Abstract Grammars Based on Transductions

Peter R.J. Asveld

Department of Computer Science, Twente University of Technology
P.O. Box 217, 7500 AE Enschede, The Netherlands

We study an abstract grammatical model in which the effect (or application) of a production — determined by a so-called transduction — plays the main part rather than the notion of production itself. Under appropriately chosen assumptions on the underlying family $\mathcal{T}$ of transductions, we establish elementary, decidability, and complexity properties of the corresponding family $\mathcal{L}(\mathcal{T})$ of languages generated by $\mathcal{T}$-grammars. These results are special instances of slightly more general properties of so-called Γ-controlled $\mathcal{T}$-grammars, since regular control does not increase the generating power of $\mathcal{T}$-grammars. In a Γ-controlled $\mathcal{T}$-grammar we restrict the iteration of $\mathcal{T}$-transductions to those sequences of transductions that belong to a given control language, taken from a family Γ of control languages.

1. Introduction

During the last decade some abstract grammatical models have been introduced, trying to provide a general framework in which all (or, at least, a considerable part of all) existing concrete grammars and rewriting systems are particular instances of these abstract models. The most well-known of these abstract models include grammar schemata, grammar forms, L forms, selective substitution grammars, and abstract (family of) grammars.

The notion of grammar scheme originates from [14] in which to each recursively enumerable trio $\mathcal{K}$ (i.e., to each r.e. language family closed under nondeterministic finite state transductions) a family of phrase structure grammars that generates $\mathcal{K}$ is associated. In context-free grammar [11] and L [21] form theory one mainly deals with normal forms for restricted classes of rewriting systems, viz. for (subfamilies of) context-free grammars and several types of L (or Lindenmayer) systems, respectively; cf. [33] and the references mentioned there. Selective substitution grammars have been motivated as a unifying approach to many different ways of rewriting strings [25, 20]. This model emphasizes the mechanism of the actual rewriting process, i.e., the details of transforming strings into other strings according to restrictions on the type and/or the application of the productions. This contrasts with the concept of abstract (family of) grammars [16] in which the notion of production as well as the mode of application are treated as free variables.

The abstract model that we study in the present paper shares this latter feature. But it emphasizes the effect of applying (abstract) productions without referring to productions themselves. More concretely, applying a production π is modeled by a mapping τ_π (called *transduction*) from strings to sets of strings, such that $\tau_\pi(x)$ is the set of all strings obtainable from the word x by applying π in some legitimate fashion. In dealing with parallel rewriting systems like ETOL systems π refers to a set of productions, called table, instead of a single production. Now by definition, τ_π characterizes completely the application of the production π. And since we restrict our attention to the effect of productions only, we need not refer to π at all, in contradistinction to the proposal in [16] where (π, τ_π) is taken as principal entity.

In our abstract approach a grammar $G = (V, \Sigma, U, S)$ consists of an alphabet V, a terminal alphabet Σ, an initial symbol S, and a finite set U of transductions. We assume that these transductions are taken from a given family $\mathcal{T}$ of transductions, and therefore we call G a $\mathcal{T}$-grammar. The language $L(G)$ generated by G consists of all terminal words obtainable by applying all sequences u from U^* to S. Equivalently, in order to obtain $L(G)$ we iterate the transductions from U, and at the end we intersect with Σ^*.

The iteration of transductions or mappings has been investigated previously in computer science; it has its roots in the work of Fleck [13], who studied pattern representation, and Wood [32], who was the first to consider iterated transductions as a grammatical concept in its own right. Continuations of this latter theme are [23, 5, 24] and, of course, the present paper.

Returning to our abstract model, a $\mathcal{T}$-grammar is in essence an ETOL system in which the finite substitutions are replaced by $\mathcal{T}$-transductions; cf. [32]. For $\mathcal{T}$ equal to some specific (sub)families of generalized sequential machine mappings this model has been studied in [13, 32, 23, 5, 24]. Analogous to the notion of Γ-controlled ETOL system [17, 22, 2] we introduce the concept of Γ-controlled $\mathcal{T}$-grammar: it consists of a $\mathcal{T}$-grammar $G = (V, \Sigma, U, S)$ provided with a control language C over U, taken from a given family Γ. The language $L(G;C)$ generated by such a controlled grammar consists of all terminal strings that can be obtained from S by applying those sequences of $\mathcal{T}$-transductions that belong to the control language C. Formal definitions and examples are given in Section 2.

In Section 3 we establish some elementary properties of (Γ-controlled) $\mathcal{T}$-grammars; viz. the fact that regular control does not increase the generating capacity of $\mathcal{T}$-grammars, and that the number of $\mathcal{T}$-transductions in such a grammar may be reduced to two or to one depending on the properties of $\mathcal{T}$. We also show inclusion relations between the families Γ of control languages, OUT($\mathcal{T}$) of output languages of $\mathcal{T}$-transductions, $\mathcal{L}(\mathcal{T})$ of languages generated by $\mathcal{T}$-grammars, and $\mathcal{L}(\mathcal{T};\Gamma)$ of languages generated by Γ-controlled $\mathcal{T}$-grammars.

Section 4 is devoted to the inclusion of $\mathcal{L}(\mathcal{T})$ and $\mathcal{L}(\mathcal{T};\Gamma)$ in the family of recursively enumerable languages and in the family of recursive languages, and to the decidability of the emptiness problem for (Γ-controlled) $\mathcal{T}$-grammars.

In Section 5 we establish upper bounds for the space and time complexity of the membership problem for $\mathcal{L}(\mathcal{T})$ and $\mathcal{L}(\mathcal{T};\Gamma)$, where $\mathcal{T}$ is a family of space- or time-bounded transductions, respectively.

Finally, Section 6 consists of concluding remarks, suggestions for further research, and open problems.

Some of the main results of this paper have been announced in [7].

2. Definitions

For all unexplained terminology from formal language theory we refer to the first few chapters of standard texts like [1, 18, 19, 27]. Some basic facts that we need from L systems theory and AFL-theory can be found in [26] and [15], respectively.

Crucial in our approach to introduce an abstract grammar is the notion of transduction.

Definition 2.1. Let V be an alphabet and let $\mathcal{P}(V^*)$ denote the power set of V^*. A *transduction* τ over V is a function $\tau: V^* \to \mathcal{P}(V^*)$ extended to languages by $\tau: \mathcal{P}(V^*) \to \mathcal{P}(V^*)$ with

$\tau(L) = \cup\{\tau(x) | x \in L\}$ for each language L over V. □

From $\varnothing \in \mathcal{P}(V^*)$ and Definition 2.1 it follows that $\tau(\varnothing) = \varnothing$. This property together with $L \cup \varnothing = \varnothing \cup L = L$ and $L\varnothing = \varnothing L = \varnothing$ for each language L over V enables us to let $\varnothing$ perform the part of blocking symbol (rejection symbol [26], or dead alley symbol) in order to abort undesirable sequences of rewriting steps; cf. [22]. Viz. we define $\tau(x) = \varnothing$ in case x should never yield a (terminal) string by an application of τ rather than $\tau(x) = F^{|x|}$ as usually in L systems theory [26], where F is a new nonterminal symbol such that τ preserves each occurrence of F in any string, and $|x|$ is the length of the string x.

Definition 2.2. Let f be an n-ary operation on languages. A family $\mathcal{T}$ of transductions is *closed* under (composition to the left with) f, if for all $\mathcal{T}$-transductions $\tau_1, \ldots, \tau_n$ over some alphabet V, there exists a $\mathcal{T}$-transduction τ over V such that for all x in V^*, $\tau(x) = f(\tau_1(x), ..., \tau_n(x))$. □

Examples of such operations which we will use in the sequel are isomorphism (''renaming of symbols''), union, and intersection with regular languages.

In many proofs we will construct a new grammar G_N from an old one G_O by attaching a finite amount of information to the symbols used in G_O. Then the transductions in G_N over this extended alphabet will be defined in terms of the old transductions of G_O using closure under isomorphism. Finally, we strip this additional information by applying an isomorphism in order to obtain words over the original alphabet. Therefore we make the following basic assumption throughout this paper.

Assumption 2.3. Henceforth $\mathcal{T}$ is a family of transductions that

(1) is closed under (composition to the left with) isomorphisms; cf. Definition 2.2,

(2) is closed under composition to the right with isomorphisms, i.e., for each $\mathcal{T}$-transduction τ_1 over V_1 and each isomorphism $i: V \to V_1$ there exists a $\mathcal{T}$-transduction τ over V such that $\tau(x) = \tau_1(i(x))$ for each x in V^*, and

(3) contains for each V the identity mapping over V. □

Notice that from Assumption 2.3 it follows that $\mathcal{T}$ also contains all isomorphisms.

We are now ready for the main formal definition.

Definition 2.4. Let $\mathcal{T}$ be a family of transductions. A *$\mathcal{T}$-grammar* $G = (V, \Sigma, U, S)$ consists of

- an alphabet V,
- a terminal alphabet Σ ($\Sigma \subseteq V$),
- an initial symbol S ($S \in V - \Sigma$), and
- a finite set U of $\mathcal{T}$-transductions over V.

The language $L(G)$ generated by G is defined by

$$L(G) = U^*(S) \cap \Sigma^* = (\cup\{\tau_p(\ldots(\tau_1(S))\ldots) | p \geq 0;\ \tau_i \in U,\ 1 \leq i \leq p\}) \cap \Sigma^*.$$

Let Γ be a family of languages. A *Γ-controlled $\mathcal{T}$-grammar* $(G;C) = (V, \Sigma, U, S, C)$ is a $\mathcal{T}$-grammar $(V, \Sigma, U, S,)$ provided with a control language C (with $C \subseteq U^*$) from Γ. The language $L(G;C)$ generated by $(G;C)$ is defined by

$$L(G;C) = C(S) \cap \Sigma^* = (\cup\{\tau_p(\ldots(\tau_1(S))\ldots) | \tau_1 \ldots \tau_p \in C\}) \cap \Sigma^*.$$

$\mathcal{L}(\mathcal{T})$ [$\mathcal{L}(\mathcal{T};\Gamma)$, respectively] is the family of languages generated by [Γ-controlled] $\mathcal{T}$-grammars, and $\mathcal{L}(\mathcal{T};m)$ [respectively $\mathcal{L}(\mathcal{T};\Gamma;m)$] is the subfamily of languages generated by [Γ-controlled] $\mathcal{T}$-grammars that possess at most m ($m \geq 1$) $\mathcal{T}$-transductions. □

Examples 2.5. (1) Let HOM and FINSUB be the families of all homomorphisms and of all finite substitutions, respectively. Then $\mathcal{L}(\text{HOM}) = \text{EDTOL}$ and $\mathcal{L}(\text{FINSUB}) = \text{ETOL}$. For Γ-controlled variations, see e.g. [2, 3, 10, 17, 22].

(2) Let $\mathcal{K}$ be a family of languages closed under isomorphism, and containing all singleton languages. A (*nondeterministic*) *$\mathcal{K}$-substitution* or *n$\mathcal{K}$-substitution* σ is a mapping $\sigma: V \to \mathcal{K}$ extended to words over V by $\sigma(\lambda) = \{\lambda\}$ (λ is the empty word), $\sigma(\alpha_1 ... \alpha_n) = \sigma(\alpha_1) ... \sigma(\alpha_n)$, $\alpha_i \in V$ ($1 \leq i \leq n$) and to languages over V by $\sigma(L) = \cup\{\sigma(x) | x \in L\}$ for each $L \subseteq V^*$. A *deterministic $\mathcal{K}$-substitution* or *d$\mathcal{K}$-substitution* σ is a mapping $\sigma: V \to \mathcal{K}$ too. But it is extended to words x over V by

$$\sigma(x) = \{h(x) | h \text{ is a homomorphism with } h(\alpha) \in \sigma(\alpha),\ \alpha \in V\},$$

and to languages L over V by $\sigma(L) = \cup\{\sigma(x) | x \in L\}$.

Let d$\mathcal{K}$-SUB [n$\mathcal{K}$-SUB] denote the family of all [non]deterministic $\mathcal{K}$-substitutions. Then $\mathcal{L}(\text{d}\mathcal{K}\text{-SUB}) = \eta(\mathcal{K})$ [respectively, $\mathcal{L}(\text{n}\mathcal{K}\text{-SUB}) = H(\mathcal{K})$], i.e., the family of languages generated by [non]deterministic $\mathcal{K}$-iteration grammars. Cf. [2, 3, 4, 8, 9], where the Γ-controlled case is also considered.

(3) A [λ-*free*] *nondeterministic generalized sequential machine with accepting states* or *NGSM* [λ*NGSM*] $\tau = (Q, \Delta_1, \Delta_2, \delta, q_0, Q_F)$ consists of

- a set of states Q with initial state q_0, and a set Q_F of final states ($q_0 \in Q$; $Q_F \subseteq Q$),
- an input alphabet Δ_1 and an output alphabet Δ_2,
- a function δ from $Q \times \Delta_1$ into the finite subsets of $Q \times \Delta_2^*$ [$Q \times \Delta_2^+$].

The function δ is extended from $Q \times \Delta_1^*$ into the finite subsets of $Q \times \Delta_2^*$ by

(i) $\delta(q, \lambda) = \{(q, \lambda)\}$,

(ii) $\delta(q, x\alpha) = \{(q', y) | y = y_1 y_2$ and for some $q'' \in Q$, $(q'', y_1) \in \delta(q, x)$ and $(q', y_2) \in \delta(q'', \alpha)\}$, where $q \in Q$, $\alpha \in \Delta_1$, $x \in \Delta_1^*$.

Each [λ-free] NGSM τ induces a transduction $\tau: \mathcal{P}(\Delta_1^*) \to \mathcal{P}(\Delta_2^*)$, called [$\lambda$-*free*] *NGSM mapping*, defined by

- $\tau(x) = \{y | (q, y) \in \delta(q_0, x)$ for some $q \in Q_F\}$, for each word x in Δ_1^*, and
- $\tau(L) = \cup\{\tau(x) | x \in L\}$, for each language L over Δ_1.

A NGSM [λNGSM] τ is called *deterministic* or *DGSM* [λ*DGSM*] if δ is a function from $Q \times \Delta_1$ into $Q \times \Delta_2^*$ [$Q \times \Delta_2^+$]. By NGSM [λNGSM] we also denote the family of [λ-free] NGSM mappings, and similarly we use DGSM [λDGSM] in the deterministic case. The language families $\mathcal{L}(\mathcal{T};\Gamma)$ and $\mathcal{L}(\mathcal{T})$ with $\mathcal{T}$ equal to λNGSM, λDGSM, NGSM, and DGSM have been investigated in [13, 32, 23, 5, 24, 12]; see [24] in particular, where e.g. the family of context-free languages is characterized by $\mathcal{L}(\mathcal{T})$ for some family $\mathcal{T}$ of restricted NGSM mappings. □

All the examples in 2.5 are transductions in the sense of Definition 2.1, and they all satisfy Assumption 2.3.

In proofs we will use the following convention. If we define a transduction τ by means of n mutually exclusive cases, characterized by n predicates $P_1(x), \ldots, P_n(x)$, then we assume that for the usually omitted $(n+1)^{st}$ otherwise case when none of the $P_i(x)$ is true, we have $\tau(x) = \varnothing$. For example ($n = 2$), when we write

$$\tau(x) = X_1 \qquad \text{if } P_1(x)$$
$$\tau(x) = X_2 \qquad \text{if } P_2(x)$$

we tacitly assume that

$$\tau(x) = \varnothing \qquad \text{otherwise.}$$

We consider two languages to be equal if they only differ at most by the empty word. Equality of language families is defined correspondingly.

3. Elementary Properties

Firstly, we establish the equivalence with respect to generating power of regularly controlled $\mathcal{T}$-grammars and uncontrolled $\mathcal{T}$-grammars. Let REG denote the family of regular languages.

Theorem 3.1. $\mathcal{L}(\mathcal{T};\text{REG}) = \mathcal{L}(\mathcal{T})$.

Proof: The inclusion $\mathcal{L}(\mathcal{T}) \subseteq \mathcal{L}(\mathcal{T};\text{REG})$ is obvious, since U^* is regular for each U.

Conversely, consider the REG-controlled $\mathcal{T}$-grammar $(G;R) = (V, \Sigma, U, S, R)$ where R is accepted by some deterministic finite automaton (Q, U, q_0, δ, Q_F), where Q is the set of states, U is the input alphabet, q_0 is the initial state, $\delta: Q \times U \to Q$ is the transition function, and Q_F is the set of final states. Define for each p in Q the isomorphism $i_p: V \to Q \times V$ by $i_p(\alpha) = (p, \alpha)$ for each α in V, and V_p by $V_p = \{(p, \alpha) | \alpha \in V\}$.

Consider the uncontrolled $\mathcal{T}$-grammar $G_0 = (V_0, \Sigma, U_0, S_0)$ with $V_0 = V \cup Q \times V$, $S_0 = (q_0, S)$, $U_0 = \{\tau_f | f \in Q_F\} \cup \{\tau_0 | \tau \in U\}$ where for each τ in U, each p in Q, and each x in V_p^*,

$$\tau_0(x) = i_q(\tau(i_p^{-1}(x))) \qquad \text{if and only if } \delta(p, \tau) = q,$$

and for each f in Q_F,

$$\tau_f(x) = \{x\} \qquad \text{if and only if } x \in V_f^*.$$

By this construction we have $L(G_0) = L(G;R)$, and hence $\mathcal{L}(\mathcal{T};\text{REG}) \subseteq \mathcal{L}(\mathcal{T})$. □

This result is a straightforward generalization of similar facts concerning the transductions mentioned in Examples 2.5(1)-(3) [2, 5, 8, 9, 17, 22]. But it also applies to, e.g., ETIL systems; cf. [26] for a definition.

Next we will reduce the number of transductions in a (controlled) $\mathcal{T}$-grammar, for which we need some additional terminology.

For each transduction τ over V, the *support* $Sup(\tau)$ of τ is $Sup(\tau) = \{x \in V^+ | \tau(x) \neq \varnothing\}$. A family $\mathcal{T}$ is closed under *disjoint union* if for all τ_1 and τ_2 in $\mathcal{T}$ with $Sup(\tau_1) \cap Sup(\tau_2) = \varnothing$, there exists a τ in $\mathcal{T}$ such that $\tau(x) = \tau_1(x) \cup \tau_2(x)$ for each x in V^*.

Let $U = \{\tau_1, \ldots, \tau_m\}$ and $U_0 = \{\sigma, \tau\}$. Then for each $m \geq 2$, we define the λ-free homomorphism $h_m: U^* \to U_0^*$ by $h_m(\tau_k) = \sigma^k \tau$ $(1 \leq k \leq m)$.

Theorem 3.2. *Let $\mathcal{T}$ be closed under disjoint union. Then $\mathcal{L}(\mathcal{T};2) = \mathcal{L}(\mathcal{T};m) = \mathcal{L}(\mathcal{T})$ and $\mathcal{L}(\mathcal{T};\Gamma;2) = \mathcal{L}(\mathcal{T};\Gamma;m) = \mathcal{L}(\mathcal{T};\Gamma)$, $m \geq 2$, provided Γ is closed under h_m for each $m \geq 2$.*

Proof: Consider a Γ-controlled $\mathcal{T}$-grammar $(G;C) = (V, \Sigma, U, S, C)$ with $U = \{\tau_1,...,\tau_m\}$. Define for each k $(1 \leq k \leq m)$ an isomorphism i_k by $i_k(\alpha) = \alpha_k$ for α in V (Each α_k is assumed to be a new symbol). We construct a Γ-controlled $\mathcal{T}$-grammar $(G_0;C_0) = (V_0, \Sigma, U_0, S, C_0)$ where $U_0 = \{\sigma, \tau\}$, $C_0 = h_m(C)$, $V_0 = V \cup \{i_k(\alpha) | \alpha \in V;\ 1 \leq k \leq m\}$, and

$$\sigma : (V \cup \{i_k(\alpha) | \alpha \in V;\ 1 \leq k \leq m-1\}) \rightarrow \{i_k(\alpha) | \alpha \in V;\ 1 \leq k \leq m\}$$

is the isomorphism defined by

$$\begin{aligned} &\sigma(\alpha) = \alpha_1 && \alpha \text{ in } V, \\ &\sigma(\alpha_k) = \alpha_{k+1} && \alpha \text{ in } V;\ 1 \leq k \leq m-1. \end{aligned}$$

For each x in $\{i_k(\alpha) | \alpha \in V;\ 1 \leq k \leq m\}^*$, $\tau(x)$ equals the following disjoint union

$$\tau(x) = \tau_1(i_1^{-1}(x)) \cup \cdots \cup \tau_k(i_k^{-1}(x)) \cup \cdots \cup \tau_m(i_m^{-1})x)).$$

It is straightforward to show that $L(G_0;C_0) = L(G;C)$, and hence for each $m \geq 2$ we have $\mathcal{L}(\mathcal{T};\Gamma;m) \subseteq \mathcal{L}(\mathcal{T};\Gamma;2)$, while the converse inclusion is trivial. Note that $\mathcal{L}(\mathcal{T};\Gamma)$ equals $\cup\{\mathcal{L}(\mathcal{T};\Gamma;m) | m \geq 0\}$.

The statement in the uncontrolled case simply follows from the observation that $L(G_0) = U_0^*(S) \cap \Sigma^* = U^*(S) \cap \Sigma^* = L(G)$. □

This property clearly extends the well-known fact that the number of substitutions in (un)controlled EDTOL, ETOL systems [26], d$\mathcal{K}$- and n$\mathcal{K}$-iteration grammars [2, 8] can be reduced to two. Note that it also applies to DGSM and λDGSM mappings.

Let $g_m: \{\tau_1,...,\tau_m\}^* \rightarrow \tau^*$ be the length-preserving homomorphism $g_m(\tau_k) = \tau$ $(1 \leq k \leq m)$. By a further restriction on the family $\mathcal{T}$ a reduction to a single transduction is possible.

Theorem 3.3. *Let $\mathcal{T}$ be closed under union. Then $\mathcal{L}(\mathcal{T};1) = \mathcal{L}(\mathcal{T};m) = \mathcal{L}(\mathcal{T})$ and $\mathcal{L}(\mathcal{T};\Gamma;1) = \mathcal{L}(\mathcal{T};\Gamma;m) = \mathcal{L}(\mathcal{T};\Gamma)$, $m \geq 1$, provided Γ is closed under g_m for each $m \geq 1$.*

Proof: Starting from $(G;C) = (V, \Sigma, U, S, C)$ and $U = \{\tau_1,...,\tau_m\}$ we construct $(G_0;C_0) = (V, \Sigma, U_0, S, C_0)$ where $C_0 = g_m(C)$, $U_0 = \{\tau\}$ with $\tau(x) = \tau_1(x) \cup \cdots \cup \tau_m(x)$ for each x in V^*. Then $L(G_0;C_0) = L(G;C)$, and $L(G_0) = L(G)$. □

This directly implies a result from [32], viz. $\mathcal{L}(\mathcal{T}) = \mathcal{L}(\mathcal{T};1)$ for $\mathcal{T}$ equal to NGSM or λNGSM.

Finally, we establish some simple inclusion relationships between language families for which we need the following notation and terminology.

A family Γ is closed under *right marking* if for each language C (over some U), and each symbol \$ not in U, the language $C\{\$\}$ — for which we usually write $C\$$ — is in Γ.

For each family $\mathcal{T}$ of transductions, OUT($\mathcal{T}$) is the family of output languages defined by

$$\text{OUT}(\mathcal{T}) = \{\tau(x) | \tau \text{ is a } \mathcal{T}\text{-transduction over } V \text{ for some } V;\ x \in V^*\}.$$

Let λHOM be the family of λ-free homomorphisms.

Theorem 3.4. *If $\mathcal{T} \supseteq$ λHOM and if Γ is closed under right marking, then*

(1) $\Gamma \subseteq \mathcal{L}(\mathcal{T};\Gamma)$,

(2) $\mathrm{OUT}(\mathcal{T}) \subseteq \mathcal{L}(\mathcal{T};\Gamma)$,

(3) $\mathrm{OUT}(\mathcal{T}) \subseteq \mathcal{L}(\mathcal{T})$,

(4) *if in addition the family* Γ *of control languages is also closed under union (or concatenation) and Kleene* $*$*, then* $\mathcal{L}(\mathcal{T}) \subseteq \mathcal{L}(\mathcal{T};\Gamma)$.

Proof: (1) Let $C \subseteq U^*$ for some U. Consider $(G_0;C_0) = (V,U,U_0,S,C_0)$ with $C_0 = C\tau_0$, $U_0 = U \cup \{\tau_0\}$, $V = U \cup \{\tau' \mid \tau \in U\} \cup \{S\}$, while all transductions in U_0 are λ-free homomorphisms defined by

$$\begin{aligned} &\tau(S) = \tau' && \tau \in U,\\ &\tau(\sigma') = \sigma\tau' && \sigma,\tau \in U,\\ &\tau(\alpha) = \alpha && \alpha,\tau \in U,\\ &\tau_0(\sigma') = \sigma && \sigma \in U,\\ &\tau_0(\sigma) = \sigma && \sigma \in U. \end{aligned}$$

Then $L(G_0;C_0) = C$, and hence $\Gamma \subseteq \mathcal{L}(\mathcal{T};\Gamma)$.

(2) Take for each τ over V and for each x in V^*, any nonempty C over some U such that U does not contain τ or τ_x. Consider $(G_{\tau x};C_{\tau x}) = (V_{\tau x},V,U_{\tau x},S,C_{\tau x})$ with $U_{\tau x} = U \cup \{\tau,\tau_x\}$, $V_{\tau x} = V \cup \{S\}$ where S is new, $C_{\tau x} = C\tau_x\tau$, each transduction in U is taken equal to the identity mapping on $V_{\tau x}$, and τ_x is the homomorphism defined by

$$\begin{aligned} &\tau_x(S) = x && \\ &\tau_x(\alpha) = \alpha && \text{for each } \alpha \text{ in } V. \end{aligned}$$

Then $L(G_{\tau x};C_{\tau x}) = \tau(x)$, and thus $\mathrm{OUT}(\mathcal{T}) \subseteq \mathcal{L}(\mathcal{T};\Gamma)$.

(3) follows from (2) and Theorem 3.1 with $\Gamma = \mathrm{REG}$.

(4) Let $G = (V,\Sigma,U,S)$ be a $\mathcal{T}$-grammar with $U = \{\tau_1,...,\tau_m\}$ and let $C \subseteq U_0^*$ be any nonempty language in Γ such that $U_0 \cap U = \varnothing$. If Γ is closed under union [concatenation] and Kleene $*$, then the control language $C_1 = (C\tau_1 \cup C\tau_2 \cup \cdots \cup C\tau_m)^*$ [$C_1 = ((C\tau_1)^*(C\tau_2)^* \cdots (C\tau_m)^*)^*$, respectively] is in Γ. Consider the Γ-controlled $\mathcal{T}$-grammar $(G_1;C_1) = (V,\Sigma,U_1,S,C_1)$ with $U_1 = U \cup U_0$, where each τ in U_0 is the identity transduction over V. Then $L(G_1;C_1) = L(G)$, and hence $L(G) \in \mathcal{L}(\mathcal{T};\Gamma)$. □

Theorem 3.4 generalizes the analogous statements for n$\mathcal{K}$-iteration grammars proved in [2]. Clearly, it also applies to d$\mathcal{K}$-iteration grammars, iterated (λ-free) NGSM and iterated (λ-free) DGSM mappings.

4. Recursively Enumerable Languages and Decidability

A transduction τ over some alphabet V is called *partial recursive* if for each x in V^*, $\tau(x)$ is a recursively enumerable (or r.e.) language. Let PRECtr be the family of partial recursive transductions. We call a transduction τ over V *monotonic* if for each x and y in V^*, $y \in \tau(x)$ implies that $|y| \geq |x|$. A transduction τ over V is called *recursive* if for each x in V^*, $\tau(x)$ is a recursive language. Let MRECtr be the family of monotonic recursive transductions. RE and REC denote the families of recursively enumerable and of recursive languages, respectively.

The proof of the following statement is straightforward; therefore it is left to the reader.

Proposition 4.1.

(1) *If* $\mathcal{T} \subseteq$ PRECtr *and* $\Gamma \subseteq$ RE, *then* $\mathcal{L}(\mathcal{T}) \subseteq$ RE *and* $\mathcal{L}(\mathcal{T};\Gamma) \subseteq$ RE.

(2) *If* $\mathcal{T} \subseteq$ MRECtr, *then* $\mathcal{L}(\mathcal{T}) \subseteq$ REC. □

Corollary 4.2. $\mathcal{L}$(PRECtr) = RE, *and* $\mathcal{L}$(MRECtr) = REC.

Proof: It follows from 4.1, 3.4(3) and the equalities OUT(PRECtr) = RE and OUT(MRECtr) = REC. □

However, RE can be obtained by iterating much simpler transductions. The following theorem is a combination of results from [13, 32, 23, 5, 24] together with Theorems 3.1-3.3.

Theorem 4.3. *If* Γ *satisfies* $\mathrm{REG} \subseteq \Gamma \subseteq \mathrm{RE}$, *then*
$\mathcal{L}(\mathrm{NGSM};\Gamma) = \mathcal{L}(\mathrm{NGSM};\mathrm{REG}) = \mathcal{L}(\mathrm{NGSM}) = \mathcal{L}(\mathrm{NGSM};1) =$
$\mathcal{L}(\mathrm{DGSM};\Gamma) = \mathcal{L}(\mathrm{DGSM};\mathrm{REG}) = \mathcal{L}(\mathrm{DGSM}) = \mathcal{L}(\mathrm{DGSM};2) = \mathrm{RE}$. □

The question remains whether $\mathcal{L}$(DGSM;1) = RE; cf. [32]. Note also that in 4.1(2) the controlled variant is absent. This is due to the following characterization of RE; cf. Theorem 2.2 in [2].

Theorem 4.4. *If* Γ *and* $\mathcal{T}$ *satisfy*

(1) $\lambda\mathrm{HOM} \subseteq \mathcal{T} \subseteq$ PRECtr,

(2) $\{h(C) \mid C \in \Gamma$; *h is an arbitrary homomorphism*$\} = \mathrm{RE}$,

then $\mathcal{L}(\mathcal{T};\Gamma) = \mathrm{RE}$.

Proof: The inclusion $\mathcal{L}(\mathcal{T};\Gamma) \subseteq \mathrm{RE}$ follows from Proposition 4.1(1).

Conversely, let $L_0 \subseteq \Sigma^*$ be any language in RE. Define the alphabet $\Sigma' = \{\alpha' \mid \alpha \in \Sigma\}$ where each α' is a new symbol, and the length-preserving homomorphism h_0 by $h_0(\alpha) = h_0(\alpha') = \alpha$. Then $L_0' = h_0^{-1}(L_0) \cap \Sigma^*\Sigma'$ is also recursively enumerable. Now 4.4(2) implies the existence of a language $C \subseteq U^*$ in Γ for some U, and of a homomorphism $h: U^* \to (\Sigma \cup \Sigma')^*$ such that $h(C) = L_0'$.

Consider the Γ-controlled $\mathcal{T}$-grammar $(G;C) = (V, \Sigma, U, S, C)$ with $V = \Sigma \cup \{S\}$, and each τ in U is a λ-free homomorphism defined by

$$\begin{aligned} \tau(S) &= h(\tau)S && \text{if and only if } h(\tau) \in \Sigma^*, \\ \tau(S) &= h_0(h(\tau)) && \text{if and only if } h(\tau) \in \Sigma^*\Sigma', \\ \tau(\alpha) &= \alpha && \text{if and only if } \alpha \in \Sigma. \end{aligned}$$

Then $L(G;C) = L_0$ (modulo λ), and therefore $\mathrm{RE} \subseteq \mathcal{L}(\mathcal{T};\Gamma)$. □

Essentially, the proof is based on the idea that long control words, i.e., long derivations, may yield relatively short terminal strings in the end; in this way it is possible to simulate an erasing homomorphism. Since each transduction in MRECtr is monotonic, applying 4.4 with $\mathcal{T}$ equal to MRECtr means that arbitrary long length-preserving subderivations may occur in Γ-controlled MRECtr-derivations. In order to obtain a ''controlled'' analogue of Theorem 4.1(2) the length of those length-preserving subderivations should be bounded uniformly; cf., e.g., Theorem 3.4 in [10], Lemma 2.3 in [4], and Theorem 3.5 in [6]. The conditions in Definition 4.5 and Lemma 4.6 do guarantee such a uniform bound. They also play a key role in the next section.

Definition 4.5. A transduction τ over some alphabet V is called *locally context-independent* if

(1) τ is monotonic, and

(2) τ is context-independent in length-preserving applications, i.e., for all x_i, y_i in V^* with $|x_i| = |y_i|$ $(i = 1, 2, 3)$, $y_1y_2y_3 \in \tau(x_1x_2x_3)$ implies $y_1y_3y_2 \in \tau(x_1x_3x_2)$. □

Henceforth we assume that all closure properties are effective.

Lemma 4.6. *Let $\mathcal{T}$ be a family of locally context-independent transductions, and let $\mathcal{T}$ contain for all alphabets V the length-preserving finite substitutions*

$$\tau(\alpha) = W \qquad\qquad \alpha \text{ in } V, \quad W \subseteq V.$$

(1) *Let Γ be a family closed under finite substitutions and under intersection with regular languages, and let $(G;C) = (V, \Sigma, U, S, C)$ be a Γ-controlled $\mathcal{T}$-grammar. Then we can effectively construct a Γ-controlled $\mathcal{T}$-grammar $(G_0;C_0) = (V, \Sigma, U_0, S, C_0)$ such that*

(1.1) *$L(G_0;C_0) = L(G;C)$, and*

(1.2) *for each x in $L(G_0;C_0)$, there is a control word $\tau_1 \ldots \tau_p$ in C_0 such that $x \in \tau_p \ldots \tau_1(S)$ and $p \leq 2|x|$.*

(2) *For each $\mathcal{T}$-grammar $G = (V, \Sigma, U, S)$, we can effectively construct a $\mathcal{T}$-grammar $G_0 = (V, \Sigma, U_0, S)$ such that*

(2.1) *$L(G_0) = L(G)$, and*

(2.2) *for each x in $L(G_0)$, there exists a word $\tau_1 \ldots \tau_p$ in U^* such that $x \in \tau_p \ldots \tau_1(S)$, and $p \leq 2|x|$.*

Proof: (1) The construction is similar to the one in Lemma 2.3 of [4]. Viz. we add new control words to C such that the corresponding derivations possess the property that each length-preserving step in such a derivation is immediately followed by a length-increasing step.

If $V = \{\alpha_1, \ldots, \alpha_k\}$ for some $k \geq 1$, then we define $U_0 = U \cup \{[\tau, q] | \tau \in U, q \in Q\}$ with $Q = \{< X_1, \ldots, X_k > | X_i \subseteq V,\ 1 \leq i \leq k\}$, and $C_0 = \sigma(C)$ where $\sigma = (Q, U, U_0, \delta, q_0, Q_F)$ is an NGSM with $q_0 = < \{\alpha_1\}, \ldots, \{\alpha_k\} >$, $Q_F = \{q_0\}$, while δ is defined by

$$\delta(< X_1, .., X_k >, \tau) = \{(< \tau(X_1) \cap V, \ldots, \tau(X_k) \cap V >, \lambda), (q_0, [\tau, < X_1, \ldots, X_k >])\} \cup$$

$$\cup \{(q_0, \tau) | < X_1, \ldots, X_k > = q_0\}.$$

(By Lemma 9.3 of [18] we have $C_0 \in \Gamma$. Notice that $C \subseteq C_0$).

We outline the way in which the new additional control words are obtained by means of σ from C, as well as the effect of these new control words. Consider an arbitrary derivation D according to $(G;C)$. At each step in D, determined by the application of some $\mathcal{T}$-transduction τ, one of the following three possibilities applies (cf. the definition of δ):

Case (a): This application of τ is length-increasing.

The corresponding transition in σ is the identity transition: $(q_0, \tau) \in \delta(q_0, \tau)$. This case does not give rise to adding new control words.

Case (b): This application of τ is length-preserving and the next step in D will also be length-preserving.

The corresponding occurrence of τ in the control word is erased, and the length-preserving context-independent effect (cf. Definition 4.5) of τ is stored by means of changing the state of σ

from $< X_1,...,X_k >$ to $< \tau(X_1) \cap V,..., \tau(X_k) \cap V >$.

Case (c): This application of τ is length-preserving but either the next step in D will be length-increasing, or this application of τ is the last step in D.

In the old control word we replace the corresponding occurrence of τ by $[\tau, < X_1,...,X_k >]$ where $< X_1,...,X_k >$ is the current state of σ in which the ultimate length-preserving effect of a consecutive sequence of erased transductions (cf. Case (b)) has been stored. This new transduction $[\tau, < X_1,...,X_k >]$ is a length-preserving finite substitution defined by

$$[\tau, < X_1,...,X_k >](\alpha_i) = \tau(X_i) \cap V \qquad \text{for each } i \ (1 \le i \le k).$$

(2) Define $U_0 = U \cup \{\tau_u \mid u \in U^+\}$ with for each u in U^+, τ_u is the length-preserving substitution defined by $\tau_u(\alpha) = u(\alpha) \cap V$ for each α in V. Then U_0 is finite, because there are only a finite number of length-preserving substitutions over V. □

We are now ready for the controlled variant of Proposition 4.1(2).

Theorem 4.7. *Let* Γ *and* $\mathcal{T}$ *satisfy the assumptions of Lemma 4.6. If* $\Gamma \subseteq$ REC *and* $\mathcal{T} \subseteq$ MRECtr, *then* $\mathcal{L}(\mathcal{T};\Gamma) \subseteq$ REC.

Proof: The algorithm of Figure 1 determines whether a word x belongs to $L(G_0;C_0)$; cf. Lemma 4.6. Since after execution of an **accept**- or **reject**-statement the algorithm is supposed to halt, termination is guaranteed for each input x. Hence $L(G_0;C_0)$ is recursive. □

```
read x ;
if x ∉ Σ+ then reject else
      for all u in C0 with |u| ≤ 2|x| do
            if x ∈ u(S) then accept
      od;
      reject
fi.
```

Figure 1.

Proposition 4.1(2) and Theorem 4.7 enable us to improve upon the decidability of the membership problem for λ-free (non)deterministic iteration grammars; cf. [2, 3].

Corollary 4.8. *Let* $\mathcal{K}$ *be a family of* λ*-free languages that contains all finite alphabets.* [*Let* Γ *be a subfamily of the recursive languages, closed under finite substitution and intersection with regular languages*]. *If* $\mathcal{K} \subseteq$ REC, *then the membership problem for* [Γ-*controlled*] *(non)deterministic* $\mathcal{K}$*-iteration grammars is decidable.* □

When $\mathcal{K}$ is not λ-free one may first use Theorem 3.1 from [2] or Theorem 3.2 from [8] to obtain an equivalent (controlled) λ-free $\mathcal{K}$-iteration grammar. However, the constructive version of both these theorems requires the decidability of the emptiness problem for $\mathcal{K}$; cf. [2, 3].

We conclude this section with a set of conditions (cf. Definition 4.9 and Theorem 4.12) that imply the decidability of the emptiness problem for Γ-controlled $\mathcal{T}$-grammars.

Definition 4.9. A family $\mathcal{T}$ of transductions is *locally regular* if for each finite set U of $\mathcal{T}$-transductions over some alphabet V, and for each subset Σ of V, there exists an equivalence relation $\equiv$ over V^* such that

(1) $\equiv$ is decidable and of finite index,

(2) for each τ in U and for all x_1, x_2 in V^*, if $x_1 \equiv x_2$, then either $\tau(x_1) = \emptyset = \tau(x_2)$, or there exist $y_i \in \tau(x_i)$ for $i = 1, 2$ such that $y_1 \equiv y_2$,

(3) Σ^* equals the union of a finite number of equivalence classes with respect to $\equiv$. □

In Definition 4.10 and Lemma 4.11 we extend the notion of Szilard language to $\mathcal{T}$-grammars; cf. [27, 31].

Definition 4.10. Let $G = (V, \Sigma, U, S)$ be a $\mathcal{T}$-grammar. The *Szilard language* of G is the language $Sz(G)$ over U defined by

$$Sz(G) = \{\tau_1 \dots \tau_n \mid \tau_n(\dots(\tau_1(S))\dots) \cap \Sigma^* \neq \emptyset;\ n \geq 0\}.$$ □

For each family $\mathcal{T}$ of transductions we call the question to decide whether $\tau(x) = \emptyset$, where τ is a $\mathcal{T}$-transduction over some alphabet V and x is a word in V^*, the *emptiness problem* for $\mathcal{T}$.

Lemma 4.11. *If $\mathcal{T}$ is a locally regular family of transductions, then for each $\mathcal{T}$-grammar $G = (V, \Sigma, U, S)$ the Szilard language $Sz(G)$ is regular. Moreover, if $\mathcal{T}$ is closed under intersection with regular languages and if the emptiness problem for $\mathcal{T}$ is decidable, then $Sz(G)$ can be constructed effectively.*

Proof: For each x in V^*, let $[x]$ be the equivalence class with respect to $\equiv$ that contains x. Define the right-linear grammar $G_0 = (V_0, U, P, S_0)$ with $V_0 - U = \{[x] \mid x \in V^*\}$, $S_0 = [S]$, and

$$P = \{[x] \to \lambda \mid [x] \subseteq \Sigma^*\} \cup \{[x] \to \tau[y] \mid y \in \tau(x);\ x, y \in V^+\}.$$

Since $\mathcal{T}$ is locally regular, V_0 and P are finite. Then $[S] \Rightarrow^* \tau_1 \dots \tau_n$ according to G_0 for some $n \geq 0$ if, and only, if $\tau_n(\dots(\tau_1(S))\dots) \cap \Sigma^* \neq \emptyset$. Consequently, $Sz(G) = L(G_0)$, and $Sz(G)$ is regular.

Since $\equiv$ is of finite index, each equivalence class with respect to $\equiv$ is regular. Therefore the fact that $\mathcal{T}$ is closed under intersection with regular languages implies that we can reduce effectively the question ''$y \in \tau(x)$?'' (cf. the definition of P) to ''$\tau(x) \cap [y] = \emptyset$?'', i.e., to the emptiness problem for $\mathcal{T}$. □

Theorem 4.12. *Let Γ and $\mathcal{T}$ be closed under intersection with regular languages, and let the emptiness problem be decidable for Γ and $\mathcal{T}$. If $\mathcal{T}$ is locally regular, then the emptiness problem for $\mathcal{L}(\mathcal{T};\Gamma)$ and for $\mathcal{L}(\mathcal{T})$ is decidable.*

Proof: By Lemma 4.11 the language $Sz(G)$ is regular and it can be constructed effectively for each Γ-controlled $\mathcal{T}$-grammar $(G;C) = (V, \Sigma, U, S, C)$. Then $C \cap Sz(G)$ is in Γ. By the definition of $Sz(G)$ we have that $L(G) = \emptyset$ if, and only, if $Sz(G) = \emptyset$. Hence $L(G;C) = \emptyset$ if, and only, if $C \cap Sz(G) = \emptyset$. □

Let $\mathcal{K}$ be a family of languages, closed under intersection with regular languages, for which the emptiness problem is decidable. Then the families d$\mathcal{K}$-SUB and n$\mathcal{K}$-SUB of Example 2.5(2) are locally regular with $x \equiv y$ if, and only, if $alph(x) = alph(y)$; cf. [2, 3] (For each word w, $alph(w)$ is the set of symbols that do occur in w).

On the other hand Theorem 4.3 implies that neither NGSM nor DGSM is locally regular. The same conclusion holds for λNGSM and λDGSM (cf. Theorem 5.4 below).

5. Complexity

In this section we determine upper bounds for the space and time complexity of languages generated by (controlled) $\mathcal{T}$-grammars; viz. Theorems 5.2, 5.5 and 5.8.

Throughout this section ''function'' means a monotone increasing function f over the natural numbers satisfying $f(n) \geq n$ for each $n \geq 0$.

Definition 5.1. Let for each function f, DSPACETR(f) [NSPACETR(f), respectively] be the family of those transductions τ that satisfy

(1) τ is locally context-independent, and

(2) there exists a [non]deterministic algorithm that can decide a query ''$y \in \tau(x)$?'' for each x and y within space $f(|y|)$. □

As usual, for each function f, DSPACE(f) [NSPACE(f), respectively] is the family of languages accepted by [non]deterministic multi-tape Turing machines that use at most $f(n)$ tape squares on each work tape during a computation on a input of length n.

Theorem 5.2. *Let f be a function.*

(1) *If $\mathcal{T} \subseteq$ NSPACETR(f), then $\mathcal{L}(\mathcal{T}) \subseteq$ NSPACE(f).*

(2) $\mathcal{L}$(NSPACETR(f)) = NSPACE(f).

(3) *Let Γ be a family closed under finite substitution and under intersection with regular languages. If $\Gamma \subseteq$ NSPACE(f), $\mathcal{T} \subseteq$ NSPACETR(f), and $f(2n) \leq c.f(n)$ for some constant c and for each $n \in \mathbb{N}$, then $\mathcal{L}(\mathcal{T};\Gamma) \subseteq$ NSPACE(f).*

Proof: (1) Consider the algorithm in Figure 2; remove the assignments in which the variable *control* is involved, and replace the last statement by **accept**.

Then each step in this modified algorithm requires at most linear space, except the test ''$z \in \tau(y)$'' for which we need $f(|z|) \leq f(|x|)$ space. Thus for $|x| = n$, the total amount of space is $O(n + f(n)) = O(f(n))$.

(2) From (1) with $\mathcal{T}$ equal to NSPACETR(f) it follows that $\mathcal{L}$(NSPACETR(f)) ⊆ NSPACE(f).

```
read x ;
control := λ ;
if x ∉ Σ+ then reject else
      y := S ;
      while y ≠ x and |y| ≤ |x| do
            guess τ ∈ U;
            guess z ∈ V+ with |y| ≤ |z| ≤ |x|;
            if z ∈ τ(y) then control := control. τ ;
                  y := z
                  else reject
            fi
      od
fi;
if control ∈ C then accept else reject fi.
```

Figure 2.

Conversely, let $L_0 \subseteq \Sigma^*$ be a language in NSPACE(f). Define $G = (V, \Sigma, \{\tau\}, S)$ with $V = \Sigma \cup \{S\}$, and τ is defined by

$$\tau(S) = L_0$$
$$\tau(w) = \{w\} \qquad \text{for each } w \text{ in } \Sigma^*.$$

Then we have $\tau \in$ NSPACETR(f), G is an NSPACETR(f)-grammar, $L(G) = L_0$, and hence NSPACE(f) $\subseteq \mathcal{L}$(NSPACETR(f)).

(3) Consider the algorithm of Figure 2. By Lemma 4.6 the last statement requires space O(f (2n)) which is O(f (n)) due to the assumption on f. So the total space needed to execute the algorithm is $O(n + f(n)) + O(f(n)) = O(f(n))$; cf. the proof of (1). □

Corollary 5.3. $\mathcal{L}$(NSPACETR(n)) = NSPACE(n). □

NSPACE(n) or, equivalently, the λ-free context-sensitive languages can be characterized by much simpler transductions than those used in 5.3. In 5.4 we combine results from [13, 32, 23, 5, 24] with Theorems 3.1-3.3.

Theorem 5.4.
$\mathcal{L}$(λNGSM;REG) = $\mathcal{L}$(λNGSM) = $\mathcal{L}$(λNGSM;1) =
$\mathcal{L}$(λDGSM;REG) = $\mathcal{L}$(λDGSM) = $\mathcal{L}$(λDGSM;2) = NSPACE(n). □

Although this result solves partially an open problem mentioned in [32], viz. $\mathcal{L}$(λDGSM;2) = NSPACE(n), the precise nature of $\mathcal{L}$(λDGSM;1) as well as an analogous characterization of DSPACE(n) are still unknown. However, it is easy to show that $\mathcal{L}$(λDGSM;1) $\subseteq$ DSPACE(n).

For a deterministic counterpart of Theorem 5.2 we can generalize the proof of Theorem 5.2 in [30] straightforwardly. This easy modification is left to the reader.

Theorem 5.5. *Let f be a function with $f(n) \geq n \log n$ for each $n \in \mathbb{N}$, and there exists a constant $c > 0$ such that $f(2n) \leq c.f(n)$ for each n. Let Γ be a family of languages closed under finite substitution and under intersection with regular languages.*

(1) *If* $\mathcal{T} \subseteq \mathrm{DSPACETR}(f)$, *then* $\mathcal{L}(\mathcal{T}) \subseteq \mathrm{DSPACE}(f)$.

(2) $\mathcal{L}(\mathrm{DSPACETR}(f)) = \mathrm{DSPACE}(f)$.

(3) *If* $\Gamma \subseteq \mathrm{DSPACE}(f)$ *and* $\mathcal{T} \subseteq \mathrm{DSPACETR}(f)$, *then* $\mathcal{L}(\mathcal{T};\Gamma) \subseteq \mathrm{DSPACE}(f)$. □

Next we turn to time-bounded transductions and time-bounded complexity classes. Instead of a single bounding function we now need a class of functions that is closed under certain operations. The following definition is a slight modification of a concept from [29].

Definition 5.6. A class $\mathcal{C}$ of functions is called *natural* if

(1) $\mathcal{C}$ contains the identity function $\lambda x.\, x$,

(2) for each f and g in $\mathcal{C}$, there is a monotone increasing function in $\mathcal{C}$ that majorizes $\lambda x.\, (f(x)+g(x))$,

(3) for each f and g in $\mathcal{C}$, there is a monotone increasing function in $\mathcal{C}$ that majorizes $\lambda x.\, (f(x)\,g(x))$, and

(4) for each f in $\mathcal{C}$, there is a monotone increasing function in $\mathcal{C}$ that majorizes $\lambda x.\, f(2x)$. □

Definition 5.7. Let for each class $\mathcal{C}$ of functions, NTIMETR($\mathcal{C}$) be the family of those transductions τ that satisfy

(1) τ is locally context-independent, and

(2) there exists a nondeterministic algorithm that can decide a query ''$y \in \tau(x)$?'' within time $f_\tau(|y|)$ for some f_τ in $\mathcal{C}$. □

For each function f, let NTIME(f) be the family of languages accepted by nondeterministic multi-tape Turing machines within time $f(n)$. For a class $\mathcal{C}$ of functions NTIME($\mathcal{C}$) is defined by $\mathrm{NTIME}(\mathcal{C}) = \cup\{\mathrm{NTIME}(f) \mid f \in \mathcal{C}\}$. Let *poly* be the class of all polynomials over the natural numbers. Obviously, poly is a natural class.

Theorem 5.8. *Let* $\mathcal{C}$ *be a natural class of functions, and let* Γ *be a family of languages closed under finite substitution and under intersection with regular languages.*

(1) *If* $\mathcal{T} \subseteq \mathrm{NTIMETR}(\mathcal{C})$, *then* $\mathcal{L}(\mathcal{T}) \subseteq \mathrm{NTIME}(\mathcal{C})$.

(2) $\mathcal{L}(\mathrm{NTIMETR}(\mathcal{C})) = \mathrm{NTIME}(\mathcal{C})$.

(3) *If* $\Gamma \subseteq \mathrm{NTIME}(\mathcal{C})$ *and* $\mathcal{T} \subseteq \mathrm{NTIMETR}(\mathcal{C})$, *then* $\mathcal{L}(\mathcal{T};\Gamma) \subseteq \mathrm{NTIME}(\mathcal{C})$.

Proof: The proof is similar to the one of Theorem 5.2. As an example we show (3). Let $(G;C) = (V, \Sigma, U, S, C)$ be a Γ-controlled $\mathcal{T}$-grammar.

Assume $U = \{\tau_1, ..., \tau_m\}$, and for each i $(1 \le i \le m)$ a query ''$z \in \tau_i(y)$?'' can be resolved within time $f_i(|z|)$ for some f_i in $\mathcal{C}$. Since $\mathcal{C}$ is natural there exists a function f in $\mathcal{C}$ that majorizes $\lambda x.\, (f_1(x) + \ldots + f_m(x))$ and hence $f(x) \ge f_i(x)$ for each x and each i $(1 \le i \le m)$.

Consider the algorithm of Figure 2. By Lemma 4.6 it suffices to execute the body of the **while**-loop at most $2n$ times where n is the length of the input. All statements in this body require time $O(n)$ only, except the test ''$z \in \tau(y)$?'' which is $O(f(n))$. Therefore this **while**-loop can be executed in time at most $O(n(n+f(n)))$. The preceding statements consume $O(n)$ time, while the last statement of the algorithm needs time $g_1(2n)$ for some g_1 in $\mathcal{C}$ (assuming that we have $C \in \mathrm{NTIME}(g_1)$). As $\mathcal{C}$ is natural, $\lambda n.g_1(2n)$ is majorized by some g in $\mathcal{C}$. Thus the total time to execute the algorithm of Figure 2 is $O(n + n(n + f(n)) + g(n))$. Since $\mathcal{C}$ is natural this is majorized by some function in $\mathcal{C}$. Hence $L(G;C) \in \mathrm{NTIME}(\mathcal{C})$. □

Corollary 5.9. $\mathcal{L}$ (NTIMETR(poly)) = NTIME(poly). □

Theorem 5.8(2) and Corollary 5.9 are variations of results established by Van Leeuwen [29] for another rather abstract grammatical model.

In addition to Theorem 5.8 we remark that from the main result in [28] it follows that if $\mathcal{T}$ contains all λ-free finite substitutions, then the membership problem for $\mathcal{L}(\mathcal{T})$ is **NP**-hard, i.e., NTIME(poly)-hard in the present notation.

6. Concluding Remarks

Controlled $\mathcal{T}$-grammars have been defined in a way such that they may be considered as generalizations of controlled iteration grammars [2, 3, 4, 8, 9] as well as of the controlled iteration of DGSM and NGSM mappings [13, 32, 23, 5, 24]. We showed that some results from these references can be extended to corresponding statements for (Γ-controlled) $\mathcal{T}$-grammars. Some of these extensions are straightforward, whereas other ones — viz. all results depending on Lemma 4.6 — only hold for grammars based on locally context-independent transductions, i.e., monotonic transductions that are context-independent in length-preserving applications (Definition 4.5). Nevertheless, it follows that some complexity classes possess stronger closure properties than those established in [4, 29, 30]. We call a family $\mathcal{K}$ of languages closed under *iterated $\mathcal{T}$-transductions* if for each language L in $\mathcal{K}$ with $L \subseteq V^*$ for some alphabet V, and each finite set U of $\mathcal{T}$-transductions over V, the language $U^*(L)$ belongs to $\mathcal{K}$.

Theorem 6.1. *Let f be a function such that there exists a constant $c > 0$ with $f(2n) \leq c.\ f(n)$ for each $n \in \mathbb{N}$.*

(1) *If $f(n) \geq n$ for each $n \in \mathbb{N}$, then* NSPACE(f) *is the smallest AFL closed under iterated locally context-independent nondeterministic f-space-bounded transductions. In particular this applies to*

- NSPACE(n), *the family of context-sensitive languages;*
- NSPACE(n^2), *the family of two-way nondeterministic nonerasing stack automaton languages;*
- DSPACE(poly).

(2) *If $f(n) \geq n \log n$ for each $n \in \mathbb{N}$, then* DSPACE(f) *is the smallest AFL closed under iterated locally context-independent deterministic f-space-bounded transductions. In particular this applies to* DSPACE($n \log n$), *the family of two-way deterministic nonerasing stack automaton languages.*

(3) *If $\mathcal{C}$ is a natural class of functions, then* NTIME($\mathcal{C}$) *is the smallest AFL closed under iterated locally context-independent nondeterministic $\mathcal{C}$-time-bounded transductions. In particular this applies to* NTIME(poly).

Proof: From 5.2(2), 5.5(2) and 5.8(2) closure under iterated $\mathcal{T}$-transductions easily follows for $\mathcal{T}$ equal to NSPACETR(f), DSPACETR(f), and NTIMETR($\mathcal{C}$), respectively. Closure under iterated $\mathcal{T}$-transductions implies closure under union, concatenation, Kleene + and λ-free homomorphism. The remaining two AFL-operations (closure under inverse homomorphism and intersection with regular languages) can be proved by standard automaton-theoretic constructions. Since each AFL closed under iterated $\mathcal{T}$-transductions includes $\mathcal{L}(\mathcal{T})$, it is easy to see that

$\mathcal{L}(\mathcal{T})$ is the smallest AFL closed under iterated $\mathcal{T}$-transductions.

For the characterization of two-way nonerasing stack automaton languages in terms of complexity classes we refer to [19]. □

It is an open problem whether a similar proposition holds for DSPACE(n), the family of deterministic context-sensitive languages.

We saw that NGSM mappings are powerful enough to generate all recursively enumerable languages: $\mathcal{L}(\text{NGSM}) = \text{RE}$ (Theorem 4.3). Now each NGSM mapping τ can be decomposed into a triple $(h_1, R, h_2) \in \lambda\text{HOM} \times \text{REG} \times \text{HOM}$ such that for each language L we have $\tau(L) = h_2(h_1^{-1}(L) \cap R)$; cf. slight modifications of the proofs of Lemma 9.3 in [19], Theorem IV.1.2 in [27], or Theorem 3.2.3 in [15]. This decomposition reflects the essential aspects of applying a production in grammatical rewriting:

(i) h_1^{-1} determines *what* ought to be rewritten,

(ii) R tells us *where* it will be replaced, and

(iii) h_2 prescribes *by which* it will be substituted.

(Notice that we have $h_1^{-1}(\lambda) = \{\lambda\}$ and hence $\tau(\lambda) = \{\lambda\}$, as h_1 is λ-free. This models the *linguistic* constraint that by applying productions from a grammar the only word derivable from λ is λ. From the previous sections it is clear that in a *mathematical* treatment of rewriting there is no need for such a constraint). This observation naturally leads to the question of characterizing well-known language families in terms of subsets of $\lambda\text{HOM} \times \text{REG} \times \text{HOM}$. We already saw an example: $\mathcal{L}(\lambda\text{HOM} \times \text{REG} \times \lambda\text{HOM}) = \text{NSPACE}(n)$, the family of context-sensitive languages (Theorem 5.4); cf. also [13, 32, 23, 5, 24]. From the proof of Lemma 9.3 in [19] it follows that $\mathcal{L}(\text{LPHOM} \times \text{T}_2 \times \text{HOM}) = \text{RE}$ — and analogously for NSPACE(n) — where LPHOM is the class of length-preserving homomorphisms, and T_2 denotes the family of 2-testable languages; cf. e.g. [27] for a definition. We may also start from a different decomposition of τ, e.g., $\tau(L) = f(h_1^{-1}(L) \cap R)$ for each L, where $(h_1, R, f) \in \lambda\text{HOM} \times \text{REG} \times \text{FINSUB}$.

Problems of this type are closely related to the subject of selective substitution grammars [25, 20], where h_1 is a length-preserving homomorphism satisfying $h_1 : (V \cup V') \to V$ with $V' = \{\alpha' \mid \alpha \in V\}$ and $h(\alpha) = h(\alpha') = \alpha$ for each α in V, $R \subseteq (V \cup V')^*$ plays the part of selector, while h_2 [or f, respectively] is a homomorphism [finite substitution] with $h_2(\alpha') \in V^*$ [$f(\alpha') \subseteq V^*$] and $h_2(\alpha) = \alpha$ [$f(\alpha) = \{\alpha\}$].

Finally, we remark that although the present paper has been inspired by [16] the central problem posed in [16] has not yet been touched.

Acknowledgment. I am indebted to Joost Engelfriet for his comment on a preliminary version of this paper.

References

1. A.V. Aho & J.D. Ullman: *The Theory of Parsing, Translation, and Compiling – Volume I: Parsing* (1972), Prentice-Hall, Englewood Cliffs, N.J.

2. P.R.J. Asveld: Controlled iteration grammars and full hyper-AFL's, *Inform. and Contr.* **34** (1977) 248-269.

3. P.R.J. Asveld: Iterated Context-Independent Rewriting – An Algebraic Approach to Families of Languages (1978), Doctoral Dissertation, Twente University of Technology, Enschede, The Netherlands.

4. P.R.J. Asveld: Space-bounded complexity classes and iterated deterministic substitution, *Inform. and Contr.* **44** (1980) 282-299.

5. P.R.J. Asveld: On controlled iterated GSM mappings and related operations, *Rev. Roum. Math. Pures et Appl.* **25** (1980) 139-145.

6. P.R.J. Asveld: Time and space complexity of inside-out macro languages, *Internat. J. Comput. Math.* **10** (1981) 3-14.

7. P.R.J. Asveld: Complexity aspects of iterated rewriting – a survey, pp. 89-105 in P.R.J. Asveld & A. Nijholt (Eds.): *Essays on Concepts, Formalisms, and Tools* (1987), Tract **42**, Centre for Mathematics and Computer Science, Amsterdam.

8. P.R.J. Asveld & J. Engelfriet: Iterated deterministic substitution, *Acta Inform.* **8** (1977) 285-302.

9. P.R.J. Asveld & J. Engelfriet: Extended linear macro grammars, iteration grammars, and register programs, *Acta Inform.* **11** (1979) 259-285.

10. P.R.J. Asveld & J. van Leeuwen: Infinite chains of hyper-AFL's, TW-memorandum No. 99 (1975), Twente University of Technology, Enschede, The Netherlands.

11. A. Cremers & S. Ginsburg: Context-free grammar forms, *J. Comput. Systems Sci.* **11** (1975) 86-117.

12. K. Culik II & T. Head: Transductions and the parallel generation of languages, *Internat. J. Comput. Math.* **13** (1983) 3-15.

13. A.C. Fleck: Formal languages and iterated functions with an application to pattern representations, Report No. 75-03 (1975) Department of Computer Science, The University of Iowa, Iowa City.

14. A. Gabrielian & S. Ginsburg: Grammar schemata, *J. Assoc. Comp. Mach.* **21** (1974) 213-226.

15. S. Ginsburg: *Algebraic and Automata-Theoretic Properties of Formal Languages* (1975), North-Holland, Amsterdam.

16. S. Ginsburg: Methods for specifying families of formal languages – past-present-future, pp. 1-22 in R.V. Book (Ed.): *Formal Language Theory: Perspectives and Open Problems* (1980), Academic Press, New York.

17. S. Ginsburg & G. Rozenberg: TOL schemes and control sets, *Inform. and Contr.* **27** (1975) 109-125.

18. M.A. Harrison: *Introduction to Formal Language Theory* (1978), Addison-Wesley, Reading, Mass.

19. J.E. Hopcroft & J.D. Ullman: *Formal Languages and Their Relation to Automata* (1969), Addison-Wesley, Reading, Mass.

20. H.C.M. Kleijn: Selective Substitution Grammars Based on Context-Free Productions (1983), Doctoral Dissertation, University of Leiden, The Netherlands.

21. H.A. Maurer, A. Salomaa & D. Wood: EOL forms, *Acta Inform.* **8** (1977) 75-96.

22. M. Nielsen: EOL systems with control devices, *Acta Inform.* **4** (1975) 373-386.

23. G. Paun: On the iteration of GSM mappings, *Rev. Roum. Math. Pures et Appl.* **23** (1978) 921-937.

24. B. Rovan: A framework for studying grammars, pp. 473-482 in J. Gruska & M. Chytil (Eds.): *Mathematical Foundation of Computer Science 1981*, Lect. Notes Comp. Sci. **118** (1981), Springer-Verlag, Berlin - Heidelberg - New York.

25. G. Rozenberg: Selective substitution grammars (towards a framework for rewriting systems) Part I: definitions and examples, *Elektron. Informationsverarbeit. Kybernetik* **13** (1977) 455-463.

26. G. Rozenberg & A. Salomaa: *The Mathematical Theory of L Systems* (1980), Academic Press, New York.

27. A. Salomaa: *Formal Languages* (1973), Academic Press, New York.

28. J. van Leeuwen: The membership question for ETOL languages is polynomially complete, *Inform. Process. Lett.* **3** (1975) 138-143.

29. J. van Leeuwen: Extremal properties of non-deterministic time-complexity classes, pp. 61-64 in E. Gelenbe & D. Potier (Eds.): *International Computing Symposium* (1975), North-Holland, Amsterdam.

30. J. van Leeuwen: A study of complexity in hyper-algebraic families, pp. 323-333 in A. Lindenmayer & G. Rozenberg (Eds.): *Automata, Languages, Development* (1976), North-Holland, Amsterdam.

31. D. Wood: A note on Lindenmayer systems, Szilard languages, spectra, and equivalence, *Internat. J. Comp. Inform. Sci.* **4** (1975) 53-62.

32. D. Wood: Iterated a-NGSM maps and Γ systems, *Inform. and Contr.* **32** (1976) 1-26.

33. D. Wood: *Grammar and L Forms: An Introduction*, Lect. Notes Comp. Sci. **91** (1980) Springer-Verlag, Berlin - Heidelberg - New York.