Hardest languages for conjunctive and Boolean grammars[☆]

Alexander Okhotin

St. Petersburg State University, 14th Line V.O., 29B, Saint Petersburg 199178, Russia

Abstract

A famous theorem by Greibach ("The hardest context-free language", *SIAM J. Comp.*, 1973) states that there exists such a context-free language L_0, that every context-free language over any alphabet is reducible to L_0 by a homomorphic reduction—in other words, is representable as an inverse homomorphic image $h^{-1}(L_0)$, for a suitable homomorphism h. This paper establishes similar characterizations for conjunctive grammars, that is, for grammars extended with a conjunction operator, as well as for Boolean grammars, which are further equipped with a negation operator. At the same time, it is shown that no such characterization is possible for several subclasses of linear grammars.

Key words: Context-free grammars, conjunctive grammars, Boolean grammars, homomorphisms, hardest language

1. Introduction

One of the central notions in the theory of computation is that of a *hard set for a class*, to which all other sets from that class can be *reduced*: that is to say, the membership problem for each set in this class can be solved by mapping an element to be tested to an instance of the membership problem for the hardest set, using a relatively easily computable *reduction function*. All basic complexity classes have their hardest sets, and their existence forms the fundamental knowledge in the area. For example, the set of Turing machines recognizing co-finite languages is complete for Σ_0^3 under recursive reductions, whereas Boolean formula satisfiability is complete for NP under polynomial-time reductions.

In the world of formal grammars, there is a result similar in spirit, established for a much more restricted notion of reducibility (which indicates a stronger result). In 1973, Greibach [12] proved that among the languages defined by formal grammars of the ordinary kind ("context-free languages"), there is a hardest set under reductions by *homomorphisms*—that is, by functions mapping each symbol of the target language's alphabet to a string over the alphabet of the hardest language. To be precise, Greibach's hardest language theorem presents a particular context-free language L_0 over an alphabet Σ_0, with the following property: for every context-free language L over any alphabet Σ, there exists such a homomorphism $h\colon \Sigma \to \Sigma_0^+$, that a string $w \in \Sigma^*$ is in L if and only if its image $h(w)$ belongs to L_0. This language L_0 is then called *the hardest context-free language*, for the reason that every context-free language is reducible to it. The hardest language is important, in particular, for serving as the worst-case example for parsing algorithms: indeed, every algorithm that parses L_0 in time $t(n)$ using space $s(n)$ therefore parses every context-free language in time $t(O(n))$ using space $s(O(n))$.

In formal language theory, this result is often regarded as a *representation theorem*, in the sense that every language in the family is representable by applying a certain operation to base languages of a restricted

[☆]A preliminary version of this paper, entitled "The hardest language for conjunctive grammars", was presented at the CSR 2016 conference held in St. Petersburg, Russia, on 9–13 June 2016, and its extended abstract appeared in the conference proceedings.

Email addresses: alexander.okhotin@utu.fi (Alexander Okhotin)

form. In the case of Greibach's theorem, there is just one base language L_0, and every context-free language L is represented as an *inverse homomorphic image* $L = \{ w \mid h(w) \in L_0 \} = h^{-1}(L_0)$. For context-free languages, it is additionally known that they are closed under taking inverse homomorphisms, and thus Greibach's theorem [12] gives a necessary and sufficient condition: a language is context-free if and only if it is representable as $h^{-1}(L_0)$, for some h. Another famous example of a representation theorem is the Chomsky–Schützenberger theorem, on which some progress has recently been made [7, 26].

The possibility of having hardest sets under homomorphic reductions was also investigated for a few other families of formal languages. Already Greibach [13] showed that for the family of *deterministic languages*—that is, the languages described by LR(1) grammars, or, equivalently, recognized by deterministic pushdown automata—there cannot be a hardest language under homomorphic reductions. Greibach [12] has also proved a variant of her theorem for a certain restricted type of Turing machines. Čulík and Maurer [8] similarly found the same kind of a hardest language in the complexity class NSPACE(n). Boasson and Nivat [6] proved that there is no hardest language in the family described by linear grammars, whereas Autebert [3] proved the same negative result for one-counter automata. For the regular languages, non-existence of a hardest language under homomorphic reductions was established by Čulík and Maurer [8]; a strengthened form of this result was established by Harju, Karhumäki and Kleijn [14].

The purpose of this paper is to prove the existence of the hardest language for another family of formal grammars, the *conjunctive grammars* [23, 27]: this is an extension of ordinary grammars that allows a conjunction of any syntactic conditions to be expressed in any rule. Consider that a rule $A \to BC$ in an ordinary grammar states that if a string w is representable as BC—that is, as $w = uv$, where u has the property B and v has the property C—then w has the property A. In a conjunctive grammar, one can define a rule of the form $A \to BC \,\&\, DE$, which asserts that every string w representable *both* as BC (with $w = uv$) *and at the same time* as DE (with $w = xy$) therefore has the property A. The importance of conjunctive grammars is justified by two facts: on the one hand, they enrich the standard inductive definitions of syntax with useful logical operations, and these operations are sufficient to express several syntactic constructs beyond the scope of ordinary grammars. On the other hand, conjunctive grammars have generally the same parsing algorithms as ordinary grammars [1, 27], and the same subcubic upper bound on the time complexity of parsing [28]. Among numerous theoretical results on conjunctive grammars, the one particularly relevant for this paper is the closure of the language family described by conjunctive grammars under inverse homomorphisms, and, more generally, under inverse deterministic finite transductions [21]. For more information on this grammar family, the reader is directed to a recent survey paper [27].

The main result of this paper, presented in Section 3, is that there exists a hardest language for conjunctive grammars. The proof of the conjunctive version of Greibach's theorem requires a construction substantially different from the one used in the case of ordinary grammars. Greibach's own proof of her theorem essentially uses the fact that the given grammar can be transformed to the *Greibach normal form* [11], with all rules of the form $A \to a\alpha$, where a is a symbol of the alphabet. On the other hand, it is not known whether the class of languages described by conjunctive grammars in the Greibach normal form—see Kuznetsov [20]—is equal to the entire class described by conjunctive grammars. For this reason, the proof of the hardest language theorem presented in this paper has to rely upon a different kind of a normal form defined by Okhotin and Reitwießner [29]. Using that normal form instead of the Greibach normal form requires a new construction for proving the theorem.

The second result of the paper is yet another analogue of Greibach's inverse homomorphic characterization, this time for the class of *Boolean grammars*. Boolean grammars [25] are a further extension of conjunctive grammars featuring a negation operator. Boolean grammars allow such rules as $A \to BC \,\&\, \neg DE$, which expresses all strings representable as BC, but not representable as DE. In Section 4, the construction of Section 3 is adapted to establish a hardest language theorem for Boolean grammars.

The last result refers to the family of *linear grammars*. It is known from Boasson and Nivat [6] that this family has no hardest language under homomorphic reductions. For completeness, a similar negative result is established in Section 5 for several subclasses of the linear grammars, including unambiguous linear grammars, LR linear grammars and LL linear grammars.

The concluding Section 6 elaborates on the prospects of finding hardest languages (or proving non-existence thereof) for other important families of formal grammars.

2. Conjunctive grammars

Rules in ordinary formal grammars may concatenate substrings to each other, and may use *disjunction* of syntactic conditions, represented by multiple rules for a nonterminal symbol. *Conjunctive grammars* extend this logic by allowing conjunction within the same kind of definitions. Nothing else is changed; in particular, in a conjunctive grammar, the properties of a substring are independent of the context in which it occurs.

Definition 1. *A conjunctive grammar is a quadruple $G = (\Sigma, N, R, S)$, in which:*

- Σ *is the* alphabet *of the language being defined;*
- N *is a finite set of symbols for the syntactic categories defined in the grammar ("nonterminal symbols");*
- R *is a finite set of* rules*, each of the form*

$$A \to \alpha_1 \,\&\, \ldots \,\&\, \alpha_m, \tag{1}$$

  where $A \in N$, $m \geqslant 1$ and $\alpha_1, \ldots, \alpha_m \in (\Sigma \cup N)^*$;
- $S \in N$ *is a symbol representing the property of being a syntactically well-formed sentence of the language ("the initial symbol").*

Each concatenation α_i in a rule (1) is called a *conjunct*. If a grammar has a unique conjunct in every rule ($m = 1$), it is an *ordinary grammar* (Chomsky's "context-free").

Each rule (1) means that any string representable as each concatenation α_i therefore has the property A. These dependencies form *parse trees with shared leaves*, where the subtrees corresponding to different conjuncts in a rule (1) define multiple interpretations of the same substring, and accordingly lead to the same set of leaves, as illustrated in Figure 1. This understanding can be formalized in several equivalent ways: by term rewriting [23], by logical deduction, and by language equations [27].

Consider one of those definitions, which extends Chomsky's definition of ordinary grammars by string rewriting, using terms instead of strings.

Definition 2 ([23])**.** *Let $G = (\Sigma, N, R, S)$ be a conjunctive grammar, and consider terms over concatenation and conjunction, with symbols from $\Sigma \cup N$ and the empty string ε as atomic terms. The relation of one-step rewriting on such terms ($\Longrightarrow$) is defined as follows.*

- *Using a rule $A \to \alpha_1 \,\&\, \ldots \,\&\, \alpha_m \in R$, any atomic subterm $A \in N$ of any term may be rewritten by the term $(\alpha_1 \,\&\, \ldots \,\&\, \alpha_m)$.*

$$\ldots A \ldots \Longrightarrow \ldots (\alpha_1 \,\&\, \ldots \,\&\, \alpha_m) \ldots$$

- *A conjunction of several identical strings may be rewritten to one such string.*

$$\ldots (w \,\&\, \ldots \,\&\, w) \ldots \Longrightarrow \ldots w \ldots \qquad\qquad (w \in \Sigma^*)$$

The language defined by a term φ is the set of all strings over Σ obtained from it in finitely many rewriting steps.

$$L_G(\varphi) = \{\, w \mid w \in \Sigma^*,\ \varphi \Longrightarrow^* w \,\}$$

The language defined by the grammar is the language defined by its initial symbol.

$$L(G) = L_G(S) = \{\, w \mid w \in \Sigma^*,\ S \Longrightarrow^* w \,\}$$

Example 1 ([23])**.** *The following conjunctive grammar describes the language $\{\, a^n b^n c^n \mid n \geqslant 0 \,\}$.*

$$\begin{array}{lcl} S & \to & AB \,\&\, DC \\ A & \to & aA \mid \varepsilon \\ B & \to & bBc \mid \varepsilon \\ C & \to & cC \mid \varepsilon \\ D & \to & aDb \mid \varepsilon \end{array}$$

In particular, the string $w = abc$ is generated by term rewriting as follows.

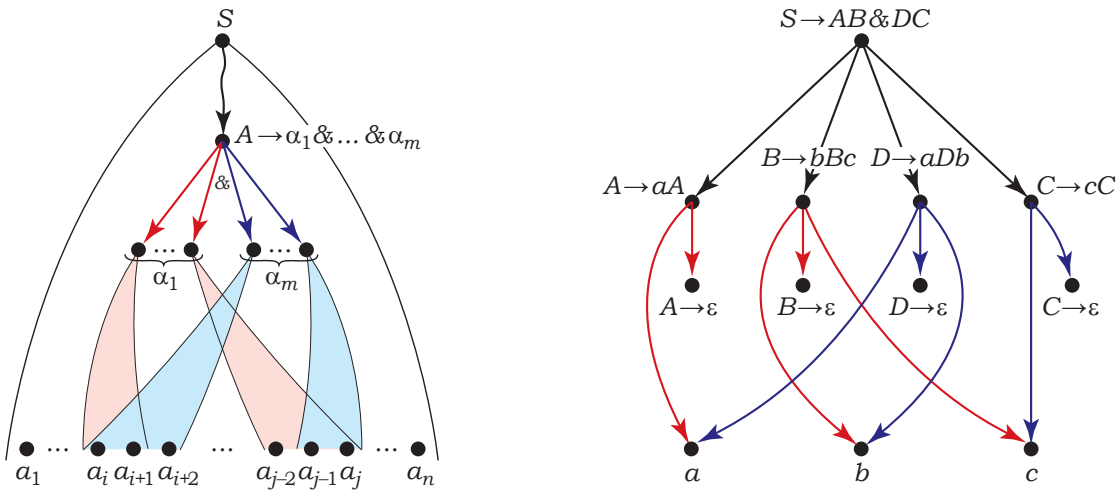

Figure 1: Parse trees in conjunctive grammars: (left) a subtree with root $A \to \alpha_1 \,\&\, \ldots \,\&\, \alpha_m$, representing m parses of a substring $a_i \ldots a_j$; (right) a tree of $w = abc$ in Example 1.

$S \Longrightarrow (AB \,\&\, DC) \Longrightarrow (aAB \,\&\, DC) \Longrightarrow (aB \,\&\, DC) \Longrightarrow (abBc \,\&\, DC) \Longrightarrow (abc \,\&\, DC) \Longrightarrow (abc \,\&\, aDbC)$
$\Longrightarrow (abc \,\&\, abC) \Longrightarrow (abc \,\&\, abcC) \Longrightarrow (abc \,\&\, abc) \Longrightarrow abc$

The proof of a hardest language theorem for conjunctive grammars presented in this paper is based upon a normal form that extends the *operator normal form* for ordinary grammars [9]. This normal form, called the *odd normal form*, derives its name from the fact that all strings generated by all nonterminals (except maybe the initial symbol) are of odd length.

Theorem A (Okhotin, Reitwießner [29])**.** *For every conjunctive grammar there exists and can be effectively constructed a conjunctive grammar $G = (\Sigma, N, R, S)$ generating the same language, which is in the* ***odd normal form****, that is, with all rules of the following form.*

$$A \to B_1 a_1 C_1 \,\&\, \ldots \,\&\, B_m a_m C_m \qquad (m \geqslant 1,\ B_i, C_i \in N,\ a_i \in \Sigma)$$
$$A \to a \qquad (a \in \Sigma)$$

If the initial symbol S is never used in the right-hand sides of any rules, then two special kinds of rules for S are also allowed.

$$S \to aA \qquad (a \in \Sigma,\ A \in N)$$
$$S \to \varepsilon$$

In the odd normal form, every conjunct contains a symbol of the alphabet, such as a in BaC. This property is essential for the argument presented in this paper, because the homomorphic image $h(a)$ of that symbol shall, in particular, encode the conjunct BaC. A substring can then be parsed according to that conjunct, beginning from the image $h(a)$, and then proceeding in both directions to parse the appropriate substrings according to B and to C.

3. Hardest language for conjunctive grammars

Let Σ and Σ_0 be two finite alphabets, the sets of strings over these alphabets are denoted by Σ^* and Σ_0^*, the empty string is denoted by ε. A homomorphism h is a function mapping Σ^* to Σ_0^* that satisfies the following conditions: the empty string is mapped to the empty string ($h(\varepsilon) = \varepsilon$), and the image of a concatenation is a concatenation of images ($h(uv) = h(u)h(v)$ for all $u, v \in \Sigma^*$). Every homomorphism

$h\colon \Sigma^* \to \Sigma_0^*$ is completely defined by the images of individual symbols from Σ, so that the image of a string $w = a_1 \dots a_n$, with $n \geqslant 0$ and $a_1, \dots, a_n \in \Sigma$, is the string $h(w) = h(a_1) \dots h(a_n) \in \Sigma_0^*$.

The goal is to establish the following analogue of Greibach's [11] hardest language theorem for conjunctive grammars.

Theorem 1. *There exists such a language L_0 over the alphabet $\Sigma_0 = \{a, b, c, d, \#\}$ described by a conjunctive grammar, that for every language L over any alphabet Σ described by some conjunctive grammar, there is such a homomorphism $h\colon \Sigma \to \Sigma_0^*$, that $L = h^{-1}(L_0)$ if $\varepsilon \notin L$, and $L = h^{-1}(L_0 \cup \{\varepsilon\})$ if $\varepsilon \in L$.*

Let $G = (\Sigma, N, R, X)$ be a conjunctive grammar that describes the language L. In view of Theorem A, every conjunct in G can be assumed to be of the form YsZ, Ys, sZ or s, where $Y, Z \in N$ and $s \in \Sigma$. The first part of the proof is the description of a particular homomorphism h_G that embeds the grammar G in the images of symbols from Σ. Once the construction of such homomorphisms is explained, the second step will be to present a conjunctive grammar G_0 that defines a string $h_G(w)$ if and only if $w \in L(G)$. Then, the language described of this grammar is the desired language L_0.

Let $\mathcal{C} = \{\alpha_1, \alpha_2, \dots, \alpha_{|\mathcal{C}|}\}$, with $\alpha_i \in N\Sigma N \cup \Sigma N \cup N\Sigma \cup \Sigma$, be an enumeration of all distinct conjuncts used in the grammar, so that each rule is of the following form.

$$A \to \alpha_{i_1} \,\&\, \dots \,\&\, \alpha_{i_m} \qquad (m \geqslant 1,\ i_1, \dots, i_m \in \{1, \dots, |\mathcal{C}|\}) \tag{2}$$

The proposed encoding of G consists of *definitions of conjuncts*. For each conjunct $\alpha_i = YsZ$, its definition is included in the image of the symbol s, so that a substring $h(usv)$ could be parsed according to YsZ beginning from $h(s)$ in the middle. The definition consists of all possible *expansions* of a conjunct, with any rules for Y and for Z substituted instead of Y and Z. Consider any two such rules.

$$Y \to \alpha_{i_1} \,\&\, \dots \,\&\, \alpha_{i_m}$$
$$Z \to \alpha_{j_1} \,\&\, \dots \,\&\, \alpha_{j_n}$$

The expansion of conjunct number i with this pair of rules represents them as two lists of conjunct numbers: the list of conjuncts $\{i_1, \dots, i_m\}$ to be used on the left, representing the rule for Y, and the list of conjuncts $\{j_1, \dots, j_n\}$ on the right. This expansion is then encoded in the image of the symbol s as a triple $\big(\{i_1, \dots, i_m\},\ i,\ \{j_1, \dots, j_n\}\big)$. The image of s contains such triples for every conjunct YsZ and for every choice of rules for Y and for Z.

The encoding uses a five-symbol alphabet $\Sigma_0 = \{a, b, c, d, \#\}$, in which the symbols have the following meaning.

- Symbols a are used to represent any reference to each conjunct α_i as a^i.
- Symbols b are used to mark each expansion of a conjunct $\alpha_i = YsZ$ by b^i.
- The symbol c represents conjunction in the right-hand side of any rule. Each rule (2) has the following symmetric *left* and *right* representations that list all conjuncts used in the rule.

$$\lambda(A \to \alpha_{i_1} \,\&\, \dots \,\&\, \alpha_{i_m}) = ca^{i_m} \dots ca^{i_1}$$
$$\rho(A \to \alpha_{i_1} \,\&\, \dots \,\&\, \alpha_{i_m}) = a^{i_1}c \dots a^{i_m}c$$

  Any expansion of a conjunct $\alpha_k = YsZ$ consists of a marker b^k preceded by a left representation of a rule r for Y and followed by a right representation of a rule r' for Z. This is a string $\lambda(r)b^k\rho(r')$. For a conjunct $\alpha_k = Ys$ with Z omitted, its expansion accordingly omits $\rho(r')$ and is of the form $\lambda(r)b^k$. Similarly, a conjunct $\alpha_k = sZ$ is expanded as $b^k\rho(r')$, where r' is a rule for Z. A conjunct $\alpha_k = s$ has a unique expansion b^k.

- The symbol d is used to separate any expansions of conjuncts with the same symbol s in the middle. The definition of a conjunct α_k, denoted by $\sigma(\alpha_k)$, is the following concatenation of all its expansions (where the product notation denotes a concatenation over all rules).

$$\sigma(\alpha_k) = \begin{cases} \prod_{\substack{r \text{ is a rule for } Y \\ r' \text{ is a rule for } Z}} \lambda(r) b^k \rho(r') d, & \text{if } \alpha_k = YsZ \\ \prod_{r \text{ is a rule for } Y} \lambda(r) b^k d, & \text{if } \alpha_k = Ys \\ \prod_{r' \text{ is a rule for } Z} b^k \rho(r') d, & \text{if } \alpha_k = sZ \\ b^k d, & \text{if } \alpha_k = s \end{cases}$$

- The separator symbol $\# \in \Sigma_0$ concludes the image $h(s)$ of any symbol $s \in \Sigma$.

The image of every symbol $s \in \Sigma$ under h consists of two parts. It begins with the set of all rules for the initial symbol, which is actually needed only in the image of the first symbol of a string. Next, after a double separator dd, there is the list of all expansions of all conjuncts containing the symbol s. The image is concluded with the separator symbol $\#$.

$$h_G(s) = \left(\prod_{r \text{ is a rule for } S} \rho(r) d \right) \cdot d \cdot \left(\prod_{\alpha_k \in \mathcal{C}} \sigma(\alpha_k) \right) \cdot \#$$

These definitions are illustrated on the following grammar.

Example 2. *Let $\Sigma = \{s, t\}$ and consider a conjunctive grammar $G = (\Sigma, \{X, Y, Z\}, R, X)$, with the following rules.*

$$\begin{aligned} X &\to tY \\ Y &\to YsZ \,\&\, ZsZ \mid t \\ Z &\to t \end{aligned}$$

This grammar defines only two strings, tt and $ttst$, and the parse tree of the string $ttst$ is presented in Figure 2.

Let the conjuncts in G be numbered as $\alpha_1 = tY$, $\alpha_2 = YsZ$, $\alpha_3 = ZsZ$ and $\alpha_4 = t$. Then, its rules have the following left and right representations.

$$\begin{aligned} \lambda(X \to tY) &= ac & \qquad \rho(X \to tY) &= ca \\ \lambda(Y \to YsZ \,\&\, ZsZ) &= ca^3ca^2 & \qquad \rho(Y \to YsZ \,\&\, ZsZ) &= a^2ca^3c \\ \lambda(Y \to t) &= a^4c & \qquad \rho(X \to tY) &= ca^4 \\ \lambda(Z \to t) &= a^4c & \qquad \rho(Z \to tY) &= ca^4 \end{aligned}$$

The image of the symbol s begins with the rule for the initial symbol X and continues with the expansions of the conjuncts YsZ and ZsZ (that is, of all conjuncts containing the symbol s). The conjunct YsZ has two expansions, one with the first rule for Y and the other with the second rule for Y. The conjunct ZsZ has only one expansion, because Z has a unique rule $Z \to t$.

$$h_G(s) = \underbrace{ac}_{\rho(X \to tY)} \boldsymbol{dd} \overbrace{ca^3ca^2}^{\lambda(Y \to YsZ \,\&\, ZsZ)} b^2 \underbrace{a^4c}_{\rho(Z \to t)} \boldsymbol{d} \overbrace{ca^4}^{\lambda(Y \to t)} b^2 \underbrace{a^4c}_{\rho(Z \to t)} \boldsymbol{d} \overbrace{ca^4}^{\lambda(Z \to t)} b^3 \underbrace{a^4c}_{\rho(Z \to t)} \boldsymbol{d}\#$$

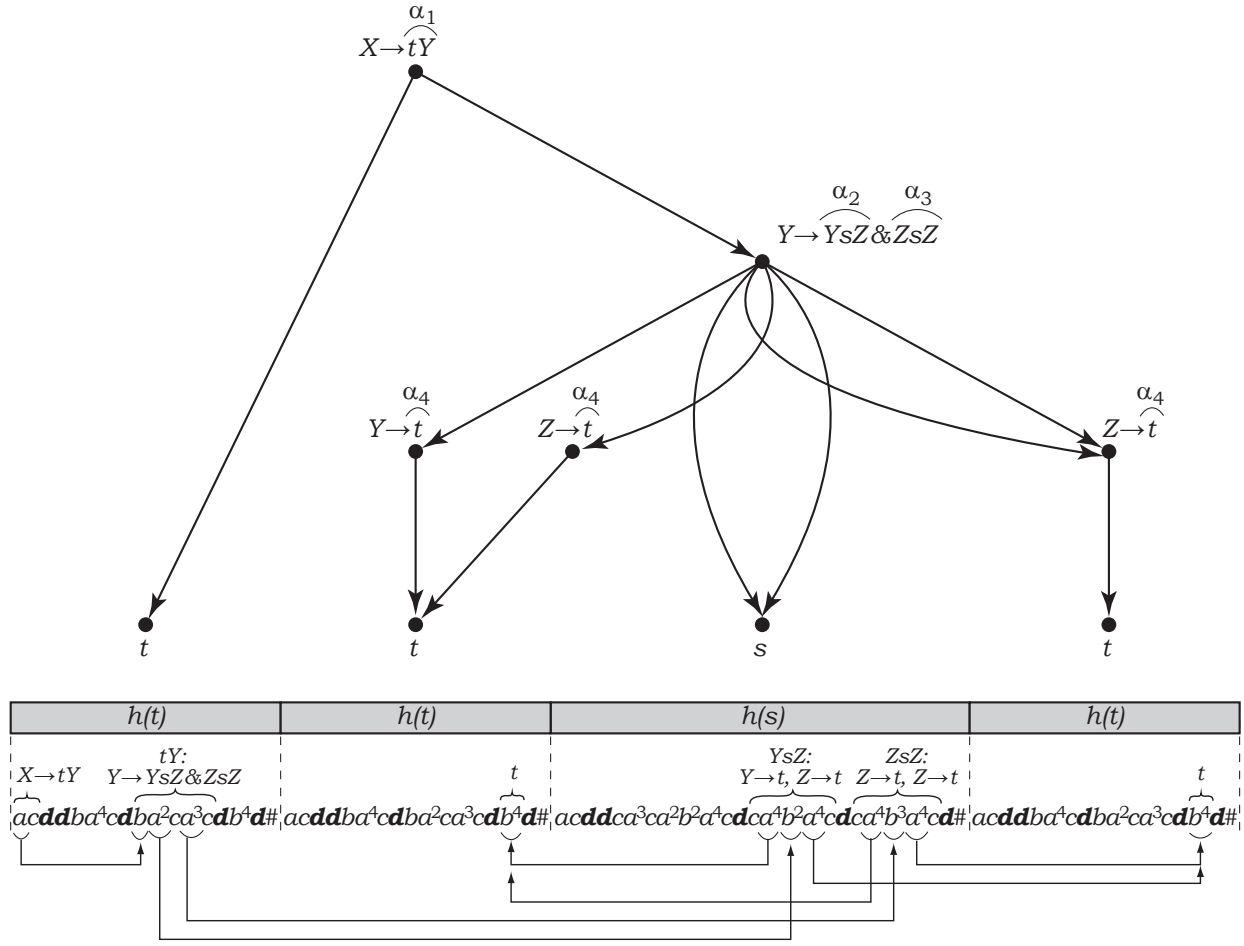

Figure 2: How a parse tree of the string $w = ttst$ is reconstructed by analyzing its image $h(w)$.

The image of t begins with the same rule for the initial symbol X. Then, there are two conjuncts to expand: tY and t. The former conjunct has two expansions corresponding to the two rules for Y, whereas the conjunct $\alpha_4 = t$ has a unique expansion not referring to any rules (b^4).

$$h_G(t) = \underbrace{ac}_{\rho(X\to tY)}\,\boldsymbol{dd}\,b\,\underbrace{a^4c}_{\rho(Y\to t)}\,\boldsymbol{d}\,b\,\overbrace{a^2ca^3c}^{\rho(Y\to YsZ\,\&\,ZsZ)}\,\boldsymbol{d}\,b^4\boldsymbol{d}\#,$$

Accordingly, the string ttst ∈ L has the following image.

$$h_G(ttst) = acddba^4cdba^2ca^3cdb^4d\#\ acddba^4cdba^2ca^3cdb^4d\#$$
$$acddca^3ca^2b^2a^4cdca^4b^2a^4cdca^4b^3a^4cd\#\ acddba^4cdba^2ca^3cdb^4d\#$$

The purpose of the encoding h_G is to test the membership of a string w in $L(G)$ by analyzing its homomorphic image $h_G(w)$, following the pointers inside the image to reconstruct a parse of w. Consider how this is done for the grammar in Example 2. Given the image $h_G(ttst)$, the analysis begins by looking at the image $h_G(t)$ of the first symbol and choosing any of the encoded rules for the initial symbol X, which are listed in the beginning of the image. There is only one such rule, $X \to \alpha_1$, where $\alpha_1 = tY$, and this rule is represented by the first symbols *acd* of the image. Here a is a reference to the conjunct α_1, the next

symbol c marks the end of the conjunct's description, and the final d indicates that there are no further conjuncts in this rule.

The code a points to the conjunct $\alpha_1 = tY$, and should accordingly be matched to any code b representing one of its expansions. Two such expansions are located in the image of every symbol t: one using a rule $Y \to t$ (ba^4c), and the other assuming that Y is expanded by the rule $Y \to YsZ \,\&\, ZsZ$ (ba^2ca^3c). Following the parse tree in Figure 2, in this case, the symbol t in tY should be the first symbol of the string $ttst$, and one should use the expansion of Y by the rule $Y \to YsZ \,\&\, ZsZ$. Hence, the recognition proceeds by matching this code a to the substring ba^2ca^3c located within the image $h_G(t)$ of the first symbol t, as demonstrated at the bottom of Figure 2 by the leftmost arrow.

The substring ba^2ca^3c points to two conjuncts, $\alpha_2 = YsZ$ and $\alpha_3 = ZsZ$, which are to be checked independently. Consider the conjunct $\alpha_2 = YsZ$, for which the image $h_G(s)$ of some symbol s should contain the corresponding code b^2, surrounded by encodings of a rule for Y and a rule for Z. The pre-image of the input string in question contains a unique symbol s, and one should use the expansion with the rules $Y \to \alpha_4$ and $Z \to \alpha_4$ used for Y and Z, where $\alpha_4 = t$. This expansion is encoded in a substring $ca^4b^2a^4c$, which points to a conjunct α_4 located to the left of the current symbol s (ca^4), and to another conjunct α_4 located to the right (a^4c). Finally, the encoding ca^4 of the conjunct on the left is matched to the string b^4 in the image $h_G(t)$ to the left, while a^4c is matched to the similar string b^4 in the other image $h_G(t)$ to the right. These connections, along with the similar parse for the conjunct ZsZ, are illustrated in Figure 2.

The above example demonstrates how a parse tree of a string w according to a grammar G can be decoded from the homomorphic image $h_G(w)$. In order to prove Theorem 1, one should implement all of this in a conjunctive grammar. This grammar $G_0 = (\Sigma_0, N_0, R_0, S_0)$ should describe the structure of any such encoding $h_G(w)$, that w is in $L(G)$, whereas all encodings $h_G(w')$ with $w' \notin L(G)$ should accordingly not be in $L(G_0)$.

The grammar G_0 uses the following 13 nonterminals; the purpose of each of them is explained below, along with the rules of the grammar.

$$N_0 = \{S_0, A, B, C, D, \overrightarrow{E}, \overrightarrow{E_+}, \overrightarrow{F}, \overleftarrow{E}, \overleftarrow{E_+}, \overleftarrow{F}, \overrightarrow{E_0}, \overrightarrow{F_0}\}$$

The main task carried out in the grammar G_0 is parsing a substring $w \in \Sigma^*$ according to a rule r of G, given its image $h(w)$. Unless w is the whole string, the matching begins in the image of either of the symbols adjacent to w. Assume that this is the symbol immediately preceding w (the other case is handled symmetrically), and let $t \in \Sigma^*$ be that symbol.

The grammar G_0 implements this by defining a string $\rho(r)dx\#h_G(w)$, where $\rho(r)dx\#$ is a suffix of $h(t)$ that begins with an encoding of the desired rule, and the remaining symbols in $h(t)$ are denoted by $x \in \{a, b, c, d\}^*$. Such a string is defined by the nonterminal $\overrightarrow{E}$, which should verify that each of the conjuncts listed in $\rho(r)$ generates the substring w.

Let the rule r be $A \to \alpha_{i_1} \,\&\, \ldots \,\&\, \alpha_{i_m}$, so that its right representation is $\rho(r) = a^{i_1}c \ldots a^{i_m}c$. Then, $\overrightarrow{E}$ describes the form of the string $a^{i_1}ca^{i_2}c \ldots a^{i_m}cdx\#h_G(w)$ by checking that w can be parsed as the conjunct α_{i_1}, and at the same time by applying the same test to the string $a^{i_2}c \ldots a^{i_m}cdx\#h_G(w)$, in order to check the rest of the conjuncts of this rule (the self-reference actually uses a variant of $\overrightarrow{E}$, denoted by $\overrightarrow{E_+}$, which is explained later).

Consider how the first conjunct α_{i_1} is checked by $\overrightarrow{E}$. Assume that it is of the form $\alpha_{i_1} = YsZ$, with $Y, Z \in N$ and $s \in \Sigma$. For w to be generated by this conjunct, there should exist a partition $w = usv$, with u generated by Y using some rule r', and with v generated by Z using a rule r''. Then the image of s contains a substring $\lambda(r')b^{i_1}\rho(r'')$ corresponding to this case, with $h(s) = x'\lambda(r')b^{i_1}\rho(r'')x''$. The task for $\overrightarrow{E}$ is to locate this substring and then apply another instance of $\overrightarrow{E}$ to the suffix $\rho(r'')x''h(v)$ in order to test the generation of v by r''; the prefix $h(u)x'\lambda(r'')$ is similarly checked by a symbol $\overleftarrow{E}$ with symmetrically defined

rules, ensuring that u is generated by r'.

$$\overbrace{a^{i_1}ca^{i_2}c\dots a^{i_m}cdx\#\cdot\overbrace{h(u)\cdot x'\ \lambda(r')}^{\overleftarrow{E}}b^{i_1}\overbrace{\rho(r'')\ x''\cdot h(v)}^{\overrightarrow{E}}}^{\overrightarrow{F}}\ \in\ L_{G_0}(\overrightarrow{E})$$

Two rules are defined for $\overrightarrow{E}$. The first rule begins the test for an encoded rule r, as explained above, with the first conjunct using an intermediate symbol $\overrightarrow{F}$ to check the conjunct's number and to apply $\overleftarrow{E}$ to the correct substring, and with the second conjunct skipping $a^{i_1}c$ and invoking $\overrightarrow{E_+}$ to test the rest of the rule r, beginning with the conjunct i_2.

$$\overrightarrow{E} \to \overrightarrow{F}\,\overrightarrow{E}\ \&\ Ac\overrightarrow{E_+} \tag{3a}$$
$$A \to aA \mid a \tag{3b}$$

The other rule for $\overrightarrow{E}$ handles the case when the rule r is entirely missing, and the image of t contains an encoding of a conjunct Xt or t. In this case, $\overrightarrow{E}$ has to ensure that w is empty, which is handled by the following rule.

$$\overrightarrow{E} \to dC\# \tag{3c}$$
$$C \to aC \mid bC \mid cC \mid dC \mid \varepsilon \tag{3d}$$

The rules for $\overrightarrow{F}$ are responsible for matching the conjunct's code a^{i_1} to the corresponding code b^{i_1} for this conjunct's expansion, and then skip the rest of the image $h(t)$ to apply $\overleftarrow{E}$ to $h(u)x'\lambda(r'')$.

$$\overrightarrow{F} \to a\overrightarrow{F}b \mid acC\#\overleftarrow{E}b \tag{3e}$$

The rest of the conjuncts of this rule (from α_{i_2} to α_{i_m}) are checked in exactly the same way, using the nonterminal $\overrightarrow{E_+}$ with the same main rule.

$$\overrightarrow{E_+} \to \overrightarrow{F}\,\overrightarrow{E}\ \&\ Ac\overrightarrow{E_+} \tag{3f}$$

Once all the conjuncts are handled, the second rule for $\overrightarrow{E_+}$ skips the remaining string $dx\#h(w)$.

$$\overrightarrow{E_+} \to dC\#D \tag{3g}$$
$$D \to C\#D \mid \varepsilon \tag{3h}$$

Symmetrically to $\overrightarrow{E}$, the nonterminal $\overleftarrow{E}$ describes a string $h_G(w)x\lambda(r)$, with $\lambda(r) = ca^{i_m}\dots ca^{i_2}ca^{i_1}$ and $x \in \{a,b,c,d\}^*$, as long as w is generated by each of the conjuncts $\alpha_{i_1}, \dots, \alpha_{i_m}$.

$$\overleftarrow{E} \to \overleftarrow{E}\,\overleftarrow{F}\ \&\ \overleftarrow{E_+}cA \mid Cd \tag{3i}$$
$$\overleftarrow{E_+} \to \overleftarrow{E}\,\overleftarrow{F}\ \&\ \overleftarrow{E_+}cA \mid DCd \tag{3j}$$
$$\overleftarrow{F} \to b\overleftarrow{F}a \mid b\overrightarrow{E}Cca \tag{3k}$$

It remains to define the rules for the initial symbol S_0, which should describe an encoding $h_G(w)$ if and only if the initial symbol X in the grammar G generates w. Let $w = s_1 \dots s_\ell$. The image of its first symbol s_1 begins with the list of all the rules for S in G; the rule for S_0 chooses one of these encoded rules and then uses another variant of $\overrightarrow{E}$, called $\overrightarrow{E_0}$, to match the whole string w according to this rule.

$$S_0 \to BdS_0 \mid \overrightarrow{F_0}\,\overrightarrow{E}\ \&\ Ac\overrightarrow{E_0} \tag{3l}$$
$$B \to aB \mid cB \mid a \mid c \tag{3m}$$

The reason for using a new nonterminal $\overrightarrow{E_0}$ instead of $\overrightarrow{E_+}$ is that the encoded rule for X is written in the image of the first symbol of the string, and that symbol has to be parsed along with the rest of the symbols. The rules for $\overrightarrow{E_0}$ are the same as for $\overrightarrow{E_+}$, except for using $\overrightarrow{F_0}$ instead of $\overrightarrow{F}$.

$$\overrightarrow{E_0} \to \overrightarrow{F_0}\overrightarrow{E} \,\&\, Ac\overrightarrow{E_0} \mid dC\#D \tag{3n}$$

The rules for $\overrightarrow{F_0}$ are different from those for $\overrightarrow{F}$ in two respects. If the conjunct in the rule for X is of the form Ys or YsZ, then s is not the first symbol of the string, and $\overrightarrow{F_0}$ locates the definition of that conjunct in the image $h(s)$ in the same way as $\overrightarrow{F}$, but has to ensure that the string parsed according to Y begins already with s_1. For that reason, unlike in the corresponding rule for $\overrightarrow{F}$, the separator symbol (#) is not used.

$$\overrightarrow{F_0} \to a\overrightarrow{F_0}b \mid ac\overleftarrow{E}b \tag{3o}$$

The other case is when the conjunct in the rule for X is of the form sY or just s: then, $s = s_1$ is the first symbol of the string, and a rule written in the first part of its image, $h'(s_1)$, should be matched to its definition included in the second part $h''(s_1)$. This is ensured in the next rule.

$$\overrightarrow{F_0} \to acCdb \tag{3p}$$

This completes the construction of the "hardest" conjunctive grammar G_0, and it remains to prove that this grammar satisfies the condition in Theorem 1.

For every symbol $s \in \Sigma$, denote the first part and the second parts of its image by $h'(s)$ and $h''(s)$, respectively.

$$h(s) = \underbrace{\left(\prod_{r \text{ is a rule for } S} \rho(r)d\right)}_{h'(s)} \cdot d \cdot \underbrace{\left(\prod_{\alpha_k \in \mathcal{C}} \sigma(\alpha_k)\right)}_{h''(s)} \cdot \#$$

The entire correctness statement of the construction reads as follows.

Lemma 1. *Let $G = (\Sigma, N, R, X)$ be a conjunctive grammar with all conjuncts of the form YsZ, Ys, sZ or s, where $Y, Z \in N$ and $s \in \Sigma$, and let $h\colon \Sigma^* \to \Sigma_0^*$ be a homomorphism constructed for G as defined above. Let $G_0 = (\Sigma_0, N_0, R_0, S_0)$ be the grammar presented above. Then, the following statements hold.*

I. *A string $a^{i_1}ca^{i_2}c\dots a^{i_m}cdx\#h(w)$, with $m \geqslant 1$, $i_1, \dots, i_m \in \{1, \dots, |\mathcal{C}|\}$, $x \in \{a,b,c,d\}^*$ and $w \in \Sigma^*$, is in $L_{G_0}(\overrightarrow{E})$ if and only if w is in $\bigcap_{j=1}^m L_G(\alpha_{i_j})$.*
A string of this form is in $L_{G_0}(\overrightarrow{E_+})$ under the same condition.

II. *A string $dx\#h(w)$, with $x \in \{a,b,c,d\}^*$ and $w \in \Sigma^*$, is in $L_{G_0}(\overrightarrow{E})$ if and only if $w = \varepsilon$.*
Such a string is in $L_{G_0}(\overrightarrow{E_+})$ regardless of w.

III. *A string $h(w)xdca^{i_m}\dots ca^{i_2}ca^{i_1}$, with $m \geqslant 1$, $i_1, \dots, i_m \in \{1, \dots, |\mathcal{C}|\}$, $x \in \{a,b,c,d\}^*$ and $w \in \Sigma^*$, is in $L_{G_0}(\overleftarrow{E})$ if and only if w is in $\bigcap_{j=1}^m L_G(\alpha_{i_j})$.*
If $s_1 \in \Sigma$ is the first symbol of w, and $w = s_1w'$, then a string $x_0dh''(s_1)h(w')xdca^{i_m}\dots ca^{i_2}ca^{i_1}$ with the partial image of the first symbol, where $x_0 \in \{a,c,d\}^*$, is in $L_{G_0}(\overleftarrow{E})$ if and only if w is in $\bigcap_{j=1}^m L_G(\alpha_{i_j})$.
Such a string is in $L_{G_0}(\overleftarrow{E_+})$ under the same condition.

IV. *A string $h(w)xd$, with $w \in \Sigma^*$ and $x \in \{a,b,c,d\}^*$, is in $L_{G_0}(\overleftarrow{E})$ if and only if $w = \varepsilon$. A string $x_0dh''(s_1)h(w')xdca^{i_m}\dots ca^{i_2}ca^{i_1}$ with the partial image of the first symbol is never in $L_{G_0}(\overleftarrow{E})$.*
Such a string is in $L_{G_0}(\overleftarrow{E_+})$ regardless of w.

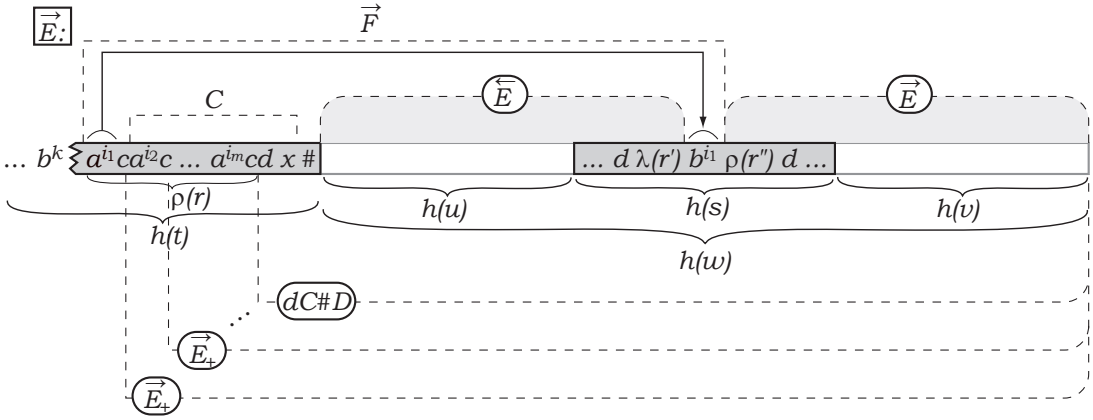

Figure 3: How the nonterminal $\overrightarrow{E}$ describes the parse of usv by a rule $r = A \to \alpha_{i_1} \& \ldots \& \alpha_{i_m}$, encoded as the string $\rho(r)dx\#h_G(w)$.

V. *A string* $a^{i_1}ca^{i_2}c \ldots a^{i_m}cdxh''(t)\#h(w)$, *with* $m \geqslant 0$, $i_1, \ldots, i_m \in \{1, \ldots, |\mathcal{C}|\}$, $x \in \{a, b, c, d\}^*dd \cup \{d\}$, $t \in \Sigma$ *and* $w \in \Sigma^+$, *is in* $L_{G_0}(\overrightarrow{E_0})$ *if and only if* tw *is in* $\bigcap_{j=1}^{m} L_G(\alpha_{i_j})$.

VI. *A string* $h(w)$, *with* $w \in \Sigma^+$, *is in* $L(G_0)$ *if and only if* w *is in* $L(G)$.

Proof. Parts I–IV are proved together within a single inductive argument on the length of strings defined by the grammar G_0.

I. According to the first rule for $\overrightarrow{E}$ (3a), a string $a^{i_1}ca^{i_2}c \ldots a^{i_m}cdx\#h(w)$ belongs to $L_{G_0}(\overrightarrow{E})$ if and only if it is in $L_{G_0}(\overrightarrow{F}\,\overrightarrow{E})$ and its suffix $a^{i_2}c \ldots a^{i_m}cdx\#h(w)$ is in $L_{G_0}(\overrightarrow{E_+})$. The latter condition, by the induction hypothesis (parts I or II), means that w is in $\bigcap_{j=2}^{m} L_G(\alpha_{i_j})$, and it remains to be proved that the former condition is equivalent to $w \in L_G(\alpha_{i_1})$.

According to the rules for $\overrightarrow{F}$ (3e), a string $a^{i_1}ca^{i_2}c \ldots a^{i_m}cdx\#h(w)$ is in $L_{G_0}(\overrightarrow{F}\,\overrightarrow{E})$ if and only if there is a partition $a^{i_1}ca^{i_2}c \ldots a^{i_m}cdx\#h(w) = a^{i_1}cx'\#yb^{i_1}z$, with $y \in L_{G_0}(\overleftarrow{E})$ and $z \in L_{G_0}(\overrightarrow{E})$. This occurrence of a substring b^{i_1} in $h(w)$ must be a part of some expansion of the conjunct α_{i_k} within the encoding $\sigma(\alpha_{i_k})$, which is a part of the image of one of the symbols of w, where it is preceded and followed by a symbol d.

First assume that $\alpha_{i_1} = YsZ$; this is the case illustrated in Figure 3. Then the above claim can be reformulated as follows: the string is in $L_{G_0}(\overrightarrow{F}\,\overrightarrow{E})$ if and only if there is a partition of w as $w = usv$, with $u, v \in \Sigma^*$ and $s \in \Sigma$, a rule r' for Y, a rule r'' for Z, and another partition of $h(s)$ as $h(s) = y'd\lambda(r')b^{i_1}\rho(r'')dz'\#$, so that $h(u)y'd\lambda(r') \in L_{G_0}(\overleftarrow{E})$ and $\rho(r'')dz'\#h(v) \in L_{G_0}(\overrightarrow{E})$. By the induction hypothesis (parts III and I) the latter two conditions are equivalent to $u \in L_G(r')$ and $v \in L_G(r'')$, respectively. This leads to the final reformulation of the claim: the string is in $L_{G_0}(\overrightarrow{F}\,\overrightarrow{E})$ if and only if there is a partition of $w = usv$, with $u, v \in \Sigma^*$ and $s \in \Sigma$, a rule r' for Y and a rule r'' for Z, satisfying $u \in L_G(r')$ and $v \in L_G(r'')$. This is exactly the condition for w to be defined by YsZ, which completes the proof of part I for $\alpha_{i_1} = YsZ$.

If Y or Z or both of them are missing, the proof is the same, with the missing substrings replaced with ε. For instance, for $\alpha_{i_1} = Ys$, the string $a^{i_1}ca^{i_2}c \ldots a^{i_m}cdx\#h(w)$ is in $L_{G_0}(\overrightarrow{F}\,\overrightarrow{E})$ if and only if there is a partition of w as $w = usv$, with $u, v \in \Sigma^*$ and $s \in \Sigma$, a rule r' for Y, and a partition of $h(s)$ as $h(s) = y'd\lambda(r')b^{i_1}dz'\#$, where $h(u)y'd\lambda(r') \in L_{G_0}(\overleftarrow{E})$ and $dz'\#h(v) \in L_{G_0}(\overrightarrow{E})$. Then, $v = \varepsilon$ by part II, and, as in the previous case, $u \in L_G(r')$ by part III. The claim is then finally reformulated as follows: the string is in $L_{G_0}(\overrightarrow{F}\,\overrightarrow{E})$ if and only if there is a partition of $w = us$, with $u \in \Sigma^*$ and $s \in \Sigma$, and a rule r' for Y satisfying $u \in L_G(r')$. This is equivalent to w being in $L_G(Ys)$.

The argument is similarly adapted to the cases of $\alpha_{i_1} = sZ$ and $\alpha_{i_1} = s$. Overall, the string $a^{i_1}ca^{i_2}c \ldots a^{i_m}cdx\#h(w)$ has been proved to be is in $L_{G_0}(\overrightarrow{F}\,\overrightarrow{E})$ if and only if $w \in L_G(\alpha_{i_1})$, and this completes the proof of part I.

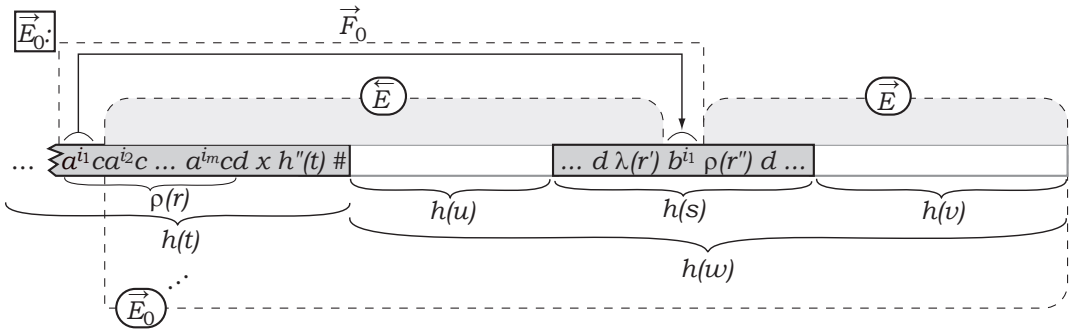

Figure 4: How the nonterminal $\overrightarrow{E_0}$ describes the parse of tw by a rule $r = A \to \alpha_{i_1} \,\&\, \ldots \,\&\, \alpha_{i_m}$, encoded as the string $\rho(r)dxh''_G(t)\#h_G(w)$: case (a).

II. A string $dx\#h(w)$ is in $L_{G_0}(\overrightarrow{E})$ if it is described by the rule (3c), which ensures that w is empty. A string $dx\#h(w)$ always belongs to $L_{G_0}(\overrightarrow{E_+})$, because the rule (3g) describe every such string.

III–IV. These conditions are symmetric to those in parts I–II, and these parts of the proof are established by the same argument.

Each condition has a second form, with a certain prefix of the image of the first symbol t of w omitted, but with $h''(t)$ included in full. This form is necessary to characterize the parse of the entire string according to the rule for the initial symbol of G, because that rule is encoded in $h'(t)$, and then $\overleftarrow{E}$ should describe $h(w)$ with a prefix of $h'(t)$ removed. Since the rules for $\overleftarrow{E}$ and $\overleftarrow{E_+}$ use only the second part of the image of each symbol, the proofs for both forms of these conditions are identical.

V. This part is proved similarly to part I. The proof is by induction on m, the number of conjuncts.

In the base case, $m = 0$, a string $dxh''(t)\#h(w)$ is always in $L_{G_0}(\overrightarrow{E_0})$, and the condition on tw holds trivially. For the induction step, a string $a^{i_1}ca^{i_2}c \ldots a^{i_m}cdxh''(t)\#h(w)$, with $m \geqslant 1$, is in $L_{G_0}(\overrightarrow{E_0})$ if it is described by the first rule for $\overrightarrow{E_0}$ (3n), that is, if it is in $L_{G_0}(\overrightarrow{F_0}\overrightarrow{E})$ and its suffix $a^{i_2}c \ldots a^{i_m}cdxh''(t)\#h(w)$ is in $L_{G_0}(\overrightarrow{E_0})$. The latter condition is equivalent to $tw \in \bigcap_{j=2}^{m} L_G(\alpha_{i_j})$ by the induction hypothesis, and the goal is to prove that $a^{i_1}ca^{i_2}c \ldots a^{i_m}cdxh''(t)\#h(w)$ is in $L_{G_0}(\overrightarrow{F_0}\overrightarrow{E})$ if and only if $tw \in L_G(\alpha_{i_1})$.

The rules for $\overrightarrow{F_0}$ (3o),(3p) assert that $a^{i_1}ca^{i_2}c \ldots a^{i_m}cdxh''(t)\#h(w)$ is in $L_{G_0}(\overrightarrow{F_0}\overrightarrow{E})$ if and only if there is

(a) either a partition $a^{i_1}ca^{i_2}c \ldots a^{i_m}cdxh''(t)\#h(w) = a^{i_1}cx'h''(t)\#yb^{i_1}z$, with $x'h''(t)\#y \in L_{G_0}(\overleftarrow{E})$ and $z \in L_{G_0}(\overrightarrow{E})$,

(b) or a partition $a^{i_1}ca^{i_2}c \ldots a^{i_m}cdxh''(t)\#h(w) = a^{i_1}cydb^{i_1}z$, with $y \in \{a,b,c,d\}^*$ and $z \in L_{G_0}(\overrightarrow{E})$.

In the case (a), this occurrence of b^{i_1} must lie in the image of one of the symbols of w, as one of the expansions of the conjunct α_{i_1}, enclosed between two symbols d. Let $\alpha_{i_1} = YsZ$, as illustrated in Figure 4. Then, this condition is reformulated as follows: there is a partition $w = usv$, a rule r' for Y a rule r'' for Z, and a partition $h(s) = y'd\lambda(r')b^{i_1}\rho(r'')dz'\#$ that satisfy $x'h''(t)\#h(u)y'd\lambda(r') \in L_{G_0}(\overleftarrow{E})$ and $\rho(r'')dz'\#h(v) \in L_{G_0}(\overrightarrow{E})$. The former, by the induction hypothesis (part III) holds if and only if $tu \in L_G(r')$, whereas the latter, by the induction hypothesis (part I) is equivalent to $v \in L_G(r'')$. Then, the claim is reformulated again: there is a partition $w = usv$, a rule r' for Y and a rule r'' for Z, for which $tu \in L_G(r')$ and $v \in L_G(r'')$. This is the condition of membership of tw in $L_{G_0}(YsZ)$.

For a conjunct $\alpha_{i_1} = Ys$, the proof is similar. If the conjunct is $\alpha_{i_1} = sZ$ or $\alpha_{i_1} = s$, the argument also proceeds in the same way, and eventually sets the condition involving the emptiness of the string tu, which is always false. Hence, conjuncts of these two forms are impossible in case (a).

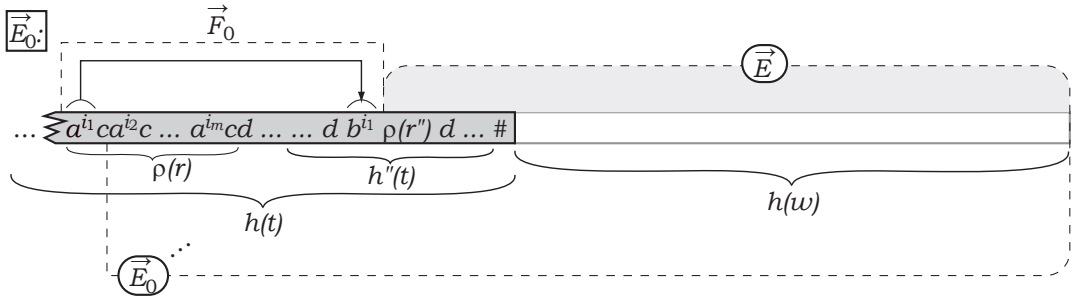

Figure 5: How the nonterminal $\overrightarrow{E_0}$ describes the parse of tw by a rule $r = A \to \alpha_{i_1} \,\&\, \ldots \,\&\, \alpha_{i_m}$, encoded as the string $\rho(r)dxh''_G(t)\#h_G(w)$: case (b).

Turning to the case (b), this instance of b^{i_1} may only belong to the image of the first symbol t. Let $\alpha_{i_1} = tZ$, as shown in Figure 4. Then the condition is equivalent to the following: there is a rule r'' for Z and a partition $h''(t) = x'db^{i_1}\rho(r'')dz'\#$ that satisfy $\rho(r'')dz'\#h(w) \in L_{G_0}(\overrightarrow{E})$. By the induction hypothesis (part I), the latter holds if and only if $w \in L_G(r'')$. This is equivalent to the membership of tw in $L_G(tZ)$.

If the conjunct is $\alpha_{i_1} = t$, the proof is similar, and it results in the condition $w = \varepsilon$. The conjunct cannot be $\alpha_{i_1} = Yt$ or $\alpha_{i_1} = YtZ$, because the form of the string in case (b) rules out such expansions. Overall, a string $a^{i_1}ca^{i_2}c \ldots a^{i_m}cdxh''(t)\#h(w)$ is in $L_{G_0}(\overrightarrow{F_0}\overrightarrow{E})$ if and only if $tw \in L_G(\alpha_{i_1})$, as claimed.

VI. The rules for S_0 (3l) ensure that a string $h(s_1 \ldots s_n)$ is in $L_{G_0}(S_0)$ if and only if it is of the form $x_1dx_2d \ldots x_kdy$, with $k \geqslant 0$, $x_1, \ldots, x_k \in \{a, c\}^+$, and with y belonging to $L_{G_0}(\overrightarrow{E_0})$ and beginning with a. Then, the whole prefix $x_1dx_2d \ldots x_kd$ is contained in $h'(s_1)$, and y must accordingly begin with $\rho(r)d$, for some rule r for S, and end with $h''(s_1)\#h(s_2 \ldots h_n)$. Let the rule r be $S \to \alpha_{i_1} \,\&\, \ldots \,\&\, \alpha_{i_m}$, then $\rho(r) = a^{i_1}c \ldots a^{i_m}c$. This means that $h(s_1 \ldots s_n)$ is in $L_{G_0}(S_0)$ if and only if there exists such a rule r for S, that $h(s_1 \ldots s_n) = xa^{i_1}c \ldots a^{i_m}cdy$, y begins with a and $y \in L_{G_0}(\overrightarrow{E_0})$. By part V, the latter condition holds if and only if $s_1 \ldots s_n \in L_G(r)$. Since r can be any rule for S, it follows that the condition $h(s_1 \ldots s_n) \in L_{G_0}(S_0)$ is equivalent to $s_1 \ldots s_n \in L_G(S)$. □

The last part of Lemma 1 then implies Theorem 1.

Even though the construction presented here proves the theorem, it would still be interesting to improve the argument. Could there be a *simpler* inverse homomorphic representation of conjunctive grammars—that is, involving a substantially smaller grammar for another hardest language? Also, it is worth recalling that Greibach's [12] original proof of the hardest ordinary grammar uses a modification of the Dyck language; could there exist any similar theoretically defined hardest language for conjunctive grammars?

4. Adaptation to Boolean grammars

A similar hardest language theorem can be established for a further extension of conjunctive grammars called *Boolean grammars* [25, 27]. In a Boolean grammar, besides disjunction and conjunction of syntactic conditions, one can also express their *negation* by putting the corresponding operator over any conjunct of any rule.

Definition 3. *A Boolean grammar is a quadruple $G = (\Sigma, N, R, S)$, where*

- *Σ is the alphabet;*
- *N is the set of symbols representing syntactic categories;*
- *R is a finite set of rules of the form*

$$A \to \alpha_1 \,\&\, \ldots \,\&\, \alpha_m \,\&\, \neg\beta_1 \,\&\, \ldots \,\&\, \neg\beta_n \tag{4}$$

with $A \in N$, $m, n \geqslant 0$, $m + n \geqslant 1$ *and* $\alpha_i, \beta_j \in (\Sigma \cup N)^*$;

- $S \in N$ *is the initial symbol.*

A rule (4) is meant to state that every string is representable in the form $\alpha_1, \ldots, \alpha_m$, but not representable in the form $\beta_1, \ldots, \beta_n$, therefore has the property A. This intuitive definition is formalized using *language equations*, that is, by representing a grammar as a system of equations with formal languages as unknowns, and using a solution of this system as the language defined by the grammar. The definition of Boolean grammars exists in two variants: the simple one, given by the author [25], and the improved definition by Kountouriotis et al. [19] based on three-valued logic. Although the simple definition handles some extreme cases of grammars improperly [19], it ultimately defines the same family of languages, and is therefore sufficient in this paper.

Definition 4 (Okhotin [25])**.** *Let* $G = (\Sigma, N, R, S)$ *be a Boolean grammar, and consider the following system of equations in which every symbol* $A \in N$ *is an unknown language over* Σ.

$$A = \bigcup_{\substack{A \to \alpha_1 \,\&\ldots\&\, \alpha_m \,\& \\ \&\, \neg\beta_1 \,\&\ldots\&\, \neg\beta_n \in R}} \left[\bigcap_{i=1}^{m} \alpha_i \;\cap\; \bigcap_{j=1}^{n} \overline{\beta_j} \right] \tag{5}$$

Each symbol $B \in N$ *used in the right-hand side of any equation is a reference to a variable, and each symbol* $a \in \Sigma$ *represents a constant language* $\{a\}$.

Assume that for every integer $\ell \geqslant 0$ *there exists a unique vector of languages* $(\ldots, L_A, \ldots)_{A \in N}$ *with* $L_A \subseteq \Sigma^{\leqslant \ell}$*, such that a substitution of* L_A *for* A*, for each* $A \in N$*, turns every equation (5) into an equality modulo intersection with* $\Sigma^{\leqslant \ell}$*. Then the system is said to have a* strongly unique solution*, and, for every* $A \in N$*, the language* $L_G(A)$ *is defined as* L_A *from the unique solution of this system. The language generated by the grammar is* $L(G) = L_G(S)$.

The odd normal form can be extended to Boolean grammars.

Theorem 2. *For every Boolean grammar there exists and can be effectively constructed a Boolean grammar generating the same language, which is in the* ***odd normal form****, that is, with all rules of the following form.*

$$A \to B_1 a_1 C_1 \,\&\ldots\&\, B_m a_m C_m \,\&\, \neg E_1 d_1 F_1 \,\&\ldots\&\, \neg E_n d_n F_n \qquad (m \geqslant 1,\ n \geqslant 0,\ B_i, C_i, E_j, F_j \in N,\ a_i, d_j \in \Sigma)$$

$$A \to a \qquad (a \in \Sigma)$$

If S *is never used in the right-hand sides of any rules, then two additional types of rules are also allowed.*

$$S \to aA \qquad (a \in \Sigma,\ A \in N)$$

$$S \to \varepsilon$$

Sketch of a proof. The transformation is carried out by the same method as in the case of conjunctive grammars [29].

Assume that the original Boolean grammar $G = (\Sigma, N, R, S)$ is in the *binary normal form* [25], that is, all its rules are of the following form.

$$A \to B_1 C_1 \,\&\ldots\&\, B_m C_m \,\&\, \neg D_1 E_1 \,\&\ldots\&\, \neg D_n E_n \,\&\, \neg\varepsilon \qquad (m \geqslant 1,\ n \geqslant 0,\ B_i, C_i, D_j, E_j \in N) \tag{6a}$$

$$A \to a \qquad (a \in \Sigma) \tag{6b}$$

Construct another Boolean grammar $G' = (\Sigma, N' \cup \{S'\}, R', S')$, in which $N' = \{\, {}_xA_y \mid A \in N,\ x, y \in \Sigma \cup \{\varepsilon\} \,\}$, with the intention that each nonterminal ${}_xA_y$ defines the set of all strings of odd length that, with x and y appended to both sides, would belong to $L_G(A)$.

$$L_{G'}({}_xA_y) = \{\, w \mid xwy \in L_G(A) \,\} \cap \Sigma(\Sigma^2)^*$$

The rules in R' are defined as follows. For every pair $(B, C) \in N \times N$, define

$$\mu_{x,y}(BC) = \{{}_xB_a \cdot a \cdot {}_\varepsilon C_y \mid a \in \Sigma\} \cup \{{}_xB_\varepsilon \cdot a \cdot {}_aC_y \mid a \in \Sigma\} \cup \{{}_xB_\varepsilon \mid y \in L_G(C)\} \cup \{{}_\varepsilon C_y \mid x \in L_G(B)\}.$$

For every rule (6a) and for all $x, y \in \Sigma \cup \{\varepsilon\}$, let $\mu_{x,y}(B_iC_i) = \{\alpha_{i,1}, \dots \alpha_{i,k_i}\}$ for each i-th positive conjunct, and let $\mu_{x,y}(D_jE_j) = \{\beta_{j,1}, \dots \beta_{j,\ell_j}\}$ for each j-th negative conjunct. Then, for any choice $(i_1, \dots, i_m)$ of numbers of positive conjuncts, the new grammar contains the rule

$${}_xA_y \to \alpha_{1,i_1} \,\&\, \dots \,\&\, \alpha_{m,i_m} \,\&\, \neg\beta_{1,1} \,\&\, \dots \,\&\, \neg\beta_{1,\ell_1} \,\&\, \dots \,\&\, \neg\beta_{n,1} \,\&\, \dots \,\&\, \neg\beta_{n,\ell_n},$$

which contains the chosen positive conjuncts and all negative conjuncts. Furthermore, for all $A \in N$, $x, y \in \Sigma^{\leqslant 1}$ and $a \in \Sigma$ satisfying $xay \in L_G(A)$, the new grammar contains the rule

$${}_xA_y \to a$$

Finally, define the rules for the initial symbol as follows:

$$\begin{aligned} &S' \to \varphi && \text{(for all } {}_\varepsilon S_\varepsilon \to \varphi \in R'\text{)}, \\ &S' \to a\ {}_aS_\varepsilon && \text{(for all } a \in \Sigma\text{)}. \end{aligned}$$

Then $L(G') = L(G)$, as desired. □

Using this normal form, one can replicate the construction in Theorem 1 by introducing an extra symbol in the encoding that expresses the negation in G, and by extending the grammar for the "hardest language" to interpret this symbol as negation in G_0.

Theorem 3. *There exists such a language L_0 over the alphabet $\Sigma_0 = \{a, b, c, d, e, \#\}$ described by a Boolean grammar, that for every language L over any alphabet Σ described by some Boolean grammar, there is such a homomorphism $h\colon \Sigma \to \Sigma_0^*$, that $L = h^{-1}(L_0)$ if $\varepsilon \notin L$ and $L = h^{-1}(L_0 \cup \{\varepsilon\})$ if $\varepsilon \in L$.*

The transformation is very similar to the one in Theorem 1. The extra symbol e is used to mark negative conjuncts.

For a Boolean grammar $G = (\Sigma, N, R, S)$ in the odd normal form, let $\mathcal{C} = \{\gamma_1, \gamma_2, \dots, \gamma_{|\mathcal{C}|}\}$, with $\gamma_i \in N\Sigma N \cup \Sigma N \cup N\Sigma \cup \Sigma$, be the set of all positive and negative conjuncts used in the grammar, so that each rule can be written in the following form.

$$A \to \gamma_{i_1} \,\&\, \dots \,\&\, \gamma_{i_m} \,\&\, \neg\gamma_{i_{m+1}} \,\&\, \dots \,\&\, \neg\gamma_{i_n} \tag{7}$$

for some $m \geqslant 1$, $n \geqslant 0$, and $i_1, \dots, i_n \in \{1, \dots, |\mathcal{C}|\}$.

Every such rule (7) again has symmetric left and right representations.

$$\begin{aligned} \lambda(A \to \gamma_{i_1} \,\&\, \dots \,\&\, \gamma_{i_m} \,\&\, \neg\gamma_{i_{m+1}} \,\&\, \dots \,\&\, \neg\gamma_{i_n}) &= ca^n e \dots ca^{i_{m+1}} eca^{i_m} \dots ca^{i_1} \\ \rho(A \to \gamma_{i_1} \,\&\, \dots \,\&\, \gamma_{i_m} \,\&\, \neg\gamma_{i_{m+1}} \,\&\, \dots \,\&\, \neg\gamma_{i_n}) &= a^{i_1} c \dots a^{i_m} cea^{i_{m+1}} c \dots ea^n c \end{aligned}$$

The rest of the transformation is the same as in the case of conjunctive grammars.

The "hardest" language L_0 is then defined by the following Boolean grammar, which extends the grammar given in Section 3 by adding several rules for handling the symbol e. These are new rules for the nonterminals $\overrightarrow{E_+}$, $\overleftarrow{E_+}$ and $\overrightarrow{E_0}$ (one for each) that actually implement the negation, as well as extra rules for B and C that

take into account the extension of the alphabet.

$$
\begin{aligned}
S &\to BdS \mid \overrightarrow{F_0}\overrightarrow{E} \,\&\, Ac\overrightarrow{E_0} \\
A &\to aA \mid a \\
B &\to aB \mid cB \mid eB \mid a \mid c \\
C &\to aC \mid bC \mid cC \mid dC \mid eC \mid \varepsilon \\
D &\to C\#D \mid \varepsilon \\
\overrightarrow{E} &\to \overrightarrow{F}\overrightarrow{E} \,\&\, Ac\overrightarrow{E_+} \mid dC\# \\
\overrightarrow{E_+} &\to \overrightarrow{F}\overrightarrow{E} \,\&\, Ac\overrightarrow{E_+} \mid \neg e\overrightarrow{F}\overrightarrow{E} \,\&\, eAc\overrightarrow{E_+} \mid dC\#D \\
\overrightarrow{F} &\to a\overrightarrow{F}b \mid acC\#\overleftarrow{E}b \\
\overleftarrow{E} &\to \overleftarrow{E}\overleftarrow{F} \,\&\, \overleftarrow{E_+}cA \mid Cd \\
\overleftarrow{E_+} &\to \overleftarrow{E}\overleftarrow{F} \,\&\, \overleftarrow{E_+}cA \mid \neg\overleftarrow{E}\overleftarrow{F}e \,\&\, \overleftarrow{E_+}cAe \mid DCd \\
\overleftarrow{F} &\to b\overleftarrow{F}a \mid b\overrightarrow{E}Cca \\
\overrightarrow{E_0} &\to \overrightarrow{F_0}\overrightarrow{E} \,\&\, Ac\overrightarrow{E_0} \mid \neg e\overrightarrow{F_0}\overrightarrow{E} \,\&\, eAc\overrightarrow{E_0} \mid dC\#D \\
\overrightarrow{F_0} &\to a\overrightarrow{F_0}b \mid acCdb \mid ac\overleftarrow{E}b
\end{aligned}
$$

A formal correctness statement should be very similar to Lemma 1, and its proof can be carried out by the same method.

5. No hardest languages for subclasses of linear grammars

A grammar $G = (\Sigma, N, R, S)$ is *linear*, if all rules in R are of the form $A \to uBv$, with $u, v \in \Sigma^*$ and $B \in N$, or $A \to w$, with $w \in \Sigma^*$. It is known from Boasson and Nivat [6] that there is no hardest language for linear grammars. The following theorem extends their result to subclasses of linear grammars.

Theorem 4. *There is no such language L_0 described by a linear grammar, that for every LL(1) linear grammar G the language it describes could be represented as $h^{-1}(L_0)$, for a suitable homomorphism h.*

The proof is by presenting a sequence of languages L_k, with $k \geqslant 1$, each described by an LL(1) linear grammar, so that for every linear grammar G_0, at least one of these languages L_k is not representable as $h^{-1}(L(G_0))$, for any homomorphism h.

Each language L_k is defined over the alphabet $\Sigma_k = \{a_1, \ldots, a_k, b\}$, as follows.

$$L_k = \{\, a_i w b^n \mid i \in \{1, \ldots, k\},\ w \in \{a_1, \ldots, a_k\}^*,\ n = |w| - |w|_{a_i} \,\} \tag{8}$$

The language L_k is described by a linear grammar $G_k = (\Sigma_k, N_k, R_k, S_k)$, with nonterminal symbols $N_k = \{S_k, A_1, \ldots, A_k\}$, where the rules for S choose i, and then each A_i counts the number of all symbols except a_i, matching them to symbols b on the right.

$$
\begin{aligned}
S_k &\to a_i A_i && (i \in \{1, \ldots, k\}) \\
A_i &\to a_j A_i b && (j \in \{1, \ldots, k\} \setminus \{i\}) \\
A_i &\to a_i A_i
\end{aligned}
$$

This grammar is actually LL(1) and in the Greibach normal form. Korenjak and Hopcroft [18] called such grammars *simple grammars*; in their terminology, the grammar G_k is *simple linear*.

As an exercise, one can prove that every linear grammar for L_k must have at least k nonterminal symbols. The below Lemma 2 presents a stronger condition: that if L_k is an inverse homomorphic image of any language L_0, then every linear grammar for that language L_0 must have at least k nonterminals. For ease of proof, the lemma assumes that the grammar is in a normal form (it could be proved without this restriction, but that is not necessary here).

Lemma 2. *Let $G = (\Sigma_0, N, R, S)$ be a linear grammar with all rules of the form $A \to bC$, $A \to Bc$ or $A \to \varepsilon$, let $k \geqslant 1$ and let $h\colon \Sigma_k^* \to \Sigma_0^*$ be a homomorphism, so that $h^{-1}(L(G)) = L_k$. Then, $|N| \geqslant k$.*

Proof. For each symbol a_t, with $t \in \{1, \ldots, k\}$, consider the following string.

$$w_t = a_t^{|N|\cdot|h(b)|} a_{t+1}^{|N|\cdot|h(a_t)|} b^{|N|\cdot|h(a_t)|}$$

(if $t = k$, then a_{t+1} denotes a_1; in general, a_{t+1} could be replaced with any symbol $a_{t'}$ different from a_t) The string w_t belongs to L_k, and hence its image $h(w_t)$ is in $L(G)$. The proof of the lemma proceeds by showing that, for each string $h(w_t)$, its parse tree can be pumped at some recurring nonterminal $B_t \in N$. Next, it will be shown that these nonterminals must be distinct for different t, for otherwise G would generate some other strings with pre-images not in L_k.

The natural way of testing that w_t is in L_k is to skip all symbols a_t in the beginning, and then match each symbol a_{t+1} to the corresponding symbol b in the last part of the string (this is what the above grammar G_k does). The key point of the proof is that any parse tree of $h(w_t)$ according to G cannot much deviate from this schedule of testing: it is bound to generate some blocks $h(a_t)$ on the left without matching them to anything on the right. This is where one can find a recurring symbol $B_t \in N$, and then pump the string to generate an unbounded number of blocks $h(a_t)$.

Claim 1. *For each $t \in \{1, \ldots, k\}$, there is such a nonterminal $B \in N$, that the following two properties hold.*

i. *Some parse tree of $h(w_t)$ contains a node labelled with B, which spans over a substring $h(a_t)^r h(a_{t+1}^{|N|\cdot|h(a_t)|})z$, where $1 \leqslant r < |N| \cdot |h(b)|$ and z is any prefix of $h(b^{|N|\cdot|h(a_t)|})$.*
ii. *There is a derivation $B \Longrightarrow^* h(a_t)^p B$, for some $p \geqslant 1$.*

Proof. Fix any parse tree of $h(w_t)$ and consider its initial segment, from the root to the the lowest node with its subtree properly containing the whole middle part $h(a_{t+1}^{|N|\cdot|h(a_t)|})$. The substring preceding the middle part and the substring following the middle part are of length $|N| \cdot |h(a_t)| \cdot |h(b)|$ each, and therefore, this segment contains at least $|N| \cdot |h(a_t)| \cdot |h(b)| + 1$ nodes.

For each node in the initial segment, the substring it encompasses begins in one of the images $h(a_t)$ and ends in one of the images $h(b)$. The position of this node relative to the images of symbols is characterized by a triple (A, i, j), where $A \in N$ is the node's label, and the numbers $i \in \{0, \ldots, h(a_t) - 1\}$ and $j \in \{0, \ldots, h(b) - 1\}$ tell the position of the first and the last symbols of this substring in the image $h(a_t)$ and in the image $h(b)$, respectively. As the initial segment contains at least $|N| \cdot |h(a_t)| \cdot |h(b)| + 1$ nodes, some triple has to appear twice in this segment. That triple is denoted by (A, i, j) and illustrated in Figure 6(left).

Between the two instances of (A, i, j) in the initial segment, there is a string of leaves u spawned off to the left, and another string of leaves v spawned off to the right (in terms of string rewriting, A is rewritten to uAv). It is known that u begins and ends at offset i within some instances of $h(a_t)$, and therefore its length is $p \cdot |h(a_t)|$, for some $p \geqslant 0$. Similarly, v begins and ends at offset j within some instances of $h(b)$, and its length is $q \cdot |h(b)|$, with $q \geqslant 0$. If the segment from A to A is removed, what remains is a valid parse tree of the following shorter string.

$$h\Big(\underbrace{a_t^{|N|\cdot|h(b)|-p} a_{t+1}^{|N|\cdot|h(a_t)|} b^{|N|\cdot|h(a_t)|-q}}_{w'_t} \Big)$$

Since the encoding is in $L(G)$, its pre-image w'_t must be is L_k. This can only happen if $q = 0$, and, accordingly, $v = \varepsilon$.

Let $h(a_t) = xy$ be the partition of $h(a_t)$ defined by the offset $i = |x|$. So far, it has been proved that there are two such instances of A in the initial segment, that between them, a string of leaves $yh(a_t)^{p-1}x$ is spawned off to the left and no leaves are spawned off to the right, as depicted in Figure 6(right); in terms of string rewriting, this is a derivation $yh(a_t)^{p-1}xA$ from A. If $i = 0$, this proves the claim, with $B = A$.

Assume that $i \neq 0$ and accordingly $x, y \neq \varepsilon$. Consider the segment between the two instances of A, and let $B \in N$ be the first nonterminal symbol on this segment that is aligned with the grid, as marked in

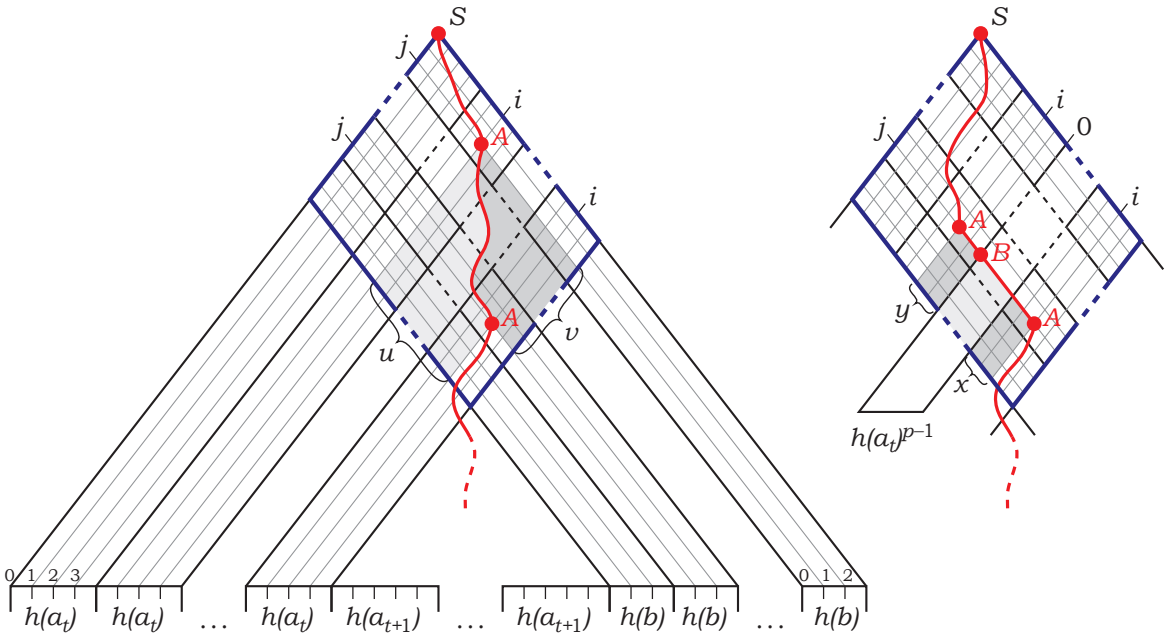

Figure 6: Initial segment of the parse tree of $h(w_t) = h(a_t)^{|N|\cdot|h(b)|}h(a_{t+1})^{|N|\cdot|h(a_t)|}h(b)^{|N|\cdot|h(a_t)|}$: (left) a repeated triple (A, i, j); (right) the only possible form of this path, and the symbol $B \in N$.

Figure 6(right). In other words, $A \Longrightarrow^* yB$ and $B \Longrightarrow^* h(a_t)^{p-1}xA$. These two fragments of the parse tree can be combined in the reverse order, so that $B \Longrightarrow^* h(a_t)^p B$, as claimed. □

Resuming the proof of Lemma 2, suppose, for the sake of a contradiction, that $|N| < k$. Then, Claim 1 is bound to assign the same nonterminal B to two different values $t, t' \in \{1, \ldots, k\}$, with $t \neq t'$. In particular, by Claim 1(i) for t, there is a parse tree of $h(w_t)$ with a node B spanning over a substring $h(a_t^r a_{t+1}^{|N|\cdot|h(a_t)|})z$, where $r \geqslant 1$, and z is a prefix of $h(b^{|N|\cdot|h(a_t)|})$. By Claim 1(ii) for t', there is a tree fragment from B to B that spawns a string $h(a_{t'})^p$, with $p \geqslant 1$, off to the right. Inserting the latter tree fragment into the former tree, at the node B, yields a valid parse tree of the following string.

$$h\Big(\underbrace{a_t^{|N|\cdot|h(b)|-r} \overbrace{\boldsymbol{a_{t'}^{p}}}^{\text{inserted}} a_t^r a_{t+1}^{|N|\cdot|h(a_t)|} b^{|N|\cdot|h(a_t)|}}_{w''} \Big)$$

Its pre-image w'' begins with the symbol a_t, because $r < |N| \cdot |h(b)|$. However, as $p \geqslant 1$, the number of symbols other than a_t is different from the number of symbols b, and therefore this string does not belong to the language L_k. The contradiction obtained proves that the grammar must contain at least k distinct nonterminal. □

With Lemma 2 established, the theorem is proved as follows.

Proof of Theorem 4. Suppose that there exists a language L_0 with the desired property. Then the linear grammar describing L_0 can be transformed to a normal form grammar $G_0 = (\Sigma_0, N, R, S)$, with all rules in R of the form $A \to bC$, $A \to Bc$ or $A \to \varepsilon$, with $b, c \in N$ and $B, C \in N$.

By the assumption, for every language L_k from the above sequence (8), there exists a homomorphism $h\colon \Sigma_k^* \to \Sigma_0^*$ satisfying $h^{-1}(L(G_0)) = L_k$. In particular, such a representation should exist for $k = |N| + 1$. Then, Lemma 2 asserts that the set N contains at least $|N| + 1$ elements, which is a contradiction. □

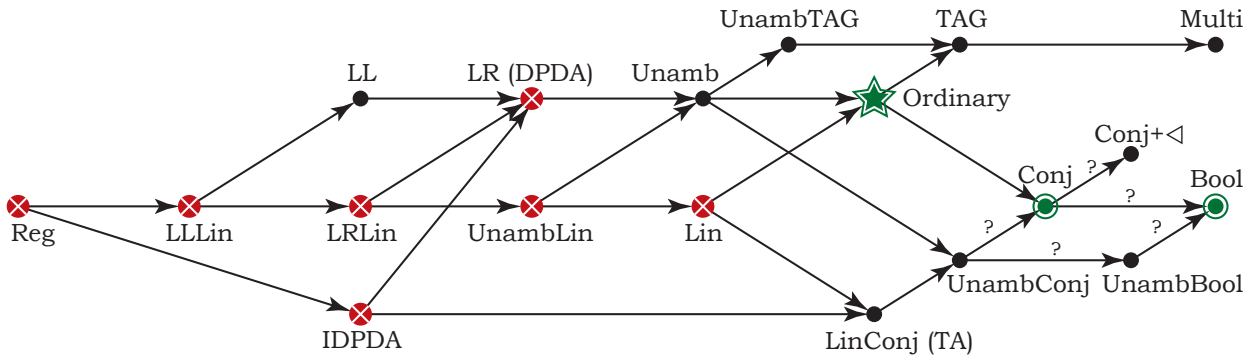

Figure 7: The known hierarchy of formal grammars; the families depicted are those defined by finite automata (*Reg*), by (linear) LL(k) grammars (*LLLin*, *LL*), by (linear) LR(1) grammars (*LRLin*, *LR*), by (linear) unambiguous grammars (*UnambLin*, *Unamb*), by ordinary ("context-free") grammars (*Ordinary*), by linear grammars (*Lin*), by (unambiguous) tree-adjoining grammars (*UnambTAG*, *TAG*), by multi-component grammars (*Multi*), by input-driven pushdown automata (*IDPDA*), by (linear) conjunctive grammars (*LinConj*, *Conj*), by unambiguous conjunctive and Boolean grammars (*UnambConj*, *UnambBool*), by Boolean grammars (*Bool*) and by grammars with context operators (*Conj* + ◁). Arrows represent inclusions, question marks indicate inclusions not known to be proper. Families with a known hardest language are encircled, families with no hardest language are crossed, families not closed under inverse homomorphisms are shown in grey.

6. Summary of results and future work

Every family of formal languages, including the families not closed under inverse homomorphisms, can potentially contain a hardest language with respect to homomorphic reductions. The question is, which families have it?

The current state of knowledge on this question is illustrated in Figure 7. The three families marked by encircled nodes each have a hardest language: these are ordinary [12], conjunctive (Section 3) and Boolean grammars (Section 4). The seven families marked by crossed out nodes cannot have a hardest language. These are the regular languages [8], as well as the families defined by LR grammars [13], linear grammars [6], and the subclasses of the latter: unambiguous linear, LR linear and LL linear grammars (Section 5). The family of input-driven pushdown automata (IDPDA), also known as visibly pushdown automata [2, 22, 30], cannot have a hardest language either: the argument for regular languages by Čulík and Maurer [8] based on the number of states in finite automata equally applies to this model. For the remaining families, the question is open.

For the *LL(k) grammars*, that is, for ordinary grammars that can be parsed by a deterministic top-down parser with k lookahead symbols, the existence of a hardest language deserves a separate investigation. The answer is hard to predict. It might happen that the parsing table of an LL(k) parser could be encoded in the homomorphic images of symbols, so that "a hardest LL parser" would simulate its operation, although it is more likely that some obstacle would make this impossible.

Next, there are the unambiguous subfamilies of ordinary, conjunctive and Boolean grammars. Ambiguity is an elusive property that is not evident from the description of a grammar. If one can encode a grammar in homomorphic images and have a "hardest" grammar simulate these encodings, then how could one prevent encoding an ambiguous grammar, and thus driving the hardest grammar into ambiguity while simulating it? For this reason, most likely, there is no hardest language for the unambiguous subfamily of ordinary grammars (*Unamb*); however, as Boasson and Nivat [6] rightfully remarked, proving that "seems to be a hard problem". The case of unambiguous conjunctive (*UnambConj*) and unambiguous Boolean grammars (*UnambBool*) would be even harder to settle, because too little is known about these grammars; so far there are only positive results on their expressive power, and it is not even known whether these grammars are less powerful than Boolean grammars of the general form.

For linear conjunctive grammars (*LinConj*), it is known that these grammars can describe a P-complete language, which is thus the hardest under logarithmic-space reductions; this is a result similar to those by Sudborough [32] and by Holzer and Lange [15]. But is there a hardest language in the sense of Greibach [12],

that is, under homomorphic reductions? Linear conjunctive grammars are notable for being equivalent to the simplest class of one-dimensional cellular automata [24], which adds interest to this question. The non-trivial positive and negative results by Terrier [33, 34] on the expressive power of these cellular automata suggest that it it is likely difficult to determine whether a hardest language exists.

Another noteworthy extension of ordinary formal grammars are the *multi-component grammars* of Vijay-Shanker et al. [35] and of Seki et al. [31]. In these grammars, a nonterminal may deal with several substrings at once, and those substrings may be arbitrarily concatenated in the rules. The maximum number of substrings handled at once is known as the *dimension* of the grammar, and there is an infinite hierarchy of representable languages with respect to this dimension. Since the language family described by grammars of every fixed dimension d is closed under inverse homomorphisms [31], there cannot be a single hardest language for all dimensions. However, it is possible that there is a hardest language for each dimension, and checking that is left for future studies. In particular, multi-component grammars of dimension 2, with a certain well-nestedness restriction, are known as *tree-adjoining grammars* [17], and this family might have a hardest language with respect to homomorphic reductions.

A similar question can be asked about *grammars with context operators*, recently defined by Barash and Okhotin [4, 5], which further extend conjunctive grammars and implement the notion of a rule applicable in a context. It is not yet known whether they are closed under inverse homomorphisms, and establishing an analogue of the odd normal form might be a challenging task. Nevertheless, the existence of a hardest language appears likely.

One more suggested research direction is to investigate new variants of another fundamental characterization of formal grammars: the *Chomsky–Schützenberger theorem*, which states that a language L is described by an ordinary grammar if and only if it is representable as $L = h(D_k \cap R)$, for some Dyck language D_k regular language R and homomorphism h. As recently shown by the author [26], if the Dyck language is expanded with neutral symbols, or, alternatively, with a certain insignificant restriction assumed on L, it is sufficient to use a *non-erasing* homomorphism. Recently, Crespi-Reghizzi and San Pietro [7] have further strengthened the theorem by showing that k depends only on the size of the alphabet and not on the grammar. Turning to other grammar families, a variant of the Chomsky–Schützenberger theorem for multi-component grammars was proved by Yoshinaka et al. [36], and for indexed grammars, Fratani and Voundy [10] obtained a similar characterization. The question is, could there be any related characterizations for the remaining families of formal grammars shown in Figure 7?

References

[1] T. Aizikowitz, M. Kaminski, "LR(0) conjunctive grammars and deterministic synchronized alternating pushdown automata", *Journal of Computer and System Sciences*, 82:8 (2016), 1329–1359.

[2] R. Alur, P. Madhusudan, "Visibly pushdown languages", *ACM Symposium on Theory of Computing* (STOC 2004, Chicago, USA, 13–16 June 2004), 202–211.

[3] J.-M. Autebert, "Non-principalité du cylindre des langages à compteur", *Mathematical Systems Theory*, 11:1 (1977), 157–167.

[4] M. Barash, A. Okhotin, "An extension of context-free grammars with one-sided context specifications", *Information and Computation*, 237 (2014), 268–293.

[5] M. Barash, A. Okhotin, "Two-sided context specifications in formal grammars", *Theoretical Computer Science*, 591 (2015), 134–153.

[6] L. Boasson, M. Nivat, "Le cylindre des langages linéaires", *Mathematical Systems Theory*, 11 (1977), 147–155.

[7] S. Crespi-Reghizzi, P. San Pietro, "The missing case in Chomsky-Schützenberger theorem", *Language and Automata Theory and Applications* (LATA 2016, Prague, Czech Republic, 14–18 March 2016), LNCS 9618, 345–358.

[8] K. Čulík II, H. A. Maurer, "On simple representations of language families", *RAIRO Informatique Théorique et Applications*, 13:3 (1979), 241–250.

[9] R. W. Floyd, "Syntactic analysis and operator precedence", *Journal of the ACM*, 10:3 (1963), 316–333.

[10] S. Fratani, E. M. Voundy, "Homomorphic characterizations of indexed languages", *Language and Automata Theory and Applications* (LATA 2016, Prague, Czech Republic, 14–18 March 2016), LNCS 9618, 359–370.

[11] S. A. Greibach, "A new normal-form theorem for context-free phrase structure grammars", *Journal of the ACM*, 12 (1965), 42–52.

[12] S. A. Greibach, "The hardest context-free language", *SIAM Journal on Computing*, 2:4 (1973), 304–310.

[13] S. A. Greibach, "Jump PDA's and hierarchies of deterministic context-free languages", *SIAM Journal on Computing*, 3:2 (1974), 111–127.

[14] T. Harju, J. Karhumäki, H. C. M. Kleijn, "On morphic generation of regular languages", *Discrete Applied Mathematics*, 15 (1986), 55–60.
[15] M. Holzer, K.-J. Lange, "On the complexities of linear LL(1) and LR(1) grammars", *Fundamentals of Computation Theory* (FCT 1993, Hungary, August 23–27, 1993), LNCS 710, 299–308.
[16] A. Jeż, A. Okhotin, "Unambiguous conjunctive grammars over a one-symbol alphabet", *Theoretical Computer Science*, 665 (2017), 13–39.
[17] A. K. Joshi, L. S. Levy, M. Takahashi, "Tree adjunct grammars", *Journal of Computer and System Sciences*, 10:1 (1975), 136–163.
[18] A. J. Korenjak, J. E. Hopcroft, "Simple deterministic languages", *7th Annual Symposium on Switching and Automata Theory* (SWAT 1966, Berkeley, California, USA, 23–25 October 1966), IEEE Computer Society, 36–46.
[19] V. Kountouriotis, Ch. Nomikos, P. Rondogiannis, "Well-founded semantics for Boolean grammars", *Information and Computation*, 207:9 (2009), 945–967.
[20] S. Kuznetsov, "Conjunctive grammars in Greibach normal form and the Lambek calculus with additive connectives", *Formal Grammar* (FG 2013, Düsseldorf, Germany, August 2013), LNCS 8036, 242–249.
[21] T. Lehtinen, A. Okhotin, "Boolean grammars and GSM mappings", *International Journal of Foundations of Computer Science*, 21:5 (2010), 799–815.
[22] K. Mehlhorn, "Pebbling mountain ranges and its application to DCFL-recognition", *Automata, Languages and Programming* (ICALP 1980, Noordweijkerhout, The Netherlands, 14–18 July 1980), LNCS 85, 422-435.
[23] A. Okhotin, "Conjunctive grammars", *Journal of Automata, Languages and Combinatorics*, 6:4 (2001), 519–535.
[24] A. Okhotin, "On the equivalence of linear conjunctive grammars to trellis automata", *RAIRO Informatique Théorique et Applications*, 38:1 (2004), 69–88.
[25] A. Okhotin, "Boolean grammars", *Information and Computation*, 194:1 (2004), 19–48.
[26] A. Okhotin, "Non-erasing variants of the Chomsky–Schützenberger theorem", *Developments in Language Theory* (DLT 2012, Taipei, Taiwan, 14–17 August 2012), LNCS 7410, 121–129.
[27] A. Okhotin, "Conjunctive and Boolean grammars: the true general case of the context-free grammars", *Computer Science Review*, 9 (2013), 27–59.
[28] A. Okhotin, "Parsing by matrix multiplication generalized to Boolean grammars", *Theoretical Computer Science*, 516 (2014), 101–120.
[29] A. Okhotin, C. Reitwießner, "Conjunctive grammars with restricted disjunction", *Theoretical Computer Science*, 411:26–28 (2010), 2559–2571.
[30] A. Okhotin, K. Salomaa, "Complexity of input-driven pushdown automata", *SIGACT News*, 45:2 (2014), 47–67.
[31] H. Seki, T. Matsumura, M. Fujii, T. Kasami, "On multiple context-free grammars", *Theoretical Computer Science*, 88:2 (1991), 191–229.
[32] I. H. Sudborough, "A note on tape-bounded complexity classes and linear context-free languages", *Journal of the ACM*, 22:4 (1975), 499–500.
[33] V. Terrier, "On real-time one-way cellular array", *Theoretical Computer Science*, 141:1–2 (1995), 331–335.
[34] V. Terrier, "Recognition of linear-slender context-free languages by real time one-way cellular automata", *Cellular Automata and Discrete Complex Systems* (*AUTOMATA 2015*, Turku, Finland, 8–10 June 2015), LNCS 9099, 251–262.
[35] K. Vijay-Shanker, D. J. Weir, A. K. Joshi, "Characterizing structural descriptions produced by various grammatical formalisms", *25th Annual Meeting of the Association for Computational Linguistics* (ACL 1987), 104–111.
[36] R. Yoshinaka, Y. Kaji, H. Seki, "Chomsky-Schützenberger-type characterization of multiple context-free languages", *Language and Automata Theory and Applications* (LATA 2010, Trier, Germany, May 24–28, 2010), LNCS 6031, 596–607.