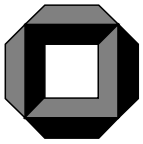

Universität Karlsruhe
Fakultät für Informatik

Henning Fernau: Observations on Grammar and Language Families

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

Interner Bericht Nr. 22/94
August 1993
erhältlich unter (available through)
ftp ftp.ira.uka.de
cd papers/techreports/1994
bin
get 1994-22.ps.Z

Contents

Observations on Grammar and Language Families

Henning Fernau

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

August 1994

Abstract: In this report, we emphasize the differences of grammar families and their properties versus language families and their properties. To this end, we investigate grammar families from an abstract standpoint, developping a new framework of reasoning. In particular when considering decidability questions, special care must be taken when trying to use decidability results (which are, in the first place, properties of grammar families) in order to establish results (e.g. hierarchy results) on language families.

We illustrate this by inspecting some theorems and their proofs in the field of regulated rewriting. In this way, we also correct the formulation of an important theorem of Hinz and Dassow.

As an exercise, we show that there is no 'effective' grammatical characterization of the family of recursive languages. Moreover, we show how to prove the strictness of the Chomsky hierarchy using decidability properties only.

Keywords: Formal languages, grammars, decidability, regulated rewriting.

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

Remark: Most of the material of this report will be published in 'fundamenta informaticae', see also [Fer94].

1 Motivation by an Introductory Example

Naturally, the theory of formal languages investigates certain properties of language families. Examples for such properties of a family $\mathcal{L}$ are:

- Is $\mathcal{L}$ closed under a certain language operation, e.g. set union?
- Hierarchy issues: What is the relation between $\mathcal{L}$ and another language family $\mathcal{L}'$? In particular, one is interested whether $\mathcal{L}$ and $\mathcal{L}'$ are incomparable, or whether $\mathcal{L}$ is included in $\mathcal{L}'$ (or vice versa) and whether such an inclusion is proper or not.

Note that we did not list decidability issues in connection with language families, although it is common practice to state something like "The word problem is decidable for regular languages.". What do we mean by these words? Maybe, it is the following:

(1) For any regular language L, there is a Turing machine (TM) T_L which, given a word w, decides whether $w \in L$ or not.

Note that the above statement contains an existential quantifier which needs not be constructive in general. There is also another constructive interpretation of the sentence "The word problem is decidable for regular languages.", namely:

(2) There is a TM which, given a description of a deterministic finite automaton (DFA) A and a word w, decides whether w is accepted by A or not.

Observe that in textbooks, this latter proposition (2) is usually proved. Of course, proposition (2) implies proposition (1).

Note that proposition (2) is basically a proposition about DFA's, but proposition (1) tells us something about the family of regular languages. Hence, an analogue to (2) needs not be true any more for other description mechanisms characterizing the regular languages.

Consider the following class of automata which obviously describes exactly the class of regular languages: A foolish finite automaton (FFA) A is specified by a quadruple (T, Q, F, δ), where T is the terminal alphabet, $Q \subset \mathbb{N}$ is a finite set of states, $F \subseteq Q$ is the set of final states, and δ is the transition function, as usual specified first as a finite function $\delta : T \times Q \to Q$, and extended by $\delta(uv, q) = \delta(u, \delta(v, q))$ to a word function. The only thing different from ordinary DFA's is that we did not itemize the initial state explicitly in our definition. Let us fix some gödelization of TM's. Denote by $T(n)$ the predicate "The TM with number n does not halt, given the argument n.". For our FFA A, let $S \subseteq Q$ denote the set of numbers $S = \{n \in Q \mid T(n)\}$. The language accepted by A is defined as

$$L(A) = \begin{cases} \emptyset, & \text{if} \quad S = \emptyset \\ \{w \in T^* \mid (\exists s \in S)(\delta(w, s) \in F)\} & \text{if} \quad S \neq \emptyset \end{cases}$$

For FFA's, the question "Is the empty word λ contained in the language described by an FFA?" is undecidable in general. This means that there is no TM that, given the

description of an FFA A, decides whether $\lambda \in L(A)$ or not. Namely, for $A = (T, Q, F, \delta)$, $\lambda \in L(A)$ iff there exists an $n \in F$ such that $T(n)$ is true. It is easily seen that the empty word problem for FFA's is Turing equivalent to the special halting problem [Odi89].

We see that in this interpretation (2), even such a special case of the word problem actually depends on the description mechanism (grammar family) under consideration and is not a property of the language family itself.

As a supplement to our list of problems treated within the theory of language families, we may list some problems encountered with grammar families $\mathcal{G}$. In this general setting, we mean by grammar family a class of finite texts which can be interpreted in a uniform way as description mechanism for formal languages. If $G \in \mathcal{G}$, according to common praxis, we write $L(G)$ in order to denote the language described by G.[1] Common questions include:

- Decidability issues, e.g. "Is there a TM which, given a grammar $G \in \mathcal{G}$ and a word w, decides whether $w \in L(G)$ or not?"
- Effective versions of the typical problems for language families listed above. Below, we will discuss this item more thoroughly.

2 Turing Equivalence of Grammar Families and Decidability of Grammar and Language Families

2.1 Turing Equivalence

One might argue that it is somewhat artificial to define devices describing regular languages in such a way that e.g. the word problem 'becomes' undecidable for this class of devices. But, what is a natural or reasonable way to describe languages? What makes DFA's more reasonable than FFA's? In some sense, from the point of view of decidability, DFA's are 'easier' than FFA's. Is it possible to say that DFA's deliver the natural description of regular languages since DFA's are **the** easiest description mechanism?

If one looks at tables in classical textbooks on formal languages containing lists of decidability questions for certain language families [sic!], one sees that any classical question is decidable for regular languages. More precisely, all these questions are decidable for DFA's.

Remarkably, in [HD89], Hinz and Dassow gave an example of a problem which is undecidable for regular grammars, as they formulated.

Lemma 2.1 (Hinz/Dassow-Lemma) There is a fixed context-free language L_0 such that there is no TM which, given a DFA A, decides whether $L(A) \subseteq L_0$ or not.

[1] Below, we will elaborate a more succinct language denotation.

We now consider another way to describe regular languages via finite automata. An easy finite automaton (EFA) A is specified by a sextuple (T, Q, F, δ, s, f), where T, Q, F, δ have the same meaning as in FFA's (and DFA's), s is the initial state (i.e. $L(A) = \{w \in T^* \mid \delta(w, s) \in F\}$) and $f \in \{0, 1\}$ is a flag such that $f = 1$ iff $L(A) \subseteq L_0$. One might look at EFA's as DFA's additionally supplied with a very special oracle.

Obviously, the problem defined by Hinz and Dassow is decidable for EFA's. Moreover, any problem decidable for DFA's is decidable for EFA's, too. Hence, in some sense EFA's are indeed 'easier' than DFA's. In other words, 'easiness' is not necessarily a basis for considering one way of describing a language family more natural than another.

Our opinion is that 'naturalness' seems to be more of a philosophical or psychological than a mathematical question. Therefore, we do not want to give further comments on this issue. But, we can try to answer the following question rigorously: "When may we call two grammar families equivalent (in a constructive or effective sense)?".

In order to do this, we adopt the ideas of Turing reducibility or equivalence from classical recursion theory [Odi89].

Definition 2.2 Let $\mathcal{G}$ and $\mathcal{G}'$ be two grammar families.

- We term two grammars $G \in \mathcal{G}$ and $G' \in \mathcal{G}'$ *equivalent* iff $L(G) = L(G')$.
- We call $\mathcal{G}$ *Turing transformable* to $\mathcal{G}'$ iff there is a TM which, given a description $G \in \mathcal{G}$ as input, produces a grammar $G' \in \mathcal{G}'$ which is equivalent to G. As a shorthand, we write $\mathcal{G} \subseteq_\mathcal{T} \mathcal{G}'$, and $G \vdash_\mathcal{T} G'$.
- We call $\mathcal{G}$ and $\mathcal{G}'$ *Turing equivalent*, $\mathcal{G} =_\mathcal{T} \mathcal{G}'$ iff $\mathcal{G} \subseteq_\mathcal{T} \mathcal{G}'$ and $\mathcal{G}' \subseteq_\mathcal{T} \mathcal{G}$.

Note that $\subseteq_\mathcal{T}$ is a reflexive and transitive relation, and that $=_\mathcal{T}$ is an equivalence relation.

One immediately sees that, if $\mathcal{G} \subseteq_\mathcal{T} \mathcal{G}'$, then, for the corresponding language families $\mathcal{L}$ and $\mathcal{L}'$, $\mathcal{L} \subseteq \mathcal{L}'$. Especially, if $\mathcal{G} =_\mathcal{T} \mathcal{G}'$, then $\mathcal{L} = \mathcal{L}'$.

Note that all 'standard notions' defining the recursively enumerable sets (the largest class of languages we consider here) are Turing equivalent in our sense, see e.g. [Odi89, Chapter 1]. We mainly refer to TM's in this paper. More precisely, we refer to the following model: A TM T computes a partial mapping $\phi : \mathbb{N} \to \Sigma^*$; the language enumerated by T is the range of ϕ. If we like to have a perpetual enumeration (without input), we might employ T by another TM using dovetailing.

2.2 Decidability of Grammar Families

Let P be some property of languages, e.g. "The considered language is empty.". Then, P can be viewed as a question concerning grammars. If $G \in \mathcal{G}$, then we say that G has property P, iff $L(G)$ possesses the property P. We use the same wording when P is additionally dependent on some parameter, e.g. in the general word problem, P depends

on the word v to be tested. In the following, we use the abbreviation P_w in order to address the general word problem $P_w(L, v) \iff v \in L$.

Definition 2.3 We say that P is *decidable for some grammar family* $\mathcal{G}$ iff there is a TM which, given a grammar $G \in \mathcal{G}$ and possibly a list of parameters of P, decides whether G has property P.

For example, the word problem P_w is decidable for the class of context-sensitive grammars, since there is a TM which, given an arbitrary context-sensitive grammar G and some word v (the list of parameters in our case), decides whether $v \in L(G)$ or not [Sal73, Theorem I.9.1]. Observe that the statement "The word problem is decidable for the class of DFA's." is equivalent to formulation (2) in the introductory section.

Lemma 2.4 If P is decidable for some grammar family $\mathcal{G}'$ and $\mathcal{G} \subseteq_{\mathcal{T}} \mathcal{G}'$, then P is decidable for $\mathcal{G}$.

We will use this lemma in order to test whether the formalisms we considered so far for describing regular languages are Turing equivalent or not.

It is easy to construct a TM which converts an EFA into an equivalent DFA. Hence, we could write $\mathcal{EFA} \subseteq_{\mathcal{T}} \mathcal{DFA}$, where $\mathcal{EFA}$ and $\mathcal{DFA}$ have their obvious meanings. On the other hand, if it were possible to transform any DFA into an equivalent EFA (which obviously exists from a nonconstructive point of view), the Hinz/Dassow problem mentioned above would become algorithmically solvable. Hence, $\mathcal{EFA} =_{\mathcal{T}} \mathcal{DFA}$ is not true, although the corresponding language classes coincide.

What about FFA's? Here, the situation is somewhat more complex.

Assume that there is a TM which, given the FFA $A = (T, Q, F, \delta)$, constructs a DFA A' being equivalent to A. We show that the predicate T defined in the first section would be recursive under this assumption. Given some number n, we consider the FFA $A = (\{a\}, \{n\}, \{n\}, (a, n) \mapsto n)$. The equivalent DFA A' would have the property $L(A') = \emptyset$ iff $T(n)$ is false. Hence, there is no effective passage from A to A'.

Now, we want to show that $\mathcal{DFA} \subseteq_{\mathcal{T}} \mathcal{FFA}$. First, fix some gödelization of TM's. Take a coding $c(n)$ of an n-state TM which loops infinitely often in any of its states, disregarding tape symbols.[2] Let c be the coding of a TM which immediately stops, neglecting the input. Given some DFA, we simply renumber its n states such that c is the initial state, and the remaining $n-1$ states are numbered $c(1), \ldots, c(n-1)$.

Hence, $\mathcal{DFA} \subseteq_{\mathcal{T}} \mathcal{FFA}$, but $\mathcal{DFA} =_{\mathcal{T}} \mathcal{FFA}$ is false.

Remarkably, when we consider the 'classical' mechanisms for the description of regular languages like deterministic finite automata, nondeterministic finite automata, right-linear grammars, and regular expressions, all these grammar families turn out to be Turing equivalent, as any basic textbook on formal languages shows.

[2] In other words, we look for infinitely many TM's which compute the nowhere defined function; their existence immediately follows by the padding lemma [Odi89].

2.3 Language and Partial Decidability

Is the way in which we transferred problems on formal languages into questions about grammars (or grammar families) the only possible one?

What do we usually mean when we call a language L 'recursive' or when we say that the general word problem is decidable for L? We mean (by definition!) that there is **some** TM T_L (generally dependent on L) which, given a word v, decides whether $v \in L$ or not.

This idea of transferring a language property into a property of grammar classes is fundamentally different from the decidability definition given above. Hence, we single out this idea by a new definition.

Definition 2.5 Let P be some property of languages, depending on a language L and, in addition, on a list of parameters l. We say that P is *language decidable for some grammar family* $\mathcal{G}$ iff, for any language L describable by some grammar from $\mathcal{G}$, there is a TM T_L which, given a list of parameters l of P, decides whether $P(L, l)$ is true or not.

For example, let us consider the following class $\mathcal{G}'$ of TM's for the acceptance of languages. A TM, given some word w, either runs forever, hence rejecting the word w, or stops in a rejecting state, hence rejecting w, or stops in an accepting state, hence accepting w. By definition, a language L is recursive if there exists a TM that decides whether a given word is member of the language or not. Within the acceptance mode just introduced for TM's, this means that a language L is recursive iff there is a TM T_L accepting L that stops on all inputs. Let $\mathcal{G} \subseteq \mathcal{G}'$ be the maximal class of TM's accepting all recursive languages. We just saw that the general word problem is language decidable for $\mathcal{G}$.

We show that the general word problem is not decidable for $\mathcal{G}$. Namely, assume the general word problem were decidable for $\mathcal{G}$. Then, let T be an arbitrary TM accepting some language which needs not be recursive. Let w denote some word coding T. With the help of T, an algorithm may construct another TM T' that accepts w if T accepts its own description w, and rejects any other input. Obviously, the language L accepted by T' is either $\{w\}$ or $\emptyset$, hence L is recursive anyway, and $T' \in \mathcal{G}$. By assumption, the question $w \in L$ is decidable, more precisely, there exists a TM that, given T' and w, decides whether $w \in L(T') = L$ or not. This in turn would solve the special word problem for TM's, contradicting our assumption.

We see that language decidability intrinsically contains an element of undecidability, namely the generally nonconstructive existential quantifier in the following sloppy but easily understandable formalization of language decidability:

$$P \text{ language decidable for } \mathcal{G} \iff (\forall L \in \mathcal{L}(\mathcal{G}))(\exists \text{ TM } T_L)(\forall l)(P(L, l) \text{ decidable by } T_L)$$

This contrasts decidability for language families which might be formalized analogously by

$$P \text{ decidable for } \mathcal{G} \iff (\exists \text{ TM } T)(\forall G \in \mathcal{G})(\forall l)(P(L(G), l) \text{ decidable by } T)$$

Obviously, language decidability does not depend on the grammar family but on the corresponding language family. Instead of saying that P is language decidable for some grammar family $\mathcal{G}$ we also say that P is decidable for the corresponding language family $\mathcal{L}(\mathcal{G})$.

Observe that the statement "The word problem is decidable for the class of regular languages." is equivalent to formulation (1) in the introductory section.

Note that if the attached list of parameters is empty or not used in the decidability question, then language decidability is always trivial and hence solvable. Therefore, a formulation like "The emptiness problem is solvable for the language family $\mathcal{L}$." makes no sense.

Comparing these two definitions (which, as we have seen, both occur in the standard theory of formal languages), we are tempted to free language decidability from its bound to formal languages, arriving at the following definition.

Definition 2.6 Let P be some property of grammars, depending on a grammar G and, in addition, on a list of parameters l. We say that P is *partially decidable for some grammar family* $\mathcal{G}$ iff, for any grammar $G \in \mathcal{G}$ there is a TM T_G that, given a list of parameters l of P, decides whether $P(G, l)$ is true or not.

Of course, for any grammar family and any property P of languages (with parameters!) partial decidability is equivalent to language decidability which in turn follows from decidability. In some sense, we may look at properties without parameters as extreme cases of properties with parameters (The TM may simply ignore the parameters.). Then, any property without parameters is trivially language decidable, which makes the unconstructibility of these definitions quite clear. Finally, we might code any problem e.g. on TM's into the list of parameters, which shows that there are (admittedly, rather artificial) problems which are partially undecidable for any grammar family.

Since language decidability applies only to language properties, decidability, language decidability and partial decidability are really different and non-trivial notions in general.

Our observations are summarized in the following.

- Let P be some property of grammars (e.g. inherited by a property of languages). Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be two grammar families such that P is decidable for $\mathcal{G}_1$ but undecidable for $\mathcal{G}_2$. Then, $\mathcal{G}_1$ and $\mathcal{G}_2$ are not Turing equivalent, $\mathcal{G}_1 \neq_\tau \mathcal{G}_2$ for short. Generally, this fact does not tell us anything about the corresponding language families.

- Let P be some property (with parameters) of languages. Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be two language families such that P is decidable for $\mathcal{L}_1$ but undecidable for $\mathcal{L}_2$. Then, $\mathcal{L}_1 \neq \mathcal{L}_2$.

Finally, we state the following observation:

Lemma 2.7 Let P be some property (with parameters) of languages. Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be two language families with $\mathcal{L}_1 \subseteq \mathcal{L}_2$. If P is decidable for $\mathcal{L}_2$, then P is also decidable for $\mathcal{L}_1$.

2.4 Effective Operations

Let F be some n-ary operation on languages. For example, set union is a binary operation on languages. We term a grammar family $\mathcal{G}$ *effectively closed under* F iff there is a TM which, given n grammars $G_1, \ldots, G_n \in \mathcal{G}$, produces another grammar $G \in \mathcal{G}$ which describes the result of the operation on the languages corresponding to $G_1, \ldots, G_n$.

We note that there is a slight difference between the transfer of closure properties (as properties of language families) into effective closure properties (as properties of grammar families) and the interpretation of predicates of languages as decidability questions concerning grammar families. We illustrate this difference with the help of the following lemma which contains a stronger premise than the analogous Lemma 2.4 regarding predicates of languages.

Lemma 2.8 Let $\mathcal{G}, \mathcal{G}'$ be two Turing equivalent grammar families. If $\mathcal{G}'$ is effectively closed under a certain language operation, then $\mathcal{G}$ is effectively closed under that operation, too.

Proof. Let the operation in question be n-ary. Take $G_1, \ldots, G_n \in \mathcal{G}$. By assumption, it is effectively possible to transform these grammars into equivalent grammars $G'_1, \ldots, G'_n \in \mathcal{G}'$. Now, a TM may compute a grammar $G' \in \mathcal{G}'$ which describes the result of the operation on the languages corresponding to $G_1, \ldots, G_n$. By Turing equivalence, $G' \in \mathcal{G}'$ is effectively transformable into an equivalent grammar $G \in \mathcal{G}$. Hence, $\mathcal{G}$ is effectively closed under the given n-ary operation. □

Observe that there are operations which are inherently ineffective. Consider e.g. the following operation under which any language family containing all finite languages is closed. Let a be some symbol. The predicate T describes the halting problem as defined above. Define

$$F(L) = \begin{cases} L \setminus \{a^3\} \cup \{a, a^2\} & \text{if } |L| = \infty \\ \{a, a^3\} & \text{if } |L| = n \wedge T(n) \\ \{a^2, a^3\} & \text{if } |L| = n \wedge \neg T(n) \end{cases}$$

Let $\mathcal{FIN}$ denote the family of grammars consisting of finite lists of words. Let $\mathcal{G}$ be some grammar family satisfying the following (rather weak) requirements:

- $\mathcal{FIN} \subseteq_{\mathcal{T}} \mathcal{G}$
- There is a TM which, given some $G \in \mathcal{G}$, enumerates the language described by G.

Then, F is not an effective operation for $\mathcal{G}$, since otherwise we can construct a TM TM solving the special halting problem: Given a number n, first, TM generates some language

containing n words, then, TM transforms this language into its grammatical representation, on which TM can effectively apply F yielding a grammar G; finally, TM listens to a simulation of G: $T(n)$ is true iff the word a appears in the enumeration process.

It seems to be reasonable to rule out operations like the above-defined F when considering effectivity of operations. But we do not define a proper class of 'admissible' operations in this paper, since we run into problems e.g. with the following operation:

$$F(L) = \begin{cases} (L \setminus \{\lambda\})^+ & \text{if } L \text{ is recursive} \\ \{\lambda\} & \text{otherwise} \end{cases}$$

The families of right-linear, context-free or context-sensitive grammars are effectively closed under F, but the family of all phrase-structure grammars is not effectively closed under F (although the corresponding family of languages is closed under F), since the encoded ineffectivity only applies in case of very powerful grammar families.

In the following, we only consider 'classical' language-theoretical operations like (inverse, λ-free) homomorphism/substitutions, catenation and its closure and Boolean operations. We term a grammar family $\mathcal{G}$ *ineffective* iff the corresponding language family is closed under some 'classical' operation, but $\mathcal{G}$ is not effectively closed under that operation.

Observe there is again a subtle problem encountered with a formulation like "The grammar family $\mathcal{G}$ is effectively closed under substitution of regular languages.", since there are two ways to interpret this phrase. Either it is meant that for any substitution operation induced by relating letters with regular languages, there is a TM which, given a grammar $G \in \mathcal{G}$, produces another grammar $G' \in \mathcal{G}$ whose language is just the substitution image of $L(G)$; or, we mean a uniform interpretation: There is a TM which, given a substitution with regular languages and a grammar $G \in \mathcal{G}$, produces another grammar $G' \in \mathcal{G}$ whose language is just the substitution image of $L(G)$. In the latter case, "given a substitution with regular languages" is of course dependent on the chosen representation of regular languages. We will apply the non-uniform interpretation throughout the present paper, but we wanted to point out the possible difficulties in the uniform case.

3 Constructible, Enumerable and Recursive Grammar Families

3.1 Constructible Grammar Families

The problems encountered with FFA's have also another reason: FFA's are unconstructible in the following sense.

Definition 3.1 Let $\mathcal{G}$ be some grammar family. We call $\mathcal{G}$ *constructible* iff there is a TM (called $\mathcal{G}$-universal machine) which, given a description $G \in \mathcal{G}$, enumerates $L(G)$. $\mathcal{G}$ is called unconstructible iff there is no $\mathcal{G}$-universal machine.

Assume the FFA's were constructible via a universal machine. We show that, in this case, the predicate T defined above is decidable. We start two TM's T_1 and T_2 in parallel. T_1 is just the n^{th} TM running on the input n. T_2 is the TM enumerating the language of the FFA $A = (\{a\}, \{n\}, \{n\}, (a, n) \mapsto n)$. T_2 eventually halts and outputs some word iff T_1 does not halt. Hence, either T_1 or T_2 halts, rendering our predicate T decidable.

Lemma 3.2 A grammar family $\mathcal{G}$ is constructible iff, for any $G \in \mathcal{G}$, there is a TM which, given some word w, halts iff $w \in L(G)$.

Proof. If G is constructible, then there exists a TM T_1 enumerating $L(G)$. Construct a TM T that, given as input w, 'listens to' T_1 until w appears in the enumeration and stops in that case; if w does not appear in the enumeration, T will never stop.

Assume that there is a TM T_2 that, given some word w, halts iff $w \in L(G)$. Construct a TM $\tilde{T}$ that tests, in a dovetailing manner, any word w using T_2 and outputs w iff T_2 eventually stops. □

We could also view constructibility in terms of transformability:

Proposition 3.3 A grammar family $\mathcal{G}$ is constructible iff $\mathcal{G}$ is Turing transformable to the family of TM's (viewed as language generators). □

This immediately implies:

Corollary 3.4 The language family $\mathcal{L}$ characterized by a constructible grammar family contains only enumerable languages.

Hence, we rule out devices which are 'too powerful' by definition.

It is instructive to compare the notion of constructibility of grammar families with notions introduced in the preceding section. Almost by definition, we obtain:

Remark 3.5 Let $\mathcal{G}$ be a grammar family for which the general word problem is decidable. Then, $\mathcal{G}$ is constructible. Furthermore, the language family corresponding to $\mathcal{G}$ contains only recursive languages.

We will come back to this topic at the end of this section.

3.2 Recursive and Enumerable Grammar Families

What do we mean by the term 'grammar family'? Essentially, a grammar family may be identified with a certain meta-language defining the syntactic structure of the grammars viewed as strings together with a formalism prescribing how to squeeze formal languages out of grammars. More formally, any grammar family $\mathcal{G}$ is given by a pair $(L_{\mathcal{G}}, \mathcal{I}_{\mathcal{G}})$, where $L_{\mathcal{G}}$ is the meta-language describing the syntactic structure of feasible grammars, and the interpretation function $\mathcal{I}_{\mathcal{G}}$ maps words (grammars) from $L_{\mathcal{G}}$ onto formal languages.

The idea of identifying grammar families with formal languages may look strange, even a bit circuitous, but is perfectly accepted e.g. when defining the family of regular expressions (mostly over a fixed finite alphabet).

There is one minor problem with meta-languages: Generally, meta-languages are languages over countable alphabets. Even the restriction to some fixed finite alphabet of terminal symbols does not overcome this infinity, since e.g. in context-free grammars, it is not possible to restrict the number of nonterminal symbols without reducing the generative power. Using a suitable recursive coding of the countable alphabet, we can assume the meta-language to be a language over a finite alphabet. A word in a meta-language represents an index of another formal language. Constructiblity means that, given such an index, it is effectively possible to pass to the represented formal language, in other words, the grammar family is effectively decodable.

How do we know that a certain word is actually interpretable as such an index? This leads us to the next 'natural' restriction on grammar families.

Definition 3.6 We call a grammar family $\mathcal{G} = (L_{\mathcal{G}}, \mathcal{I}_{\mathcal{G}})$ *recursive* iff $L_{\mathcal{G}}$ is recursive.

Obviously, 'indexness' may be tested algorithmically exactly for recursive grammar families.

For example, it is easily seen that $\mathcal{FFA}$ and $\mathcal{DFA}$ are recursive, but $\mathcal{EFA}$ is not recursive by the Hinz/Dassow-Lemma.

Lemma 3.7 A grammar family $\mathcal{G} = (L_{\mathcal{G}}, \mathcal{I}_{\mathcal{G}})$ is constructible and recursive iff the following two functions are partial recursive:[3]

$$f(w,G) = \begin{cases} \uparrow & \text{if} \quad G \in L_{\mathcal{G}} \wedge w \notin \mathcal{I}_{\mathcal{G}}(G) \\ 1 & \text{if} \quad G \in L_{\mathcal{G}} \wedge w \in \mathcal{I}_{\mathcal{G}}(G) \\ 2 & \text{if} \quad G \notin L_{\mathcal{G}} \end{cases}$$

$$g(G) = \begin{cases} 0 & \text{if} \quad G \notin L_{\mathcal{G}} \\ 1 & \text{if} \quad G \in L_{\mathcal{G}} \end{cases}$$

Interestingly, it is sometimes possible to change an unconstructible grammar family into a nonrecursive one and vice versa. For example, consider the grammar family $\mathcal{G}$ of NSEFA's (not so easy fa's) derived from EFA's. Syntactically, there is no difference between NSEFA's and EFA's, that is $L_{\mathcal{G}} = L_{\mathcal{EFA}}$. An NSEFA $A = (T, Q, F, \delta, s, f)$ is based on the DFA $E_A = (T, Q, F, \delta, s)$. We define as interpretation function

$$\mathcal{I}_{\mathcal{G}}(A) = \begin{cases} \emptyset & \text{if} \quad (f = 0 \wedge \mathcal{I}_{\mathcal{DFA}}(E) \subseteq L_0) \vee (f = 1 \wedge \mathcal{I}_{\mathcal{DFA}}(E) \not\subseteq L_0) \\ \mathcal{I}_{\mathcal{DFA}}(E) & \text{otherwise} \end{cases}$$

Again, L_0 denotes the context-free language constructed by Hinz/Dassow. $\mathcal{G}$ is recursive but not constructible.

[3]Following Odifreddi, $\uparrow$ means 'undefined' or, concerning TM's, 'not halting'.

Similarly, by adding a piece of information, it is sometimes possible to turn an unconstructible mechanism into a constructible one, e.g. FFA's into DFA's. It is even possible that, by such an addendum, the considered mechanism 'becomes' nonrecursive, e.g. turning FFA's into EFA's.

Recall that we called a property P regarding grammars decidable for a particular grammar family $\mathcal{G}$ iff there is a TM which, given a description $G \in \mathcal{G}$, decides whether G has property P or not. When identifying $\mathcal{G}$ with a formal language, this is not quite the kind of decidability questions tackled normally, since, as we have seen, a TM might not be able to check whether its input word w corresponds to some grammar in $\mathcal{G}$ or not. Therefore, we consider the following function induced by P and $\mathcal{G}$:

$$\mathcal{P}_{\mathcal{G}}(w) = \begin{cases} 0 & \text{if} \quad w \in \mathcal{G} \wedge w \text{ has } \neg P \\ 1 & \text{if} \quad w \in \mathcal{G} \wedge w \text{ has } P \\ 2 & \text{if} \quad w \notin \mathcal{G} \end{cases}$$

Presupposing that P is decidable for $\mathcal{G}$ and that $\mathcal{G}$ is recursive, we see that $\mathcal{P}_{\mathcal{G}}$ is recursive. Moreover, if $\mathcal{G}'$ is a recursive grammar family and $\mathcal{G}' \subseteq_{\mathcal{T}} \mathcal{G}$, then $\mathcal{P}_{\mathcal{G}'} \leq_{\mathcal{T}} \mathcal{P}_{\mathcal{G}}$ in the classical sense of Turing reducibility, since

$$\mathcal{P}_{\mathcal{G}'}(w') = \begin{cases} \mathcal{P}_{\mathcal{G}}(w) & \text{if} \quad w' \in \mathcal{G}' \wedge w' \vdash_{\mathcal{T}} w \\ 2 & \text{if} \quad w' \notin \mathcal{G}' \end{cases}$$

Sometimes it is sufficient to presuppose that it is algorithmically possible to list all grammars of a specific language family. We single out this property in the following definition.

Definition 3.8 We call a grammar family $\mathcal{G} = (L_{\mathcal{G}}, \mathcal{I}_{\mathcal{G}})$ *enumerable* iff $L_{\mathcal{G}}$ is enumerable.

Note that a constructible and enumerable grammar family $\mathcal{G}$ supplies us with an indexing scheme of TM's enumerating just the languages generable by some $G \in \mathcal{G}$. More precisely, we arrive at enumerable indices in this case [Odi89, p.226]. If $\mathcal{G}$ allows effective complementation (with respect to a suitable alphabet), then we even obtain characteristic indices. We will exploit this fact below.

Let $\mathcal{G}$ be an enumerable grammar family. This means that there is a TM that enumerates any grammar in $\mathcal{G}$ at least once. Hence, there is an effective indexing scheme for $\mathcal{G}$. In this case, we simply write $\mathcal{G} = \{G_1, G_2, \ldots\}$. Of course, we may also list all words over some fixed alphabet Σ, say $w_1, w_2, \ldots$

By a standard diagonal argument, we obtain:

Lemma 3.9 If $\mathcal{G}$ is an enumerable and constructible grammar family with $L_{\mathcal{G}} \subseteq \Sigma^*$ for some finite alphabet Σ, then the diagonal language $D_{\mathcal{G}} = \{w_i \mid w_i \in \mathcal{I}_{\mathcal{G}}(G_i), G_i \in L_{\mathcal{G}}\}$ is enumerable, but its complement is not describable by any $G \in \mathcal{G}$. □

Theorem 3.10 If $\mathcal{G}$ is an enumerable and constructible grammar family characterizing the enumerable languages with $L_{\mathcal{G}} \subseteq \Sigma^*$ for some finite alphabet Σ, then $\Sigma^* \setminus D_{\mathcal{G}}$ is not enumerable. □

Similarly, we see:

Lemma 3.11 Let $\mathcal{G}$ be an enumerable grammar family with $L_{\mathcal{G}} \subseteq \Sigma^*$ for some finite alphabet Σ. If the general word problem is decidable for $\mathcal{G}$, then the diagonal language $D_{\mathcal{G}} = \{w_i \mid w_i \in \mathcal{I}_{\mathcal{G}}(G_i), G_i \in L_{\mathcal{G}}\}$ is recursive.

Proof. By assumption, there is an enumeration $\mathcal{G} = \{G_1, G_2, \ldots\}$. Furthermore, it is possible to list all words over Σ effectively, say $\Sigma^* = \{w_1, w_2, \ldots\}$. Since the general word problem is decidable for $\mathcal{G}$, the following function is total recursive:

$$f(i,j) = \begin{cases} 0 & \text{if } w_j \notin \mathcal{I}_{\mathcal{G}}(G_i) \\ 1 & \text{if } w_j \in \mathcal{I}_{\mathcal{G}}(G_i) \end{cases}$$

The diagonal language $D_{\mathcal{G}}$ is recursive: Given a word w, a TM may first compute an index i such that $w = w_i$, then generate G_i and check whether $w_i \in \mathcal{I}_{\mathcal{G}}(G_i)$. □

Theorem 3.12 If $\mathcal{G}$ is an enumerable grammar family characterizing the enumerable languages with $L_{\mathcal{G}} \subseteq \Sigma^*$ for some finite alphabet Σ, then the general word problem is undecidable for $\mathcal{G}$. □

Note that our last two theorems are usually the basic steps in order to show Rice's theorem e.g. for TM's [HU79]. What is lacking in order to show an analogue to Rice's theorem for grammar families characterizing the enumerable languages? In general, we do not have something like composition or the S_m^n theorem. On the other hand, if we knew something like composition being effectively possible for a specific enumerable grammar family and if we knew some non-trivial property of enumerable languages solvable for that particular grammar class, then this grammar family cannot describe any enumerable language.

We try to capture the above idea in the following theorem:

Theorem 3.13 Let $\mathcal{G}$ be a grammar family consisting of grammars that generate words starting from some start symbol in such a way that the possible direct derivations are independent of the number of the derivation step.[4] Then, the following properties imply that there is an enumerable language not describable (generable) by any grammar from $\mathcal{G}$.

- $\mathcal{G}$ is enumerable.

[4] In case of such a dependence (as encountered for example in time-varying grammars and function-limited systems [Fer91]), we are in need of a more envolved construction.

- $\mathcal{G}$ is effectively closed under A transductions. [5]
- There is a language property P which is non-trivial for the familiy $\mathcal{L}$ of languages describable via $\mathcal{G}$ but decidable for $\mathcal{G}$.

More precisely, the above conditions imply that, for $\mathcal{G}$, the general word problem is decidable.

Proof. By assumption, there is a non-empty language $L \in \mathcal{L}$ such that $P(L)$ is true and $P(\emptyset)$ is false.[6] Let L be generable via the grammar $G_L \in \mathcal{G}$ having S as its start symbol. Let G be an arbitrary grammar from $\mathcal{G}$ and w be some word. By assumption, a TM may construct another grammmar $G_w \in \mathcal{G}$ generating $\{S\}$ iff $w \in \mathcal{I}_{\mathcal{G}}(G)$ and generating $\emptyset$ iff $w \notin \mathcal{I}_{\mathcal{G}}(G)$ using a suitable A transduction. Now, another grammar G'_w may use G_w and start to enumerate L simulating G_L iff $w \in \mathcal{I}_{\mathcal{G}}(G)$. By assumption, this coupling is possible disregarding the number of the derivation step encountered when starting with the simulation of G_L. Hence, $\mathcal{I}_{\mathcal{G}}(G'_w) = L$ iff $w \in \mathcal{I}_{\mathcal{G}}(G)$ and $\mathcal{I}_{\mathcal{G}}(G'_w) = \emptyset$ iff $w \notin \mathcal{I}_{\mathcal{G}}(G)$. Therefore, $P(\mathcal{I}_{\mathcal{G}}(G'_w))$ iff $w \in \mathcal{I}_{\mathcal{G}}(G)$. By assumption, the first property is effectively testable. □

A simple corollary of the above theorem is the fact that programmed grammars with appearance checking are strictly more powerful than programmed grammars without appearance checking, since the emptiness problem is solvable for the latter grammar class.[7] Note that this fact was proved quite recently [HD89] using other techniques.

3.3 An Exercise on Recursive Languages

As an example, we consider the family of recursive languages. It is an open question whether there is a 'natural' grammatical characterization of this language class. As we will show in the following, any grammar family characterizing the recursive languages must have some strange properties.

For the sake of convenience, we restrict ourselves to the consideration of languages over a fixed alphabet Σ. In the following, we discuss any grammar family $\mathcal{G}$ under the next aspects:

- Is $\mathcal{G}$ constructible (A) or unconstructible (B)?
- Is $\mathcal{G}$ recursive (a) or enumerable but nonrecursive (b) or even nonenumerable (c)?
- Is there a TM which, given a description $G \in \mathcal{G}$, computes a description $G' \in \mathcal{G}$ such that $L(G') = \Sigma^* \setminus L(G)$? In other words: Is complementation effective? (1) Or is complementation not effective? (2)

[5] Consult [HU79] on A transducers (generalized sequential machines that may react on λ-inputs).
[6] If $P(\emptyset)$ is true, just switch to $\neg P$ instead of P in the above argument.
[7] As regards formal definitions, see below. Check [DP89] for the required properties.

This leaves us with twelve different cases in total. We denote these cases, according to the parenthesized symbols above, for example by (Ac2). We present solutions in ten cases: Either we present a grammar family characterizing the recursive languages over Σ having the required properties or we prove (in two cases) that such a grammar family cannot exist. We were not able to solve the remaining two cases, namely (Aa2) and (Ab2). In particular, the following question is open: Is there a constructible enumerable grammar family with noneffective complementation that describes the recursive languages or not?

First, we present our negative result solving (Aa1) and (Ab1) in the form of a theorem.

Theorem 3.14 Let $\mathcal{G}$ be a constructible and enumerable grammar family characterizing the recursive languages over Σ. Then, complementation is not effective.

In other words: Any constructible and enumerable grammar family characterizing the recursive languages is ineffective.

Proof. Let $\mathcal{G}$ be a constructible and enumerable grammar family with effective complementation characterizing the recursive languages over Σ.

This implies that there is a TM which, given a grammar $G \in \mathcal{G}$, produces a grammar $G' \in \mathcal{G}$ such that $L(G) = \Sigma^* \setminus L(G')$.

Therefore, the following function is computable (cf. Lemma 3.7):

$$f(w,G) = \begin{cases} 0 & \text{if} \quad G \in L_{\mathcal{G}} \wedge w \notin \mathcal{I}_{\mathcal{G}}(G) \\ 1 & \text{if} \quad G \in L_{\mathcal{G}} \wedge w \in \mathcal{I}_{\mathcal{G}}(G) \\ \uparrow & \text{if} \quad G \notin L_{\mathcal{G}} \end{cases}$$

Assume $f = \phi_e$ for a suitable enumeration of the partial recursive functions $\{\phi_e\}$. By S_m^n theorem, $g(w) = \phi_{S_1^1(e,G)}(w)$ is the total recursive characteristic function of $\mathcal{I}_{\mathcal{G}}(G)$. This allows us to specify the enumerable subset $\{S_1^1(e,G) \mid G \in L_{\mathcal{G}}\}$ of the set of characteristic indices of the recursive languages such that any recursive language is represented by at least one index within our subset. This contradicts the effective Cantor theorem [Odi89] (Proposition II.2.1 in connection with a remark on page 234). □

(Ac1) A constructible, nonenumerable grammar family with effective complementation characterizing the recursive languages: Consider as grammars pairs (T_1, T_2) of TM's such that $L(T_1) = \Sigma^* \setminus L(T_2)$. Define as interpretation $\mathcal{I}(T_1, T_2) = L(T_1)$. It is easily seen that this family is not recursive. If it were enumerable, we would have found an enumerable set of characteristic indices listing all recursive languages, again contradicting the effective Cantor theorem.

(Ac2) A constructible, nonenumerable grammar family with noneffective complementation characterizing the recursive languages: Consider as descriptions just the TM's which enumerate some recursive language. By [Odi89, II.5.19], the set of the enumerable indices of all recursive sets is nonenumerable.

(Ba1) An unconstructible, recursive grammar family with effective complementation characterizing the recursive languages: Consider as descriptions pairs (T_1, T_2) of TM's.

Define the following interpretation function:

$$\mathcal{I}(T_1, T_2) = \begin{cases} \emptyset & \text{if} \quad L(T_1) \neq \Sigma^* \setminus L(T_2) \\ L(T_1) & \text{if} \quad L(T_1) = \Sigma^* \setminus L(T_2) \end{cases}$$

(Ba2) An unconstructible, recursive grammar family with noneffective complementation characterizing the recursive languages may be obtained in the following way: Take as descriptions TM's and define as interpretation

$$\mathcal{I}(T) = \begin{cases} \emptyset & \text{if} \quad L(T) \text{ is not recursive} \\ L(T) & \text{if} \quad L(T) \text{ is recursive} \end{cases}$$

Adding an additional flag indicating whether the n^{th} TM with input n halts or not (n ='length of description'), we obtain the following two cases from (Ba1) and (Ba2), respectively.

(Bb1) An unconstructible, nonrecursive but enumerable grammar family with effective complementation characterizing the recursive languages.

(Bb2) An unconstructible, nonrecursive but enumerable grammar family with noneffective complementation characterizing the recursive languages.

(Bc1) An unconstructible, nonenumerable grammar family with effective complementation characterizing the recursive languages: Consider as descriptions pairs (T_1, T_2) of TM's that compute only total enumerations $\mathbb{N} \to \Sigma^*$. Define the interpretation function as follows:

$$\mathcal{I}(T_1, T_2) = \begin{cases} \emptyset & \text{if} \quad L(T_1) \neq \Sigma^* \setminus L(T_2) \\ L(T_1) & \text{if} \quad L(T_1) = \Sigma^* \setminus L(T_2) \end{cases}$$

By [Odi89, page 146], the set of TM's computing only total enumerations is not enumerable.

(Bc2) An unconstructible, nonenumerable grammar family with noneffective complementation characterizing the recursive languages: Consider as grammars TM's computing total enumerations of languages. We change the definition of the language described by such a TM T as in case (Ba2).

As argued above, the language decidability of the general word problem is an intrinsic (if you like: **the** intrinsic) property of the family of recursive languages. Is it possible that there is a grammar family $\mathcal{G}$ characterizing the recursive languages such that the general word problem is decidable for $\mathcal{G}$? The next theorem shows that this is not possible, under rather weak conditions.

Theorem 3.15 Let $\mathcal{G}$ be an enumerable grammar family characterizing the recursive languages over Σ. Then, the general word problem is undecidable for $\mathcal{G}$.

Proof. Assuming the contrary, we may apply Lemma 3.11. By diagonalization, the complement of the diagonal language is not describable by any $G \in \mathcal{G}$. □

We might formulate the last theorem in the following manner:

- Let $\mathcal{G}$ be an enumerable grammar family describing only recursive languages over Σ. If the general word problem is decidable for $\mathcal{G}$, then there is a recursive language not describable by any $G \in \mathcal{G}$.

  Hence, the family of languages characterized by $\mathcal{G}$ is strictly contained in the family of recursive languages.

In this way, we see that Theorem 3.15 also adds to Theorem 3.13, since we learned that any grammar family satisfying the assumptions of Theorem 3.13 does not characterize the recursive languages.

Interestingly, by this theorem we may use a decidability property of grammar families in order to show that the corresponding language family is strictly contained in the family of recursive languages. This applies especially to the family of context-sensitive languages and to certain families defined via regulated rewriting. Later on, we will discuss these cases in detail.

The following proposition links our last theorem with the preceding considerations on complementation within grammar families characterizing the recursive languages.

Theorem 3.16 Let $\mathcal{G}$ be a constructible grammar family with effective complementation describing recursive languages only. Then, the general word problem is decidable for $\mathcal{G}$.

Proof. Given some grammar G, a TM T_1 may compute another grammar G' describing the complement of $\mathcal{I}_{\mathcal{G}}(G)$. By assumption, two other TM's T_2 and T_3 may enumerate G and G', respectively. Hence, there exists another TM T_4 that dovetails the enumerations of T_2 and T_3 in order to decide the question whether a given word w is contained in $\mathcal{I}_{\mathcal{G}}(G)$ (then it will be enumerated by T_2) or not (then it will be enumerated by T_3). □

As an application of this theorem, we can derive Theorem I.9.1 in [Sal73] from the recently proved fact [Sze88b, Sze88a, Imm88] that the family of context-sensitive grammars allows effective complementation. It is not possible to reason 'the other way round' coming from the decidability of the word problem leading to effective complementation, as the family of context-free grammars shows.

Note that the proof resembles Post's characterization of recursiveness. Furthermore, the combination of our last two theorems yields another proof of Theorem 3.14.

4 Examples from Literature

Our considerations might look quite academic so far. In this section, we look for concrete examples found in literature.

4.1 Programmed Grammars

A *programmed grammar* ([Ros69, Sto71, DP89]) is a construct $G = (V_N, V_T, P, S)$, where V_N, V_T, and S are, as in Chomsky grammars, the alphabet of nonterminal symbols, the

alphabet of terminal symbols, and the start symbol, 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 called core 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. For (x, r_1) and (y, r_2) in $V_G^* \times \mathrm{Lab}(P)$, we write $(x, r_1) \Rightarrow (y, r_2)$ iff either

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

or

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

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 $L(G) = \{w \in V_T^* \mid (S, r_1) \stackrel{*}{\Rightarrow} (w, r_2) \text{ for some } r_1, r_2 \in \mathrm{Lab}(P)\}$. The family of languages generated by programmed grammars of the form $G = (V_N, V_T, P, S)$ containing only context-free core rules is denoted by $\mathcal{L}(\mathrm{P,CF,ac})$. When no appearance checking features are involved, i.e. $\phi(r) = \emptyset$ for each rule in P, we are led to the families $\mathcal{L}(\mathrm{P,CF})$. The special variant of a programmed grammar, where the success field and the failure field coincide for each rule r in the set P of productions, is said to be a programmed grammar with *unconditional transfer* . According to the notation which has been introduced so far, we denote the class of languages generated by programmed grammars with context-free productions and unconditional transfer by $\mathcal{L}(\mathrm{P,CF,ut})$.

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

If erasing rules are forbidden, we replace the component CF by CF–λ in that notation, again, i.e. we get $\mathcal{L}(\mathrm{P,CF\text{–}\lambda,ac})$, $\mathcal{L}(\mathrm{P,CF\text{–}\lambda,ut})$, ... When talking about properties of the corresponding grammar families, we write e.g. $\mathcal{G}(\mathrm{P\text{-}left,CF,ut})$.

For the convenience of the reader, we quote some results from [Ros69].

Theorem 4.1 The prefix property is decidable for $\mathcal{G}(\mathrm{P\text{-}left,CF\text{–}\lambda,ut})$; i.e. there is a TM which, given some $G \in \mathcal{G}(\mathrm{P\text{-}left,CF\text{–}\lambda,ut})$ and some string x, decides whether there exists a string y such that $xy \in L(G)$.

Theorem 4.2 Every recursively enumerable language can be generated with tails by some $G \in \mathcal{G}(\mathrm{P\text{-}left,CF\text{–}\lambda,ac})$. This means that there is a TM that, given some phrase structure grammar G' with $L(G') \subseteq V_T^*$, can construct some $G \in \mathcal{G}(\mathrm{P\text{-}left,CF\text{–}\lambda,ac})$ such that $x \in L(G')$ iff $xdc^n \in L(G)$ for some $n > 0$ (where $c, d \notin V_T$).

Rosenkrantz correctly stated [Ros69, Corollary 4, page 128]:

Corollary 4.3 $\mathcal{L}(\text{P-left,CF}\text{–}\lambda, \text{ut}) \subsetneq \mathcal{L}(\text{P-left,CF}\text{–}\lambda, \text{ac})$.

His very instructive argument goes as follows: Let M be some nonrecursive language generated with tails by some $G \in \mathcal{G}(\text{P-left,CF}\text{–}\lambda, \text{ac})$, i.e. $M = \{x \mid xdc^n \in L(G) \text{ for some } n > 0\}$. If the language families in question coincided, there would be a $G' \in \mathcal{G}(\text{P-left,CF}\text{–}\lambda, \text{ut})$ such that $L(G') = L(G)$. The prefix problem is decidable for G'. Hence, there is a TM T which, given some word xd, outputs 1 iff xd is prefix of some word of $L(G')$ iff $x \in M$. Therefore, there is a TM T_M which, first, given some word x, appends d and then uses T in order to solve the membership problem for M. Hence, the language families cannot coincide.

Note that we need not obtain the grammar G' effectively out of G.

Two years later, Stotskii continued the investigation of programmed grammars [Sto71]. He shows e.g. the following result:

Theorem 4.4 For every enumerable set M of natural numbers, it is possible to construct a $G_M \in \mathcal{G}(\text{P-left,CF}\text{–}\lambda, \text{ut})$ with terminal alphabet $V_T = \{a, b, d, e\}$ such that $L(G_M)$ possesses the following properties:

1. For every $m \in M$, we find a word of the form $a^m db^n e$ in $L(G_M)$ (with $n > 0$).
2. Any word of the form $a^m db^n e \in L(G_M)$ determines a number $m \in M$.
3. Any word in $L(G_M)$ which is not of the form $a^m db^n e$ does not end with e.

Stotskii stated [Sto71, Corollary 1, page 96]:

Corollary 4.5 $\mathcal{L}(\text{P-left,CF}\text{–}\lambda, \text{ut})$ is not closed under intersection with regular languages.

He gave the following correct proof: Fix some nonrecursive enumerable set of numbers M. If $\mathcal{L}(\text{P-left,CF}\text{–}\lambda, \text{ut})$ were closed under intersection with regular languages, $L = L(G_M) \cap \{a, b, d\}^*\{e\}$ would lie in $\mathcal{L}(\text{P-left,CF}\text{–}\lambda, \text{ut})$. Now, $a^m d$ is a prefix of some word in L iff $m \in M$. Using some grammar G generating L, a TM can check '$m \in M$?' employing the solvability of the prefix problem for G.

Observe again that we need not obtain the grammar G effectively.

4.2 A Metatheorem of Hinz and Dassow

In 1989, Hinz and Dassow gave a nice metatheorem [HD89] stated in the following form:[8]

Claim 4.6 Let the emptiness problem of a language class $\mathcal{L}$ be decidable. Let $\mathcal{L}$ be closed under intersection with regular languages.

[8] As defined in [HD89], $VC(M_0)$ is the set of valid computation of a specific fixed TM M_0. The language L_0 considered in the Hinz/Dassow-Lemma is just the complement of $VC(M_0)$.

- Then $VC(M_0) \notin \mathcal{L}$. Moreover, if $\mathcal{L}$ contains the context-free languages, then $\mathcal{L}$ is not closed under complement and not closed under intersection.
- If $\mathcal{L}'$ is a language class such that every recursively enumerable language is the homomorphic image of a language in $\mathcal{L}'$, then is a language in $\mathcal{L}' \setminus \mathcal{L}$.

By this metatheorem, they solved the old problem in the field of regulated rewriting whether or not appearance checking enhances the descriptive power of programmed grammars.[9] Unfortunately, the formulation of the metatheorem contains one subtle drawback already encountered and analyzed in the previous section. First of all, we state two correct versions of the metatheorem.

Correction 4.7 Let $\mathcal{G}$ be a grammar family with the following properties:

- There is a TM which, given some $G \in \mathcal{G}$, decides whether $\mathcal{I}_{\mathcal{G}}(G) = \emptyset$ or not.
- There is a TM which, given some $G \in \mathcal{G}$ and a DFA E, constructs a $G' \in \mathcal{G}$ such that $\mathcal{I}_{\mathcal{G}}(G') = \mathcal{I}_{\mathcal{G}}(G) \cap L(E)$.

Then, we find:

1. $VC(M_0)$ cannot be described by a grammar from $\mathcal{G}$. If any context-free language can be described by a grammar from $\mathcal{G}$, then the corresponding language family $\mathcal{L}(\mathcal{G})$ is neither closed under complementation nor under intersection.
2. Let $\mathcal{G}'$ be another grammar family having the property that any enumerable language is the homomorphic image of a language described by a grammar in $\mathcal{G}'$. Then, there is a language generable by some $H' \in \mathcal{G}'$ that is not generable by any $H \in \mathcal{G}$.
3. There is a TM which, given some $G \in \mathcal{G}$ and some word w, decides whether $w \in \mathcal{I}_{\mathcal{G}}(G)$ or not. Hence, given some $G \in \mathcal{G}$, we know that $\mathcal{I}_{\mathcal{G}}(G)$ is recursive, since we can construct two TM's that enumerate $\mathcal{I}_{\mathcal{G}}(G)$ and its complement, respectively. Especially, $\mathcal{G}$ is constructible.

When analyzing the proof of this theorem, at first glance we seem to need more effective closure properties, since the emptiness problem is a problem without parameters. In the following formulation (and proof) of the theorem, it becomes clear that the predicate used in the proof is indeed a predicate with parameters, hence allowing the separation of language classes.

Correction 4.8 Consider the following predicate for languages:

$$P(L, E) = \begin{cases} \text{true} & \text{if } E \in \mathcal{DFA} \wedge L \cap L(E) = \emptyset \\ \text{false} & \text{otherwise} \end{cases}$$

Assume that P is decidable for the language family $\mathcal{L}$. Then, we find:

[9] Compare also to Theorem 3.13 above.

1. $VC(M_0) \notin \mathcal{L}$. If any context-free language is contained in $\mathcal{L}$, then $\mathcal{L}$ is neither closed under complementation nor under intersection.

2. Let $\mathcal{L}'$ be language family such that every recursively enumerable language is the homomorphic image of a language in $\mathcal{L}'$. Then, P is not decidable for $\mathcal{L}'$. Hence, there is a language in $\mathcal{L}' \setminus \mathcal{L}$.

3. Every language in $\mathcal{L}$ is recursive.

Proof.

1. Assume that $VC(M_0) \in \mathcal{L}$. Since P is decidable for $\mathcal{L}$, there is a TM which, given a DFA E, decides whether $VC(M_0) \cap L(E) = \emptyset$ or not. Since $VC(M_0) \cap L(E) = \emptyset \iff L(E) \subseteq L_0$, this contradicts the Hinz/Dassow-Lemma.

   If $\mathcal{L}$ contains the context-free languages, $\mathcal{L}$ is not closed under complement and not under intersection, because $VC(M_0)$ can be represented as the complement of a context-free language or the intersection of two context-free languages.

2. Assume $\mathcal{L}' \subseteq \mathcal{L}$. By Lemma 2.7, P is decidable for $\mathcal{L}'$. It is known that there is a TM which, given a DFA E, produces another DFA E' with $L(E') \subseteq h^{-1}(L(E))$. Let $L' \in \mathcal{L}'$ be such that $h(L') = VC(M_0)$. By assumption, there is a TM which, given E', decides whether $P(L', E')$ or not. But $P(L', E') \iff L' \cap h^{-1}(L(E)) = \emptyset \iff h(L') \cap L(E) = \emptyset \iff L(E) \subseteq L_0$, which contradicts the Hinz/Dassow-Lemma.

3. Any word w can be effectively transformed into a DFA accepting only w. Now, a TM can use the decidability of P for $\mathcal{L}$ in order to solve the word problem for $\mathcal{L}$.

□

Since the family of programmed grammars without appearance checking (with or without λ-productions; with left derivation or free interpretation) satisfies the conditions of $\mathcal{G}$ in the metatheorem, we may correctly conclude that the inclusion

$$\mathcal{L}(\text{P,CF}\text{–}\lambda) \subseteq \mathcal{L}(\text{P,CF}\text{–}\lambda, \text{ac})$$

is proper, as stated in [HD89, Corollary 5]. Moreover, the corresponding language classes are not closed under complementation and not closed under intersection.

Note that all grammar families considered in this section are both recursive and constructible. This means that noneffective components are not obviously included in these families.

4.3 A Study on the Chomsky Hierarchy

If some predicate P is decidable for the language family $\mathcal{L}$ but not for $\mathcal{L}'$, we can conclude that $\mathcal{L} \neq \mathcal{L}'$. In this way, the strictness of the Chomsky hierarchy is proved below without using the traditional pumping lemmata.

In order to distinguish regular and context-free languages, we consider the following predicate P_r.

$$P_r(L, E) = \begin{cases} \text{true} & \text{if } E \in \mathcal{DFA} \wedge (\Delta^* \setminus L) \cap L(E) = \emptyset \\ \text{false} & \text{otherwise} \end{cases}$$

It is easily seen that P_r is decidable for $\mathcal{DFA}$: Given two DFA's A_L and E, a TM may first construct a DFA A'_L accepting $\Delta^* \setminus L$ and then construct another DFA A accepting the intersection of the languages $L(A'_L)$ and $L(E)$. Moreover, a TM may check whether $L(A)$ is empty or not. Hence, P_r is decidable for the class of regular languages $\mathcal{L}(\text{REG})$.

In [HD89], Hinz and Dassow constructed a context-free language $L_0 \subseteq \Delta^*$ with the property that there is no TM T_{L_0} such that T_{L_0}, given some DFA E, decides whether $(\Delta^* \setminus L_0) \cap L(E)$ is empty or not. Hence, P_r is not decidable for the class of context-free languages $\mathcal{L}(\text{CF})$.

Note that this result is true for any Δ containing more than one letter, although Hinz and Dassow used a rather large alphabet in their proof. Namely, both the family of DFA's and the family of context-free grammars is effectively closed under gsm mappings and inverse gsm mappings, enabling an effective coding/decoding of large alphabets into two-letter-alphabets.

Therefore, we have shown:

- The family of regular languages over Δ does not coincide with the family of context-free languages over Δ if $|\Delta| > 1$.

This result could also have been shown by the following argument. We consider the following predicate $P(L, E)$: $P(L, E)$ is true iff the language accepted by the given DFA E is contained in L.

It is well-known that there is a TM which, given as input two DFA's E' and E, decides whether $L(E') \subseteq L(E)$ or not. In other words, P is decidable for the grammar family $\mathcal{DFA}$. Therefore, P is decidable for the family of regular languages.

If P were decidable for the family of context-free languages, in particular, there would be a TM which, given as input a DFA E, decides whether $L(E) \subseteq L_0$ or not, where L_0 is defined as in the Hinz/Dassow-Lemma. Obviously, this contradicts the Hinz/Dassow-Lemma.

Furthermore, L_0 is a concrete example of a non-regular context-free language.

Next, we consider the following predicate for languages:[10] $P_p(L, v)$ is true iff v is the prefix of some word in L.

[10] The idea to consider this predicate was taken from [Sto71].

First, we show that P_p is decidable for the class of context-free grammars. By [HU79, Section 4.4], we know that a TM can transform any given context-free grammar G into an equivalent one G' containing no useless variables (if $L(G)$ is not empty), additionally indicating whether $L(G)$ is empty or not. If $L(G)$ is empty, v cannot be prefix of $L(G)$. Otherwise, a TM may list all sentential forms of length $< |v| + k$ (where k is the length of the longest right-hand side of a production in G') derivable by G' and check whether v is prefix of one of these sentential forms. If it is so, $P_p(L(G), v)$ is true; otherwise, it is false.

Secondly, we show that P_p is not decidable for $\mathcal{L}(\mathrm{CS})$. It can be proved like [Sal73, Theorem III.9.8] that any enumerable language L is 'context-sensitive with tails', i.e. there is a context-sensitive language M such that $w \in L$ iff $wba^i \in M$ for some $i \geq 0$ (where a, b are 'new' symbols). Now, wb is a prefix of a word in M iff w is contained in L, where L may be nonrecursive. Hence, P_p is not decidable for the class of context-sensitive languages. Again, note that the fact that in our proof the alphabet of the context-sensitive languages contains at least three symbols is not essential and can be overcome by a suitable coding/decoding.

Hence, we have proved:

- The family of context-free languages over Δ does not coincide with the family of context-sensitive languages over Δ if $|\Delta| > 1$.

Finally, consider the following properties of the family of context-sensitive grammars $\mathcal{G}$:

- $\mathcal{G}$ is a constructible and enumerable grammar family with effective complementation. Then, Theorem 3.14 proves that $\mathcal{G}$ does not characterize the recursive languages over Δ.

- $\mathcal{G}$ is a constructible and enumerable grammar family with effective complementation describing recursive languages only. By Theorem 3.16, the word problem P_w is decidable for $\mathcal{G}$.[11] By Theorem 3.15, $\mathcal{G}$ does not characterize the recursive languages.

It is well-known that the word problem P_w is not decidable for the family of enumerable languages $\mathcal{L}(\mathrm{RE})$, but it is decidable for $\mathcal{L}(\mathrm{REC})$ by definition.

This completes our proof of the strictness of the Chomsky hierarchy using decidability arguments only.

[11] This was of course known before [Sal73].

5 Conclusions

We hope that this paper revives research regarding fundamentals of formal language theory. Maybe, grammar families, formalized via a pair consisting of (1) a meta-language describing the syntax of the grammars and (2) an interpretation function, can serve as a starting point of such investigations. In particular, such research might re-connect the theory of formal languages with recursion theory.

For someone viewing from the field of recursion theory, it might also be interesting to consider relativizations of our observations. For example, the proof of Theorem 3.14 may be relativized (in the sense of recursion theory) leading to the following more general result.

Theorem 5.1 Let $m, n \geq 0$, $k := \max(m, n)$. Let $\mathcal{G}$ be a grammar family characterizing the languages over Δ recursive in $\emptyset^{(k)}$. If $\mathcal{I}_{\mathcal{G}}$ is computable in $\emptyset^{(n)}$ and $L_{\mathcal{G}}$ is enumerable in $\emptyset^{(m)}$, then complementation is not computable in $\emptyset^{(k)}$. □

As usual "computable in $\emptyset^{(0)}$" simple means computable. "Computable in $\emptyset^{(1)}$" means that a TM, additionally endowed with a read-only tape (oracle) containing a 1 (0) on the n^{th} cell iff the TM with number n stops (does not stop) when working on the input n. Since we can diagonalize the TM's computing in $\emptyset^{(1)}$ as well, we also get oracle models for computability in $\emptyset^{(2)}$, $\emptyset^{(3)}$ and so on. A language L is "enumerable in $\emptyset^{(n)}$" if there is a TM enumerating L with the help of the oracle $\emptyset^{(n)}$. A language L is "recursive in $\emptyset^{(n)}$" if there is a TM which, given a word w, decides whether $w \in L$ or not, with the help of the oracle $\emptyset^{(n)}$.

Furthermore, it is interesting to pursue the question under which condition one should reject a grammar family for being 'noneffective' or 'unnatural'. E.g., one might require that if it is known that a language family is closed under some 'natural' operation, any 'natural' grammar family characterizing this particular language family should be effectively closed under this operation. With this definition in mind, we may conclude that there is no 'natural' grammar family characterizing the recursive languages.

Finally, it would be interesting to give an example of two recursive and constructible grammar families characterizing the same language family but being Turing inequivalent. Likewise, it would be intriguing to show that there is no such example.

Acknowledgements

We want to thank all our colleagues and formal language specialists with whom we had stimulating discussions on this subject during the last months.

References

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

[Fer91] H. Fernau. On function-limited Lindenmayer systems. *J. Inf. Process. Cybern. EIK (formerly Elektron. Inf.verarb. Kybern.)*, 27(1):21–53, 1991.

[Fer94] H. Fernau. On grammar and language families. To appear in Fundamenta Informaticae.

[HD89] F. Hinz and J. Dassow. An undecidability result for regular languages and its application to regulated rewriting. *EATCS Bulletin*, 38:168–173, 1989.

[HU79] J. E. Hopcroft and J. D. Ullman. *Introduction to automata theory, languages, and computation*. Reading (MA): Addison-Wesley, 1979.

[Imm88] N. Immerman. Nondeterministic space is closed under complementation. *SIAM Journal Comput.*, 17(5):935–938, 1988.

[Odi89] P. Odifreddi. *Classical Recursion Theory*, volume 125 of *Studies in Logic and Foundations of Mathematics*. Amsterdam: North Holland, 1989.

[Ros69] D. J. Rosenkrantz. Programmed grammars and classes of formal languages. *Journal of the Association for Computing Machinery*, 16(1):107–131, 1969.

[Sal73] A. K. Salomaa. *Formal Languages*. Academic Press, 1973.

[Sto71] E.D. Stotskii. Формальные грамматики и ограничения на вывод. Проблемы передачи информации, VII(1):87–101, 1971.

[Sze88a] R. Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26:279–284, 1988.

[Sze88b] R. Szelepcsényi. The method of forcing for nondeterministic automata. *EATCS Bulletin*, 33:96–100, 1988.