Modularization of Regular Growth Automata

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

Abstract

Regular growth automata form a class of infinite machines, in which all local computations are performed by finite state automata. We present some results which are relevant to application in practice; apart from runtime, the most important one is *modularization*, that is, abstraction over subroutines. We use the new techniques to prove some results on substitution.

1 Introduction

We recently introduced the concept of **regular growth automata** (RGA, we refer the reader to (2) for background, examples and discussion of related work; for some related work see (2),(2)). Whereas previously we focused on mathematical results and provided some formal examples, we will here focus on adapting the concept to make it useful for linguistic application. This will be accomplished via *modularization* of automata; we study how large automata can be reasonably split up into smaller and simpler ones.

Organization is as follows: in the next section, we present definitions and give an overview of the most important formal features of regular growth automata. The third section is about treating regular growth automata in a modular fashion, while preserving determinism, which is a very favorable property of RGA. In our view, this is the key to making the concept applicable on a broader scale. As we argue in the last section, this might also provide new insights in linguistic theory, as it gives us a new perspective on syntactic structures.

2 Formal Concepts

2.1 Definitions

In a sense, regular growth automata are infinite extensions of finite state automata. In FSA, states are atomic symbols like letters;[1] in regular growth automata, the state set $Q \subseteq \Omega^*$ is a *stringset* formed out of a finite alphabet. The most important property of infinite state machines in general is the computability of state transitions. We will require that transition relations on Q be *regular*.[2] The subsequent definitions follow (2).[3]

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 (synchronous)[4]* ***regular****, if there is a finite state automaton over $(\Sigma_\perp)^2$ recognizing $\otimes R$.*

[1]For this reason, they have also been called the *internal alphabet* in early work on the topic.

[2]For mathematical background see (2).

[3]We confine ourselves to binary relations, but the concept generalizes without complication.

[4]There is some confusing usage, as some authors use

Regular relations as defined here form a proper subset of the relations defined by finite state transducers in general. We use them for three reasons: firstly, they are closed under Boolean operations (contrary to transducer defined relations), secondly, they allow at most a constant growth of strings, a property whose importance will become clear later on, and thirdly, we have not met any case where the additional power provided by transducer relations would have been of any use. We call the class of transducers which computes regular relations **synchronous transducers**. In the sequel we will restrict ourselves to this subclass, so we will omit the adjective, when there is no danger of confusion.

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 $op_x \in Op_\Sigma$ with the relations they induce on Ω^*. In addition, the following hold:*

1. *F is a regular set;*
2. *for every transducer op_σ, $((q_i, \sigma), q_j) \in \delta$ exactly if $q_j \in op_\sigma(q_i)$;*
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 $\sigma \in \Sigma$, $op_\sigma(\alpha)$ is also in Q.*

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

A regular growth automaton is deterministic exactly if the letter operators represent (partial) functions on Ω^*, the RGA is total exactly if the letter operators represent total functions/relations. We can easily totalize RGAs by totalizing the letter operators, sending all previously undefined inputs to a new absorbing state.[5] We strongly conjecture that deterministic and non-deterministic automata are non-equivalent, but still lack a conclusive proof (we will see some evidence later on).[6] We will write *RGA* or regular growth automaton if we make statements valid for both cases; we will write *DGA* for deterministic, *NGA* for non-deterministic automata. We will focus on the deterministic case, which has many favourable properties, as we will see.

Note that there is some redundancy in our presentation, for the letter operators serve to specify the transition function and state set. We keep this redundancy for ease of presentation: if we fix the letter operators, Q and δ are fixed, as well as Σ and Ω. The only additional information we need is the set of accepting states F. When we construct automata, we will thus construct letter operators and an accepting state set, nothing more. We write $\hat{\delta}$ for the transition function generalized from letters to string in the obvious fashion. In case we are handling with several automata, we mark automata and their components with a subscript as in δ_{RGA} to ensure unique reference. Then we have the following:

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

We write $L(RA)$ for the language a given automaton recognizes. There are two ways to conceive of a regular growth automaton: the first one is to think of it as an automaton with an infinite state set which is already specified; alternatively we can think of it as a machine which constructs its own state set only when given an input.

Definition 5 *Two strings $\vec{x}, \vec{y}$ are* ***Nerode equivalent*** *in a language L, in symbols $\vec{x} \sim_L \vec{y}$, if for all $\vec{z}$, $\vec{x}\vec{z} \in L$ iff $\vec{y}\vec{z} \in L$.*

the term regular for "finite state computable", while others use the term in our sense. We will usually simply say regular, and mean it in the sense defined here.

[5]We call an absorbing a state which is not accepting and from which all transitions point at the state itself.

[6]The classical determinization algorithm via powerset construction does *not* preserve the regularity of the automaton.

From a language theoretic perspective note that, as an ordinary finite state machine, it computes Nerode-equivalences. From a practical perspective, the most remarkable property is that the automaton constructs its state set in runtime; assuming that its input has some upper bound, it will be nothing but a finite state machine. But it is a finite state machine which is constructed *on the fly*, that is, given an input, the necessary states and *only* the necessary states are generated.

In this paper, we will add another property to the one of being dynamic (in the above sense), namely the one of being *modular*: we will show that in regular growth automata, we can *abstract* over certain subsystems of language, factorizing them into modules. This remedies a notorious weakness of classical finite state approaches: the lack of abstraction ((2)).

2.2 Recognizing Power

We call the class of languages recognized by some DGA (NGA) $\mathfrak{L}_{DGA}$ ($\mathfrak{L}_{NGA}$). As we have shown previously, $\mathfrak{L}_{DGA}$ forms a Boolean Algebra. It contains languages which are not context-free, not mildly context sensitive and not semilinear, however there is strong evidence that it does not contain all context-free languages (see below, and contrary to $\mathfrak{L}_{NGA}$, for there is an algorithm which converts any CFG into a NGA). $\mathfrak{L}_{DGA}$ gives thus a true cut across the Chomsky hierarchy, which we judge to be possibly relevant for formal linguistics.

2.3 Runtime

Definition 6 *A finite state transducer T is* ***functional****, if for any given input T computes at most one output. It is* ***deterministic****, if $\delta_T(Q \times \Omega)$ is a (partial) function.*

As is well-known, the latter implies the former, but the former does not imply the latter; for *DGA*, we do require our transducers to be functional, but not to be deterministic.

Lemma 7 *A synchronous functional transducer T computes the output for a given input of length n in $O(n)$ steps.*

Proof. This is obvious, if the transducer is deterministic.[7] If a synchronous transducer is non-deterministic, then there is a constant upper bound to non-determinism: the states $q_i \in \hat{\delta}(q_0, \vec{w}, \vec{v})$ are less than k for all $\vec{w}, \vec{v} \in \Sigma^*$, that is, $|\hat{\delta}(q_0, \vec{w}, \vec{v})| \leq k$. This is obviously due to the fact that T has finitely many states. We show that also the set of *pairs* of states and output strings $\langle \vec{v}, q_i \rangle \in \hat{\delta}(q_0, \vec{w})$ is constantly bounded for any $\vec{w}$, that is, for all $\vec{w}$, $|\hat{\delta}(q_0, \vec{w})| \leq k$. This is because the set of states we can go to is bounded, and for a given input, if we go into one state, we need to write at most one output. For if we have two output words for a prefix by going to a single state, then we can discard both: if it were possible to reach an accepting state from the current one, then the transducer would no longer be functional, contrary to our assumptions. Therefore, for each letter we read, we have to compute at most k transitions and write an output on at most k possible output words. The output word for an input of length n is thus computed in $\leq kn + l$ steps (l is the length of the constantly bounded suffix we might add). ⊣

Theorem 8 *For any string $\vec{w}$, $|\vec{w}| = n$, a given DGA accepts or rejects $\vec{x}$ in $O(n^2)$ steps.*

Proof. As our transducers are synchronous and functional, the length of the states[8] grows (at most) proportionally to the size of the input string. Put $l := max(\{n : n = |op_\sigma(\vec{x})| - |\vec{x}| : \sigma \in \Sigma\})$. For lemma 7, the transducers compute the transitions for states of length m in $\leq km$ steps; as the DGA is deterministic, the letter operators are functional. To calculate the state $\vec{x}$, for $\vec{x} = \hat{\delta}_{DGA}(\epsilon, \vec{w})$ and $|\vec{w}| = n$, we thus have to perform at most $k \times l + k \times 2l + ... + k \times nl = \sum_{i=1}^{n} k(i \times l)$ steps. Checking whether the state is accepting takes at most $l \times n$ steps; so accepting or rejecting takes $O(\sum_{i=1}^{n} kil + ln)$ steps. This is proportional to n^2. ⊣

Note that this is strong evidence for the conjecture that $\mathfrak{L}_{DGA}$ does not contain the context free languages, for it is widely believed that the lowest possible bound for parsing CFLs is higher

[7] A proof of this can be found in (2).

[8] Recall that RGA-states are strings!

than quadratic (see (2)).

For non-deterministic automata, things are much worse: we have to assume that in the worst case runtime is *exponential* in terms of input strings: for in general, a word of length n might lead us to k^n different states, and so we might have to perform exponentially many computations. This is a very good reason to take care that a DGA remains deterministic under various transformations.

3 Automaton Construction and Regular Growth Subroutines

3.1 Automaton Construction

For application, the most urgent question is how to construct a regular growth automaton given a language. We have no way of defining directly the state set: we can only define it indirectly via the letter operators, and so we have to be aware of the consequences in the infinite of the interaction of our finite state transducers. Constructing a large set of transducers which interact in the desired way can be a tremendous task.

In this section, we propose a procedure to facilitate the construction of regular growth automata: we will factorize languages into certain *sublanguages*, and adress these by interacting *subroutines*. For example, in natural language, this means that for a given category, say, a NP*[nom]*, we construct a subroutine, which covers all its possible contents regardless of the contexts in which it occurs; and we construct a *matrix automaton* in which it is embedded and which provides its possible contexts, as main clause, complement clause etc. This is not a trivial task, given that we do *not* have an a priori notion of constituency in our approach! We thus have two problems: care for the distribution of an NP*[nom]*, and care for its internal structure. However, each of these seem to be much more feasible problems than both at once.

We will show how to construct transducers out of subroutines, such that the letter operators compute the union of all subroutines they occur in, but where it depends on the context (i.e., the current state) which subroutine is called. We will do this first for simple and then for recursive substitution. If we want to preserve determinism when merging subroutines in the general case, there is however an important condition: there must not be a locally unbounded ambiguity on which subroutine is called at a certain point.

3.2 Regular Growth Subroutines: Equivalence

Definition 9 *A* ***(deterministic) regular growth subroutine*** *is a (D)GA,*

1. *with a regular set of initial states $I \subseteq \Omega^*$ instead of ϵ, and*
2. *for all $op_\sigma \in Op_\Sigma$, all $q_i \in I$ and $\vec{w} \in \Omega^*$, $op_\sigma(q_i\vec{w}) = q_i\vec{v}$ for some $\vec{v} \in \Omega^*$; that is, operators only write and change suffixes to initial states.*

A (deterministic) regular growth subroutine (RGS/DGS) is thus a generalization of a DGA; a DGA is a DGS with the empty string as initial state. A note on the second condition: if subroutines only write suffixes to their initial states, this facilitates their global interaction when we embed them into larger automata. We use prefixes in I to encode global states, and the suffix to encode the state of the subroutine. Importantly, we do not add any additional recognizing power to our automata by generalizing them to subroutines in the above way:

Theorem 10 *For any language L, $L = L(DGS)$ for some DGS exactly if there is a DGA such that $L = L(DGA)$.*

Note that for the nondeterministic case, this result would be trivial, but it is not for the deterministic one. The idea of the proof is quite simple: we use a partition of I to simulate it on a finite tuple. As the proof itself is quite lengthy, we present it in the appendix. The concept of subroutines is very useful from the following perspective: if we merge several DGA into into a single automaton, the resulting automaton will recognize simply the union of their languages. If we merge several DGS with a DGA, this will not be necessarily the case: the initial state sets provide us with a tool to control when a DGS is *called*. We can thus convert DGA into DGS, to be able to merge them in a *controlled* fashion; the equivalence is important for it means we can conceive and check DGA and DGS independently and equivalently. Recall the the definition of $\sim_L$, the Nerode-equivalence in L:

Definition 11 *We say a subroutine S is* ***embedded*** *within a RGA, if*

1. $I_S \cap Q_{RGA} \neq \emptyset$ *and*

2. $F_S \cap Q_{RGA} \neq \emptyset$

3. *if* $\hat{\delta}_{RGA}(q_0, \vec{u}) \in I_S$, *then for all* $\vec{v}$, $\vec{w} \in L(S)$, $\vec{u}\vec{v} \sim_{L(RGA)} \vec{u}\vec{w}$,

that is, S must be reachable, it must lead back into the main automaton, and for all prefixes which call the subroutine, the stringset the subroutine computes forms an equivalence class with respect to the language of the matrix machine.

Note that with this definition, we do not say anything on the internal structure of the automata or the operators, nor of the independent existence of a subroutine in the above sense. Embedded subroutines can be *implicit*, that is, they are computed, but at no point written out as such.

Definition 12 *For a subroutine S embedded within an RGA, if $q \in I_S$, we say that q* ***calls*** *S. If there is a $q \in Q$ such that $op_a(q) \subseteq I_S$, we say that a* ***calls*** *S. A* ***characteristic prefix*** *$\vec{a}$ of a subroutine S, written as $\vec{a} \in cp(S)$, is a word for which holds: for all $q \in Q$, if $\hat{\delta}(q, \vec{a})$ is defined,*[9] *then $\hat{\delta}(q, \vec{a})$ calls S.*

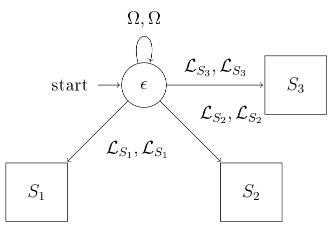
A subroutine S is called ***characteristic*** *if for all $q \in I$, we have $q = \hat{\delta}(q_j, \vec{a})$ for some $\vec{a} \in cp(S)$, where there is a constant k such that for all $\vec{a} \in cp(S)$, $|\vec{a}| \leq k$. A (sub)****language*** *L is characteristic if $L = \{\vec{x}\vec{y} : \vec{x} \in cp(S)$ and $\vec{y} \in L(S)\}$; S (and L) is* ***strictly*** *characteristic if in addition a word in $L(S)$ is not the prefix of another word in $L(S)$.*

[9]And not an absorbing state, we should add, but we leave the issue of total versus partial automata aside; it does not pose any serious problems.

An embedded subroutine is thus characteristic if we always know from a prefix when it is called; it is strictly characteristic if in addition we know when it ends.

Definition 13 *Two subroutines S_1 and S_2 (two languages L_1 and L_2) are called* ***distinct****, if they are characteristic and have disjoint sets of characteristic prefixes; they are* ***strictly distinct****, if in addition they are strictly characteristic.*

It is crucial to see that the distinction of subroutines is orthogonal to the distinction between letter operators: if subroutines embedded in a *DGA* share an alphabet, we have different subroutines within *one* letter operator. We write $op_\sigma^{S_i}$ for the function of a subroutine S_i computed by *one* operator. $op_\sigma^{S_i}(\vec{x})$ thus means that op_σ computes its function in S_i on a substring $\vec{x}$.


Ω, Ω
start
ϵ
$\mathcal{L}_{S_3}, \mathcal{L}_{S_3}$
S_3
$\mathcal{L}_{S_2}, \mathcal{L}_{S_2}$
$\mathcal{L}_{S_1}, \mathcal{L}_{S_1}$
S_1
S_2
Fig.1: Blueprint of a letter operator computing three subroutines; it computes the identity, until some marker in the input activates a subroutine. Note that these markers are *not* characteristic prefixes; rather, the characteristic prefixes put the marker into the state-string.

3.3 Example I: Simple Substitution

As an example of how to model constituent-like dependencies, we show how to use subroutines for simple letter substitution. Note that inserting subroutines is quite trivial; the problem is to use subroutines *and* preserve determinism of the automaton. The treatment will not be very detailed; in particular, we will not look "inside" the transducers and spell out the operations. This is due to the fact that firstly, we do not really *need* to - which is one of the greatest merits of our approach; and secondly, because transducers quickly result to be large and hard to read. We encourage the skeptical reader to check that all conditions we state below can indeed be checked by synchronous finite state transducers. In the sequel, we will make an abuse of ontology for the sake of readibility and construct regular expressions over languages. Recall that distinct languages have decompositions into a set of bounded prefixes, which call a unique subroutine, and the language of the subroutine itself.

Proposition 14 *Let Σ be an alphabet, and let $\{L_\sigma : \sigma \in \Sigma\}$ be a set of pairwise distinct languages, such that $L_\sigma = cp(DGS_\sigma) \cdot L(DGS_\sigma)$ for all $\sigma \in \Sigma$. Now let $i : \Sigma \to \{L_\sigma : \sigma \in \Sigma\}$ be a one-to-one mapping, where $i(\sigma) = L_\sigma$. For every $L_M \subseteq \Sigma^*$, if there is a DGA_M such that $L(DGA_M) = L_M$, then there is a $DGA_{i(M)}$ such that $L(DGA_{i(M)}) = i[L_M]$.*

Proof. First we change the state alphabet Ω_M of DGA_M to make it disjoint from all other Ω_x, $x \neq M$. The new operators in $Op_\Sigma^{DGA_{i(M)}}$ are constructed as follows: first we take care of the characteristic prefixes; for simplicity we assume they are single letters. If they are not, they are constantly bounded, and so we have some finite amount of non-determinism; this can always be determinized with a tuple construction.[10] To make the treatment more compact, we leave this construction to the reader.

The characteristic prefixes are the letters which compute the interaction of the routines; we add a letter $\mathcal{S}_\sigma$ to $\Omega_{i(M)}$ for each $\sigma \in \Sigma$, where for all $\alpha \neq i(M)$, $\mathcal{S}_\sigma \notin \Omega_\alpha$, and define the operators as follows:

1. For all $a \in cp(S_\sigma) : \sigma \in \Sigma$, $op_a^{i(M)}$ is defined on all and only on states in $(\epsilon|(\Omega_M^*((F_\sigma) : \sigma \in \Sigma)^*\mathcal{S}_x((F_\sigma) : \sigma \in \Sigma))$,
2. If $a \in cp(S_\sigma)$, then $op_a^{i(M)}(\epsilon) = (op_\sigma^M(\epsilon))\mathcal{S}_\sigma(op_a^{S_\sigma}(\epsilon))$,
3. If $a \in cp(S_\sigma)$, then $op_a^{i(M)}(\vec{c}\vec{x}\mathcal{S}_\tau\vec{z}) = op_\sigma^M(\vec{c})\vec{x}\vec{z}\mathcal{S}_\sigma op_a^{S_\sigma}(\epsilon)$, provided it is defined, and where $\vec{c}$ is the longest prefix of the state in Ω_M.

That is, operators for characteristic prefixes (i) only operate on states which consist of a prefix $p_M \in \Omega_M^*$, followed by a (possibly empty) sequence of accepting states F_σ of the subroutines. (ii) They operate on p_M as the letters to for which they are substituted, (iii) they put or change and postpone a distinguished marker $\mathcal{S}_\sigma$ at the end of the string, which marks the beginning of the operating scope of a substitution language, and (iv) then start computing the subroutine.

All other letter operators simply compute their previous function as subroutines, where each subroutine S_σ computes L_σ, and where the initial state set for S_σ is $\Omega^*\mathcal{S}_\sigma$. Before they read the marker, they simply compute the identity, so:

(1) $$op_a^{i(M)}(\vec{x}\mathcal{S}_\sigma\vec{y}) = \vec{x}\mathcal{S}_\sigma op_a^{S_\sigma}(\vec{y}),$$

Note that our characteristic prefixes make sure there is only one marker $\mathcal{S}_\sigma$ in any state-string. The set of final states $F_{i(L)}$ is now simply the set of all sequences of only accepting states for all languages/subroutines, with possibly a marker $\mathcal{S}_\tau$:

(2) $$F_{i(L)} := F_M \cdot ((F_\tau) : \tau \in T)^*\mathcal{S}_x((F_\tau) : \tau \in T).$$ ⊣

Note that in this case, we need the condition of distinctness of sublanguages, to make sure we know when to call a subroutine. When we treat recursive substitution, we will see that we need strict distinctness, to also know when we can stop a subroutine.

3.4 Example II: Recursive Substitution

We are surely not only interested in simple substitution, but also *recursive* substitution, that is, we want to be able to call new subroutines during the execution of subroutines. As an example, we will prove the following:

Proposition 15 *Let $P \subseteq \Sigma$; let $(L_\rho) : \rho \in P$ be a set of strictly distinct languages, such that $L_\rho = cp(DGS_\rho) \cdot L(DGS_\rho)$ and $L_\rho \subseteq \Sigma^*$ for all $\rho \in P$. Define $i : P \to (L_\rho) : \rho \in P$ as a one-to-one mapping, where $i(\rho) = L_\rho$. Let $L_M \subseteq \Sigma^*$ be a language such that $L_M = L(DGA_M)$. Then there is a DGA_P which recognizes L_P, which is defined as follows:*

1. If $\vec{x} \in L_M$, then $\vec{x} \in L_P$.

2. If $\vec{x}\rho\vec{y} \in L_P$, then $\vec{x}i(\rho)\vec{y} \in L_P$.

[10] Alternatively, we can conceive of characteristic prefix strings as *chunks*: as regular relations are closed under composition, we can only let the composition compute the desired function; technically, this amounts to the same, namely a tuple construction.

Proof. We construct DGA_P. First we change the state alphabet Ω_M such that is disjunct from all other Ω_x, $x \neq M$. The new operators $Op_\Sigma^{DGA_P}$ are constructed as follows: first we take care of the characteristic prefixes; for simplicity we assume again they are single letters, and use letters $\mathcal{S}_\rho$ as before.[11]

1. If $a \in cp(S_\rho)$, then $op_a^P(\vec{x}) = op_\rho^M(\vec{x})\mathcal{S}_\rho op_a^{S_\rho}(\epsilon)$, if $\vec{x} \in \Omega_M^*$.
2. If $a \in cp(S_\rho)$, then $op_a^P(\vec{x}\mathcal{S}_\sigma\vec{y}\vec{z}) = \vec{x}\mathcal{S}_\sigma(op_\rho^{S_\sigma}(\vec{y}))\vec{z}\mathcal{S}_\rho(op_a^{S_\rho}(\epsilon))$,
   where $\vec{z} \in (\epsilon|(S_\rho : \rho \in P)\Omega_P^*)$, and $\vec{y}$ is the *rightmost* substring matching this conditions which does not contain any symbol $\mathcal{S}_\rho$ and is *not* in any $F_{\rho'}$.

Though notation becomes a bit opaque at this point, the concept is quite simple: we simply make sure that in our DGA_P states every subroutine has a clearly distinguished operating scope, where the most deeply embedded subroutine is written as the rightmost one in the state string. We thus translate a structure of the form $[_1[_2[_3]]]$ into a form $[_1][_2][_3]$. This is works because sublanguages are strictly distinct, so we know exactly when a subroutine is completed, and we can ignore all substrings in some F_ρ. The other letters operate as follows:

1. $op_\tau^P(\vec{x}\mathcal{S}_\sigma\vec{y}\vec{z}) = \vec{x}\mathcal{S}_\sigma(op_\tau^{S_\sigma}(\vec{y}))\vec{z}$, where $\vec{z} \in (\epsilon|(S_\rho : \rho \in P)\Omega_P^*)$, and $\vec{y}$ is the rightmost substring which satisfies this conditions which does not contain a $\mathcal{S}_\rho$ and is not in $F_{\rho'}$;
2. $op_\tau^P(\vec{x}\vec{y}) = (op_\tau^M(\vec{x}))\vec{y})$, where $\vec{x} \in \Omega_M^*$ and $\vec{y} \in ((\mathcal{S}_\rho F_\rho) : \rho \in P)^*$

The set of accepting states is

$$F_P := F_M \cdot ((\mathcal{S}_\rho F_\rho) : \rho \in P)^*. \qquad (3) \qquad \dashv$$

Note that, while we do not require that any of the languages involved be context-free, this substitution *is* in a precise sense "context-free",[12] as the distribution of the substituted language equals the distribution of a single letter. However, this is of course only one particular case of substitution; recall that the sets of initial states and final states of a subroutine are regular sets; so we can take any substitution point which can be defined by a regular set of states. We see that we can *decouple* the complexity of the substitution/upcalling of subroutines from the complexity of the routines itself. We thus can reasonably encode a difference between local complexity (in the sense of subroutine) and a global complexity (in the sense of substitution conditions), and we can restrict them independently.

4 Outlook: Regular Growth Automata and Linguistic Theory

In the last section we saw how the factorization was useful to find a (quite) simple solution to complex problems of substitution. The main goal was to show that factorization of automata into subautomata makes them easier to handle in application. We want to underline that the reason we think modular treatment is much easier for linguistic application is not only the structure of regular growth automata, but in particular the structure of natural languages. This is partly due to the fact that we often[13] find some kind of constituent structure in natural languages; but there is more to that: we often find very complex local structures, whose scope however is often quite limited. If we look for example at systems of clitic pronouns, we find a very unpredictable behaviour, and a large number of complex rules which possibly even have to take into account segmental and suprasegmental phonology. If we look at constituents which have a (possibly) very complex internal structure, as clausal complements, we find that their distribution is highly regular and follows very simple rules.[14]

This observation is by no means new: for example, the well-known work of John Hawkins is mostly concerned with these and similar ob-

[11] There is a $\mathcal{S}_\rho$ for each $\rho \in P$ and $\mathcal{S}_\rho \notin \Omega_\alpha$, for all $\alpha \neq P$.

[12] More p precisely: deterministic context-free.

[13] Some people say: always.

[14] This asymmetry gets even more striking if we also consider morphology as a part of syntactic structure.

servations. We will not go into detail here, but we think that principles as *early immediate constituents*, *minimize domains* etc. (see (2),(2)) can be roughly restated, with a slightly shifted perspective, in the following way:

(4) The more complex a constituent is, the more regular[15] is its distribution.

We think that this important observation has had unduly little impact on linguistic theory proper. The main reason for this seems to us that usually we posit fully specified constituent structures (of some kind or other) for all utterances: and once we make this step, we are unable to partially redeem it. One could see it as a disadvantage to regular growth automata that they cannot be mapped onto such a constituent structure, contrary to, for example, pushdown automata. However, in this context there is an equal advantage: for a pushdown automaton, the notion of a subroutine does not make much sense, for all configurations are equally well-suited and, in fact, can be easily treated as subroutines. For our automata this is not the case: *because* the notion of a constituent is so problematic, the notion of an embedded subroutine is *truly meaningful.* It allows to capture the asymmetry of local and global structure. This is the reason we think our approach is well suited for natural language descriptions not only from a practical, but also a theoretical point of view: though we give up the notion of a fully specified constituent structure, we can much more easily treat the rich local structure of natural languages with compact local subroutines, and its much more austere global structure by means of very simple interactions of routines. This in turn might facilitate a more realistic view on the organization of linguistic knowledge.

Acknowledgements

The author wants to thank Marcus Kracht and Jens Michaelis for their support, and three anonymous reviewers for their helpful comments.

[15]Here we can read *regular* both in its colloquial and in its technical meaning.

5 Appendix: Proof of Theorem 10

Proof. One direction is immediate. To see the if-direction, we show how to convert a DGS into a DGA. By definition, for the prefixes of states which are in I, the output equals the input, and as we have one output and the transducer is functional, there is a single state we go to (this is the inverse reasoning from lemma 7).[16] Assume without loss of generality that our letter operators are total. We write

(5) $\vec{x} \sim_{op_\sigma} \vec{y}$ if $\hat{\delta}_{op_\sigma}(q_0,\vec{x}) = (\vec{x}', q_i)$ and $\hat{\delta}_{op_\sigma}(q_0,\vec{y}) = (\vec{y}', q_i)$ for some $\vec{x}', \vec{y}'$.

For the above reasons, $\sim_{op_\sigma}$ is an equivalence relation. We form partitions I with the equivalence classes induced by $\sim_{op_\sigma}$, for each $op_\sigma \in Op_\Sigma$: put $P_{op_\sigma} := I/\sim_{op_\sigma}$. As our operators are finite, $|I/\sim_{op_\sigma}| \leq k$ for each $\sigma \in \Sigma$. Next, by intersecting the elements across the different partitions, we construct a more fine-grained partition, which forms a partition (though not maximal) with respect to every $\sim_{op_\sigma}$: $op_\sigma \in Op_\Sigma$:

(6) $\mathcal{J} := \bigcap_{op_\sigma \in Op_\Sigma} P_{op_\sigma}$.[17]

$\mathcal{J}$ is a partition of I, and for each $J \in \mathcal{J}$ and for each $op_\sigma \in Op_\Sigma$ there is a unique state q such that $\hat{\delta}_{op_\sigma}(q_0,\vec{j}) = (\vec{j}', q)$ for some $\vec{j}'$, and all $\vec{j} \in J$. Crucially, while its elements might be of infinite cardinalitiy, $\mathcal{J}$ itself is of finite cardinality. Put $l = |\mathcal{J}|$. The initial state of our DGA is an l-tupel, where each $\pi_i : i \leq l$ is assigned a $J \in \mathcal{J}$ via a bijection $\phi(\mathcal{J}) \to M$,where $M \subset \mathbb{N}$ and for all $m \in M$, $m \leq l$. We construct the new letter operators op_σ^{DGA}, which operate on $(\Omega^k)^*$, as follows:

(7) $op_\sigma^{DGA}(\langle \vec{x}_1, \vec{x}_2, ..., \vec{x}_l \rangle = \langle op_\sigma^{q_i}(\vec{x}_1), op_\sigma^{q_j}(\vec{x}_2), ..., op_\sigma^{q_m}(\vec{x}_k) \rangle$,

where $op_\sigma^{q_x}$ is op_σ with its initial state changed to q_x; and for every $\pi_i : i \leq l$, we make sure that if $\phi^{-1}(i) = J$, then for all $\vec{j} \in J$, $\hat{\delta}_{op_\sigma}(q_0,\vec{i}) = \langle \vec{w}, q_x \rangle$ for some $\vec{w}$.

We thus simulate the possibly infinite set I with a finite set of tuples, where each projection represents a set of strings for which all op_σ go into a unique state. The operations on projections are still synchronous regular, and as the DGS was deterministic, the tuple size of the DGA remains constant.

[16]Provided the transducer is not redundant; if it is, then the states can be merged.

[17]Note that this is a somewhat sloppy notation, as we do not intersect the partitions, but their elements.

Now to F_{DGA}. Regular languages are closed under projection and cylindrification; so we take as accepting states the set of all l-cylindrifications of F_{DGS}, that is, the set of all l-tuples of which one projection is in F_{DGS}. Call this set $F_{DGS'}$. Finally, we need to take care of the case where a prefix in I has some influence on membership in F. Here we use the fact that regular languages are closed under prefixation: for two languages, L_1, L_2, we put $L_1 \backslash L_2 := \{\vec{v} : \exists \vec{w} \in L_1 : \vec{w}\vec{v} \in L_2\}$. It is well-known that if L_1 and L_2 are regular, then so is $L_1 \backslash L_2$, the set of strings which, given a prefix from L_1, result in a word in L_2. We thus have to put $F_{DGA} := I \backslash F_{DGA'}$. The resulting DGA does the job as required. $\dashv$

References

Didier Caucal. Synchronization of regular automata. In Rastislav Královic and Damian Niwinski, editors, *MFCS*, volume 5734 of *Lecture Notes in Computer Science*, pages 2–23. Springer, 2009.

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

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

John A. Hawkins. *Efficiency and Complexity in Grammars.* Oxford Univ. Pr., Oxford, 2004.

Lillian Lee. Fast context-free grammar parsing requires fast boolean matrix multiplication. *J. ACM*, 49(1):1–15, 2002.

Mehryar Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311, 1997.

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

Shuly Wintner. Strengths and weaknesses of finite-state technology: a case study in morphological grammar development. *Natural Language Engineering*, 14(4):457–469, 2008.

Christian Wurm. Regular growth automata: Properties of a class of finitely induced infinite machines. In *Proceedings of the 12th Mathematics of Language, Springer LNCS*, 2011.