How Powerful is Unconditional Transfer? —When UT meets AC.—

Henning Fernau and Frank Stephan

WSI-97-7

Henning Fernau
Wilhelm-Schickard-Institut für Informatik
Universität Tübingen
Sand 13
D-72076 Tübingen
Germany

Frank Stephan
Mathematisches Institut
Universität Heidelberg
Im Neuenheimer Feld 294
D-69120 Heidelberg
Germany

Email: fernau@informatik.uni-tuebingen.de, fstephan@math.uni-heidelberg.de

Telefon: (07071) 29-77569
Telefax: (07071) 68142

Telefon: (06221)548205
Telefax: (06221)544465

ISSN 0946-3852

How Powerful is Unconditional Transfer? —When UT meets AC.—

Henning Fernau[1] and Frank Stephan[2]

[1] Wilhelm-Schickard-Institut für Informatik, Universität Tübingen,
Sand 13, 72076 Tübingen, Germany,
Email: fernau@informatik.uni-tuebingen.de,
Phone: +49/7071/2977569, Fax: +49/7071/68142.
[2] Mathematisches Institut, Universität Heidelberg,
Im Neuenheimer Feld 294, 69120 Heidelberg, Germany,
Email: fstephan@math.uni-heidelberg.de,
Phone: +49/6221/548205, Fax: +49/6221/544465.

Abstract. We prove that every recursively enumerable language can be generated by a programmed grammar with context-free core rules using unconditional transfer with leftmost derivation of type 3 or type 2. We show that every recursively enumerable tally language can be generated by a programmed grammar with context-free core rules using unconditional transfer or by some k-limited ET0L system, where $k \in \mathbf{N}$ is arbitrary.

1 Introduction

In a series of papers, the first author has studied the concept of unconditional transfer in regulated rewriting and its reflection in the field of restricted parallel rewriting, see [5, 8, 9].

It has been an open question whether programmed grammars with context-free core rules using unconditional transfer characterize the recursively enumerable languages or not. Intuitively, we expected a negative answer to this question, since programmed grammars with context-free core rules using unconditional transfer possess a decidable emptiness problem, as it is known since the first paper of Rosenkrantz on this topic [15], while all non-trivial questions about say type-0-grammars cannot be answered algorithmically by

Rice's theorem. Similar comments are valid for k-limited ET0L system (as introduced by Wätjen in [18]), where $k \in \mathbf{N}$ is arbitrary.

Therefore, the main theorems of this paper are quite unexpected. (A) We prove that every recursively enumerable language can be generated by a programmed grammar with context-free core rules using unconditional transfer with leftmost derivation (of type 3). (B) We show that every recursively enumerable tally language, i.e., a language over a singleton alphabet, can be generated by (i) a k-limited ET0L system, where $k \in \mathbf{N}$ is arbitrary, or (ii) a programmed grammar with context-free core rules using unconditional transfer. This yields an alternative proof for the non-recursiveness of k-limited ET0L languages [6].

Interestingly, the "construction" given in the proofs is necessarily not algorithmic (as pointed out in our previous papers), since (e.g., in case (B)) (1) we distinguish whether the recursively enumerable tally language L (given, e.g., by a programmed grammar with context-free core rules using appearance checking) is empty or not, and (2) we need to know the length of the shortest word in L.

The result is instructive, since the decidability of the emptiness problem is not coded artificially into programmed grammar with context-free core rules using unconditional transfer. In other words, there are "Turing machines" (which are even recursive and constructible in the sense introduced in [7]) for which an analogue of Rice's theorem is not true.

On the other hand, we get (easy) characterizations of the context-free, context-sensitive and recursively enumerable languages by context-free programmed grammars with unconditional transfer working under leftmost derivations of type 1 and type 2.

For the convenience of the reader, we repeat the necessary definitions in the beginning of the sections.

Conventions: $\subseteq$ denotes inclusion, $\subset$ denotes strict inclusion, $|M|$ is the number of elements in the set M. Partial mappings are denoted by $\multimap\!\!\to$, while $\to$ denotes total mappings. $\mathbf{N}$ is the set of natural numbers excluding zero. The empty word is denoted by λ.

In contrast to the usual conventions in formal language theory, we do not consider two languages L_1, L_2 to be equal if $L_1 \setminus \{\lambda\} = L_2 \setminus \{\lambda\}$, since this would destroy much of the non-constructive flavour of our results. In case we adhere to this convention regarding language families, we add " mod λ". Recall the Chomsky hierarchy $\mathcal{L}(\text{REG}) \subset \mathcal{L}(\text{CF}) \subset \mathcal{L}(\text{CS}) \subset \mathcal{L}(\text{RE})$.

2 Some combinatorics on words

In order to prove our first main result, we need a result of Higman [10, Theorem 4.4]. Let $u, v \in \Sigma^*$, $u = u_0 u_1 \dots u_n$. We say that u divides v, written $u \mid v$, if $v \in \Sigma^* u_0 \Sigma^* u_1 \Sigma^* \dots \Sigma^* u_n \Sigma^*$. Naturally, division is a partial ordering of Σ^*. Sometimes, u is called a (sparse) subword of v.

Theorem 1 [Higman] *Every set L of words over a finite alphabet has a finite subset L' such that every word in L has a sparse subword in L'.*

Observe that this result is not effective, that means Theorem 1 gives no algorithm how to determine such a finite set. Indeed for $L \in \mathcal{L}(\mathrm{RE})$, such an algorithm yielding the desired $L' \subseteq L$ would need the halting problem as an oracle. (A similar observation is valid with respect to higher levels of the arithmetical hierarchy.) Nevertheless, such an algorithm exists for regular expressions, so the following observation is of interest.

Theorem 2 *Let $\mathcal{G}$ be a grammar family possessing a decidable emptiness problem and being effectively closed under intersection with regular sets. Then, there exists an algorithm which constructs for every given grammar $G \in \mathcal{G}$ a finite set $L' \subseteq L(G)$ such that for every $v \in L$ there exists a $u \in L'$ with $u \mid v$.*

In the following, we give a proof of this fact, albeit it follows directly from [11, Theorem 3.5]. [3]

Proof. Let $G \in \mathcal{G}$ be given, where $L(G) \subseteq \Sigma^*$. For every $w \in \Sigma^*$, the set $I(w) = \{u \in \Sigma^* : w \mid u\}$ (algebraically, this is just the ideal generated by $\{w\}$ in the division ordering) is regular. Similarly,

$$I(\{w_1, \dots, w_n\}) := I(w_1) \cup \dots \cup I(w_n)$$

is regular. Our problem is to find a finite set L' such that $L(G) \subseteq I(L')$, or equivalently, $L(G) \cap (\Sigma^* \setminus I(L')) = \emptyset$. By assumption, the latter property can be tested algorithmically. So, an algorithm for constructing such a finite set L' may take some enumeration of $L(G)$, say $w_0, w_1, \dots$ (which can be obtained

[3] The work of van Leeuwen was unknown to us when writing this paper. We learnt about it via a paper of Bucher, Ehrenfeucht and Hausler [1] (which contains quite an interesting list of open problems in the area); independently, we were informed about van Leeuwen's work by L. Ilie at DLT conference in Greece.

taking some enumeration of Σ^*, deciding upon a given word $w \in \Sigma^*$ whether $\{w\} \cap L(G) \neq \emptyset$) and compute

$L' := \{w_0\};\ i := 1;$
`while` $L(G) \cap (\Sigma^* \setminus I(L')) \neq \emptyset$ `do begin`
  $L' := L' \cup \{w_i\};\ i := i + 1$
`end`

The program terminates because of Theorem 1. □

Observe that it would be also possible to compute the set of minimal elements of $L(G)$ with respect to the division ordering in this way, denoted by FMS(G). More formally, we consider the set

$$\mathrm{FMS}_{\mathcal{G}} = \{(\mathrm{G}, \mathrm{x}) : \mathrm{G} \in \mathcal{G}, \mathrm{x} \in \mathrm{FMS}(\mathrm{G})\}.$$

Our previous quite general result shows that under the assumptions of Theorem 2 regarding $\mathcal{G}$, $\mathrm{FMS}_{\mathcal{G}}$ is recursive. We will come back to the issue of the recursivity or enumerability of $\mathrm{FMS}_{\mathcal{G}}$ and related sets and functions in section 6.

In case of simple description mechanisms, such a set L' may be computed quite easily. For example, let $\mathcal{R}(\Sigma) \subseteq (\Sigma \cup \{(,),\cup,*\})^*$ denote the class of regular expressions over Σ. $\mathcal{R}(\Sigma)$ is the smallest language over $\Sigma \cup \{(,),\cup,*\}$ satisfying:

- $\Sigma^* \subset \mathcal{R}(\Sigma)$.
- If $R_1, R_2 \in \mathcal{R}(\Sigma)$, then $(R_1 R_2)$, $(R_1 \cup R_2)$ and R_1* lie in $\mathcal{R}(\Sigma)$.

Along this recursive scheme, we can define a function L' by:

- $L'(R_1) = \{R_1\}$ if $R_1 \in \Sigma^* \subset \mathcal{R}(\Sigma)$.
- If $R_1, R_2 \in \mathcal{R}(\Sigma)$, then $L'((R_1 R_2)) = L'(R_1)L'(R_2)$, $L'((R_1 \cup R_2)) = L'(R_1) \cup L'(R_2)$ and $L'(R_1*) = \{\lambda\}$.

It is easily seen that L', when applied to some regular expression R, delivers a finite subset of the language L described by R such that for every $v \in L$ there exists a $u \in L'$ with $u \mid v$.

Van Leeuwen has shown the constructibility of the Higman set for a certain superclass of the context-free languages.

3 Programmed grammars

A *programmed grammar* [4, 15, 17] 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 $V_G^* \times \mathrm{Lab}(P)$, (as usual, $V_G = V_T \cup V_N$ denotes the total alphabet) we write $(x, r_1) \Rightarrow (y, r_2)$ iff either

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

or $y = x$, the rule $r_1 : \alpha \to \beta$ for some $(r_1 : \alpha \to \beta,\ \sigma(r_1),\ \phi(r_1)) \in P$ is not applicable to x, 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

$$L(G) = \{w \in V_T^* : (S, r_1) \stackrel{*}{\Rightarrow} (w, r_2) \text{ for some } r_1, r_2 \in \mathrm{Lab}(P)\}.$$

The family of languages generated by programmed grammars containing only context-free core rules is denoted by $\mathcal{L}(\mathrm{P}, \mathrm{CF}, \mathrm{ac})$. When no appearance checking features are involved, i.e., $\phi(r) = \emptyset$ for each rule in P, we are led to the family $\mathcal{L}$(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 called a programmed grammar with *unconditional transfer*. The corresponding language family is denoted by $\mathcal{L}(\mathrm{P}, \mathrm{CF}, \mathrm{ut})$. For convenience, we do not write both the success and the failure field, but use, following Rosenkrantz [15], only one go-to-field. Observe that, due to our definition of derivation, a production with empty go-to-field is never applicable. Therefore, we assume that go-to-fields are never empty throughout the rest of our paper.

Originally, Rosenkrantz considered programmed grammars with *leftmost derivations* (*of type 3, as it is named in* [4]), i.e., α is not contained in z_1 in the definition (1) above. We denote the corresponding language family, e.g., by $\mathcal{L}(\mathrm{P\text{--}left\text{-}3}, \mathrm{CF}, \mathrm{ut})$. Following Stotskii, we call programmed grammars where leftmost derivation is not enforced in the derivation process programmed grammars under *free interpretation.*

We briefly recall that $\mathcal{L}(\mathrm{P}, \mathrm{CF}, \mathrm{ac}) = \mathcal{L}(\mathrm{P–left\text{-}3}, \mathrm{CF}, \mathrm{ac}) = \mathcal{L}(\mathrm{RE})$, see [15, Theorem 5] and [4, Theorem 1.2.5].

Theorem 3 *$\mathcal{L}(RE) = \mathcal{L}$(P–left-3, CF, ut).*

The following proof is quite similar to [5, Theorem 4.7].

Proof. We have to prove the inclusion "$\subseteq$".

Consider the recursively enumerable language $L \subseteq \Sigma^*$. By Theorem 1, there exists a finite subset L' of L such that

$$L = \bigcup_{w \in L'} (L \cap I(w)),$$

where $I(w)$ denotes the ideal $\{u \in \Sigma^* : w \mid u\}$. Obviously, $L(w) := L \cap I(w)$ is just the shuffle of $I(w)$ with the quotient of L by the regular set

$$I(w) = \{x_0 w_1 x_1 \dots w_n x_n : x_i, w_i \in \Sigma^*, w = w_1 \dots w_n\}.$$

Hence, $L(w) \in \mathcal{L}(\mathrm{RE})$. By [15, Theorem 5] or [4, Theorem 1.2.5] together with [4, Lemma 1.4.8], $L(w)$ can be generated by a (P–left-3,CF,ac) grammar.

We want to be a bit more careful about the construction of G. Assume we have a type-0 grammar in Kuroda normal form for $L(w)$. In the construction of [4, Theorem 1.2.5], the simulation of the type-0 grammar is done in a special coding which is eventually transferred into the desired representation of the terminal word.

Since we know that every word of $L(w)$ has w as divisor, it is easy to alter the construction such that we start with a string of the form

$$S' S_0 w_1 S_1 \dots w_n S_n,$$

where S' starts the simulation as in the proof of [4, Theorem 1.2.5], while the output routine is changed such that the symbols S_i "fill in" the desired parts of the terminal word.

Using Lemma 1.4.8 in [4], a (P–left-3,CF,ac) grammar $G = (V_N, \Sigma, P, S)$ starting with $S \to S' S_0 w_1 S_1 \dots w_n S_n$ can be constructed. Let $P = \{p_1, \dots, p_m\}$, and $(p_\mu : A_\mu \to w_\mu, \sigma(p_\mu), \phi(p_\mu))$.

We give a (P–left-3,CF,ut) grammar $\tilde{G} = (\tilde{V}_N, \Sigma, \tilde{P}, \tilde{S})$ with $L(\tilde{G}) = L(w)$. Since $\mathcal{L}(\mathrm{P–left\text{-}3}, \mathrm{CF}, \mathrm{ut})$ is easily seen to be closed under finite union, this shows that $L \in \mathcal{L}(\mathrm{P–left\text{-}3}, \mathrm{CF}, \mathrm{ut})$.

If V is an alphabet, $\overline{V} = \{\overline{a} : a \in V\}$ is the set of barred symbols. Consider the morphisms $g : V_G^* \to \overline{V_G}^*$ given by $A \mapsto \overline{A}$, if $A \in V_G$, $g' : V_G^* \to (\overline{V_N} \cup \Sigma)^*$ defined by $A \mapsto \overline{A}$, if $A \in V_N$, and $a \mapsto a$, if $a \in \Sigma$, and $h : V_G^* \to V_N^*$, $A \mapsto A$, if $A \in V_N$, and $a \mapsto \lambda$, if $a \in \Sigma$. Let $\tilde{V}_N = V_N \cup \overline{V_G} \cup \{\tilde{S}, F, E\} \cup E_\Sigma$, where $V_N = \{B_1, \ldots, B_\ell\}$ and $\Sigma = \{a_1, \ldots, a_r\}$, $E_\Sigma = \{E_\rho : 1 \leq \rho \leq r\}$. $\tilde{P} = \{\text{init}\} \cup \{p_\mu^+, p_\mu^-, p_\mu', p_\mu'' : 1 \leq \mu \leq m\} \cup \{t_1, \ldots, t_{\ell+1}\} \cup \{T_\rho, T_\rho', T_\rho'', T_\rho''' : 1 \leq \rho \leq r\}$ contains

- an initialization production $(\text{init} : \tilde{S} \to h(x)Eg'(x), \{p_\mu^- : A_\mu = S\})$ instead of $S \to x$ where x denotes $S'S_0w_1S_1 \ldots w_nS_n$,
- simulation productions for $\mu = 1, 2, \ldots, m$:

$$\begin{array}{lll}
(p_\mu^- & : & h(A_\mu) \to F, \{q^+, q^- : q \in \phi(p_\mu)\}), \\
(p_\mu^+ & : & h(A_\mu) \to E, \{p_\mu'\}), \\
(p_\mu' & : & E \to h(w_\mu), \{p_\mu''\}), \\
(p_\mu'' & : & g(A_\mu) \to g(w_\mu), \{q^+, q^- : q \in \sigma(p_\mu)\} \cup \{t_1\}).
\end{array}$$

- termination productions:

$$\begin{array}{llll}
(t_i & : & B_i \to F, \{t_{i+1}\}) & , \text{for } i = 1, 2, \ldots, \ell - 1, \\
(t_\ell & : & B_\ell \to F, \{t_{\ell+1}\}) & , \\
(t_{\ell+1} & : & E \to E_1 \ldots E_r, \{T_1\}) & , \\
(T_\rho & : & g(a_\rho) \to E_\rho, \{T_\rho'\}) & , \text{for } a_\rho \in \Sigma, \\
(T_\rho' & : & E_\rho \to \lambda, \{T_\rho''\}) & , \\
(T_\rho'' & : & E_\rho \to E_\rho a_\rho, \{T_\rho, T_\rho'''\}) & , \\
(T_\rho''' & : & g(a_\rho) \to F, \{T_{(\rho \bmod \mathrm{r})+1}\}) & .
\end{array}$$

In the successful case, the derivation is simulated as follows: if

$$x = x_0w_1x_1 ... w_nx_n$$

is a sentential form derived via grammar G, and rule p_μ is to be applied to x, then

$$h(x)Eg(x_0)w_1g(x_1) \ldots w_ng(x_n)$$

represents x in the simulating grammar $\tilde{G}$, where the simulating grammar has to guess beforehand whether to simulate the negative case via p_μ^- or the positive case via the sequence p_μ^+, p_μ', and p_μ''. Especially, at the end, a terminal string x is represented by $Eg(x_0)w_1g(x_1)\dots w_ng(x_n)$. Now, all barred terminal symbols are converted into their unbarred counterparts.

If the positive case was entered erroneously during a simulation of rule p_μ, i.e., no occurrence of A_μ is present in the current sentential form, the success witness E is erased. In the termination phase, this would lead to the erasure of all nonterminals without introducing terminals as "compensation". Therefore, the shortest word of $L(w)$, namely w, is derived in this way. $\square$

Let us mention that due to the equivalences presented in [8, 9], a similar statement is also true in the case of, e.g., matrix grammars with unconditional transfer. Since the erasing rules in the cycle of rules T_ρ, T'_ρ, T''_ρ are crucial, our simulation technique does not work in the case of forbidding erasing rules.
Observe that one of the usual conventions of formal language theory, namely considering two languages to be equal if they coincide "modulo the empty word", would simplify the argument of the previous theorem considerably. On the other hand, the non-effective flavour of the proof would fade away, and this flavour is necessary in view of our results in [5, 8]. More precisely, since $L(\lambda) = L \bmod \lambda$, we need not refer to Theorem 1 using that convention.

If we consider an arbitrary (P–left-3,CF,ut) grammar generating a tally language, i.e., a language over a one-letter alphabet, it is obvious that the same grammar, considered under free instead of leftmost derivations, generates the same language. This immediately implies:

Corollary 4 $\{L \in \mathcal{L}(RE) : L \subseteq \{a\}^*\} = \{L \in \mathcal{L}(P, CF, ut) : L \subseteq \{a\}^*\}$. $\square$

4 Limited ET0L systems

A *k-limited ET0L system* (abbreviated as klET0L system) is a quintuple $G = (V, V_T, \{P_1, \dots, P_r\}, \omega, k)$ where the terminal alphabet V_T is a non-empty subset of the alphabet V, $\omega \in V^+$, $k \in \mathbf{N}$, 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 $a \to w_a \in P_i$, so that each P_i defines a finite substitution σ_i from V^* to the powerset of V^*. According to G, $x \Rightarrow y$ (for $x, y \in V^*$) iff there is a table P_i and partitions $x = x_0\alpha_1x_1\dots\alpha_nx_n$, $y = x_0\beta_1x_1\dots\beta_nx_n$ such that $\alpha_\nu \to \beta_\nu \in P_i$ for each $\nu = 1, 2, \dots, n$, and, for each $a \in V$, we have $k_a = |\{\nu : \alpha_\nu = a\}| \leq k$ where $k_a < k$ implies that a is not contained in $x_0x_1\dots x_n$.

The language generated by a klET0L system G is $L(G) = \{w \in V_T^* : \omega \overset{*}{\Rightarrow} w\}$. The corresponding language class is denoted by $\mathcal{L}(k\text{lET0L})$.

We know by results of [2, 18] that, for each $k \geq 1$,

$$\mathcal{L}(k\text{lET0L}) \subseteq \mathcal{L}(1\text{lET0L}) = \mathcal{L}(\text{P}, \text{CF}, \text{ut}) \,. \tag{2}$$

The same relation holds for deterministic systems, which means that each table defines a homomorphism. This restriction leads to klEDT0L systems.

Theorem 5 *For each $k \in \mathbf{N}$, we find:*

$$\{L \in \mathcal{L}(RE) : L \subseteq \{a\}^*\} = \{L \in \mathcal{L}(klEDT0L) : L \subseteq \{a\}^*\}.$$

Proof. Consider $L \in \mathcal{L}(\text{RE})$, $L \subseteq \{a\}^*$. Assume that L is not empty. L can be defined by a 1-limited deterministic ET0L system

$$G = (V, \{a\}, \{P_1, \ldots, P_r\}, \omega, 1)$$

in view of the previous corollary and Equation 2. Since we only consider tally languages, it is quite clear that the deterministic system

$$G^k = (V, \{a\}, \{P_1, \ldots, P_r\}, \omega^k, k)$$

generates $(L)^k$. Let $A, B, B', B'', B''', E, F$ be new symbols, with

$$\tilde{V} = \{A, B, B', B'', B''', E, F\} \quad \text{and} \quad \mathrm{V}' = \mathrm{V} \cup \tilde{\mathrm{V}}.$$

The homomorphism $h : V'^* \to V'^*$ is given by $x \mapsto x$ for $x \notin \{a, A\}$, and $a \mapsto A$, $A \mapsto a$. For $1 \leq i \leq r$, let

$$P_i' = h(P_i) \cup \{X \to F : X \in h(\tilde{V}) \setminus \{B\}\} \cup \{B \to B\},$$

where $h(P_i) = \{h(x) \to h(y) : x \to y \in P_i\}$. Define

$$\begin{aligned}
Q_0 &= \{A \to A, B \to B'E\} \cup \{X \to F : X \in V' \setminus \{A, B\}\}, \\
Q_1 &= \{A \to E, E \to E, B' \to B'', a \to a\} \\
&\quad \cup \{X \to F : X \in V' \setminus \{a, A, B', E\}\}, \\
Q_2 &= \{A \to A, E \to \lambda, B'' \to B''', a \to a\} \\
&\quad \cup \{X \to F : X \in V' \setminus \{a, A, B'', E\}\}, \\
Q_3 &= \{A \to A, E \to Ea, B''' \to B', a \to a\} \\
&\quad \cup \{X \to F : X \in V' \setminus \{a, A, B''', E\}\}, \\
Q_4 &= \{B''' \to \lambda, a \to a\} \cup \{X \to F : X \in V' \setminus \{a, B'''\}\}.
\end{aligned}$$

The k-limited EDT0L system

$$G' = (V', \{a\}, \{P_1', \dots, P_r', Q_0, Q_1, Q_2, Q_3, Q_4\}, h(\omega^k)B, k)$$

generates L. Especially, after having applied Q_0 without introducing a failure symbol, we have a string of the form $A^{km}B'E$. Q_1 turns k occurrences of A into E, so that there are $k+1$ E's after applying Q_1 . Q_2 then erases k of the E's, so that Q_3 can use the only remaining E to generate one a. □

This result may shed some new light on the still unsolved inclusion in Eq. 2.

Let us note that leftmost derivations have also been discussed for klT0L systems in [14]. Since the construction of [2] showing the equivalence of programmed grammars with unconditional transfer and 1lET0L systems carries over to the case of leftmost derivations, it easily follows that 1lET0L systems with leftmost derivations characterize the recursively enumerable sets, too. A combination of the arguments of our two main theorems together with [2] even yields that the recursively enumerable sets can also be characterized by k-limited ET0L systems with left derivations. Especially, the simulation technique of a 1-limited derivation by a k-limited derivation shown in the preceding theorem has to be used also at each single derivation step simulation according to the technique presented in the proof of Theorem 3. We briefly sketch such a simulation now. A current sentential form can be represented by

$$(h(x))^k E g(x_0) w_1 g(x_1) \dots w_n g(x_n).$$

In case of a simulation of the positive case (which is the tricky one), a sequence of table applications would turn $g(x_0)(x_1)\dots g(x_n) = v_1 \dots v_r$ into $(v_1')^{q_1} \dots (v_r')^{q_r}$, where $q_j = k$ if the corresponding letter v_j corresponds to the leftmost occurrence of the left-hand side A_i of the original rule to be simulated, and $q_j = 1$ otherwise. By a nondeterministic table, the k occurrences of such v_j' are turned into (hopefully) one occurrence of v_j'' and $k-1$ occurrences of v_j''', which can be tested using non-occurrence checks. Now, the actual simulation of the rule $A_i \to w_i$ can take place. We would get Theorem 5 as a corollary.

5 Other leftmost derivation definitions

Dassow and Paun [4] give two further notions of leftmost derivation for programmed grammars. Since unconditional transfer has not been considered in these cases, we fill in this gap here.

In a leftmost derivation step of type 1, always the leftmost occurrence of a nonterminal has to be rewritten. In [4, Theorem 1.4.1] it is shown that arbitrary programmed grammars with context-free core rules with such derivation interpretation characterize just the context-free languages. This result trivially carries over to the case of unconditional transfer as well, so that we can state without further proof (where the denotation of the language classes is obvious):

Theorem 6 $\mathcal{L}(P\text{–}left\text{-}1, CF, ut) = \mathcal{L}(P\text{–}left\text{-}1, CF{-}\lambda, ut) = \mathcal{L}(CF)\ mod\ \lambda.$ □

The idea of a leftmost derivation step of type 2 is to replace always the leftmost occurrence of a nonterminal which can be rewritten according to the set of applicable rules at the present stage of derivation. More formally, given a programmed grammar $G = (V_N, V_T, P, S)$, we say that a derivation according to G is *leftmost of type 2* if it develops as follows:

1. Start with S and apply any rule $(r : S \to x, \sigma(r), \phi(r))$ in P.

2. If the current string x has been obtained by a successful application of rule r', we set $R = \sigma(r')$. Similarly, if the current string x has been obtained by an application of rule r' in appearance checking manner, we set $R = \phi(r')$. Let $V \subset V_N$ be the set of all left-hand sides of rules in R. In any case, we want to apply a rule $r \in R$ to x. Let $(r : \alpha \to \beta, \sigma(r), \phi(r))$, then a derivation step is made as in the free derivation step definition, with the exceptions that (a) z_1 does not contain an occurrence of a letter from R in Eq. (1) above, and (b) an application of rule r in appearance checking manner is only possible when no rule from R has been applicable, i.e., x does not contain an occurrence of a letter from R.

This interpretation of leftmost derivation allows us to characterize two other families of the Chomsky hierarchy with the help of unconditional transfer.

Theorem 7 • $\mathcal{L}(P\text{–}left\text{-}2, CF, ut) = \mathcal{L}(RE)$,
• $\mathcal{L}(P\text{–}left\text{-}2, CF{-}\lambda, ut) = \mathcal{L}(CS)\ mod\ \lambda$.

Proof. By [4, Theorem 1.4.3] we know that programmed grammars with context-free core rules without appearance checking working in leftmost-2 manner characterize the enumerable or context-sensitive languages, depending on whether we admit erasing rules or not.

So, let $L \in \mathcal{L}(\mathrm{CS})$, $L \subseteq V_T^+$ be given. Consider the representation

$$L = \bigcup_{a \in V_T} L[a]\{a\} \cup (L \cap V_T), \text{ where } L[a] = \{w \in V_T^+ : wa \in L\}\,.$$

By the closure properties of $\mathcal{L}(\mathrm{CS})$ and [4, Theorem 1.4.3], there is a programmed grammar $G[a]$ with context-free core rules without appearance checking working in leftmost-2 manner for each $L[a]$.

We give a programmed grammar $G = (V_N, V_T, P, S)$ with context-free core rules with unconditional transfer working in leftmost-2 style generating $L[a]\{a\}$. By the easily seen closure of $\mathcal{L}(\mathrm{P\text{–}left\text{-}2}, \mathrm{CF}{-}\lambda, \mathrm{ut})$ under union, the containment $L \in \mathcal{L}(\mathrm{P\text{–}left\text{-}2}, \mathrm{CF}{-}\lambda, \mathrm{ut})$ follows.

Let $G[a] = (V_N', V_T, P', S')$. Define $V_N = V_N' \cup \{S, E\}$, where S, E are new nonterminals. Let

$$\begin{aligned} P' &= \{(\mathrm{init} : S \to S'E, \mathrm{Lab}(P)), (\mathrm{term} : E \to a, \{\mathrm{term}\})\} \\ &\cup \{(r : A \to w, \sigma(r) \cup \{\mathrm{term}\}) : (r : A \to w, \sigma(r)) \in P'\}\ . \end{aligned}$$

In any derivation starting with S' which has not terminated yet but still can terminate, we have one occurrence of E. If no occurrence of a left-hand side of a rule in $\sigma(r)$ is present in the current sentential form, then we are forced to use the terminating rule term. Either we now have successfully derived a terminal string, or we can never terminate. □

So, we can summarize our results on leftmost derivations and unconditional transfer in the following corollary.

Corollary 8

$$\begin{aligned} \mathcal{L}(\mathit{CF}) &= \mathcal{L}(\mathit{P\text{–}left\text{-}1, CF, ut}) = \mathcal{L}(\mathit{P\text{–}left\text{-}1, CF}{-}\lambda, \mathit{ut})\ \mathit{mod}\ \lambda \\ &\subset \mathcal{L}(\mathit{P\text{–}left\text{-}3, CF}{-}\lambda, \mathit{ut}) \subset \\ \mathcal{L}(\mathit{CS}) &= \mathcal{L}(\mathit{P\text{–}left\text{-}2, CF}{-}\lambda, \mathit{ut}) \subset \\ \mathcal{L}(\mathit{RE}) &= \mathcal{L}(\mathit{P\text{–}left\text{-}2, CF, ut}) = \mathcal{L}(\mathit{P\text{–}left\text{-}3, CF, ut}) \end{aligned}$$

Derivations of mode left-1 can be introduced for klET0L systems as well, leading to another characterization of the context-free languages. Left-2 derivations do not make sense, since we have rules for every symbol in every table.

6 A recursion to recursion theory

Recursion theoretic notations and definitions can be found in the book of Odifreddi [13]. Note that a problem A is many-one reducible to B iff there is a computable function f such that $A(x) = B(f(x))$ for all x.

As already said in the Introduction, our main results are unexpected, since Rosenkrantz [15] proved that grammars with unconditional transfer have a decidable emptiness problem. Therefore, it is interesting to see which other problems (which are related to the proof of our main result 5) are computable for grammars with unconditional transfer (in contrast to say type-0-grammars).

First, we consider the problem of determining the length of the shortest word. More precisely, we have as input of LSW the description of a grammar $G \in \mathcal{G}$, where $\mathcal{G}$ is some fixed grammar family, and as output the length of the shortest word in $L(G)$, if $L(G) \neq \emptyset$. So, LSW is a partial mapping $\mathcal{G} \multimap \mathbf{N}_0$. In the following, let $\mathcal{G}_{ut}$, $\mathcal{G}_{ac}$, or $\mathcal{G}_{kl}$ denote the families of programmed grammars with unconditional transfer and context-free, possibly erasing core rules, of programmed grammars with appearance checking and context-free, possibly erasing core rules, or of klET0L systems, respectively. We add left-3 in our notations, if we enforce leftmost derivations of type 3.

Theorem 9 *LSW is partial recursive for $\mathcal{G}_{ut}$, $\mathcal{G}_{ut,left-3}$ and $\mathcal{G}_{kl}$, but is is not partial recursive for $\mathcal{G}_{ac}$ or $\mathcal{G}_{ac,left-3}$.*

Proof. It is clear that a non-trivial property like LSW cannot be computed for Turing machines or, equivalently, $\mathcal{G}_{ac}$ or $\mathcal{G}_{ac,left-3}$. Since $\mathcal{G}_{kl}$ can be effectively transformed into $\mathcal{G}_{ut}$, the latter class remains to be treated more thoroughly.

Let $G \in \mathcal{G}_{ut}$ be given. Assume $L(G) \subseteq V^*$. First, an algorithm can check whether $L(G)$ is empty, in which case it does not terminate, or not. If $L(G) \neq \emptyset$, the algorithm subsequently (for $n = 0, 1, 2, \ldots$) checks $(*)$ whether $L_n(G) := L(G) \cap \{w \in \Sigma^* : |w| \leq n\}$ is empty or not. Since $L(G) \neq \emptyset$, this checking procedure will terminate, hence yielding the smallest n such that the intersection $(*)$ is non-empty. This n is the value LSW(G) we were looking for.

How does the algorithm solve $(*)$? Consider $G = (V, V_T, P, S)$ a bit more in detail. We construct a grammar $G' = (V', V_T, P', S) \in \mathcal{G}_{ut}$ such that $L(G') = L_n(G)$, so that checking $(*)$ results in checking $L(G')$ for emptiness. Let H be a new symbol, $V' = V \cup \{H\}$, and let h be a morphism that assigns to $w \in V_G^*$ the word H^m, where m equals the number of occurrences of terminals in w. P' consists of all rules $(r : A \to wh(w), \sigma(r) \cup \{t_1\})$, where $(r : A \to w, \sigma(r)) \in P$, and, in addition, of rules $(t_i : H \to \lambda, \{t_{i+1}\})$ for $i = 1, 2, \ldots, n$ and of the rule $(t_{n+1} : H \to H, \{t_{n+1}\})$.

Finally, it clearly makes no difference in our construction whether we consider free derivations or leftmost derivations of type 3. $\square$

Observe that our last theorem shows that the conditions involved in transforming an arbitrary $G \in \mathcal{G}_{ac}$ (generating a tally language) into some equivalent $G' \in \mathcal{G}_{kl}$ are themselves computable for $\mathcal{G}_{kl}$, but clearly not for $\mathcal{G}_{ac}$. Moreover, it is easy to see that LSW is computable for $\mathcal{G}_{ac}$ using a halting problem oracle, and, on the other hand, the halting problem is solvable for a Turing machine which is given LSW as an oracle. Clearly, the halting problem for Turing machines is equivalent to the emptiness problem for $\mathcal{G}_{ac}$. Now, we could transform a given $G = (V, V_T, P, S) \in \mathcal{G}_{ac}$ into $G' = (V, \{a\}, P', S)$, where $P' = \{(r : A \to f(w), \sigma(r)) : (r : A \to w, \sigma(r)) \in P\} \cup \{(new : S \to a, \{new\})\}$. Here, f is the inclusion morphism of V^* into V_G^*, trivially extended to a morphism of V_G^* into V_G^*. Now, $L(G) = \emptyset$ if and only if LSW$(G') = 1$.

In the proof of Theorem 3, another function played a key rôle in the construction, namely a function delivering to a given grammar G generating a nonempty language the uniquely determined finite minimal set FMS$(G) \subseteq L(G)$ such that, for every $v \in L(G)$, there exists a $u \in$ FMS(G) with $u|v$, where minimality refers to division ordering. More formally, we consider the set

$$\mathrm{FMS}_{\mathcal{G}} = \{(\mathrm{G}, \mathrm{x}) : \mathrm{G} \in \mathcal{G}, \mathrm{x} \in \mathrm{FMS}(\mathrm{G})\}$$

and discuss its enumerability and recursivity. Similar, we consider the set

$$\mathrm{SMW}_{\mathcal{G}} = \{(\mathrm{G}, \mathrm{x}) : \mathrm{G} \in \mathcal{G}, \mathrm{x} \in \mathrm{SMW}(\mathrm{G})\},$$

where SMW(G) is the set of shortest (minimal) words of $L(G)$. It is clear that for every grammar G generating a non-empty language $L(G) \subseteq V^*$, the number of of shortest (minimal) words, defined as

$$\mathrm{NSW(G)} := |\mathrm{SMW(G)}|,$$

is greater than 0. In this case, we have

$$\mathrm{NSW(G)} = |\mathrm{FMS(G)} \cap \mathrm{V^{LSW(G)}}| = |\mathrm{L(G)} \cap \mathrm{V^{LSW(G)}}|.$$

Moreover, if $\mathrm{FMS}_{\mathcal{G}}$ is recursive (or enumerable) for an arbitrary grammar family $\mathcal{G}$, $\mathrm{SMW}_{\mathcal{G}}$ is recursive (or enumerable), while LSW and NSW are recursive (or partial recursive) for $\mathcal{G}$ in that case. Therefore, LSW and NSW are not partial recursive for $\mathcal{G}_{ac[,left-3]}$, and SMW and FMS are not enumerable for $\mathcal{G}_{ac[,left-3]}$. If SMW is recursive for $\mathcal{G}$, then NSW is recursive for $\mathcal{G}$, too.

Furthermore, observe that if we restrict ourselves to grammar families generating tally languages, SMW = FMS. NSW (and hence LSW) is not recursive for tally Turing machines, since it is just the halting problem.

Theorem 10 *For grammars of the type $\mathcal{G}_{ut,left-3}$, $\mathcal{G}_{ut}$ and $\mathcal{G}_{kl}$, the problem to decide whether* $\mathrm{NSW(G)} \geq 2$ *is many-one equivalent to the halting problem K for Turing machines.*

Proof. We show the theorem for $\mathcal{G}_{ut,left-3}$, the other two cases can be obtained via easy modifications.

First, we fix some Gödelization of register machines, which are known to be equivalent to Turing machines, see [16]. For the given classes of grammars, the value of LSW is computable without an oracle. Thus the set

$$\{G : (\exists v, w \in L(G))\, [v \neq w \wedge |v| = |w| = \mathrm{LSW}(G)]\}$$

is enumerable and so many-one-reducible to the halting problem K. So it remains to show that K is also many-one-reducible to this set.

The proof is a combination of techniques developed in Theorem 3 together with the register machine simulations contained in [5]. We extend a grammar $G'_n = (V', \{E\}, P', S')$ which can produce the word E iff the n-th register machine holds with input n. Furthermore, it may produce in any case some string in V'^* without being able to derive a terminal string except λ. This grammar can be computed effectively from n.

The new grammar $G_n = (V' \cup \{A, B, E, S\}, \{a, b\}, P, S)$ is a sequence of three steps. st denotes the set of the starting place of the G'_n. $ex_0, ex_1, ex_2, ex_3, ex_4, in_0, in_1$ are new labels which are not accessed by P'. P' is extended to P in the following way: P contains all productions $(pl, pr, dt \cup \{ex_0\})$ with $(pl, pr, dt) \in P'$ plus the new productions $(in_0 : S \to S'AB, st)$, $(in_1 : S \to ab, \{ex_0\})$, $(ex_0 : E \to BA, \{ex_1\})$, $(ex_1 : A \to a, \{ex_2\})$, $(ex_2 : B \to b, \{ex_3\})$, $(ex_3 : A \to \lambda, \{ex_4\})$, $(ex_4 : B \to \lambda, \{ex_4\})$. So the first step is the initialization, the second one the simulation of G'_n and the third one produces either the output ab or – provided that the simulation process of G'_n terminated correctly yielding the string EAB – the output ba.

If the n-th machine terminates, then there is a way to produce first S' from S, then E from S' via emulating G' and then ba from E via the productions labelled with $ex_0, ex_1, ex_2, ex_3, ex_4$. If the n-th machine does not terminate, then the grammar either never reaches a state ex_i or it directly produces ab via the production labeled by in_1 or it reaches ex_0 where instead of E some word in V'^*AB is present. If this word is just AB, then it is transformed to ab, and if it contains some nonterminals from V', then these are never removed and so the grammar does not produce valid output.

So the following holds: if the n-th machine with input n halts, then $L(G_n) = \{ab, ba\}$, if it does not halt, then $L(G_n) = \{ab\}$. Thus, one has $n \in K \Leftrightarrow \mathrm{NSW(G_n)} \geq 2$, and so the reverse many-one reduction is obtained. □

Therefore, the set $\mathrm{FMS}_{\mathcal{G}}$ cannot be computed by a Turing machine for $\mathcal{G} = \mathcal{G}_{ut,left-3}$. So, the information put into the grammar constructed in Theorem 3 cannot be retrieved algorithmically from some arbitrary grammar with unconditional transfer. Note that for appearance checking the many-one-equivalence does not hold. There, the class of the grammars with at least two shortest words is only Turing equivalent to K but properly above K with respect to many-one reductions.

Corollary 11 *1. For the grammar families $\mathcal{G}_{ut,left-3}$, $\mathcal{G}_{ut}$ and $\mathcal{G}_{kl}$, SMW is enumerable but not recursive.*

2. For grammar families $\mathcal{G}_{ac,left-3}$ and $\mathcal{G}_{ac}$, SMW is not enumerable.

Proof.

1. The previous theorem shows that SMW is not recursive, since otherwise

the problem treated in the previous theorem would be decidable. The enumerability of SMW trivial, since LSW is partial recursive and $L(G)$ is enumerable.

2. If SMW were enumerable, LSW would be partial recursive, too, contradicting Theorem 9.

□

So, the non-recursive function NSW is approximable from below in the sense that the sets

$$\{(G,x): x \leq \mathrm{NSW(G), G} \in \mathcal{G}_{\mathrm{ut[,left-3]}}\}$$

are enumerable by the preceding corollary.

7 Final remarks

It is easy to change the tables of the klET0L system constructed in our second main theorem in a way such that they are deterministic. Hence, we have a second proof (different from the one presented in [6]) for the fact that the deterministic klET0L languages contain non-recursive sets. Wätjen and Spilker have shown that the membership problem is decidable for klEDT0L systems with one table [21, Theorem 8]. (In fact, klED0L languages are context-sensitive, see [20].) Together with our result, we have a new proof for the fact that klEDT0L systems with more than one table can generate more languages than systems with only one table, see also [19]. It remains an open problem whether the membership problem is decidable for klE0L systems (not enforcing determinism) or not.

Observe how different open questions are posed by leftmost derivations of type 3 and free derivations in programmed grammars:

Corollary 12 • *$\mathcal{L}$(P–left-3, CF, ut) = $\mathcal{L}$(P–left-3, CF, ac), while the strictness of the inclusion $\mathcal{L}$(P, CF, ut) $\subseteq$ $\mathcal{L}$(P, CF, ac) is unknown, refer also to [12].*

- *$\mathcal{L}$(P–left-3, CF–λ, ut) $\subset$ $\mathcal{L}$(P–left-3, CF–λ, ac), while the strictness of the inclusion $\mathcal{L}$(P, CF $-$ λ, ut) $\subseteq$ $\mathcal{L}$(P, CF–λ, ac) is unknown.* □

Let us point the reader finally to various other open questions in this context. What is the exact relation between:

- $\mathcal{L}(\text{P–left-3}, \text{CF}{-}\lambda, \text{ut})$ and $\mathcal{L}(\text{P}, \text{CF}{-}\lambda, \text{ac})$ or $\mathcal{L}(\text{P}, \text{CF} - \lambda, \text{ut})$ (It is known that $\mathcal{L}(\text{P}, \text{CF} - \lambda, \text{ut})$ is not contained in $\mathcal{L}(\text{P–left-3}, \text{CF}{-}\lambda, \text{ut})$, since the latter language family has a decidable prefix problem, see [15].),
- $\mathcal{L}(k\text{lET0L}) \subseteq \mathcal{L}(\text{P}, \text{CF}, \text{ut})$ (also without erasing rules),
- $\mathcal{L}(k\text{lE0L})$ and $\mathcal{L}(\text{REC})$.

Let us remark that most of the open problems listed in [4, Open Problem 1.4.1], like the strictness of $\mathcal{L}(\text{P}, \text{CF}) \subseteq \mathcal{L}(\text{P–left-3}, \text{CF})$ (also without erasing rules), has been answered by a number of separation results quite recently [3].

Acknowledgments: The first author was supported by Deutsche Forschungsgemeinschaft grant DFG La 618/3-1/2. The second author was supported by Deutsche Forschungsgemeinschaft grant DFG Am 60/9-1. We gratefully acknowledge the presentation of this work by our colleague Markus Holzer at the conference "Developments in Language Theory" in 1997.

References

[1] W. Bucher, A. Ehrenfeucht, and D. Haussler. On total regulators generated by derivation relations. *Theoretical Computer Science*, 40:131–148, 1985.

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

[3] J. Dassow, H. Fernau, and Gh. Păun. On the leftmost derivation in matrix grammars. Work in progress, 1997.

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

[5] H. Fernau. Membership for 1-limited ET0L languages is not decidable. *J. Inf. Process. Cybern. EIK*, 30(4):191–211, 1994.

[6] H. Fernau. Membership for k-limited ET0L languages is not decidable. *Journal of Automata, Languages and Combinatorics*, 1:243–245, 1996.

[7] H. Fernau. On grammar and language families. *Fundamenta Informaticae*, 25(1):17–34, 1996.

[8] H. Fernau. On unconditional transfer. In W. Penczek and A. Szałas, editors, *MFCS'96*, volume 1113 of *LNCS*, pages 348–359, 1996.

[9] H. Fernau. Unconditional transfer in regulated rewriting. Technical Report WSI–96–21, Universität Tübingen (Germany), Wilhelm-Schickard-Institut für Informatik, 1996. A part of this report has been accepted for publication in Acta Informatica.

[10] G. Higman. Ordering by divisibility in abstract algebras. *Proceedings of the London Mathematical Society (3)*, 2(7):326–336, 1952.

[11] J. van Leeuwen. Effective constructions in well-partially-ordered monoids. *Discrete Mathematics*, 21:237–252, 1978.

[12] E. Moriya. Some remarks on state grammars and matrix grammars. *Information and Control*, 23:48–57, 1973.

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

[14] A. Peitz. k-uniform-links-limitierte Sprachen und Systeme. Studienarbeit, TU Braunschweig (Germany), Institut für Theoretische Informatik, September 1990.

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

[16] J. C. Shepherdson and H. E. Sturgis. Computability of recursive functions. *Journal of the Association for Computing Machinery*, 10:217–255, 1963.

[17] E. D. Stotskii. Control of the conclusion in formal grammars. *Problemy peredachi informacii; translated: Problems of information transmission*, 7(3):257–270, 1971 (Translation 1973).

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

[19] D. Wätjen. A weak iteration theorem for k-limited E0L systems. *J. Inf. Process. Cybern. EIK*, 28(1):37–40, 1992.

[20] D. Wätjen. k-limited ED0L languages are context-sensitive. *EATCS Bulletin*, 61:89–91, 1997.

[21] D. Wätjen and H. Spilker. Decidability results concerning k-limited ED0L systems. *Information Processing Letters*, 59:13–17, 1996.