Parsing CFGs and PCFGs with a Chomsky-Schützenberger representation

Mans Hulden

University of Arizona
HFST Research Group, University of Helsinki
mhulden@email.arizona.edu

Abstract

We present a parsing algorithm for arbitrary context-free and probabilistic context-free grammars based on a representation of such grammars as a combination of a regular grammar and a grammar of balanced parentheses, similar to the representation used in the Chomsky-Schützenberger theorem. The basic algorithm has the same worst-case complexity as the popular CKY and Earley parsing algorithms frequently employed in natural language processing tasks. As natural languages rarely take advantage of the crucial distinguishing feature between regular and context-free languages, that of center embedding, we also investigate methods to speed up parsing at the cost of some overgeneration by forgoing the enforcement of proper nesting of constituents in the algorithm.

Keywords: context-free grammar parsing, context-free grammar approximation, finite automata

1. Introduction

A common observation about descriptions of syntactic structure of natural language is that the vast majority of sentences annotated as context-free grammars for the most part exhibits nearly right or left-linear structure. Syntactic trees representing sentences tend very rarely, and in limited fashion, to take advantage of center-embedding, the crucial distinguishing feature between regular languages and context-free languages. Regular languages can be recognized and parsed in linear time, while the time taken to parse sentences characterized by arbitrary context-free grammars grows for all practical purposes with the cube of the length of a sentence. These facts have made it an appealing enterprise to attempt to approximate context-free grammars as regular languages, or otherwise limit the flexibility of parsing algorithms, in the hope that one could improve upon the efficiency of parsing and recognition.

In this paper we will present a parsing method for probabilistic and non-probabilistic context-free grammars based on a finite-state representation derived from the Chomsky-Schützenberger (C-S) Theorem (Chomsky and Schützenberger, 1963). In the representation, a context-free grammar G is constructed as a combination of a regular grammar R and a simple context-free grammar that consists only of balanced parentheses D. The resulting parser has the same cubic asymptotic complexity and bears many similarities to the classical Cocke-Kasami-Younger (CKY) algorithm (Younger, 1967) and the Earley parsing algorithm (Earley, 1968). It is also very simple to implement, assuming one has access to algorithms that perform a basic construction of finite automata. It may also yield more efficient parsing methods for other (P)CFG-based systems.

In addition to the basic algorithm, we present another alternate approach in which we consider modifications and constraints to the regular grammar R, and bypass the enforcement of the proper nesting of constituents D, to yield a linear time parsing algorithm that approximates the original grammar, which may be useful for specific NLP tasks. This modification is a superset approximation of the original grammar G, which can be tightened quite flexibly to the desired level of approximation.

2. Preliminaries

2.1. Notation

We will employ the standard notation of context-free grammars: a context free-grammar (CFG) G is a 4-tuple $(\Sigma_{NT}, \Sigma_T, P, S)$, where Σ_{NT} is the set of non-terminal symbols, Σ_T a set of terminal symbols, used in a set of production rules P of the form $A \rightarrow \alpha$ where $A \in \Sigma_{NT}$ and α is a sequence drawn from $(\Sigma_{NT} \cup \Sigma_T)^+$. S is a symbol from Σ_{NT} designated as the start symbol. A CFG is in Chomsky Normal Form (CNF) if all productions are of the form $A \rightarrow a$ or $A \rightarrow BC$, where $a \in \Sigma_T$ and $\{B, C\} \in \Sigma_{NT}$.

Additionally, we will use short extended regular expressions to portray regular languages (and finite automata) and assume familiarity with the notational devices of L^*, $(L \cap L')$, $\neg L$, $(L \cup L')$, LL', denoting the Kleene closure of a language, intersection, complement, union and concatenation of languages, respectively.

2.2. The Chomsky-Schützenberger Theorem

The essential difference between regular grammars and context-free grammars—that of the possibility of self-embedding—is captured by the Chomsky-Schützenberger theorem, which essentially says every context-free grammar is a combination of local constraints on well-formedness (expressible as a regular grammar), and global constraints, which reduces to the idea that some elements must be properly nested.

More formally, the theorem states that for every CFL G, there exists regular language R and two homomorphisms g and h, such that

$$L(G) = h(g^{-1}(D) \cap R) \quad (1)$$

where D is a language consisting of different types of balanced parentheses (a Dyck language).

Although the original proof of the theorem did not do so, the language D can be connected directly to the context-free grammar in question and characterized in terms of parenthesis symbols that encode a derivation of strings in the context-free language. The role of R is to restrict the occurrence of these parentheses locally, while D enforces proper nesting. Under such an interpretation, h is a homomorphism that deletes the parentheses, while g is a homomorphism that deletes actual terminal symbols Σ_T, and hence g^{-1} 'inserts' terminal symbols in a language arbitrarily.

2.3. An encoding for CFGs

A way of performing a C-S encoding of a CFL—one that also yields a constructive proof of the theorem—is to declare a parenthesis language D over an alphabet $\Sigma_{()}$ that contains $2n$ symbols for every context-free rule in a grammar, where n is the number of symbols on the right hand side, and each such parenthesis represents a stage of derivation of rule. That is, for each rule of the form:

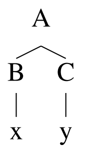
$$A \to \alpha \tag{2}$$

we include symbols $(^1_{A\to\alpha} \dots (^n_{A\to\alpha}$, and $)^1_{A\to\alpha} \dots)^n_{A\to\alpha}$ in $\Sigma_{()}$ where n is the number of elements on the right-hand side of the rule, or $|\alpha|$.

For example, if a grammar includes the rules

$$A \to BC,\ B \to x,\ C \to y \tag{3}$$

we include symbols $(^1_{A\to BC}$, $)^1_{A\to BC}$, $(^2_{A\to BC}$, and $)^2_{A\to BC}$ for the first rule (and likewise for the subsequent ones) and encode a derivation:



(4)

as the string

$$(^1_{A\to BC}(^1_{B\to x}\mathbf{x})^1_{B\to x})^1_{A\to BC}(^2_{A\to BC}(^1_{C\to y}\mathbf{y})^1_{C\to y})^2_{A\to BC}$$

That is, we represent the parse of a string as the bracketed preorder traversal of the derivation tree.

The connection to the above theorem is that if we declare a homomorphism h to be $\Sigma_{()} \to \epsilon$ and apply it to the above string, the result is naturally a string over only the terminal symbols, namely xy.

Given this, a CFG G with terminal symbols Σ_T and rules of the form $A \to \alpha$, where α is an arbitrary string of terminals and nonterminals, can be represented as $h(g^{-1}D_{()} \cap R)$, where $D_{()}$ is the language of balanced parentheses over the set of parenthesis symbols $\Sigma_{()}$, and R an intersection of five regular languages over $\Sigma = (\Sigma_T \cup \Sigma_{()})$, specified as follows:

(a) $(^1_{S\to\alpha}\Sigma^*$ where S is the initial symbol in G

(b) $(^i_{A\to B_1\dots B_n} \rightsquigarrow (^i_{B_i\to C_1\dots C_n}$ for $B_i \in \Sigma_{NT}$

(c) $)^i_{A\to B_1\dots B_n} \rightsquigarrow (^{i+1}_{A\to B_1\dots B_n}$ for $i < n$

(d) $)^n_{A\to B_1\dots B_n} \rightsquigarrow)_{ANY} \cup \#$ for $i = n$

(e) $(^i_{A\to B_1\dots B_n} \rightsquigarrow B_i\ )^i_{A\to B_1\dots B_n}$ for $B_i \in \Sigma_T$

Here, the notation $x \rightsquigarrow y$ denotes the idea that any instance of a symbol x must be immediately followed by y. Such constraints are clearly expressible as regular languages. In fact $x \rightsquigarrow y$ can be considered shorthand for the extended regular expression:

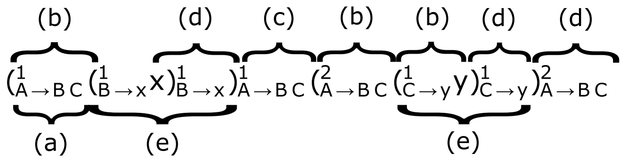
$$\neg(\Sigma^* x \neg(y\Sigma^*))$$

We also assume we can declare the constraint to hold both ways as well, i.e. $x \leftrightsquigarrow y$ would denote:

$$\neg(\Sigma^* x \neg(y\Sigma^*)) \cap \neg(\neg(\Sigma^* x)y\Sigma^*)$$

Also, the abstract symbol $\#$ denotes the end-of-string. The homomorphism g is simply $\Sigma_T \to \epsilon$ (delete terminal symbols), and h is $\Sigma_{()} \to \epsilon$ (delete parentheses). The different regular languages which are intersected encodes strictly local requirements on the ordering of the parenthesis symbols: (b) requires that an open parenthesis be immediately followed by another open parenthesis representing the first nonterminal on the right hand side; (c) enforces that a closing parenthesis representing a nonfinal constituent in a rule be followed immediately by the opening parenthesis for the same rule; (d) says that any parenthesis representing any final constituent in a rule be followed either by any other closing parenthesis or the end-of-word; (e) says that a parenthesis representing a rule constituent yielding a terminal symbol be followed by that terminal symbol and a closing parenthesis. Additionally constraint (a) enforces that any string begin with the start symbol.[1]

The constraints as given above would then apply to the example tree (4) and its string encoding as follows:



3. Parsing with C-S representations

Having in this way made the Chomsky-Schützenberger theorem more concrete by encoding the parse of a sentence as a string, we can see that this immediately yields a parsing algorithm for context-free languages.

Suppose that we have a context-free grammar G and construct from it the regular grammar R as described in steps (a)–(e) above, and wish to parse a sentence $w_1w_2\dots w_n$. Now we can easily construct a finite-state automaton R^w encoding $h^{-1}(w_1w_2\dots w_n)$, that is, an automaton that accepts only the sentence to be parsed, with arbitrary sequences of parentheses interspersed (see figure 1, step 1). This is clearly a matter of first constructing an automaton that accepts only the sentence $w_1w_2\dots w_n$, and

[1]The proof that the constraints are both necessary and sufficient for, together with D, encoding the CFG is a fairly simple induction, see e.g. Salomaa (1973) or Kozen (1997) for proofs with similar representations.

subsequently enhancing the automaton with self-loops for each symbol in $\Sigma_{()}$. Now, we can calculate the new finite automaton $R^{local} = R^w \cap R$ (step 2), that accepts all the locally correct parses of the sentence at hand. Obviously R^{local} overgenerates in the sense that it may contain invalid parses where parenthesis symbols are out of alignment, i.e. not properly nested. However, if we from the automaton representing R^{local} extract only the set of words where the parentheses are properly nested, this set equals precisely the correct parses of the sentence $w_1w_2 \ldots w_n$ in question with respect to the grammar G.

To sum up, the steps in figure 1 are:

(1) Calculate $R^w = h^{-1}(w)$

(2) Calculate $R^{local} = R \cap h^{-1}(w)$

(3) Extract from (2) the set of words where parentheses are balanced

Step (3), extracting from R^{local} only those words where parentheses are properly aligned is addressed by algorithm 1.

3.1. The algorithm

The objective of algorithm 1 is to extract all the paths that contain only balanced parentheses from the finite-state automaton produced by step (2). To this end, we maintain an agenda A which contains state pairs. Initially, the only pairs in A are those where there is a transition from one state to the other with a nonterminal symbol. From the agenda A we choose a state pair and expand it to produce a new state pair if there are transitions on the left and right with balanced parentheses. This is done in lines 14–16. We also need to merge pairs P_1 and P_2 if they represent parts of the same constituent and P_1 forms a word $(^i_\alpha \ldots)^i_\alpha$ and P_2 $(^j_\alpha \ldots)^j_\alpha$ for some rule α by checking if P_1 and P_2 share a state on the left or right. This is done in lines 8–13. Note that we do not explicitly need to check the contents of the words formed by P_1 and P_2 or that the indices i and j are such that $j = i + 1$ since this has already been taken care of by the local grammar. Hence, the finite automaton we are extracting the paths containing balanced parentheses from will never contain adjacent pairs of states in such a configuration without the indices and rule components being compatible. In other words, we need only look at the state numbers to perform the joining of constituents. Finally, whenever we encounter a state pair such that one state is the initial state and the other a final state, we have found a parse. It is assumed that in lines 17–19 we maintain backtraces of the pairs we added to the agenda A so that the string representing the parse can be reconstructed as one is found.

Algorithm 1: ExtractBalanced

```
    Input: $FSM = (V, E)$
1   begin
2       foreach $(p, q) \in V$ where there is a transition
        $p \xrightarrow{s} q$ and $s \in \Sigma_T$ do
3           add $(p, q)$ to $A$ as unmarked
4       end
5       while there is an unmarked pair in $A$ do
6           Choose a pair $P = (p, q)$ from $A$
7           Mark $P$
8           if exists a pair $Q = (p', q')$ in $A$ such that
            $q' = p$ then
9               add $(p', q)$ to $A$ as unmarked
10          end
11          if exists a pair $Q = (p', q')$ in $A$ such that
            $p' = q$ then
12              add $(p, q')$ to $A$ as unmarked
13          end
14          if exists transitions $(p' \xrightarrow{s} p)$, $(q \xrightarrow{t} q')$,
            where $s = (^i_R, t = )^j_R$ and $i = j$ then
15              add $(p', q')$ to $A$ as unmarked
16          end
17          if $p \in V_{start}$ and $q \in V_{final}$ then
18              Return(Backtrace(P))
19          end
20      end
21  end
```

4. Analysis

Before we move on to consider enhancements to the parsing algorithms, let us first briefly analyze the complexity of parsing a sentence.

First, let us represent the size of the local grammar R, which depends on G, by some constant c. It is worth noting that the constraints (a)–(e) which R is constructed from are all strictly local, maximally 3-testable languages. This leads to that the size of R grows additively with each grammar rule.[2]

The task of constructing from a sentence to be parsed the automaton R^w (step 1) takes time proportional to the length of the sentence, i.e. $|w|$. When intersecting two finite automata the result grows as the product of the two. In this case, since $|R| = c$ and R^w has $|w|$ states, we have that steps (1) and (2) are of time complexity $c|w|$, where c depends on the grammar, i.e. $O(|w|)$.

For the analysis of the algorithm for ExtractBalanced, let us restrict ourselves to cases where the grammar is given in Chomsky Normal Form. This is reflected in the automaton produced by step (2) in such a way that the maximum index i of a bracket $(^i_R$ representing a rule is 2. Interestingly, we can thus find an isomorphism between the algorithm that locates paths that contain balanced parentheses in the automaton and the CKY algorithm. That is, each pair p, q added at lines 8–13 represents a complete subtree over some span i, j for the sentence to be parsed. Obviously, the if checks at lines 8–17 take $O(1)$ time assuming some suitable indexing of the pairs stored. For any subparse of a subword of length n there are $n - 1$ ways of breaking it up into two constituents $(^1_R \ldots)^1_R(^2_R \ldots)^2_R$. This

[2]This additive growth is important since it is indeed possible to construct many different types of local grammars R that work correctly in tandem with the balanced-parenthesis-grammar D. However, not all of them exhibit subexponential complexity as the construction in this paper does.

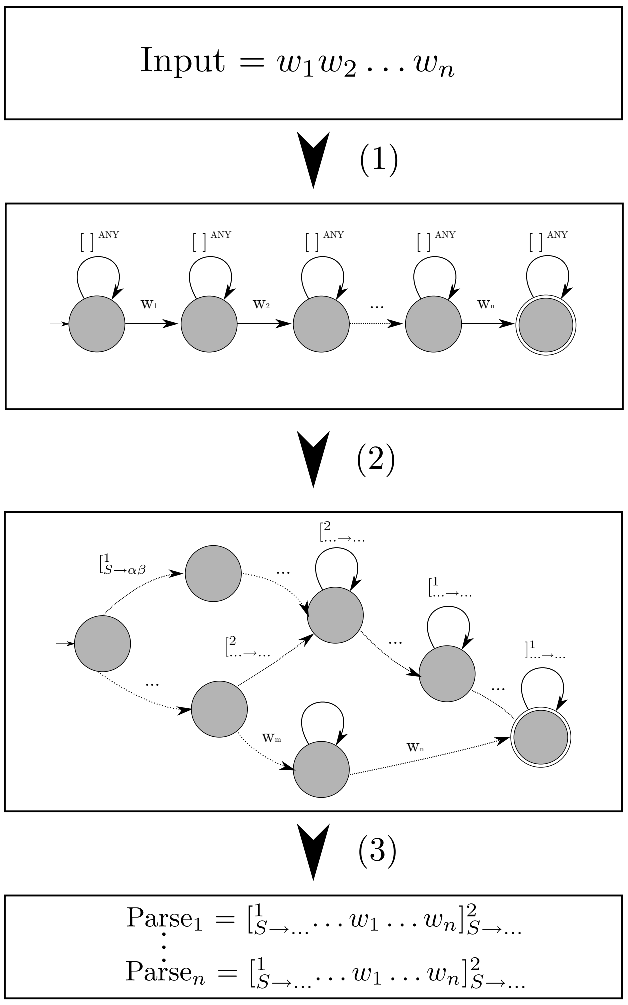

Input $= w_1 w_2 \dots w_n$
(1)
[ ] ANY
w_1
w_2
...
w_n
(2)
$[^1_{S\rightarrow\alpha\beta}$
$[^2_{\dots\rightarrow\dots}$
$[^1_{\dots\rightarrow\dots}$
$]^1_{\dots\rightarrow\dots}$
w_m
w_n
...
(3)
$\text{Parse}_1 = [^1_{S\rightarrow\dots}\dots w_1 \dots w_n]^2_{S\rightarrow\dots}$
$\text{Parse}_n = [^1_{S\rightarrow\dots}\dots w_1 \dots w_n]^2_{S\rightarrow\dots}$

Figure 1: *Basic workflow for parsing a CFG or PCFG.*

means each possible pair (of which there are n^2, n being the number of states in the automaton) added to A may be added $O(n)$ times. The total complexity is then $O(n^3)$, and given that the size of the automaton n is proportional to $|w|$, also $O(|w|^3)$.

In the above, we are not committing ourselves to any particular queuing strategy for the state pairs in A. There are many options available of how to proceed in this respect. For instance, lines 2–4 immediately add all the spans representing the terminals on the agenda after which we subsequently iterate the search until the agenda is empty. This is not strictly necessary as one can proceed left-to-right in the parse by only adding the states representing the first two terminals in the strings, and then, once A is empty, add the third, etc. This strategy would correspond somewhat to Earley's algorithm (Earley, 1968), and could avoid some extra work in that it may reduce the construction of subparses (state pairs) that are not actually parts of complete parses.

4.1. Weights and PCFGs

In many cases we would like to include treatment of probabilistic grammars in such a way that each production $A \rightarrow \alpha$ has a weight associated with it, and each parse therefore also has a total probability associated with it. This is simply a matter of storing with each A, an associated probability, or cost, and handling the maintenance of the associated subparses accordingly. There are several potential strategies regarding how this could be handled. We have chosen in our implementation to mark certain transitions in the automaton with a weight. More specifically, each transition with a symbol $(^1_R$ carries the weight/cost of rule R, other transitions carrying cost 0. Obviously, every correct parse will always include the first opening bracket symbol for each constituent, and hence it is sufficient to mark only these.

Naturally, if we are only interested in a single most likely parse, we can include a Viterbi approximation and for each A on the agenda, store only the one with lowest probability (recall from the above that each state pair represents a combination of a span and constituent).

5. Approximation strategies

So far we have only developed our approach with the intent of not approximating the grammar G at all, but representing it through a combination of regular languages and balanced parentheses. However, depending on the grammar at hand, it may be profitable to forgo step (3)—which has cubic complexity—in the parsing and instead tighten the local grammar slightly with additional constraints. In such an approach we would consider all words in the automaton from step (2) and disregard the ones with parentheses out of balance. Of course, this set of words should be finite for the approach to work.

5.1. An exponential parsing strategy

An obvious, though not efficient alternative to step (3) in the parsing scheme is to model the language $D_{()}$ as a regular language. In such a case it is advantageous if the grammar is given in CNF (Chomsky Normal Form), since we know the maximal level of nesting expected for a string of a given length, and can characterize $D^{reg}_{()}$ accordingly. Assuming we have $D^{reg}_{()}$, which depends on the length of the string w to be parsed, we can calculate:

$$g^{-1}(D^{reg}_{()}) \cap R \cap h^{-1}(w) \qquad (5)$$

yielding a regular language that accepts all and only the valid parse trees for the string w, encoded as a string. In other words, we perform steps (1) and (2), but instead of searching the automaton for strings with balanced parentheses, we intersect the result with a regular language containing only balanced parentheses (and terminal symbols) up to some required level of nesting. Unfortunately, exactly characterizing $D^{reg}_{()}$ to the required level produces an exponential growth in the state complexity of the automaton that encodes $D^{reg}_{()}$. Parsing in this manner directly is clearly infeasible, although the state complexity of the grammar R may be small (and is a constant) and the state complexity of $h^{-1}(w)$ is linear in proportion to $|w|$.

5.2. Adding constraints on productions

Constraints (a)-(e) are both necessary and sufficient for the Chomsky-Schützenberger representation. As we have included only the bare minimum of constraints for the C-S representation to work, the grammar R is very loose in the sense that the output of step (2) in original parsing strategy severely overgenerates. A reasonable question to ask, then, is whether additional constraints could be put into R, producing a tighter grammar, and where the output of step (2) would be a superset of the valid parses, yet not one that contains an infinite (or very large number) of parses. If this were feasible and step (2) would produce a reasonable number of parses, some of which would be spurious, we could in linear time produce a set of parse candidates, which could be pruned by individually checking the proper nesting of parentheses. If the ratio of spurious parses to legitimate parses in step (2) would be reasonably small (for a given grammar), it may be worthwhile to skip algorithm 1 altogether and instead just inspect the strings in R^{local}.

One additional constraint, the lack of which causes a large number of spurious parses in the original parsing strategy (before step (3)), is to add a long-distance constraint that enforces that each opening parenthesis $(^i_R$ is some time later in the word followed by the corresponding closing parenthesis, and vice versa, i.e.

(f) $(^i_R \leftrightsquigarrow \Sigma^*)^i_R$ for all rules R and components i

Such a constraint would naturally not enforce the proper ordering of any of the parentheses, i.e. not rule out sequences such as $(_1 \ldots (_2 \ldots)_1 \ldots)_2$, but would still make the grammar R overgenerate much less. However, the problem with adding such a constraint is that R grows very quickly with the number of rules if we add (f) to the set of constraints in R.

A more prudent strategy then is to modify R so that it keeps track of parenthesis order *and* balance to some *limit* of nesting: i.e.:

(f') $(^i_R \leftrightsquigarrow D^{approx})^i_R$ for all rules R and components i

where D^{approx} is the language of balanced parentheses up to some level of nesting n. This will of course also grow quickly, but as natural languages rarely exhibit center embedding up to two or three levels (Karlsson, 2007), we assume n will be kept close to this limit. This approximation limit in (f') need not be fixed for all types of sentences and can of course be extended beyond n for some subset of sentence structures or phrases which are known to occur frequently and thus need tracking of parenthesis to a deeper level. This last strategy seems to be the most profitable one to pursue and develop further in view of the very idiosyncratic nature of natural language parse trees with little center embedding.

6. Conclusion and further work

We have presented an overall strategy for parsing CFGs and PCFGs through a representation of a CFG as a combination of a regular grammar and a grammar of balanced parentheses. Using this encoding, we present a general cubic-time algorithm for parsing of arbitrary context-free grammars, which can also be used for parsing PCFGs. The resulting algorithm is simple to implement assuming one can construct finite-state automata from regular expressions.[3]

Additionally, we have considered the opportunities the encoding we have presented offers for improving parsing efficiency by regular approximation that takes into account idiosyncrasies of natural language CFG representations.

Apart from its direct usability, we also expect the basic approach to be applicable to a number of recent approaches that use CFG representations for NL tasks such as lexicalized PCFGs (Charniak, 1997) or bilexical grammar parsing tasks (Eisner, 1997), to name a few.

Having presented an algorithm for parsing CFGs and PCFGs as well as a finite-state approximation and given an analysis of the worst-case complexity is obviously not enough for practical purposes. Many of the gains that could arrive by using the approximation strategy hinge on further constraining the local grammar (which is finite-state) in such a way that it takes advantage of the specific properties of some natural language grammar at hand—perhaps one induced from a treebank or corpus. The required tightness of the local approximation and the related gains in parsing efficiency are empirical questions that need to be pursued further with actual large-scale natural language grammars.

References

Charniak, E. (1997). Statistical parsing with a context-free grammar and word statistics. In *Proceedings of the 14th National Conference on Artificial Intelligence*, pages 598–603.

Chomsky, N. and Schützenberger, M.-P. (1963). The algebraic theory of context-free languages. In Braffort, P. and Hirschberg, D., editors, *Computer Programming and Formal Systems*, pages 118–161. North Holland.

Earley, J. (1968). *An efficient context-free parsing algorithm*. PhD thesis, Carnegie-Mellon University, Pittsburgh, Pa.

Eisner, J. (1997). Bilexical grammars and a cubic-time probabilistic parser. In *Proceedings of the 1997 International Workshop on Parsing Technologies*.

Hulden, M. (2009). foma: a finite-state compiler and library. In *EACL 2009 Proceedings*, pages 29–32.

Karlsson, F. (2007). Constraints on multiple center-embedding of clauses. *Journal of Linguistics*, 43(2):365–392.

Kozen, D. C. (1997). *Automata and Computability*. Springer.

Salomaa, A. (1973). *Formal Languages*. Academic Press, New York.

Younger, D. H. (1967). Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10:189–208.

[3]We have made an implementation of the basic algorithm for CFGs and a Viterbi PCFG-parser using the finite-state toolkit *foma* (Hulden, 2009).