Regular Growth Automata: Properties of a class of finitely induced infinite machines

Christian Wurm
Fakultät für Linguistik und Literaturwissenschaften, Universität Bielefeld
CITEC Bielefeld
cwurm@uni-bielefeld.de

June 7, 2011

Abstract

We present a class of infinite automata, in which all local computations are performed by finite state machines. These automata characterize an abstract family of languages which does not seem to coincide with any other known class, and which seems to cut across the Chomsky hierarchy. We show results regarding recognizing power and closure properties, and sketch the use of machine growth as a refined measure of complexity with respect to some well-known measures.

1 Introduction

We introduce a new type of infinite state automaton, where all local computations are performed by finite state automata (FSA): transition relations are computed by synchronous transducers; accepting states are recognized by a FSA. We have to distinguish the deterministic and the non-deterministic case: the former describes an abstract family of languages, but also the latter has some interesting properties.

We start out by presenting our linguistic motivation, which for reasons of space might be shorter than necessary. The main body of work in this paper is the following: we show that our automata compute classes of languages interesting from a formal point of view as well as from a linguistic point of view, being situated somewhere near the mildy context-sensitive languages (in the sense of MCFG-recognizable) and the PTIME languages; then we present their closure properties, and how they are related to some similar, but non-equivalent concepts. In the end, we will only briefly touch upon a possible application as a refined measure of complexity for formal languages, leaving application in linguistic theory for future work.

Acknowledgements The author would like to thank the three reviewers for their constructive comments, and Marcus Kracht, without whose support this paper would not have been written.

2 Linguistic Motivation

Our linguistic motivation is the following: generally, in formal language theory things get interesting only beyond "a certain constant k"; anything which is constantly bounded can in principle be factored out in some way or other, and therefore is of no relevance to formal complexity. In natural languages, on the contrary, most really interesting structures are only observed within a certain local bound: the applicability of complex rules often seems to be sensitive to the context of application.

Interestingly, this single phenomenon is traditionally attributed to two entirely different sources: one source is "performance restrictions", that is, most prominently, restrictions on working memory. The most (in)famous case in point is multiple center embedding, as exemplified in the following example:[1]

(1) people people people see, see, see.

English syntax gives rise to sentences of the form $people^n\ see^n$; however only under the assumption, that relative clause embedding is *recursive* - while as a matter of fact, speakers do not understand structures of this type for $n > 2$.

A case of context-sensitivity which is attributed to an entirely different source is WH-movement. The most famous case in point are the so-called island-constraints, which form a core aspect for most linguistic theories and rule out sentences as the following:

(2) *Who is she sad because died?

The distinction of the two sources is of course not arbitrary: there are two different kinds of complexity which underly these phenomena. In the first case, the crucial point seems to be the complexity of a parser configuration, that is, the amount of information the parser needs to memorize at a certain point, while analyzing incrementally. In the second case, the crucial point is the complexity of the rule schema.

Though these two phenomena are generally thougt to be somehow interrelated (as pointed out by [3]), in formal linguistics restrictions of the first kind are mostly considered not to be interesting from a theoretical point of view (but see [6],[11]). This attitude seems to be partly due to the fact that we lack tools to give theoretically interesting descriptions of such phenomena: once one assumes abstract invisible structures as trees or feature structures, which are encoded in our utterances, one loses sensitivity to the "raw" string context, and only regains it at the price of even more abstract (and unelegant) rules. What we attempt is to develop an account in which these both kinds of complexity are

[1] I owe this particular example to Jens Michaelis

directly related. Therefore, we try to work without abstract structures beyond mere strings.

The price we have to pay is: we no longer have a finite set of categories (somewhat sloppily identifying automaton states with non-terminals of a regular grammar). What we gain is: we characterize complex structures in terms of simpler ones; this allows us to remain aware of both complexity in the ("competence"-)sense of complexity of rule schemata, and in the ("performance"-)sense of string context or parser configuration. That way, we might at some point be able to reduce them to a common source.

3 Regular Growth Automata: Definitions

A finite state automaton is defined as a tuple $\langle q_0, Q, F, \Sigma, \delta \rangle$, with a finite state set Q. We will present an extension of this concept with an infinite state set $Q \subseteq \Omega^*$, for Ω a finite set of symbols. In FSA, states are atomic symbols like letters; we thus extend this concept to strings.[2] To preserve computability, we will require that transition relations on Q be *regular*. The following definitions follow [9], for more mathematical background see [2]. We confine ourselves to binary relations, but the following concepts generalize to arbitrary relations without complications.

Definition 1 *Put $\Sigma_\perp := \Sigma \cup \{\perp\}$, for $\perp \notin \Sigma$. The **convolution** of a tuple of strings $\langle \vec{x}_1, \vec{x}_2 \rangle \in (\Sigma^*)^2$, written as $\otimes\langle \vec{x}_1, \vec{x}_2 \rangle$ of length $max(\{|\vec{x}_i| : i \in \{1, 2\}\})$ is defined as follows: the kth component of $\otimes\langle \vec{x}_1, \vec{x}_2 \rangle$ is $\langle \sigma_1, \sigma_2 \rangle$, where σ_i is the k-th letter of $\vec{x}_i$ provided that $k \leq |\vec{x}_i|$, and $\perp$ otherwise.*
*The **convolution of a relation** $R \subseteq (\Sigma^*)^2$, written $\otimes R$, is the the set $\{\otimes\langle \vec{x}_1, \vec{x}_2 \rangle : \langle \vec{x}_1, \vec{x}_2 \rangle \in R\}$.*

Definition 2 *A relation $R \in (\Sigma^*)^2$ is called **regular**, if there is a finite state automaton over $\Sigma_\perp^2$ recognizing $\otimes R$.*

Regular relations are computed by finite state transducers which do not have ϵ transitions in only one projection $\pi_i(R)$, except for when this transition can be run through only once in a transduction. Recall that $\langle \epsilon, \epsilon \rangle$-transitions can be eliminated, whereas transitions with a tuple $\bar{t}$, where $\pi_i(\bar{t}) = \epsilon, \pi_j(\bar{t}) \neq \epsilon$, where $i \neq j$, in general cannot be eliminated and genuinely increase expresssive power; however, transducers with an upper *bound* for the ϵ-transition in one transduction can be synchronized, that is, changed in a way that all ϵ transitions are final.

We call the class of transducers which compute regular relations **synchronous transducers**. As in the sequel we will restrict ourselves to this subclass, we will omit the adjective, as long as there is no danger of confusion.[3]

[2] Usually, state sets are not regarded as stringsets generated by some mechanism, although there is some recent work in this direction, see for example [1]

[3] This restriction with respect to full expressive power of transducers is mostly motivated by favorable closure properties; regular relations form a Boolean algebra, transducer definable relations are not closed under intersection.

Definition 3 *We define a* ***regular growth automaton*** *(RGA) as a tuple* $\langle \epsilon, Q, F, \delta, \Sigma, Op_\Sigma, \Omega \rangle$*, where* Ω *is a finite set of symbols, the state alphabet,* $Q \subseteq \Omega^*$ *is the state set,* $\epsilon \in Q$ *is the initial state,* $F \subseteq \Omega^*$ *is the set of accepting states,* $\delta \subseteq Q \times \Sigma \times Q$ *the transition relation,* Σ *a finite input alphabet;* Op_Σ *is a set of synchronous transducers, with one* op_x *for each* $x \in \Sigma$*, where* Ω *is the input and output alphabet for all* $op_x \in Op_\Sigma$*. In the sequel, we identify the* $op_x \in Op_x$ *with the relations they induce on* Ω*. In addition, the following hold:*

1. *F is a regular set;*
2. *for every transducer op_x, $((q_i, x), q_j) \in \delta$, exactly if $q_j \in op_x(q_i)$.*
3. *Q is recursively defined as the smallest set such that (i) ϵ is in Q, (ii) if α is in Q, then for all $x \in \Sigma$, $op_x(\alpha)$ is also in Q.*

We call the transducers in Op_Σ ***letter operators****.*

We will sometimes say that an automaton is regular or not, and speak of its regularity, as shorthand for: it is a regular growth automaton etc. An alternative, more restrictive definition for point *1.* would be: *1.'* F is a star-free set. For everything we do in this paper, it does not make a difference, but under certain circumstances it will. Star-free languages form a proper subset of the regular languages; but here we make no use of full expressive power of regular languages. We use this concept, because it is better known and maybe more natural in terms of automata (star-free languages are generated by so-called counter-free automata). The star-free languages are preferable if we adopt a logical perspective: synchronous regular relations are first order definable, and so are the star-free languages, whereas regular languages are not (for a comprehensive reference, see [7]).

A regular growth automaton is deterministic exactly if the letter operators represent (partial) functions, that is, for any input compute (at most) one output. The RGA is total exactly if the letter operators represent total functions/relations. We will consider partial and total automata without paying much attention to the difference; we can easily totalize RGAs by totalizing the letter operators, sending all previously undefined inputs to a new absorbing state. But we have to consider deterministic and non-deterministic automata separately, the classical determinization algorithm via powerset construction does **not** preserve the regularity of the automaton; in addition, in the worst case the state set would become uncountable. We conjecture that the latter properly include the former (in the sense of recognized languages, which we define below). We will write *RGA* ($\mathfrak{RGA}$) or regular growth automaton if we make statements valid for both cases; we will use *DGA* ($\mathfrak{DGA}$) for deterministic, *NGA* for non-deterministic automata.

For the description of regular growth automata, note the importance of the letter operators: when they are given, Q and δ are fixed, as well as Σ and Ω. The only additional information we need is the set of accepting states F. Note the following: we can specify a set F which is not a subset of Q. The accepting

states effectively reachable are given by the intersection $Q \cap F$. For the manner in which we define acceptance below, this is not a problem, but still noteworthy, as Q is in general not a regular set, and so neither is $Q \cap F$!

We generalize the transition function δ from letters to string in the obvious fashion. Then we have the following:

Definition 4 *A regular growth automaton RGA recognizes a language L, if* $L = \{\vec{x} : \delta_{RGA}(\epsilon, \vec{x}) \in F_{RGA}\}$.

We write $L(RGA)$ for the language a given automaton recognizes, and RGA_L for an automaton which recognizes a given language L. Note that regular growth automata are in general infinite. We can establish a connection to finite state automata as follows. Let $L^{\leq n}$ denote $L \cap \Sigma^{\leq n}$, where $\Sigma^{\leq n} := \bigcup_{i=0}^{n} \Sigma^i$.

Definition 5 *We write* RGA_n *for a regular growth automaton whose state set* Q_n *is the smallest set satisfying the following conditions:*

1. *For* RGA_0*, we have* $Q_0 = \{\epsilon\}$*, and*
2. *for* RGA_{n+1}*, we have for all* $\sigma \in \Sigma$ *and for all* $q \in Q_n$*,* $op_\sigma(q) \in Q_{n+1}$.

Thus, RGA_n *computes only words of length* $\leq n$*. We write say* $\mathfrak{RGA}_n$ *to denote a given automaton* $\mathfrak{RGA}$ *whose state set is restricted to* Q_n.

Then for any $n \in \mathbb{N}$, RGA_n is a special case of a finite state automaton. We say a regular growth automaton is finite (without restriction), if there is a constant k, such that for the state set Q of $RGA_n, |Q| \leq k$ for $n \to \infty$. For every regular language R, there is a finite regular growth automaton such that $L(RGA) = R$, and if $R = L(RGA)$ for a finite regular growth automaton, then R is regular.

4 Recognizing Power

We define the class $\mathfrak{L}_{RGA}$ as the class of languages L for which there is a regular growth automaton RGA such that $L = L(RGA)$. We distinguish between the classes of deterministic regular growth automata and non-deterministic automata, which induce the classes $\mathfrak{L}_{DGA}$ and $\mathfrak{L}_{NGA}$, respectively. Obviously, the latter includes the former. We start by showing some examples of languages included in $\mathfrak{L}_{DGA}$, and then turn to $\mathfrak{L}_{NGA}$.

4.1 The deterministic case

We now turn to some interesting languages which *can* be recognized by some DGA.

Lemma 6 *Let* $L \subset \Sigma^*$*,* $\Sigma = \{a_1, a_2, ..., a_k\}$ *for arbitrary constant* k*. If* L *is of the form* $L := \{a_1^n a_2^n ... a_k^n : n \in \mathbb{N}\}$*, then there is a DGA which recognizes* L.

As a proof, we will treat one case, from which it will be easily seen how this works for the general case.

4.1.1 $\{a^n b^n c^n : n \in \mathbb{N}\}$

We show how to characterize $L_1 := \{a^n b^n c^n : n \in \mathbb{N}\}$, and how to generalize the procedure to arbitrary crossing dependencies. The letter operators work as follows:[4]

$$op_a : \begin{cases} \vec{x} \to \vec{x}0, & \text{provided } \vec{x} \in 0^* \\ \text{undefined otherwise} \end{cases} \tag{3}$$

$$op_b : \begin{cases} \vec{x}0\vec{y} \to \vec{x}1\vec{y}, & \text{provided that } \vec{x} \in 0^*, \text{ and } \vec{y} \in 1^* \\ \text{undefined otherwise} \end{cases} \tag{4}$$

$$op_c : \begin{cases} \vec{x}1 \to \vec{x}2 & \text{for arbitrary } \vec{x} \\ \vec{x}12 \to \vec{x}2 & \text{for arbitrary } \vec{x} \\ \text{undefined otherwise} \end{cases} \tag{5}$$

We put $F := \{2\}$. It is easy to see how to generalize this procedure: we simply use letter operators of the type of op_b for any of the inner letters, restricting their domain such that they only apply in the desired position. The letter operator for the initial/final letter works as op_a/op_c, respectively.

4.1.2 Copy language

We now consider the copy language, that is, $L_2 := \{\vec{x}\vec{x} : \vec{x} \in \Sigma^*\}$. We put $\Sigma := \{a, b\}$, but treat the language in a manner that can be easily generalized to arbitrary alphabets. This language is quite difficult to treat, so we will go a bit more into detail.

For L_2, we cannot map any two different words onto the same state, for there are no two different equivalent prefixes. In addition, for L_2, any arbitrary sequence $\vec{p} \in \Sigma^*$ is a prefix of the language, since $\vec{p}\vec{p} \in L_2$. So there is a bijection between prefixes/words of L_2 and Q, and transducers must be total functions in the domain Ω^*.

Furthermore, we cannot just mark entire substrings as copied or uncopied, for the following reason:

Definition 7 *$(\vec{y}, \vec{z})$ form a **c-decomposition** of $\vec{x}$, if*

1. *$\vec{y}\vec{z} = \vec{x}$, for $\vec{y}, \vec{z} \neq \epsilon$ and*
2. *$\vec{y} = \vec{z}\vec{v}$, for some $\vec{v} \neq \epsilon$.*

If there is a c-decomposition of a word, then a prefix of it has already been copied. For our treatment of the copy language, we must encode all c-decompositions of a word in Σ^* when mapping it to a state-string in Q, as they provide the information which is necessary to compute the suffixes which result in a word of L_2. Now we have the following:

[4] In case the operators are simple enough, we prefer this way of writing them, for it is much easier to read. The diffident reader may convince himself that these function can be easily computed by synchronous finite state transducers.

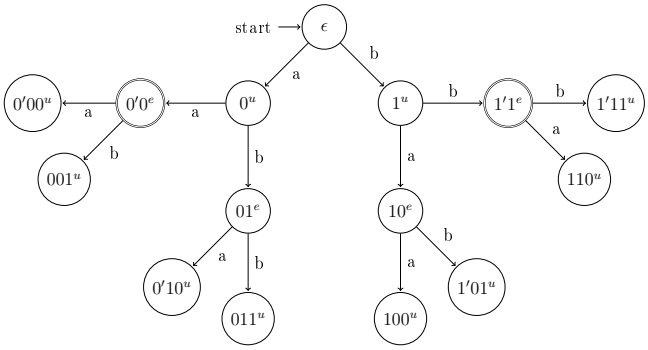
Lemma 8 *For any $n \in \mathbb{N}$, we find an $\vec{x} \in \Sigma^*$, such that $\vec{x}$ has n distinct c-decompositions.*

Proof. To see this, imagine a string as a kind of fractal: let $\vec{x}$ be of the form $\vec{x'}\vec{z}\vec{x'}$. It has at least one c-decomposition. Now, imagine $\vec{x'}$ is again of the form $\vec{x''}\vec{z}\vec{x''}$; then we have at least two c-decompositions. This procedure can be arbitrarily iterated. ⊣

This means, when mapping strings to states we need to encode (i) the information about the letters in the word, and (ii) an unbounded amount of information on possible c-decompositions.

We do this by granting us two letters in Ω for each letter in Σ, say $\{0, 0', 1, 1'\}$. As we add new letters to a string, each marker for a c-decomposition will move (starting from the left edge) one to the right, or be discarded. In addition, we put a single marker into each state-string, which markes the middle of a string with an uneven number of letters, and the left edge of the right half of a string with an even number of letters (both with distinct symbols, of course); Ω contains thus additional letters $\{0^u, 0^e, 1^u, 1^e\}$. We know that $\vec{x}$ is in L_2, exactly if a decomposition marker has reached the right edge of the first half of a string with an even number of letters, that is, is adjacent to the "even" maker; consequently, we put $F := \Omega^*(0'|1')(0^e|1^e)\Omega^*$.

We present the transition function for the letter operator in the appendix, as it is quite hard to read, and instead show the initial tree of the resulting regular growth automaton:



4.1.3 $\{a^{2^n} : n \in \mathbb{N}\}$

Let $L_3 \subset a^*$ be the language $\{a^{2^n} : n \in \mathbb{N}_0\}$.

Lemma 9 *There is a DGA such that $L(DGA) = L_3$.*

Proof. We put $\Omega := \{x, y\}$. op_a works as follows:

$$op_a : \begin{cases} \epsilon \to x \\ x^n y^m x^o \to x^{n+1} y^{m-1} x^{o+1}, & n, o \geq 0, m > 0 \\ y^n x^m y^o \to y^{n+1} x^{m-1} y^{o+1}, & n, o \geq 0, m > 0 \end{cases} \tag{6}$$

We have thus a sequence of states:

(7) $\epsilon, x, yy, xyx, xxxx, yxxxy, yyxxyy, yyyxyyy, yyyyyyyy$ etc.

We put $F := x^* \cup y^*$. ⊣

Note that the set of accepting states which is actually reached is only a small subset of F, namely: $L_F := \{x^{2^{2n}}, y^{2^{2n+1}} : n \in \mathbb{N}_0\}$. This set is not semilinear, and as semilinear languages are closed under intersection, we have the following corollary:

Corollary 10 *The state set Q of an approximative automaton need not be semi-linear. More generally: The transitive closure of a word under regular relations, $\bigcup_{i\in\mathbb{N}} \vec{y} : \vec{x}T^i\vec{y}$, for $\vec{x}$ an arbitrary word and T a regular relation, is not semilinear.*

4.2 The non-deterministic case

Let $\mathfrak{L}_{CF}$ be the class of languages L for which there is a context-free grammar G such that $L = L(G)$.

Lemma 11 $\mathfrak{L}_{CF} \subset \mathfrak{L}_{NGA}$.

Proof. We show inclusion: assume without loss of generality that G is in Greibach normal form. For all rules of the form $S \to x\ \vec{N}$, put $op_x(\epsilon) = \vec{N}^T$ (for $\vec{N}$ a possibly empty string of non-terminals, $(-)^T$ the transpose). For all rules of the form $N \to y\ \vec{M}$, put $op_y(\vec{O}N) = \vec{O}\vec{M}^T$. Put $F := \{\epsilon\}$.[5] As we have finitely many rules, operators remain finite state. ⊣

There is strong evidence that $\mathfrak{L}_{DGA}$ does *not* contain $\mathfrak{L}_{CF}$, which we will present in another paper. That would also mean that $\mathfrak{L}_{DGA}$ yields a true cut across the Chomsky-hierarchy.

As we said, we have no general algorithm to convert an NGA into a DGA. We conjecture that there is a proper inclusion not only of the classes of automata, but also of the languages they recognize:

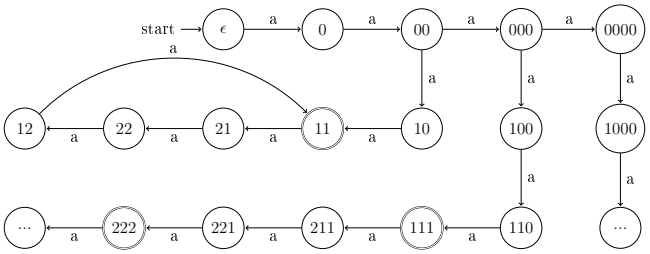
Conjecture 12 $\mathfrak{L}_{NGA} \supsetneq \mathfrak{L}_{DGA}$

As one example of a language for which there is a NGA, but seems to be no DGA we take $L_{nop} := \{a^n : n \text{ is not a prime}\}$. For simplicity, we directly show the (initial fragment of) the automaton, as the transducers are hard to read and to interpret.

We put $F := 1^* \cup 2^*$. This automaton works as follows: one branch goes simply along all numbers, construing $\{0^n : n \in \mathbb{N}\}$. In addition, we have transitions from each 0^k into a finite subautomaton, which recognizes all $a^m, m \in \bigcup_{i\geq 2}\{k \cdot i\}$.

As we will see later on, the class $\mathfrak{L}_{DGA}$ is closed under complement; therefore, if $L_{nop} \in \mathfrak{L}_{DGA}$, then for $L_{prime} := \{a^n : n \text{ is a prime number}\}$, we also have $L_{prime} \in \mathfrak{L}_{DGA}$.

[5] In case we do not want the empty string in L, we need an extra symbol and an extra mapping.



We strongly conjecture that $L_{prime} \notin \mathfrak{L}_{DGA}$; unfortunately we still lack a conclusive proof. Note also that we do not know whether $\mathfrak{L}_{NGA}$ is closed under complement; so we do not know whether $L_{prime} \in \mathfrak{L}_{NGA}$.

5 Closure Properties

5.1 Boolean closure

We show the following:

Theorem 13 *The class $\mathfrak{L}_{DGA}$ is closed under union, intersection and complement. The class $\mathfrak{L}_{NGA}$ is closed under union and intersection.*

Proof. We start with union. We give an algorithm which works for both the deterministic and non-deterministic case, which preserves determinism if both automata are deterministic.

Given two automata, $\mathfrak{RGA}_1, \mathfrak{RGA}_2$, firstly we totalize $\mathfrak{RGA}_1$ and $\mathfrak{RGA}_2$ by adding an absorbing state $\diagdown\!\!\diagup$. Without loss of generality, assume that $\Sigma_1 = \Sigma_2$.[6] If this is not the case, we simply align their alphabets by adding the missing letters σ, and constructing new letter operators op_σ where $op_\sigma(\vec{x}) = \diagdown\!\!\diagup$ for all $\vec{x} \in \Omega_i^*$. We form Ω_3 over the alphabet $\Omega_1 \times \Omega_2$, $Q_3 \subseteq \Omega_3^*$. Our initial state is $\langle \epsilon, \epsilon \rangle$.

We construe the operators in Op_Σ^3 as follows (note that for the first time time, we look inside the transducers; do not confuse transducer states, transitions etc. with *RGA*-states!): Take the cartesian product, $Q_a^1 \times Q_a^2$ as state set of op_a^3; the pair of their initial states as new initial state, and the pairs of final states as its final states. The transducers op_σ^3 now operate on the tuples in Ω_3; and the new transition function $\delta^3_{op_a}$ is defined as follows: $(\langle x', y' \rangle, \langle q_{i'}, q_{j'} \rangle) \in \delta^3_{op_a}(\langle q_i, q_j \rangle, \langle x, y \rangle)$, iff $((q_i, x)) \in \delta^1_{op_a}(q_{i'}, x')$ and $((q_j, y)) \in \delta^2_{op_a}(q_{j'}, y')$. This procedure preserves functionality of the operators, and therefore determinism in the resulting $\mathfrak{RGA}_3$. Back in $\mathfrak{RGA}_3$, we now have $\langle q_1, q_2 \rangle \in \delta_3(\epsilon, w)$ iff $q_1 \in \delta_1(\epsilon, w)$ and $q_2 \in \delta_2(\epsilon, w)$.

[6] If $\Sigma_1 \cap \Sigma_2 = \emptyset$, there is a much quicker procedure: First, we have to take care that that the two alphabets Ω_1 and Ω_2 of the letter operators are distinct. Then just put $RGA_3 = \langle \epsilon, F_1 \cup F_2, Q_1 \cup Q_2, \Sigma_1 \cup \Sigma_2, Op_{\Sigma_1} \cup Op_{\Sigma_2}, \Omega_1 \cup \Omega_2 \rangle$. This does the job as required, and as the alphabets are distinct, there is no interference of the letter operators.

We put $F_3 := \{\otimes\langle\vec{x},\vec{y}\rangle : \vec{x} \in F_1 \text{ or } \vec{y} \in F_2\}$.[7] We thus accept any state, for which the left projection would result in F_1 or the right projection would be F_2. Note that we do not enhance our recognizing power by forming states over tuples.

It is obvious how intersection is formed: just put $F_3 := \{\otimes\langle\vec{x},\vec{y}\rangle : \vec{x} \in F_1$ and $\vec{y} \in F_2\}$. This requires that each projection be a member of its original accepting state set.

Both of these constructions work equally for deterministic and non-deterministic regular growth automata. The construction of an *RGA* for the complement of a language $L = L(\mathfrak{RGA})$ is straightforward in the deterministic case: for $L = L(\mathfrak{DGA})$, we first totalize $\mathfrak{DGA}$. Let $(-)^C$ denote the set-theoretic complement. Then $L^C = L(\mathfrak{DGA}^C)$, where $\mathfrak{DGA}^C := \langle\epsilon, Q, Op_\Sigma, \Sigma, \Omega, \delta, F^C\rangle)$. ⊣

For a non-deterministic automaton $\mathfrak{NGA}, L = L(\mathfrak{NGA})$, it does not seem possible to construct an *RGA* which recognizes L^C. The reason is that not all states which are reached by words in $L(\mathfrak{NGA})$ must be in F, but only some.

5.2 Homomorphism

Lemma 14 *$\mathfrak{L}_{RGA}$ is closed under ϵ-free homomorphism.*

Proof. If our homomorphism maps two letters, say a and b, onto one, say a, then we construct our operator op'_a for $h(L)$ as follows: let $R \subseteq \Omega^* \times \Omega^*$ be the regular relation described by op_a, S be the relation of op_b. Then op'_a describes the relation $S \cup R$, which in general is no longer a function, even in case the original operators where functional. This can be iterated for any homomorphism arbitrarily. ⊣

This does, however, not work any more if we map a letter to ϵ. Assume we map a to ϵ. Then, the set of states associated with the empty string in $h(L)$ is $\bigcup op_a^*(\epsilon)$, which need not be a regular set. The same holds for the state sets associated to all other words. Therefore, we cannot make sure that transition relations are regular.[8]

Lemma 15 *Both $\mathfrak{L}_{DGA}$ and $\mathfrak{L}_{NGA}$ are closed under inverse homomorphism.*

Proof. Let h be a homomorphism. If $h(a) = h(b) = a$, and $h(L) \in \mathfrak{L}_{RGA}$, then $op_{a_{h(L)}} = op_{a_L} = op_{b_L}$. This can be iterated arbitrarily. For the case $h(a) = \epsilon$, let op_a compute the identity function. This does the job as required, and preserves determinism: we do not have more outputs for a single operator than before. ⊣

[7] For the sake of uniformity, we here use synchronous relations; otherwise we would surpass the capacities of our letter operators and lose first order definability.

[8] On the other side, allowing empty operators would be a considerable extension with respect to our original definition, which we do not consider very reasonable for our purposes.

6 Related Concepts

6.1 Automatic Structures

The concept of automatic presentation of structures was introduced in [5]; an excellent more recent survey is provided by [9]. For comparison, we will restrict ourselves to structures of finite signature, that is, with a finite set of relation symbols.

Definition 16 *Let $\mathfrak{B} := (B; (R_i^{\mathfrak{B}})_{i \leq k})$ be a relational structure. An automatic presentation of $\mathfrak{B}$ consists of a mapping μ and a tuple of automata $\overline{M} := (M_A, M_=, ((M_i)_{i \leq k}))$, such that*

1. *M_A recognizes a set $A \subseteq \Sigma^*$,*
2. *the mapping $\mu : A \to dom(\mathfrak{B})$ surjective, and*
3. *for every atomic relation $R_i^{\mathfrak{B}}$ of $\mathfrak{B}$ (of arity r_i), the relation*
   $\mu^{-1}(R_i^{\mathfrak{B}}) := \{(w_1, ..., w_{r_1}) \in A^{r_i} : R_i^{\mathfrak{B}}(\mu(w_1), ..., \mu(w_{r_i}))\}$
   is regular and its convolution is recognised by the automaton M_i

We will identify functions with their graphs; this generalizes the concept to any structure, and instead of automatic presentation of a structure we will simply say automatic structure. We can conceive of the letter operators of an *RGA* as binary predicates, which fulfill the third condition. We have already seen that the state sets of regular growth automata do not fulfill the first two conditions; we will now present a stronger result which shows that this results in a genuine increase of power for generation of graphs as well as of languages. We use a theorem of [9], which allows us to substitute *surjective* in the second condition with *bijective*. As we only need to consider binary relations, we will introduce the notion of an automatic graph:

Definition 17 *An automatic structure $\mathfrak{B} := (B; (R_i^{\mathfrak{B}})_{i \leq k})$ is an **automatic graph**, if for all $R_i^{\mathfrak{B}}$, $r_i = 2$.*

We interpret the object domain as vertices, predicates as edges. We do the same thing for regular growth automata, and speak of a **regular growth graph** (*RGG*). It is clear that if an automatic structure is equivalent to a regular growth graph, then it is an automatic graph.

We show that there are (deterministic) RGGs, minimal with respect to some language L, for which there is no isomorphic automatic graph. This form of non-equivalence is quite strong and entails (among other) that there is no sense in which automatic graphs generate the class of languages $\mathfrak{L}_{RGA}$.

Recall the copy language L_2, over Σ; we put $L_c := \{xc : x \in L_2, c \notin \Sigma\}$. We show that the RGG of L_c is not automatic. The graph is easily constructed: for Op_Σ, leave everything as before, and define op_c as follows: the only strings it accepts as input are the strings formerly being in F, and the only output it gives

is $\top$ (a symbol not used by any other operator). The new set F for RGA_{L_c} is $\{\top\}$. Thus, op_c is similar to the chararcteristic function of Q.

Now, assume this graph is automatic. Then the state set Q forms a regular set. Let L_{op_c} be $\{\vec{x} : op_c(\vec{x}) = \top\}$. As regular sets are closed under intersection, we know that $R := Q \cap L_{op_c}$ is again regular. Regular sets are as well closed under regular relations. Now we know that every $q \in Q - \top$, there must be a bijective mapping $b : \delta(\epsilon, w) \to w$, which at least in this direction is regular. This is mandatory, for otherwise the operators would not know whether they are copying a letter or not. Let T be a transducer effecting b. By assumption, $T(R)$ again must be a regular language. But $T(R)$ is the copy language itself, which is not regular, as is well known. This is the desired contradiction.

On this occasion, we see that RGGs are interesting in itself: a sets nodes Q_L, such that the paths $\{(\epsilon, q) : q \in Q_L\}$ carry words from a language $L \in \mathfrak{L}_{RGA}$, are precisely those which can be mapped onto a single node. We also easily see that every RGA is equivalent to an NGA where F contains only single node.

6.2 Automaticity

We now introduce the concept of automaticity, which has a tradition which goes back to [12], but has been most extensively studied by [10] and a series of subsequent papers by the same first author.

The concept of automaticity of a language is as follows: It describes the size the smallest finite state automaton, which recognizes words of a given language up to a certain length n. Abstracting over n, *automaticity* $A_L(n)$ is a function over n which describes the number of states needed in order to recognize $L^{\leq n}$. This is thus a measure which tells us, how close the characteristic function of the language is to a finite state function. As there can be considerable differences in size between the smallest deterministic FSA and the smallest non-deterministic FSA for the same language, we have to distinguish between the deterministic and non-deterministic case, $DA_L(n)$ and $NA_L(n)$ We put $|\mathfrak{A}| := |Q|$, for Q the stateset of a finite state machine (or transducer) $\mathfrak{A}$.

(8) $DA_L(n) = min\{|\mathfrak{A}| : \text{for } \mathfrak{A} \text{ a deterministic } FSA \text{ and } L(\mathfrak{A}) \cap \Sigma^{\leq n} = L \cap \Sigma^{\leq n}\}$

(9) $NA_L(n) = min\{|\mathfrak{A}| : \text{for } \mathfrak{A} \text{ a non-deterministic } FSA \text{ and } L(\mathfrak{A}) \cap \Sigma^{\leq n} = L \cap \Sigma^{\leq n}\}$

We will also write $A_L(n)$ if we make statements valid for both cases. Nerode-equivalence $\sim_L$ is defined as follows:

(10) $\vec{x} \sim_L \vec{y}$ iff for all $\vec{z}$, $\vec{x}\vec{z} \in L$ iff $\vec{y}\vec{z} \in L$.

For $DA_L(n)$, we map all Nerode-equivalent prefixes onto one state; deterministic automaticity as a function tells us how many equivalence classes of $\sim_L$ we have for $L^{\leq n}$ (for proof of this see [4]).

This measure of descriptive complexity has however two serious shortcomings, when we apply it to linguistic theory: firstly, it does not adress the question

of how to *construct* the automaton for $L^{\leq n}$; therefore, we do not even have a finite characterization of the language in question. Secondly, we know that it is a computable problem (see [10]) to determine $A_L(n)$ for any $L^{\leq n}$. Automaticity does however not adress the question of how complex the computations are which are involved in construction the automaton. We have already remedied the first issue when using regular growth automata instead, for they provide finite characterizations of languages; we now adress the second one.

7 Automaton Growth: Refining Automaticity

We provide a refined version of automaticity for languages within $\mathfrak{L}_{RGA}(\mathfrak{L}_{DGA})$. Matters are a bit more complicated, as we have to take care that we consider growing automata, while we keep fixed letter operators.

Definition 18 *Let $\mathcal{RGA}_L^k$ be the set of regular growth automata, such that $L = L(RGA)$ for each $RGA \in \mathcal{RGA}_L^k$, and for all $op_\sigma^{RGA} \in Op_\Sigma^{RGA}$, $|op_\sigma^{RGA}| \leq k$, where $k \in \mathbb{N}$. The **k-automatic approximation complexity** of L is a function over the length n of words of L, given by the smallest number of states of the $RGA \in \mathcal{RGA}_L^k$, restricted to RGA_n:*

$$AAC_L^k(n) = min\{|RGA_n| : RGA \in \mathcal{RGA}_L^k\}$$

Definition 19 *We say that $\mathcal{RGA}_L^k$ is **convergent**, if $AAC_L^k(n) = (AAC_L^l(n) + i)$ for $n \to \infty$, any constant $l > k$ and some constant i. By $\mathcal{RGA}_L$ we denote $\mathcal{RGA}_L^i$, where i the smallest integer such that $\mathcal{RGA}_L^i$ is convergent; $\mathcal{RGA}_L$ is thus the smallest convergent set of regular growth automata for a given language.*

Obviously, if $l > k$ and $\mathcal{RGA}_L^k$ is convergent, then so is $\mathcal{RGA}_L^l$. Intuitively, that $\mathcal{RGA}_L^k$ is convergent means that larger letter operators do not reduce the growth of the automaton, except for finite fragments. Now we are ready to define the complexity $AAC_L(n)$ in general:

Definition 20 *The **automatic approximation complexity** of L is given by the smallest number of states of the $RGA \in \mathcal{RGA}_L$, restricted to RGA_n:*

$$AAC_L(n) = min\{|RGA_n| : RGA \in \mathcal{RGA}_L\}$$

Of course, we have to distinguish the deterministic and the non-deterministic case again; we will use $AAC_L(n)$ for the general case, which comprises both. For both cases it is clear that $A_L(n) \leq AAC_L(n)$. What we still have to prove is the following conjecture:

Conjecture 21 *There is an $L \in \mathfrak{L}_{RGA}$, such that $AAC_L(n) \gneq A_L(n)$.*

That is to say, we conjecture that there are languages, for which there are only regular growth automata with some states which could be deleted for every $L^{\leq n}$, which however cannot be deleted in a given automaton if we want to

preserve regularity *and* want it to recognize the entire language. We think a case in point could be L_{nop}, which we discussed above. Clearly, the minimal automaton for L_{nop} is at most linear; however, the arguments against a linear RGA for L_{nop} are the similar to those against a DGA: we are probably unable to filter out the prime factors on a straight line; we need at least one infinite branch to count up, from which we calculate the new factors we need to take into account, and one branch for each factor smaller than (half of) the number of ls we are computing. This is the reason we believe the above conjecture to be true, at least for the non-deterministic case.

A conclusive proof would be an interesting result: it would mean that in regular growth automata, complexity of automaton construction is mapped onto the size of the automaton.

8 Finite-state Philosophy

Why, could one ask, should the concept of FSA be that important to natural languages, when it is generally acknowledged that regular languages are insufficient to give a formal model of natural language? There are two answers, one is empirical, the other one theoretical. We start with the former. The class of regular languages, though definitely to weak, seems nonetheless to be of fundamental importance for natural languages: it is a well-known observation, that any structure, whose regularity cannot be captured by means of regular rules, is heavily restricted in its occurence in the sense we pointed out in the introduction (at least, we are not aware of any exception).[9] One can surely attribute this to plain performance restrictions; but one has to be clear that this does *not* "explain away" the fact; it simply attributes little importance to it.

This has been the common attitude in most of modern syntactic theory. On the contrary, we take the stance that one should attribute more importance to a phenomenon which is as persistent across speakers and languages. It has been very fruitful for syntactic theory to make some idealizations on recursive rule applicability - maybe at this point it will be fruitful to look what happens when we give up some of these idealizations. Importantly, we do not simply reject them, but rather try to be more aware of the point where we leave the firm basis which is provided by the observed data.

We come to the second point. One of the most remarkable properties of finite state state automata is that their state transitions describe a map from the input alphabet Σ^* to a finite monoid, which consists of the transition relations, and where concatenation is interpreted as relation composition. Letters and strings are thus mapped onto the transitions they induce on the automaton. That means that strings are mapped onto the same element, iff they can be interchanged in the language and context in question.

The other way round, this transition monoid of a language and an appropriate inverse mapping give rise to an automaton itself. Automaton states are thus

[9] Note that some authors less proficient in formal language theory even equate recursion itself with recursion of non-regular rules.

intimately linked to the algebraic structure of a language. If the monoid of non-equivalent syntactic contexts of a language is finite, it gives rise to an FSA. If the set is infinite, it gives rise to an infinite automaton - but in general, there is no effective computation for such a machine. This is precisely the problem we adress.

Coming back to the prior point: we think that linguistic knowledge mainly consists of a (finite) set of local routines with a (finite) set of appropriate contexts. As we have seen, from the algebraic point of view, this would amount to a regular language. But in our view, what we have in addition is the capacity to *abstract* new patterns from these finite routines. Abstraction is an inferential process, that is, it goes *beyond* our basic grammatical knowledge; therefore we find heavy restrictions on it. In our approach we can capture this by conceiving of regular growth automata as finite machines which only grow *if necessary*, and maybe not beyond a certain point or rate. In this paper, we have focused on the case where there is no restriction on growth; however, in the last section we have roughly pointed out how one can distinguish classes of descriptional complexity also by automaton growth.

We emphasize that this is but another way to look at the fact that natural languages are *almost* regular in the above sense - but maybe one which might give us a new perspective onto an old problem.

9 Conclusion and Further Work

We have characterized a class of automata with many interesting properties; the most important being surely effective computability. These automata characterize two classes of languages, of which one, $\mathfrak{L}_{DGA}$ is closed under Boolean operations, and does not seem to coincide with any other known class. In our view, $\mathfrak{L}_{DGA}$ is an interesting candidate for characterizing formal complexity of natural languages: even though this class does not seem to contain all context free languages, we get some characteristic languages from the class of mildly context sensitive and PTIME languages; this might be an interesting cut right across the Chomsky-hierarchy. There is an additional feature which makes $\mathfrak{L}_{DGA}$ particularly interesting: it combines the properties we already mentioned with the property, that there must be a single *common representation* for all parses of a string. In our treatment of the copy language, we have seen that there are strings which contain an arbitrary amount of "structural ambiguity", if we think of structure as the crossing dependencies of letters (the same holds, by the way, for the nested dependencies of the palindrome language, which we did not discuss). The reason there is a DGA for these languages is that we can represent all these "structures" in a single string-state. We find similar phenomena in natural language, where we have the difference between "genuine" garden paths and only "theoretical" garden paths, the former being close to incomprehensible (to the naive speaker), the latter not at all:[10]

[10] For some background, see [8].

(11) a. The horse raced past the barn fell.
b. She drove my aunt from London's car.

We hope to make this vague relation between states and structures in the canonical phrase structure sense more precise in further work; for there definitely is a need to make this relation more precise - this becomes clear as soon as we also want to cope with semantics. The conjecture that there might be a connection between such phenomena as garden paths and the theory of formal complexity of natural language brings us back to the point we mentioned in the very beginning: in linguistics, there is a fundamental distinction between "competence-complexity" on the one side, being a property of rules, and "performance-complexity" on the other, being a property of abstract parser configurations; we now add an additional kind: the complexity of a *set* of parser configurations for a single ambiguous sentence/prefix. We think it is the main advantage of our approach that, up to a certain point, all three kinds of complexity interact directly. We do not know whether this is the right way to understand the formal foundations of human language; but think this might be a way to gain new insights and generalizations.

10 Appendix: Transducers for the Copy Language

We now present the letter operators for L_2 in the form of a transition table for finite state transducers.

Table 1: Transition table of op_a, for states and input letters

	ϵ	0	$0'$	1	$1'$	0^u	1^u	0^e	1^e
q_0	$0^u, q_1$	$0', q_1$	$0', q_2$	$1, q_1$	$1, q_2$	$0, q_3$	$1, q_3$	$\emptyset$	$\emptyset$
q_1	$0, q_4$	$0, q_1$	$0, q_2$	$1, q_1$	$1, q_2$	$0, q_3$	$1, q_3$	$0^u, q_1$	$1^u, q_1$
q_2	$\emptyset$	$0', q_1$	$0', q_2$	$1, q_1$	$1, q_2$	$0', q_3$	$1, q_3$	$0^u, q_1$	$1^u, q_1$
q_3	$\emptyset$	$0^e, q_1$	$\emptyset$	$1^e, q_1$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
q_4	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
where $F := \{q_4\}$									

Note that the automaton is *total*, despite not all transitions are defined: most of the of the non-defined transition are simply unreachable. The only exception to this is q_4, which is the only accepting state; here, the partial definition provides functionality despite non-determinism. we have to make sure that only at the very end there is an empty input transition adding a 0.

q_0 is the only state allowed to introduce a new marker into the string, and is touched only once per transduction. q_1 and q_2 work together in transducing most of almost any of the input strings; q_2 remembers the "overhead" of a marker, and attaches it to the next letter or kills it. q_3 is needed to help in properly moving the middle index. Note that no decomposition marker ever trespasses a middle marker.

We obtain op_b by simply substituting 0 for 1 and some other minor changes in the above table: it only affects the transitions from q_2, where we change the distribution of the primes: for input $0/0'$, output is $0, q_1/q_2$; for input $1/1'$, output is $1', q_1/q_2$.

To see that this mapping does the job as required, note that we can send an unbounded number of markers to the right, each of them marking one possible c-decomposition.

References

[1] Stephane Demri, Alain Finkel, Valentin Goranko, and Govert van Drimmelen. Model-checking CTL^* over flat Presburger counter systems. *Journal of Applied Non-classical Logics*, 20:313–343, 2010.

[2] Christiane Frougny and Jacques Sakarovitch. Synchronized rational relations of finite and infinite words. *Theor. Comput. Sci.*, 108(1):45–82, 1993.

[3] John A. Hawkins. *A Performance Theory of Order and Constituency*. Cambridge Univ. Pr., Cambridge, 1994.

[4] Janis Kaneps and Rusins Freivalds. Minimal nontrivial space complexity of probabilistic one-way turing machines. In Branislav Rovan, editor, *MFCS*, volume 452 of *Lecture Notes in Computer Science*, pages 355–361. Springer, 1990.

[5] Bakhadyr Khoussainov and Anil Nerode. Automatic presentations of structures. In Daniel Leivant, editor, *LCC*, volume 960 of *Lecture Notes in Computer Science*, pages 367–392. Springer, 1994.

[6] András Kornai. Natural languages and the chomsky hierarchy. In *Proceedings of the 2nd European Conference of the ACL 1985*, pages 1–7, 1985.

[7] Robert MacNaughton and Seymour Papert. *Counter-free Automata*. M.I.T. Press, Cambridge, 1971.

[8] M. Marcus, D. Hindle, and M. Fleck. D-Theory: Talking about talking about trees. In *Proceedings of the 21st Annual Meeting of the ACL 1983*, pages 129–136, 1983.

[9] Sasha Rubin. Automata presenting structures: A survey of the finite string case. *Bulletin of Symbolic Logic*, 14(2):169–209, 2008.

[10] Jeffrey Shallit and Yuri Breitbart. Automaticity: Properties of a measure of descriptional complexity. In Patrice Enjalbert, Ernst W. Mayr, and Klaus W. Wagner, editors, *STACS*, volume 775 of *Lecture Notes in Computer Science*, pages 619–630. Springer, 1994.

[11] Edward P. Stabler. The finite connectivity of linguistic structure. In C. Clifton, L. Frazier, and K. Rayner, editors, *Perspectives on Sentence Processing*, pages 303–336. Lawrence Erlbaum, 1994.

[12] B. A. Trakhtenbrodt and Y. M. Barzdin. *Finite Automata*. North Holland, Amsterdam, London, 1973.