LFG Generation from Acyclic F-Structures is NP-Hard

Jürgen Wedekind
University of Copenhagen
Department of Nordic Studies
and Linguistics
jwedekind@hum.ku.dk

Ronald M. Kaplan
Stanford University
Linguistics Department
rmkaplan@stanford.edu

The universal generation problem for LFG grammars is the problem of determining whether a given grammar derives any terminal string with a given f-structure. It is known that this problem is decidable for acyclic f-structures. In this brief note, we show that for those f-structures the problem is nonetheless intractable. This holds even for grammars that are off-line parsable.

The universal generation problem for LFG grammars (Kaplan and Bresnan 1982) is the problem of determining for an arbitrary grammar G and an arbitrary f-structure F whether G derives any terminal string with F. This has been shown to be undecidable even for grammars that are off-line parsable (Wedekind 2014). If F is acyclic, however, Wedekind and Kaplan (2012) have shown that the problem is decidable. They prove that the set of strings that an LFG grammar relates to an acyclic f-structure can be described by a context-free grammar. Decidability of the problem then follows because the emptiness problem is decidable for context-free languages. To date, however, the complexity status of this problem has been unknown.

In this brief note, we show the intractability of LFG's generation problem from acyclic f-structures by polynomial-time reduction from the 3-SAT problem, a problem that is known to be NP-complete. The 3-SAT problem is the problem of determining the satisfiability of a Boolean formula in conjunctive normal form where each of the conjoined clauses is a disjunction of three literals. That is, each formula is a conjunction of the form $C_1 \wedge .. \wedge C_m$, each clause C_j is a disjunction of the form $l_{j_1} \vee l_{j_2} \vee l_{j_3}$, and each literal l_{j_k}, $k = 1, .., 3$, is a propositional variable p_i or a negated variable $\neg p_i$. Without loss of generality, we assume in the following that every literal occurs only once in a clause and a clause does not contain both, a variable and its complement.

Submission received: 13 March 2021; revised version received: 10 June 2021; accepted for publication: 16 July 2021.

https://doi.org/10.1162/COLI_a_00419

To state the generation problem more formally, recall that an LFG grammar G defines a binary derivation relation Δ_G between terminal strings and f-structures, as given in (1).

(1) $\Delta_G(s, F)$ if and only if G derives terminal string s with f-structure F

The generation problem from acyclic f-structures is then the problem of determining for an arbitrary LFG G and an arbitrary acyclic f-structure F whether $\{s \mid \Delta_G(s, F)\}$ is empty or not.

Reductions to Problem-specific Grammars

For any instance $\psi = C_1 \wedge .. \wedge C_m$ of the 3-SAT problem over variables $p_1, .., p_n$, we construct an LFG grammar G_ψ and an acyclic f-structure F_ψ such that there is a string s and $(s, F_\psi) \in \Delta_{G_\psi}$ if and only if ψ is satisfiable.

The grammar G_ψ includes the start rule

(2) $\mathrm{S} \rightarrow \underset{\uparrow=\downarrow}{\mathrm{P}_1} \; .. \; \underset{\uparrow=\downarrow}{\mathrm{P}_n}$

and for each propositional variable p_i two terminal rules of the form (3a,b).

(3) a. $\mathrm{P}_i \rightarrow \$ \bigwedge_{\substack{p_i \text{ occurs in } C_j \\ j=1,..,m}} (\uparrow \mathrm{C}_j) = \mathrm{TRUE}$

b. $\mathrm{P}_i \rightarrow \$ \bigwedge_{\substack{\neg p_i \text{ occurs in } C_j \\ j=1,..,m}} (\uparrow \mathrm{C}_j) = \mathrm{TRUE}$

(The conjunction symbol is usually omitted.) In this construction, the rules in (3) record for each variable p_i the clauses C_j that are true if p_i is true (by the annotations $(\uparrow \mathrm{C}_j) = \mathrm{TRUE}$ in (3a)) and the clauses that are true if $\neg p_i$ is true (3b). Thus, there must be a truth assignment for the variables that makes all clauses C_j of $C_1 \wedge .. \wedge C_m = \psi$ true if and only if G_ψ derives terminal string $\$^n$ with the f-structure

(4) $\begin{bmatrix} \mathrm{C}_1 & \mathrm{TRUE} \\ \vdots & \\ \mathrm{C}_m & \mathrm{TRUE} \end{bmatrix}$

in which for all clauses C_j the C_j attributes have value TRUE. Hence, ψ is satisfiable if and only if G_ψ derives a terminal string with the f-structure F_ψ in (4).

G_ψ has $2n + 1$ rules, a single start rule (2) of length n with n annotations and two rules (3) of constant length for each of the n propositional variables with a total of $3m$ annotations. The input structure F_ψ has size m (measured in the number of attribute-value pairs). The rules and the input can be constructed just by scanning ψ, which is of length $3m$, from left to right. However, in each step, the list of rules already built is scanned to check whether a new annotation has to be added to an existing P_i rule or a new P_i rule has to be created. During the same scan a new daughter is added to the start rule if needed. Thus, the total time needed to construct the grammar rules and the input is at most of order $m \cdot n$, a polynomial in the size of the original 3-SAT problem.

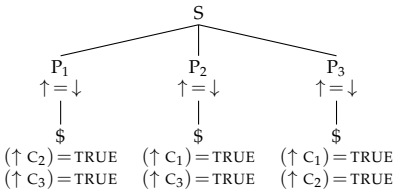

Figure 1
One of 5 annotated c-structure derivations that the LFG grammar with the rules in (6) provides for the input f-structure (7). This derivation corresponds to the truth-value assignment on which p_1 is false and p_2 and p_3 are true.

By construction, there are 2^n annotated c-structure derivations in G_ψ, for any 3-SAT instance ψ over n propositional variables, and all those derivations derive the string $\$^n$. Thus, in the worst case, 2^n annotated c-structure derivations may have to be examined to determine whether $(s, F_\psi) \in \Delta_{G_\psi}$, for any string s.[1]

As a simple illustration consider the satisfiable formula in (5).

(5) $\psi = C_1 \wedge C_2 \wedge C_3 = (p_1 \vee p_2 \vee p_3) \wedge (\neg p_1 \vee \neg p_2 \vee p_3) \wedge (\neg p_1 \vee p_2 \vee \neg p_3)$

For this formula the construction results in the rules in (6) and the input structure (7). The P rules in the left column reflect the positive literals and the ones in the right the negative ones.

(6) $\mathrm{S} \rightarrow \underset{\uparrow=\downarrow}{\mathrm{P}_1}\ \underset{\uparrow=\downarrow}{\mathrm{P}_2}\ \underset{\uparrow=\downarrow}{\mathrm{P}_3}$

$\mathrm{P}_1 \rightarrow \underset{(\uparrow \mathrm{C}_1)=\mathrm{TRUE}}{\$}$ $\qquad$ $\mathrm{P}_1 \rightarrow \underset{\substack{(\uparrow \mathrm{C}_2)=\mathrm{TRUE} \\ (\uparrow \mathrm{C}_3)=\mathrm{TRUE}}}{\$}$

$\mathrm{P}_2 \rightarrow \underset{\substack{(\uparrow \mathrm{C}_1)=\mathrm{TRUE} \\ (\uparrow \mathrm{C}_3)=\mathrm{TRUE}}}{\$}$ $\qquad$ $\mathrm{P}_2 \rightarrow \underset{(\uparrow \mathrm{C}_2)=\mathrm{TRUE}}{\$}$

$\mathrm{P}_3 \rightarrow \underset{\substack{(\uparrow \mathrm{C}_1)=\mathrm{TRUE} \\ (\uparrow \mathrm{C}_2)=\mathrm{TRUE}}}{\$}$ $\qquad$ $\mathrm{P}_3 \rightarrow \underset{(\uparrow \mathrm{C}_3)=\mathrm{TRUE}}{\$}$

(7) $\begin{bmatrix} \mathrm{C}_1 & \mathrm{TRUE} \\ \mathrm{C}_2 & \mathrm{TRUE} \\ \mathrm{C}_3 & \mathrm{TRUE} \end{bmatrix}$

There are 8 annotated c-structure derivations that this grammar provides for the terminal string \$\$\$, but only 5 of them are assigned the f-structure (7). One of those derivations is depicted in Figure 1.

1 On this analysis the number of derivations depends on the number of variables but not on the number of clauses m. Tovey (1982) relates these parameters by showing that 3-SAT is NP-complete for instances where each variable or its complement appears in at most four clauses. Thus, $3m \leq 4n$, and hence $n \geq \frac{3}{4}m$. This establishes that the number of annotated c-structure derivations that have to be inspected in order to solve the acyclic generation problem can also be exponential in (a fraction of) the number of clauses.

In order to guarantee decidability of the recognition problem, Kaplan and Bresnan (1982) introduced a constraint, later called the Off-line Parsability Constraint, that proscribes empty productions and nonbranching dominance chains and thus bounds the number and size of the c-structures of a string by a function of the length of that string. Because the grammars G_Ψ do not contain empty productions and do not produce nonbranching dominance chains, the acyclic generation problem is intractable even for off-line parsable LFGs.

Note also that a transposition of this reduction can be used to show the intractability of the recognition problem for off-line parsable LFGs. This transposition is intrinsically simpler than Berwick's original reduction (Berwick 1982). The grammar G'_Ψ includes the start rule

$$(8)\quad \mathrm{S} \rightarrow \underset{\uparrow=\downarrow}{\mathrm{C}_1} \;\cdots\; \underset{\uparrow=\downarrow}{\mathrm{C}_m}$$

and for each literal l_{j_k} of $l_{j_1} \vee l_{j_2} \vee l_{j_3} = C_j$ a terminal rule of the form (9).

$$(9)\quad \mathrm{C}_j \rightarrow \$ \qquad (\uparrow \mathrm{P}_i) = \begin{cases} \textsc{true} & \text{if } l_{j_k} = p_i \\ \textsc{false} & \text{if } l_{j_k} = \neg p_i \end{cases}$$

Here, the rules in (9) encode truth-value assignments for the variables that could separately satisfy each clause and the annotations of (8) ensure that the assignments are consistent. By construction, there are 3^m annotated c-structure derivations for a string of length m that may have to be inspected to determine whether there is an f-structure F with $(\$^m, F) \in \Delta_{G'_\Psi}$.

G'_Ψ has $3m + 1$ rules: One rule of constant size for each literal in each disjunctive clause C_j (i.e., in total $3m$ rules) and a single start rule of length m with m annotations. Because the rules for the literals of each clause are independent of the rules for the literals in other clauses, rescanning of already constructed rules is not required. Thus, the total time needed to construct the grammar rules is at most of order m.

We have demonstrated that LFG's acyclic generation and recognition problems can be reduced from the 3-SAT problem in polynomial time. Because the satisfiability of the f-description of a given annotated c-structure can be tested quickly, LFG's recognition problem is in NP and hence NP-complete. For the acyclic generation problem, however, it is not yet clear whether it belongs to NP, because the problem of deciding whether the input f-structure and the f-structure assigned to a given derivation are structurally identical is an instance of the isomorphism problem for labeled directed acyclic graphs. Because it is not yet known whether that problem can be solved in polynomial time (Basin 1994), with current knowledge we can only establish that the acyclic generation problem is NP-hard.

In these reductions, the size parameters n and m of the 3-SAT problem instances are reflected in certain size parameters of the corresponding LFG grammars, namely, the length of the rules and the number of attributes. These technical demonstrations reveal the expressive power of the basic LFG formalism, but they do not immediately carry over to the way that the formalism is deployed in linguistic practice. Grammars of natural language are not revised and specialized for every input that is presented for recognition or generation. Rather, they describe particular natural languages with a fixed number of rules and attributes that are intended to operate correctly on inputs

of arbitrary size. We can make our analysis more directly relevant by providing a grammatical framework with a fixed set of rules and attributes that can reduce 3-SAT problems of any size. In this framework the particular problem to be solved is not encoded in the grammar but is presented as the input, either as a string or an f-structure.

Reductions to Generic Grammars

We first describe the generic reduction for the recognition problem. We encode the specific literals p_i, $\neg p_i$ as sequences of i \$ terminals followed by $+$ or $-$ ($\$^i+$, $\$^i-$). The literal $\neg p_3$ is thus represented as the string $\$\,\$\,\$-$. A clause is represented as the concatenation of the representations of its 3 literals and a whole problem as the concatenation of the representations of its clauses. Hence, the 3-SAT formula (5) is represented as the terminal string $\$ + \$\,\$ + \$\,\$\,\$ + \$ - \$\,\$ - \$\,\$\,\$ + \$ - \$\,\$ + \$\,\$\,\$ -$. The LFG grammar consists of the 8 rules in (10). The S rules generalize the start rule in (8) to an unlimited number of clauses. As now string encodings of satisfiable 3-SAT instances are to be recognized, the C rules expand to three L daughters for deriving representations of the three literals and they guess, similar to (9) but now through trivial annotations, the true literal in each clause. The L rules derive the string representations of the literals $\$^i$ + or $\$^i$ − and assign to them attribute-chain encodings of the form $\text{P}^i\ \text{VAL}\ \text{TRUE}$ or $\text{P}^i\ \text{VAL}\ \text{FALSE}$.

(10) $\text{S} \rightarrow \underset{\uparrow=\downarrow}{\text{C}}\ \underset{\uparrow=\downarrow}{\text{S}}$ $\qquad \text{S} \rightarrow \underset{\uparrow=\downarrow}{\text{C}}$

$\text{C} \rightarrow \underset{\uparrow=\downarrow}{\text{L}}\ \text{L}\ \text{L}$ $\qquad \text{C} \rightarrow \text{L}\ \underset{\uparrow=\downarrow}{\text{L}}\ \text{L}$ $\qquad \text{C} \rightarrow \text{L}\ \text{L}\ \underset{\uparrow=\downarrow}{\text{L}}$

$\text{L} \rightarrow \$\ \underset{(\uparrow \text{P})=\downarrow}{\text{L}}$ $\qquad \text{L} \rightarrow \underset{(\uparrow \text{VAL})=\text{TRUE}}{+}$ $\qquad \text{L} \rightarrow \underset{(\uparrow \text{VAL})=\text{FALSE}}{-}$

In this construction, the trivial annotations ensure the consistency of the truth assignments guessed in the C rules because there will be a clash if two chain-encodings of the same variable bottom out in different VAL assignments. Thus a 3-SAT problem instance is satisfiable if and only if its terminal-string representation belongs to the language of the LFG. The terminal-string representation of a 3-SAT instance ψ with m clauses can be constructed by scanning ψ from left to right. Thus the total time needed to construct the input string is at most of order m.

The generic reduction for the acyclic generation problem is more involved. For convenience, we use the traditional parenthetic notation for optional annotated categories and optional annotations and $\{..|..\}$ for disjunction.

We define a generic grammar G and construct for any 3-SAT problem instance ψ an f-structure F_ψ such that G derives a terminal string with F_ψ if and only if ψ is satisfiable. F_ψ contains an encoding of ψ and a solution structure, and the units of both structures are linked by edges labeled with the attribute S.

We represent the problem with attribute-chain encodings for the literals and also for the clauses: p_i is represented by $\text{P}^i\ \text{POS}$, $\neg p_i$ by $\text{P}^i\ \text{NEG}$, and C_j by C^j. Then, a 3-SAT problem with n variables and m clauses is encoded in the input f-structure through chains that match the regular expression $\text{P}^i\ \{\text{POS}|\text{NEG}\}\ \text{C}^j\ \text{OCC}+$, with $i \leq n, j \leq m$, where $\text{P}^i\ \text{POS}\ \text{C}^j\ \text{OCC}+$ records that p_i occurs in C_j and $\text{P}^i\ \text{NEG}\ \text{C}^j\ \text{OCC}+$ records that

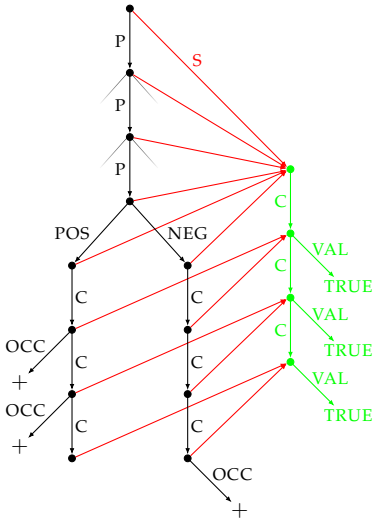

Figure 2
A schematic representation of the input f-structure for our 3-SAT example (5). The problem encoding is depicted in black, the solution structure in green, and the S edges in red. Only the encodings for p_3 and $\neg p_3$ are shown in detail.

$\neg p_i$ occurs in C_j. For our 3-SAT example (5), the problem encoding is schematically illustrated in the black substructure of the input f-structure depicted in Figure 2.

The solution structure corresponds to an attribute-chain conversion of the input structure (4) of the problem-specific reduction, that is, it is represented through attribute-value chains C^j VAL TRUE, $j = 1, .., m$. The S edges link the encoding of every literal to the root of the solution structure and the encoding of every clause to its encoding in the solution structure. For our 3-SAT example (5), these components of the input f-structure are depicted in green (solution structure) and red (S edges).

The rules in (11) derive the chain encodings for an arbitrary number of literals.

(11) $\text{S} \rightarrow \quad \begin{array}{c} \text{P} \\ (\uparrow \text{P}) = \downarrow \\ (\uparrow \text{S}) = (\downarrow \text{S}) \end{array}$

$$\text{P} \rightarrow \left(\begin{array}{c} \text{P} \\ (\uparrow \text{P}) = \quad \downarrow \\ (\uparrow \text{S}) = (\downarrow \text{S}) \end{array} \right) \left\{ \begin{array}{cc} \text{C}_{\text{TRUE}} & \text{C} \\ (\uparrow \text{POS}) = \downarrow & (\uparrow \text{NEG}) = \downarrow \\ (\uparrow \text{S}) = (\downarrow \text{S}) & (\uparrow \text{S}) = (\downarrow \text{S}) \end{array} \middle| \begin{array}{cc} \text{C} & \text{C}_{\text{TRUE}} \\ (\uparrow \text{POS}) = \downarrow & (\uparrow \text{NEG}) = \downarrow \\ (\uparrow \text{S}) = (\downarrow \text{S}) & (\uparrow \text{S}) = (\downarrow \text{S}) \end{array} \right\}$$

The S reentrancies link all encodings to the same solution structure. The disjunction guesses either the positive or the negative literal to be true and this nondeterministic guess is marked at the end of the derivation of the literals for a variable by the TRUE tag.

The C productions in (12) allow it to encode through the optional OCC + annotations for each literal all possible occurrences in an arbitrary number of clauses, and the S reentrancies link every derived clause representation to the representation of that clause in the solution structure.

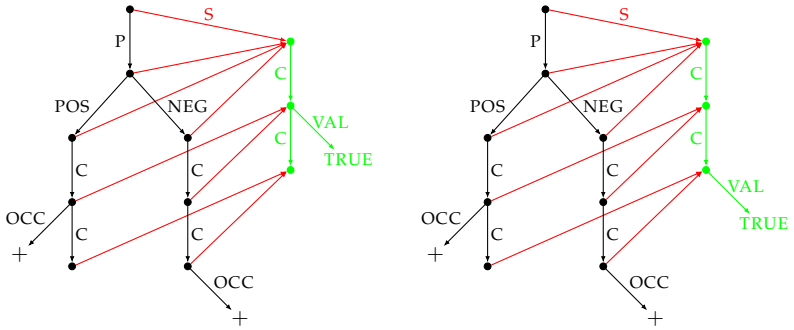

Figure 3
For the encoding of the simple unsatisfiable 3-SAT problem $(p_1 \vee p_1 \vee p_1) \wedge (\neg p_1 \vee \neg p_1 \vee \neg p_1)$, the TRUE guess for the POSitive literal of the P rule results in the f-structure on the left side and the alternative TRUE guess for the NEGative in the one on the right side (if the option OCC + of the L rules is exactly selected for those clauses in which the literal occurs). Because both structures are missing the VAL TRUE information at one clause representation, the problem is unsatisfiable.

$$(12)\quad \mathrm{C_{TRUE}} \rightarrow \left(\begin{array}{c}\mathrm{C_{TRUE}}\\(\uparrow \mathrm{C}) = \downarrow\\(\uparrow \mathrm{S\,C}) = (\downarrow \mathrm{S})\\\left(\begin{array}{c}(\downarrow \mathrm{OCC}) = +\\((\downarrow \mathrm{S\ VAL}) = \mathrm{TRUE})\end{array}\right)\end{array}\right) \$ \qquad \mathrm{C} \rightarrow \left(\begin{array}{c}\mathrm{C}\\(\uparrow \mathrm{C}) = \downarrow\\(\uparrow \mathrm{S\,C}) = (\downarrow \mathrm{S})\\((\downarrow \mathrm{OCC}) = +)\end{array}\right) \$$

The given rules can certainly derive for any 3-SAT problem ψ a string with an f-structure that contains the encoding of ψ. To see that the rules derive the problem encoding together with the solution structure only if ψ is satisfiable, let us consider the derivations that yield the problem encoding for a particular 3-SAT instance with n variables and m clauses. Because the disjunction of the P rules encodes the possible truth assignments to a variable, the recursive application of the P rule alternatives encodes in 2^n derivations all possible truth assignments for the n variables of the given problem. By construction only $\mathrm{C_{TRUE}}$ but not C rules expand derivations for literals that are assigned true. Also, if a literal occurs in C_j then only those rules add VAL TRUE to the encoding of C_j in the solution structure, to indicate that the variable assignment that makes that literal true also makes C_j true.

Thus, if a clause representation in the solution structure is missing the VAL TRUE information, the truth assignment encoded in that derivation does not make that clause true. Hence, individual derivations match the input if and only if there is at least one variable assignment that satisfies every clause. A problem is unsatisfiable if no derivation matches the input. This can be made particularly clear by the trivial unsatisfiable 3-SAT problem instance $(p_1 \vee p_1 \vee p_1) \wedge (\neg p_1 \vee \neg p_1 \vee \neg p_1)$, even though it does not comply with the simplifying assumptions that we made at the beginning. For the encoding of this problem, the LFG grammar provides the derivations depicted in Figure 3.

The generic LFG has 11 rules. The input f-structure, which has maximum depth $n + m + 2$, can be constructed by scanning ψ (with length $3m$) from left to right and by creating in each step an occurrence encoding of a literal in a clause. Because the input

structure has to be scanned top-down in each step, the total time needed to construct the input is at most of order $m^2 + mn$.[2]

Concluding Remarks

This note has introduced new results concerning the complexity of LFG generation (and recognition) for grammars that assign acyclic f-structures to input strings. We observed that reductions to problem-specific grammars where the size parameters of the 3-SAT problem instances are reflected in grammar-size parameters are not particularly relevant to the linguistic enterprise. More relevant are the generic reductions where the LFG grammar is kept fixed across all possible 3-SAT problem instances. These show that computational complexity in the worst case can grow exponentially as a function of the size of the input f-structure or string, and that grammar or derivation restrictions of some sort must be imposed for tractability of LFG generation and recognition.

Wedekind and Kaplan (2020) introduced a subclass of LFG grammars with particular restrictions that exclude our generic reduction grammars but ensure that at least the recognition problem is tractable for inputs of arbitrary length. Wedekind and Kaplan further argue that grammars that meet the conditions of this *k*-bounded subclass are still expressive enough for natural language description (see also Kaplan and Wedekind 2019). However, it is at this point still an open question whether generation is polynomial for arbitrary grammars in this linguistically plausible subclass or whether polynomial generation can be established only for grammars within a yet more restricted proper subclass of the *k*-bounded class.

Acknowledgments

The authors would like to thank the three anonymous reviewers for their helpful suggestions and comments on earlier drafts of this squib.

References

Basin, David A. 1994. A term equality problem equivalent to graph isomorphism. *Information Processing Letters*, 51(2):61–66. https://doi.org/10.1016/0020-0190(94)00084-0

Berwick, Robert C. 1982. Computational complexity and Lexical-Functional Grammar. *American Journal of Computational Linguistics*, 8(3–4):97–109.

Kaplan, Ronald M. and Joan Bresnan. 1982. Lexical-Functional Grammar: A formal system for grammatical representation. In Joan Bresnan, editor, *The Mental Representation of Grammatical Relations*. MIT Press, Cambridge, MA, pages 173–281.

Kaplan, Ronald M. and Jürgen Wedekind. 2019. Tractability and discontinuity. In *Proceedings of the International Lexical-Functional Grammar Conference 2019*, pages 130–148, Stanford, CA.

Tovey, Craig A. 1982. A simplified NP-complete satisfiability problem. *Discrete Applied Mathematics*, 8(1):85–89.

Wedekind, Jürgen. 2014. On the universal generation problem for unification grammars. *Computational Linguistics*, 40(3):533–538. https://doi.org/10.1162/COLI_a_00191

Wedekind, Jürgen and Ronald M. Kaplan. 2012. LFG generation by grammar specialization. *Computational Linguistics*, 38(4):867–915. https://doi.org/10.1162/COLI_a_00113

Wedekind, Jürgen and Ronald M. Kaplan. 2020. Tractable Lexical-Functional Grammar. *Computational Linguistics*, 46(3):515–569. https://doi.org/10.1162/coli_a_00384

2 Note that the generic reduction for generation, as well as the problem-specific reductions, also work for general SAT problem instances, and that the 3-literal rule for the recognition problem can easily be extended so that it solves such problem instances too.