NON-TERMINAL COMPLEXITY OF SIMPLE SEMI-CONDITIONAL GRAMMARS

HENNING FERNAU, SANJAY JAIN, LINUS RICHTER, FRANK STEPHAN, AND DAN TURETSKY

Abstract. We study the complexity of simple semi-conditional grammars (SSCGs) in terms of the number of their terminals and non-terminals. We show that SSCGs with three non-terminals can generate all RE languages; two non-terminals suffice for linear languages; and one suffices for unary regular languages. We also determine both upper bounds and fundamental limitations of SSCGs: while certain one-non-terminal SSCGs can generate non-context-free languages, every language generated by a one-non-terminal SSCG is in non-deterministic linear space. We also prove that there exists a regular language of alphabet size ≥ 3 which cannot be generated by any one-non-terminal SSCG. Finally, we prove that membership testing for SSCGs of two non-terminals is already **NP**-hard.

1. Introduction

1.1. **Background.** We study the descriptive complexity of certain regulated rewriting grammar formalisms which have been previously proven to be computationally complete. More precisely, we investigate the *non-terminal complexity* of such grammars. Our motivating question is:

> What is the smallest number of non-terminals sufficient to describe each recursively enumerable (RE) language?

For the very classical rewriting mechanisms—such as graph-controlled, programmed, or matrix grammars (all with appearance checking)—this question was studied in a sequence of papers, finally bringing the number of required non-terminals down to 2, 3, and 3, respectively [FFOR07]. There, it was also shown that one non-terminal is insufficient to produce all RE languages, but sufficient to obtain all linear languages with graph control. Conversely, it was also remarked at the end of [FFOR07] that Example 4.1.1 in [DP89] shows that this dramatic collapse of the non-terminal hierarchy does not occur with random context grammars with appearance checking; in contrast, for these grammars, the family of languages generable with k non-terminals is a proper subclass of the family of languages generable with $k+1$ non-terminals. Hence, it is interesting to study related grammatical mechanisms. This is why we study so-called *simple semi-conditional grammars* in this paper.

Generalising conditional grammars due to Kelemen in 1984 [Kel84], Păun introduced *semi-conditional grammars* (SCGs) one year later [Pău85]. Here, each rule is associated with two strings called the *permitting* and *forbidden string*. A rule can be applied to a sentential form w only if w contains the permitting string (i.e., positive context) and

Date: June 11, 2026.

Sanjay Jain was supported by NUS Provost chair grant E-252-00-0021-01. Linus Richter was fully supported by Singapore Ministry of Education grant MOE-000538-01. Additionally, Sanjay Jain and Frank Stephan were partially supported by Singapore Ministry of Education grant MOE-000538-01. Dan Turetsky was supported by the Royal Society of New Zealand.

does not contain the forbidden string (i.e., negative context) as a subword. A semi-conditional grammar is termed *simple* (denoted SSCG) if for each rule at most either the permitting string or the forbidden string is present [MG94]. Notice that SSCGs generalise Kelemen's conditional grammars, too. Quite some research focused on the so-called *degree* of (simple) semi-conditional grammars, which is the pair formed by upper-bounds on the length of the permitting and forbidden strings. Based on these bounds, further bounds on other descriptive complexity measures were investigated [FKOR21]. For instance, the number of conditional rules have been of interest in [Mv02].

In this paper, our main focus is the number of non-terminals of these grammars. This measure of descriptive complexity has been investigated before with semi-conditional grammars, but only in passing. In [FKO18], it was shown that five non-terminals suffice in the case of SCGs of degree $(4, 1)$ to describe every RE language, while in [FKOR21], it was proven that six non-terminals are enough for this purpose for SSCGs. What can be achieved if only one, two, or three non-terminals are allowed, disregarding other measures of descriptive complexity?

1.2. **Our Theorems and the Structure of This Paper.** We show that every RE language can be generated by an SSCG with three non-terminals (Theorem 7.1). Furthermore, it is shown that every linear language can be generated by an SSCG using two non-terminals (Theorem 3.1), and every unary regular (i.e. unary context-free) language can be generated using one non-terminal (Theorem 4.1). However, there are regular languages over alphabets of size 3 which cannot be generated by any SSCG with only one non-terminal (Theorem 4.2). It is open at present if every regular language over a binary alphabet can be generated by an SSCG with two non-terminals.

We also show that there are SSCGs with one non-terminal which generate a non-context free language (Theorem 5.1). However, a language generated by any SSCG with one non-terminal is in non-deterministic linear space (Theorem 5.2). Finally, we give a complexity characterisation for membership testing: for SSCGs with only 2 non-terminals, membership testing is already **NP**-hard (Theorem 6.1).

A summary of our results can be found in Table 1, where we refer to the Chomsky Hierarchy explicitly. We refer the reader to the Complexity Zoo [Aar25] for further details on complexity classes and their relationships. For a survey on regulated rewriting, we refer to the textbook of Dassow and Păun [DP89].

2. Formal Definitions and Conventions

We assume a basic understanding of formal language theory as provided in standard references (e.g. [HU79, Sal73]). In particular, we assume familiarity with the language classes of the Chomsky hierarchy: regular, (linear,) context-free, context-sensitive, (recursive,) and recursively enumerable, listed in ascending order of complexity. In parentheses, we have added well-known intermediate classes.

All alphabets in this paper are finite.

We now define (simple) semi-conditional grammars. We fix some notation: if w is a word formed with letters from the alphabet Σ, written as $w \in \Sigma^*$, then $y \in \Sigma^*$ is a **subword** (sometimes also called a factor or infix) of w if there are words $x, z \in \Sigma^*$ such that $w = xyz$. The set of subwords of w is written as $\mathsf{sub}(w)$. As is customary, ε denotes the empty word, and we define $\Sigma^+ = \Sigma^* - \{\varepsilon\}$.

Definition 2.1. *A* **semi-conditional grammar** *is a quadruple* $G = (V_N, V_T, P, S)$, *where*

Shortform	Name	Condition on SSCG	Condition on Usual Grammar	Class
unary REG	Regular with unary alphabet	SBS/EQT: $\Delta = \{S\}$ (Theorem 4.1)	see below	EQT: Unary Context-Free
REG, CH3	Regular	SBS: $\Delta = \{S, T\}$ (Theorem 4.2)	$A \to aB \vert b$	EQT: SPACE($O(1)$)
LIN	Linear	SBS: $\Delta = \{S, T\}$ and $A \to vaw$ (Theorem 3.1)	$A \to aB \vert Ba \vert a$	SBS: NLOGSPACE
CH2, CF	Context-Free	SBS: $\Delta = \{S, T, U\}$ (Theorems 4.2 + 7.1)	$A \to BC \vert a$	SBS: POLYLOGSPACE SBS: P, Nick's Class
CS, CH1	Context-Sensitive	EQT: ε-free	$v \to w$ with $\vert v\vert \leq \vert w\vert$	EQT: NLINSPACE
—	decidable, recursive	—	—	characteristic function total recursive
RE	recursively enumerable	EQT: $\Delta = \{S, T, U\}$ (Theorem 7.1)	no restriction	ranges of enumeration procedures

TABLE 1. EQT means "equal to," SBS means "is contained in," SPS means "strictly contains." For SSCGs, Δ denotes the set of distinct non-terminals.

- V_N *is the non-terminal alphabet,*
- V_T *is the terminal alphabet,* $V_T \cap V_N = \emptyset$*,*
- $V = V_N \cup V_T$ *is the total alphabet,*
- $S \in V_N$ *is the starting symbol,*
- P *is a finite set of (labeled) productions of the form* $\ell\colon (A \to x, \alpha, \beta)$ *with*
  - $A \in V_N$
  - $x \in V^*$
  - $\alpha, \beta \in V^+ \cup \{0\}$*, where* $0 \notin V$ *is a special symbol, intuitively meaning that the condition is missing*
  - ℓ *is the label of the production*

The production $\ell\colon (A \to x, \alpha, \beta)$ can be applied to a word $u \in V^*V_NV^*$ if and only if

$$A \in \mathsf{sub}(u) \text{ and } (\alpha \in \mathsf{sub}(u) \text{ or } \alpha = 0) \text{ and } (\beta \notin \mathsf{sub}(u) \text{ or } \beta = 0).$$

Under these conditions, $u = u_1Au_2$ will be transformed into $v = u_1xu_2$, which is denoted by $u \Rightarrow_\ell v$. In such a case, we call α **permitting** and β **forbidden**. When there is no risk of confusion, we avoid mentioning the label and simply write $u \Rightarrow v$. The **language** of G is defined as

$$L(G) = \{\, w \in V_T^*\colon S \Rightarrow^* w \,\}$$

where $\Rightarrow^*$ denotes the reflexive and transitive closure of $\Rightarrow$.

Definition 2.2. *A semi-conditional grammar G is called* **simple** *(denoted by SSCG) if for every production rule $\ell\colon (A \to x, \alpha, \beta) \in P$ we have $\alpha = 0$ or $\beta = 0$.*

A production rule $\ell\colon (A \to x, \alpha, \beta)$ in an SSCG is called a **presence rule** if $\beta = 0$, and an **absence rule** if $\alpha = 0$. For ease of notation, for presence/absence rules, we call the corresponding strings α and β **presence/absence strings**. A rule is **applicable** to some sentential form if the corresponding condition of said rule is satisfied by the sentential form.

We denote non-terminals by S, T, U or S, T or S, depending on the number of terminals required. Terminals are indexed by natural numbers; for the k-element language we hence use $1, \ldots, k$, as well as 0 which, we recall, is a special symbol.

Further, we fix the following:

- Terminal symbols are denoted by letters from the beginning of the alphabet:

  $$a, b, c, \ldots$$

- Words (terminals and non-terminals) are denoted by letters from the end of the alphabet:

  $$u, v, w, x, y, z$$

- Non-terminals are denoted by capital letters from the beginning of the alphabet:

  $$A, B, C, \ldots$$

  In simulated languages, we index the set of non-terminals by the natural numbers:

  $$Z_1, Z_2, Z_3, \ldots$$

  In such cases, we decide that Z_0 is the starting symbol.

3. Two Non-Terminals Suffice for Linear Languages

Our first result shows that any linear language, and thus any regular language, can be generated by an SSCG with two non-terminals.

Theorem 3.1. *Linear languages can be generated by simple semi-conditional grammars with non-terminals S, T only.*

Proof. Suppose the linear grammar G for a language L has non-terminals $Z_0, Z_1, Z_2, \ldots$, where Z_0 is the starting symbol in G. By an intermediate application of the Chomsky Normal Form Theorem, all rules in G may be assumed to be of the form $Z_i \to Z_j a$ or $Z_i \to bZ_j$ or $Z_i \to c$, where $a, b, c \neq \varepsilon$ are members of the alphabet.

Fix two non-terminals S, T, where S denotes the starting symbol in our SSCG. The following table describes the SSCG for the language L. By Definition 2.2, it suffices to define presence and absence rules:

Rule	Condition on Subword	Condition on G
$S \to \varepsilon$	T absent	$\varepsilon \in L$
$S \to a$	T absent	$Z_0 \to a$
$S \to aT^iST^i$	T absent	$Z_0 \to aZ_i$
$S \to T^iST^ia$	T absent	$Z_0 \to Z_ia$
$S \to aT^jST^j$	bT^iST^i present	$Z_i \to aZ_j$
$S \to aT^jST^j$	T^iST^ib present	$Z_i \to aZ_j$
$S \to T^jST^ja$	bT^iST^i present	$Z_i \to Z_ja$
$S \to T^jST^ja$	T^iST^ib present	$Z_i \to Z_ja$
$S \to a$	bT^iST^i present	$Z_i \to a$
$S \to a$	T^iST^ib present	$Z_i \to a$
$T \to \varepsilon$	S absent	after termination

Observe that since the alphabet of G is finite, the table above is also finite. Hence, the induced SSCG is indeed well-defined.

Verification. Let $u \in L$ be given by a derivation sequence $S \Rightarrow^* u$. Suppose the derivation has steps $u_1, \ldots, u_k$. By definition of linear grammars, each u_i for $i < k$ contains exactly one Z_{n_i}, where $u_1 = Z_{n_1} = Z_0$ and $u_k = u$.

Let $v_1, \ldots, v_k$ be a parallel derivation operating on our SSCG, where $v_1 = S$, and v_{i+1} is defined in the obvious fashion to replicate the operation in u_{i+1}. Observe that, except possibly at the start, each copy of T^iST^i has a character from Σ either to its left or to its right, but not both. For each i, let $\overline{u}_i, \overline{v}_i \in \Sigma^*$ denote the words obtained by deleting all instances of Z_j for all j in $\overline{u}_i$, and all instances of S and T in $\overline{v}_i$. It is easily seen by induction that $\overline{u}_i = \overline{v}_i$ for all $i \leq k$. The other direction is obtained analogously, mutatis mutandis. □

Theorem 3.1 is optimal: in Theorem 4.2 below we show that there exists a regular (and hence linear) language which cannot be generated by an SSCG with only one non-terminal.

4. Regular Languages

In this section, we consider the strength of SSCGs with one non-terminal for generating regular languages. First, we show that every unary regular language (and thus

every unary context-free language) can be generated by an SSCG with one non-terminal. In Theorem 4.2, we show that there are regular languages which cannot be generated by SSCGs having only one non-terminal.

Theorem 4.1. *Unary context-free languages can be recognised by simple semi-conditional grammars with one non-terminal.*

Proof. Consider any context-free language L over a unary alphabet $\{1\}$, and recall that such languages are always regular [GR62]. Further, note that, by Parikh's Theorem, there exists an h such that $L = A \cup B \cdot \left(1^h\right)^*$, where A and B are finite sets and $\varepsilon \notin B$. The following SSCG with non-terminal S recognises L.

	Rule	Condition on Subword	Condition on L
(1)	$S \to w$	1 absent	$w \in A$
(2)	$S \to wS$	1 absent	$w \in B$
(3)	$S \to 0^h S$	1 present	—
(4)	$S \to \varepsilon$	1 present	—

Verification. If $w \in A \cup B$ then clearly our SSCG generates it through rules (1), (2), and (4). Otherwise, $w = u1^{(nh)}$ for some $u \in B$ and some n. Our SSCG obtains w then through rule (2), followed by n applications of rule (3), and one application of rule (4). It is clear that these are the only words generated by our SSCG. □

We now show that there exists a regular language over three characters which cannot be obtained through a simple semi-conditional grammar with only one non-terminal.

Theorem 4.2. *Fix $\Sigma = \{a, b, c\}$ and let $L = a^* \cup b^* \cup c^*$. There is no simple semi-conditional grammar with only one non-terminal S which generates L.*

Proof. We will show that every SSCG of exactly one non-terminal which generates L also generates a string containing at least two distinct characters from Σ, which, by definition of L, proves the theorem.

In our proof, we carry out a case analysis based on the different productions. To limit the number of those—and hence the number of cases we need to consider—we note the following:

- Without loss of generality, assume that no side condition is "presence of S" or "absence of S". These can be omitted without loss of generality.
- If there is a rule $S \to \beta$ in absence of α, and there is a rule $S \to \beta$ in presence of α', with α' being a substring of α, then we can drop these rules and replace them by $S \to \beta$, without any side condition.
- Since $\varepsilon \in L$ and presence rules cannot eliminate all S, there must either be an unconditional rule $S \to \varepsilon$ or an absence rule $S \to \varepsilon$.

We now move on to our detailed case analysis. This leads to a lot of short-lived notation, which we try to preempt by the following convention:

- Numbered rules (e.g. $R_{1,d}$, R_3) are **global names**. In contrast, rules without subscripts (e.g. R, R',) are **local names** and only refer to the case/subcase in which they appear.

We now commence the proof proper.

Let G be an SSCG. Consider the following rules which each may or may not exist in G. Observe that we indicate additional constraints in the last column.

	Rule	Condition on Subword	Constraint
$(R_{1,d})$	$S \to \varepsilon$	absence string contains d	$d \in \Sigma$
(R_2)	$S \to \varepsilon$	no side condition	—
(R_3)	$S \to \varepsilon$	S^h absent	h maximised
(R_4)	$S \to \varepsilon$	$S^{h'}$ present	h' minimised

Note that if both R_3 and R_4 are present, then one may assume w.l.o.g. that $h' > h$; otherwise, $S \to \varepsilon$ is applicable regardless of the sentential form containing S).

Now, assume G generates L. We argue by contradiction.

The following claim is immediate from the fact that $d^* \subset L(G)$ for every $d \in \Sigma$.

Claim 1. *Let $d \in \Sigma$. There exist m, m' with $m + m' > 0$, such that*

$$S \Rightarrow^* d^m S d^{m'}$$

where m, m' depend on d.

We will use this claim to produce contradictions in the main part of this proof. To do so, we will repeatedly focus on the derivation in G of single characters from Σ. For $d \in \Sigma$, define the derivation D_d by

$$D_d \colon S \Rightarrow_1^* S^g \Rightarrow_2 S^i d S^j \Rightarrow_3^* d \tag{4.1}$$

where, as in Claim 1, all of g, i, j depend on d. For convenience, we have labelled the derivation steps

$$\Rightarrow_1^*, \quad \Rightarrow_2, \quad \text{and} \quad \Rightarrow_3^*$$

and we refer to them repeatedly below. We also repeatedly use the following fact about the structure of $\Rightarrow_3^*$ in D_d:

Claim 2. *The derivation step $\Rightarrow_3^*$ in D_d is w.l.o.g. of one of the following forms:*

(1) $\Rightarrow_3^*$ is the empty step;
(2) $\Rightarrow_3^*$ is a repeated application of $R_{1,e}$ for some $e \in \Sigma - \{d\}$;
(3) $\Rightarrow_3^*$ is a repeated application of an absence rule $R_{5,d}$ eliminating instances of S;
(4) $\Rightarrow_3^*$ is a repeated application of an absence rule $R_{5,d}$ and a presence rule $R_{6,d}$, both eliminating instances of S;
(5) $\Rightarrow_3^*$ is a repeated application of an absence rule $R_{7,d} \colon S \to S^{\geq 2}$ whose absence string contains $e \in \Sigma - \{d\}$, followed by repeated applications of $R_{5,d}$ and $R_{6,d}$.
(6) $\Rightarrow_3^*$ is a sequence of presence rules and absence rules, whose absence strings contain only d (if any characters from Σ), followed by repeated applications of $R_{5,d}$ and $R_{6,d}$.

Proof of Claim 2. Case (1) follows in the case that $i = j = 0$ in (4.1).

For case (2), note that if $\Rightarrow_3^*$ uses $R_{1,e}$ for some $e \neq d$, then a repeated application of said rule will yield d, by the structure of the left-hand side of $\Rightarrow_3^*$.

Now, let $R_{5,d} \colon S \to \varepsilon$ be an absence rule, and suppose case (3) fails. We show that we must be in case (4), (5), or (6).

First, w.l.o.g. we may assume that $R_{5,d}$ is the last rule applied to eliminate all instances of S. (Note that it may be the case that no rule of the form $R_{5,d}$ exists.) Since case (3) fails, there must be a presence rule $R_{6,d}$ which is applied *before* the last sequence of $R_{5,d}$-applications has become valid. Note that the presence string of $R_{6,d}$ and the absence string of $R_{5,d}$ satisfy the following relationship:

For any subword in the derivation step $\Rightarrow_3^*$ containing the presence string of $R_{6,d}$, there exists an instance of S after whose elimination:
- $R_{6,d}$ is applicable, or
- $R_{5,d}$ is applicable, or
- no instances of S remain.

In the last case, we are clearly done. Similarly, once $R_{5,d}$ becomes applicable, we use it repeatedly to eliminate all instances of S, also completing the proof. This gives case (4).

Finally, it is possible that $\Rightarrow_3^*$ first *adds* instances of S. To that end, let $R_{7,d}\colon S \to S^{\geq 2}$ be an absence rule whose absence string contains $e \in \Sigma - \{d\}$. (Note the role of d.) Since $\Rightarrow_3^*$ eventually eliminates all instances of S, observe that $R_{7,d}$ must indeed be an absence rule, and that $R_{7,d}$ must be applicable *before* the repeated applications of $R_{6,d}$ and, perhaps eventually, $R_{5,d}$. Indeed, it follows that $R_{7,d}$ is applicable until the presence string for $R_{6,d}$ has been generated. This gives case (5).

If $R_{7,d}$ is not used, then, by exhaustion, only presence rules or absence rules whose absence strings only contain d (if any characters from Σ) can precede $R_{6,d}$. This is exactly case (6). ⊣

We now carry out an analysis of each case of the derivation D_d in (4.1), each of which will lead to a contradiction through Claim 1. The cases depend on the productions in our SSCG G.

Case 1: There exist $R_{1,d}$ and $R_{1,e}$ for $d \neq e$, or there exists R_2. Since the case for R_2 is analogous, we only show the case for $R_{1,d}$ and $R_{1,e}$ below.

Consider $f \in \Sigma - \{d, e\}$, and the derivation

$$D_f\colon S \Rightarrow_1^* S^g \Rightarrow_2 S^i f S^j \Rightarrow_3^* f.$$

Suppose step $\Rightarrow_2$ uses an absence rule R. If the absence string of R contains a character in $\{d, e\}$, then w.l.o.g. let it be d. Now, consider the following derivation, whose steps are justified individually: for some $m + m' > 0$, we have

$$S \underbrace{\Rightarrow^*}_{\text{Claim 1}} e^m S e^{m'} \underbrace{\Rightarrow^*}_{R} e^m S^{i'} f S^{j'} e^{m'} \underbrace{\Rightarrow^*}_{R_{1,d} \text{ or } R_2} e^m f e^{m'}. \tag{4.2}$$

for some i', j'.

Otherwise, step $\Rightarrow_2$ uses some presence rule R with presence string $S^{g'}$, for some $g' \leq g$. Assume this for the remainder of Case 1.

Suppose step $\Rightarrow_1^*$ uses some absence rule $R'\colon S \to S^{>1}$ with absence string containing a character in $\{d, e\}$, say d. Then, for some $m + m' > 0$ and some g'', we have

$$S \underbrace{\Rightarrow^*}_{\text{Claim 1}} e^m S e^{m'} \underbrace{\Rightarrow^*}_{R'} e^m S^{g''} e^{m'} \underbrace{\Rightarrow^*}_{R_{1,d}} e^m S^g e^{m'} \underbrace{\Rightarrow}_{R} e^m S^i f S^j e^{m'} \underbrace{\Rightarrow^*}_{R_{1,d}} e^m f e^{m'}. \tag{4.3}$$

Otherwise, step $\Rightarrow_1^*$ does not use any absence rule $R'\colon S \to S^{>1}$ with absence string containing a character in $\{d, e\}$. Then, for some $m + m' > 0$, we have

$$S \underbrace{\Rightarrow^*}_{\text{Claim 1}} e^m S e^{m'} \underbrace{\Rightarrow^*}_{\Rightarrow_1^* \text{ in } D_f} e^m S^g e^{m'} \underbrace{\Rightarrow}_{R} e^m S^i f S^j e^{m'} \underbrace{\Rightarrow^*}_{R_{1,d}} e^m f e^{m'}. \tag{4.4}$$

Since Equations (4.2) to (4.4) all generate strings of more than one character from Σ, Case 1 is proven.

Case 2: There exists $R_{1,d}$ for exactly one $d \in \Sigma$. Let e, f be the two other characters in $\Sigma - \{d\}$. We handle a simple case first. W.l.o.g. fix e and suppose in the derivation

$$D_e \colon S \Rightarrow_1^* S^g \Rightarrow_2 S^i e S^j \Rightarrow_3^* e$$

that step $\Rightarrow_2$ uses some rule R which is either an unconditional rule (i.e. $\alpha = \beta = 0$ in Definition 2.2) or an absence rule whose absence string does not contain f. Then, for $m + m' \geq 1$ and some i', j', we have

$$S \underbrace{\Rightarrow^*}_{\text{Claim 1}} f^m S f^{m'} \underbrace{\Rightarrow}_{R} f^m S^{i'} e S^{j'} f^{m'} \underbrace{\Rightarrow^*}_{R_{1,d}} f^m e f^{m'} \tag{4.5}$$

which generates a string containing two distinct characters from Σ, giving a contradiction.

Thus, assume that in the derivation D_f (respectively D_e), the rule having f (resp. e) on the right-hand side is rule $R_{8,f}$ (resp. $R_{8,e}$) which is either a presence rule whose presence string contains only instances of S, or an absence rule whose absence string contains e (resp. f). Now, we focus on the steps in the derivation

$$D_d \colon S \Rightarrow_1^* S^g \Rightarrow_2 S^i d S^j \Rightarrow_3^* d.$$

If step $\Rightarrow_2$ requires an absence rule whose absence string contains a character other than d, then w.l.o.g. assume this character is e.

We break down the remaining argument for Case 2 into subcases.

Case 2.1: Step $\Rightarrow_3^*$ requires $R_{7,d}$. Suppose

$$S^i d S^j \underbrace{\Rightarrow^*}_{R_{7,d}} \alpha \underbrace{\Rightarrow^*}_{R_{6,d}} d$$

where α also contains the presence string of $R_{6,d}$. Assume w.l.o.g. that α starts with S (if α ends with S, argue similarly). Since $R_{7,d}$ only adds instances of S and so α does not contain e, the rule $R_{8,f}$ is applicable in the derivation below, which also uses Claim 2 for D_d:

$$S \underbrace{\Rightarrow^*}_{\Rightarrow_1^* + \Rightarrow_2} S^i d S^j \underbrace{\Rightarrow^*}_{R_{7,d}} \alpha \underbrace{\Rightarrow^*}_{R_{7,d}} S^p \alpha \underbrace{\Rightarrow^*}_{R_{8,f}} S^{p'} f S^{p''} \alpha \underbrace{\Rightarrow^*}_{R_{6,d}} f \alpha \underbrace{\Rightarrow^*}_{D_d} f d \tag{4.6}$$

where

$$p \geq \begin{cases} 1 + \text{length of the presence string in } R_{8,f} & \text{if } R_{8,f} \text{ is a presence rule} \\ 1 & \text{otherwise.} \end{cases}$$

Note that the "otherwise"-case is sufficient as it is assumed that if $R_{8,f}$ is an absence rule then that absence rule contains the terminal e.

Case 2.2: Not Case 2.1 and the step $\Rightarrow_1^*$ requires a rule R of the form $S \to S^{\geq 2}$, whose absence string contains a character from Σ. In this case, again using Claim 2 for D_d,

$$S \underbrace{\Rightarrow^*}_{D_d} S^g \underbrace{\Rightarrow^*}_{R} S^{g+p} \underbrace{\Rightarrow}_{R_{8,f}} S^g S^{p'} f S^{p''} \underbrace{\Rightarrow^*}_{R_{1,d}} S^g f \underbrace{\Rightarrow^*}_{D_d} df \tag{4.7}$$

where, as before,

$$p \geq \begin{cases} 1 + \text{length of the presence string in } R_{8,f} & \text{if } R_{8,f} \text{ has a presence rule} \\ 1 & \text{otherwise} \end{cases}$$

which suffices, as before, since we assumed that if $R_{8,f}$ has an absence rule, then that absence rule can only contain the terminal e.

Case 2.3: Not Case 2.1 and Not Case 2.2. In this case, the derivation D_d only requires absence of e (that is, any absence rule appearing in this derivation contains either e or no characters from Σ). Thus, Claim 2 for D_d implies

$$S \underbrace{\Rightarrow^*}_{\text{Claim 1}} f^m S f^{m'} \underbrace{\Rightarrow^*}_{D_d} f^m d f^{m'} . \tag{4.8}$$

(Note that this derivation could also yield df.)

Since derivations (4.5) to (4.8) all generate strings of more than one character from Σ, Case 2 is also proven.

Case 3: Rules $R_{1,d}$ and R_2 do not exist for any $d \in \Sigma$. Observe that in this case R_3 must exist in G since otherwise G could not generate ε. Further, recall that all rules

$$R_{5,d} \colon S \to \varepsilon$$

are absence rules with absence string S^h.

Further, the presence strings of the rules $R_{6,d}$ are of the form $S^{i'} d S^{j'}$, where $i', j' \leq h$, and at most one of i', j' can be h (recall that $S \to \varepsilon$ in presence of S^h cannot be valid).

For $d \in \Sigma$, we again focus on

$$D_d \colon S \Rightarrow_1^* S^g \Rightarrow_2 S^i d S^j \Rightarrow_3^* d\,.$$

Our aim is to limit the number of distinct absence rules containing a character in absence strings used in D_d. We call such rules **special rules**. A rule is **S_d-special** if it is an absence rule of the form $S \to S^{\geq 2}$, and its absence string contains $d \in \Sigma$. For example, $R_{7,d}$ is either S_e- or S_f-special, where $e, f \neq d$.

We do not have much control over the derivation step $\Rightarrow_2$ in D_d. In step $\Rightarrow_1^*$ of D_d, note that the absence rules containing a character from Σ in the absence string are S_x-special, as there are no $R_{1,x}$ rules in the derivation—this follows since neither Case 1 nor Case 2 applies.

Below, we modify the derivation of step $\Rightarrow_1^*$ in D_d to ensure that at most one S_x-special rule is used. This simplifies our case analysis later.

The derivation step $\Rightarrow_1^*$. If step $\Rightarrow_1^*$ uses no S_x-special rules, then there is nothing to do.

If Rule R_4 is used, then $g \geq h' - 1$, as there is no valid condition for usage of any rule $S \to \varepsilon$ which is satisfied by $S^{h'-1}$, since $h' - 1 \geq h$). Then, using any S_x-special rule R one obtains

$$S \underbrace{\Rightarrow^*}_{R} S^{g'} \underbrace{\Rightarrow^*}_{R_4} S^g$$

where $g' \geq g$.

Note that the applications of R_4 need not be required. So, suppose R_4 is never used.

If some S_x-special rule

$$R_{10} \colon S \to S^p$$

with $2 \leq p < h$ is used, then let $m \in \{1, 2, \ldots, p-1\}$ be such that $g \equiv m \mod (p-1)$. Then

$$S \underbrace{\Rightarrow}_{R_{10}} S^p \underbrace{\Rightarrow^*}_{R_3} S^m \underbrace{\Rightarrow^*}_{R_{10}} S^g$$

which is as desired.

Now, we are left with only those cases in which

- rule R_4 is not used, and
- only S_x-special rules with $S \to S^p$, with $p \geq h$, are used.

Note that R_3 is not used after any of the special rules in this part of the derivation, since R_3 requires the absence of S^h, by definition. If multiple S_x-special rules are used, then define the absence rule

$$R_{11}\colon S \to S^p$$

with p minimal, and observe that

$$S \underbrace{\Rightarrow^*}_{R_{11}} S^{1+r(p-1)}$$

for any $r \geq 0$, and that

$$(*) \qquad 2h-1 \leq 2p-1 \leq g \leq g+i'+j'+1 \leq 1+r'(p-1)$$

for large enough r', where i', j' are as they appear in $R_{6,d}$; otherwise, take $i' = j' = 0$.

In summary, from the above analysis, we know that either

$$S \underbrace{\Rightarrow^*}_{R_{11}} S^g$$

or

$$S \underbrace{\Rightarrow^*}_{R_{11}} S^{g'} \text{ and } S \underbrace{\Rightarrow^*}_{R_{11}} S^{g''}$$

and

$$2p-1 = g' \leq g \leq g'' = r'(p-1)+1$$

which follows from (∗). Furthermore, these derivations use at most one S_x-special rule each.

In the case $g' < g$, we have $g \geq 2h$, and thus $i \geq h$ or $j \geq h$. Therefore, the derivation $\Rightarrow_3^*$ uses rule $R_{6,d}$. We may also choose the instance of S in derivation step $\Rightarrow_2$ so that both $i \geq h-1$ and $j \geq h-1$. Thus, $R_{6,d}$ can always be made applicable in $S^i d S^j$, and even if we choose g' or g'' instead of g, if the rule R used in $\Rightarrow_2$ is applicable to $S^{g'}$ ($S^{g''}$, respectively), then the same derivation $S^{g'} \Rightarrow^* d$ ($S^{g''} \Rightarrow^* d$, respectively) can be carried out, too. Note:

- if R is a presence rule, then it is also applicable to $S^{g''}$, and
- if R is an absence rule, then it is also applicable to $S^{g'}$.

Thus, we may assume w.l.o.g. that in D_d, g is modified to g' or g'', based on whether we have $S^{g'} \Rightarrow^* S^i d S^j$ or $S^{g''} \Rightarrow^* S^i d S^j$.

In conclusion, we know that $\Rightarrow_1^*$ and $\Rightarrow_3^*$ use at most one absence rule each, which is S_x-special for some x. Furthermore, the S_x-special rule used in $\Rightarrow_1^*$ can be assumed to be the same for all D_x (with g depending on $x \in \Sigma$).

We now complete the argument of this Case 3 by another case analysis. First, assume:

◇ no S_x-special rules are used

Then, we may choose any d and consider again

$$(\dagger) \qquad D_d\colon S \Rightarrow_1^* S^g \Rightarrow_2 S^i d S^j \Rightarrow_3^* d\,.$$

If $\Rightarrow_2$ is an absence rule and contains some character in Σ, assume w.l.o.g. that it is e. Let $f \in \Sigma - \{d, e\}$. Then, for $m + m' > 0$, Claim 2 implies

$$(4.9) \qquad S \underbrace{\Rightarrow^*}_{\text{Claim 1}} f^m S f^{m'} \underbrace{\Rightarrow^*}_{D_d} f^m d f^{m'}.$$

Next, we assume:

- ◇ there exists an S_d-special rule R_{12} in G which is the only S_x-special rule appearing in $\Rightarrow_1^*$ of (†).

In (†), suppose $\Rightarrow_2$ employs rule R, which is either a presence rule or an absence rule with absence string containing a character in $\Sigma - \{d\}$; w.l.o.g. assume this is e. We may assume that if any S_x-special rule is used in $\Rightarrow_3^*$, then its use is made void by using R_{12} in $S^g \Rightarrow^* S^{\geq g+2h}$; this is done by splitting $S^{\geq 2h}$ into sufficiently large blocks either side of d so that $R_{6,d}$ becomes applicable. In particular, for $m + m' > 0$,

$$S \underbrace{\Rightarrow^*}_{\text{Claim 1}} f^m S f^{m'} \underbrace{\Rightarrow^*}_{R_{12}} f^m S^{\geq g+2h} f^{m'} \underbrace{\Rightarrow}_{R} f^m S^{i+h} d S^{\geq j+h} f^{m'} \underbrace{\Rightarrow^*}_{R_{6,d} \text{ or } R_{5,d}} f^m d f^{m'} \tag{4.10}$$

where the last step uses either $R_{6,d}$ until all S are eliminated, or $R_{5,d}$, if applicable (cf. Claim 2).

Finally, the only case left is that in which:

- ◇ the step $\Rightarrow_2$ uses an absence rule R containing no characters from $\Sigma - \{d\}$.

In that case, if $\Rightarrow_3^*$ uses an S_x-special rule, and its absence string has a character in $\Sigma - \{d\}$, assume w.l.o.g. it is e. Then, for $m + m' > 0$,

$$S \underbrace{\Rightarrow^*}_{\text{Claim 1}} f^m S f^{m'} \underbrace{\Rightarrow^*}_{D_d} f^m S^g f^{m'} \underbrace{\Rightarrow}_{R} f^m S^i d S^j f^{m'} \underbrace{\Rightarrow^*}_{D_d} f^m d f^{m'}. \tag{4.11}$$

Since derivations (4.9) to (4.11) all generate words with distinct characters from Σ, the result follows.

The above exhaustive case analysis shows that no SSCG can recognise our fixed ternary language L. This completes the proof. □

Since all unary regular languages need only one non-terminal (Theorem 4.1) while there exists a ternary regular language which requires two non-terminals (Theorem 4.2), the answer to the following question would prove optimality:

Question 4.3. Can all regular (or linear) languages over a binary alphabet be recognised by a simple semi-conditional grammar with only one non-terminal?

5. Complexity of SSCGs With One Non-Terminal

In this section, we consider the complexity strength of SSCGs with one non-terminal. We first show that such SSCGs can generate non-context free languages.

Theorem 5.1. *Let $\Sigma = \{0, 1, 2, 3, 4, 5\}$. There is a non-context-free language on Σ which can be generated by a simple semi-conditional grammar with only one non-terminal.*

Proof. The idea is to build a language L such that

$$L \cap \{0\}^+\{1\}\{23\}^+\{4\}\{5\}^+ = \{\, 0^n 1 (23)^n 45^n : n \text{ is odd} \,\}. \tag{$*$}$$

Since context-free languages are closed under intersections with regular languages [HU79, p. 135], it follows from the pumping lemma for context-free languages that L is not context-free.

We define our SSCG G as follows:

	Rule	Condition on Subword	Explanation
(I)	$S \to 0S23S5$	0 absent	initialisation
(L1)	$S \to 0S3$	$3S5$ present	loop
(L2)	$S \to 2S5$	$0S3$ present	loop
(L3)	$S \to 0S2$	$2S5$ present	loop
(L4)	$S \to 3S5$	$0S2$ present	loop
(T1)	$S \to 1$	$3S5$ present	termination
(T2)	$S \to 4$	012 present	termination

Verification. We verify that $L(G)$ satisfies (∗). It is easily seen that any generation of a word must begin with the initialisation step, as none of the other rules are applicable otherwise. It is now readily verified that the sequence

$$\text{(I)} - \text{(L1)} - \text{(L2)} - \text{(L3)} - \text{(L4)}$$

generates the word

$$0^3S(23)^3S5^3$$

where each loop step is applied to that S-instance which does *not* appear in the relevant side condition. By induction, the sequence

$$\text{(I)} - \big(\text{(L1)} - \text{(L2)} - \text{(L3)} - \text{(L4)}\big)^n - \text{(T1)} - \text{(T2)}$$

generates the word

$$0^{2n+1}\,1\,(23)^{2n+1}\,4\,5^{2n+1}$$

which proves the $\supseteq$-inclusion in (∗).

To prove equality, first note that the loop can only be carried out in order. Further, by a case analysis for all four loop stages, it is seen that an early termination, by applying (T1) and (T2) before completing a full loop, does not generate a word in $\{0\}^+\{1\}\{23\}^+\{4\}\{5\}^+$. The same occurs in case that any (Li) or (Ti) is applied to an S-instance different from the intended one. This completes the argument. □

By coding, the language L above can be constructed over a binary alphabet, but we note that that would require a larger window width for the required or forbidden words.

5.1. **Decidability of One-Non-Terminal SSCGs.** In contrast to the previous theorem, there is a clear limitation to the complexity of languages which can be generated by SSCGs with only one non-terminal.

Theorem 5.2. *If there is only one non-terminal S in a simple semi-conditional grammar then the language it generates is decidable, even in non-deterministic linear space.*

To give the proof, we require an additional lemma. First, we fix some notation. Let G be an SSCG of one non-terminal, which we denote by S. Let m be the length of the largest side condition. Let p be the length of the largest output of any of the rules in G. Assume $p > 1$, as grammars with $p = 1$ are trivial. Fix an alphabet Σ.

Lemma 5.3. *Suppose $u, v \in (\Sigma \cup \{S\})^*$ and $n > m$. Then a production rule ℓ is applicable to wS^mu if and only if it is applicable to wS^nu.*

Proof. The subwords of wS^mu and wS^nu of length at most m are the same. □

The following lemma forms the core of the proof of Theorem 5.2.

Lemma 5.4. *Suppose $w \in \Sigma^n$ is generable from G. Then there exists $k \in \mathbb{N}$ such that w has a G-derivation τ of length k*

$$\tau\colon \{0, 1, \ldots, k-1\} \to (\Sigma \cup \{S\})^*$$

where $\tau(i) \to \tau(i+1)$ is one step of the derivation, and which satisfies

$$\tau(0) = S \qquad \text{and} \qquad \tau(k-1) = w$$

such that, for every $i < |\tau| = k$, we have

$$|\{\, a \in \tau(i)\colon a = S \,\}| \leq (m + p - 1)(n + 1) + pn\,.$$

Observe that this bound is linear in n, and that m and p are fixed in G.

Proof. Fix some σ which generates w from G. We define a new derivation sequence τ which is as desired. By definition of grammars, we have $\sigma(0) = \tau(0) = S$, and by the end we also have $\sigma(|\sigma| - 1) = \tau(|\tau| - 1) = w$. In fact, we will impose that $|\sigma| = |\tau|$, for ease of presentation. The reader may consider this to be an application of a new rule, $S \to S$. Of course, such an extended derivation sequence can later be shortened by removing the repeated steps.

Fix $i < |\sigma|$.

For any occurrence of a symbol $a \in \Sigma$ in $\sigma(i)$, by considering the remainder of the derivation sequence σ after step i, we may associate this occurrence of a with a corresponding occurrence of a in w. This gives us some $j < n$ with $w(j) = a$, which is the occurrence's **final position** in w. Suppose that

$$\sigma(i) = S^{r_0} a_0^i S^{r_1} a_1^i S^{r_2} \cdots S^{r_{k-1}} a_{k-1}^i S^{r_k} \tag{5.1}$$

whose sequence (r_s) depends on i. Let j_s^i denote the final position of a_s^i in w.

Stage	$\sigma(i)$				
$i-1$		$\cdots$	S	$\cdots$	
i		$\cdots$	a_s^i	$\cdots$	
$\vdots$			$\downarrow$		
$\lvert\sigma\rvert - 1$	$\cdots$	$w(j_s^i - 1)$	a_s^i	$w(j_s^i + 1)$	$\cdots$

FIGURE 1. The final occurrence of some a_s^i in w, which appeared in derivation step i.

Throughout the construction of τ, we will preserve the structure of the words $\sigma(i)$ in τ: for σ as in (5.1), we will define τ so that

$$\tau(i) = S^{r'_0} a_0^i S^{r'_1} a_1^i S^{r'_2} \cdots S^{r'_{k-1}} a_{k-1}^i S^{r'_k}$$

whose sequence (r'_s) also depends on i, and so that $k \leq n$.

The remainder of the proof hinges on the structure of this derivation τ, which we study via the indices j_s^i. For unity of presentation, it is helpful to define

$$j_{-1}^i = -1 \quad \text{and} \quad j_k^i = n\,.$$

Our goal is to construct a derivation sequence τ whose coefficients r'_s satisfy the following relation, defined inductively on i (to keep the notation moderately simple, we now drop the superscript i). For every $s < k$ (as per (5.1)):

(I1) If $r_s < m(j_s - j_{s-1})$ then $r'_s = r_s$.

(I2) If $r_s \geq m(j_s - j_{s-1})$ then $r_s \geq r'_s \geq m(j_s - j_{s-1})$.

In particular,

$$r_s \geq r'_s \quad \text{for all } s < k$$

and

$$\text{if } r_s > r'_s \quad \text{then} \quad r_s > r'_s \geq m(j_s - j_{s-1})\,. \tag{5.2}$$

We now construct the derivation sequence τ—and hence the sequence (r'_s)—from σ while maintaining the inductive hypothesis.

We begin with $\tau(0) = \sigma(0) = S$, and clearly our inductive assumption is maintained.

Suppose $\sigma(i+1)$ is given and $\tau(i)$ already exists. Observe that, by induction and Lemma 5.3, the production rules applicable to $\sigma(i)$ are also applicable to $\tau(i)$. The following is readily verified.

Claim 1. *The productions applicable to $\sigma(i)$ are of one of the following three forms:*

- $S \to \varepsilon$
- $S \to S^t$ *for some* $t > 0$
- $S \to S^t u S^{t'}$ *where* $t, t' \geq 0$ *and* u *begins and ends with a non-terminal*

Using the claim, we now structure our inductive argument to construct τ obeying the inductive hypotheses (I1) and (I2). Our action depends on the production ℓ applied to $\sigma(i)$:

(1) $\underline{\ell = S \to \varepsilon.}$ Suppose ℓ is applied to the block $a_{s-1}S^{r_s}a_s$ in $\sigma(i)$. Thus, $\sigma(i+1)$ contains

$$a_{s-1}S^{r_s-1}a_s.$$

- If $r_s = r'_s$, then apply ℓ to $\tau(i)$ in exactly the same way.
- If $r_s > r'_s$, then let $\tau(i+1) = \tau(i)$.

(2) $\underline{\ell = S \to S^t.}$ Suppose ℓ is applied to the block $a_{s-1}S^{r_s}a_s$ in $\sigma(i)$. Thus, $\sigma(i+1)$ contains

$$a_{s-1}S^{r_s+t-1}a_s.$$

- If $r_s = r'_s$ and $r_s < m(j_s - j_{s-1})$, then apply ℓ to $\tau(i)$ in exactly the same way.
- Otherwise, let $\tau(i+1) = \tau(i)$.

(3) $\underline{\ell = S \to S^t u S^{t'}.}$ We assume that u is a word beginning and ending with terminal symbols. We will generate $\tau(i+1)$ by applying the same rule to $\tau(i)$. Some care must be taken in choosing which S form $\tau(i)$ to apply the rule to, since we must maintain the relationship between every r and r' at step $i+1$.

Suppose that ℓ is applied to an instance of S in the block $a_{s-1}S^{r_s}a_s$ in $\sigma(i)$, which we may hence write as $\alpha\, a_{s-1}S^{r_s}a_s\, \beta$ for some α, β. Thus, $\sigma(i+1)$ is of the form

$$\alpha\, a_{s-1}S^{q}uS^{\tilde{q}}a_s\, \beta\,.$$

Observe that $q + \tilde{q} = r_s + t + t' - 1$.

- If $r_s = r'_s$, then apply ℓ to $\tau(i)$ in exactly the same way as in $\sigma(i)$ (i.e. choose the same instance of S in $\tau(i)$).
- If $r_s > r'_s$ and q is sufficiently small so that case (I1) applies to its block inside $\sigma(i+1)$, we choose an S in $\tau(i)$ such that $\tau(i+1)$ contains

$$a_{s-1}S^{q}uS^{\tilde{q}'}a_s\,.$$

We will necessarily have $\tilde{q} \geq \tilde{q}'$ since $r_s > r'_s$, but note that by additivity and the inductive assumption on r'_s, $\tilde{q}$ satisfies (I2) and $\tilde{q}'$ will be sufficiently large.

- If $r'_s < r_s$ and $\tilde{q}$ is sufficiently small that it will be in case (I1) for $\sigma(i+1)$, we do as in the previous case, mutatis mutandis.
- If $r'_s < r_s$ and q and $\tilde{q}$ are both sufficiently large that they will be in case (I2) for $\sigma(i+1)$, we choose an S in $\tau(i)$ such that $\tau(i+1)$ contains the subword

$$a_{s-1}S^{q'}uS^{\tilde{q}'}a_s$$

where $q' \leq q$ and $\tilde{q}' \leq \tilde{q}$, and both are sufficiently large. This is possible by additivity and the inductive assumption on r'_s.

Claim 2. *The inductive hypothesis on the various r'_s is maintained, for all $i < |\sigma|$.*

Proof of Claim 2. This is immediate from the construction. ⊣

It follows that $\tau(|\tau|-1) = \sigma(|\sigma|-1) = w$, and each $\tau(i+1)$ is generated from $\tau(i)$ by the application of a legal rule from G or by simply doing nothing, so τ is a (generalised) derivation sequence for w.

With τ constructed as above, the following claim gives the required bound.

Claim 3. *For $\tau(i) = S^{r'_0}a_0S^{r'_1}a_1\cdots S^{r'_{k-1}}a_{k-1}S^{r'_k}$, we have*

$$\sum_{s=0}^{k} \max\left(0, r'_s - m(j_s - j_{s-1}) - p + 1\right) \leq kp\,. \tag{5.3}$$

Proof of Claim 3. Since τ is constructed by induction on i, we prove this claim also by induction. Note that (5.3) holds for $i = 0$ since we postulated that $\sigma(0) = \tau(0) = S$.

If $\tau(i)$ and $\tau(i+1)$ have the same terminal symbols, then either $\tau(i+1) = \tau(i)$, or $\tau(i+1)$ was formed from $\tau(i)$ by applying a rule of the form $S \to \varepsilon$, or $\tau(i+1)$ was formed from $\tau(i)$ by applying a rule of the form $S \to S^t$ with $t \leq p$.

In the first two cases, the inequality is clearly preserved from $\tau(i)$ to $\tau(i+1)$.

In the last case, note that by construction, the r'_s from $\tau(i)$ to which we are applying the rule must have

$$r'_s < m(j_s - j_{s-1})\,.$$

Then, the corresponding r'_s in $\tau(i+1)$ will have

$$r'_s < m(j_s - j_{s-1}) + p$$

and so it will not participate in the sum for $\tau(i+1)$. Thus, the non-zero summands for $\tau(i+1)$ are the same as those for $\tau(i)$, meaning the inequality is preserved.

If $\tau(i+1)$ was formed from $\tau(i)$ by the application of a rule of the form $S \to S^tuS^{t'}$, then consideration of the subcases will reveal that the total amount by which the sum has increased is bounded above by $t + t' < p$. Since $\tau(i+1)$ contains at least one more terminal symbol than $\tau(i)$, the inequality is preserved. ⊣

Since $\sum_{s=0}^{k}(j_s - j_{s-1}) = -j_{-1} + j_k = n+1$, (5.3) implies

$$\sum_{s=0}^{k} r'_s \leq m(n+1) + (k+1)(p-1) + kp\,.$$ □

Theorem 5.2 now follows immediately: by Lemma 5.4, to determine membership in the language generated by G, a search over derivation sequences bounded linearly in the number of non-terminals is sufficient. This completes the proof of Theorem 5.2.

Question 5.5. Can Theorem 5.2 be improved, either by weakening hypotheses or by strengthening the complexity conclusion?

6. Membership Testing of SSCGs Is NP-Hard

The next result shows that testing membership in SSCG languages is **NP**-hard even for SSCGs with only two non-terminals.

Theorem 6.1. *The following problem is* **NP***-hard.*

> *Given a simple semi-conditional grammar G of two non-terminals, $2n+4$ terminals, $O(n)$ rules, and an $O(n^3)$ length word u, can G generate u?*

If the given grammar G produces with every rule application at least one terminal, then the problem is **NP***-complete.*

Remark. This hardness is present even in the absence of ε-rules in the grammar. If, furthermore, the grammar contains at least one terminal on the right-hand side of every rule, then the problem is **NP**-complete. The below proof satisfies this hardness relation; the **NP**-completeness part of the claim follows from the fact that one can guess the derivation. Further, the number of steps is bounded by the length of the input word. All conditions, as well as the applicability of rules, can be checked in polynomial time.

Proof of Theorem 6.1. We reduce SAT for CNF formulas to the problem stated in the theorem, thus proving **NP**-hardness.

Fix n.

Below, we define an SSCG G with the following two properties:

(1) For every SAT formula P there exists a word $u_P \in L(G)$ which uniquely codes P in $L(G)$.
(2) Every SAT formula P which is coded in $L(G)$ is satisfiable.

The SSCG G will also generate words which do not code SAT formulas as per our coding above. Hence, we denote the set of **coded formulas** in $L(G)$ by $\tilde{L}(G)$, and we note that

$$\tilde{L}(G) \subset L(G)\,.$$

This is not an issue to prove **NP**-hardness, though, since the sublanguage of all syntactically correctly coded SAT formulas in $L(G)$ is regular, as the following claim shows:

Claim 1. *$\tilde{L}(G) = L(G) \cap R$ for some regular language R. In other words, there exists a regular expression which separates the syntactically correct from the syntactically incorrect instances.*

Proof of Claim 1. With n fixed, a suitable regular expression is readily established via the coding given below. ⊣

To explain the coding, we first define the language. Denote the terminals by

$$x_1, \ldots, x_{n+1}, y_1, \ldots, y_{n+1}, \circ, \bullet, \wr, \star$$

Each SAT instance is made up of clauses, which are each followed by a separator pair $\circ\bullet$ (the symbols $\circ$ avoid ε appearing on the right-hand side of productions). The instance itself is concluded by the pair $\circ\star$. The end of all variable listings is highlighted by the symbol $\wr$. Further, we use the following conventions to express SAT formulas:

- All literals are represented by a single terminal, and the clauses $x_i \vee y_i$ are omitted.
- The disjunctions between literals are omitted.
- For all k, we define $y_k = \neg\, x_k$.
- Every instance is concluded by a listing of all (positive) variables $x_1, \ldots, x_{n+1}$.

	Rule	Condition on Subword	Explanation
(I1)	$S \to TT \bullet S$	x_1 absent	initialisation
(I2)	$S \to x_1 S \star$	x_1 absent	initialisation
(I3)	$S \to x_1 S S \star$	x_1 absent	initialisation
(L1)	$S \to x_k S \mid x_k S S$	$x_{k-1} S \star$ present	loop
(L2)	$S \to x_k \mid x_k S$	$x_{k-1} S S \star$ present	loop
(L3)	$T \to y_k T \mid y_k \mid x_k T$	$x_k S \star$ present	loop
(L4)	$T \to x_k T \mid x_k \mid y_k T$	$x_k S S \star$ present	loop
(T1)	$S \to \wr$	$x_{n+1} S S \star$ present	termination
(T2)	$S \to \wr S$	$x_{n+1} S \star$ present	termination
(T3)	$S \to \circ$	$\wr S$ present	termination
(T4)	$T \to \circ$	S absent	termination

TABLE 2. We indicate the options in rules (Li) by (Li-j); for instance, the rule (L1-2) is "$S \to x_k S S$".

As a result of our coding (see Table 2 below), x_{n+1} cannot be incorporated into any SAT formula, but is instead required to terminate. Hence, we use $n+1$ many variables to code n-variable SAT formulas. For example, a typical instance of a coded 3-variable SAT formula (i.e. with $n = 3$) is given by

$$x_1 x_3 \circ\bullet x_2 y_3 \circ\bullet x_1 x_2 x_3 x_4 \wr \circ\star$$

(note the presence of x_4) which codes the SAT formula

$$(x_1 \vee x_3) \wedge (x_2 \vee \neg x_3)\,.$$

If the separator policy is violated, the formula is incorrect.

In Table 2 below, we construct the required SSCG. Since the number of variables in our SAT formulation is fixed as n, the SSCG's loop rules are valid for $k = 1, 2, \ldots, n+1$.

Explanation. Suppose P is a SAT formula. We show how to generate u_P in G.

First, rule (I1) is applied as many times as needed to generate the required number of clauses in P. We call these the **coded clauses**. For instance, we can generate four coded clauses by applying (I1) four times, yielding

$$S \Rightarrow^* TT \bullet TT \bullet TT \bullet TT \bullet S\,.$$

During generation, a double TT indicates a clause not yet satisfied, while a single T indicates a clause which is satisfied. New variables are instantiated once, either *positively* as in (L1-2) and (L2-2); or *negatively*, as in (L1-1) and (L2-1). Whenever a positively instantiated variable is included in a clause as a positive, then we use (L4-2) to do so. This eliminates one T-instance in the clause, hence coding that this clause is satisfied. We do the same for negatively instantiated variables which appear as a negative, this time using (L3-2). Observe that these are the only ways to eliminate the double TT in a syntactically correct word.

We can now deduce a satisfying valuation for P from the derivation sequence for u_P, by specifying for $k = 1, 2, \ldots, n$:

$$\begin{aligned} x_k = 1 &\iff x_k SS\star \text{ appears in the derivation} \\ y_k = 1 &\iff x_k S\star \text{ appears in the derivation} \end{aligned} \tag{6.1}$$

It is verified by construction that no word which codes a SAT formula can contain both $S\star$ and $SS\star$. Hence, this defines a unique valuation map—but this does not yet show that this valuation map is a satisfying assignment. This is what we establish now.

Claim 2. *$\tilde{L}(G) = \{\, u_P \colon P \text{ is satisfiable} \,\}$, or in other words:*

(1) All syntactically correct instances are satisfiable with respect to at least one variable assignment.

(2) Unsatisfiable instances cannot be generated, unless they are syntactically incorrect.

Proof of Claim 2. We prove (2) first. Suppose P is coded in $\tilde{L}(G)$ via the word u_P. We show that P is satisfiable. In particular, P will be satisfied by the satisfaction definition as given in (6.1) above, denoted by v. To see this, suppose there exists a clause in P not satisfied by v, which we assume w.l.o.g. to be of the form

$$x_1 \vee x_2 \vee x_3\,. \tag{6.2}$$

Since v does not satisfy the clause, the derivation of u_P must contain the words

$$x_1 S\star\,, \qquad x_2 S\star\,, \qquad \text{and} \qquad x_3 S\star\,.$$

By the rules of the grammar, the only way in which v could not possibly satisfy (6.2) is by applying the third option in rule (L3) for all three variables x_1, x_2, and x_3. But then, the following derivation of subwords inside u_P must be present:

$$TT\bullet \Rightarrow x_1 TT\bullet \Rightarrow x_1 x_2 TT\bullet \Rightarrow x_1 x_2 x_3 TT\bullet$$

This derivation cannot be concluded to yield a syntactically correctly coded SAT formula, which contradicts the fact that $u_P \in \tilde{L}(G)$. The same argument applies for longer clauses, or those including both positive and negative variables.

To prove (1), we show that every satisfiable SAT formula in n variables can be generated. First, observe that for any SAT formula P:

$$\begin{gathered} P \text{ is coded in } \tilde{L}(G) \\ \iff \\ \text{In each coded clause } C \text{ in } P, \text{ one instance of } T \text{ is eliminated} \\ \iff \\ P \text{ has a satisfying valuation} \end{gathered} \tag{6.3}$$

Now, let P be a SAT formula of k clauses

$$P = \bigwedge_{i<k} C_i$$

where each clause C_i has m_i variables

$$C_i = \bigvee_{j<m_i} x_{i,j}\,.$$

Suppose v is a satisfying valuation. By definition, each C_i contains a positive (or negative) variable x which is made true by v. When x is coded into the coded clauses inside the word u_P, rule (L4-2) (or rule (L3-2) in the negative case) is applied, hence eliminating

a T-instance from the coded clause C_i. Since this is true for every clause, the formula P can be generated by (6.3), as required. ⊣

NP-hardness is now immediate, since the transformation

$$P \mapsto u_P$$

is computable in linear time. Together with Claim 1, this suffices. Since the grammar G produces with every rule application at least one terminal, membership of words in $L(G)$ is in **NP**; one can guess the derivation. Consequently, if the given grammar G in the statement of the theorem is assumed to produce with every rule application at least one terminal, then the problem is **NP**-complete. □

Note that the bound $O(n^3)$ in the theorem follows from an *a priori* estimate of the maximum number of clauses in a SAT formula on n variables without clause repetition.

7. RE Languages

We complete this paper by showing the strength of SSCGs with three non-terminals.

Theorem 7.1. *Every RE language is generated by some SSCG with three non-terminals.*

Proof. Let L be RE. We describe the construction of our SSCG G which has three non-terminals S, T, U.

Consider a non-deterministic counter automaton A with r counters. We will interpret it as a language-generating device as follows. Based on the current state, the counter automaton can do one of the following:

(1) check the status of a particular counter (zero/non-zero), or
(2) increment/decrement a counter, or
(3) print a terminal symbol on the one-way write-only tape.

The counter automaton A then transitions to the next state which in case (1) above depends on the result of the check.

In any derivation of $u \in L$ in our SSCG G, the sequence of sentential forms will mimic the simulation of the counter automaton. (Note that one-step simulations in A may need several steps in G.)

For ease of presentation, we will first describe the derivation for strings $u \in L$ of length at least

$$1 + (r+1)(r+2)/2$$

where the counter automaton has already printed the first $(r+1)(r+2)/2$ characters of u. We consider the more general case later.

We will use certain sequences of T, U in sentential forms as codes for

- states and dummy states (which are used for a sequence of steps in the derivation in G to implement one-step simulations of A) and
- messages which code the presence or absence of certain patterns in the current sentential form.

Each code is of the form

$$\left(U^4 T \left(\{U^4, U^5\} T\right)^3 U^4 T U^6 T\right)^{r'} U^7 T \tag{7.1}$$

where r' is sufficiently large to code all messages, states, and dummy states.

We also require that

$$\left(U^4T\left(\{U^4,U^5\}T\right)^3U^4TU^6T\right) \tag{7.2}$$

has exactly one U^5 in each of its r' many occurrences. Hence, all codes have the same length.

The following claim is proven by induction on the complexity of words.

Claim 1. *Suppose c is a code as in (7.1). Let c' be the result of applying exactly one of the following changes to c:*

- *A finite number of T-instances are changed into U-instances.*
- *A finite number of U-instances are changed into ε-instances.*
- *A finite number of U-instances are changed into TT-instances.*

Then c' is not a code.

We now work in G.

At the start of any simulation of a step of the counter automaton A (except at initialisation time), the sentential form in G will be of the following form:

$$T\Sigma^1T^{\ell+C_1}\Sigma^2T^{\ell+C_2}\cdots\Sigma^rT^{\ell+C_r}\Sigma^{r+1}\Sigma^*S(code)^+ \tag{7.3}$$

where:

$$\begin{aligned} C_i &= \text{value of the } i\text{-th counter} \\ \ell &= \text{sufficiently large prefixed number so that interferences with anything} \\ &\qquad \text{appearing after } S \text{ is not possible in the descriptions below} \\ code &= \text{sequence of states, the first of which corresponds to state of } A \text{ being simulated} \end{aligned}$$

This definition is recursive. The first *code* after S denotes the current state corresponding to the state of the counter automaton being simulated. The codes appearing after the first *code* can be ignored, as they only describe the history of the simulation.

Any serious violations in the above format, due to SSCG rules being employed in an unintended manner, will be caught separately (cf. Section 7.3).

Intuitively, one can think of the counter values and the current state as instantaneous descriptions of the counter automaton, with the string formed by using the characters in Σ in the sentential form as already output by the counter automaton.

We now describe how to do the steps in the simulation. To do this, we describe the message generation rules, and show how to implement counter automaton instructions.

7.1. **Rules for message generation.** Below, we give the rules which govern the generation of messages in our SSCG. Messages may be generated at any step of the derivation, though they will only be relevant based on the instruction being processed. If irrelevant messages are generated, then they will be caught in the error checking below (see Section 7.3). A message M is always placed immediately after S in the sentential form.

	Rule	Condition on Subword	Explanation
(M1)	$S \to SM$	$T\Sigma^iT^{\ell+1}$ present	M is "counter i is non-zero"
(M2)	$S \to SM$	$T\Sigma^iT^{\ell+1}$ absent	M is "counter i is zero"
(M3)	$S \to SM$	$T\Sigma^iUT^{\ell-1}$ present	M is "counter i is being updated"
(M4)	$S \to SM$	$T\Sigma^iUT^{\ell-1}$ absent	M is "counter i is not being updated"

We also include the following collection of rules, denoted by (M5), which handles format and update errors:

(M5) If u is any of

$$U\Sigma \quad UUTT \quad TU^iT \text{ (for } 0 < i < 4) \quad \Sigma UU \quad TTU \quad \Sigma TU \quad \Sigma UTU \quad T\Sigma T \quad U^8$$

and u is present/absent, then $S \to SM$ where

$$M \text{ is “}u\text{ is present”}/\text{“}u\text{ is absent”}.$$

The role of (M5) is elucidated when we handle all kinds of sanity checks in Section 7.3.

7.2. **Processing of a counter automaton instruction.** Each counter instruction in A is processed as follows. We specify the messages which are generated, and the sequence of steps the simulation takes to carry out the instruction. Further, we explain the process of checking whether the generated messages correspond exactly to the instructions which are being executed.

For ease of presentation, we identify codes for (dummy) states/messages with their state/messages. Suppose the current state is s_j. The state names

$$d\,i\,s_j$$

where $i \in \mathbb{N}$ are dummy states related to the state s_j. These are required to carry out counter automaton instructions which need several steps in our SSCG.

Below, we give the rules of the grammar interspersed with explanations of how the instructions are implemented.

(A) Checking the counter value of counter i to be zero/non-zero in any state:

	Rule	Condition on Subword	Explanation
(1)	$S \to Ss_k$	SMs_j present	s_k is code for next state M is “counter i is zero/non-zero”

(B) Printing of a character in a state:

	Rule	Condition on Subword	Explanation
(2)	$S \to aSs_k$	Ss_j present	s_k is code for next state a is char being printed

(C) Incrementing/decrementing the i-th counter in state s_j:

The rules below use the structure of codes as defined in (7.3) in an essential way. Similarly, dummy states, which are needed to simulate the one-step-transition in the counter automaton A, play an important role here.

	Rule	Condition on Subword	Explanation
(3)	$S \to Sd1s_j$	Ss_j present	
(4)	$T \to U$	$Sd1s_j$ present	change first T in counter which is to be updated
(5)	$S \to Sd2s_j$	$Sd1s_j$ present	
(6)	$S \to Sd3s_j$	$SMd2s_j$ present	M is sequence of messages: "counter i is updated, none other are updated" and no formatting errors
(7)	$U \to \varepsilon \mid TT$	$Sd3s_j$ present	decrement/increment counter
(8)	$S \to Sd4s_j$	$SMd3s_j$ present	M is sequence of messages: "no counter is updated" and no formatting errors
(9)	$S \to Ss_k$	$Sd4s_j$ present	s_k is code for next state

(D) If s_j is a halting state:

	Rule	Condition on Subword	Explanation
(10)	$S \to \varepsilon$	Ss_j present	

(E) Termination rules:

	Rule	Condition on Subword	Explanation
(11)	$T \to \varepsilon$ $U \to \varepsilon$	S absent	

Note that while S is present, only the rules in (C) can update U and T in some dummy states. We explain how to check whether that is done consistently in the following section.

7.3. **Checking Format Errors.**

(i) In states where T is converted to U:
- Check that

$$U\Sigma \quad \Sigma UU \quad TTU \quad \Sigma TU \quad \Sigma UTU$$

are not present (this ensures that to the left of S the only T converted to U is the leftmost T in any counter).
- Check that U^8 is not present (this ensures that to the right of S no T is converted to U).
- Check that there is a $T\Sigma T$ in the sentential form (this ensures that the first T to the left of S is preserved).

(ii) In states where U is converted to ε, check that the codes to the right of S make sense. No

$$UTT \quad TU^iT \text{ (for } 0 < i < 4) \quad T(U^4T)^5 \quad S(U^4T)^5$$

nor a sequence of six $TU^{<6}T$ without a TU^6T / TU^7T in between, is present. Each adjacent pair of TU^6T, without an U^7T in between, is separated by five

copies of $TU^{<6}T$.

Each pair of adjacent TU^7T is separated by $6r'$-many occurrences of $TU^{<7}T$.

(iii) When U is converted to TT, check that there is no UTT.

The above leaves the possibility that the first state codes or messages which appear after S, but before the first state code, are void. However, in that case no progress can be made: S can never be removed (none of the rules above apply, and any use of reinitialisation below will fail).

7.4. **Initialisation.** We now describe how smaller strings are generated, and how the initial configuration (after printing a string of length $(r+1)(r+2)/2$ by the counter automaton) is obtained.

	Rule	Condition on Subword	Comments
(I1)	$S \to u$	T absent	to generate words u of length $\leq (r+1)(r+2)/2$
(I2)	$S \to v$	T absent	v is of the form: $T\Sigma T^{\ell+C_1}T \dots T^{\ell+C_r}\Sigma^{r+1}SC$ where C is the start state

We use (I2) to generate strings of length exceeding $(r+1)(r+2)/2$. Further, we assume in the above that the counter automaton is modified so that, initially, it guesses a string of length $(r+1)(r+2)/2+1$, and then goes to a particular counter state C before progressing.

Note that there are always at least two copies of T to the right of S when a rule to convert T to U is used. Thus, no initialisation rules can be applied after the first initialisation.

Further, if we use a universal counter automaton, then the number of counters is fixed, and we can assign the initial counter values based on the program run by the universal counter automaton. This ensures that in the above construction, the length of the side conditions are in fact independent of the RE language.

By the above argument, our SSCG G simulates the counter automaton A which we fixed at the beginning of this proof. By the rules of our SSCG, it can be readily verified that G generates exactly the counter automaton's language L. Hence, the proof is complete. □

Remark (Side condition sizes independent of alphabet size)**.** In some situations, researchers may wish the condition sizes to be independent of the alphabet size. This only affects the printing rules in the above construction, and, correspondingly, the code size may increase. To handle this, one can proceed as follows.

Firstly, assume that the program simulated by a universal counter automaton intends to write the symbol a_k. Depending on this symbol, it guesses whether a pre-determined counter (we use counter $r+1$ for this purpose) to have value k. in order to verify that the printing was completed successfully, the counter automaton counts down to 0 in counter $r+1$ using rules (3) to (6) above. If the counter didn't have value k, then the automaton never halts. Otherwise, the automaton halts, and it is confirmed that the printing was completed successfully.

To simulate the above step, we add the following rules. Below, let d denote a printing state, and d', d'' dummy printing states.

	Rule	Condition on Subword	Comments
(12)	$S \to a_k T^k S d'$	$\Sigma S\, d$ present	exists for each a_k in alphabet
	$S \to SM$	$\Sigma^{r+1}\Sigma T$ present/absent	M codes presence/absence
(13)	$S \to S\, d''$	$SM\, d'$ present	M codes presence of $\Sigma^{r+1}\Sigma T$

In Rule (12) above, we assume the alphabet is of the form $\{a_2, \ldots, a_h\}$ for some $h \in \mathbb{N}$, in order to avoid trivial counter values.

After the above, the universal counter machine proceeds with the simulation, assuming the guess of the counter is complete.

Now, the rest of the construction is similar to earlier, except that the $(r+1)$-th counter is considered 0 if the sentential form contains a substring of the form $\Sigma^{r+2}S$. If instead the sentential form contains a substring of the form $\Sigma^{r+2}T$, then it is non-zero.

Question 7.2. Can every regular language over a binary alphabet be generated by an SSCG with two non-terminals?

References

[Aar25] Scott Aaronson. Complexity Zoo, 2025. https://complexityzoo.net/Complexity_Zoo.

[DP89] J. Dassow and Gh. Păun. *Regulated Rewriting in Formal Language Theory*, volume 18 of *EATCS Monographs in Theoretical Computer Science*. Springer, 1989.

[FFOR07] Henning Fernau, Rudolf Freund, Marion Oswald, and Klaus Reinhardt. Refining the nonterminal complexity of graph-controlled, programmed, and matrix grammars. *J. Autom. Lang. Comb.*, 12(1-2):117–138, 2007.

[FKO18] H. Fernau, L. Kuppusamy, and R. O. Oladele. New nonterminal complexity results for semi-conditional grammars. In F. Manea, R. G. Miller, and D. Nowotka, editors, *Sailing Routes in the World of Computation, 14th Conference on Computability in Europe, CiE*, volume 10936 of *LNCS*, pages 172–182. Springer, 2018.

[FKOR21] Henning Fernau, Lakshmanan Kuppusamy, Rufus O. Oladele, and Indhumathi Raman. Improved descriptional complexity results for simple semi-conditional grammars. *Fund. Inform.*, 181(2-3):189–211, 2021.

[GR62] Seymour Ginsburg and H. Gordon Rice. Two families of languages related to algol. *J. ACM*, 9(3):350–371, July 1962.

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation*. Addison-Wesley Series in Computer Science. Addison-Wesley Publishing Co., Reading, MA, 1979.

[Kel84] J. Kelemen. Conditional grammars: motivations, definition and some properties. In I. Péak and J. Szép, editors, *Proc. Automata, Languages and Mathematical Systems*, pages 110–123. Karl Marx University of Economics, Budapest, Hungary, 1984.

[MG94] A. Meduna and A. Gopalaratnam. On semi-conditional grammars with productions having either forbidding or permitting conditions. *Acta Cybernet.*, 11(4):307–323, 1994.

[Mv02] Alexander Meduna and Martin Švec. Reduction of simple semi-conditional grammars with respect to the number of conditional productions. *Acta Cybernet.*, 15(3):353–360, 2002.

[Pău85] Gheorghe Păun. A variant of random context grammars: semi-conditional grammars. *Theor. Comput. Sci.*, 41(1):1–17, December 1985.

[Sal73] Arto Salomaa. *Formal languages*. ACM Monograph Series. Academic Press [Harcourt Brace Jovanovich, Publishers], New York-London, 1973.

FB IV – Informatikwissenschaften Theoretische Informatik Campus II / Geozentrum Gebäude H, Universität Trier 54286 Trier, Germany
Email address: fernau@uni-trier.de

School of Computing, National University of Singapore, 13 Computing Drive, Singapore 117417, Republic of Singapore
Email address: sanjayjain@nus.edu.sg

Department of Mathematics, National University of Singapore, 10 Lower Kent Ridge Road, Singapore 119076
Email address: richter@nus.edu.sg
URL: https://linus-richter.github.io

School of Computing, National University of Singapore, 13 Computing Drive, Singapore 117417 and Department of Mathematics, National University of Singapore, 10 Lower Kent Ridge Road, Singapore 119076
Email address: fstephan@nus.edu.sg
URL: https://www.comp.nus.edu.sg/~fstephan/

School of Mathematics and Statistics Victoria University of Wellington Cotton Building, Gate 7, Kelburn Parade Wellington, New Zealand
Email address: dan.turetsky@vuw.ac.nz