efficient than one that multiplies them out. Even if semantics cannot take advantage of the factoring, there is no harm in keeping the representation in factored form, because it is straightforward to expand (23) into a list of trees (though it may be somewhat slow).

This example is relatively simple because "is" helps the parser determine the value of *i* and *j*. Now let us return to example (19b) where "is" does not separate the two strings of PPs. Again, we determine the power series by multiplying the two subcases:

(24) $$\text{is } \left(N \sum_i Cat_i (P\ N)^i\right) \left(\sum_j Cat_j (P\ N)^j\right)$$
$$= \text{is } N \sum_i \sum_j Cat_i\ Cat_j (P\ N)^{i+j}$$

However, this form is not so useful for parsing because the parser cannot easily determine *i* and *j*, the number of prepositional phrases in the subject and the number in the predicate. It appears the parser will have to compute the product of two Catalans for each way of picking *i* and *j*, which is somewhat expensive.[11] Fortunately, the Catalan function has some special properties so that it is possible algebraically to remove the references to *i* and *j*. In the next section we show how this expression can be reformulated in terms of *n*, the total number of PPs.

6.1 Auto-Convolution of Catalan Grammars

Some readers may have noticed that expression (24) is in *convolution* form. We will make use of this in the reformulation. Notice that the Catalan series is a fixed point under *auto-convolution* (except for a shift); that is, multiplying a Catalan power series (i.e., $1 + x + 2x^2 + 5x^3 + 14x^4 + \ldots\ Cat_i x^i \ldots$) with itself produces another polynomial with Catalan coefficients.[12] The multiplication is worked out for the first few terms.

$$\begin{array}{rrrrrrl}
 & 1\ + & x\ + & 2x^2\ + & 5x^3\ + & 14x^4 & + \ldots \\
\times & 1\ + & x\ + & 2x^2\ + & 5x^3\ + & 14x^4 & + \ldots \\
\hline
 & 1\ + & x\ + & 2x^2\ + & 5x^3\ + & 14x^4 & + \ldots \\
 & & x\ + & x^2\ + & 2x^3\ + & 5x^4 & + \ldots \\
 & & & 2x^2\ + & 2x^3\ + & 4x^4 & + \ldots \\
 & & & & 5x^3\ + & 5x^4 & + \ldots \\
+ & & & & & 14x^4 & + \ldots \\
\hline
 & 1 & +\ 2x\ + & 5x^2\ + & 14x^3\ + & 42x^4 & + \ldots
\end{array}$$

This property can be summarized as:

(25) $$\sum_i Cat_i\ x^i \sum_j Cat_j\ x^j = \sum_n Cat_{n+1}\ x^n$$

where n equals i+j.

Intuitively, this equation says that if we have two "every way ambiguous" (Catalan) constructions, and we combine them in every possible way (convolution), the result is an "every way ambiguous" (Catalan) construction. With this observation, equation (24) reduces to:

(26) $$\text{is } \left(N \sum_i Cat_i (P\ N)^i\right)\left(\sum_j Cat_j (P\ N)^j\right)$$
$$= \text{is } N \sum_n Cat_{n+1} (P\ N)^n$$

Hence the number of parses in the auxiliary-inverted case is the Catalan of one more than in the non-inverted cases. As predicted, EQSP found the following inverted sentences to be more ambiguous than their non-inverted counterparts (previously discussed on page 142) by one Catalan number.

[11] Earley's algorithm and most other context-free parsing algorithms actually work this way.

[12] The proof immediately follows from the z-transform of the Catalan series (Knuth 1975, p. 388): $zB(z)^2 = B(z) - 1$.

1	Was the number?
2	Was the number of products?
5	Was the number of products of products?
14	Was the number of products of products of products?
42	Was the number of products of products of products of products?

1	It was the number.
1	It was the number of products.
2	It was the number of products of products.
5	It was the number of products of products of products.
14	It was the number of products of products of products of products.

How could this result be incorporated into the table lookup pseudo-parser? Recall that the pseudo-parser implements Catalan grammars by returning an index into the Catalan table. For example, if there were i PPs, the parser would return: (CAT-TABLE i). We now extend the indexing scheme so that the parser implements a series connection of two Catalan grammars by returning one higher index than it would for a simple Catalan grammar. That is, if there were n PPs, the parser would return (CAT-TABLE (+ n 1)).

Series connections of Catalan grammars are very common in every day natural language, as illustrated by the following two sentences, which have received considerable attention in the literature because the parser cannot separate the direct object from the prepositional complement.

(27a) I saw the man on the hill with a telescope ...
(27b) Put the block in the box on the table in the kitchen ...

Both examples have a Catalan number of ambiguities because the auto-convolution of a Catalan series yields another Catalan series.[13] This result can improve parsing performance because it suggests ways to re-organize (compile) the grammar so that there will be fewer references to quantities that are not readily available. This re-organization will reap benefits that chart parsers (e.g., Earley's algorithm) do not currently achieve because the re-organization is taking advantage of a number of combinatoric regularities, especially *convolution,* that are not easily encoded into a chart. Section 9 presents an example of the re-organization.

[13] There is a difference between these two sentences because "put" subcategorizes for two objects unlike "see." Suppose we analyze "see" as lexically ambiguous between two senses, one that selects for exactly two objects like "put" and one that selects for exactly one object as in "I saw it." The first sense contributes the same number of parses as "put" and the second sense contributes an additional Catalan factor.

6.2 Chart Parsing

Perhaps it is worthwhile to reformulate chart parsing in our terms in order to show which of the above results can be captured by such an approach and which cannot. Traditionally, chart parsers maintain a chart (or matrix) M, whose entries M_{ij} contain the set of category labels that span from position i to position j in the input sentence. This is accomplished by finding a position k between i and j such that there is a phrase from i to k that can combine with another phrase from k to j. An implementation of the inner loop looks something like:

```
(28) $M_{ij} := \{ \}$
     loop for k from i to j do
          $M_{ij} := M_{ij} \cup M_{ik} * M_{kj}$
```

Essentially, then, a chart parser is maintaining the invariant

(29) $$M_{ij} = \sum_k M_{ik} \cdot M_{kj}$$

where addition and multiplication of matrix elements is related to parallel and series combination. Thus chart parsers are able to process very ambiguous sentences in polynomial time, as opposed to exponential (or Catalan) time.

However, the examples above illustrate cases where chart parsers are not as efficient as they might be. In particular, chart parsers implement convolution the "long way," by picking each possible dividing point k, and parsing from i to k and from k to j; they do not reduce the convolution of two Catalans as we did above. Similarly, chart parsers do not make use of the "every way ambiguous" generalization; given a Catalan grammar, chart parsers will eventually enumerate all possible values of i, j, and k.

7. Computing the Power Series Directly from the Grammar

Thus far, most of our derivations have been justified in terms of successive approximation. It is also possible to derive some interesting (and well-known) results directly from the grammar itself. Suppose, for the sake of discussion, that we choose to analyze adjuncts with a right branching grammar.[14] (By convention, terminal symbols appear in lower case.)

(30) $\text{ADJS} \rightarrow \text{adj ADJS} \mid \Lambda$

First we translate the grammar into an equation in the usual way. That is, ADJS is modeled as a parallel combination of two subgrammars, adj ADJS and Λ. (Λ, the empty string, is modeled as 1 because it is the

[14] A similar analysis of adjuncts is adopted in Kaplan and Bresnan 1981. This analysis can also be defended on performance grounds as an efficiency approximation. (This approximation is in the spirit of pseudo-attachment (Church 1980).)

identity element under series combination, i.e., multiplication.)

(31a) ADJS $\rightarrow$ adj ADJS | Λ

(31b) ADJS = adj $\cdot$ ADJS + 1

We can simplify (31b) so the right hand side is expressed in terminal symbols alone, with no references to non-terminals. This is very useful for processing because it is much easier for the parser to determine the presence or absence of terminals than of non-terminals. That is, it is easier for the parser to determine, for example, whether a *word* is an adj, than it is to decide whether a *substring* is an ADJS phrase. The simplification moves all references to ADJS to the left hand side, by subtracting from both sides,

(31c) ADJS – adj $\cdot$ ADJS = 1

factoring the left hand side,

(31d) (1 – adj)ADJS = 1

and dividing from both sides,

(31e) ADJS = $(1 - \text{adj})^{-1}$

By performing the long division, we observe that (31) has unit coefficients.

(31f) $$\frac{1}{1-\text{adj}} = 1 + \frac{\text{adj}}{1-\text{adj}} = 1 + \text{adj} + \frac{\text{adj}^2}{1-\text{adj}}$$
$$= 1 + \text{adj} + \text{adj}^2 + \frac{\text{adj}^3}{1-\text{adj}} = ... = \sum_n \text{adj}^n$$

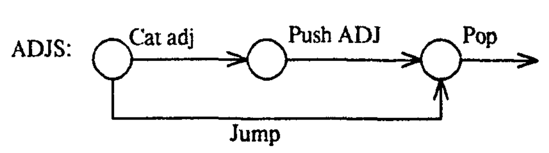
Grammars like ADJS will sometimes be referred to as a *step,* by analogy to a unit step function in electrical engineering.

8. Computing the Power Series from the ATN

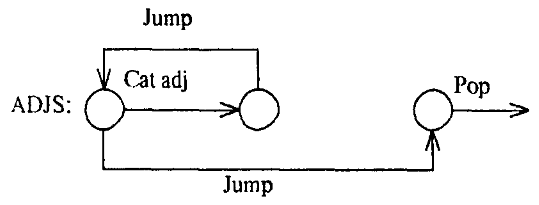
This section will re-derive the power series for the unit step grammar directly from the ATN representation by treating the networks as *flow graphs* (Oppenheim 1975). The graph transformations presented here are directly analogous to the algebraic simplifications employed in the previous section.

First we translate the grammar into an ATN in the usual way (Woods 1970).

(32) ADJS $\rightarrow$ adj ADJS | Λ

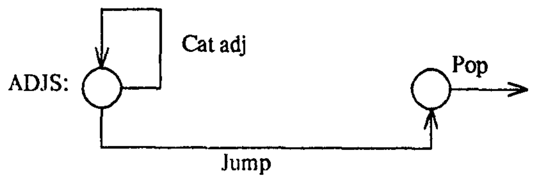
(33)


This graph can be simplified by performing a compiler optimization call *tail recursion* (Church and Kaplan 1981 and references therein). This transformation replaces the final push arc with a jump:

(34)


Tail recursion corresponds directly to the algebraic operations of moving the ADJS term to the left hand side, factoring out the ADJS, and dividing from both sides.

Then we remove the top jump arc by series reduction. This step corresponds to multiplying by 1 since a jump arc is the ATN representation for the identity element under series combination.

(35)


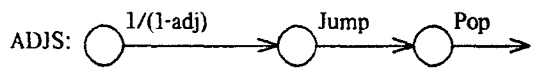
The loop can be treated as an infinite series:

(36) $1 + \text{adj} + \text{adj}^2 + \text{adj}^3 + ...$

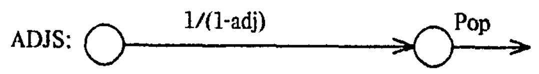
where the zero-th term corresponds to zero iterations around the loop, the first term corresponds to a single iteration, the second term to two iterations, and so on. Recall that (36) is equivalent to:

(37) $\frac{1}{1-\text{adj}}$

With this observation, it is possible to open the loop:

(38)


After one final series reduction, the ATN is equivalent to expression (31e) above.

(38g)


Intuitively, an ATN loop (or step grammar) is a division operator. We now have composition operators for parallel composition (addition), series composition (multiplication), and loops (division).

An ATN loop can be implemented in terms of the table lookup scheme discussed above. First we reformulate the loop as an infinite sum:

(39) $\frac{1}{1-adj} = \sum_i adj^i$

Then we construct a table so that the i^{th} entry in the table tells the parser how to parse i occurrences of adj.

9. An Example

Suppose for example that we were given the following grammar:

(40a) S → NP VP ADJS
(40b) S → V NP (PP) ADJS ADJS
(40c) VP → V NP (PP) ADJS
(40d) PP → P NP
(40e) NP → N | NP PP
(40f) ADJS → adj ADJS | Λ

(In this example we will assume no lexical ambiguity among N, V, P, and adj.)

By inspection, we notice that NP and PP are Catalan grammars and that ADJS is a Step grammar.

(41a) $PP = \sum_{i>0} Cat_i(P\ N)^i$

(41b) $NP = N \sum_i Cat_i(P\ N)^i$

(41c) $ADJS = \sum_i adj^i$

With these observations, the parser can process PPs, NPs, and ADJSs by counting the number of occurrences of terminal symbols and looking up those numbers in the appropriate tables. We now substitute (41a-c) into (40c).

(42) $VP = V\ NP\ (1 + PP)ADJS$
$= V \left(N \sum_i Cat_i(P\ N)^i\right)\left(\sum_i Cat_i(P\ N)^i\right)\left(\sum_i adj^i\right)$

and simplify the convolution of the two Catalan functions

(43) $VP = V \left(N \sum_i Cat_{i+1}(P\ N)^i\right)\left(\sum_i adj^i\right)$

so that the parser can also find VPs by just counting coccurrences of terminal symbols. Now we simplify (40a-b) so that S phrases can also be parsed by just counting occurrences of terminal symbols. First, translate (40a-b) into the equation:

(44) S = NP VP ADJS + V NP (1+PP) ADJS ADJS

and then expand VP using (42)

(45) S = NP (V NP (1+PP) ADJS) ADJS
+ V NP (1+PP) ADJS ADJS

and factor

(46) $S = (NP + 1)\ V\ NP\ (1+PP)\ ADJS^2$

That can be simplified considerably because

(47) $NP\ (1 + PP) = N \sum_i Cat_i(P\ N)^i \sum_i Cat_i(P\ N)^i$
$= N \sum_i Cat_{i+1}(P\ N)^i$

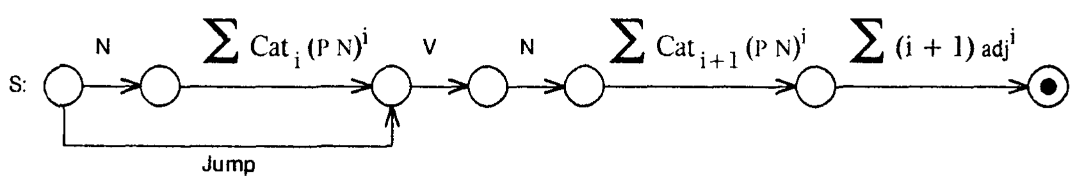
and

(48) $ADJS^2 = \sum_i adj^i \sum_i adj^i = \sum_i (i + 1)adj^i$

so that

(49) $S = \left(N \sum_i Cat_i(P\ N)^i + 1\right)$
$V\ N \sum_i Cat_{i+1}(P\ N)^i$
$\sum_i (i + 1)adj^i$

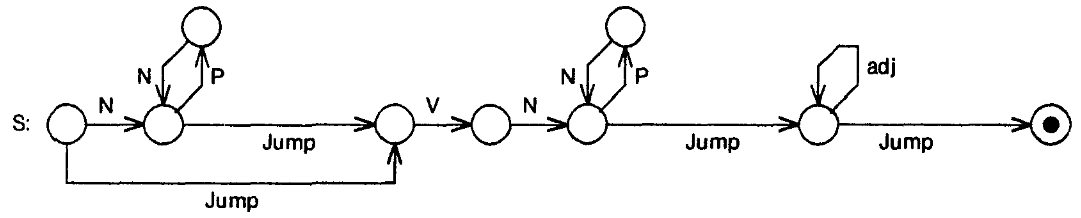
which has the following ATN realization:

(50) 

The entire example grammar has now been compiled into a form that is easier for parsing. This formula says that sentences are all of the form:

(51) $S \overset{*}{\rightarrow} (N\ (P\ N)^*)\ V\ N\ (P\ N)^*\ adj^*$

which could be recognized by the following finite state machine:

(52) 

Furthermore, the number of parse trees for a given input sentence can be found by multiplying three numbers: (a) the Catalan of the number of P N's before the verb, (b) the Catalan of one more than the number of P N's after the verb, and (c) the ramp of the number of adj's. For example, the sentence

(53) The man on the hill saw the boy with a telescope yesterday in the morning.

has $Cat_1 * Cat_2 * 3 = 6$ parses. That is, there is one way to parse "the man on the hill," two ways to parse "saw the boy with a telescope" ("telescope" is either a complement of "see" as in (54a-c) or is attached to "boy" as in (54d-f)), and three ways to parse the adjuncts (they could both attach to the S (54a,d), or they could both attach to the VP (54b,e), or they could split (54c,f)).

(54a) [The man on the hill [saw the boy with a telescope] [yesterday in the morning.]]
(54b) The man on the hill [[saw the boy with a telescope] [yesterday in the morning.]]
(54c) The man on the hill [[saw the boy with a telescope] yesterday] in the morning.
(54d) [The man on the hill saw [the boy with a telescope] [yesterday in the morning.]]
(54e) The man on the hill [saw [the boy with a telescope] [yesterday in the morning.]]
(54f) The man on the hill [saw [the boy with a telescope] yesterday] in the morning.

All and only these possibilities are permitted by the grammar.

10. Conclusion

We began our discussion with the observation that certain grammars are "every way ambiguous" and suggested that this observation could lead to improved parsing performance. Catalan grammars were then introduced to remedy the situation so that the processor can delay attachment decisions until it discovers some more useful constraints. Until such time, the processor can do little more than note that the input sentence is "every way ambiguous." We suggested that a table lookup scheme might be an effective method to implement such a processor.

We then introduced rules for combining primitive grammars, such as Catalan grammars, into composite grammars. This linear systems view "bundles up" all the parse trees into a single concise description capable of telling us everything we might want to know about the parses (including how much it might cost to ask a particular question). This abstract view of ambiguity enables us to ask questions in the most convenient order, and to delay asking until it is clear that the pay-off will exceed the cost. This abstraction was very strongly influenced by the notion of *delayed binding*.

We have presented combination rules in three different representation systems: power series, ATNs, and context-free grammars, each of which contributed its own insights. Power series are convenient for defining the algebraic operations, ATNs are most suited for discussing implementation issues, and context-free grammars enable the shortest derivations. Perhaps the following quotation best summarizes our motivation for alternating among these three representation systems:

A thing or idea seems meaningful only when we have *several* different ways to represent it – different perspectives and different associations. Then you can turn it around in your mind, so to speak; however, it seems at the moment you can see it another way; you never come to a full stop. (Minsky 1981, p. 19)

In each of these representation schemes, we have introduced five primitive grammars: Catalan, Unit Step, 1, and 0, and terminals; and four composition rules: addition, subtraction, multiplication, and division. We have seen that it is often possible to employ these analytic tools in order to re-organize (compile) the grammar into a form more suitable for processing efficiently. We have identified certain situations where the ambiguity is combinatoric, and have sketched a few modifications to the grammar that enable processing to proceed in a more efficient manner. In particular, we have observed it to be important for the grammar to avoid referencing quantities that are not easily determined, such as the dividing point between a noun phrase and a prepositional phrase as in

(55) Put the block in the box on the table in the kitchen ...

We have seen that the desired re-organization can be achieved by taking advantage of the fact that the autoconvolution of a Catalan series produces another Catalan series. This reduced processing time from $O(n^3)$ to almost linear time. Similar analyses have been discussed for a number of lexically and structurally ambiguous constructions, culminating with the example in section 9, where we transformed a grammar into a form that could be parsed by a single left-to-right pass over the terminal elements. Currently, these grammar reformulations have to be performed by hand. It ought to be possible to automate this process so that the reformulations could be performed by a grammar compiler. We leave this project open for future research.

11. Acknowledgments

We would like to thank Jon Allen, Sarah Ferguson, Lowell Hawkinson, Kris Halvorsen, Bill Long, Mitch Marcus, Rohit Parikh, and Peter Szolovits for their very useful comments on earlier drafts. We would especially like to thank Bill Martin for initiating the project.

References

Church, K. 1980 *On Memory Limitations in Natural Language Processing.* MIT/LCS/TR-245, and IULC.

Church, K. and Kaplan, R. 1981 *Removing Recursion from Natural Language Processors Based on Phrase-Structure Grammars.* Paper presented at Conference on Modeling Human Parsing Strategies, University of Texas at Austin.

Dostert, B. and Thompson, F. 1971. How Features Resolve Syntactic Ambiguity. In Minker, J. and Rosenfeld, S., eds., *Proceedings of the Symposium on Information Storage and Retrieval.*

Earley, J. 1970 An Efficient Context-Free Parsing Algorithm, *CACM* 13:2.

Harrison, M. 1978 *Introduction to Formal Language Theory.* Addison Wesley.

Kaplan, R. 1972 Augmented Transition Networks as Psychological Models of Sentence Comprehension, *Artificial Intelligence,* 3:77-100.

Kaplan, R. and Bresnan, J. 1981 Lexical-Functional Grammar: A Formal System for Grammatical Representation. In Bresnan, J., ed., *The Mental Representation of Grammatical Relations.* MIT Press.

Knuth, D. 1975 *The Art of Computer Programming. Vol. 1: Fundamental Algorithms.* Addison Wesley.

Liu, C. and Liu, J. 1975 *Linear Systems Analysis.* McGraw Hill.

Malhotra, A. 1975 *Design Criteria for a Knowledge-Based English Language System for Management: An Experimental Analysis.* MIT/LCS/TR-146.

Martin, W., Church, K., and Patil, R. 1981 *Preliminary Analysis of a Breadth-First Parsing Algorithm: Theoretical and Experimental Results.* MIT/LCS/TR-261.

Minsky, M. 1981 *Music, Mind, and Meaning.* MIT A.I. Memo No. 616.

Oppenheim, A. and Schafer, R. 1975. *Digital Signal Processing.* Prentice Hall.

Sager, N. 1973 The String Parser for Scientific Literature. In Rustin, R., ed., *Natural Language Processing.* Algorithmic Press.

Salomaa, A. 1973 *Formal Languages.* Academic Press

Woods, W. 1970 Transition Network Grammars for Natural Language Analysis, *CACM* 13:10.