Linguistic Theory in Statistical Language Learning

Christer Samuelsson
Bell Laboratories, Lucent Technologies
600 Mountain Ave, Room 2D-339,
Murray Hill, NJ 07974, USA
christer@research.bell-labs.com

Abstract

This article attempts to determine what elements of linguistic theory are used in statistical language learning, and why the extracted language models look like they do. The study indicates that some linguistic elements, such as the notion of a word, are simply too useful to be ignored. The second most important factor seems to be features inherited from the original task for which the technique was used, for example using hidden Markov models for part-of-speech tagging, rather than speech recognition. The two remaining important factors are properties of the runtime processing scheme employing the extracted language model, and the properties of the available corpus resources to which the statistical learning techniques are applied. Deliberate attempts to include linguistic theory seem to end up in a fifth place.

1 Introduction

What role does linguistics play in statistical language learning? "None at all!" might be the answer, if we ask hard-core speech-recognition professionals. But even the most nonlinguistic language model, for example a statistic word bigram model, actually relies on key concepts integral to virtually all linguistic theories. Words, for example, and the notion that sequences of words form utterances.

Statistical language learning is applied to some set of data to extract a language model of some kind. This language model can serve a purely decorative purpose, but is more often than not used to process data in some way, for example to aid speech recognition. Anyone working under the pressure of producing better results, and who employs language models to this purpose, such a researchers in the field of speech recognition, will have a high incentive of incorporating useful aspects of language into his or her language models. Now, the most useful, and thus least controversial ways of describing language will, due to their usefulness, find their way into most linguistic theories and, for the very same reason, be used in models that strive to model language successfully.

So what do the linguistic theories underlying various statistical language models look like? And why? It may be useful to distinguish between those aspects of linguistic theory that are incidentally in the language model, and those that are there intentionally. We will start our tour of statistical language learning by inspecting language models with "very little" linguistic content, and then proceed to analyse increasingly more linguistic models, until we end with models that are entirely linguistic, in the sense that they are pure grammars, associated with no statistical parameters.

2 Word N-gram Models

Let us return to the simple bigram word model, where the probability of each next word is determined from the current one. We already noted that this model relies on the notion of a word, the notion of an utterance, and the notion that an utterance is a sequence of words.

The way this model is best visualized, and as it happens, best implemented, is as a finite-state automaton (FSA), with arcs and states both labelled with words, and transition probabilities associated with each arc. For example, there will be one state labelled *The* with one arc to each other state, for example to the state *Cat*, and this arc will be labelled *cat*. The reason for labelling both arcs and states with words is that the states constitute the only memory device available to an FSA. To remember that the most recent word was "cat", all arcs labelled *cat* must fall into the same state *Cat*. The transition probability from the state *The* along the unique arc labelled *cat* to the state *Cat* will be the probability of the word "cat" following the word "the", $P(cat \mid the)$.

More generally, we enumerate the words

$\{w_1, \ldots, w_N\}$ and associate a state S_i with each word w_i. Now the automaton has the states $\{S_1, \ldots, S_N\}$ and from each state S_i there is an arc labelled w_j to state S_j with transition probability $P(w_j \mid w_i)$, the word bigram probability. To establish the probabilities of each word starting or finishing off the utterance, we introduce the special state S_0 and special word w_0 that marks the end of the utterance, and associate the arc from S_0 to S_i with the probability of w_i starting an utterance, and the arc from S_i to S_0 with the probability of an utterance ending with word w_i.

If we want to calculate the probability of a word sequence $w_{i_1} \ldots w_{i_n}$, we simply multiply the bigram probabilities:

$$\begin{aligned} P(w_{i_1} \ldots w_{i_n}) &= \\ &= P(w_{i_1} \mid w_0) \cdot P(w_{i_2} \mid w_{i_1}) \cdot \ldots \cdot P(w_0 \mid w_{i_n}) \end{aligned}$$

We now recall something from formal language theory about the equivalence between finite-state automata and regular languages. What does the equivalent regular language look like? Let's just first rename S_0 S and, by stretching it just a little, let the end-of-utterance marker w_0 be ϵ, the empty string.

$$\begin{array}{ll} S \to w_i S_i & P(w_i \mid \epsilon) \\ S_i \to w_j S_j & P(w_j \mid w_i) \\ S_i \to \epsilon & P(\epsilon \mid w_i) \end{array}$$

Does this give us any new insight? Yes, it does! Let's define a string rewrite in the usual way: $\alpha A \gamma \Rightarrow \alpha \beta \gamma$ if the rule $A \to \beta$ is in the grammar. We can then derive the string $w_{i_1} \ldots w_{i_n}$ from the top symbol S in $n{+}1$ steps:

$$S \Rightarrow w_{i_1} S_{i_1} \Rightarrow w_{i_1} w_{i_2} S_{i_2} \Rightarrow \ldots \Rightarrow w_{i_1} \ldots w_{i_n}$$

Now comes the clever bit: if we define the derivation probability as the product of the rewrite probabilities, and identify the rewrite and the rule probabilities, we realize that the string probability is simply the derivation probability. This illustrates one of the most central aspects of probabilistic parsing:

String probabilities are defined in terms of derivation probabilities.

So the simple word bigram model not only employs highly useful notions from linguistic theory, it implicitly employs the machinery of rewrite rules and derivations from formal language theory, and it also assigns string probabilities in terms of derivation probabilities, just like most probabilistic parsing schemes around. However, the heritage from finite-state automata results in simplistic models of interword dependencies.

General word N-gram models, of which word bigram models are a special case with "N" equal to two, can be accommodated in very much the same way by introducing states that remember not only the previous word, but the N-1 previous words. This generalization is purely technical and adds little or no linguistic fuel to the model from a theoretical point of view. From a practical point of view, the gain in predictive power using more conditioning in the probability distributions is very quickly overcome by the difficulty in estimating these probability distributions accurately from available training data; the perennial sparse-data problem.

So why does this model look like it does? We conjecture the following explanations: Firstly, it is directly applicable to the representation used by an acoustic speech recognizer, and this can be done efficiently as it essentially involves intersecting two finite-state automata. Secondly, the model parameters — the word bigram probabilities — can be estimated directly from electronically readable texts, and there is a lot of that available.

3 Tag N-gram Models

Let us now move on to a somewhat more linguistically sophisticated language model, the tag N-gram model. Here, the interaction between words is mediated by part-of-speech (PoS) tags, which constitute linguistically motivated labels that we assign to each word in an utterance. For example, we might look at the basic word classes adjectives, adverbs, articles, conjunctions, nouns, numbers, prepositions, pronouns and verbs, essentially introduced already by the ancient Greek Dionysius Thrax. We immediately realise that this gives us the opportunity to include a vast amount of linguistic knowledge into our model by selecting the set of PoS tags appropriately; consequently, this is a much debated and controversial issue.

Such a representation can be used for disambiguation, as in the case of the well-known, highly ambiguous example sentence "Time flies like an arrow". We can for example prescribe that "Time" is a noun, "flies" is a verb, "like" is a preposition (or adverb, according to your taste), "an" is an article, and that "arrow" is a noun. In effect, a label, i.e., a part-of-speech tag, has been assigned to each word. We realise that words may be assigned different labels in different context, or in different readings; for example, if we instead prescribe that "flies" is a noun and "like" is a verb, we get another reading of the sentence.

What does this language model look like in more detail? We can actually recast it in virtually the same terms as the word bigram model, the only difference being that we interpret each state S_i as a PoS tag (in the bigram case, and as a tag sequence in the general N-gram case):

$$\begin{array}{ll} S \to w_k S_i & P(S_i \,\&\, w_k \mid S) \\ S_i \to w_k S_j & P(S_j \,\&\, w_k \mid S_i) \\ S_i \to \epsilon & P(\epsilon \mid S_i) \end{array}$$

Note that we have now separated the words w_k from the states S_i and that thus in principle, any state can generate any word. This is actually a slightly more powerful formalism than the standard hidden-markov model (HMM) used for N-gram PoS tagging (5). We recast it as follows:

$$\begin{array}{ll} S \to T_i S_i & P(S_i \mid S) \\ S_i \to T_j S_j & P(S_j \mid S_i) \\ S_i \to \epsilon & P(\epsilon \mid S_i) \\ T_i \to w_k & P(w_k \mid T_i) \end{array}$$

Here we have the rules of the form $S_i \to T_j S_j$, with the corresponding probabilities $P(S_j \mid S_i)$, encoding the tag N-gram statistics. This is the probability that the tag T_j will follow the tag T_i, (in the bigram case, or the sequence encoded by S_i in the general N-gram case). The rules $T_i \to w_k$ with probabilities $P(w_k \mid T_i)$ are the lexical probabilities, describing the probability of tag T_i being realised as word w_k. The latter probabilities seem a bit backward, as we would rather think in terms of the converse probability $P(T_i \mid w_k)$ of a particular word w_k being assigned some PoS tag T_i, but one is easily recoverable form the other using Bayesian inversion:

$$P(w_k \mid T_i) \quad = \quad \frac{P(T_i \mid w_k) \cdot P(w_k)}{P(T_i)}$$

We now connect the second formulation with the first one by unfolding each rule $T_j \to w_k$ into each rule $S_i \to T_j S_j$. This lays bare the independence assumption

$$P(S_j \,\&\, w_k \mid S_i) = P(S_j \mid S_i) \cdot P(w_k \mid T_j)$$

As should be clear from this correspondence, the HMM-based PoS-tagging model can be formulated as a (deterministic) FSA, thus allowing very fast processing, linear in string length.

The word string $w_{k_1} \ldots w_{k_n}$ can be derived from the top symbol S in $2n+1$ steps:

$$S \Rightarrow T_{i_1} S_{i_1} \Rightarrow w_{k_1} S_{i_1} \Rightarrow w_{k_1} T_{i_2} S_{i_2} \Rightarrow w_{k_1} w_{k_2} S_{i_2} \Rightarrow \ldots \Rightarrow w_{k_1} \ldots w_{k_n}$$

The interpretation of this is that we start off in the initial state S, select a PoS tag T_{i_1} at random, according to the probability distribution in state S, then generate the word w_{k_1} at random according to the lexical distribution associated with tag T_{i_1}, then draw a next PoS tag T_{i_2} at random according to the transition probabilities associated with state S_{i_1}, hop to the corresponding state S_{i_2}, generate the word w_{k_2} at random according to the lexical distribution associated with tag T_{i_2}, etcetera.

Another general lesson can be learned from this: If we wish to calculate the probability of a word string, rather than of a word string with a particular tag associated with each word, as the model does as it stands, it would be natural to sum over the set of possible ways of assigning PoS tags to the words of the string. This means that:

The probability of a word string is the sum of its derivation probabilities.

The model parameters $P(S_j \mid S_i)$ and $P(w_k \mid T_j)$ can be estimated essentially in two different ways. The first employs manually annotated training data and the other uses unannotated data and some reestimation technique such as Baum-Welch reestimation (1). In both cases, an optimal set of parameters is sought, which will maximize the probability of the training data, supplemented with a portion of the black art of smoothing. In the former case, we are faced with two major problems: a shortage of training data, and a relatively high noise level in existing data, in terms of annotation inconsistencies. In the latter case, the problems are the instability of the resulting parameters as a function of the initial lexical bias required, and the fact that the chances of finding a global optimum using any computationally feasible technique rapidly approach zero as the size of the model (in terms of the number of tags, and N) increases. Experience as shown that, despite the noise level, annotated training data yields better models.

Let us take a step back and see what we have got: We have notion of a word, the notion of an utterance, the notion that an utterance is a sequence of words, the machinery of rewrite rules and derivations, and string probabilities are defined as the sum of the of derivation probabilities. In addition to this, we have the possibility to include a lot of linguistic knowledge into the model by selecting an appropriate set of PoS tags. We also need to somehow specify the model parameters $P(S_j \mid S_i)$ and $P(w_k \mid T_j)$. Once this is done, the model is completely determined. In particular, the only way that syntactic relations are modelled are by the probability of one PoS tag given the previous tag (or in the general N-gram case, given the previous N-1 tags). And just as in the case of word N-grams, the sparse data problem sets severe bounds on N, effectively limiting it to about three.

We conjecture that the explanation to why this model looks like it does is that it was imported wholesale from the field of speech recognition, and proved to allow fast, robust processing at accuracy level that until recently were superior to, or on par with, those of hand-crafted rule-based approaches.

4 Stochastic Grammar Models

To gain more control over the syntactic relationships between the words, we turn to stochastic context-free grammars (SCFGs), originally proposed by Booth and Thompson (4). This is the framework in which we have already discussed the N-gram models, and it has been the starting point for many excursions into probabilistic-parsing land. A stochastic context-free grammar is really just a context-free grammar where each grammar rule has been assigned a probability. If we keep the left-hand-side (LHS) symbol of the rule fix, and sum these probabilities over the different RHSs, we get one, since the probabilities are conditioned on the LHS symbol.

The probability of a particular parse tree is the probability of its derivation, which in turn is the product of the probability of each derivation step. The probability of a derivation step is the probability of rewriting a given symbol using some grammar rule, and equals the rule probability. Thus, the parse-tree probability is the product of the rule probabilities. Since the same parse tree can be derived in different ways by first rewriting some symbol and then another, or vice versa, we need to specify the order in which the nonterminal symbols of a sentential form are rewritten. We require that in each derivation step, the leftmost nonterminal is always rewritten, which yields us the leftmost derivation. This establishes a one-to-one correspondence between parse trees and derivations.

We now have plenty of opportunity to include linguistic theory into our model by the choice of syntactic categories, and by the selection of grammar rules. The probabilistic limitations of the model mirror the expressive power of context-free grammars, as the independence assumptions exactly match the compositionality assumptions. For this reason, there is an efficient algorithm for finding the most probable pares tree, or calculating the string probability under an SCFG. The algorithm is a variant of the Cocke-Kasami-Younger (CKY) algorithm (17), but can also be seen as an incarnation of a more general dynamic-programming scheme, and it is cubic in string length and grammar size. We conjecture that exactly the properties of SCFGs discussed in this paragraph explain why the model looks like it does.

We again have the choice between training the model parameters, the rule probabilities, on annotated data, or use unannotated data and some reestimation method like the inside-outside algorithm, which is the natural generalization of the Baum-Welch method of the previous section. If the chances of finding a global optimum were slim using the Baum-Welch algorithm, they're virtually zero using the inside-outside algorithm. There is also very much instability in terms of what set of rule probabilities one arrives at as a function of the initial assignment of rule probabilities in the reestimation process. The other option, training on annotated data, is also problematic, as there is precious little of it available, and what exist is quite noisy. A corpus of CFG-analysed sentences is known as a tree bank, and tree banks will be the topic of the next section.

As we have been stressing, the key idea is to assign probabilities to derivation steps. If we instead look at the rightmost derivation in reverse, as constructed by an LR parser, we can take as the derivation probability the probability of the action sequence, i.e., the product of the probabilities of each shift and reduce action in it. This isn't exactly the same think as an SCFG, since the probabilities are typically not conditioned on the LHS symbol of some grammar rule, but on the current internal state and the current lookahead symbol. As observed by Fernando Pereira (12), this gives us the possibility to throw in a few psycho-linguistic features such as right association and minimal attachment by preferring shift actions to reductions, and longer reductions to shorter ones, respectively. So if these features are present in language, they should show up in our training data, and thus in our language model. Whether these features are introduced or incidental is debatable.

We can take the idea of derivational stochastic grammars one step further and claim that a parse tree constructed by any sequence of derivation actions, regardless of what the derivation actions are, should be assigned the product of the probabilities of each derivation step, appropriately conditioned. This idea will be crucial for the various extensions to SCFGs discussed in the next section.

5 Models Using Tree Banks

As previously mentioned, a tree bank is a corpus of CFG-annotated sentences, i.e., a collection of parse trees. The mere existence of a tree bank actually inspired a statistic language model, namely the data-oriented parsing (DOP) model (3) advocated by Remko Scha and Rens Bod. This model parses not only with the entire tree bank as its grammar, but with a grammar consisting of each subtree of each tree in the tree bank. One interesting consequence of this is that there will in general be many different leftmost derivations of any given parse tree. This can most easily be seen by noting that there is one leftmost derivation for each way of cutting up a parse tree into subtrees. Therefore, the parse probability is defined as the sum of the derivation probabilities, which is the source to the NP-hardness of finding the most probable parse tree for a given input sentence under this model, as demonstrated by Khalil Sima'an (15).

There aren't really that many tree banks around, and the by far most popular one for experimenting with probabilistic parsing is the Penn Treebank (11). This leads us to the final source of influence on the linguistic theory employed in statistical language learning: the available training and testing data.

The annotators of the Penn Treebank may have overrated the minimal-attachment principle, resulting in very flat rules with a minimum of recursion, and thus in very many rules. In fact, the Wall-Street-Journal portion of it consists of about a million words analysed using literally tens of thousands of distinct grammar rules. For example, there is one rule of the form

$$NP \rightarrow Det\ Noun\ (,\ Noun\ )^n\ Conj\ Noun$$

for each value of n seen in the corpus. There is not even close to enough data to accurately estimate the probabilities of most rules seen in the training data, let alone to achieve any type of robustness for unseen rules. This inspired David Magerman and subsequently Michael Collins to instead generate the RHS dynamically during parsing.

Magerman (10) grounded this in the idea that a parse tree is constructed by a sequence of generalized derivation actions and the derivation probability is the parse probability, a framework that is sometimes referred to as history-based parsing (2), at least when decision trees are employed to determine the probability of each derivation action taken. More specifically, to allow us to assemble the RHSs as we go along, any previously constructed syntactic constituent is assigned the role of the leftmost, rightmost, middle or single daughter of some other constituent with some probability. It may or may not also be the syntactic head of the other constituent, and here we have another piece of highly useful linguistic theory incorporated into a statistical language model: the grammatical notion of a syntactic head. The idea here is to propagate up the lexical head to use (amongst other things) lexical collocation statistics on the dependency level to determine the constituent boundaries and attachment preferences.

Collins (6; 7) followed up on these ideas and added further elegance to the scheme by instead generating the head daughter first, and then the rest of the daughters as two zero-order Markov processes, one going left and one going right from it. He also managed to adapt essentially the standard SCFG parsing scheme to his model, thus allowing polynomial processing time. It is interesting to note that although the conditioning of the probabilities are top-down, parsing is performed bottom-up, just as is the case with SCFGs. This allows him to condition his probabilities on the word string dominated by the constituent, which he does in terms of a distance between the head constituent and the current one being generated. This in turn makes it possible to let phrase-boundary indicators such as punctuation marks influence the probabilities, and gives the model the chance to infer preferences for, e.g., right association.

In addition to this, Collins incorporated the notion of lexical complements and wh-movement à la Generalized Phrase-Structure Grammar (GPSG) (8) into his probabilistic language model. The former is done by knocking off complements from a hypothesised complement list as the Markov chain of the siblings of the head constituent are generated. The latter is achieved by adding hypothesised NP gaps to these lists, requiring that they be either matched against an NP on the complement list, or passed on to one of the sibling constituents or the head constituent itself, thus mimicking the behavior of the "slash feature" used in GPSG. The model learns the probabilities for these rather sophisticated derivation actions under various conditionings. Not bad for something that started out as a simple SCFG!

6 A Non-Derivational Model

The Constraint Grammar framework (9) introduced by Fred Karlsson and championed by Atro Voutilainen is a grammar formalism without derivations. It's not even constructive, but actually rather destructive. In fact, most of it is concerned with destroying hypotheses. Of course, you first have to have some hypotheses if you are going to destroy them, so there are a few components whose task it is to generate hypotheses. The first one is a lexicon, which assigns a set of possible morphological readings to each word. Then the rules of the grammar take turn discarding morphological readings based on their syntactic context. To limit the desolation brought about by the grammar, no rule is allowed to remove a word's last reading. Once the morphological slaughter is over, the (few) surviving readings are assigned possible syntactic labels, and a new pack of ravenous rules are unleashed, this time a set of syntactic disambiguation rules.

Take as an example a simple morphological disambiguation rule like

> *If the preceding word is known to be a determiner, remove any verb reading of the current word.*[1]

Such rules can easily be extracted from annotated data using statistical techniques. If for example

$$\frac{P(Det\ Verb)}{P(Det) \cdot P(Verb)} \ll 1$$

and if the involved probabilities have been estimated from sufficient amounts of data, we may safely infer the above stated rule. Unfortunately, in a rather elaborate set of experiments (14), statistically induced constraint grammars, whose performance seemingly was on par with better known statistical disambiguation methods, failed miserably when compared with a hand-crafted constraint grammar.

This begs the question "How are these constraint grammars manufactured?" It turns out that the expert linguist postulates a grammar rule, which is immediately evaluated using a large corpus of annotated data, and its impact is presented to the grammarian. If it does well, i.e., if it removes a lot of incorrect readings, but few correct ones, it is considered good and retained. A novice, such as the author of the current article, might propose a rule to the effect of discarding any *noun* reading that follows a determiner, after which the excellent grammar-development tools designed by Pasi Tapanainen (16) will reveal that the rule removed a quarter of a million correct readings, but only three incorrect ones, and the proposed rule, together with the novice, will be thrown out the window.

At a recent international conference, where an article (13) was presented demonstrating that a particular constraint grammar outperformed a particular statistical tagger on a common disambiguation task, a member of the audience posed the question "Have you considered incorporating statistics into your framework?" whereupon one of the authors replied "We *are* using statistics, in a sensible way." That is one way of viewing it. Another would be to answer the question "What role does linguistics play in statistical language learning?" with "The linguist *is* the statistical language learner!"

[1]Participles, as in "the given example" are not considered to be verb readings in this context.

References

[1] L. E. Baum. 1972. "An Inequality and Assiciated Maximization Technique in Statistical Estimation of Probabilistic Functions of Markov Processes". In *Inequalities 3*, pp. 1–8.

[2] Ezra Black, Fred Jelinek, John Lafferty, David M. Magerman, Robert Mercer and Salim Roukos. 1993. "Towards History-based Grammars: Using Richer Models for Probabilistic Parsing". In *Procs. 28th Annual Meeting of the Association for Computational Linguistics*, pp. 31–37. ACL.

[3] Rens Bod. 1995. *Enriching Linguistics with Statistics: Performance Models of Natural Language*. ILLC Dissertation Series 1995-14.

[4] T. Booth and R. Thompson. 1973. "Applying Probability Measures to Abstract Languages". In *IEEE Transactions on Computers, 22(5)*.

[5] Kenneth W. Church. 1988. "A Stochastic Parts Program and Noun Phrase Parser for Unrestricted Text". In *Procs. 2nd Conference on Applied Natural Language Processing*, pp. 136–143. ACL.

[6] Michael Collins. 1996. "A New Statistical Parser Based on Bigram Lexical Dependencies". In *Procs. 34th Annual Meeting of the Association for Computational Linguistics*, pp. 184–191. ACL.

[7] Michael Collins. 1997. "Three Generative, Lexicalized Models for Statistical Parsing". In *Procs. 35th Annual Meeting of the Association for Computational Linguistics*, pp. 16–23. ACL.

[8] Gerald Gazdar, Ewan Klein, Geoffrey Pullum and Ivan Sag. 1985. *Generalized Phrase Structure Grammar*. Harvard University Press.

[9] Fred Karlsson, Atro Voutilainen, Juha Heikkilä and Arto Anttila, eds. 1995. *Constraint Grammar. A Language-Independent System for Parsing Unrestricted Text.* Mouton de Gruyter.

[10] David M. Magerman. 1995. "Statistical Decistion-Tree Models for Parsing". In *Procs. 33rd Annual Meeting of the Association for Computational Linguistics*, pp. 276–283. ACL.

[11] Mitchell P. Marcus, Beatrice Santorini and Mary Ann Marcinkiewicz. 1993. "Building a Large Annotated Corpus of English: the Penn

Treebank". *Computational Linguistics 19(2)*, pp. 313–330. ACL.

[12] Fernando Pereira. 1985. "A New Characterization of Attachment Preferences". In *Natural Language Parsing*, pp. 307–319. Cambridge University Press.

[13] Christer Samuelsson and Atro Voutilainen. 1997. "Comparing a Linguistic and a Stochastic Tagger". In *Procs. Joint 35th Annual Meeting of the Association for Computational Linguistics and 8th Conference of the European Chapter of the Association for Computational Linguistics*, pp. 246–253. ACL.

[14] Christer Samuelsson, Pasi Tapanainen and Atro Voutilainen. 1996. "Inducing Constraint Grammars". In *Grammatical Inference: Learning Syntax from Sentences*, pp. 146–155, Springer Verlag.

[15] Khalil Sima'an. 1996. "Computational Complexity of Probabilistic Disambiguations by means of Tree-Grammars". In *Procs. 16th International Conference on Computational Linguistics*, at the very end. ICCL.

[16] Pasi Tapanainen. 1996. *The Constraint Grammar Parser CG-2*. Publ. 27, Dept. General Linguistics, University of Helsinki.

[17] David H. Younger 1967. "Recognition and Parsing of Context-Free Languages in Time n^3". In *Information and Control 10(2)*, pp. 189–208.