Uniform Recognition for Acyclic Context-Sensitive Grammars is NP-complete

Erik Aarts*
Research Institute for Language & Speech
Trans 10
3512 JK Utrecht
The Netherlands

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 3 (regular), type 2 (context-free), type 1 (context-sensitive) or of type 0 (unrestricted). It is easy to decide whether a string is in the language generated by a regular or context-free grammar. For context-free grammars input strings can be recognized in a time that is polynomial in the length of the input string as well as in the length of the grammar. Earley [1970] has shown a bound of $\mathcal{O}(|G|^2n^3)$ where G is the size of the grammar and n the length of the input string. Recognition for context-sensitive grammars is harder: it is PSPACE-complete [Garey and Johnson, 1979], referring to [Kuroda, 1964] and [Karp, 1972]. Recognition of type 0 languages is undecidable (see e.g. Lewis and Papadimitriou [1981]).

The area between context-free grammars and context-sensitive grammars is interesting for two reasons. First, people have tried to describe natural languages with rewrite grammars. Context-free grammars do not seem powerfull enough to describe natural languages. Context-free grammars generate context-free languages. Natural languages are probably not context-free. The counterexamples of sentences that can not be described with a context-free grammar are always a bit artificial. Very big subparts of natural languages are context-free. A grammar for natural languages has to be only a bit stronger than context-free. That's why we are interested in grammars that are between context-free and context-sensitive.

The second perspective is the one of efficient processability. In a context-free model, sentences can be processed efficiently. In a context-sensitive one, they can not. It is very interesting to know where the border lies: in which models sentences can be processed efficiently and in which ones they can not?

In the 60's and 70's, attempts have been made to put restrictions on context-sensitive grammars in order to generate context-free languages. Examples are Book [1972], Hibbard [1974] and Ginsburg and Greibach [1966]. Baker [1974] has shown that these methods come down to the same more or less. They all block the use of context to pass information through the

*The author was sponsored by project NF 102/62-356 ('Structural and Semantic Parallels in Natural Languages and Programming Languages'), funded by the Netherlands Organization for the Advancement of Research (NWO).

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].

Peters Jr. and Ritchie [1973] proposed a linguistically motivated change in the definition of the notion grammar. Subsequent replacements in a string are replaced by node admissibility constraints in the parse trees of sentences in a context-free grammar. However, this formalism leads to generation of context-free languages too. The approach of restricting grammars such that they generate context-free languages does not seem interesting from the natural language perspective nor from the efficiency perspective. The only advantages of this kind of restrictions lie in the possibilty to describe a context-free language in a different way, which may be easier for some purpose.

Another argument against blocking information [Baker, 1974] is the problem of *unbounded dependencies*. Unbounded dependencies are dependencies over an unbounded distance. Wh-movement is an example of it. The number of unbounded dependencies in natural language is (almost) always restricted. Models that restrict the *amount* of information that can be sent seem to come closer to models of human language than models restrict the *distance* over which information can be sent.

In the 70's and 80's attention has shifted to the perspective of efficient processing. Context-sensitive grammars have been restricted so that complexity of recognition lies somewhere between PSPACE and $\mathcal{P}$. Book [1978] has shown that for *linear time* context-sensitive grammars recognition is NP-complete even for (some) fixed grammars. Furthermore there is a result that recognition for *growing* context-sensitive grammars is polynomial for fixed grammars [Dahlhaus and Warmuth, 1986]. This article also tries to define a border between nearly-efficient and just-efficient models.

We can define the notions uniform (or universal) recognition and recognition for a fixed grammar as follows.

UNIFORM RECOGNITION
INSTANCE: A grammar G and a string w.
QUESTION: Is w in the language generated by G ?

The grammar, as well as the input string are inputs for the problem (these two types of input are easily confused!). The uniform recognition problem is one problem.

There are infinitely many other problems:

Suppose we have a grammar G.

RECOGNITION FOR FIXED GRAMMAR G
INSTANCE: A string w.
QUESTION: Is w in the language generated by G ?

Things are getting even more difficult when we say things like: "For every grammar G RECOGNITION FOR FIXED GRAMMAR G ...". The difference between uniform recognition and recognition for *all* fixed G can be illustrated with an example from Barton Jr., Berwick and Ristad [1987]. They show that uniform recognition for *unordered* context-free grammar (UCFG) can be done in time $\mathcal{O}(2^{|G|}n^3)$. It has not been shown that the uniform recognition problem is in $\mathcal{P}$. For every G, however, the fixed recognition problem can be solved in time $\mathcal{O}(n^3)$ and all these problems are in $\mathcal{P}$. Barton Jr., Berwick and Ristad [1987] show the problem to be polynomial for any fixed grammar by a compilation step. The UCFG is compiled into a big context-free grammar. They use this grammar and the Earley algorithm in order to prove a polynomial bound. Just forgetting about the grammar size (replacing $|G|$ by a constant) gives a polynomial bound too. It is not clear why Barton Jr., Berwick and Ristad [1987] always associate the fixed grammar problem with compilation (cf. their pp. 27-30, 64-79 and 202-206).

This article is about uniform recognition for one type of restricted context-sensitive grammars, the *acyclic* context sensitive grammars (ACSG's). We prove it to be NP-complete. This means they are as complex as the Agreement Grammars and the Unordered CFG's of Barton Jr., Berwick and Ristad [1987]. ACSG's are the pure rewrite grammars in this group. They fit in the Chomsky hierarchy.

The Uniform Recognition Problem

CSG	PSPACE-complete
ACSG	NP-complete
AG	
UCFG	
CFG	P

One might ask when we can *use* acyclic context-sensitive grammars. One can use them everywhere where one wants to use context-sensitive grammars. But one has to be careful: cycles are not allowed. This property of acyclicity can be checked easily[1]. For most purposes one does not need cycles at all. One field where context-sensitive grammars can be used is e.g. morphology. Characters in a word are often changed when some suffix is added. These changes in a word are context-sensitive and can be described by a context-sensitive grammar. Once a character is changed, we normally do not want to change it back, the grammar we use is an acyclic one.

The complexity of recognition for ACSG is lower than in the unrestricted case (CSG, with complexity PSPACE) because we restrict the *amount* of information that can be passed through the sentence. The number of messages that can be sent is limited (and we do not block the messages by barriers as in Baker [1974] !). In the unrestricted case we can send messages that *leave no trace.* E.g. after a message that changes 0's into 1's 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 context-sensitive grammars, this is not possible. Every messages leaves a trace 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

[1]It is much easier than checking whether a CSG is a *linear time* CSG as defined by Book [1978]. One has to reason about length of possible derivations. In ACSG, derivations are short as a result of their acyclicity.

$\alpha Z\beta \to \alpha\gamma\beta$ 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
$Z \to \gamma$ where $Z \in V \setminus \Sigma$; $\gamma \in V^*$.

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
$x_1 = S$, 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 $\alpha Z\beta \to \alpha\gamma\beta \in R$ there is a rule $Z \to \gamma \in R'$. There are no other rules in R'. Note that the associated grammar does not contain empty productions.

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

An example of a context-sensitive grammar with the corresponding context-free rules is:

context-sensitive rules	context-free part
$1 \to [0]\ 2$	$1 \to 2$
$0 \to 1\ [2]$	$0 \to 1$
$2 \to [1]\ 0$	$2 \to 0$

This context-sensitive grammar is cyclic. It is able to permute 0's and 1's.

Recognition is NP-complete

In this section we prove that the recognition problem for acyclic context-sensitive grammars is NP-complete.

UNIFORM RECOGNITION FOR ACYCLIC CONTEXT-SENSITIVE GRAMMAR
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 UNIFORM RECOGNITION FOR ACYCLIC CONTEXT-SENSITIVE GRAMMAR is NP-complete, we first prove some theorems and lemmas.

Suppose $G' = (V', \Sigma', R', S')$ is an acyclic *context-free* grammar. The function $ld(G'', n)$ is the length of the longest derivation from any input word with length n $(n \geq 1)$ using grammar

G''.

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

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. The grammar must contain a branching rule. A branching rule is a rule whose righthand-side contains more than one element. Grammars without branching rules only generate words with length 1. In the worst case we can apply all (maximal $|R'|-1$) non-branching rules once to all symbols of the input with length $n+1$. This means that we have $((|R'|-1)(n+1))$ applications of rules. Then we apply a branching rule and get a word with length n (or smaller). The length of any derivation of this word is maximally $ld(G',n)$. For $ld(G',n+1)$ we have:

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

Lemma 1.2: $ld(G,n) \leq \frac{1}{2}|R|n(n+1)+1$. ($G$ is the acyclic *context-sensitive* grammar earlier mentioned).

Proof: Every derivation in an acyclic context-sensitive grammar is a derivation in the associated context-free grammar. The number of rules in the associated context-free grammar equals the number of rules in the acyclic context-sensitive grammar[3]. $\square$

Theorem 1: UNIFORM RECOGNITION FOR ACYCLIC CONTEXT-SENSITIVE GRAMMAR 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 context-sensitive grammar has polynomial length. Therefore, this nondeterministic algorithm runs in polynomial time and it recognizes exactly $L(G)$. $\square$

Theorem 2: There is a transformation f of 3SAT to UNIFORM RECOGNITION FOR ACYCLIC CONTEXT-SENSITIVE GRAMMAR.

[2]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.

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

Proof: First we transform the instances of 3SAT to those of UNIFORM RECOGNITION FOR ACYCLIC CONTEXT-SENSITIVE GRAMMAR. 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 "ini $\neg$ u_3 u_2 $\neg$ u_1 u_3 $\neg$ u_2 u_1".

The transformation should be done as follows.

The symbols "$\vee$", "$\wedge$" and the brackets "(" and ")" are left out of the new formula in order to keep the grammar smaller. An extra symbol is added in front of the formula. This symbol has to initialize all variables. We use the symbol "ini" for it and we call it the ini-symbol.

In Appendix A the grammars for all different m (the number of variables in the formula) can be found. The terminal symbols are: $\Sigma = \{\text{ini}, \neg, u_i\}$ $(1 \leq i \leq m)$. The startsymbol S is "s". The number of rules of the grammar is cubic in m. We can show how this grammar recognizes a satisfiable formula of 3-SAT by applying the grammar rules bottom-up.

All u_i are initialized as true or false and their values are sent through the formula from left to right.

Most nonterminal symbols have three subparts: the original terminal symbol, some variable and the value of that variable (true or false). E.g. the symbol "u_3u_2t" has been derived (bottom-up) from the terminal symbol u_3 and passes the information that u_2 has been made true.

When the value of u_i crosses u_i, u_i is turned into true or false (t or f). When e.g. u_3 "hears" from its left neighbour that u_3 has been initialized as false, "u_3u_2t" will be replaced by "fu_3f"[4].

We end up with the ini-symbol followed by a sequence of t's and f's. These sequences together form an "s" when none of the clusters of truthvalues contains three f's. The values of the u_i can only be sent in a fixed order: first u_1, then u_2 etc. When not all values are sent, the u's are not changed into 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 of the formula "ini u2 $\neg$ u3 u1" (where $m = 3$).

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: UNIFORM RECOGNITION FOR ACYCLIC CONTEXT-SENSITIVE GRAMMAR is NP-complete.

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

Recognizing Power

Any context-free grammar can be transformed into an acyclic context-free grammar without loss of recognizing power. A cycle can be removed by introduction of a new symbol. This symbol rewrites to any member of the cycle. Any context-free grammar with empty productions can be changed into a context-free grammar without empty productions that recognizes the same language. There's one exception here: languages containing the empty string can not be generated. Any acyclic context-free grammar without empty productions is an acyclic context-sensitive grammar. Therefore, ACSG's recognize all context-free languages that do not contain the empty word.

[4] "$\neg u_3$f u_3u_2t" will be replaced by "tu_3f": $\neg$ u_3 must get the value *true* when u_3 is initialized as *false*.

Furthermore, acyclic context-sensitive grammars 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]	A → a	S → A B B C
X → [X] B B [B]	B → [B] X [X]	B → b	
X → [X] B B C [C]	B → [B] X [C]	C → c	

A derivation of " A A B B B B C C ":

S $\Rightarrow$ A B B C $\Rightarrow$ A B X C $\Rightarrow$ A X X C $\Rightarrow$ A X B B C C $\Rightarrow$ A A B B B B C C
$\overset{*}{\Rightarrow}$ a a b b b b c c.

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

Discussion

We have proved that UNIFORM RECOGNITION FOR ACYCLIC CONTEXT-SENSITIVE GRAMMAR is NP-complete. It turns out to be important for complexity of recognition with context-sensitive grammars whether sending information leaves a trace.

We have reduced 3-SAT to the uniform recognition problem for acyclic context-sensitive grammars. Every 3-SAT formula results in a different grammar. Probably it is not possible to construct an acyclic context-sensitive grammar that recognizes *all* 3-SAT formulas. My conjecture is that ACSG-recognition is not NP-hard for any fixed grammar. If this is not true, there would exist a grammar that recognizes all 3-SAT formulas. For this grammar the recognition problem would be NP-hard. In such a grammar, not every 3-SAT variable is encoded in a different symbol in the grammar. The variables are numbered and their numbers are encoded in sequences of 0's and 1's e.g. . A grammar that recognizes all 3-SAT formula's must be able to compare such sequences. It must e.g. be able to recognize the language $\{ww \mid w \in V^*\}$. If w is a number, two numbers are compared. Context-sensitive grammars can recognize ww. Some can even recognize all 3-SAT formula's.

ACSG's are not that strong. They can not even recognize ww. Any ACSG can compare only a fixed number of characters (only fixed amounts of information can be sent). Therefore my conjecture is that the recognition problem for any *fixed* grammar is not so hard: it's polynomial. Chart parsers for ACSG have been designed and implemented [Aarts, 1991]. They recognize inputs for many hard grammars in polynomial time. It is hard to prove, however, that they run in polynomial time for every grammar. If it could be proved, complexity of ACSG-recognition is similar to complexity of UCFG-recognition: NP-complete for the uniform case and a known algorithm that runs in time something like $\mathcal{O}(2^{|G|}n^3)$ (polynomial in n but not in G).

The polynomial bound (which has not been proved yet) would be an explanation of the fact that humans can process language efficiently. Humans have a fixed grammar in mind which does not change. The complexity of recognition with a fixed grammar should be compared with the speed of human language processing. The arguments of Barton Jr., Berwick and Ristad [1987] against this are based on two kinds of arguments. The first has to do with compilation or preprocessing. We have polynomial bounds without compilation or preprocessing (just fix $|G|$). These arguments do not seem to hold. The other ones have to do with language acquisition. When a child is learning a language, the grammar she uses is changing. At every sentence utterance or understanding the grammar seems to be fixed. The difference between uniform recognition and recognition for any fixed grammar is that small that we can not draw conclusions about what kind of processing children perform when learning a language.

Acknowledgements

I want to thank Peter van Emde Boas, Reinhard Muskens, Mart Trautwein and Theo Jansen for their comments on earlier versions of this paper. Peter van Emde Boas showed me a simplification in the construction of the grammar.

References

Aarts, E., Recognition for Acyclic Context-Sensitive Grammars is probably Polynomial for Fixed Grammars, Tilburg University, ITK Research Memo no. 8, 1991.

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.

Lewis, H. R. and C. H. Papadimitriou, *Elements of the theory of computation*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

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

Peters Jr., P. S. and R. W. Ritchie, Context-Sensitive Immediate Constituent Analysis: Context-Free Languages Revisited, *Math. Systems Theory*, *6*(4), 324–333, 1973.

Appendix A

m is the number of variables in the 3-SAT formula. The variables i, j, k, l are necessary to define the grammar. u_i, u_j, u_k, u_l range over the variables in the 3-SAT formula.
$i, j \in \{1, \ldots, m-1\}$
$k, l \in \{1, \ldots, m\}$

In order to save space boolean variables are introduced. They can reduce 16 rules to one rule:
$tv, tv', tv'', tv''' \in \{t, f\}$
$\tilde{tv}$ is the negated value of tv
and is $\in \{t, f\}$

A. First initialize u_1:

$\text{ini–}\mathrm{u}_1 tv \rightarrow \text{ini}$

B. Pass the value of u_1 through the whole string:

$\neg \mathrm{u}_1 tv \rightarrow [\text{ini–}\mathrm{u}_1 tv]\ \neg$
$\neg \mathrm{u}_1 tv \rightarrow [\mathrm{u}_{i+1}\mathrm{u}_1 tv]\ \neg$
$\neg \mathrm{u}_1 tv \rightarrow [tv'\mathrm{u}_1 tv]\ \neg$

$\mathrm{u}_{i+1}\mathrm{u}_1 tv \rightarrow [\text{ini–}\mathrm{u}_1 tv]\ \mathrm{u}_{i+1}$
$\mathrm{u}_{i+1}\mathrm{u}_1 tv \rightarrow [\mathrm{u}_{j+1}\mathrm{u}_1 tv]\ \mathrm{u}_{i+1}$
$\mathrm{u}_{i+1}\mathrm{u}_1 tv \rightarrow [tv'\mathrm{u}_1 tv]\ \mathrm{u}_{i+1}$
$\mathrm{u}_{i+1}\mathrm{u}_1 tv \rightarrow [\neg \mathrm{u}_1 tv]\ \mathrm{u}_{i+1}$

C. u_1's are turned into true or false when its value is passed:

$tv\mathrm{u}_1 tv \rightarrow [\text{ini–}\mathrm{u}_1 tv]\ \mathrm{u}_1$
$tv\mathrm{u}_1 tv \rightarrow [\mathrm{u}_{j+1}\mathrm{u}_1 tv]\ \mathrm{u}_1$
$tv\mathrm{u}_1 tv \rightarrow [tv'\mathrm{u}_1 tv]\ \mathrm{u}_1$

D. $\neg$'s disappear when the variables behind them are made true or false:

$\tilde{tv}\mathrm{u}_1 tv \rightarrow \neg \mathrm{u}_1 tv\ \ \mathrm{u}_1$

E. Initialise the next variable u_{i+1}:

$\text{ini–}\mathrm{u}_{i+1} tv \rightarrow \text{ini–}\mathrm{u}_i tv'$

F. Pass the value through the formula across $\neg$'s:

$\neg \mathrm{u}_{j+1} tv \rightarrow [\text{ini–}\mathrm{u}_{j+1} tv]\ \neg \mathrm{u}_j tv'$
$\neg \mathrm{u}_{j+1} tv \rightarrow [\mathrm{u}_k \mathrm{u}_{j+1} tv]\ \neg \mathrm{u}_j tv'$
$j < k-1$
$\neg \mathrm{u}_{j+1} tv \rightarrow [tv''\mathrm{u}_{j+1} tv]\ \neg \mathrm{u}_j tv'$

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

$tv''\mathrm{u}_{j+1} tv \rightarrow [\text{ini–}\mathrm{u}_{j+1} tv]\ tv''\mathrm{u}_j tv'$
$tv''\mathrm{u}_{j+1} tv \rightarrow [\mathrm{u}_k \mathrm{u}_{j+1} tv]\ tv''\mathrm{u}_j tv'$
$j < k-1$
$tv''\mathrm{u}_{j+1} tv \rightarrow [tv'''\mathrm{u}_{j+1} tv]\ tv''\mathrm{u}_j tv'$

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

$\mathrm{u}_l \mathrm{u}_{j+1} tv \rightarrow [\text{ini–}\mathrm{u}_{j+1} tv]\ \mathrm{u}_l \mathrm{u}_j tv'$
$j < l-1$
$\mathrm{u}_l \mathrm{u}_{j+1} tv \rightarrow [\mathrm{u}_k \mathrm{u}_{j+1} tv]\ \mathrm{u}_l \mathrm{u}_j tv'$
$j < l-1,\ j < k-1$
$\mathrm{u}_l \mathrm{u}_{j+1} tv \rightarrow [tv''\mathrm{u}_{j+1} tv]\ \mathrm{u}_l \mathrm{u}_j tv'$
$j < l-1$
$\mathrm{u}_l \mathrm{u}_{j+1} tv \rightarrow [\neg \mathrm{u}_{j+1} tv]\ \mathrm{u}_l \mathrm{u}_j tv'$
$j < l-1$

I. These u's must be made true or false because the information about their initialization has arrived:

$tv\mathrm{u}_{i+1} tv \rightarrow [\text{ini–}\mathrm{u}_{i+1} tv]\ \mathrm{u}_{i+1}\mathrm{u}_i tv'$
$tv\mathrm{u}_{i+1} tv \rightarrow [\mathrm{u}_k \mathrm{u}_{i+1} tv]\ \mathrm{u}_{i+1}\mathrm{u}_i tv'$
$i < k-1$
$tv\mathrm{u}_{i+1} tv \rightarrow [tv''\mathrm{u}_{i+1} tv]\ \mathrm{u}_{i+1}\mathrm{u}_i tv'$

J. $\neg$'s disappear again:

$\tilde{tv}\mathrm{u}_{i+1} tv \rightarrow \neg \mathrm{u}_{i+1} tv\ \ \mathrm{u}_{i+1}\mathrm{u}_i tv'$

K. All values of u's have been passed now, start building an S:

$tv \rightarrow tv\mathrm{u}_m tv'$

$\mathrm{s} \rightarrow \text{ini–}\mathrm{u}_m tv$

s → s t t t
s → s t t f
s → s t f t
s → s f t t
s → s f f t
s → s f t f
s → s t f f

Appendix B

A possible derivation for the 3-SAT formula ($u_2 \vee \neg\, u_3 \vee u_1$). u_1 and u_2 are initialized as true. u_3 is initialized as false. The formula is changed into a cluster of three t's.
In front we see the string that is changing. Behind is indicated which rule is applied. The characters A, B, . . . show from which groups of rules the rule is taken.

ini	u_2	$\neg$	u_3	u_1	A	ini–u_1t $\rightarrow$ ini
ini–u_1t	u_2	$\neg$	u_3	u_1	B	u_2u_1t $\rightarrow$ [ini–u_1t] u_2
ini–u_1t	u_2u_1t	$\neg$	u_3	u_1	E	ini–u_2t $\rightarrow$ ini–u_1t
ini–u_2t	u_2u_1t	$\neg$	u_3	u_1	B	$\neg u_1$t $\rightarrow$ [u_2u_1t] $\neg$
ini–u_2t	u_2u_1t	$\neg u_1$t	u_3	u_1	I	tu_2t $\rightarrow$ [ini–u_2t] u_2u_1t
ini–u_2t	tu_2t	$\neg u_1$t	u_3	u_1	E	ini–u_3f $\rightarrow$ ini–u_2t
ini–u_3f	tu_2t	$\neg u_1$t	u_3	u_1	B	u_3u_1t $\rightarrow$ [$\neg u_1$t] u_3
ini–u_3f	tu_2t	$\neg u_1$t	u_3u_1t	u_1	F	$\neg u_2$t $\rightarrow$ [tu_2t] $\neg u_1$t
ini–u_3f	tu_2t	$\neg u_2$t	u_3u_1t	u_1	G	tu_3f $\rightarrow$ [ini–u_3f] tu_2t
ini–u_3f	tu_3f	$\neg u_2$t	u_3u_1t	u_1	C	tu_1t $\rightarrow$ [u_3u_1t] u_1
ini–u_3f	tu_3f	$\neg u_2$t	u_3u_1t	tu_1t	H	u_3u_2t $\rightarrow$ [$\neg u_2$t] u_3u_1t
ini–u_3f	tu_3f	$\neg u_2$t	u_3u_2t	tu_1t	F	$\neg u_3$f $\rightarrow$ [tu_3f] $\neg u_2$t
ini–u_3f	tu_3f	$\neg u_3$f	u_3u_2t	tu_1t	G	tu_2t $\rightarrow$ [u_3u_2t] tu_1t
ini–u_3f	tu_3f	$\neg u_3$f	u_3u_2t	tu_2t	J	tu_3f $\rightarrow$ $\neg u_3$f u_3u_2t
ini–u_3f	tu_3f	tu_3f		tu_2t	G	tu_3f $\rightarrow$ [tu_3f] tu_2t
ini–u_3f	tu_3f	tu_3f		tu_3f	K	t $\rightarrow$ tu_3f
ini–u_3f	t	tu_3f		tu_3f	K	t $\rightarrow$ tu_3f
ini–u_3f	t	t		tu_3f	K	t $\rightarrow$ tu_3f
ini–u_3f	t	t		t	K	s $\rightarrow$ ini–u_3f
s	t	t		t	K	s $\rightarrow$ s t t t
		s				