Membership for Limited ET0L Languages Is Not Decidable

Henning Fernau

Lehrstuhl Informatik für
Ingenieure und Naturwissenschaftler
Universität Karlsruhe (TH)
Am Fasanengarten 5
D-76128 Karlsruhe
Germany
email: `fernau@ira.uka.de`

Christmas 1994

Joseph went to *register* with Mary,
who was promised in marriage to him.
Luke 2, 5a

Abstract: In this paper, we show how to encode arbitrary enumerable set of numbers given by *register* machines within limited EPT0L systems and programmed grammars with unconditional transfer. This result has various consequences, e.g. the existence of nonrecursive sets generable by 1lET0L systems or by programmed grammars with unconditional transfer. Moreover, ordered grammars are strictly less powerful than 1lET0L systems.

Keywords: Formal languages, parallel and regulated rewriting.

C.R. Categories: F.4.2, F.4.3

Contents

1 Introduction

Regulated rewriting is one of the main topics of formal language theory [3, 13], since there, basically context-free rewriting mechanisms are enriched by different kinds of regulations, hence generally enhancing the generative power of such devices enormously. More recently, investigation of limited forms of parallel rewriting became popular, see e.g. the works of Wätjen [16, 18]. Such investigations add to the understanding of parallelism in rewriting and are, at first glance, a rather different form of context-free rewriting. The indepence of regulated and parallel rewriting is stressed e.g. in [17, 10]. Nevertheless, one of the most beautiful facts of this theory is the close connection between these two variations of context-freeness, first observed by Dassow [2], cf. also [4, 5] and [7, Theorem 3.1].

In this paper, we present some new decidability results for programmed grammars with unconditional transfer and for limited ET0L systems and their consequences. The maybe most surprising result is taken as the headline of this paper. Furthermore, we present some facts from the literature with shortened and/or corrected proofs in some detail in order to give sort of overview on the area, since [15, 14] might not be readily available.

A *programmed grammar* ([12, 15, 3]) is a construct $G = (V_N, V_T, P, S)$, where V_N, V_T, and S are the set of nonterminals, the set of terminals and the start symbol, respectively, and P is a finite set of productions of the form $(r : \alpha \to \beta, \sigma(r), \phi(r))$, where $r : \alpha \to \beta$ is a rewriting rule labelled by r and $\sigma(r)$ and $\phi(r)$ are two sets of labels of such core rules in P. By $\mathrm{Lab}(P)$ we denote the set of all labels of the productions appearing in P. Mostly, we identify $\mathrm{Lab}(P)$ with P. For (x, r_1) and (y, r_2) in $V_G^* \times \mathrm{Lab}(P)$, we write $(x, r_1) \Rightarrow (y, r_2)$ iff either

$$x = z_1\alpha z_2,\ y = z_1\beta z_2,\ (r_1 : \alpha \to \beta,\ \sigma(r_1),\ \phi(r_1)) \in P,\ \text{ and } r_2 \in \sigma(r_1) \qquad (1)$$

or

$$x = y,\ \text{ the rule } r_1 : \alpha \to \beta \text{ for some production } (r_1 : \alpha \to \beta,\ \sigma(r_1),\ \phi(r_1)) \in P$$
$$\text{is not applicable to } x,\ \text{ and } r_2 \in \phi(r_1)\,.$$

In the latter case, the derivation step is done in appearance checking mode. The set $\sigma(r_1)$ is called success field and the set $\phi(r_1)$ failure field of r_1. The language generated by G is defined as

$$L_{\mathrm{gen}}(G) = \{w \in V_T^* \mid (S, r_1) \overset{*}{\Rightarrow} (w, r_2) \text{ for some } r_1, r_2 \in \mathrm{Lab}(P)\}\,.$$

The family of languages generated by programmed grammars containing only context-free core rules is denoted by $\mathcal{L}_{\mathrm{gen}}(\mathrm{P,CF,ac})$. When no appearance checking features are involved, i.e. $\phi(r) = \emptyset$ for each rule in P, we are led to the family $\mathcal{L}_{\mathrm{gen}}(\mathrm{P,CF})$. The special variant of a programmed grammar where the success field and the failure field coincide for each rule in the set P of productions is said to be a programmed grammar with *unconditional transfer* . For convenience, we do not write both the success and the failure field, but use, following Rosenkrantz [12], only one go-to-field. Observe that due to our definition of derivation, a production with empty go-to-field is never applicable. Hence, we assume without saying that grammars with unconditional transfer contain only productions with non-empty go-to-fields. We shall denote the class of languages generated by programmed grammars with context-free productions and with unconditional transfer by $\mathcal{L}_{\text{gen}}$(P,CF,ut). If erasing rules are forbidden, we replace the component CF by CF–λ in our notations.

Besides this mode (called 'free interpretation' by Stotskii), we also consider leftmost derivations. In this case, we additionally require in Eq. (1) that z_1 does not contain an occurrence of α.[1] We indicate this modification by putting 'P–left' instead of 'P' in our notations.

Rosenkrantz showed in [12, page 126]:

Theorem 1.1 There is an algorithm which, given an arbitrary (P–left,CF$-\lambda$,ut) grammar G and a word x, decides whether there is a word in $L_{\text{gen}}(G)$ having x as one of its prefixes.

In particular, we need the following corollary in the following:

Corollary 1.2 For any (P–left,CF$-\lambda$,ut)-language L, there is a Turing machine T_L which, given a word x, decides whether there is a word in L having x as one of its prefixes.

In [16], a new type of parallel derivation was examined, so-called limited L systems. A *k-limited ET0L system* (abbreviated as klET0L system) is a quintuple $G = (V, V', \{P_1, \ldots, P_r\}, \omega, k)$ where V' is a non-empty subset (terminal alphabet) of the alphabet V, $\omega \in V^+$, and each so-called table P_i is a finite subset of $V \times V^*$ which satisfies the condition that, for each $a \in V$, there is a word $w_a \in V^*$ such that $a \to w_a \in P_i$, such that each P_i defines a finite substitution $\sigma_i : V^* \to 2^{V^*}$. G is called propagating if no table contains an erasing production $a \to \lambda$. According to G, $x \Rightarrow y$ (for $x, y \in V^*$) iff there is a table P_i and partitions $x = x_0\alpha_1x_1 \cdots \alpha_nx_n$, $y = x_0\beta_1x_1 \cdots \beta_nx_n$ such that $\alpha_\nu \to \beta_\nu \in P_i$ for each $1 \le \nu \le n$, and, for each $a \in V$, $k_a = |\{\nu \mid \alpha_\nu = a\}| \le k$ where $k_a < k$ implies that a is not contained in $x_0x_1 \cdots x_n$.

The language generated by a generating klET0L system G is $L_{\text{gen}}(G) = \{w \in V'^* \mid \omega \overset{*}{\Rightarrow} w\}$. The corresponding language class is denoted by $\mathcal{L}_{\text{gen}}(k\text{lET0L})$. Since $\mathcal{L}_{\text{gen}}(k\text{lET0L}) \subseteq \mathcal{L}_{\text{gen}}(1\text{lET0L}) = \mathcal{L}_{\text{gen}}(\text{P}, \text{CF} - \lambda, \text{ut})$ by results of [16, 2], we mostly restrict ourselves to that particular language class. The class of languages generated by propagating klET0L systems if denoted by $\mathcal{L}_{\text{gen}}(k\text{lEPT0L})$.

As basic model of computation we consider register machine programs, or — equivalently — while programs [11]. An *r-register machine or r-RM* consists of r registers $R_1, \ldots R_r$, each of them capable of storing one natural number $\rho_1, \ldots, \rho_r$. It can be supplied with a program (r-RMP) which obeys the following syntactical restrictions (the semantics is indicated in parentheses):

[1]This mode is called 'leftmost of type 3' in [3].

- a_i $(1 \leq i \leq r)$ is an r-RMP. (Increment the content ρ_i of register i by one.)
- s_i $(1 \leq i \leq r)$ is an r-RMP. (If $\rho_i > 0$, decrement ρ_i by one.)
- If P_1 and P_2 are r-RMPs, then so is $P_1 P_2$. (Follow first the instructions of P_1 and then the instructions of P_2.)
- If P is an r-RMP, then so is $(P)_i$ $(1 \leq i \leq r)$. (While $\rho_i > 0$ do P.)
- Nothing else is an r-RMP.

We define the function $f_P : \mathbb{N}_0 \multimap\!\!\rightarrow \mathbb{N}_0$ computed by P as follows: Initially, an r-RM is given r-RMP P, and the argument n for which we want to know $f_P(n)$ is stored in R_1, whence $\rho_2 = \cdots = \rho_r = 0$. Then, our r-RM follows the instructions of P according to the above-sketched semantics. When the r-RM, due to some infinite looping, does not stop, $f_P(n) = \perp$ (undefined). Otherwise, r-RM will eventually stop. Then, $f_P(n) = \rho_r$.

r-RMPs (while-programs) are a well-known formalization of computability. Hence, for every recursively enumerable set M of natural numbers, there exists an r-RMP P such that M is the range of f_P.

2 How to represent enumerable sets with 1lEPT0L systems

We proceed with the following steps. First, we give a proof of the fact that we may code any enumerable set of natural numbers in a certain sense by (P–left,CF$-\lambda$,ut) grammars (a proof which is basically shorter than the one of [15, Теорема 3.4] and uses better known facts). We show how to simulate a given (P,CF$-\lambda$,ut) grammar G via a 1lEPT0L-system G' such that $L_{\text{gen}}(G)\{\#\} = L_{\text{gen}}(G')$. On the other hand, Dassow showed in [2] how to simulate a 1lEPT0L-system G' via an equivalent (P,CF$-\lambda$,ut) grammar G. All these simulations are effective ones.

Theorem 2.1 There is an algorithm which, given an arbitrary r-RMP P implementing a function $f_P : \mathbb{N}_0 \multimap\!\!\rightarrow \mathbb{N}_0$ with range M, computes a (P–left,CF$-\lambda$,ut) grammar G with the property $m \in M \iff c^m c_r b^{n_r} c_{r-1} b^{n_{r-1}} \cdots c_1 c_0 \in L_{\text{gen}}(G)$ for some $n_r, \ldots, n_1 \in \mathbb{N}_0$. Furthermore, any word $wc_0 \in L_{\text{gen}}(G)$ is of the form $w = c^m c_r b^{n_r} c_{r-1} b^{n_{r-1}} \cdots c_1$.

Proof. Synctactically, G has as terminal alphabet $\{c, c_0, \ldots, c_r, b\}$ and the nonterminal alphabet consists of S, the start symbol, F, a distinguished failure symbol, σ, a symbol whose occurrence at the end of a sentential form proves the correctness of the simulation of P so far (success witness); the content ρ_i of the register R_i is represented as $A_i^{\rho_i} C_i$, and B_r is a special rubbish symbol for the simulation of register number r. Furthermore, there are help symbols A, A'.

In the following, we describe recursively a procedure how to transform an arbitrary r-RMP P into G in a uniform way.

The most lengthy part of our simulation (which has to be added within Stotskii's work) is a sweeping routine which sweeps the rubbish symbol B_r to the right of the simulated register r and turns it into a rubbish terminal b. Consider the following sequence of productions:

S_0	$B_r \to A_r$	$\{S_1, S_2\}$	$A_r^{i+1} C_r b^j A_{r-1}^k \cdots C_1 \sigma$
S_1	$A_r \to A'$	$\{S_1, S_2\}$	
S_2	$A_r \to A''$	$\{S_3\}$	
S_3	$A_r \to F$	$\{S_4\}$	$A'^i A'' C_r b^j A_{r-1}^k \cdots C_1 \sigma$
S_4	$C_r \to b$	$\{S_5\}$	$A'^i A'' b^{j+1} A_{r-1}^k \cdots C_1 \sigma$
S_5	$A'' \to C_r$	$\{S_6\}$	$A'^i C_r b^{j+1} A_{r-1}^k \cdots C_1 \sigma$
S_6	$A' \to A_r$	$\{S_6, S_7\}$	
S_7	$A' \to F$	$\{\text{exit}\}$	$A_r^i C_r b^{j+1} A_{r-1}^k \cdots C_1 \sigma$

On the right, we see the effect of the sequence of productions on some string of the form $B_r A_r^i C_r b^j A_{r-1}^k \cdots C_1 \sigma$. Observe that if we start with no σ within the string, we leave the sweeping procedure without σ. Note that we will enter the sweep macro only when we know that there is something to sweep.

Now, we give, for each possible elementary part of P, a set of simulating productions for G.

- a_i: (entry : $C_i \to A_i C_i, \{\text{exit}\}$), $(1 \leq i \leq r)$.
- s_i: (entry : $A_i \to b, \{\text{exit}\}$), $(1 \leq i < r)$.
- s_r: (entry : $C_r \to C_r, \{S', S''\}$), ($S' : A_r \to F, \{\text{exit}\}$), ($S'' : A_r \to \sigma, \{S'''\}$), ($S''' : \sigma \to B_r, \{\text{sweep}\}$).
- $(\tilde{P})_i$: (entry : $C_i \to C_i, \{P_i', P_i''\}$), ($P_i' : A_i \to F, \{\text{exit}\}$), ($P_i'' : A_i \to \sigma A_i, \{P_i'''\}$), ($P_i''' : \sigma \to b, \{\tilde{P}\}$) for $1 \leq i < r$. After finishing with the simulation of $\tilde{P}$, the simulation goes back to label 'entry'.
- $(\tilde{P})_r$: (entry : $C_r \to C_r, \{P_r', P_r''\}$), ($P_r' : A_r \to F, \{\text{exit}\}$), ($P_r'' : A_r \to \sigma A_r, \{P_r'''\}$), ($P_r''' : \sigma \to b, \{\text{sweep}\}$). After 'sweep', the simulation of the body $\tilde{P}$ starts. After finishing with the simulation of $\tilde{P}$, the simulation goes back to label 'entry'.

We initialize our simulation with the productions (start : $S \to C_r \cdots C_1 \sigma, \{\text{count,exit}\}$), (count : $C_1 \to A_1 C_1, \{\text{count,exit}\}$). This means that we generate an arbitrary argument $\rho_1 \in \mathbb{N}_0$, coded in the form $C_r \cdots C_2 A_1^{\rho_1} C_1 \sigma$. Furthermore, note that the fact that no symbol A_i occurs in $C_r \cdots C_2 A_1^{\rho_1} C_1 \sigma$ for $i > 1$ corresponds to the convention that computations of RMPs start with empty registers besides the argument register R_1.

Then, we simulate a run of the program P. Before an instruction of the RMP is simulated, we have register contents $\rho_1, \ldots, \rho_r$. This is encoded by $A_r^{\rho_r} C_r b^{m_r} A_{r-1}^{\rho_{r-1}} \cdots b^{m_2} A_1 C_1 \sigma$

for some $m_j \in \mathbb{N}_0$. In case of subtraction of register r and the while-loop simulation, we consider two cases: Either the corresponding register is empty (1) or not (2). If the simulation wrongly enters path (1), the failure symbol F is introduced. If the simulation wrongly enters path (2), the success indicator σ is erased.[2]

Hence by induction, after a successful simulation of P on argument n, in case $f_P(n) = m$ is defined, we arrive at a string of the form $A_r^{\rho_r} C_r b^{n_r} A_{r-1}^{\rho_{r-1}} \cdots b^{n_2} A_1 C_1 \sigma$ for some $n_j \in \mathbb{N}_0$.

We introduce as terminating productions: $(\text{term} : A_r \to c, \{\text{term}, t_1\})$, $(t_1 : C_r \to c_r, \{t_2\})$, $(t_2 : A_{r-1} \to b, \{t_2, t_3\})$, $(t_3 : C_{r-1} \to c_{r-1}, \{t_4\})$, ..., $(t_{2r} : \sigma \to c_0, \{t_{2r}\})$. The claim of the theorem follows. $\square$

If we consider the above-given grammar not with leftmost but with free interpretation, the simulation still works, and the assertion of the theorem also keeps true.

Theorem 2.2 There is an algorithm which, given an arbitrary r-RMP P implementing a function $f_P : \mathbb{N}_0 \multimap \mathbb{N}_0$ with range M, computes a (P,CF$-\lambda$,ut) grammar G with the property $m \in M \iff c^m c_r b^{n_r} c_{r-1} b^{n_{r-1}} \cdots c_1 c_0 \in L_{\text{gen}}(G)$ for some $n_r, \ldots, n_1 \in \mathbb{N}_0$. Furthermore, any word $wc_0 \in L_{\text{gen}}(G)$ is of the form $w = c^m c_r b^{n_r} c_{r-1} b^{n_{r-1}} \cdots c_1$.

Now, we present the announced transformation lemma.

Lemma 2.3 There is an algorithm which, given an arbitrary (P,CF$-\lambda$,ut) grammar G, produces a 1lEPT0L system G' such that $L_{\text{gen}}(G)\{\#\} = L_{\text{gen}}(G')$, where # is a special symbol.

Proof. We can take the proof of Dassow [2, Claim 2] almost literally.[3] Our comments refer to the notation within that proof. Instead of the initialization table, we take an axiom of the form $S[p]$ for any production indicator $[p]$. To any table described by Dassow, we add productions $\# \to F$ in order to prevent shortcuts. Finally, we have a (second) termination table h'_T with $h'_T([p]) = \{\#\}$, $h'_T(a) = \{a\}$ for any terminal symbol a, $h'_T(A) = \{F\}$ otherwise. $\square$

Combining the last lemma with the previous theorem, we immediately get:

Theorem 2.4 There is an algorithm which, given an arbitrary r-RMP P implementing a function $f_P : \mathbb{N}_0 \multimap \mathbb{N}_0$ with range M, computes a 1lEPT0L system G with the property $m \in M \iff c^m c_r b^{n_r} c_{r-1} b^{n_{r-1}} \cdots c_1 c_0 \# \in L_{\text{gen}}(G)$ for some $n_r, \ldots, n_1 \in \mathbb{N}_0$. Furthermore, any word $wc_0\# \in L_{\text{gen}}(G)$ is of the form $w = c^m c_r b^{n_r} c_{r-1} b^{n_{r-1}} \cdots c_1$.

[2]The trick with the success witness σ is due to Stotskii.

[3]A small correction is contained in [5].

3 Simulation of (P,CF$-\lambda$,ut) grammars by klEPT0L systems

This whole section is devoted to the proof of the following encoding theorem, which is furthermore a rather weak one. Hopefully, using the techniques presented in its proof, improvements of such a helpful theorem are possible.

Theorem 3.1 Let $k \geq 2$. There is an algorithm which, given an arbitrary r-RMP P implementing a function $f_P : \mathbb{N}_0 \multimap\!\!\rightarrow \mathbb{N}_0$ with range M, computes a klEPT0L system G with the property $m \in M \iff c^m c_r b^{n_r} c_{r-1} b^{n_{r-1}} \cdots c_1 c_0 \$ b'^j \# \in L_{\text{gen}}(G)$ for some $n_r, \ldots, n_1, j \in \mathbb{N}_0$. Furthermore, any word $wc_0\$b'^j\# \in L_{\text{gen}}(G)$ with $w \in \{c, c_r, \ldots, c_1, b\}^*$ is of the form $w = c^m c_r b^{n_r} c_{r-1} b^{n_{r-1}} \cdots c_1$.

More precisely, we present a rather lengthy proof of the following lemma from which the theorem follows.

Lemma 3.2 Let $k \geq 2$. There is an algorithm which, given an arbitrary (P,CF$-\lambda$,ut) grammar G, produces a klEPT0L system G' such that $w \in L_{\text{gen}}(G) \iff w\$b^j\# \in L_{\text{gen}}(G')$ for some $j \in \mathbb{N}_0$, where $\$, b, \#$ are special symbols.

Proof. Let $G = (V_N, V_T, P, S)$ be a (P,CF$-\lambda$,ut) grammar. $\text{Lab}(P) = \{p_1, \ldots, p_m\}$ with $(p_i : A_i \to w_i, P_i)$. We construct a klEPT0L grammar $G' = (V, V', P', S', k)$ having the claimed property. Let $V' = V_T \cup \{\$, b, \#\}$ and $V = V' \cup V_N \cup \{[p], [p,1], \ldots, [p,4] \mid p \in P\} \cup \{S', F, \sigma, \sigma', \sigma''\}$.

In the following, we describe the different kinds of tables included in P'. We start with an initialization table h_I with $h_I(S') = \{S[p_i]\sigma \mid \exists i(A_i = S)\}$ and $h_I(X) = \{F\}$ otherwise. The simulation is terminated applying the termination table h_T defined by $h_T([p]) = \{\$\}$ for $p \in P$, $h_T(\sigma) = \{\#\}$, $h_T(x) = \{x\}$ for $x \in V'$, and $h_T(Y) = \{F\}$ otherwise.

The actual simulation is done by a sequence of simulating tables $s_{i,1}, \ldots, s_{i,5}$ (or s_i^- if the production p_i is used in appearance checking) which are applied sequentially (this is enforced by using markers $[p,j]$ and $[p]$). The plan is the next: if $(w, p_i) \Rightarrow (v, q_i)$ holds in G (with $q_i \in P_i$), then via 5 derivation steps of G' (or 1 step if the production p_i is used in appearance checking), we obtain $v[q_i]b^l\sigma$ from $w[p_i]b^j\sigma$ for some $l \geq j$. On the other hand, using our simulating tables 5 times (or one time if the production p_i is used in appearance checking), starting with $w[p_i]b^j\sigma$, we get $v[q_i]b^l\sigma$ for some $l \geq j$ which implies that $(w, p_i) \Rightarrow (v, q_i)$ holds in G (with $q_i \in P_i$). Furthermore, sentential forms of the form $u[p]b^j\sigma$ can only occur in a derivation according to G' before applying s_i^-, $s_{i,1}$ or h_T (only these types of tables can deal with the occurrence of $[p_i]$ in the sentential form), and after applying s_i^-, $s_{i,5}$ or h_I (which produce exactly such an occurrence).

Applying the first simulating table $s_{i,1}$, we try to mark the A_i in the present sentential form we want to replace. Of course, there are various pitfalls we have to omit: (1) What happens if A_i does not occur in our sentential form? (2) How do we prevent G' from applying this selection twice, thrice, ..., k times? The basic idea to check these mistakes is again the employment of success witnesses.

First we handle the difficulty (1).

$$\begin{aligned}
s_i^-(A_i) &= \{F\} \\
s_i^-(X) &= \{X\} \text{ for } X \in \{\sigma, b\} \cup V_N \cup V_T \setminus \{A_i\} \\
s_i^-([p_i]) &= \{[q_i])\} \\
s_i^-(Y) &= \{F\} \text{ otherwise}
\end{aligned}$$

$$\begin{aligned}
s_{i,1}(A_i) &= \{A_i, \sigma\} \\
s_{i,1}(\sigma) &= \{\sigma^k\} \\
s_{i,1}(X) &= \{X\} \text{ for } X \in \{b\} \cup V_N \cup V_T \setminus \{A_i\} \\
s_{i,1}([p_i]) &= \{[p_i, 1]\} \\
s_{i,1}(Y) &= \{F\} \text{ otherwise}
\end{aligned}$$

$$\begin{aligned}
s_{i,2}(\sigma) &= \{\sigma'\} \\
s_{i,2}(X) &= \{X\} \text{ for } X \in \{b\} \cup V_N \cup V_T \\
s_{i,2}([p_i, 1]) &= \{[p_i, 2]\} \\
s_{i,2}(Y) &= \{F\} \text{ otherwise}
\end{aligned}$$

In case no A_i has been converted into σ by $s_{i,1}$ erroneously, all occurrences of the success witness σ are deleted by $s_{i,2}$. Now, we deal with difficulty (2). In the errorfree case, there are exactly k occurrences of σ' and one occurrence of σ as suffix of the sentential form.

$$\begin{aligned}
s_{i,3}(\sigma) &= \{b\sigma''\sigma\} \\
s_{i,3}(\sigma') &= \{\sigma''\} \\
s_{i,3}(X) &= \{X\} \text{ for } X \in \{b\} \cup V_N \cup V_T \\
s_{i,3}([p_i, 2]) &= \{[p_i, 3]\} \\
s_{i,3}(Y) &= \{F\} \text{ otherwise}
\end{aligned}$$

In case more than one occurrence A_i has been converted into σ by $s_{i,1}$ erroneously, there will be finally a witness b of this error to the left of \$. In the errorfree case, there are now exactly $k+1$ occurrences of σ''; k of them will be converted to b by the next table.

In case σ has been left in a place other than as suffix, there is now a symbol σ'' at the end of the word. This will be converted to b by the next table, hence witnessing this error.

$$\begin{aligned}
s_{i,4}(\sigma'') &= \{b\} \\
s_{i,4}(X) &= \{X\} \text{ for } X \in \{\sigma, b\} \cup V_N \cup V_T \\
s_{i,4}([p_i, 3]) &= \{[p_i, 4]\} \\
s_{i,4}(Y) &= \{F\} \text{ otherwise}
\end{aligned}$$

We conclude our simulation cycle doing the actual derivation step.

$$\begin{aligned}
s_{i,5}(\sigma'') &= \{w_i\} \\
s_{i,5}(X) &= \{X\} \text{ for } X \in \{\sigma, b\} \cup V_N \cup V_T \\
s_{i,5}([p_i, 4]) &= \{[q_i] \mid q_i \in P_i\} \\
s_{i,5}(Y) &= \{F\} \text{ otherwise}
\end{aligned}$$

□

4 Non-closure and undecidability results

For the convenience of the reader, below we repeat some of Stotskii's arguments in the first two theorems.

Theorem 4.1 $\mathcal{L}_{\text{gen}}(\text{P–left}, \text{CF} - \lambda, \text{ut})$ is not closed under intersection with regular languages.

Proof. Let M be some enumerable but nonrecursive set of natural numbers given as the range of some r-RMP. By Theorem 2.1, there is a language $L_M \in \mathcal{L}_{\text{gen}}(\text{P–left}, \text{CF} - \lambda, \text{ut})$ encoding M in the sense that

$$m \in M \iff c^m c_r b^{n_r} c_{r-1} b^{n_{r-1}} \cdots c_1 c_0 \in L_M$$

for some $n_j \in \mathbb{N}_0$. If $\mathcal{L}_{\text{gen}}(\text{P–left}, \text{CF} - \lambda, \text{ut})$ were closed under intersection with regular languages,

$$L = L_M \cap \{c\}^*\{c_r\}\{b\}^* \cdots \{c_1 c_0\}$$

would lie in $\mathcal{L}_{\text{gen}}(\text{P–left}, \text{CF} - \lambda, \text{ut})$. Obviously, $m \in M$ iff $c^m c_r$ is the prefix of some word of L. By Rosenkrantz' Theorem in the form 1.2, the latter property is decidable, contradicting our assumption on M. □

Stotskii claims that the same is valid for $\mathcal{L}_{\text{gen}}(\text{P}, \text{CF} - \lambda, \text{ut})$, too. This would entail the non-closure of $\mathcal{L}_{\text{gen}}(k\text{lEPT0L})$ under intersection with regular languages for arbitrary $k \geq 2$. Unfortunately, we are not convinced by Stotskii's argument.

As already stated in [15], we can conclude the following undecidability result.

Theorem 4.2 Let w be a non-empty word. In general, there is no Turing machine T_w which, given a (P–left,CF$-\lambda$,ut) grammar G, decides whether w is the suffix of some word of $L_{\text{gen}}(G)$ or not.

Proof. Similarly to the proof of Theorem 2.1, to any enumerable set M we can construct a (P–left,CF$-\lambda$,ut) grammar G such that w is the suffix of some word in $L_{\text{gen}}(G)$ iff $M \neq \emptyset$.[4] □

We can also state the theorem in another way which is more appropriate when separating language classes or disproving closure properties.[5]

Theorem 4.3 Let $L \in \mathcal{L}_{\text{gen}}(\text{P–left}, \text{CF} - \lambda, \text{ut})$. In general, there is no Turing machine T_L which, given an arbitrary (non-empty) word w, decides whether w is the suffix of some word of L or not.

Proof. In the initialization phase of the simulation in the proof of Theorem 2.1, we could have recorded the original argument n in an additional 'register' to the right. Starting with some arbitrary partial recursive function $f : \mathbb{N}_0 \multimap\!\!\rightarrow \mathbb{N}_0$, in this way we arrive at a (P–left,CF$-\lambda$,ut)-language L with the property $x_n c_1 d^n c_0 \in L$ for some x_n iff $f(n)$ is defined. In this case, of course, $x_n = c^{f(n)} c_r \cdots b^{n_2}$ as before. Since in general there is no Turing machine T_f which, given a natural number n, decides whether $f(n)$ is defined or not, the required machine T_L cannot exist either. □

The argument in the preceding proofs are also valid when we consider free interpretation instead of leftmost. Since (P,CF$-\lambda$,ut) grammars are trivially effectively closed under mirror operation, we immediately get the corresponding undecidability results for the prefix properties.[6]

Corollary 4.4 Let w be a non-empty word. There is no Turing machine T_w which, given a (P,CF$-\lambda$,ut) grammar G, decides whether w is the suffix/prefix of some word of $L_{\text{gen}}(G)$ or not. □

Corollary 4.5 Let $L \in \mathcal{L}_{\text{gen}}(\text{P}, \text{CF} - \lambda, \text{ut})$. There is no Turing machine T_L which, given an arbitrary (non-empty) word w, decides whether w is the suffix/prefix of some word of L or not. □

By Lemma 2.3, we know that any given (P,CF$-\lambda$,ut) grammar G can be effectively transformed into a 1lEPT0L system G' such that $L_{\text{gen}}(G') = L_{\text{gen}}(G)\#$. Hence, the suffix/prefix problems under consideration cannot be algorithmically solvable for 1lEPT0L systems as well.

[4]In our above construction, we have $w = c_0$.

[5]This idea has to be added in [15]. The proof given there is the one contained in the preceding theorem.

[6]The corresponding Lemma 1.5.8 in [3] is wrong.

Corollary 4.6 Let w be a non-empty word. There is no Turing machine T_w which, given a 1lEPT0L system G, decides whether w is the suffix/prefix of some word of $L_{\text{gen}}(G)$ or not. $\square$

Corollary 4.7 Let $L \in \mathcal{L}_{\text{gen}}(\text{1lEPT0L})$. There is no Turing machine T_L which, given an arbitrary (non-empty) word w, decides whether w is the suffix/prefix of some word of L or not. $\square$

We were not able to show these properties for klEPT0L systems ($k \geq 2$) as well.

Contrary to the facts described up to now, Rosenkrantz and Stotskii proved that for any of the grammar families considered in this paper, the emptiness problem is algorithmically solvable.

We now prove a new result on programmed grammars (limited systems) concerning their power comparing to recursive languages.

Theorem 4.8 Each of the families $\mathcal{L}_{\text{gen}}(\text{P–left}, \text{CF}, \text{ut})$, $\mathcal{L}_{\text{gen}}(\text{P}, \text{CF}, \text{ut})$ and $\mathcal{L}_{\text{gen}}(\text{1lET0L})$ contains languages which are not recursive.

Proof. We consider again the grammar G_M produced via Theorem 2.1 to some nonrecursive set M. It is simple to alter the construction such that in the end all $b's$ are erased. Hence, we get a language $L_M \in \mathcal{L}_{\text{gen}}(\text{P–left}, \text{CF}, \text{ut})$ with the property $c^m \cdots c_1 c_0 \in L_M$ iff $m \in M$. If L_M were recursive, M would be so, too. Similarly, one can prove the claim for the other two language classes, using Theorems 2.2 and 2.4 instead. $\square$

In particular, we get the announced property that the membership problem is not solvable for 1lET0L languages in general. Unfortunately, this proof does not work for klET0L languages for $k \geq 2$, since our encoding theorem for them is essentially too weak, because we cannot simply erase the b and b' occurring in our construction simultaneously.

In the following, we consider various non-closure properties which also might help to discern the language families under discourse from other ones encountered in the literature.

The families $\mathcal{L}_{\text{gen}}(\text{P}, \text{CF} - \lambda, \text{ut})$ and $\mathcal{L}_{\text{gen}}(\text{1lEPT0L})$ are trivially closed under mirror operation. Contrary to this situation, Stotskii found, combining 1.2 and 4.3:

Theorem 4.9 $\mathcal{L}_{\text{gen}}(\text{P–left}, \text{CF} - \lambda, \text{ut})$ is not closed under mirror operation. $\square$

Because for any of the non-erasing grammars/systems the membership problem is solvable, the corresponding language families with an unsolvable suffix problem cannot be closed under the operator INIT defined by INIT(L)=$\{x \mid \exists y(xy \in L)\}$. Similarly, none of the language families under consideration is closed under quotient with regular sets defined by $L/R = \{x \mid \exists y \in R(xy \in L)\}$ (In this case, we can check the success witness c_3 of our encoding construction quite easily.).

Theorem 4.10 Each of the families $\mathcal{L}_{\text{gen}}(\text{P}, \text{CF}-\lambda, \text{ut})$ and $\mathcal{L}_{\text{gen}}(1\text{lEPT0L})$ is not closed under INIT nor under quotient with regular sets. $\mathcal{L}_{\text{gen}}(\text{P–left}, \text{CF}-\lambda, \text{ut})$ is not closed under quotient with regular sets. $\square$

Finally, we consider our language classes with erasing restricted to singleton terminal alphabets. We use a prime to indicate this subclass.

Theorem 4.11 Let $k \geq 1$. Let $\mathcal{L} \in \{\mathcal{L}_{\text{gen}}(\text{P–left}, \text{CF}, \text{ut}), \mathcal{L}_{\text{gen}}(\text{P}, \text{CF}, \text{ut}), \mathcal{L}_{\text{gen}}(k\text{lET0L})\}$. If $\mathcal{L}$ is closed under intersection with regular sets, then $\mathcal{L}' = \mathcal{L}'_{\text{gen}}(\text{RE})$.

Proof. Each $\mathcal{L}$ is closed under homomorphism. By closure under intersection with regular sets, it would be possible to filter out those words which actually encode a given arbitrary recursively enumerable language M over a one-letter-alphabet. Now, an erasing homomorphism could be used to produce M. $\square$

Observe that this proof is also valid for klET0L languages with $k \geq 2$, since the regular set can use the important position information of b and b', respectively.

Unfortunately, we were not able to convert unary to say binary number representation using any sort of our grammars. This would allow us to establish the following nice relationship: $\mathcal{L}$ is closed under intersection with regular languages iff $\mathcal{L} = \mathcal{L}_{\text{gen}}(\text{RE})$.

Remark 4.12 If $\mathcal{L}_{\text{gen}}(k\text{lET0L})$ were closed under intersection with regular languages, the construction of Lemma 3.2 would yield $\mathcal{L}_{\text{gen}}(\text{P}, \text{CF}, \text{ut}) \subseteq \mathcal{L}_{\text{gen}}(k\text{lET0L})$. On the other hand, $\mathcal{L}_{\text{gen}}(\text{P}, \text{CF}, \text{ut}) \supseteq \mathcal{L}_{\text{gen}}(k\text{lET0L})$ by [4, Theorem 4.5].[7] Therefore, under the above closure assumption, $\mathcal{L}_{\text{gen}}(k\text{lET0L}) = \mathcal{L}_{\text{gen}}(1\text{lET0L})$. This would answer a question raised in [16].

5 Hierarchy results

First of all, we compare the power of erasing in the language families under consideration. This new result easily follows by the fact that any non-erasing family only contains recursive sets.

Theorem 5.1

- $\mathcal{L}_{\text{gen}}(\text{P–left}, \text{CF}-\lambda, \text{ut}) \neq \mathcal{L}_{\text{gen}}(\text{P–left}, \text{CF}, \text{ut})$,
- $\mathcal{L}_{\text{gen}}(\text{P}, \text{CF}-\lambda, \text{ut}) \neq \mathcal{L}_{\text{gen}}(\text{P}, \text{CF}, \text{ut})$
- $\mathcal{L}_{\text{gen}}(1\text{lEPT0L}) \neq \mathcal{L}_{\text{gen}}(1\text{lET0L})$ $\square$

In [5], we raised the question whether 1lEPT0L systems are as powerful as ordered grammars with context-free λ-free core rules or not. Here, we can settle this question.

[7]Any klET0L system is also a $\mathcal{PER}$lET0L system in the sense of [4].

An *ordered grammar* (cf. [8, 15, 3]) is a quintuple $G = (V_N, V_T, P, S, \prec)$, where V_N, V_T, P, and $S \in V_N$ are the nonterminal alphabet, terminal alphabet, set of productions, and start symbol, respectively. $\prec$ is a partial order on P. A production $A \to w \in P$ is applicable to a string $x = x_1 A x_2$, if there is no production $A' \to w' \in P$ with $A' \to w' \prec A \to w$ such that A' occurs in x; the application of $A \to w$ to x yields $y = x_1 w x_2$. We shall write $x \Rightarrow y$ in that case. As usual, $\overset{*}{\Rightarrow}$ denotes the reflexive and transitive closure of $\Rightarrow$, and $L(G_\prec) = \{x \in V_T^* \mid S \overset{*}{\Rightarrow} w\}$. Let $\mathcal{L}_{\text{gen}}(\text{O}, \text{CF}[-\lambda])$ be the language class generated by ordered grammars with context-free [λ-free] core rules.

Theorem 5.2 $\mathcal{L}_{\text{gen}}(\text{O}, \text{CF}[-\lambda]) \subset \mathcal{L}_{\text{gen}}(\text{1lE[P]T0L})$

Proof. The inclusion was shown in [5, Lemma 3.10] whose proof is also valid in the case when we admit λ-rules. Since ordered grammars have a decidable prefix problem [15], the inclusion is strict in the λ-free case. Since for arbitrary ordered grammars, the membership problem is solvable [1, Corollary 3.8], the existence of a nonrecursive language in $\mathcal{L}_{\text{gen}}(\text{1lET0L})$ proves the last claim. $\square$

Another interesting open question is the interrelation between (1) programmed grammars with unconditional transfer and such ones without appearance checking and (2) limited and uniformly limited ET0L systems. Note that there are very close connections between question (1) and (2). By Theorem 4.8, we can answer these questions at least partially, since for programmed grammars without appearance checking and uniformly limited ET0L systems, the membership problem is solvable.

Theorem 5.3

- $\mathcal{L}_{\text{gen}}(\text{P}, \text{CF}, \text{ut}) \not\subseteq \mathcal{L}_{\text{gen}}(\text{P}, \text{CF})$
- $\mathcal{L}_{\text{gen}}(\text{1lET0L}) \not\subseteq \mathcal{L}_{\text{gen}}(\text{ulET0L})$

In particular, this means that even in case e.g. $\mathcal{L}_{\text{gen}}(\text{ulET0L}) \subseteq \mathcal{L}_{\text{gen}}(\text{1lET0L})$, this inclusion has to be strict. Note that in case of accepting grammars/systems, these opposite strict inclusions do actually hold [1, 7].

Similarly, the connection with context-sensitive languages was still open. Our above considerations show

Theorem 5.4 Neither $\mathcal{L}_{\text{gen}}(\text{1lET0L}) = \mathcal{L}_{\text{gen}}(\text{P}, \text{CF}, \text{ut})$ nor $\mathcal{L}_{\text{gen}}(\text{P–left}, \text{CF}, \text{ut})$ contains only context-sensitive languages. $\square$

What about the other inclusions in question? Since each of the language families (with erasing productions!) is easily seen to be closed under homomorphism, we find:

Theorem 5.5

- $\mathcal{L}_{\text{gen}}(\text{CS}) \subseteq \mathcal{L}_{\text{gen}}(\text{1lET0L})$ iff $\mathcal{L}_{\text{gen}}(\text{1lET0L}) = \mathcal{L}_{\text{gen}}(\text{P}, \text{CF}, \text{ut}) = \mathcal{L}_{\text{gen}}(\text{RE})$.
- $\mathcal{L}_{\text{gen}}(\text{CS}) \subseteq \mathcal{L}_{\text{gen}}(\text{P–left}, \text{CF}, \text{ut})$ if and only if $\mathcal{L}_{\text{gen}}(\text{P–left}, \text{CF}, \text{ut}) = \mathcal{L}_{\text{gen}}(\text{RE})$. $\square$

Finally, let us mention that from 1.2 and the corresponding undecidability results for 1lEPT0L and (P,CF$-\lambda$,ut)-languages it is easily seen [15] that neither $\mathcal{L}_{\text{gen}}$(1lEPT0L) nor $\mathcal{L}_{\text{gen}}$(P,CF$-\lambda$,ut) is contained in $\mathcal{L}_{\text{gen}}$(P–left,CF$-\lambda$,ut). In [15], it is claimed that the reversed inclusions do not hold either. We do believe in this result, but not in the proof given there. So, we regard this question as still open.

6 Conclusions

In the present paper, we furthered the research on programmed grammars with unconditional transfer and at the same time that on limited ET0L systems. Especially, we showed that there are nonrecursive languages generable by limited ET0L systems. Note that our construction shows that the membership problem is also unsolvable in general for 1lT0L systems. Hence, we partially answered a question raised in [16] for kl0L systems.

The most important question in our area seems to be whether limited ET0L systems (or, equivalently, programmed grammars with unconditional transfer) can generate all recursively enumerable languages. There is some evidence against it, like the solvability of the emptiness problem, but it might also be that we encountered a rather strange class of grammars characterizing the enumerable languages. Obviously, there is no algorithmical transformation of a say type-0 grammar into an equivalent 1lET0L system. Hence, other non-constructive proof techniques for showing this equivalence must be applied.

The present paper also continues our investigations on how to use decidability results in order to separate language classes [6], since we basically separate $\mathcal{L}_{\text{gen}}$(O,CF $-\lambda$) from $\mathcal{L}_{\text{gen}}$(klEPT0L) by the (un)decidability of the prefix problem.

Note that due to the nature of the proof of the encoding theorems, there may be also connections to the so-called mappings investigations [3, Chapter 9.5].

In the light of the new results, further investigations of variants of limited L systems which are at least as powerful as 1lET0L systems (e.g. function-limited systems with bounded or even not bounded recursive functions [4] or partition-limited systems [9][8]) are interesting. Maybe, we can find even new characterizations of enumerable sets using in principle only context-free derivations.

References

[1] H. Bordihn and H. Fernau. Accepting grammars with regulation. *International Journal of Computer Mathematics*, 53:1–18, 1994.

[2] J. Dassow. A remark on limited 0L systems. *J. Inf. Process. Cybern. EIK (formerly Elektron. Inf.verarb. Kybern.)*, 24(6):287–291, 1988.

[8]Gärtner's PhD on the subject will appear soon.

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

[4] H. Fernau. On function-limited Lindenmayer systems. *J. Inf. Process. Cybern. EIK (formerly Elektron. Inf.verarb. Kybern.)*, 27(1):21–53, 1991.

[5] H. Fernau. Adult languages of propagating systems with restricted parallelism. *J. Inf. Process. Cybern. EIK (formerly Elektron. Inf.verarb. Kybern.)*, 29(5):249–267, 1993.

[6] H. Fernau. Observations on grammar and language families. Technical Report 22/94, Universität Karlsruhe, Fakultät für Informatik, Aug. 1994.

[7] H. Fernau and H. Bordihn. Remarks on accepting parallel systems. *International Journal of Computer Mathematics*, 57(3+4), 1994.

[8] I. Friš. Grammars with partial orderings of the rules. *Information and Control (now Information and Computation)*, 12:415–425, 1968.

[9] S. Gärtner. Partitions-limitierte Lindenmayersysteme. In W. Thomas, editor, *2. Theorietag 'Automaten und Formale Sprachen'*, volume 9220 of *Technische Berichte*, pages 23–27. Universität Kiel, 1992.

[10] S. Gärtner. A remark on the regulation of klET0L systems. *Information Processing Letters*, 48:83–85, 1993.

[11] P. Odifreddi. *Classical Recursion Theory*, volume 125 of *Studies in Logic and Foundations of Mathematics*. Amsterdam: North Holland, 1989.

[12] D. J. Rosenkrantz. Programmed grammars and classes of formal languages. *Journal of the Association for Computing Machinery*, 16(1):107–131, 1969.

[13] A. K. Salomaa. *Formal Languages*. Academic Press, 1973.

[14] E. Stotskii. Control of the conclusion in formal grammars. *Problemy peredachi informacii; translated: Problems of information transmission*, 7(3):87–102, 1971.

[15] E. Stotskii. Управление выводом в формальных грамматиках. Проблемы передачи информации, VII(3):87–102, 1971.

[16] D. Wätjen. k-limited 0L systems and languages. *J. Inf. Process. Cybern. EIK (formerly Elektron. Inf.verarb. Kybern.)*, 24(6):267–285, 1988.

[17] D. Wätjen. Regulation of k-limited ET0L systems. *International Journal of Computer Mathematics*, 47:29–41, 1993.

[18] D. Wätjen and E. Unruh. On extended k-uniformly-limited T0L systems and languages. *J. Inf. Process. Cybern. EIK (formerly Elektron. Inf.verarb. Kybern.)*, 26(5/6):283–299, 1990.