Analyzing Context-Free Grammars Using an Incremental SAT Solver

Roland Axelsson[1], Keijo Heljanko[2,⋆], and Martin Lange[1]

[1] Institut für Informatik, Ludwig-Maximilians-Universität München, Germany
[2] Department of Information and Computer Science,
Helsinki University of Technology (TKK), Finland

Abstract. We consider bounded versions of undecidable problems about context-free languages which restrict the domain of words to some finite length: inclusion, intersection, universality, equivalence, and ambiguity. These are in (co)-NP and thus solvable by a reduction to the (un-)satisfiability problem for propositional logic. We present such encodings – fully utilizing the power of incrementat SAT solvers – prove correctness and validate this approach with benchmarks.

1 Introduction

Context-free grammars (CFG) and languages (CFL) have been used intensively by computer scientists and linguists since Chomsky formalized them in 1956. They have applications in compiler design, speech processing, bioinformatics, static program analysis, XML processing, etc.

The word problem for CFGs is decidable in cubic time and quadratic space and the Pumping Lemma for CFLs [1] provides a criterion by which the emptiness problem becomes decidable as well. However, it has since long been known that the following problems are undecidable: *universality* (given a CFG G over some alphabet Σ, is $L(G) = \Sigma^*$?); *inclusion*, *intersection*, and *equivalence* (given two CFG G_1 and G_2, is $L(G_1) \subseteq L(G_2)$, is $L(G_1) \cap L(G_2) = \emptyset$, and is $L(G_1) = L(G_2)$?)

Another very important undecidable problem is *ambiguity* – is there a word which has at least two different parse trees w.r.t. a given CFG? After seeing little progress for many years, this problem has recently attracted attention again [3,8] which is, e.g., due to its importance in compiler design and bioinformatics.

Due to decidability of the word problem these problems are all (co)-semi-decidable through an enumeration of Σ^*. Hence *bounded* versions of these problems become decidable. For example, the *bounded universality* problem is: given a CFG G and a $k \in \mathbb{N}$, does $L(G)$ contain all words of length $\leq k$? Since the word problem is even decidable in polynomial time, they are in (co-)NP and can therefore be solved by a polynomial reduction to (UN-)SAT, provided that k is given in unary coding.

⋆ Work financially supported by Academy of Finland (project 112016) and Technology Industries of Finland Centennial Foundation.

L. Aceto et al. (Eds.): ICALP 2008, Part II, LNCS 5126, pp. 410–422, 2008.

The use of modern SAT solvers such as zChaff [6] has proved to be extremely beneficial in areas like computer aided verification, AI planning, theorem proving, cryptanalysis, electronic design automation, etc. Here we show that, apparently, formal language theory is also among them.

The observation about decidability of the above bounded problems is not new although we are not aware of any work that exploits this idea thoroughly in order to tackle the unbounded (and undecidable) problems. Here we present optimized reductions of these bounded problems to SAT s.t. a SAT solver can find witnesses, resp. counterexamples for these problems. The basis for the reduction is the well-known CYK algorithm [9]. We generate propositional logic constraints encoding which nonterminals may/must occur at certain positions in a CYK parse table. A slightly different approach to encoding CYK parsing into SAT has been independently discovered in [7]. However, the textbook version of CYK is unsuitable since it requires the CFG to be in Chomsky Normal Form (CNF) which may incur an exponential blow-up in the grammar. That would clearly be counterproductive in the search for optimized reductions. We therefore develop and use an optimized version of CYK which may be known in the community but does not seem to have made it into the literature. When it comes to ambiguity, we even must. Note that the transformation into CNF does not preserve ambiguity. We therefore use a different normal form without these deficiencies.

The crucial difference between our symbolic encoding and an explicit execution of the CYK algorithm though, is the absence of an input word. This has to be "guessed" by the SAT solver and the constraints will ensure (non)-inclusion in the languages of some CFGs. Thus, we do not only get that the SAT formula is satisfiable iff the language of the CFG, say, contains an ambiguous word of length $\leq k$. The satisfying assignment also encodes this word as well as two different parse trees.

Note that CYK tables are in some sense closed under extension "to the right": the triangular table of size $(k+1) \times (k+1)$ can be obtained from the one of size $k \times k$ by adding a column of length $k+1$. This is what makes *incremental* SAT solving predestined for solving bounded CFL problems. If a witness/counterexample of size $\leq k$ is not found, additional constraints for a greater bound plus a few changes in the current constraints yield the new formula. Incremental SAT solvers maximally utilize information gathered in solving a SAT instance to solve the next "bigger" but structurally very similar one. Such solvers are therefore of particular interest for our setting.

This is clearly just a semi-decision procedure for the unbounded versions of the considered problems. But it has distinct advantages over approximation approaches for ambiguity [3,8]. While the accuracy of answers given by those depends on the quality of the approximation (that may produce false-positives), our approach is only limited by time and available memory; the structure of the produced formula does not pose any difficulty to the SAT solver. A report on an empirical evaluation is included after some preliminary definitions, and the presentation as well as exemplary correctness proofs of encodings for the above problems. On the other hand, our approach is clearly not complete and not

meant to replace approximation approaches to ambiguity. Instead, they could also be combined, e.g. to provide a search for smallest witnesses to the problems of horizontal and vertical ambiguity in [3] for instance.

2 Preliminaries

Let Σ be an alphabet. As usual, we write $|w|$ for the length of a word w, Σ^* for the set of finite words over Σ, $\Sigma^{\leq k}$ for $\{w \in \Sigma^* \mid |w| \leq k\}$ for any $k \in \mathbb{N}$, ϵ for the empty word and uv or LL' to denote concatenation of words, sets, languages, etc. If $w = a_1 \dots a_n$ we write $w^{i..j}$ for its subword $a_i \dots a_j$ of length $j - i + 1$. A *context-free grammar* is a tuple $G = (N_G, \Sigma, P_G, S_G)$ where N_G is a finite set of non-terminals, Σ is an alphabet, $N_G \cap \Sigma = \emptyset$, $S_G \in N_G$ is the starting symbol and $P_G \subset N_G \times (N_G \cup \Sigma)^*$ is a finite set of production rules. We use infix notation $A \to \alpha$ to denote $(A, \alpha) \in P_G$. As the size of G we define $|G| := |N| + \sum_{A \to \alpha} |\alpha|$.

The *derivation* relation $\Rightarrow_G \subseteq (N \cup \Sigma)^+ \times (N \cup \Sigma)^*$ is defined as $\alpha A \beta \Rightarrow_G \alpha\gamma\beta$ iff $A \to \gamma \in P$ for $\alpha, \beta, \gamma \in (N \cup \Sigma)^*, A \in N$. We will drop the index G if it becomes clear from the context.

$L(G) := \{w \in \Sigma^* \mid S_G \Rightarrow^* w\}$ is the *language* of G. An alternative way to define the derivation of a word is via the existence of *parse trees*. We assume the reader familiar with these fundamental concepts and refer to [4] for details.

A word w is *ambiguous* w.r.t. a CFG G, if there are two different parse trees for w w.r.t. G. For a CFG G let $amb(G)$ denote the set of words that are ambiguous w.r.t. G. G itself is called *ambiguous* if $amb(G) \neq \emptyset$.

In the following we will always assume the context-free languages under consideration not to contain the empty word ϵ. This is not a restriction but simplifies the presentation.

Definition 1. A CFG is in *binary normal form* (2NF), if for all $(A \to \alpha) \in P_G$ we have $\alpha \in \{\epsilon\} \cup \Sigma \cup N \cup NN$. It is *acyclic* if for all $A \neq B \in N$ we have, if $A \Rightarrow^* B$ then $B \not\Rightarrow^* A$. It is *reduced* if all nonterminals are *reachable* and *productive*, i.e. for all $A \in N$ there are sentential forms α, β and a word $w \in \Sigma^*$ s.t. $S \Rightarrow^* \alpha A \beta$ and $A \Rightarrow^* w$.

Lemma 1. *For every CFG G there is a CFG G' in reduced acyclic 2NF, computable in time $\mathcal{O}(|G|^2)$ s.t. $L(G) = L(G')$ and $|G'| = \mathcal{O}(|G|)$.*

Lemma 2. *Let G be an acyclic grammar. There exists a well-founded strict partial order $> \subseteq N_G^2$, s.t. if $A \Rightarrow^* B$ and $A \neq B$ then $A > B$.*

3 The Encoding

The task is now to create propositional logic constraints from a CFG G and a $k \in \mathbb{N}$ that are satisfiable iff $L(G)$ is (k-bounded) universal, ambiguous, etc. Let $G = (N, \Sigma, P, S)$ in reduced, acyclic 2NF and k be fixed. We use two kinds

of propositional variables: X_i^a for every $a \in \Sigma$ and every $1 \leq i \leq k$ stating that the i-th symbol of a witnessing word is a. An assignment to these variables corresponds to the choice of a witness w. The other kind is $X_{i,j}^A$ and states that nonterminal A derives the subword $w^{i..j}$. Let $\mathcal{P} = \{X_i^a, X_{i,j}^A \mid a \in \Sigma, A \in N, 1 \leq i \leq j \leq k\}$. In the following, η will denote an assignment of these variables to $\{\mathtt{tt}, \mathtt{ff}\}$, and we write $\eta(\mathcal{C}) = \mathtt{tt}$ if η satisfies all the constraints $\mathcal{C}$ under the usual interpretation of the operators in propositional logic.

With incrementality in mind we define constraint sets w.r.t. some p, p' s.t. $1 \leq p \leq p' \leq k$. Intuitively, these contain the constraints for the columns $p, \ldots, p'$ of a CYK table. Note that they may use variables with indices below p.

Let $\Phi = \{\varphi_1, \ldots, \varphi_n\}$ be an ordered set of propositional formulas. There is a standard trick to state that at most one of them holds by introducing auxiliary variables Y_i for $1 \leq i \leq n+1$.

$$One(\Phi) \;:=\; \{\; (\varphi_i \rightarrow \neg Y_i \wedge Y_{i+1}) \wedge (Y_i \rightarrow Y_{i+1}) \;\mid\; 1 \leq i \leq n \;\}$$

With this macro, we can easily state that each position in a witnessing word is occupied by a unique symbol.

$$\mathcal{W}(p, p') \;:=\; \{\; One(\{X_i^a \mid a \in \Sigma\}) \wedge \bigvee_{a \in \Sigma} X_i^a \;\mid\; p \leq i \leq p' \;\}$$

Lemma 3. *For any η, $\eta(\mathcal{W}(p, p')) = \mathtt{tt}$ iff there exists a unique sequence $b_p, \ldots, b_{p'}$, s.t. $\eta(X_i^{a_i}) = \mathtt{tt}$ iff $b_i = a_i$ for all $p \leq i \leq p', a_i \in \Sigma$.*

We will therefore simply write $w_\eta^{p,p'}$ for the unique $w^{p..p'}$ induced by η. We encode a derivation with the help of constraints $\mathcal{R}(p, p') :=$

$$\{\; X_{i,j}^A \leftrightarrow \bigvee_{\substack{A \rightarrow a \\ if\ i=j}} X_i^a \;\vee\; \bigvee_{\substack{A \rightarrow B \\ A \neq B}} X_{i,j}^B \;\vee\; \bigvee_{\substack{A \rightarrow BC\, or \\ A \rightarrow CB \\ A \neq B, C \Rightarrow^* \epsilon}} X_{i,j}^B \vee \bigvee_{A \rightarrow BC} \bigvee_{h=i}^{j-1} (X_{i,h}^B \wedge X_{h+1,j}^C)$$
$$\mid A \in N, p \leq i \leq j \leq p' \}$$

This encoding splits up the derivation of $w_\eta^{i,j}$ by non-terminal A into the following four cases (marked by the big disjunctions): derivation of a single terminal, two cases of single non-terminal derivations and the derivation of composites. Note that pre-computing the set of all nonterminals C s.t. $C \Rightarrow^* \epsilon$ can be done in time $\mathcal{O}(|G|)$. It is also a necessary preliminary step during the transformation into 2NF. So far, $\mathcal{R}(p, p')$ contains a bi-implication. However, for some problems, implications in one direction only will suffice. For example, when encoding bounded emptiness, the $\leftarrow$-parts are unnecessary. In general, the $\rightarrow$-parts express soundness of the encoding and are used to express that something is derivable; the $\leftarrow$-parts encode completeness and can be used to express that a word is *not* derivable. We write $\mathcal{R}^{\rightarrow}(p, p')$ and $\mathcal{R}^{\leftarrow}(p, p')$ for the soundness, resp. completeness parts only.

Lemma 4. *Let $k > 0$, η be an assignment s.t. $\eta(\mathcal{R}^{\rightarrow}(1, k) \cup \mathcal{W}(1, k)) = \mathtt{tt}$ and $w = w_\eta^{1,k}$. Then for all $A \in N_G$, all $1 \leq i \leq j \leq k$ we have: if $\eta(X_{i,j}^A) = \mathtt{tt}$ then $A \Rightarrow^* w^{i..j}$.*

Proof. Suppose $\eta(X^A_{i,j}) = \mathtt{tt}$. We prove the claim by induction on $j - i$ where we refer to the four different (big) disjunctions in $\mathcal{R}^{\rightarrow}(1,k)$ as "blocks 1–4".

Base case ($i = j$): Clearly, at least one variable from blocks 1-3 has to be evaluated to $\mathtt{tt}$. Block 4 evaluates to $\mathtt{ff}$ for $i = j$.

1. $\eta(X^a_i) = \mathtt{tt}$. There must be a rule $A \to w^i$ and therefore $A \Rightarrow^* w^i$.
2. $\eta(X^B_{i,i}) = \mathtt{tt}$. We proceed by well-founded induction on $>$. Suppose for all $B < A$, we have $B \Rightarrow^* w^i$ if $\eta(X^B_{i,i}) = \mathtt{tt}$. Because of the rule $A \to B$ we also have $A \Rightarrow^* w^i$.
3. $\eta(X^B_{i,i}) = \mathtt{tt}$. Analogous to (2).

Inductive case ($i < j$): Block 1 evaluates to $\mathtt{ff}$ for $i < j$, so at least one disjunct from blocks 2–4 has to evaluate to $\mathtt{tt}$.

1. $\eta(X^B_{i,j}) = \mathtt{tt}$. Same as in the base case.
2. $\eta(X^B_{i,h}) = \eta(X^C_{h+1,j}) = \mathtt{tt}$ (4). In particular, h, B, C exist and $i \leq h < j$. Clearly $h - i \leq j - i$ and $j - (h+1) \leq j - i$ and therefore by induction hypothesis $B \Rightarrow^* w^{i..h}$ and $C \Rightarrow^* w^{h+1..j}$. As $A \to BC$ it follows that $A \Rightarrow^* w^{i..j}$. □

Lemma 5. *Let $k > 0$, η be an assignment s.t. $\eta(\mathcal{R}^{\leftarrow}(1,k) \cup \mathcal{W}(1,k)) = \mathtt{tt}$ and $w = w^{1,k}_\eta$. Then for all $A \in N_G$, all $1 \leq i \leq j \leq k$ we have: if $A \Rightarrow^* w^{i..j}$ then $\eta(X^A_{i,j}) = \mathtt{tt}$.*

Proof. Again, we prove this by induction on $j - i$. Let $w = w_\eta$. In the base case suppose $A \Rightarrow^* w^{i..i} = a$. Thus, $\eta(X^a_i) = \mathtt{tt}$. Furthermore, there is a derivation tree with root A and leaf front a. Clearly, whenever a node in this tree has two successors labeled B and C then $B \Rightarrow^* \epsilon$ or $C \Rightarrow^* \epsilon$. Because of 2NF, a must be generated by some rule $B \to a$, and because of block 1 we have $\eta(X^B_{i,i}) = \mathtt{tt}$. A separate induction on the height of the tree – using blocks 2–4 – shows that we have $\eta(X^C_{i,i}) = \mathtt{tt}$ for all predecessors of this B in this tree, including the root A. The crucial insight to the applicability of this induction is the fact that in this parse tree the node labels on the path from the root to the leaf a are strictly decreasing w.r.t. $>$ according to Lemma 2.

Now assume $j > i$ and $A \Rightarrow^* w^{i..j}$. Hence, we have a parse tree t with root A whose leaf front is $w^{i..j}$. For a node n in t we write $w(n)$ for the subword of $w^{i..j}$ that constitutes of the leaf labels in the subtree under n. Furthermore, for two words u, v we write $u \prec v$ if u is a genuine subword of v.

Note that $|w^{i..j}| \geq 2$, and – because of 2NF – leaves in this tree have a direct predecessor n that can only have a single successor. Therefore, for each such n we have $w(n) \prec w^{i..j}$. Note that $w(n_0) = w^{i..j}$ for n_0 the root of t. Hence, there must be a highest (closest to the root) node n in this tree, that is labeled with some $B \in N_G$ and has two successors n_1 and n_2 labeled with some C, resp. D, s.t. $w(n_1) \prec w(n) = w(n_0)$ and $w(n_2) \prec w(n) = w(n_0)$. Hence, $w(n_1) = w^{i..h}$ and $w(n_2) = w^{h+1..j}$ for some $i \leq h < j$. But then we have $C \Rightarrow^* w^{i..h}$, $D \Rightarrow^* w^{h+1..j}$, and, by hypothesis, $\eta(X^C_{i,h}) = \eta(X^D_{h+1,j}) = \mathtt{tt}$. Since $B \to CD$ we have $\eta(X^B_{i,j}) = \mathtt{tt}$ by block 4. Finally, the path from the node labeled B to

	$\mathcal{R}_\mathcal{C}(p,p')$			
$\mathcal{C}$	$\mathcal{R}^{\rightarrow}_{G}(p,p')$	$\mathcal{R}^{\leftarrow}_{G}(p,p')$	$\mathcal{R}^{\rightarrow}_{G'}(p,p')$	$\mathcal{R}^{\leftarrow}_{G'}(p,p')$
$\text{bINCL}_{G,G'}$	**X**			**X**
bUNIV_{G}		**X**		
$\text{bISECT}_{G,G'}$	**X**		**X**	
$\text{bEQUIV}_{G,G'}$	**X**	**X**	**X**	**X**

Fig. 1. How to use the $\mathcal{R}$-constraints

the root A must be strictly increasing w.r.t. $<$ again, and an induction on its length eventually shows $\eta(X^A_{i,j}) = \mathtt{tt}$ using blocks 1–3. □

3.1 Constraints for Particular Problems

We will now assemble the above constraints in order to obtain encodings of the following problems. Let G, G' be CFG and $k > 0$.

- *Bounded Inclusion* (bINCL): does $\forall w \in \Sigma^{\leq k} : w \in L(G) \Rightarrow w \in L(G')$ hold?
- *Bounded Universality* (bUNIV): is $\Sigma^{\leq k} \subseteq L(G)$?
- *Bounded Intersection* (bISECT): is there a $w \in \Sigma^k \cap L(G) \cap L(G')$?
- *Bounded Equivalence* (bEQUIV): does $\forall w \in \Sigma^{\leq k} : w \in L(G) \Leftrightarrow w \in L(G')$ hold?

For those that take two CFG G, G' as input we write $\mathcal{R}_G$ to clarify which CFG the constraints refer to.

The following is not hard to prove using the fact that the word problem for a CFG can be solved in polynomial time. Note that bounded ambiguity is missing. It will be treated separately below.

Proposition 1. *For unarily encoded $k \in \mathbb{N}$, the problems* bINCL*,* bUNIV*,* bEQUIV *are in co-NP, and* bISECT *is in NP.*

All of these encodings have a similar structure: they take some form of the $\mathcal{R}$-constraints plus a single problem specific one constraining the grammar's starting symbols. We therefore define

$$\mathcal{C}(p,p') \;=\; \mathcal{W}(p,p') \;\cup\; \mathcal{R}_\mathcal{C}(p,p') \;\cup\; \mathcal{S}_\mathcal{C}(p,p')$$

for $\mathcal{C} \in \{\text{bINCL}, \text{bUNIV}, \text{bISECT}, \text{bEQUIV}\}$. The $\mathcal{R}$-parts can be obtained from the table in Fig. 1. The $\mathcal{S}$-part is always a single constraint $\mathcal{S}_\mathcal{C} := \{\bigvee_{j=p}^{p'} T_\mathcal{C}(j)\}$ with

$$T_{\text{bINCL}}(j) := X^{S_G}_{1,j} \wedge \neg X^{S_{G'}}_{1,j} \qquad T_{\text{bUNIV}}(j) := \neg X^{S_G}_{1,j}$$
$$T_{\text{bISECT}}(j) := X^{S_G}_{1,j} \wedge X^{S_{G'}}_{1,j} \qquad T_{\text{bEQUIV}}(j) := X^{S_G}_{1,j} \leftrightarrow \neg X^{S_{G'}}_{1,j}$$

We write bINCL(p) for bINCL($1, p$), etc. The following theorem confirms the introductory statement about the reductions from these bounded problems to SAT being polynomial. Its proof is straight-forwardly based on standard techniques for obtaining conjunctive normal form and therefore not presented here.

Proposition 2. *Let G, G' be CFGs, $k > 0$. For any set of constraints $\mathcal{C} \in \{$ bINCL*(k)*, bUNIV*(k)*, bISECT*(k)*, bEQUIV*(k) *$\}$ there is an equivalent propositional formula $\Phi_{\mathcal{C}}$ in conjunctive normal form over $\mathcal{O}(|N_G \cup N_{G'}| \cdot k^2)$ many variables s.t. $|\Phi_{\mathcal{C}}| = \mathcal{O}((|G| + |G'|) \cdot k^3)$.*

We will prove correctness of one of these reductions, namely for bINCL. The others are proved in a similar way.

Theorem 1. *Let G, G' be CFGs in reduced acyclic 2NF, $k > 0$. Then* $\text{bINCL}_{G,G'}(k)$ *is satisfiable iff there is a $w \in \Sigma^{\leq k}$ s.t. $w \in L(G) \setminus L(G')$.*

Proof. ($\Rightarrow$) Suppose η is a satisfying evaluation of $\text{bINCL}_{G,G'}(1, k)$. Let $w = w_\eta^{1,k}$ according to Lemma 3. We will show that there is a $k' \leq k$ s.t. $S_G \Rightarrow^* w^{1..k'}$ and $S_{G'} \not\Rightarrow^* w^{1..k'}$. Let k' be the least j s.t. $\eta(X_{1,j}^{S_G}) = \texttt{tt}$ and $\eta(X_{1,j}^{S_{G'}}) = \texttt{ff}$. Its existence is guaranteed by the specific constraints for bINCL. The rest follows immediately from Lemmas 4 and 5.

($\Leftarrow$) W.l.o.g we assume that the counterexample w is of minimal length k, i.e. that $\text{bINCL}_{G,G'}(k')$ is unsatisfiable for any $k' < k$. We construct an evaluation η of $\text{bINCL}_{G,G'}(1, k)$ as follows.

$$\eta(X_i^a) = \texttt{tt} \text{ iff } w^i = a \qquad \eta(X_{i,j}^A) = \texttt{tt} \text{ iff } A \Rightarrow^* w^{i..j}$$

for all $A \in N_G \cup N_{G'}$, $1 \leq i \leq j \leq k$. A simple inspection of the constraints in bINCL(k) shows that they are all fulfilled by η. □

Theorem 2. *Let G, G' be CFGs in 2NF, $k > 0$. Then we have*

- $\text{bUNIV}_G(k)$ *is satisfiable iff there is a $w \in \Sigma^{\leq k}$ s.t. $w \notin L(G)$.*
- $\text{bISECT}_{G,G'}(k)$ *is satisfiable iff there is a $w \in \Sigma^{\leq k}$ s.t. $w \in L(G) \cap L(G')$.*
- $\text{bEQUIV}_{G,G'}(k)$ *is satisfiable iff there is a $w \in \Sigma^{\leq k}$ s.t. $w \in L(G) \setminus L(G')$ or $w \in L(G') \setminus L(G)$.*

A counterexample for the universality problem could therefore be found by iteratively checking the constraint sets bUNIV(1), bUNIV(2), ... for satisfiability. Note that bUNIV($k+1$) contains many constraints already present in bUNIV(k). In fact, for all of the above problems we have the following relation. Let $0 < k < k'$.

$$\mathcal{C}(k') \;=\; \big(\mathcal{C}(k) \setminus \bigcup_{1 \leq p \leq p' \leq k} \mathcal{S}_{\mathcal{C}}(p, p')\big) \cup \mathcal{C}(k+1, k') \cup \mathcal{S}_{\mathcal{C}}(k+1, k')$$

Hence, these constraints support incrementality in the sense that the wider range $\Sigma^{\leq k'}$ can be checked by modifying the constraints for the smaller range $\Sigma^{\leq k}$. Furthermore, the increase need not take place in steps of size 1 only.

3.2 Ambiguity

We define the bounded ambiguity problem bAMB for a grammar G in reduced acyclic 2NF and a $k \geq 1$ in a non-obvious way: is there a nonterminal $A \in N_G$ and a word $v \in \Sigma^{\leq k}$ s.t. v has at least two different parse trees with roots labeled A that differ in a node on level 1?

Note that a word w is ambiguous in the original sense w.r.t. a grammar G iff it has two different parse trees that differ in a node (determined by the derived subword under that node and the node's label) which is not the root. Therefore, these trees must have a subtree each with equally labeled roots and equal derived subwords that differ on level 1. In other words, a derivation for w derives a subword v from a nonterminal A by using two different rules for A or using one rule in two different ways.

By not looking for ambiguous words, but ambiguous subwords, found witnesses explain the reason for ambiguity more clearly. For example, if the examined grammar was an ambiguous one for Java, then the witness may not be a whole Java program but just an ambiguous Java expression. Furthermore, this definition of bounded ambiguity allows for much more compact encodings. Finally, if a CFG is reduced, i.e. all terminals are reachable and productive then we have the following property: if (v, A) is an instance of bAMB for a CFG G as defined above, then there is an ambiguous $w \in L(G)$ s.t. $w = uvz$ for some $u, z \in \Sigma^*$. The converse direction holds trivially. Thus, bounded ambiguity in our sense is just a more detailed description of bounded ambiguity as one may expect it.

Proposition 3. *The problem* bAMB *is solvable in NP for unarily encoded* $k \in \mathbb{N}$.

Before we can present the encoding we need to reconsider the transformation of a CFG into reduced acyclic 2NF. Remember that acyclicity is necessary for the $\mathcal{R}$-constraints to be correct. However, it requires the removal of productions of the form $A \to A$ after replacing nonterminals with equivalence class representants in the construction of Lemma 1. But then the transformation does not preserve ambiguity anymore, because such a cyclic rule can be its cause.

Definition 2. An extended CFG is a tuple $G = (N, \Sigma, P, S, M, E)$ like a CFG with $E \subseteq M \subseteq N$ called the *ambiguously nullable nonterminals* and the *ambiguous nonterminals*. The notions of language, derivability, 2NF, acyclicity, reducedness etc. are defined as for a CFG. However, we define $amb(G) = \{w \mid$ there are two different parse trees for w, or there is one parse tree containing a nonterminal $A \in M\ \}$.

Then we can reformulate Lemma 1 for the new purpose as follows.

Lemma 6. *For every CFG* G *there is an extended CFG* G' *in acyclic and reduced 2NF, computable in time* $\mathcal{O}(|G|^2)$ *s.t.* $L(G') = L(G)$*, and* $|G'| = \mathcal{O}(|G|)$*. Moreover, we have* $amb(G') = amb(G)$*, and* $A \in E$ *iff there are two different parse trees with root* A *and leaf front* ϵ*.*

Proof. Let $G = (N, \Sigma, P, S)$ be a CFG. It can be reduced and transformed into 2NF in time $\mathcal{O}(|G|)$. Define $G' := (\tilde{N}, \Sigma, \tilde{P}, \tilde{S}, M, E)$ as the canonical factorisation of G under the equivalence relation $A \sim B$ iff $A \Rightarrow^* B \Rightarrow^* A$. I.e. its non-terminals are equivalence classes $\tilde{A}$ under this relation, and the production rules of G' are canonically derived from those in G. It should be clear that G' is also reduced. Let E consist of all $\tilde{A}$ that can derive ϵ in at least two different ways. This can be computed in time $\mathcal{O}(|G'|)$. Define $M := E \cup \{\tilde{A} \mid A \Rightarrow^+ A\}$. Note that M can be computed in time $\mathcal{O}(|G|^2)$. In order to make G' acyclic, simply remove all productions of the form $\tilde{A} \to \tilde{A}$.

It is not hard to see that $amb(G') \subseteq amb(G)$ holds. For the converse direction, assume that $t_1 \neq t_2$ are two parse trees for some w w.r.t. G. Let $\tilde{t}_1$ and $\tilde{t}_2$ result from them by replacing every node label A with $\tilde{A}$ and collapsing edges of the form $\tilde{A} \to \tilde{A}$. Note that these are parse trees for w w.r.t. G'. If $\tilde{t}_1 \neq \tilde{t}_2$ then $w \in amb(G')$. Otherwise, if $\tilde{t}_1 = \tilde{t}_2$ then either they coincide because of a collapsed edge in some t_i. In this case, $\tilde{t}_1$ must contain some $\tilde{A} \in M$ and therefore $w \in amb(G')$. Or there are nodes with labels A and B in t_1 and t_2 that get mapped to the same node $\tilde{A}$ in $\tilde{t}_1$, i.e. $A \sim B$ and therefore $\tilde{A} \in M$. □

We are now ready to describe the SAT encoding of bounded ambiguity. As above, we assume a macro $Two(\Phi)$ which, for an ordered set Φ of propositional formulas, is satisfiable iff there is an assignment satisfying at least two formulas out of Φ. It can easily be constructed by introducing at most $2 \cdot |\Phi| + 2$ new variables, c.f. the construction of *One* above.

Let G be an extended CFG in reduced, acyclic 2NF. The $\mathcal{W}$-constraints remain the same. Since "having two different parse trees" entails being derivable, we also add the $\mathcal{R}^{\to}$ constraints defined above. Finally, we simply have to state that there is a nonterminal which forms the root of the parse (sub)tree which is either an ambiguous nonterminal or to which two different productions apply.

$$\begin{aligned}\mathrm{bAMB}(k,k') \;:=\; & \mathcal{W}(k,k') \;\cup\; \mathcal{R}^{\to}(k,k') \;\cup \\ & \{\bigvee_{j=k}^{k'} \Big(\bigvee_{A \in M_G} X^A_{1,j} \;\vee \bigvee_{\substack{A \to BC \text{ or} \\ A \to CB \\ A \neq B, C \in E_G}} (X^A_{1,j} \wedge X^B_{1,j}) \;\vee \bigvee_{A \in N_G \setminus M_G} \big(X^A_{1,j} \wedge Two(\mathcal{P}_{A,j})\big) \Big)\}\end{aligned}$$

where

$$\begin{aligned}\mathcal{P}_{A,j} \;:=\; & \{\; X^B_{1,j} \;\mid\; A \to \alpha \in \{B, BC, CB\} \text{ with } C \Rightarrow^* \epsilon \;\} \\ & \cup \{\; X^a_1 \;\mid\; A \to a \text{ and } j = 1 \;\} \\ & \cup \{\; X^b_{1,h} \wedge X^C_{h+1,j} \;\mid\; A \to BC, 1 \leq h < j \;\}\end{aligned}$$

encodes all the different productions that can be made at the root labeled A of a parse tree for a word of length j. Again, let $\mathrm{bAMB}(k) := \mathrm{bAMB}(1,k)$. It is not difficult to see that the encoding of this problem supports incrementality as well. In each increment, the $\mathcal{W}$- and $\mathcal{R}^{\to}$-constraints remain, the other one has to be deleted, etc.

Lemma 6 together with an argument similar to that in the proof of Thm. 1 yields correctness of the encoding.

Theorem 3. *Let G be a CFG in 2NF, $k > 0$. Then* bAMB(k) *is satisfiable iff there are $u, v, z \in \Sigma^*$ s.t. $uvz \in L(G)$, $|v| \leq k$ and there are two different parse trees for w that differ on level 1 of the subtree for v.*

Proposition 4. *Let G be a CFG, $k > 0$. Then* bAMB(k) *can be equivalently translated into a propositional formula Φ in conjunctive normal form over $\mathcal{O}(|N_G| \cdot k^2)$ many variables s.t. $|\Phi| = \mathcal{O}(|G| \cdot k^3)$.*

4 Comparison

A prototype implementation of the reduction approach (`cfganalyzer`) has been implemented for all 5 bounded problems mentioned above. It is written in OCaml 3.09.3, uses zChaff version 2007.03.12 as a linked-in incremental SAT solver and is available online.[1]

Of the problems discussed here, ambiguity is the one to which most attention has been paid and for which a number of tools is available. These basically split up into three different approaches: (1) brute-force ambiguity detection, (2) LRR detection and (3) language approximation.

(1) Brute-force ambiguity detection systematically generates parse trees of a certain maximal size and looks for double appearances of the derived words. Ambiguous words which exceed the bound are not found – as in our approach. The crucial difference though is the use of a high-performance SAT solver as a back-end. While brute-force ambiguity detectors need to generate all parse trees for a certain bound one-by-one, our reduction covers all parse trees for that bound at once, and it is up to the SAT solver to find two in its solution space. In terms of complexity: we use a polynomial reduction to an NP-problem while (1) is an exponential reduction to a problem in P (finding equal strings in lists). The performance discrepancies between derivation generators and `cfganalyzer` can be seen by comparing Fig. 2 to the results of `AMBER` in [2]; `cfganalyzer` is more than 1000 times faster on subwords of the same size as words in e.g. the Pascal grammar and capable of pushing the bounds to $k = 25$ in reasonable time where `AMBER` is already at 100.000 sec for $k = 17$.

(2) LRR or LR-regularity is a generalisation of the well-known LR(k) grammar classes [5]. Instead of a k-symbol lookahead, an LRR parser considers regular equivalence classes on the remaining input and reports parsing conflicts. LRR detectors rely on the fact that every LRR-grammar is unambiguous and simply check a given grammar for this property. But since not every unambiguous grammar is LRR this method is of course also incomplete. Although being relatively fast, common LR(k) parsers such as `yacc` often reveal little about the causes of conflicts. Another positive effect of our approach is that it does always offer a detailed report on the cause of the ambiguity upon termination, i.e. provides two parsetrees for the ambiguous subword.

[1] `http://www.tcs.ifi.lmu.de/~mlange/cfganalyzer`,
We would also like to thank Harri Haanpää (TKK) and Anders Møller (Århus) for kindly providing us with benchmark CFGs.

(3) Methods of the third kind usually are complete but not sound by over-approximating the grammar. False-positives can occur because the language of the approximated grammar is a superset of the original one. Examples are the ACLA framework [3] or Schmitz's method [8]. Our approach does not easily compare to those since it is an under-approximation: it is sound, and complete only in the sense that it produces no spurious reports. It does however not terminate on unambiguous inputs. Hence, the situation is dual to that of the over-approximation approaches which can reliably report unambiguity. Because of this duality these two approaches combine well: a reported potential ambiguity of an over-approximation tool may be confirmed as a fact by `cfganalyzer` and a seemingly non-terminating run of it can be verified as unambiguous by such a tool.

To measure the performance of `cfganalyzer` it was run on 81 ambiguous grammars from bioinformatics, ambiguous variants of programming languages as well as on a larger number of toy examples from [2]. Note that unambiguous ones are meaningless benchmarks here. Crucial for the performance of the tool on ambiguous examples is of course the size of the grammar as it directly influences the bound k up to which witnesses are found before the SAT solver runs out of memory. Their number of rules varies between 3 and 862 (in 2NF). Most grammars have less than 200 rules, but among the 13 grammars with number of rules above 200, there are such prominent examples as C (413 rules), SML (304), Pascal (337), Elsa C++ (862) and SQL (202). Fig. 2 gives an overview of the performance on these in relation to the witness size k for ambiguous subwords. All ambiguities in the given grammars were confirmed by `cfganalyzer`.

We have also examined `cfganalyzer`'s performance on the bounded equivalence problem. It not only is the most difficult of the other problems but it also has obvious applications in CFG design whenever one grammar serves as a specification and another as an implementation, and one wants to ensure that they generate the same language. The following scenario provides a nice test suite. At Helsinki University of Technology, students are given veral descriptions of CFLs

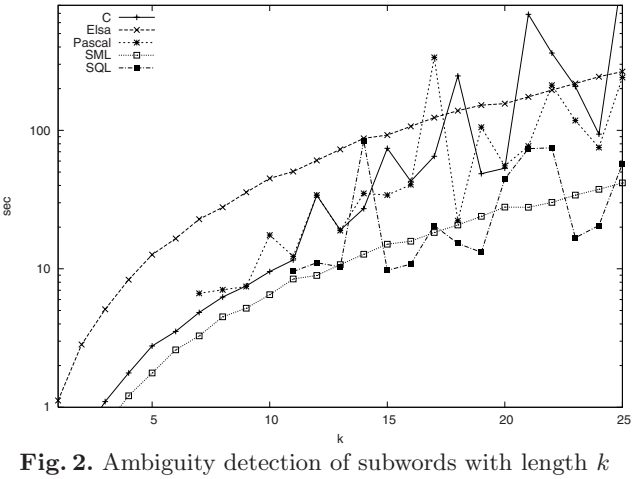

Fig. 2. Ambiguity detection of subwords with length k

and their task is to come up with CFGs which generate them. An automatized homework grading system has collected approx. 2000 student submissions for 40 different CFLs. Currently, unequivalence is only tested by sampling random words and checking that they are in both or neither of the two languages.

For running qualitatively better tests – `cfganalyzer` will not miss counterexamples up to the given bound unlike the testing approach – we have checked each of the 2000 grammars against all the sample solutions over the same alphabet (which of course makes the equivalence test fail for a large percentage). First, a coarse mapping of the submissions to the solution grammars was made, sorting out all submissions which were inequivalent to all solutions within a bound of $k \leq 15$ already (less than 0.1s in most tests). The remaining 251 grammars which potentially matched a solution were given a more thorough check by setting the maximum bound up to $k = 50$. Checking this range took on average 23.41s which is well below the time it would take to test all $|\Sigma|^{51} - 1$ words of length upto 50. This confirms `cfganalyzer`'s feasibility and usefulness in set-ups that have to deal with CFGs in an automatic fashion.

5 Conclusion

The previous section shows that undecidable problems of CFLs can be (under-) approximated by bounding the search space of witnesses / counterexamples and using an incremental SAT solver for finding them. This approach is sound and "complete upto termination": it does not yield false-positives but, while unambiguity for example cannot be proved but only insinuated by the lack of found witnesses. This complements other work on ambiguity detection, in particular over-approximations which are complete – they can prove unambiguity – but not sound. The prototype implementation `cfganalyzer` shows feasibility of this approach: it has found ambiguity of large real-world grammars in short time. It also shows that this approach by far outperforms other existing and comparable approaches, e.g. under-approximations like the brute-force enumeration of parse trees of bounded length.

References

1. Bar-Hillel, Y., Perles, M., Shamir, E.: On formal properties of simple phrase structure grammars. Zeitschrift für Phonologie, Sprachwissenschaft und Kommunikationsforschung 14, 113–124 (1961)
2. Basten, H.J.S.: The usability of ambiguity detection methods for context-free grammars. In: Johnstone, A., Vinju, J. (eds.) Eighth Workshop on Language Descriptions, Tools, and Applications (LDTA 2008), Budapest, Hungary (April 2008)
3. Brabrand, C., Giegerich, R., Møller, A.: Analyzing ambiguity of context-free grammars. In: Holub, J., Žďárek, J. (eds.) CIAA 2007. LNCS, vol. 4783, pp. 214–225. Springer, Heidelberg (2007)
4. Hopcroft, J., Ullman, J.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Reading (1979)

5. Culik II, K., Cohen, R.S.: LR-regular grammars - an extension of LR(k) grammars. Journal of Computer and System Sciences 7(1), 66–96 (1973)
6. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient sat solver. In: DAC, pp. 530–535. ACM, New York (2001)
7. Quimper, C.-G., Walsh, T.: Decomposing global grammar constraints. In: Bessière, C. (ed.) CP 2007. LNCS, vol. 4741, pp. 590–604. Springer, Heidelberg (2007)
8. Schmitz, S.: Conservative ambiguity detection in context-free grammars. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 692–703. Springer, Heidelberg (2007)
9. Younger, D.H.: Recognition and parsing of context-free languages in time n^3. Information and Control 10(2), 372–375 (1967)