Lynx: Building a Statistical Parser from a Rule-Based Parser

Peter Venable
School of Computer Science
Carnegie Mellon University
5000 Forbes Ave., Pittsburgh, PA 15213
pvenable@cs.cmu.edu

Abstract

We present a generative probabilistic grammar model based on the link grammar formalism. Describing the generative process, we show a practical way to break down the probability of a sentence and its linkage into a product of probabilities. Then we present a parser trained upon the output of an existing (imperfect) parser, and compare its results to those of its teacher. The comparison shows that our parser makes some appropriate generalizations to the grammar of its teacher.

1 Introduction

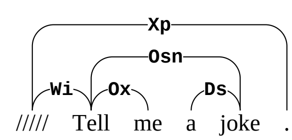
In order to parse sentences accurately, a computer model of a language's grammar must capture both lexical and structural information. Context-free grammar formalisms often handle structure succinctly but lack lexical information. Other popular models, such as trigrams, are highly lexical but are unable to capture structure. However, simple adaptations of these models, such as dependency grammars, can handle some structure by modeling connections between words rather than just consecutive word sequences. Similarly, context-free grammars have been extended with lexical statistics. Taking these adaptations a step further, we designed a formalism that combines a context-free grammar model with full lexical statistics, developed a model based on it, and built a parser using that model.

While it would be very difficult to write rules that would capture the knowledge of a statistical language model, it is relatively straightforward to convert rule-based knowledge into statistical knowledge. Given a probabilistic model capable of capturing the important features of the grammar, those rules can be converted into statistical form by parsing a set of sentences with the rule-based parser, and then using these sentences as training data for the statistical parser. No recoding of rules nor annotating of parser must be done by hand. Using this approach, we train a new statistical parser based on the existing hand-built rule-based *Link Parser*, dubbing it *Lynx* for "Link, Yet Not eXactly."

Beginning with a description of the link grammar formalism and its existing parser, we describe the generative model it inspired, and how we chose to decompose the probabilities to make them tractable. Then we present the Lynx parser with its parse algorithm and describe how we used the output of the link grammar parser to train the Lynx parser. Comparing the two parsers' results, we find that the Lynx parser makes some appropriate generalizations to the grammar of its teacher, but fails to surpass it overall, though a simple combination of the two parsers does better than either alone. Discussion is followed by comparison with related work.

2 Link and Lynx

Link grammar, designed by Daniel Sleator and David Temperley (1991) represents grammatical relationships as labeled links, called connectors, each connecting two words and thereby signifying a grammatical relationship between them. Thus it is a kind of dependency grammar. A sentence with all its connectors is called a *linkage*, and shows the grammatical structure of the entire sentence. Each word in a sentence is connected to one or more other words such that the entire linkage is a connected planar graph, as in this typical sentence:



Some connector types:

- `Xp` - words to periods at ends of sentences
- `Wi` - imperatives to the *wall* (/////)
- `Osn` - transitive verbs to singular direct objects
- `Ox` - transitive verbs to singular indirect objects
- `Ds` - determiners to singular nouns

The ordered lists of connectors attached to a word on the left and right, respectively, together constitute that word's *disjunct* in a linkage. In this example, the disjunct of the word "me" is a single connector on the left side, labeled `Ox`. The disjunct for "Tell" has three connectors: `Wi` on the left, and `Osn` and `Ox` on the right.

For most words, multiple disjuncts are possible. The collection of disjuncts allowed for each known type of word comprises the knowledge of a link grammar model. Along with a planarity constraint (links never cross), it defines which linkages are considered grammatical. No statistics are involved. The *Link Parser* uses this knowledge to parse English sentences, finding valid linkages given a sentence, and using heuristics to select the best among legal parses. Linkages are ordinarily acyclic, but there are a few situations in which a cycle is appropriate, such as when a sentence contains a relative clause.

2.1 Lynx

The Lynx model, presented here, is based on link grammar, but its knowledge consists of contextual probabilities of connectors and words rather than a fixed set of disjuncts associated with each word. Thus while linkages have the same form, the finding the best parse for a sentence requires a search driven by probabilities. The Link Parser distinguishes valid from invalid parses by determining whether a parse fits the rules encoded in its database of disjuncts for each word in the sentence, but requires extra knowledge to choose the best parse. The Lynx model assigns probabilities to the various connectors for each word (conditioned on other connectors) and is thus able to assign a probability to any parse. In the Lynx parser's output, the links form a directed acyclic graph rooted at the *left wall*, a special word before the first word of the sentence.

3 The Generative Model

Probabilistic parsing models fall into two main categories: *discriminative* models, which estimate the probability of a parse given a sentence, assigning probability mass to all linkages possible for a given sentence; and *generative* models, which assign probability mass jointly over all sentence-linkage pairs. In natural language, a generative model looks from the point of view of the speaker generating a sentence, while a discriminative (comprehension) model looks from the point of view of a listener who knows what words were said but must determine their structure. An advantage of generative models is that they can be simpler to design, train, and use.

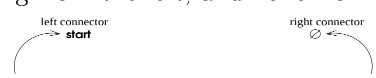
The Lynx model is a generative model, generating a sentence together with its parse. We decompose the joint probability of a sentence and linkage as the product of the probabilities of all its words and connectors. These components are generated in a well-defined order consistent with the linkage's directed graph. In general, the probability of each word or connector may be conditioned on all the words and connectors generated so far, which we call the context $\mathcal{C}$. After describing the order of generation, we describe how we condition on a limited part of the context to make the model practical.

Definitions:

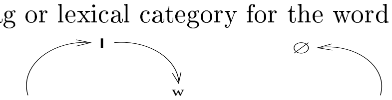
w	- a word
c	- a connector
l	- a connector coming from the left
r	- a connector coming from the right
$\emptyset$	- no connector
t_w	- the tag (category) of word w
$\mathbf{c}_w$	- the 1 or 2 connector(s) by which w was generated
$\mathbf{w}_c$	- the word(s) at the start of connector(s) $\mathbf{c}$
a_c	- the connector directly above c, from same source
$cycle$	- true only if a cycle is closed at this point
$\mathcal{C}$	- the context of any item, which is a partial parse

The simplest way to describe a generative model is to step through the process of generating a parsed sentence. At each step, the decisions are made according to a probability distribution conditioned on the parse generated so far.

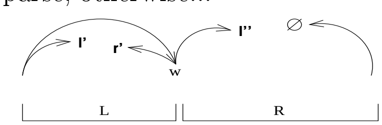
1. Start with no words, a special start-connector coming from the left, and none from the right.



2. Generate a word to be the target of a current connector. If there are connectors coming from both left and right, we first choose which one to use (or both, forming a cycle). We also choose a tag or lexical category for the word.



3. Generate new connectors coming into the subranges to the left and right of the recently-generated word (except where connectors are left over unused from a previous step). If both new connectors are *nil*, we're done with this sub-parse; otherwise...



4. Recursively generate two sub-linkages, calling steps 2-4 for the regions L and R, to the left and right, respectively, of the current word.

4 Making the Model Practical

Given an order of generation, we then need to decide what parts of the context are relevant for each word or link generated. A perfect model would capture the true dependencies among all the decisions in the entire generation process, but some independence assumptions are clearly required to make the model practical. Fortunately, intuition tells us that many such assumptions are reasonable. Following are the assumptions we have chosen, attempting to balance the increased model accuracy provided by more dependencies against the increased efficiency and generalization provided by greater independence.

$$
\begin{aligned}
P(cycle|\mathcal{C}) &\approx P\big(cycle|l,r\big) && (1)\\
P(w|\mathcal{C}) &\approx \begin{array}{l} P\big(w|t_w,\mathbf{c}_w,\mathbf{w}_{\mathbf{c}_w}\big)\times \\ P\big(t_w|\mathbf{c}_w,\mathbf{w}_{\mathbf{c}_w},cycle\big)\end{array} && (2)\\
P(c|\mathcal{C}) &\approx P\big(c|w_c,t_{w_c},\mathbf{c}_{w_c},a_c\big) && (3)
\end{aligned}
$$

Assumption (1) states that the probability of closing a cycle at any given point depends only on the two incoming connectors. Assumption (2) states that the probability of a word being generated from a specific context depends only on the tag of that word, the incoming connector(s), the words at the other end of the incoming connector(s), and whether a cycle is being closed at that word. Assumption (3) states that the probability of a connector depends only on the word from which the connector springs, the tag of that word, the connector(s) which generated that word, and the connector (if any) above the new connector.

4.1 Further Refinement

The choice of independence assumptions has a large effect on parsing speed. For example, if we were to add just one more dependency such as making the probability of a connector also depend on $w_{\mathbf{c}_{w_c}}$, the word from which its parent connector sprang, worst-case parse time would go from cubic to quadratic in sentence length.

Conversely, if we made the independence assumptions too strong, the model would lose its power to capture the grammar of the language. With a few more minor independence assumptions (see table 1), the probabilities of assumptions (1) through (3) can be broken down further to simplify the model, reducing space and time requirements and providing smoothing for sparse data. Approximation (4) states that the probability of a word can be broken down into independent components, based on its tag, incoming connectors, and other words.

Approximations (5) and (6) together account for six separate cases, each with separate probability distributions. Left and right are separate. Every connector depends on the word from which it sprouts. If a connector is not the first to sprout in that direction from its parent word, it also depends on the previous connector to do so. Otherwise, in addition to the word from which it sprouts, it depends on that word's tag and parent connector.

Approximation (7) shows three distinctions. First, there are three separate distributions, depending on whether the word in question is being generated from the left, the right, or both (which is a cycle). Second, the probability contributions from the left and right connectors are mutually independent. Third, each distribution is smoothed with a linearly interpolated back-off from to dependence on the incoming connector and its parent word to dependence only on the connector.

5 The Parse Algorithm

To parse a sentence is to find the most probable linkage given the sentence. To do so, we use a two-phase dynamic beam search.

The algorithm, in simplified form, is as follows:

```
ParseSentence( sentence )
   Prob_BestParse ← ChooseWord( range, start )
ChooseWord( range, context )
   foreach( word ∈ range )
     foreach( tag(word) )
       find P( tag, word | context )
   foreach( P(word,tag) > threshold )
     prob ← P(word) *
         ChooseConnectors( subrange_L, context_L ) *
         ChooseConnectors( subrange_R, context_R )
   return max( prob )
ChooseConnectors( range, context )
   foreach( incoming connector pair (L,R) ∈ range )
     find P(L,R | context)
   foreach( P(L,R) > threshold )
     prob ← P(L,R) * ChooseWord( range, subcontext )
   return max( prob )
```

The parse proceeds through a pair of mutually recursive functions, one to select a probable word at the end of a current connector, and one to select probable connectors to branch out into the current range. To begin, the context has a special start connector coming from the left and no connector from the right. To choose a word within a context, we look at the actual words available in the current range being parsed, finding the probability in context of that word with each reasonable tag for that word. We store these scored word-tag pairs in a heap, and then pull out the most probable few for further consideration, using a dynamic threshold. Thus this is a dynamic beam search. For each word-tag pair considered further, we recursively look at the subranges on the left and right of that word. To generate connectors within a subrange, we find the probability of each reasonable pair of incoming connectors (one from each side) given the context. Storing these on a heap, we then consider in detail the most probable among them, again using a dynamic threshold to determine at how many to recursively look for words to which to connect them. The recursion halts when both incoming connectors are *nil*, which is required if the subrange has size zero.

6 Experimental Results

6.1 Training

The Lynx grammar model is learned from a corpus of linkages. For the comparisons in this paper, we trained the Lynx model on 22,000 sentences from the Wall Street Journal, parsed by the Link

$$
\begin{aligned}
P\big(w|t_w,\mathbf{c}_w,\mathbf{w}_{\mathbf{c}_w}\big) &\approx \lambda_1 P\big(w|t_w\big)+\lambda_2 P\big(w|\mathbf{c}_w\big)+\lambda_3 P\big(w|\mathbf{w}_{\mathbf{c}_w}\big) && (4)\\
P\big(l|w_l,t_{w_l},\mathbf{c}_{w_l},a_l\big) &\approx \begin{cases} P(l|w_l,a_{w_l}) & a_{w_l}\neq\emptyset\\ P(l|w_l,t_{w_l},l_{w_l}) & a_{w_l}=\emptyset\wedge r_{w_l}=\emptyset\\ P(l|w_l,t_{w_l},r_{w_l}) & a_{w_l}=\emptyset\wedge r_{w_l}\neq\emptyset \end{cases} && (5)\\
P\big(r|w_r,t_{w_r},\mathbf{c}_{w_r},a_r\big) &\approx \begin{cases} P(r|w_r,a_{w_r}) & a_{w_r}\neq\emptyset\\ P(r|w_r,t_{w_r},r_{w_r}) & a_{w_r}=\emptyset\wedge l_{w_r}=\emptyset\\ P(r|w_r,t_{w_r},l_{w_r}) & a_{w_r}=\emptyset\wedge l_{w_r}\neq\emptyset \end{cases} && (6)\\
P\big(t_w|\mathbf{c}_w,\mathbf{w}_{\mathbf{c}_w},cycle\big) &\approx \begin{cases} \big(\lambda_1 P(t_w|l_w)+\lambda_2 P(t_w|l_w,w_{l_w})+ \\ \lambda_3 P(t_w|r_w)+\lambda_4 P(t_w|r_w,w_{r_w})\big) & cycle\\ \big(\lambda_1 P(t_w|l_w)+\lambda_2 P(t_w|l_w,w_{l_w})\big) & c_w=l_w\\ \big(\lambda_1 P(t_w|r_w)+\lambda_2 P(t_w|r_w,w_{r_w})\big) & c_w=r_w \end{cases} && (7)
\end{aligned}
$$

Table 1: probability approximations

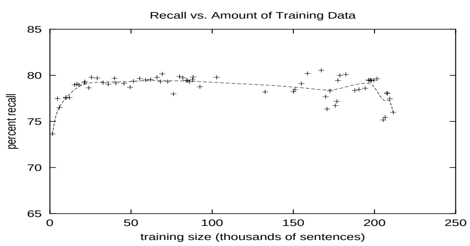


Figure 1: Lynx recall vs. training size

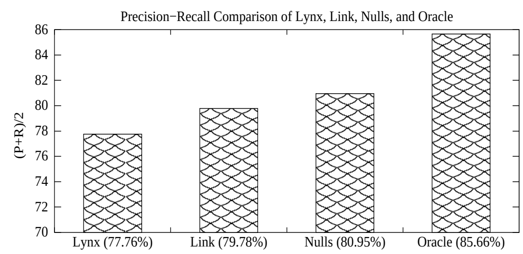


Figure 2: Precision-Recall Comparison of Lynx, Link, Nulls, and Oracle

	Link	*Lynx*	*Oracle*	*Nulls*
Link		166	166	79
Lynx	240		240	168
Oracle	0	0		0
Nulls	72	87	159	

Figure 3: The number of sentences (out of 552) for which the column method beats the row method.

Parser. That is, the sentences are first parsed by the Link Parser, the output of which, in the form of sentence linkages, is used to build the model for the Lynx parser, which simply counts occurrences of the various fragments contained in its model (described above). These counts are then used in the Lynx parser to determine the probability, according to the model, of each hypothesized linkage as search progresses. While we would expect better results from hand-built training data, automatically-generated training data costs much less to produce, and we want to demonstrate that a decent parser can be constructed even from errorful training data.

To see how much training data the Lynx model needs for best results, we built a series of models from differing amounts of training data, testing each model for constituent bracketing based on the same held-out set of 552 hand-parsed sentences from the Penn Treebank to produce the recall curve in figure 1. The corresponding precision curve has the same shape, but is lower. This experiment indicates that at least 30,000 sentences should be used, but that increasing training size beyond 100,000 would be counterproductive.

6.2 Comparison with the Link Parser

One of our goals is to see if the Lynx parser can excel beyond the capabilities of the parser from which it is trained, the Link parser. A test set of 552 sentences was parsed by each parser: the Link parser, the Lynx parser, an oracle which always selects the better of the two, and a selector called *Nulls*, which selects the Lynx parser's output only if the Link Parser's output contains *null links*. A null link is used by the Link Parser to connect adjacent words when no other connector is valid; it signals the inability of the Link Parser to fully parse the sentence. The scores are the average of precision and recall for constituent bracketing on each sentence, as compared to a hand-parsed standard provided by the Penn Treebank.

The bar chart in figure 2 shows the average score of each parser. The Lynx parser is not yet as accurate as the Link Parser from which its training data came, but it is close. While is it not too surprising that an oracle would score higher than either real parser, it is interesting that a simple selector can do

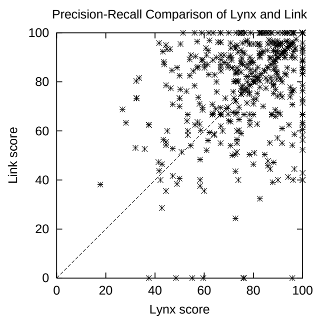


Figure 4: Lynx vs. Link, sentence-by-sentence

so, and particularly in this case because the Lynx parser was trained exclusively on the output of the Link Parser. This improvement therefore shows that the Lynx parser is taking what it learns from the Link Parser and generalizing it in a productive way. It also demonstrates a small increase in total accuracy through a simple combination of two parsers without reliance on an (unavailable) oracle.

To more precisely distinguish the contributions of each parser, the chart in figure 3 shows the number of sentences for which each method (listed in the column headings) scores better than each of the other methods (listed in the row headings). While at this point the Lynx parser is still out-scored by its predecessor, it does score better than the Link Parser on 166 out of 552 sentences, 79 of which the Link Parser knew it was unable to parse correctly.

To illustrate the wide range of scores between the two parsers, figure 4 shows for each sentence the score of each parser. Sentences for which the two parsers got the same answer are on the diagonal. The wide scatter shows that for many sentences one does well and the other poorly, while for many others they both score similarly. This low correlation shows that the Link and Lynx parsers have different areas of strength and weakness, explaining the high score of our imaginary oracle, which takes the better score for each sentence.

7 Discussion

If the Lynx parser were merely a new parser attempting to improve upon the Link Parser, these scores would be very disappointing. However, it is significant that although the Lynx parser learned "all it knows" from the Link Parser, it was able to outperform its teacher on 30% of all sentences.

In the task of modeling the grammar of a language, as in many machine-learning tasks, there's a tension between under- and over-generalizing. If a rule-based grammar model is too general, it will over-generate, coming up with ungrammatical parses, but if it's too narrow, it will fail to parse some grammatical sentences.

Consider the example sentence from section 2, "Tell me a joke." In fact, the Link Parser fails to parse this sentence because it has inadequate support for ditransitive verbs, such as *tell*, which has both a direct object "joke" and an indirect object "me." Link grammar treats "tell" as a monotransitive verb, taking only one object, either direct or indirect. Thus it fails on sentences that use "tell" ditransitively, such as "Tell me a joke," or "Tell him your story." A naive fix to this problem would be to allow "tell" to be either mono- or ditransitive, but without the right constraints this would cause over-generalization. For example, it would misinterpret phrases such as "Tell the computer scientists," incorrectly allowing "the computer" to be the indirect object and "scientists" the direct object. A correct parser must distinguish which noun phrases are appropriate direct and indirect objects, respectively, for this verb. Here "scientists" and its derived phrase "the computer scientists" would be appropriate indirect objects, but neither "the computer" nor "the computer scientists" would be an appropriate direct object of "tell."

Here is where the Lynx parser shines. Its use of probabilities with appropriate independence assumptions, rather than fixed rules, allows a much greater variety of disjuncts, while its natural lexicalization selects the appropriate links for the specific words in a sentence. It correctly parses "Tell me a joke." The Lynx model's training data (from the Link Parser) include examples of "tell" being used with only a direct object, and examples of it having only an indirect object, thus allowing it to generalize and allow both. It avoids the over-generalization pitfall described above, because the lexical statistics for the two kinds of objects are kept separate, thus allowing the direct-object examples to inform Lynx as to what kind of noun phrases make good direct objects of "tell," and likewise for indirect objects. Thus in many cases the Lynx parser can appropriately generalize the grammar it learns.

8 Related Work

Lafferty, Sleator, and Temperley introduced a generative grammar model (Grammatical Trigrams) that combined the strengths of trigrams and a link grammar (Lafferty et al., 1992). The key advantage of this combination is that the result is both highly lexical and able to capture long-range dependencies out of reach of ordinary trigrams, and advantage which is appropriated both by the Lynx model and the following related work. Both the Lynx model and the Grammatical Trigrams model are based on the same underlying link grammar, described in (Sleator and Temperley, 1991). Both adopt very similar mathematical models, recursively dividing the sentence the same way. The biggest difference between the two models is that by generating a disjunct, Grammatical Trigrams chooses all the links coming out of a word simultaneously; Lynx on the other hand generates links one at a time, conditioning them on nearby links. This stronger independence among links allows the Lynx model more flexibility in generating disjuncts; this generalizing power is one of the Lynx model's strengths.

Fong and Wu (1995) simplified the model of Lafferty, et al. by conflating all link labels and bounding the number of connectors allowed at each word. Much like the Grammatical Trigrams model, Restricted Probabilistic Link Grammar (RPLG) generates words, orientations, and disjuncts based on (w_l, w_r, l, r). RPLG treats all links the same (ignoring labels), which greatly reduces the space of possible disjuncts, thus making learning more feasible but at the same time reducing the predictive power of links within the model. Unlike Grammatical Trigrams, RPLG and Lynx condition word probabilities on link orientation.

Similarly, Eisner (1997) used a dependency grammar with unlabeled links (model C). Like the Lynx model but unlike Grammatical Trigrams and RPLG, model C generates links with their corresponding words and subtrees one at a time rather than as a monolithic disjunct for each word. Also, in model C each word is marked with a category tag, upon which other probabilities depend, thus reducing the effects of lexical sparsity. This technique is also adopted by the Lynx model, as described in section 3 above. Since model C uses unlabeled links, in contrast to the Lynx model's labeled links, it has stronger dependencies among adjacent words, while the Lynx model has strong dependencies between words and their grandparents in the tree structure.

Other lexical generative grammars includes the work of Michael Collins, whose model 1 parser (1997), like the Lynx parser, finds the most probable parse tree for a sentence by maximizing the joint probability of the sentence and its parse tree. Like the Lynx parser, Model 1 is trained on a tree bank of examples rather than using an explicit grammar, but it achieves significantly higher precision and recall, being trained on hand-built examples rather than an errorful automatically-parsed training set. Unlike link grammar, it is a probabilistic context-free grammar,distinguishing leaves, which hold the words from internal nodes.

A lexicalized Markov grammar parsing model achieving precision-recall even higher than Collins' was presented by Eugene Charniak (2000), using a probability model inspired by maximum entropy.

9 Conclusions

We presented a generative probabilistic grammar model based on the link grammar formalism. Describing the generative process, we showed one way to break down the probability of a sentence and its linkage into a product of the probabilities of words and connectors. Then we presented a parser that finds the most probable linkage for a sentence, describing how it is trained upon linkages generated by an existing imperfect parser, and compared its results to that of its teacher, showing that while the Lynx parser has not yet generally surpassed in accuracy the link grammar parser from whose output it was trained, it has done so for a significant subset of the test data, making some appropriate generalizations to the grammar of its predecessor.

Thus we showed that a probabilistic grammar model based on a formalism much more linguistically informed than the trigrams of traditional statistical language modeling can be learned from examples, resulting in a model powerful enough to capture complex grammatical structure unavailable to simplistic statistical models, and at the same time flexible enough to capture lexical subtleties. Avoiding the necessity of painstakingly defining not only the rules of a language's grammar, but also their exceptions, this method allows rapid adaptation to specialized vocabulary, idioms, dialects, and even foreign languages. One of the keys to success is a smart combination of generalization and specificity, which neither generalizes to the point of sloppiness, nor remains too narrow to handle situations not seen in training data. The Lynx parser demonstrates this combination of structure and flexibility.

10 Acknowledgements

Many thanks to John Lafferty, for providing direction for this research; thanks also to Daniel Sleator and Davy Temperley.

References

Eugene Charniak. 2000. A maximum-entropy-inspired parser. *Proceedings of NAACL-2000.*

Michael Collins. 1997. Three generative, lexicalized models for statistical parsing. *Proceedings of the 35th Annual Meeting of the ACL.*

Jason Eisner. 1997. An empirical comparison of probability models for dependency grammar. *University of Pennsylvania Technical Report.*

John Lafferty, Daniel Sleator, and Davy Temperley. 1992. Grammatical trigrams: A probabilistic model of link grammar. *AAAI Fall Symposium on Probabilistic Approaches to Natural Language.*

Eva Wai man Fong and Dekai Wu. 1995. Learning restricted probabilistic link grammars. *University of Science and Technology, Hong Kong, Technical Report*, HKUST-CS95-27.

Daniel Sleator and Davy Temperley. 1991. Parsing english with a link grammar. *Carnegie Mellon Technical Report*, CMU-CS-91-196.