[redacted]

ITK RESEARCH REPORT

STITUTE FOR LANGUAGE TECHNOLOGY AND ARTIFICIAL INTELLIGENCE INSTITUUT VOOR TAAL- EN KENNISTECHNOLOGIE

ITK Research Report
August 8, 1991

Recognition for Acyclic Context-Sensitive Grammars is NP-complete

Erik Aarts

No. 29

ISSN 0924-7807

Abstract

Context-sensitive grammars in which each rule is of the form $\alpha Z\beta \rightarrow \alpha\gamma\beta$ are acyclic if the associated context-free grammar with the rules $Z \rightarrow \gamma$ is acyclic. The problem whether an input string is in the language generated by an acyclic context-sensitive grammar is NP-complete.

Introduction

One of the most well-known classifications of rewrite grammars is the Chomsky hierarchy. Grammars and languages are of type 0 (unrestricted), type 1 (context-sensitive), type 2 (context-free) or of type 3 (regular). Much research has been done involving regular and context-free grammars. Context-free languages can be recognized in a time that is polynomial in the length of the input and the length of the grammar [Earley, 1970]. Recognition of type 0 languages is undecidable. We see two majors tracks for the research on grammars which lie between these two grammar classes.

First, people have tried to put restrictions on context-sensitive grammars in order to generate context-free languages. Among them are Book [1972], Hibbard [1974] and Ginsburg and Greibach [1966]. Baker [1974] has shown that these attacks come down to the same more or less. They all block the use of context to pass information through the string. Book [1973] gives an overview of attempts to generate context-free languages with non-context-free grammars. How to restrict permutative grammars in order to generate context-free languages is described in Mäkkinen [1985].

The other track is the track of complexity of recognition. One of the best introductions to complexity theory is Garey and Johnson [1979]. They state that recognition for context-sensitive grammars is PSPACE-complete (referring to [Kuroda, 1964] and [Karp, 1972]). Some people have tried to put restrictions on CSG's so that recognition lies somewhere between PSPACE and $\mathcal{P}$. Book [1978] has shown that for *linear time* CSG's recognition is NP-complete even for (some) fixed grammars. Furthermore there is a result that recognition for *growing* CSG's is polynomial for fixed grammars [Dahlhaus and Warmuth, 1986]. This is the line I am following.

In this article I will consider one type of restricted context-sensitive grammars, the *acyclic* context sensitive grammars. The complexity of recognition is lower than in the unrestricted case because we restrict the *amount* of information that can be sent (and we do not block it by barriers!). In the unrestricted case we can send messages that *leave no trace*. After a message that changes 0's into 1's e.g. we can send a message that does the reverse. In sending a message from one position in the sentence to another, the intermediate symbols are not changed. In fact they are changed twice: back and forth. With acyclic csg's, this is not possible and the amount of information that can be sent is restricted by the grammar.

Definitions

A *grammar* is a 4-tuple, $G = (V, \Sigma, R, S)$, where
V is a set of symbols, $\Sigma \subset V$ is the set of terminal symbols.
$R \subset V^+ \times V^*$ is a relation defined on strings. Elements of R are called rules.
$S \in V \setminus \Sigma$ is the startsymbol.

A grammar is *context-sensitive* if each rule is of the form
$\alpha Z \beta \rightarrow \alpha \gamma \beta$ where $Z \in V \setminus \Sigma$; $\alpha, \beta, \gamma \in V^*$; $\gamma \neq e$.
A grammar is *context-free* if each rule is of the form
$Z \rightarrow \gamma$ where $Z \in V \setminus \Sigma$; $\gamma \in V^*$; $\gamma \neq e$.

Derivability ($\Rightarrow$) between strings is defined as follows:
$u\alpha v \Rightarrow u\beta v$ $(u, v, \alpha, \beta \in V^*)$ iff $(\alpha, \beta) \in R$.
The transitive closure of $\Rightarrow$ is denoted by $\overset{+}{\Rightarrow}$. The transitive reflexive closure of $\Rightarrow$ is denoted by $\overset{*}{\Rightarrow}$. The *language* generated by G is defined as
$L(G) = \{w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w\}$.

A *derivation* of a string δ is a sequence of strings $x_1, x_2, \ldots, x_n$ with
$S = x_1$, for all i $(1 \leq i < n)$ $x_i \Rightarrow x_{i+1}$ and $x_n = \delta$.

A context-free grammar is *acyclic* if there is no $Z \in V \setminus \Sigma$ such that
$Z \overset{+}{\Rightarrow} Z$. This implies that there is no string $\alpha \in V^*$ such that $\alpha \overset{+}{\Rightarrow} \alpha$.

We can map a context-sensitive grammar G onto its *associated* context-free grammar G' as follows: If G is (V, Σ, R, S) then G' is (V, Σ, R', S) where for every rule $\alpha Z \beta \rightarrow \alpha \gamma \beta \in R$ there is a rule $Z \rightarrow \gamma \in R'$. There are no other rules in R'.

We call G *acyclic* iff the associated context-free grammar G' is acyclic.

The notation we use for context-sensitive rules is as follows: the rule

$\alpha Z\beta \rightarrow \alpha\gamma\beta$ is written as $Z \rightarrow [\alpha_1][\alpha_2]\ldots[\alpha_i]\ \gamma\ [\beta_1][\beta_2]\ldots[\beta_j]$ with $\alpha = [\alpha_1][\alpha_2]\ldots[\alpha_i]$ and $\beta = [\beta_1][\beta_2]\ldots[\beta_j]$, with $\alpha_k, \beta_l \in V (1 \leq k \leq i, 1 \leq l \leq j)$.

Recognition is NP-complete

In this section we prove that the recognition problem for acyclic context-sensitive grammars is NP-complete. Acyclic CSG will be abbreviated as ACSG.

RECOGNITION FOR ACYCLIC CSG
INSTANCE: An acyclic context-sensitive grammar $G = (V, \Sigma, R, S)$ and a string $w \in \Sigma^*$.
QUESTION: Is w in the language generated by G ?

Before we prove that RECOGNITION FOR ACYCLIC CSG is NP-complete, we first prove some theorems and lemmas.

The function $ld(G'', n)$ is the length of the longest derivation from any input word with length n using grammar G''. Suppose $G' = (V', \Sigma', R', S')$ is an acyclic *cfg*.

Lemma 1.1[1]: $ld(G', n) \leq \frac{1}{2}|R'|n(n+1) + 1$

Proof: With induction to n.

Basic step: $n = 1$. In the worst case we can apply all rules once. The length of this derivation is $|R'| + 1$. So $ld(G', 1) = |R'| + 1$.

Induction step. We have an input word with length $n + 1$. We will try to derive the startsymbol by bottom-up application of rules on it.
There must be a branching rule. In the worst case we can apply all (maximal $|R'| - 1$) non-branching rules once to all symbols of an input with length $n+1$. This means that we have $((|R'|-1)(n+1))$ applications of rules. When we apply a branching rule we get a word with length n (or smaller). The

[1]With some more effort we can prove the linear bound $ld(G', n) \leq (2n - 1)|R'| + n$. We are only interested in a polynomial bound, however.

length of any derivation of this word is maximal $ld(G',n)$. For $ld(G',n+1)$ we have:

$$\begin{aligned}
ld(G',n+1) &\le ld(G',n) + ((|R'|-1)(n+1)+1)\\
&= \tfrac{1}{2}|R'|n(n+1)+1+((|R'|-1)(n+1)+1)\\
&< \tfrac{1}{2}|R'|n(n+1)+1+|R'|(n+1)\\
&= \tfrac{1}{2}|R'|n(n+1)+1+\tfrac{1}{2}2|R'|(n+1)\\
&= \tfrac{1}{2}|R'|(n+2)(n+1)+1\\
&= \tfrac{1}{2}|R'|(n+1)(n+2)+1 \;. \;\square
\end{aligned}$$

Lemma 1.2: $ld(G,n) \le \frac{1}{2}|R|n(n+1)+1$. (G is the acyclic *csg* earlier mentioned).

Proof: Every derivation in an acyclic csg is a derivation in the associated cfg. The number of rules in the associated cfg equals the number of rules in the acyclic csg[2]. □

Theorem 1: RECOGNITION FOR ACYCLIC CSG is in NP

Proof: A nondeterministic algorithm can guess every (bottom-up) replacement of some substring until the startsymbol has been found. This process will not take more steps than the length of the longest derivation. The longest derivation in an acyclic csg has polynomial length. Therefore, this nondeterministic algorithm runs in polynomial time and it recognizes exactly $L(G)$. □

Theorem 2: There is a transformation f of 3SAT to RECOGNITION FOR ACYCLIC CSG.

Proof: First we transform the instances of 3SAT to those of RECOGNITION FOR ACYCLIC CSG. An example of this transformation is:

$(\neg\ u_3 \vee u_2 \vee \neg\ u_1) \wedge (u_3 \vee \neg\ u_2 \vee u_1)$, a 3-SAT instance, is transformed into "v_1 v_2 v_3 not u_3 u_2 not u_1 u_3 not u_2 u_1".

[2]This is not quite true. Two context-sensitive rules can be mapped on the same context-free rule. The associated cfg can have less rules than the acyclic csg. In this case, lemma 1.2 is still true, of course.

$v_1 \ldots v_m$ and $u_1 \ldots u_m$ are boolean variables. For all i $(1 \leq i \leq m)$ the value of v_i must be equal to the value of u_i. We "extract" the variables from the formula.

"$\vee$", "$\wedge$" and brackets "(" and ")" are left out of the new formula in order to keep the grammar smaller. "$\neg$" is replaced by "not". When n is the length of the original formula the length of the new input is smaller than $2n$. This length differs only linearly in the length n of the original input.

In Appendix A the grammars for all different m can be found. The terminal symbols are: $\Sigma = \{v_i, u_i, \text{not}\}$ $(1 \leq i \leq m)$. The startsymbol S is "s". It can best be seen how these grammars recognize the satisfiable formulas of 3-SAT by applying the grammar rules bottom-up.

The values of all v_i are initialised and sent through the formula from left to right. The corresponding u_i get the same value as v_i when the information about the instantiation of the value of v_i arrives.

Most of the nonterminal symbols have two subparts: the original terminal symbol and the value that is passed. The symbol "u_3u_2t" means: I was originally u_3 and I am passing the information that v_2 has been made true.

When the value of v_i crosses u_i, u_i is turned into true or false (t or f). When u_3 "hears" from its left neighbour that v_3 has been initialized as false, "u_3u_2t" will be replaced by "fu_3f"[3].

We end up with a sequence of initialised v's followed by a sequence of t's and f's. These sequences together form an "s" in case there are no clusters of three f's. The values of the v_i can only be sent in a fixed order: first v_1, then v_2 etc. When not all values are sent, the u's are not made t or f. For every variable we can send only one value. Hence only satisfiable formula's can form an "s". The grammars recognize exactly all satisfiable formulas.□
Appendix B contains an example of a derivation for $m = 3$ of the formula "v1 v2 v3 u2 not u3 u1".

Theorem 3: f is polynomially computable.

Proof: The transformation of instances is polynomial. The number of grammar rules is cubic in m, the number of variables. □

Theorem 4: RECOGNITION FOR ACYCLIC CSG is NP-complete.

Proof: Follows from Theorems 1, 2 and 3. □

[3] "$\text{not}u_3f\ u_3u_2t$" will be replaced by "tu_3f"

Recognizing Power

ACSG's recognize all context-free languages. Any context-free grammar can be transformed into an acyclic context-free grammar without loss of recognizing power. Any acyclic context-free grammar is an acyclic context-sensitive grammar.

Furthermore, ACSG's recognize languages that are not context-free. One example is the language

$$\{a^n b^{2^n} c^n \mid n \geq 1\}$$

This language is recognized by the grammar ("x" is a nonterminal):

x → [a] a b b [b]	b → [a] x [x]	s → a b b c
x → [x] b b [b]	b → [b] x [x]	
x → [x] b b c [c]	b → [b] x [c]	

A derivation of " a a b b b b c c ":

s ⇒ a b b c ⇒ a b x c ⇒ a x x c ⇒ a x b b c c ⇒ a a b b b b c c.

With the pumping lemma one can prove that the language is not context-free.

Conclusions

We have proved that recognition for ACSG is NP-complete. It turns out to be very important for complexity of recognition with csg's whether sending information leaves a trace.

Restricting the amount of information that can be sent seems an approach that comes closer to models of human language than blocking the sending of information by barriers. In natural languages one finds ***unbounded dependencies*** which are dependencies over an unbounded distance. The number of unbounded dependencies in natural language are (almost) always restricted. The polynomial bound would be an explanation of the fact that humans can process language efficiently. Humans have a fixed grammar in mind which does not change. So the complexity of recognition with a fixed grammar should be compared with the speed of human language processing.

We have encoded 3-SAT in various acyclic context-sensitive grammars now. I think it is not possible to write an acyclic context-sensitive grammar that recognizes all 3-SAT formulas. We cannot encode 3-SAT in the input sentence (when the csg is acyclic). Therefore I think that the recognition problem for any fixed grammar is polynomial. The proof of this has not been found yet (nor a proof of the counterpart). It is the subject of ongoing research.

References

Baker, B. S., Non-context-Free Grammars Generating Context-Free Languages, *Inform. and Control*, *24*, 231–246, 1974.

Barton Jr., G. E., R. C. Berwick and E. S. Ristad, *Computational complexity and natural language*, MIT Press, Cambridge, MA, 1987.

Book, R. V., Terminal context in context-sensitive grammars, *SIAM J. Comput.*, *1*, 20–30, 1972.

Book, R. V., On the Structure of Context-Sensitive Grammars, *Internat. J. Comput. Inform. Sci.*, *2*, 129–139, 1973.

Book, R. V., On the Complexity of Formal Grammars, *Acta Inform.*, *9*, 171–181, 1978.

Dahlhaus, E. and M. K. Warmuth, Membership for Growing Context-Sensitive Grammars Is Polynomial, *Internat. J. Comput. Inform. Sci.*, *33*, 456–472, 1986.

Earley, J., An Efficient Context-Free Parsing Algorithm, *Comm. ACM*, *13*(2), 94–102, Feb. 1970.

Garey, M. R. and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, San Francisco, CA, 1979.

Ginsburg, S. and S. A. Greibach, Mappings which Preserve Context Sensitive Languages, *Inform. and Control*, *9*, 563–582, 1966.

Hibbard, T. N., Context-Limited Grammars, *J. Assoc. Comput. Mach.*, *21*(3), 446–453, July 1974.

Karp, R. M., Reducibility among combinatorial problems, in *Complexity of Computer Computations*, edited by R. E. Miller and J. W. Thatcher, pp. 85–103, Plenum Press, New York, 1972.

Kuroda, S. -Y., Classes of Languages and Linear-Bounded Automata, *Inform. and Control*, *7*, 207–223, 1964.

Mäkkinen, E., On Permutative Grammars Generating Context-Free Languages, *BIT*, *25*, 604–610, 1985.

Appendix A

The grammar contains variables which range over (m is the number of variables in the formula):

$i, j \in \{1, \ldots, m-1\}$
$k, l \in \{1, \ldots, m\}$
$tv, tv', tv'', tv''' \in \{t, f\}$
$\tilde{tv}$ is the negated value of tv and is $\in \{t, f\}$

Initialise u_1:

$v_1 u_1 tv \rightarrow v_1$

Pass the value of u_1 through the whole string:

$v_{i+1} u_1 tv \rightarrow [v_i u_1 tv]\ v_{i+1}$

$\text{not}u_1 tv \rightarrow [v_m u_1 tv]\ \text{not}$
$\text{not}u_1 tv \rightarrow [u_{i+1} u_1 tv]\ \text{not}$
$\text{not}u_1 tv \rightarrow [tv' u_1 tv]\ \text{not}$

$u_{i+1} u_1 tv \rightarrow [v_m u_1 tv]\ u_{i+1}$
$u_{i+1} u_1 tv \rightarrow [u_{j+1} u_1 tv]\ u_{i+1}$
$u_{i+1} u_1 tv \rightarrow [tv' u_1 tv]\ u_{i+1}$
$u_{i+1} u_1 tv \rightarrow [\text{not}u_1 tv]\ u_{i+1}$

u_1 is turned into true or false while passing its value:

$tv u_1 tv \rightarrow [v_m u_1 tv]\ u_1$
$tv u_1 tv \rightarrow [u_{j+1} u_1 tv]\ u_1$
$tv u_1 tv \rightarrow [tv' u_1 tv]\ u_1$

not disappears when the related variable is made true or false:

$\tilde{tv} u_1 tv \rightarrow \text{not}u_1 tv\ u_1$

Initialise u_{i+1}:

$v_{i+1} u_{i+1} tv \rightarrow v_{i+1} u_i tv'$
Pass its value through the sequence of v's:

$v_{i+1} u_{j+1} tv \rightarrow [v_i u_{j+1} tv]\ v_{i+1} u_j tv'$
$i > j$

Pass the value through the formula across not's:

$\text{not}u_{j+1} tv \rightarrow [v_m u_{j+1} tv]\ \text{not}u_j tv'$
$\text{not}u_{j+1} tv \rightarrow [u_k u_{j+1} tv]\ \text{not}u_j tv'$
$j < k - 1$
$\text{not}u_{j+1} tv \rightarrow [tv'' u_{j+1} tv]\ \text{not}u_j tv'$

Pass the value through the formula across t's and f's:

$tv'' u_{j+1} tv \rightarrow [v_m u_{j+1} tv]\ tv'' u_j tv'$
$tv'' u_{j+1} tv \rightarrow [u_k u_{j+1} tv]\ tv'' u_j tv'$
$j < k - 1$
$tv'' u_{j+1} tv \rightarrow [tv''' u_{j+1} tv]\ tv'' u_j tv'$

Across u's which should not be made true or false:

$u_l u_{j+1} tv \rightarrow [v_m u_{j+1} tv]\ u_l u_j tv'$
$j < l - 1$
$u_l u_{j+1} tv \rightarrow [u_k u_{j+1} tv]\ u_l u_j tv'$
$j < l - 1,\ j < k - 1$
$u_l u_{j+1} tv \rightarrow [tv'' u_{j+1} tv]\ u_l u_j tv'$
$j < l - 1$
$u_l u_{j+1} tv \rightarrow [\text{not} u_{j+1} tv]\ u_l u_j tv'$
$j < l - 1$

These u's must be made true or false:

$tv u_{i+1} tv \rightarrow [v_m u_{i+1} tv]\ u_{i+1} u_i tv'$
$tv u_{i+1} tv \rightarrow [u_k u_{i+1} tv]\ u_{i+1} u_i tv'$
$i < k - 1$
$tv u_{i+1} tv \rightarrow [tv'' u_{i+1} tv]\ u_{i+1} u_i tv'$

not's disappear again:

$\tilde{tv} u_{i+1} tv \rightarrow \text{not} u_{i+1} tv\ u_{i+1} u_i tv'$

All values have been passed now, start building an S:

$tv \rightarrow tv u_m tv'$

$q_m \rightarrow v_m u_m tv$

$q_m \rightarrow q_m$ t t t
$q_m \rightarrow q_m$ t t f
$q_m \rightarrow q_m$ t f t
$q_m \rightarrow q_m$ f t t
$q_m \rightarrow q_m$ f f t
$q_m \rightarrow q_m$ f t f
$q_m \rightarrow q_m$ t f f

$q_i \rightarrow v_i u_i tv\ q_{i+1}$

$s \rightarrow q_1$

Appendix B

A possible derivation:

```
v1       v2       v3       u2       not       u3       u1
v1u1t    v2       v3       u2       not       u3       u1
v1u1t    v2u1t    v3       u2       not       u3       u1

v1u1t    v2u1t    v3u1t    u2       not       u3       u1
v1u1t    v2u2t    v3u1t    u2       not       u3       u1
v1u1t    v2u2t    v3u1t    u2u1t    not       u3       u1

v1u1t    v2u2t    v3u2t    u2u1t    not       u3       u1
v1u1t    v2u2t    v3u2t    u2u1t    notu1t    u3       u1
v1u1t    v2u2t    v3u2t    tu2t     notu1t    u3       u1

v1u1t    v2u2t    v3u3f    tu2t     notu1t    u3       u1
v1u1t    v2u2t    v3u3f    tu2t     notu1t    u3u1t    u1
v1u1t    v2u2t    v3u3f    tu2t     notu2t    u3u1t    u1

v1u1t    v2u2t    v3u3f    tu3f     notu2t    u3u1t    u1

v1u1t    v2u2t    v3u3f    tu3f     notu2t    u3u1t    tu1t
v1u1t    v2u2t    v3u3f    tu3f     notu2t    u3u2t    tu1t
v1u1t    v2u2t    v3u3f    tu3f     notu3f    u3u2t    tu1t
v1u1t    v2u2t    v3u3f    tu3f     notu3f    u3u2t    tu2t

v1u1t    v2u2t    v3u3f    tu3f           tu3f         tu2t
v1u1t    v2u2t    v3u3f    tu3f           tu3f         tu3f

v1u1t    v2u2t    v3u3f    t              tu3f         tu3f
v1u1t    v2u2t    v3u3f    t              t            tu3f
v1u1t    v2u2t    v3u3f    t              t            t
v1u1t    v2u2t    q3       t              t            t
v1u1t    v2u2t                     q3
v1u1t                              q2
                                   q1
                                   s
```

Datum	nr.	auteur	titel
17-01-1989	1	H.C. Bunt	On-line Interpretation in Speech Understanding and Dialogue Systems
17-01-1989	2	P.A. Flach	Concept Learning from Examples Theoretical Foundations
07-03-1989	3	O. De Troyer	RIDL*: A Tool for the Computer-Assisted Engineering of Large Databases in the Presence of Integrity Constraints
07-03-1989	4	E.T. Thijsse	Something you might want to know about "wanting to know"
28-04-1989	5	H.C. Bunt	A Model-theoretic Approach to Multi-Database Knowledge Representation
16-06-1989	6	E.J. vd. Linden	Lambek theorem proving and feature unification
27-06-1989	7	H.C. Bunt	DPSG and its use in sentence generation form meaning representations
17-11-1989	8	R. Berndsen, H. Daniëls	Qualitative Economics in Prolog
25-01-1990	9	P. Flach	A simple concept learner and its implementation
25-01-1990	10	P. Flach	Second-order inductive learning
25-01-1990	11	E. Thijsse	Partical logic and modal logic: a systematic survey
07-02-1990	12	F. Dols	The Representation of Definite Descriptions
08-03-1990	13	R.J. Beun	The Recognition of Declarative Questions in Information Dialogues
13-03-1990	14	H.C. Bunt	Language Understanding by Computer: Developments on the Theoretical Side
19-03-1990	15	H.C. Bunt	DIT Dynamic Interpretation in Text and dialogue

Datum	**nr.**	**auteur**	**titel**
04-04-1990	16	R. Ahn, H. Kolb	Discourse Representation meets Constructive Mathematics
17-04-1990	17	G. Minnen, E.J. vd. Linden	Algorithmen for generation in lambek theorem proving
29-06-1990	18	H.C. Bunt	DPSG and its use in parsing
17-07-1990	19	H. Kolb	Levels and Empty? Categories in a Principles and Parameters Approach to Parsing
27-07-1990	20	H.C. Bunt	Modular Incremental Modelling Belief and Intention
23-08-1990	21	F. Dols	Nog niet verschenen
23-08-1990	22	F. Dols	Nog niet verschenen
14-11-1990	23	P. Flach	Inductive characterisation of database relations
06-12-1990	24	E. Thijsse	Definability in partial logic: the propositional part
21-05-1990	25	H. Weigand	Modelling Documents
21-05-1991	26	O. Troyer	Object Oriented methods in data engineering
28-05-1991	27	O. Troyer	The O-O Binary Relationship Model
03-07-1991	28	E. Thijsse	On total awareness logics
08-08-1991	29	E. Aarts	Recognition for Acyclic Context Sensitive Grammars is NP-complete

ITK: P.O. BOX 90153 5000 LE TILBURG THE NETHERLANDS