Toward a Compositional Theory of Leftist Grammars and Transformations[⋆]

P. Chambart and Ph. Schnoebelen

LSV, ENS Cachan, CNRS
61, av. Pdt. Wilson, F-94230 Cachan, France

Abstract. Leftist grammars [Motwani *et al.*, STOC 2000] are special semi-Thue systems where symbols can only insert or erase to their left. We develop a theory of leftist grammars *seen as word transformers* as a tool toward rigorous analyses of their computational power. Our main contributions in this first paper are (1) constructions proving that leftist transformations are closed under compositions and transitive closures, and (2) a proof that bounded reachability is NP-complete even for leftist grammars with acyclic rules.

1 Introduction

Leftist grammars were introduced by Motwani *et al.* to study accessibility and safety in protection systems [7]. In this framework, leftist grammars are used to show that restricted accessibility grammars have decidable accessibility problems (unlike the more general access-matrix model).

Leftist grammars are both surprisingly simple and surprisingly complex. Simplicity comes from the fact that they only allow rules of the form "$a \to ba$" and "$cd \to d$" where a symbol inserts, resp. erases, another symbol to its left *while remaining unchanged*. But the combination of insertion and deletion rules makes leftist grammars go beyond context-sensitive grammars, and the decidability result comes with a high complexity-theoretical price [5]. Most of all, what is surprising is that apparently leftist grammars had not been identified as a relevant computational formalism until 2000.

The known facts on leftist grammars and their computational and expressive power are rather scarce. Motwani *et al.* show that it is decidable whether a given word can be derived (accessibility) and whether all derivable words belong to a given regular language (safety) [7]. Jurdziński and Loryś showed that leftist grammars can define languages that are not context-free [6] while leftist grammars restricted to acyclic rules are less expressive since they can only recognize regular languages. Then Jurdziński showed a PSPACE lower bound for accessibility in leftist grammars [4], before improving this to a nonprimitive-recursive lower bound [5].

Jurdziński's results rely on encoding classical computational structures (linear-bounded automata [4] and Ackermann's function [5]) in leftist grammars. Devising such encodings is difficult because leftist grammars are very hard to control. Thus, for computing Ackermann's function, devising the encoding is actually not the hardest part: the

⋆ Work supported by the Agence Nationale de la Recherche, grant ANR-06-SETIN-001.

harder task is to prove that the constructed leftist grammar cannot behave in unexpected ways. In this regard, the published proofs are necessarily incomplete, hard to follow, and hard to fully acknowledge. The final results and intermediary lemmas cannot easily be adapted or reused.

Our Contribution. We develop a compositional theory of leftist grammars and leftist transformations (i.e., operations on strings that are computed by leftist grammars) that provides fundamental tools for the analysis of their computational power. Our main contributions are effective constructions for the composition and the transitive closure of leftist transformations. The correctness proofs for these constructions are based on new definitions (e.g., for greedy derivations) and associated lemmas.

A first application of the compositional theory is given in Section 6 where we prove the NP-completeness of bounded reachability questions, even when restricted to acyclic leftist grammars.

A second application, and the main reason for this paper, is our forthcoming construction proving that leftist grammars can simulate lossy channel systems and "compute" all multiply-recursive transformations and nothing more (based on [3]), thus providing a precise measure of their computational power. Finally, after our introduction of Post's Embedding Problem [1, 2], leftist grammars are another basic computational model that will have been shown to capture exactly the notion of multiply-recursive computation.

As further comparison with earlier work, we observe that, of course, the complex constructions in [4, 5] are built modularly. However, the modularity is not made fully explicit in these works, the interfacing assumptions are incompletely stated, or are mixed with the details of the constructions, and correctness proofs cannot be given in full.

Outline of the Paper. Basic notations and definitions are recalled in Section 2. Section 3 defines leftist grammars and proves a generalized version of the completeness of greedy derivations. Sections 4 introduces leftist transformers and their sequential compositions. Section 5 specializes on the "simple" transformers that we use in Section 6 for our encoding of 3SAT. Finally Section 7 shows that so-called "anchored" transformers are closed under the transitive closure operation, this in an effective way. For lack of space, several proofs have been omitted in this extended abstract: they can be found in the long version of this paper, freely available at the arXiv.

2 Basic Definitions and Notations

Words. We use $x, y, u, v, w, \alpha, \beta, \ldots$ to denote words, i.e., finite strings of symbols taken from some alphabet. Concatenation is denoted multiplicatively with ε (the empty word) as neutral element, and the length of x is denoted $|x|$.

The congruence on words generated by the equivalences $a \approx aa$ (for all symbols a in the alphabet) is called the *stuttering equivalence* and is also denoted $\approx$: every word x has a minimal and canonical stuttering-equivalent x' obtained by repeatedly eliminating symbols in x that are adjacent to a copy of themselves.

We say that x is a *subword* of y, denoted $x \sqsubseteq y$, if x can be obtained by deleting some symbols (an arbitrary number, at arbitrary positions) from y. We further write $x \sqsubseteq_\Sigma y$

when all the symbols deleted from y belong to Σ (NB: we do not require $y \in \Sigma^*$), and let $\sqsupseteq$ denote the inverse relation $\sqsubseteq^{-1}$.

Relations and Relation Algebra. We see a relation R between two sets X and Y as a set of pairs, i.e., some $R \subseteq X \times Y$. We write $x\,R\,y$ rather than $(x,y) \in R$. Two relations R and R' can be composed, denoted multiplicatively with $R.R'$, and defined by $x\ (R.R')\ y \overset{\text{def}}{\Leftrightarrow} \exists z.\big(x\,R\,z \wedge z\,R'\,y\big)$.

The union $R + R'$, also denoted $R \cup R'$, is just the set-theoretic union. R^n is the n-th power $R.R \ldots R$ of R and R^{-1} is the inverse of R: $x\,R^{-1}\,y \overset{\text{def}}{\Leftrightarrow} y\,R\,x$. The transitive closure $\bigcup_{n=1,2,\ldots} R^n$ of R assumes $Y = X$ and is denoted R^+, while its reflexive-transitive closure is $R^+ \cup Id_X$, denoted R^*.

Below we often use notations from relation algebra to state simple equivalences. E.g., we write "$R = R'$" and "$R \subseteq S$" rather than "$x\,R\,y$ iff $x\,R'\,y$" and "$x\,R\,y$ implies $x\,S\,y$". Our proofs often rely on well-known basic laws from relation algebra, like $(R.R')^{-1} = R'^{-1}.R^{-1}$, or $(R + R').R'' = R.R'' + R'.R''$, without explicitly stating them.

3 Leftist Grammars

A *leftist grammar* (an LGr) is a triple $G = (\Sigma, P, g)$ where $\Sigma \cup \{g\} = \{a, b, \ldots\}$ is a finite *alphabet*, $g \notin \Sigma$ is a *final symbol* (also called "*axiom*"), and $P = \{r, \ldots\}$ is a set of production rules that may be *insertion rules* of the form $a \to ba$, and *deletion rules* of the form $cd \to d$. For simplicity, we forbid rules that insert or delete the axiom g (this is no loss of generality [6, Prop. 3]).

Leftist grammars are not context-free (deletions are contextual), or even context-sensitive (deletions are not length-preserving). For our purposes, we consider them as string rewrite systems, more precisely semi-Thue systems. Writing Σ_g for $\Sigma \cup \{g\}$, the rules of P define a 1-step rewrite relation in the standard way: for $u, u' \in \Sigma_g^*$, we write $u \Rightarrow^{r,p} u'$ whenever r is some rule $\alpha \to \beta$, u is some $u_1 \alpha u_2$ with $|u_1 \alpha| = p$ and $u' = u_1 \beta u_2$. We often write shortly $u \Rightarrow^r u'$, or even $u \Rightarrow u'$, when the position or the rule involved in the step can be left implicit. On the other hand, we sometimes use a subscript, e.g., writing $u \Rightarrow_G v$, when the underlying grammar has to be made explicit.

A *derivation* is a sequence π of consecutive rewrite steps, i.e., is some $u_0 \Rightarrow^{r_1,p_1} u_1 \Rightarrow^{r_2,p_2} u_2 \cdots \Rightarrow^{r_n,p_n} u_n$, often abbreviated as $u_0 \Rightarrow^n u_n$, or even $u_0 \Rightarrow^* u_n$. A subsequence $(u_{i-1} \Rightarrow^{r_i,p_i} u_i)_{i=m,m+1,\ldots,l}$ of π is a *subderivation*. As with all semi-Thue systems, steps (and derivations) are closed under adjunction: if $u \Rightarrow u'$ then $vuw \Rightarrow vu'w$.

Two derivations $\pi_1 = (u \Rightarrow^* u')$ and $\pi_2 = (v \Rightarrow^* v')$ can be concatenated in the obvious way (denoted $\pi_1.\pi_2$) if $u' = v$. They are *equivalent*, denoted $\pi_1 \equiv \pi_2$, if they have same extremities, i.e., if $u = v$ and $u' = v'$.

We say that $u \in \Sigma^*$ is *accepted by G* if there is a derivation of the form $ug \Rightarrow^* g$ and we write $L(G)$ for the set of accepted words, i.e., the language recognized by G.

We say that $I \subseteq \Sigma^*$ is an *invariant* for an LGr $G = (\Sigma, P, g)$ if $u \in I$ and $ug \Rightarrow vg$ entail $v \in I$. Knowing that I is an invariant for G is used in two symmetric ways: (1) from $u \in I$ and $ug \Rightarrow^* vg$ one deduces $v \in I$, and (2) from $ug \Rightarrow^* vg$ and $v \notin I$ one deduces $u \notin I$.

3.1 Graphs and Types for Leftist Grammars

When dealing with LGr's, it is convenient to write insertion rules under the simpler form "$a \rightarrow b$", and deletion rules as "$d \dashrightarrow c$", emphasizing the fact that a (resp. d) is not modified during the insertion of b (resp. the deletion of c) on its left. For $a \in \Sigma_g$, we let $\mathbf{ins}(a) \overset{\text{def}}{=} \{b \mid P \ni (a \rightarrow b)\}$ and $\mathbf{del}(a) \overset{\text{def}}{=} \{b \mid P \ni (a \dashrightarrow b)\}$ denote the set of symbols that can be inserted (respectively, deleted) by a. We write $\mathbf{ins}^+(a)$ for the smallest set that contains b and $\mathbf{ins}^+(b)$ for all $b \in \mathbf{ins}(a)$, while $\mathbf{del}^+(b)$ is defined similarly. We say that a is *inactive* in a LGr if $\mathbf{del}(a) \cup \mathbf{ins}(a) = \varnothing$.

It is often convenient to view LGr's in a graph-theoretical way. Formally, the *graph* of $G = (\Sigma, P, g)$ is the directed graph τ_G having the symbols from Σ_g as vertices and the rules from P as edges (coming in two kinds, insertions and deletions). Furthermore, we often decorate such graphs with extra bookkeeping annotations.

We say that G "*has type* τ" when τ_G is a sub-graph of τ. Thus a "type" is just a restriction on what are the allowed symbols and rules between them. Types are often given schematically, grouping symbols that play a similar role into a single vertex. For

insertion:
deletion:
Σ
g

Fig. 1. Universal type (schematically).

example, Fig. 1 displays schematically the type (parametrized by the alphabet) observed by all LGr's.

3.2 Leftmost, Pure and Eager Derivations

We speak informally of a "letter", say a, when we really mean "an occurrence of the symbol a" (in some word). Furthermore, we follow letters along steps $u \Rightarrow v$, identifying the letters in u and the corresponding letters in v. Hence a "letter" is also a sequence of occurrences in consecutive words along a derivation.

A letter a is a n-th *descendant* of another letter b (in the context of a derivation) if a has been inserted by b (when $n = 1$), or by a $(n-1)$-th descendant of b.

Given a step $u \Rightarrow^{r,p} v$, we say that the p-th letter in u, written $u[p]$, is the *active letter*: the one that inserts, or deletes, a letter to its left. This is often emphasized by writing the step under the form $(u =)u_1au_2 \Rightarrow u'_1au_2 (= v)$ (assuming $u[p] = a$).

A letter is *inert* in a derivation if it is not active *in any step* of the derivation. A set of letters is inert if it only contains inert letters. A derivation is *leftmost* if every step $u_1au_2 \Rightarrow u'_1au_2$ in the derivation is such that u_1 is inert in the rest of the derivation.

A letter is *useful* in a derivation $\pi = (u \Rightarrow^* v)$ if it belongs to u or v, or if it inserts or deletes a useful letter along π. This recursive definition is well-founded: since letters only insert or delete to their left, the "inserts-or-deletes" relation between letters is acyclic. A derivation π is *pure* if all letters in π are useful. Observe that if π is not pure, it necessarily inserts at some step some letter a (called a *useless letter*) that stays inert and will eventually be deleted.

A derivation is *eager* if, informally, deletions occur as soon as possible. Formally, $\pi = (u_0 \Rightarrow^{r_1,p_1} u_1 \cdots \Rightarrow^{r_n,p_n} u_n)$ is not eager if there is some u_{i-1} of the form $w_1 baw_2$ where b is inert in the rest of π and is eventually deleted, where P contains the rule $a \dashrightarrow b$, and where r_i is not a deletion rule.[1]

A derivation is *greedy* if it is leftmost, pure and eager. Our definition generalizes [4, Def. 4], most notably because it also applies to derivations $ug \Rightarrow^* vg$ with nonempty v. Hence a subderivation π' of π is leftmost, eager, pure, or greedy, when π is.

The following proposition generalizes [4, Lemma 7].

Proposition 3.1 (Greedy derivations are sufficient). *Every derivation π has an equivalent greedy derivation π'.*

Proof. With a derivation π of the form $u_0 \Rightarrow^{r_1,p_1} u_1 \Rightarrow^{r_2,p_2} u_2 \cdots \Rightarrow^{r_n,p_n} u_n$, we associate its *measure* $\mu(\pi) \overset{\text{def}}{=} \langle n, p_1, \ldots, p_n \rangle$, a $(n+1)$-tuple of numbers. Measures are linearly ordered with the lexicographic ordering, giving rise to a quasi-ordering, denoted $\leq_\mu$, between derivations. A derivation is called *μ-minimal* if any equivalent derivation has greater or equal measure.

We can now prove Prop. 3.1 along the following lines: first prove that every derivation has a μ-minimal equivalent, then show that μ-minimal derivations are greedy. $\square$

Observe that $\leq_\mu$ is compatible with concatenation of derivations: if $\pi_1 \leq_\mu \pi_2$ then $\pi.\pi_1.\pi' \leq_\mu \pi.\pi_2.\pi'$ when these concatenations are defined. Thus any subderivation of a μ-minimal derivation is μ-minimal, hence also greedy.

μ-minimality is stronger than greediness, and is a powerful and convenient tool for proving Prop. 3.1. However, greediness is easier to reason with since it only involves local properties of derivations, while μ-minimality is "global". These intuitions are reflected by, and explain, the following complexity results.

Theorem 3.2. *1.* Greediness *(deciding whether a given derivation π in the context of a given LGr G is greedy) is in* L.
2. μ-Minimality *(deciding whether it is μ-minimal) is* coNP*-complete, even if we restrict to acyclic LGr's.*

Proof. 1. Being leftmost or eager is easily checked in logspace (i.e., is in L). Checking non-purity can be done by looking for a *last* inserted useless letter, hence is in L too.
2. μ-minimality is obviously in coNP. Hardness is proved as Coro. 6.9 below, as a byproduct of the reduction we use for the NP-hardness of Bounded Reachability. $\square$

4 Leftist Grammars as Transformers

Some leftist grammars are used as computing devices rather than recognizers of words. For this purpose, we require a strict separation between input and output symbols and speak of *leftist transformers*, or shortly LTr's.

[1] Eagerness does not require that r_i deletes b: other deletions are allowed, only insertions are forbidden.

4.1 Leftist Transformers

Formally, an LTr is a LGr $G = (\Sigma, P, g)$ where Σ is partitioned as $A \uplus B \uplus C$, and where symbols from A are inactive in P and are not inserted by P (see Fig. 2). This is denoted $G : A \vdash C$. Here A contains the *input symbols*, B the *temporary symbols*, and C the *output symbols*, and G is more conveniently written as $G = (A, B, C, P, g)$. When there is no need to distinguish between temporary and output symbols, we write G under the form $G = (A, D, P, g)$, where $D \overset{\text{def}}{=} B \cup C$ contains the *"working" symbols*,

$$\begin{tikzcd} A & D \arrow[l, dotted] \arrow[loop above] \arrow[loop below, dotted] & g \arrow[l] \arrow[l, dotted, shift left] \end{tikzcd}$$

Fig. 2. Type of leftist transformers.

A consequence of the restrictions imposed on LTr's is the following:

Fact 4.1 *A^*D^* is an invariant in any LTr $G = (A, D, P, g)$.*

With $G = (A, B, C, P, g)$, we associate a *transformation* (a relation between words) $R_G \subseteq A^* \times C^*$ defined by

$$u\, R_G\, v \overset{\text{def}}{\Leftrightarrow} ug \Rightarrow^*_G vg \wedge u \in A^* \wedge v \in C^*$$

and we say that *G realizes R_G*. Finally, a *leftist transformation* is any relation on words realized by some LTr. By necessity, a leftist transformation can only relate words written using disjoint alphabets (this is not contradicted by $\varepsilon\, R_G\, \varepsilon$).

Leftist transformations respect some structural constraints. In this paper we shall use the following properties:

Proposition 4.2 (Closure for leftist transformations). *If $G : A \vdash C$ is a leftist transformer, then $R_G = (\sqsupseteq_A . \approx . R_G . \approx)$.*

4.2 Composition

We say that two leftist transformations $R_1 \subseteq A_1^* \times C_1^*$ and $R_2 \subseteq A_2^* \times C_2^*$ are *chainable* if $C_1 = A_2$ and $A_1 \cap C_2 = \varnothing$. Two LTr's are chainable if they realize chainable transformations.

Theorem 4.3. *The composition $R_1.R_2$ of two chainable leftist transformations is a leftist transformation. Furthermore, one can build effectively a linear-sized LTr realizing $R_1.R_2$ from LTr's realizing R_1 and R_2.*

For a proof, assume $G_1 = (A_1, B_1, C_1, P_1, g)$ and $G_2 = (A_2, B_2, C_2, P_2, g)$ realize R_1 and R_2. Beyond chainability, we assume that $A_1 \cup B_1$ and $B_2 \cup C_2$ are disjoint, which can be ensured by renaming the intermediary symbols in B_1 and B_2. The composed LTr $G_1.G_2$ is given by

$$G_1.G_2 \overset{\text{def}}{=} (A_1, B_1 \cup C_1 \cup B_2, C_2, P_1 \cup P_2, g).$$

$$\begin{tikzcd}
& & g \arrow[dl, "P_1"'] \arrow[dl, dotted] \arrow[dr, "P_2"] \arrow[dr, dotted] & \\
A_1 & D_1 (\supseteq A_2) \arrow[l, dotted, "P_1"] \arrow[loop above, "P_1"] \arrow[loop below, dotted, "P_1"] & & D_2 \arrow[ll, dotted, "P_2"] \arrow[loop right, "P_2"] \arrow[loop below, dotted, "P_2"]
\end{tikzcd}$$

Fig. 3. The type of $G_1.G_2$.

$$\begin{tikzcd}
\Sigma \arrow[loop above] & g \arrow[l]
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
A & C \arrow[l, dotted] \arrow[loop above] & g \arrow[l]
\end{tikzcd}$$

Fig. 4. Types of insertion grammars (left) and simple leftist transformers (right).

This is indeed a LTr from A_1 to C_2. See Fig. 3 for a schematics of its type. Since $G_1.G_2$ has all rules from G_1 and G_2 it is clear that $(\Rightarrow_{G_1} + \Rightarrow_{G_2}) \subseteq \Rightarrow_G$, from which we deduce $R_{G_1}.R_{G_2} \subseteq R_{G_1.G_2}$. Furthermore, the inclusion in the other direction also holds:

Lemma 4.4 (Composition Lemma). $R_{G_1.G_2} = R_{G_1}.R_{G_2}$.

Remark 4.5 (Associativity). The composition $(G_1.G_2).G_3$ is well-defined if and only if $G_1.(G_2.G_3)$ is. Furthermore, the two expressions denote exactly the same result. $\square$

5 Simple Leftist Transformations

As a tool for Sections 6 and 7, we now introduce and study restricted families of leftist grammars (and transformers) where deletion rules are forbidden (resp., only allowed on A).

An *insertion grammar* is a LGr $G = (\Sigma, P, g)$ where P only contain insertion rules. See Fig. 4 for a graphic definition. For an arbitrary leftist grammar G, we denote with $G^{\mathbf{ins}}$ the insertion grammar obtained from G by keeping only the insertion rules.

The *insertion relation* $I_G \subseteq \Sigma^* \times \Sigma^*$ associated with an insertion grammar $G = (\Sigma, P, g)$ is defined by $u\ I_G\ v \overset{\text{def}}{\Leftrightarrow} ug \Rightarrow_G^* vg$. Obviously, $I_G \subseteq \sqsubseteq_\Sigma$. Observe that I_G is not necessarily a leftist transformation since it does not require any separation between input and output symbols.

A *simple* leftist transformer is an LTr $G = (A, B, C, P, g)$ where $B = \varnothing$ and where no rule in P erases symbols from C. See Fig. 4 for a graphic definition. We give, without proof, an immediate consequence of the definition:

Lemma 5.1. *Let $G = (A, \varnothing, C, P, g)$ be a simple LTr and assume $ug \Rightarrow_G^k vg$ for some $u \in A^*$ and $v \in C^*$. Then $k = |u| + |v|$.*

Given a simple LTr $G = (A, \varnothing, C, P, g)$ and two words $u = a_1 \cdots a_n \in A^*$ and $v = c_1 \cdots c_m \in C^*$, we say that a non-decreasing map $h : \{1, \ldots, n\} \to \{1, \ldots, m\}$ is a G-*witness* for u and v if P contains the rules $c_{h(i)} \dashrightarrow a_i$ and $c_{j+1} \to c_j$ (for all $i = 1, \ldots, n$ and $j = 1, \ldots, m$, with the convention that $c_{m+1} = g$). Finally, we write $u\ \nabla_G\ v$ when such a G-witness exists. Clearly, $\nabla_G \subseteq R_G$. Indeed, when G is a simple transformer, ∇_G can be used as a restricted version of R_G that is easier to control and reason about.

Lemma 5.2. *Let $G = (A, \varnothing, C, P, g)$ be a simple LTr. Then $R_G = \nabla_G.I_{G^{\mathbf{ins}}}$.*

Combining Lemma 5.2 with $Id_{C^*} \subseteq I_{G^{\mathbf{ins}}} \subseteq \sqsubseteq_C$, we obtain the following weaker but simpler statement.

Corollary 5.3. *Let $G = (A, \varnothing, C, P, g)$ be a simple LTr. Then $\nabla_G \subseteq R_G \subseteq \nabla_G. \sqsubseteq_C$.*

5.1 Union of Simple Leftist Transformers

We now consider the combination of two simple LTr's $G_1 = (A, \varnothing, C_1, P_1, g)$ and $G_2 = (A, \varnothing, C_2, P_2, g)$ that transform from a same A to disjoint output alphabets, i.e., with $C_1 \cap C_2 = \varnothing$. We define their *union* with $G_1 + G_2 \overset{\text{def}}{=} (A, \varnothing, C_1 \cup C_2, P_1 \cup P_2, g)$. This is clearly a simple LTr with $(R_{G_1} + R_{G_2}) \subseteq R_{G_1+G_2}$. It further satisfies:

Lemma 5.4. *If $u\ R_{G_1+G_2}\ v$ then $u\ (R_{G_1} + R_{G_2})\ v'$ for some $v' \sqsubseteq v$.*

Proof. Assume $u\ R_{G_1+G_2}\ v$. With Cor. 5.3, we obtain $u\ \nabla_{G_1+G_2}\ v'$ for some $v' = c_1 \cdots c_m \sqsubseteq v$. Hence $G_1 + G_2$ has insertion rules $c_{j+1} \longrightarrow c_j$ for all $j = 1, \ldots, m$, and deletion rules of the form $c_{h(i)} \dashrightarrow u[i]$. Since C_1 and C_2 are disjoint, either all these rules are in G_1 (and $u\ \nabla_{G_1}\ v'$), or they are all in G_2 (and $u\ \nabla_{G_2}\ v'$). Hence $u\ (R_{G_1} + R_{G_2})\ v'$. $\square$

6 Encoding 3SAT with Acyclic Leftist Transformers

This section proves the following result.

Theorem 6.1. Bounded Reachability *and* Exact Bounded Reachability *in leftist grammars are* NP*-complete, even when restricting to acyclic grammars.*

(Exact) Bounded Reachability is the question whether there exists a n-step derivation $u \Rightarrow^n v$ (respectively, a derivation $u \Rightarrow^{\leq n} v$ of non-exact length at most n) between given u and v. These questions are among the simplest reachability questions and, since we consider that the input n is given in unary,[2] they are obviously in NP for leftist grammars (and all semi-Thue systems).

Consequently, our contribution in this paper is the NP-hardness part. This is proved by encoding 3SAT instances in leftist grammars where reaching a given final v amounts to guessing a valuation that satisfies the formula. While the idea of the reduction is easy to grasp, the technicalities involved are heavy and it would be difficult to really prove the correctnessof the reduction without relying on a compositional framework like the one we develop in this paper. It is indeed very tempting to “prove” it by just running an example.

Rather than adopting this easy way, we shall describe the reduction as a composition of simple leftist transformers and use our composition theorems to break down the

[2] It is natural to begin with this assumption when considering fundamental aspects of reachability since writing n more succinctly would blur the complexity-theoretical picture.

correctness proof in smaller, manageable parts. Once the ideas underlying the reduction are grasped, a good deal of the reasoning is of the type-checking kind: verifying that the conditions required for composing transformers are met.

Throughout this section we assume a generic 3SAT instance $\Phi = \bigwedge_{i=1}^{m} C_i$ with m 3-clauses on n Boolean variables in $X = \{x_1, \ldots, x_n\}$. Each clause has the form $C_i = \bigvee_{k=1}^{3} \varepsilon_{i,k} x_{i,k}$ for some polarity $\varepsilon_{i,k} \in \{+,-\}$ and $x_{i,k} \in X$. (There are two additional assumptions on Φ that we postpone until the proof of Coro. 6.5 for clarity.) We use standard model-theoretical notation like $\models \Phi$ (validity), or $\sigma \models \Phi$ (entailment) when σ is a Boolean formula or a Boolean valuation of some variables.

We write $\sigma[x \mapsto b]$ for the extension of a valuation σ with (x,b), assuming $x \notin Dom(\sigma)$. Finally, for a valuation $\theta : X \to \{\top, \bot\}$ and some $j = 0, \ldots, n$, we write θ_j to denote the restriction $\theta_{|\{x_1,\ldots,x_j\}}$ of θ on the first j variables.

6.1 Associating an LTr G_Φ with Φ

For the encoding, we use an alphabet $\Sigma = \{T_i^j, U_i^j, T'^j_i, U'^j_i \mid i = 1, \ldots, m \wedge j = 0, \ldots, n\}$, i.e., $4(n+1)$ symbols for each clause. The choice of the symbols is that a U means "*Undetermined*" and a T means "*True*", or determined to be valid.

For $j = 0, \ldots, n$, let $V_j \stackrel{\text{def}}{=} \{U_1^j, \ldots, U_m^j, T_1^j, \ldots, T_m^j\}$, $V'_j \stackrel{\text{def}}{=} \{U'^j_1, \ldots, U'^j_m, T'^j_1, \ldots, T'^j_m\}$, and $W_j \stackrel{\text{def}}{=} V_j \cup V'_j$, so that Σ is partitioned in levels with $\Sigma = \bigcup_{j=0}^{n} W_j$. With each $x_j \in X$ we associate two intermediary LTr's:

$$G_j^\top \stackrel{\text{def}}{=} (W_{j-1}, \varnothing, V_j, P_j, g), \qquad\qquad G_j^\bot \stackrel{\text{def}}{=} (W_{j-1}, \varnothing, V'_j, P'_j, g)$$

with sets of rules P_j and P'_j. The rules for $G_j^\top$ are given in Fig. 5: some deletion rules are conditional, depending on whether x_j appears in the clauses $C_1, \ldots, C_m$. The rules for $G_j^\bot$ are obtained by switching primed and unprimed symbols, and by having conditional rules based on whether $\neg x_j$ appears in the C_i's. One easily checks that $G_j^\top$ and $G_j^\bot$

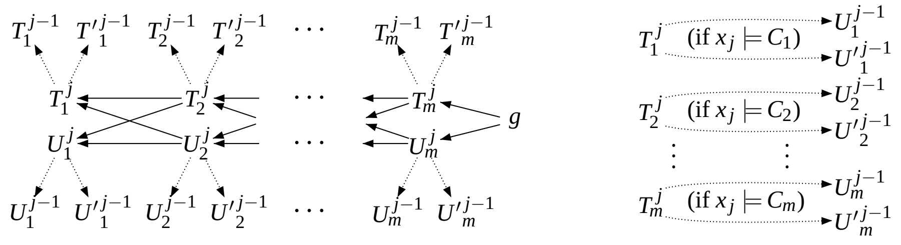

Fig. 5. P_j, the rules for $G_j^\top$: Fixed part on left, conditional part on right.

are indeed simple transformers. They have same inputs and disjoint outputs so that the union $(G_j^\top + G_j^\bot) : W_{j-1} \vdash W_j$ is well-defined. Hence the following composition is well-formed:

$$G_\Phi \stackrel{\text{def}}{=} (G_1^\top + G_1^\bot).(G_2^\top + G_2^\bot) \cdots (G_n^\top + G_n^\bot).$$

We conclude the definition of G_Φ with an intuitive explanation of the idea behind the reduction. G_Φ operates on the word $u_0 = U_1^0 \cdots U_m^0$ where each U_i^0 stands for "*the validity* *of clause C_i is undetermined at step* 0 *(i.e., at the beginning)*". At step j, $G_j^\top + G_j^\perp$ picks a valuation for x_j: $G_j^\top$ picks "$x_j = \top$" while $G_j^\perp$ picks "$x_j = \perp$". This transforms U_i^{j-1} into U_i^j, and T_i^{j-1} into T_i^j, moving them to the next level. Furthermore, an undetermined U_i^{j-1} can be transformed into T_i^j if C_i is satisfied by x_j. In addition, and because $G_j^\top$ and $G_j^\perp$ must have disjoint output alphabets, the symbols in the V_j's come in two copies (hence the V_j''s) that behave identically when they are input in the transformer for the next step.

The reduction is concluded with the following claim that we prove by combining Corollaries 6.5 and 6.8 below.

$$\begin{aligned}\Phi \text{ is satisfiable iff } & U_1^0 U_2^0 \cdots U_m^0 g \Rightarrow_{G_\Phi}^{2mn} T_1^n T_2^n \cdots T_m^n g \\ \text{iff } & U_1^0 U_2^0 \cdots U_m^0 g \Rightarrow_{G_\Phi}^{\leq 2mn} T_1^n T_2^n \cdots T_m^n g \qquad \text{(Correctness)} \\ \text{iff } & U_1^0 U_2^0 \cdots U_m^0 g \Rightarrow_{G_\Phi}^{*} T_1^n T_2^n \cdots T_m^n g.\end{aligned}$$

Observe finally that G_Φ is an acyclic grammar in the sense of [6], that is to say, its rules define an acyclic "*may-act-upon*" relation between symbols. Such grammars are much weaker than general LGr's since, e.g., languages recognized by LGr's with acyclic deletion rules (and arbitrary insertion rules) are regular [6].

Remark 6.2. The construction of G_Φ from Φ, mostly amounting to copying operations for the $G_j^\top$'s and $G_j^\perp$'s, to type-checking and sets-joining operations for the composition of the LTr's, can be carried out in logarithmic space. $\square$

6.2 Correctness of the Reduction

We say that a word u is *j-clean* if it has exactly m symbols and if $u[i] \in \{T_i^j, T'^j_i, U_i^j, U'^j_i\}$ for all $i = 1,\ldots,m$. It is *$\top$-homogeneous* (resp. *$\perp$-homogeneous*) if it does not contain any (resp., only contains) primed symbols.

Let $0 \leq j \leq n$ and θ_j be a Boolean valuation of $x_1,\ldots,x_j$: we say that a j-clean u *respects* (Φ under) θ_j when, for all $i = 1,\ldots,m$, $\theta_j \models C_i$ when $u[i]$ is determined (i.e., $\in T_i^j + T'^j_i$). Finally u *codes* (Φ under) θ_j if additionally each $u[i]$ is determined when $\theta_j \models C_i$. Thus, a word u that codes some θ_j exactly lists (via determined symbols) the clauses of Φ made valid by θ_j, and the only flexibility in u is in using the primed or the unprimed copy of the symbols. Hence there is only one j-clean u coding θ_j that is $\top$-homogeneous, and only one that is $\perp$-homogeneous. If u respects θ_j instead of coding it, more latitude exists since symbols may be undetermined even if the corresponding clause is valid under θ_j.

Assume that, for some $j \in \{1,\ldots,n\}$, u_{j-1} codes θ_{j-1} and u_j codes θ_j. Write b for $\theta(x_j)$ (NB: $b \in \{\top,\perp\}$).

Lemma 6.3. *If u_j is b-homogeneous then $u_{j-1} \nabla_{G_j^b} u_j$.*

Proof. Let $h \overset{\text{def}}{=} Id_{\{1,\ldots,m\}}$. We claim that h is a G_j^b-witness for u_{j-1} and u_j, i.e., that G_j^b contains the required insertion and deletion rules.

Insertions. G^b_j has all insertion rules $g \to u_j[m] \to u_j[m-1] \to \ldots \to u_j[1]$ (leftmost rules in Fig. 5) since u_j is b-homogeneous.
Deletions. G^b_j has all deletion rules $u_j[i] \dashrightarrow u_{j-1}[i]$. Firstly, both undetermined symbols U^i_j and U'^i_j may delete their counterparts U^i_{j-1} and U'^i_{j-1}, and similarly for the determined symbols (the unconditional deletion rules in Fig. 5). This is used if C_i is not more valid under θ_j than under θ_{j-1}. Secondly, if C_i is valid under θ_j but not under θ_{j-1}, then $x_j \models C_i$ (or $\neg x_j \models C_i$, depending on b) and the conditional rules in Fig. 5 allow a determined T^j_i (or T'^j_i depending on b) to delete U^{j-1}_i or U'^{j-1}_i. ☐

Lemma 6.4. *If u_j is b-homogeneous, then $u_{j-1}g \Rightarrow^{2m}_{G^b_j} u_jg$.*

Proof. From $u_{j-1} \nabla_{G^b_j} u_j$ (Lemma 6.3) we deduce $u_{j-1} R_{G^b_j} u_j$, i.e., $u_{j-1}g \Rightarrow^*_{G^b_j} u_jg$, by Lemma 5.2, and then $u_{j-1}g \Rightarrow^{2m}_{G^b_j} u_jg$ by Lemma 5.1. ☐

Corollary 6.5. *If Φ is satisfiable, then $U^0_1 \cdots U^0_m g \Rightarrow^{2mn}_{G_\Phi} T^n_1 \cdots T^n_m g$.*

Proof. Since Φ is satisfiable, $\theta \models \Phi$ for some valuation θ. For $j = 1,\ldots,m$, we write b_j for $\theta(x_j)$ and let u_j be the only j-clean b_j-homogeneous word that codes for θ_j.

We now make two assumptions on Φ that are no loss of generality. First we require that no clause C_i contains both a literal and its negation, hence no C_i is tautologically valid. Then $u_0 \overset{\text{def}}{=} U^0_1 \cdots U^0_m$ codes the empty valuation θ_0. Second, we require that Φ is only satisfiable with $b_n = \top$ (which can be easily ensured by adding a few extra variables). Then necessarily $u_n = T^n_1 \cdots T^n_m$.

Lemma 6.4 gives $u_0g \Rightarrow^{2m}_{G^{b_1}_1} u_1g \Rightarrow^{2m}_{G^{b_2}_2} u_2g \cdots \Rightarrow^{2m}_{G^{b_n}_n} u_ng$. Since $\Rightarrow_{G^b_j} \subseteq \Rightarrow_{G_j} \subseteq \Rightarrow_{G_\Phi}$ for all b and j, we deduce $u_0g \Rightarrow^{2mn}_{G_\Phi} u_ng$ as claimed. ☐

Fix some θ, some $j \in \{1,\ldots,n\}$ and let $b = \theta(x_j)$.

Lemma 6.6. *If u respects θ_{j-1} and $u \nabla_{G^b_j} v$, then v respects θ_j.*

Proof. Write l for $|v|$. From $u \nabla_{G^b_j} v$ (witnessed by some h) we deduce that G^b_j has insertion rules $g \to v[l] \to v[l-1] \to \ldots \to v[1]$. Inspecting Fig. 5, we conclude that necessarily $l \le m$. Since deletion rules $v[h(i)] \dashrightarrow u[i]$ are required for all $i = 1,\ldots,m$, we further see from Fig. 5 that h is injective, so that $l \ge m$. Finally $l = m$, $h = Id_{\{1,\ldots,m\}}$, v is j-clean and b-homogeneous.

Now, knowing that G^b_j contains the rules $v[i] \dashrightarrow u[i]$, we show that v respects θ_j. Suppose, by way of contradiction, that it does not. Thus there is some $i \in \{1,\ldots,m\}$ with $v[i] = T^j_i$ (assuming $b = \top$ w.l.o.g.) while $\theta_j \not\models C_i$ (so that $\theta_{j-1} \not\models C_i$). From $\theta_j \not\models C_i$ we deduce that $x_j \not\models C_i$. Hence G^b_j does not have the conditional rules $T^j_i \dashrightarrow U^{j-1}_i$ and $T^j_i \dashrightarrow U'^{j-1}_i$. Thus $u[i] \notin \{U^{j-1}_i, U'^{j-1}_i\}$. But then u does not respect θ_{j-1}, contradicting our assumption. ☐

We immediately deduce:

Lemma 6.7. *If $x\ R_{G_j^b}\ y$ and there is some $u \sqsubseteq x$ that respects θ_{j-1}, then there is some $v \sqsubseteq y$ that respects θ_j.*

Proof. From the Closure Property 4.2, we get $u\ R_{G_j^b}\ y$. Then, from $R_{G_j^b} \subseteq \nabla_{G_j^b}.\sqsubseteq$ (Coro. 5.3) we deduce $u\ \nabla_{G_j^b}\ v$ for some $v \sqsubseteq y$. Now v respects θ_j thanks to Lemma 6.6. □

Corollary 6.8. *If $U_1^0 \cdots U_m^0 g \Rightarrow^*_{G_\Phi} T_1^n \cdots T_m^n g$, then Φ is satisfiable.*

Proof. Write u_0 for $U_1^0 \cdots U_m^0$ and u_n for $T_1^n \cdots T_m^n$. From the definition of G_Φ and the Composition Lemma 4.4, we deduce that there exist some words $u_1, \ldots, u_{n-1}$ such that $u_{j-1}\ R_{G_j^\top + G_j^\perp}\ u_j$ for all $j = 1, \ldots, n$.

With Lemma 5.4, we further deduce that there exist some words $u_1', \ldots, u_n'$ and Boolean values $b_1, \ldots, b_n$ such that $u_j' \sqsubseteq u_j$ and $u_{j-1}\ R_{G_j^{b_j}}\ u_j'$ for all $j = 1, \ldots, n$. Hence also $u_{j-1}'\ R_{G_j^{b_j}}\ u_j'$ by Prop. 4.2 (and letting $u_0' = u_0$).

Write θ for $[x_1 \mapsto b_1, \ldots, x_n \mapsto b_n]$. With Lemma 6.7, induction on j, and since u_0' respects θ_0, we further deduce that there exists some words $u_1'', \ldots, u_n''$ such that, for all $j = 1, \ldots, n$, $u_j'' \sqsubseteq u_j'$ and u_j'' respects θ_j. From $|u_n''| = m$ (it respects θ) and $u_n'' \sqsubseteq u_n$, we deduce that $u_n'' = u_n$. Finally, $\theta \models \Phi$ since u_n'' respects θ and $u_n'' = u_n = T_1^n \cdots T_m^n$. □

Corollary 6.9. *μ-Minimality of a derivation is* coNP*-hard.*

Proof (Sketch). We define G_Φ' by taking G_Φ, adding k extra symbols $a_1, \ldots, a_k$, and adding the following two sets of rules:
(1) all $a_{i-1} \rightarrow a_i$ and $a_{i-1} \dashrightarrow a_i$ for $i = 1, \ldots, k$ (with the convention that a_0 is T_1^n);
(2) all $a_k \dashrightarrow U_i^0$ for $i = 1, \ldots, m$.

Observe that G_Φ' is acyclic. It has a derivation $\pi : U_1^0 \cdots U_m^0 g \Rightarrow^{2m+2k} T_1^n \cdots T_m^n g$ of the following form:

$$\begin{aligned} U_1^0 \cdots U_m^0 g \Rightarrow^m U_1^0 \cdots U_m^0 T_1^n \cdots T_m^n g \Rightarrow^k U_1^0 \cdots U_m^0 a_k a_{k-1} \cdots a_1 T_1^n \cdots T_m^n g \\ \Rightarrow^m a_k a_{k-1} \cdots a_1 T_1^n \cdots T_m^n g \Rightarrow^k T_1^n \cdots T_m^n g. \end{aligned}$$

This derivation uses the extra symbols to bypass the normal behaviour of G_ϕ. If k is large enough, i.e., $k > m(n-1)$, π is μ-minimal if, and only if, Φ is not satisfiable. □

7 Anchored Leftist Transformers and Their Transitive Closure

When $b_1, b_2 \in B$ are two different working symbols, and (A, B, C, P, g) is a LTr, we call $G = (A, B, C, b_1, b_2, P, g)$ an *anchored* LTr, or shorly an ALTr. With an ALTr G we associate an *anchored tranformation* $S_G \subseteq A^* \times C^*$ defined by

$$u\ S_G\ v \overset{\text{def}}{\Leftrightarrow} b_1 u g \Rightarrow^*_G b_2 v g.$$

Here the *anchors* b_1, b_2 are used to control what happens at the left-hand end of transformed words. Mostly, they ensure that the derivation $b_1 u g \Rightarrow^* b_2 v g$ goes all the way to the left and erases b_1 rather than stopping earlier. One intuitive way of seeing S_G is that it is a variant of R_G *restricted to derivations that replace the anchors*.

A first difficulty for building the transitive closure of an anchored transformation $S_G \subseteq A^* \times C^*$ is that the input and output sets are disjoint (a requirement that allowed the developments of Sections 4 and 5). To circumvent this, we assume w.l.o.g. that A and C are two different copies of a same set, equipped with a bijective renaming $h : C^* \to A^*$. Then, the closure $S_G.(h.S_G)^*$ behaves like we would want S_G^+ to behave.

For the rest of this section, we assume h is a bijection between C and A. W.l.o.g., we write A and C under the forms $A = \{a_1, \ldots, a_n\}$ and $C = \{c_1, \ldots, c_n\}$ so that $h(c_i) = a_i$ for all $i = 1, \ldots, n$. Then h is lifted as a (bijective) morphism $h : C^* \to A^*$ that we sometimes see as a relation between words.

The exact statement we prove in this section is the following:

Theorem 7.1 (Transitive Closure). *Let $G : A \vdash C$ be an ALTr such that $S_G = S_G. \sqsubseteq_C$. Then there exists an ALTr $G^{(+)} : A \vdash C$ such that $S_{G^{(+)}} = S_G.(h.S_G)^*$.*
Furthermore, it is possible to build $G^{(+)}$ from G using only logarithmic space.

Let $b_1, b_2 \notin A \cup C$. The ALTr $\mathsf{R}_{b_2,b_1} \stackrel{\text{def}}{=} (C, b_2, b_1, A, P_{\mathsf{R}}, g)$ with

$$P_{\mathsf{R}} \stackrel{\text{def}}{=} \left\{ \begin{matrix} g \to a_i, a_i \to a_j, a_i \to b_1 \\ a_i \dashrightarrow c_i, b_1 \dashrightarrow b_2 \end{matrix} \;\middle|\; \text{for all } i, j = 1, \ldots, n \right\}$$

is called a *renamer (of C to A)*, and often shortly written R. Observe that $\mathsf{R} : C \vdash A$ is indeed an ALTr. It further satisfies $S_{\mathsf{R}} = \approx . \sqsubseteq .h$.

We shall now glue an ALTr $G : A \vdash C$ with the renamer $\mathsf{R} : C \vdash A$ into some larger LGr H. But before this can be done we need to put some wrapping control on G (and on R) that will let us track what comes from G inside H's derivations.

Formally, given an ALTr $G = (A, B, C, b_1, b_2, P, g)$ and two new anchor symbols $\square_1, \square_2 \notin \Sigma_g$, we let $\Sigma_\square \stackrel{\text{def}}{=} \{\square_1, \square_2\}$ and define a new ALTr $F_{G,\square_1,\square_2}$ (or shortly just F_G) for "*wrapping G with $\square_1, \square_2$*", and given by $F_{G,\square_1,\square_2} \stackrel{\text{def}}{=} (\overline{A}, \overline{B}, \overline{C}, \square_1, \square_2, P', g)$ where
– $\overline{A} \stackrel{\text{def}}{=} A \cup A' \cup \{b_1, b_1'\}$, A', b_1' being a copy of A, b_1,
– $\overline{B} \stackrel{\text{def}}{=} \{\square_1, \square_2\} \cup B \setminus \{b_1\}$,
– $\overline{C} \stackrel{\text{def}}{=} C \cup \{b_2\} \cup C' \cup B' \setminus \{b_1'\}$, B' and C' being copies of B and C.
Finally, let $D \stackrel{\text{def}}{=} C \cup B$ and $D' \stackrel{\text{def}}{=} C' \cup B'$. (The copies are denoted by priming the original symbols, and a primed set like $A' = \{a' \mid a \in A\}$ is just the set of corresponding primed symbols.) The rules in P' are derived from the rules of P in the following way.

kept: P' retains all rules of P that do not erase a letter in $A \cup \{b_1\}$,
replace: P' has a rule $d' \dashrightarrow a$ for each rule $d \dashrightarrow a$ in P that erases a letter in $A \cup \{b_1\}$,
mirror: P' has a rule $d \to d'$ for each $d \in D$,
clean: P' has all rules $d' \dashrightarrow e'$ and $\square_2 \dashrightarrow a'$ for $d', e' \in D' \setminus \{b_1'\}$ and $a' \in A' \cup \{b_1'\}$,
***b*-rules:** P' has the rules $\square_2 \dashrightarrow \square_1$ and all rules $d' \to \square_2$ for $d' \in D' \setminus \{b_1'\}$.

We now relate the derivations in G and the derivations in F_G. For this, assume $u \in (A + b_1)^*$ and $v \in (C + b_2)^+$.

Lemma 7.2. *1. If $u.g \Rightarrow_G^+ v.g$ then for all words $\alpha \in (A' + b_1')^*$ there exists a symbol $\beta \in C' \cup \{b_2'\}$ such that $\Box_1.\alpha.u.g \Rightarrow_{F_G}^+ \Box_1.\alpha.\beta.v.g \Rightarrow_{F_G}^+ \Box_2.\beta.v.g$.*
2. Reciprocally, for all $\alpha \in (A' + b_1')^*$, for all $\beta \in (C' + b_2')^+$ if $\Box_1.\alpha.u.g \Rightarrow_{F_G}^+ \Box_2.\beta.v.g$ then $u.g \Rightarrow_G^+ v.g$.

Thus we can relate anchored derivations in F_G with anchored derivations in G via:

Corollary 7.3. *Let $u \in (A + b_1)^*$ and $v \in (C + b_2)^+$. Then $b_1.u.g \Rightarrow_G^+ b_2.v.g$ if and only if there exists $\beta \in (C' \cup \{b_2'\})$ such that $\Box_1.\alpha.b_1.u.g \Rightarrow_{F_G}^+ \Box_2.\beta.b_2.v.g$. In other words, $u\, S_G\, v$ iff $\alpha.b_1.u\, S_{F_G}\, \beta.b_2.v$ for some $\beta \in (C' \cup \{b_2'\})$.*

We may now glue the wrapped versions of G and its associated R. Recall that $F_G = (\overline{A}, \overline{B}, \overline{C}, \Box_1, \Box_2, P', g)$. We denote the set of new symbols with $\Sigma \overset{\text{def}}{=} \overline{A} \cup \overline{B} \cup \overline{C}$ and observe that F_{R} (short for $F_{\mathsf{R}_{b_2,b_1}, \Box_2, \Box_1}$), being some $(C \cup C' \cup \{b_2, b_2'\}, \Sigma_\Box, \overline{A}, \Box_2, \Box_1, P_{\mathsf{R}}', g)$, does not use more symbols. Let $H \overset{\text{def}}{=} (\Sigma, P_H, g)$ be the LGr such that and $P_H = P' \cup P_{\mathsf{R}}'$. Essentially, H is a union of the two wrapping ALTr's. Note that H is *not a LTr* since it does not respect any distinction between input, intermediary, and output symbols.

Lemma 7.4. *Let $\alpha, \beta \in \underline{A}'^+$ and $u, v \in A^*$. If $\Box_1.\alpha.u.g \Rightarrow_H^* \Box_1.\beta.v.g$ and $S_G = (\sqsubseteq_A .S_G. \sqsubseteq_C)$ then $u \sqsubseteq_A .(S_G.\overline{h})^*\, v$.*

We now extend H to turn it into an ALTr $H' : \dot{A} \vdash A \cup A'$, introducing again new copies, denoted $\dot{a}$, …, of previously used symbols and writing $\dot{u} = \dot{a}_1 \dot{a}_2 \ldots \dot{a}_n$ for the dotted copy of some $u = a_1 a_2 \ldots a_n$. Formally,

$$H' \overset{\text{def}}{=} (\dot{A}, B \cup B' \cup C \cup C' \cup \{\Box_1, \Box_2, \dot{\Box}_1, \dot{\Box}_2\}, A \cup A', \dot{\Box}_1, \dot{\Box}_2, P'', g)$$

where P'' extends P_H by the rules $\dot{\Box}_2 \dashrightarrow \dot{\Box}_1$, $\Box_1 \longrightarrow \dot{\Box}_2$, and all $a \dashrightarrow \dot{a}$ for $a \in A$.

The anchored transformation $S_{H'}$ computed by H' is captured by the following:

Lemma 7.5. *Let $u, v \in A^*$. Then $\dot{u}\, S_{H'}\, \Box_1.\beta.v$ for some $\beta \in A'^+$ iff $u\, [\overline{h}. \sqsubseteq_A .(S_G.\overline{h})^*]\, v$.*

We are nearly done. There only remains to compose H' with a LTr that checks for the presence of $\Box_1.\beta$ (and then erases it). For this last step, we shall use further dotted copies $\ddot{\Sigma}$, $\dddot{\Sigma}$, …, of the previously used symbols.

Formally, we define two new ALTr's T_1 and T_2: see full version. The rules of T_1 ensure that it satisfies

$$u\, S_{T_1}\, v \text{ iff } u = \Box_1.\alpha.b_1.u' \text{ and } \ddot{u}\, I_{T_1^{\mathbf{ins}}}\, v. \qquad (T_1\text{-spec})$$

Regarding T_2, let $u \in (\ddot{A} \cup \ddot{A}' \cup \{\ddot{b}_1, \ddot{b}_1'\})^*$ and $v \in \dddot{A}^*$. If $\ddot{u}'$ is the largest subword of u such that $u' \in A^*$, then

$$u\, S_{T_2}\, v \text{ iff } \dddot{u}' \sqsubseteq_{\dddot{A}} v. \qquad (T_2\text{-spec})$$

Combining (T_1-spec) and (T_2-spec) we obtain

$$u\, S_{T_1}.S_{T_2}\, v \text{ iff } u = \Box_1.\alpha.b_1.u' \text{ and } \dddot{u}' \sqsubseteq_{\dddot{A}} v.$$

Composing these LTr's as $H'.T_1.T_2$ yields a resulting $G^{(+)} : \dot{A} \vdash \dddot{A}$, which, up to a bijective change of symbols, is what we need to build to prove Theorem 7.1.

8 Conclusion

In this paper we introduce a notion of transformations computed by leftist grammars and define constructions showing how these transformations are effectively closed under sequential composition and transitive closure.

These operations require that some “typing” assumptions are satisfied (e.g., we only know how to build a transitive closure on leftist transformers that are “*anchored*”) which may be seen as a lack of elegance and generality of the theory, but which we see as an indication that leftist grammars are very hard to control and reason about.

Anyway, the restrictive assumptions are not a problem for our purposes: we intend to rely on the compositional foundations for building, in a modular way, complex leftist grammars that are able to simulate lossy channel systems. Here the modularity is essential not so much for *building* complex grammars. Rather, it is essential for proving their correctness by a divide-and-conquer approach, in the way we proved the correctness of our encoding of 3SAT instances in Section 6.

As another direction for future work, we would like to mention that the proof that accessibility is decidable for LGr’s (see [7]) has to be fixed and completed.

Acknowledgements. Sylvain Schmitz helped tremendously with his numerous remarks and suggestions.

References

1. P. Chambart and Ph. Schnoebelen. Post embedding problem is not primitive recursive, with applications to channel systems. In *Proc. FST&TCS 2007*, volume 4855 of *Lecture Notes in Computer Science*, pages 265–276. Springer, 2007.
2. P. Chambart and Ph. Schnoebelen. The ω-regular Post embedding problem. In *Proc. FOSSACS 2008*, volume 4962 of *Lecture Notes in Computer Science*, pages 97–111. Springer, 2008.
3. P. Chambart and Ph. Schnoebelen. The ordinal recursive complexity of lossy channel systems. In *Proc. LICS 2008*, pages 205–216. IEEE Comp. Soc. Press, 2008.
4. T. Jurdziński. On complexity of grammars related to the safety problem. *Theoretical Computer Science*, 389(1–2):56–72, 2007.
5. T. Jurdziński. Leftist grammars are nonprimitive recursive. In *Proc. ICALP 2008*, volume 5126 of *Lecture Notes in Computer Science*, pages 51–62. Springer, 2008.
6. T. Jurdziński and K. Loryś. Leftist grammars and the Chomsky hierarchy. *Mathematical Systems Theory*, 41(2):233–256, 2007.
7. R. Motwani, R. Panigrahy, V. A. Saraswat, and S. Venkatasubramanian. On the decidability of accessibility. In *Proc. STOC 2000*, pages 306–315. ACM Press, 2000.