Rend. Istit. Mat. Univ. Trieste
Volume 43 (2011), 95–110

Language Sampling for Universal Grammars

Luca Bortolussi and Andrea Sgarro

Abstract. *In this paper we present a sampling algorithm for constrained strings representing the state of parameters of a universal grammar. This sampling algorithm has been used to assess statistical significance of the parametric comparison method, a new syntax-based approach to reconstruct linguistic phylogeny.*

Keywords: Monte-Carlo Sampling, Universal Grammars, Language Phylogeny.
MS Classification 2010: 65C60, 68U99

1. Introduction

Historical linguistics is a discipline studying the evolution of languages in the past, with the ultimate aim of gaining a better understanding of past events of human history. A major technique is the construction of phylogenies of current (and extinct) languages. The main methodologies for this task are based on the so called *classical comparative method* [8], exploiting lexical and phonetic relationships to prove language relatedness and to construct language distances (a step required for algorithmic phylogenetic tree reconstruction [4]). Unfortunately, the evolutionary speed of these kinds of data makes them useless to recover relationships more distant in time than 10,000 years.

A recent approach to circumvent these limitations is the *Parametric Comparison Method* (PCM) [5, 6], using syntactic digital data obtained from parameters of the *Universal Grammar* (UG) [3]. Universal Grammar is a recent theory trying to explain language diversity and acquisition in terms of a *finite* number of (binary) switches encoded in the brain that are fixed during language learning and precisely define the syntax of a language. As parameters are supposed to evolve at a much slower rate than lexicon, PCM compares (some) parametric values of languages and extracts phylogenetic information from them. The digital nature of such data parallels the use of DNA information to reconstruct phylogeny of biological species. Unlike with DNA, however, parameters of the UG are interrelated by a complex network of logical dependencies, a fact making statistical analysis of data much more complicated.

In this paper we tackle the problem of sampling uniformly from the set of strings corresponding to admissible values of parameters of a UG, a necessary step in order to perform statistical analysis.

In our setting, there is a fixed number n of boolean parameters of the universal grammar, which are related among them according to logical dependencies. This means that not all possible assignments of boolean values to parameters are *admissible*. Furthermore, certain parameters may not have a defined value under certain circumstances (i.e. depending on the state of related parameters), hence they may have three values: +, -, and 0 (undefined).

The relationships between parameters are given in terms of propositional formulae, where atoms predicate the value of a single parameter. These formulae can be essentially seen as constraints on the space $\mathcal{A}^n$ of strings of length n on the ternary alphabet $\mathcal{A} = \{+, -, 0\}$, so that the space of admissible languages $\mathcal{L}$ is a proper subset of $\mathcal{A}^n$. Hence, the problem we are facing is sampling uniformly from $\mathcal{L}$.

One simple solution is to use a rejection sampling technique: we generate an element of $\mathcal{A}^n$ uniformly (which can be done easily by sampling each position in the string independently) and then we check whether or not it satisfies the constraints. Unfortunately, this approach is unfeasible, because $\mathcal{L}$ is very small with respect to $\mathcal{A}^n$, hence this method is much too costly. The solution we propose here is to use a modification of this basic rejection sampling technique: we sample from a subspace $\mathcal{L}_0$ such that $\mathcal{L} \subset \mathcal{L}_0 \subset A^n$, and such that the relative dimension of $\mathcal{L}$ in $\mathcal{L}_0$ is reasonably large. The main difficulty is to find such a space $\mathcal{L}_0$ together with an efficient algorithm to sample uniformly from it.

Our main strategy is to investigate the structure of the constraints on the languages. If these constraints are simple enough, we can build a data structure enabling fast sampling. If all constraints were of this kind, we would have an efficient sampler. However, just a small fraction of constraints is simple enough. In order to make the method feasible, we preprocess the space of parameters by merging some of them in order to simplify the structure of the constraints. This approach is fit to deal with the number of parameters typical of the linguistical applications we have to face.

In the rest of the paper, we first present formally the data we have to work with, namely parameters and rules (Section 2). Subsequently, we recall the basics of rejection sampling (Section 3), and then we present our fast sampling based on a data structure managing simple constraints, called *sampling structure* (Section 4). This part is the core of the paper: we define formally the data structure used for the sampling, proving the correctness of our algorithm. In Section 5, we deal with the problem of merging parameters together so as to simplify the structure of rules. Finally, in Section 6, we discuss the performances of the algorithm on real data and draw some conclusions.

2. Parameters and Rules

As customary in science, we work with a model of a certain aspect of reality. Our interest is in the historical evolution of languages, and we focus on a particular description of languages based on the universal grammar [3, 6]. Essentially, each language is characterized by a set of boolean parameters, which fix a certain feature of the syntax of the language, mimicking the way language grammar is learnt and "stored" by human brain. Abstracting from the precise meaning of parameters, each language is represented by a tuple of values for the parameters considered.

However, parameters are not independent of one another, but rather they are connected by an intricate network of logical relationships. More precisely, it is meaningful to consider a parameter only if certain other parameters have specific values. For instance, a parameter P_2 may be considered only if the parameter P_1 has value + (i.e. the characteristic it encodes is present). These relationships build up a causal structure on parameters, which is free of logical vicious circles, meaning that parameters can be ordered so that the meaningfulness of parameter P_j depends only on parameters P_i, for $i < j$. Rules for parameters are usually expressed by propositional formulae, specifying which values of parameters are required in order for parameter P_j to be meaningful. For instance, the formula for parameter P_3 may be $\phi_3 = (x_1 = +) \wedge (x_2 = -)$.

We introduce now some notational conventions that will be used throughout the paper. We assume we have n parameters, indicated by $P_1, \ldots, P_n$. Each parameter P_i can take values in a finite domain $\mathcal{D}_i$. Usually, $\mathcal{D}_i = \{+, -\}$, so that P_i is binary. However, for reasons that will be made clear in Section 5, we prefer to work in a more general setting, without restricting the cardinality of $\mathcal{D}_i$ (essentially, we want to deal with macro-parameters, constructed by merging together several "basic" parameters, thus having more than 2 possible states). In addition, each parameter can take a special value, 0, which indicates that it is not meaningful in the current context (i.e., given the value of the parameters it depends on).

Hence, the proper domain for parameter P_i is $\tilde{\mathcal{D}}_i = \mathcal{D}_i \cup \{0\}$, and the space of tuples we need to consider is $\mathcal{D} = \prod_{i=1}^{n} \tilde{\mathcal{D}}_i$.

Only the elements of $\mathcal{D}$ that satisfy a given set of constraints, expressed in terms of rules, are *admissible*. A *rule* is a pair (P_j, ϕ_j), where P_j is a parameter and ϕ_j is a propositional formula. The atoms are equalities of the form $x_i = a_i$, where x_i is a variable referring to the parameter P_i and $a_i \in \mathcal{D}_i$ is a possible value of P_i. We require that all the variables appearing in the formula ϕ_j are among $x_1, \ldots, x_{j-1}$. We consider a set of rules $\mathcal{R}$, containing one rule for each parameter P_j (note that a formula ϕ can be a tautology). We say that a tuple $\mathbf{a} \in \mathcal{D}$ *satisfies* a rule (P_j, ϕ_j) in $\mathcal{R}$ if and only if the formula $\phi_j[\mathbf{a}]$ is true. A tuple $\mathbf{a} \in \mathcal{D}$ is *admissible* if and only if it satisfies all the rules of the set $\mathcal{R}$.

3. Rejection Sampling

Rejection sampling is a standard sampling technique [2, 7] to sample indirectly from a target probability distribution. More precisely, suppose we want to sample from a distribution $q_1(x)$ on space X, but we do not know how to sample from q_1. However, we have a sampling algorithm for another distribution $q_2(x)$ on X, and we know that there exists a constant $M > 0$ such that, for each $x \in X$, $q_1(x) \leq Mq_2(x)$. Rejection sampling consists in sampling from q_2 and accepting a sample with probability $\frac{q_1(x)}{Mq_2(x)}$. More precisely, the sampling algorithm works as follows:

1. sample x from q_2;
2. sample u from the uniform distribution in $[0, 1]$;
3. accept x if $u < \frac{q_1(x)}{Mq_2(x)}$.

In our context, rejection sampling in even simpler. Suppose we want to sample uniformly from a subset $\mathcal{L} \subseteq \mathcal{D}$, but we only have an uniform sampler for an intermediate space $\mathcal{L}_0$, such that $\mathcal{L} \subseteq \mathcal{L}_0 \subseteq \mathcal{D}$. In this setting, an element $x \in \mathcal{L}_0$ is sampled with probability $\frac{1}{|\mathcal{L}_0|}$, and the constant M is $M = \frac{|\mathcal{L}_0|}{|\mathcal{L}|}$. This implies that the acceptance rule simplifies to: *accept x if and only if it belongs to $\mathcal{L}$.*

The problem with rejection sampling lies in the constant M. In fact, M can be seen as the average number of trials one has to do in order to generate an element of $\mathcal{L}$. Hence, if M is very large, the rejection sampling becomes highly inefficient.

In our application context, if we choose $\mathcal{L}_0 = \mathcal{D} = \mathcal{A}^n$, from which uniform sampling is very easy to implement (just select a value for each parameter uniformly and independently), we obtain a constant which is incredibly large (of the order of 10^{18} in the case of [6], see also Section 6). Hence, the only way of using rejection sampling in our context is to identify a much smaller super-space of $\mathcal{L}$. We tackled the problem in the following way:

- we identified a simple form for rules that allows fast uniform sampling, using a dedicated data structure;
- we separated actual rules into two subsets, those that allow fast sampling and the others;
- we sample parameters governed by simple rules using the dedicated sampling and we sample the other parameters uniformly, then accept if all rules are satisfied (i.e. if the language so generated is in $\mathcal{L}$).

In this way, we are sampling from an intermediate space $\mathcal{L}_0$, as we never generate sequences of parameters that violate a simple rule.

Unfortunately, also this approach is limited, because the set of bad rules is too large (resulting in a large constant M). In order to reduce its size (hence the constant M), in Section 5 we merge some parameters into macro-parameters. In this way, we remove some rules (as they are taken into account in defining the allowed values of the macro-parameter) and we simplify other rules, making them simple enough to be dealt with by means of the fast sampler.

4. Sampling Structures

Sampling directly admissible languages according to a uniform probability distribution is complicated because of the complexity of the rules. However, a direct sampling algorithm may be feasible if we impose sufficient restrictions on the rules. In particular, we will consider restrictions allowing the construction of a collection of decision trees to uniformly sample the value of parameters in *linear* time with respect to n.

In order to construct our decision trees, the meaningfulness of a parameter P_j should depend only on another *single* parameter P_i, $i < j$. This surely holds if the formula for P_j contains a single atom, namely it is of the form $x_i = a_i$ (*simple rules*). This also holds for formulae that are disjunctions of atoms, such that only one atom can be true at a given time (*exclusive OR rules*), due to constraints imposed by rules on the parameters involved in the disjunction. In this latter case, we just need to consider one parameter, the one whose atom is true. Hence, in the following we assume that the set of rules $\mathcal{R}$ contains only tautologies, simple rules and exclusive OR rules;[1] in subsection 4.2 we show how to check whether an OR rule is indeed exclusive. A set $\mathcal{R}$ of this kind will be dubbed *simple*. Under this restriction, we will show how to build a set of trees (a forest) and how to use it for sampling uniformly. We will refer to this forest as the *sampling structure* associated with $\mathcal{R}$.

4.1. Definition of the Sampling Structure

The sampling structure associated with a simple set of rules $\mathcal{R}$ contains two kinds of nodes: *parameter nodes*, or *p-nodes*, and *value nodes*, or *v-nodes*. The former represent parameters, while the latter encode the different values that a parameter can take. These nodes will be annotated: p-nodes store the index of the parameter they are associated with, while v-nodes store both the index of the parameter and the value of the element they refer to. Edges in this graph represent two different things: edges from p-nodes to v-nodes simply connect each p-node with all the possible values its parameter can take, while edges from

[1] Note that simple rules are special cases of exclusive OR rules, with just 1 disjunct.

v-nodes to p-nodes encode the dependencies between parameters through rules. Essentially, there will be an edge from a v-node for parameter P_i and value a_i to a p-node for parameter P_j if and only if $x_i = a_i$ is a disjunct in the rule for P_j. Furthermore, we will require that this forest contains the information of all the rules in $\mathcal{R}$. We now provide a formal definition of the sampling structure (a forest, actually) associated with a given simple set of rules $\mathcal{R}$.

DEFINITION 4.1. *A* sampling structure *for a simple set of rules $\mathcal{R}$ is a tuple $T = (P, V, E, \iota, \lambda)$, where:*

- *P is the set of p-nodes;*
- *V is the set of v-nodes;*
- *$E \subset (P \times V) \cup (V \times P)$ is the set of edges;*
- *$\iota : P \cup V \to \{1, \ldots, n\}$ associates with each node the index of the parameter it refers to;*
- *$\lambda : V \to \bigcup_{i=1}^{n} \mathcal{D}_i$ associates with each v-node v a value $\lambda(v) \in \mathcal{D}_{\iota(v)}$.*

T satisfies the following properties:

1) *for all $i = 1, \ldots, n$, there exists $p \in P$ such that $\iota(p) = i$ (completeness of p-nodes);*
2) *$(p, v) \in E$ implies that $\iota(p) = \iota(v)$ (coherence between a p-node and its children v-node);*
3) *for each $v \in V$ there exists $p \in P$ such that $(p, v) \in E$ (each v-node has a p-parent);*
4) *for each $p \in P$, there exist $v_1, \ldots, v_h \in V$, $h = |\mathcal{D}_{\iota(p)}|$, such that $(p, v_i) \in E$ and $\lambda(v_i) \neq \lambda(v_j)$, for each $i \neq j$ (each p-node has children v-nodes for all possible values);*
5) *if $(P_j, (x_{i_1} = a_{i_1}) \vee \ldots \vee (x_{i_k} = a_{i_k})) \in \mathcal{R}$, then for each $v \in V$ such that $\iota(v) = i_s$ and $\lambda(v) = a_{i_s}$, there exists $p \in P$ with $\iota(p) = j$ and $(v, p) \in E$ (disjuncts imply edges);*
6) *for each $(v, p) \in E$, $x_{\iota(p)} = \lambda(v_{\iota(v)})$ is a disjunct in the rule for $P_{\iota(p)}$ (edges imply disjuncts).*

In graph-theoretic terms, the previous definition corresponds to a forest, whose roots correspond to independent parameters, i.e. those having a tautology assigned by the rules in $\mathcal{R}$. Each parameter can be associated with one or more p-nodes. Exclusive OR rules with more than one disjunct multiply the

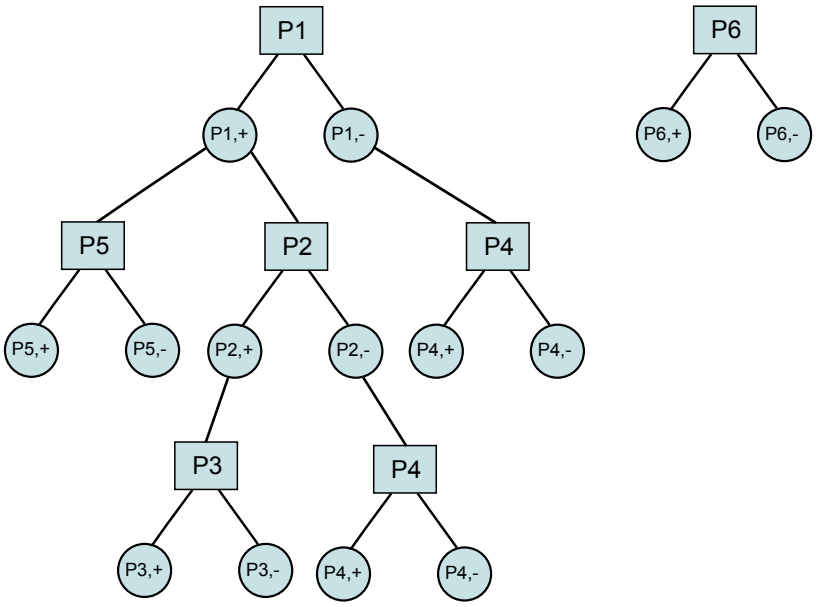

Figure 1: An example of a sampling structure for parameters $P_1, \ldots, P_6$, all taking values in $\{+, -\}$, and subject to rules $\mathcal{R} = \{(P_1, true), (P_2, x_1 = +), (P_3, x_2 = +), (P_4, (x_1 = -) \vee (x_2 = -)), (P_5, x_1 = +), (P_6, true)\}$.

occurrences of p-nodes for a given parameter: a chain of exclusive OR rules can provoke an exponential growth of p-nodes (with respect to n). However, this combinatorial explosion can be readily tamed, see Section 4.5 below.

As an example, consider parameters $P_1, \ldots, P_6$, all taking values in $\{+, -\}$, and subject to the following set of rules $\mathcal{R} = \{(P_1, true), (P_2, x_1 = +), (P_3, x_2 = +), (P_4, (x_1 = -) \vee (x_2 = -)), (P_5, x_1 = +), (P_6, true)\}$. Observe that $\mathcal{R}$ is simple because the rule for P_4 is a disjunctive OR rule: P_2 is defined if and only if P_1 is set to $+$. The sampling structure associated with this set of rules and parameters is shown in Figure 1.

4.2. Exclusive OR Rules and Sampling Tree Structure

Before entering into the details of using a sampling structure to sample admissible languages, we describe some structural properties that can be used to check if the OR rules in the set of rules $\mathcal{R}$ are indeed exclusive. We collect these properties in the following proposition.

PROPOSITION 4.2. *Let $T = (P, V, E, \iota, \lambda)$ be a sampling structure for the set of simple rules $\mathcal{R}$.*

1) *For each independent parameter P_j of $\mathcal{R}$, there is a single p-node $p \in P$ such that $\iota(p) = j$, and it is the root of a tree in the forest T.*

2) *All p-nodes for the same parameter P_j belong to the same tree of the forest T.*

3) *Let p_1, p_2 be two nodes for the same parameter, i.e. $\iota(p_1) = \iota(p_2)$, and let w be the lowest common ancestor of p_1 and p_2, $w = LCA(p_1, p_2)$. Then $w \in P$ (i.e. w is a p-node).*

Proof.

1) This is straightforward, otherwise the property 6 of Definition 4.1 would be violated.

2) Suppose not, and let P_j be the first parameter violating the property, so that there are two nodes p_1 and p_2 for P_j belonging to different trees in the forest. The ancestor p-nodes of p_1 and p_2 are all different (due to the choice of P_j), hence the parent p-nodes and v-nodes of p_1 and p_2 correspond to two different disjuncts in the rule for P_j. Disjointness of ancestor p-nodes implies that there is an admissible tuple that satisfies both disjuncts, constructed by assigning to the parameter of each p-node p the value of its children v-nodes v in the path to p_1 or p_2, i.e. by assigning to $P_{\iota(p)}$ the value $\lambda(v)$. This contradicts the hypothesis that $\mathcal{R}$ is simple.

3) Suppose not, and let P_j be the first parameter violating the property, so that there are two nodes p_1 and p_2 for P_j whose LCA is a v-node v. Due to the choice of P_j, the p-nodes in the paths from v to p_1 and from v to p_2 correspond to disjoint parameters, hence the parent p-nodes and v-nodes of p_1 and p_2 correspond to two different disjuncts of the OR rule for P_j. Reasoning as in the previous point, there is an admissible tuple that satisfies both disjuncts, a contradiction. □

The last proposition gives a way to check if a set of rules containing just OR rules violates the property of being simple (namely, if all OR rules are exclusive). In fact, one just has to construct the sampling structure of $\mathcal{R}$ and check if points 2 and 3 of the previous proposition are violated or not. If they are violated, one can conclude that some OR rules are not exclusive. Actually, one can also prove the inverse of points 2 and 3: if the sampling structure is such that all p-nodes for the same parameter are in the same tree and their

LCA is always a p-node, then the rule set $\mathcal{R}$ is simple.[2] This provides a characterization of simple rule sets in terms of their sampling structures.

4.3. Instances and Languages

The key notion to use a sampling structure T to sample admissible languages is that of an *instance* of T. Intuitively, an instance is a sub-forest of T that can be mapped to a single admissible language. It is constructed starting from the roots and recursively picking only one child for each p-node, and all children of v-nodes. Choosing a child of a p-node corresponds to fixing the value of a parameter. Choosing all children of a v-node is necessary because we need to fix all meaningful parameters, i.e. those whose formulae in $\mathcal{R}$ are true.

We first formally define an instance of T, and then prove that instances of T and admissible languages are in bijection.

DEFINITION 4.3. *Let $T = (P, V, E, \iota, \lambda)$ be a sampling structure for the set of simple rules $\mathcal{R}$. An* instance *I of T is a subgraph $(P_I, V_I, E_I, \iota, \lambda)$ of T such that:*

1) *roots of T are in I;*
2) *if $p \in P_I$ then there exists an unique $v \in V$ such that $v \in V_I$ and $(p, v) \in E_I$ (an instance contains just one child for each p-node);*
3) *if $v \in V_I$, then for each $p \in P$ such that $(v, p) \in E$, it holds that $p \in P_I$ and $(v, p) \in E_I$ (an instance contains all children of v-nodes).*

In order to prove that instances and admissible languages are in one to one correspondence, we will show how we can construct an admissible language from an instance and, viceversa, how to construct an instance from an admissible language.

First, consider an instance I, and define a language $\mathbf{a}^I$ according to the following rule:

$$a_j^I = \begin{cases} \lambda(v), & if\ v \in V_I \wedge \exists p \in P_I : \iota(p) = j \wedge (p, v) \in E_I \\ 0, & \text{otherwise.} \end{cases}$$

In order for $\mathbf{a}^I$ to be well defined, the number of p-nodes for parameter P_j in an instance must be at most one. This is guaranteed by the following lemma.

---

[2]The proof works as follows. Suppose the OR rule for parameter P_j is not exclusive. Then there is an admissible language that satisfies two disjuncts of this rule. Use the value of parameters in this language to choose a child v-node for each p-node in T, like in the language to instantiate construction of the next section. In this way, we will find two p-nodes for P_j. But their LCA must be a p-node, hence we should have chosen two different v-nodes for it, a contradiction.

LEMMA 4.4. *Let I be an instance of a sampling structure $T = (P, V, E, \iota, \lambda)$ for simple rules $\mathcal{R}$. For each parameter P_j, there exists at most one node $p \in P_I$ such that $\iota(p) = j$.*

Proof. Suppose not, and let P_j be the first parameter having two p-nodes in I, say p_1 and p_2. Then, as each p-node in I has just one child and p_1 and p_2 belong to the same tree of T, the LCA of p_1 and p_2 must be a v-node, in contradiction with point 3 of Proposition 4.2. □

We now need to prove that $\mathbf{a}^I$ satisfies all the rules of $\mathcal{R}$.

LEMMA 4.5. *The tuple $\mathbf{a}^I$ satisfies all the rules in $\mathcal{R}$.*

Proof. We prove the lemma by finite induction on the parameter index j.

($j = 1$) The constraint formula for P_1 is always a tautology, hence there is only one p-node for it, and all its values are admissible.

($j - 1 \Rightarrow j$) We first consider the case in which $a_j^I = 0$. In this case, the boolean formula for P_j must be false, otherwise there would be a true disjunct, say $x_i = a_i^I$, with $i < j$, so that there would be a v-node v_i in I with $\lambda(v_i) = a_i^I$ such that one of its children p-nodes would not belong to I. Now, suppose $a_j^I \neq 0$. Let p be the node for P_j in I, and let v be its parent v-node. Then, by definition of sampling structure $x_{\iota(v)} = \lambda(v) = a_{\iota(v)}^I$ is a disjunct in the boolean formula for P_j, which is therefore true. □

We now consider an admissible language $\mathbf{a}$ and associate to it an instance $I_{\mathbf{a}}$. The construction is done recursively in a simple way:

- add to $I_{\mathbf{a}}$ the p-node for P_1, its child v-node v with $\lambda(v) = a_1$ and all p-nodes children of v;
- if $a_j \neq 0$, consider the only p-node for parameter P_j in the instance $I_{\mathbf{a}}$ being constructed, and add to $I_{\mathbf{a}}$ its child v-node v with $\lambda(v) = a_j$ and all p-nodes children of v
- if $a_j = 0$, do nothing;

We observe that step 2 of the previous construction can always be carried out. In fact, there is always just one p-node for P_j in case $a_j \neq 0$, as there must be a v-node corresponding to one of the disjuncts in the rule for P_j previously inserted in $I_{\mathbf{a}}$ ($\mathbf{a}$ is admissible) and all its children are also in $I_{\mathbf{a}}$. Hence, the following lemma holds.

LEMMA 4.6. *$I_{\mathbf{a}}$ is an instance of T.* □

We have just defined two mappings, one from admissible languages to instances and the other from instances to admissible languages. By construction of these mappings, it holds that they are one the inverse of the other, namely $I_{\mathbf{a}^I} = I$ and $\mathbf{a}^{I_{\mathbf{a}}} = \mathbf{a}$. This is sufficient to prove the following theorem.

THEOREM 4.7. *Let $\mathcal{R}$ be a simple set of rules and T be its sampling structure. Then instances of T and admissible languages of $\mathcal{R}$ are in bijection.* □

This theorem is the key to the sampling algorithm: in order to sample uniformly admissible languages for a simple rule set, we can sample uniformly instances from the associated sampling structure.

4.4. Uniform Sampling of Instances

We now turn to detail the sampling mechanism of instances, proving that it samples instances according to the uniform probability distribution. A key step towards the sampling algorithm is to count all possible instances of each tree and subtree of a sampling structure. This is necessary because, in order to sample uniformly, we need to know how many instances can be generated choosing one or another child v-node of a p-node. The counting function is defined inductively on the height of nodes of each tree of the sampling structure, distinguishing between v-nodes and p-nodes, as instances treat them differently.

DEFINITION 4.8. *Let $T = (P, V, E, \iota, \lambda)$ be a sampling structure for simple rules $\mathcal{R}$. The instance-counting function $N : P \cup V \to \mathbb{N}$ is defined recursively as follows:*

1) *for each v-node v of height $h(v) = 0$, $N(v) = 1$;*
2) *for each p-node p of height $h > 0$, with children $v_1, \ldots, v_k$, $N(p) = N(v_1) + \ldots + N(v_k)$;*
3) *for each v-node v of height $h > 0$, with children $p_1, \ldots, p_k$, $N(v) = N(p_1) \cdot \ldots \cdot N(p_k)$.*

The correctness of the previous definition is easily proved by induction on the height h of a node. Intuitively, at each internal p-node, we add to an instance just one child, hence we need to add up the number of instances of the children, while for internal v-nodes, we add all children to each instance, hence we need to consider all possible combinations , hence take the product. If $p_1, \ldots, p_k$ are the roots of the distinct trees of T, then the total number of instances (and, accordingly, of admissible languages) is $N(T) = N(p_1) \cdot \ldots \cdot N(p_k)$.

The sampling algorithm needs to pick an instance among the possible ones, according to the uniform distribution. The idea is to choose an instance, following Definition 4.3, by choosing a v-node for each p-node inserted in the

instance. More precisely, we say that a p-node is *active* if it has been inserted in the instance, but one of its children v-nodes has still to be selected. The sampling algorithm that constructs $I = (P, V, E)$ is the following:

$P \leftarrow \{\text{roots of } T\}$
$A \leftarrow \{\text{roots of } T\}$ {A is a set containing active p-nodes}
while $A \neq \emptyset$ **do**
  Remove p from A
  Choose child v_i of p among $v_1, \ldots, v_k$ with probability $\frac{N(v_i)}{N(p)}$
  Add v_i to V and (p, v_i) to E
  Add the children nodes $p_1, \ldots, p_s$ of v_i to P and to A, and (v_i, p_i) to E
end while

The previous algorithm samples an instance with uniform probability. The fact that it generates instances is straightforward (it replicates the recursive definition of an instance). As for the uniform probability, observe that the probability π with which a generic instance is generated is the product of the probabilities of the choices performed in its internal p-nodes. Now, pick a generic factor of this product π, namely $\frac{N(v)}{N(p)}$. If v is an internal v-node, then all its children $p_1, \ldots, p_s$ are inserted in the instance, and they contribute to the product π with factors $\frac{N(v_i)}{N(p_i)}$. Now, the numerator $N(v) = N(p_1) \cdot \ldots \cdot N(p_s)$ cancels out with the terms $N(p_i)$ of the denominator. Therefore, the only factors left at numerator are $N(v)$ for v leaf, hence the numerator equals 1. Similarly, the only factors that remain at the denominator are the terms $N(p)$, for each root p of T. It follows that $\pi = \frac{1}{N(T)}$, showing that instances are sampled uniformly.

4.5. Compact Sampling Trees

The complexity of the sampling algorithm of the previous section is linear in the size of T. However, as anticipated at the end of Section 4, the size of T can grow quicker than linearly with n, due to the fact that exclusive OR rules may introduce many p-nodes for the same parameter P_j. For instance, consider a sampling structure for parameters $P_1, \ldots, P_n$, with values in $\{+, -\}$, and subject to rules $\mathcal{R} = \{(P_1, true), (P_2, x_1 = +), \ldots, (P_i, (x_{i-1} = +) \vee (x_{i-2} = -)), \ldots\}$. It is easy to see that there are exactly $i - 1$ p-nodes for parameter P_i ($i > 2$), hence the total number of p-nodes is quadratic in n (see Figure 2 left).

This combinatorial growth can be avoided by relaxing the constraint that T is a collection of trees, and allowing it to be a collection of direct acyclic graphs (DAGs). The key observation, in fact, is that all subtrees rooted at p-nodes p_i for parameter P_j are isomorphic. Hence, we can merge them into a single tree, effectively inserting one single p-node for P_j. If we perform this collapsing from the last parameter backwards, at the end we obtain a collection of DAGs,

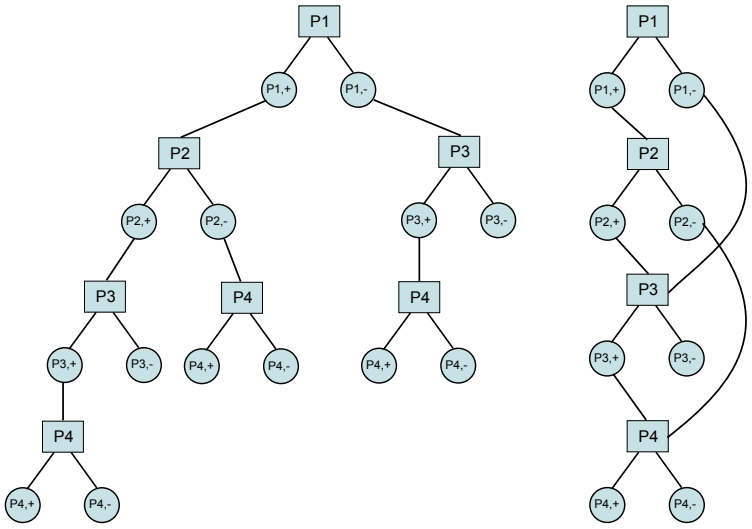

Figure 2: An example of a sampling structure (left) and of a compact sampling structure (right) for parameters $P_1, \ldots, P_4$, all taking values in $\{+, -\}$, and subject to rules $\mathcal{R} = \{(P_1, true), (P_2, x_1 = +), (P_3, (x_1 = +) \vee (x_2 = -)), (P_4, (x_2 = +) \vee (x_3 = -))\}$.

with exactly n distinct p-nodes, one for each parameter. We call this object a *compact sampling structure* (see Figure 2 right).

Instances for a compact sampling structure can be defined similarly to sampling structures. It is easy to see that the set of instances is the same for both structures (the exclusive nature of OR rules implies that we can reach a p-node only through a single incoming edge at a time). Hence, sampling structures and compact sampling structures can be used interchangeably. This linear representation allows one to sample an admissible language in time $O(n)$, using $O(n + |\bigcup_i \mathcal{D}_i|)$ extra space.

5. Parameter Merging

In the previous section, we discussed how to construct an efficient data structure to sample uniformly from a space of languages defined by a restricted set of rules, namely exclusive disjunctions of literals. Unfortunately, the rules governing real parameters of the universal grammar can be dramatically more complex. In these cases, the previous approach is not applicable, hence we have to resort to the rejection sampling strategy discusses in Section 3. However,

this still results in a sample space which is too large in practice. In order to reduce the dimension of the sampling space overapproximating the space of admissible languages, we can use a different approach, merging sets of parameters into a bigger macro-parameter so that the resulting set of rules simplifies. Therefore, the parameters to be merged are those involved in formulae that are not exclusive disjunctions.

As an example, consider four parameters, P_1, P_2, P_3, P_4, taking values in $\{+, -\}$ and subject to rules $\mathcal{R} = \{(P_1, true),\ (P_2, x_1 = +),\ (P_3, true),\ (P_4, (x_2 = +) \wedge (x_3 = -))\}$. The rules for P_1, P_2, and P_3 are simple, but the rule for P_4 is not. A possible solution is merging parameters P_2 and P_3 into a macro-parameter $P_{2,3}$, which can take four values, i.e. $\{++, +-, -+, --\}$. This makes the rule for P_4 simple: $(P_4, x_{2,3} = +-)$. However, merging P_2 and P_3 creates an additional problem: the possible values that $P_{2,3}$ can take depend on the value of P_1, contradicting the basic property of the rule sets, i.e. that the truth of a rule implies that a parameter is meaningful and can take any value, independently of other parameters. The solution is simple: we need to merge P_1 and $P_{2,3}$ into a bigger macro-parameter. The so-obtained parameter $P_{1,2,3}$ can take only five possible values, $\{+++, ++-, +-+, +--, -00\}$, due to restrictions imposed by internal rules between parameters that are merged.

The previous example gives an intuition of the problems involved in parameter merging. More precisely, one has to merge a (minimal) set of parameters satisfying the following consistency condition: the rule governing its meaningfulness is of the exclusive OR type, and when the rule is true, each of its values is admissible. A simple way to guarantee this constraint is to require a macro-parameter to be independent. The set of possible values of a macro-parameter is constructed by taking into account the existent relations holding between merged parameters.

Practically, once we select the first set of parameters to be merged by looking at a single, non-simple rule, we also need to merge with them all their ancestors parameters. This approach has the drawback of generating macro-parameters with large set of values. Hence, there is a trade off between space complexity of the sampling structures and time complexity of the rejection sampling approach.

6. Experimental Results and Conclusions

In this section, we present some experimental results of our method, applied on a set of 62 parameters, governed by the rules in Table 1. Details on these parameters, and on the resulting phylogenies, can be found in [1].

As we can see, some rules tend to be highly complex, so we had to merge several parameters before constructing a sampling structure. We merged parameters 25, 26, 27, 30, 31, 32, 33, 37, 39, 40, 42, 43, 44, 45, 46, 47, 48, 49 into

$(P_1, true),\ (P_2, P_1 = +),\ (P_3, P_2 = +),\ (P_4, P_1 = +),\ (P_5, P_2 = +),\ (P_6, P_5 = +),\ (P_7, true)$
$(P_8, P_7 = +),\ (P_9, P_6 = +),\ (P_{10}, P_5 = - \vee P_6 = - \vee P_7 = +),\ (P_{11}, P_{10} = -),\ (P_{12}, P_7 = +)$
$(P_{13}, true),\ (P_{14}, P_1 = + \wedge P_8 = + \wedge P_{13} = -),\ (P_{15}, P_{12} = + \wedge P_{14} = -),\ (P_{16}, P_7 = + \wedge P_{12} = -)$
$(P_{17}, P_7 = +),\ (P_{18}, P_5 = +),\ (P_{19}, P_5 = +),\ (P_{20}, true),\ (P_{21}, true)$
$(P_{22}, (P_5 = - \vee P_6 = - \vee P_7 = +) \wedge P_{21} = -),\ (P_{23}, P_{22} = +),\ (P_{24}, P_{21} = + \vee P_{22} = +),\ (P_{25}, true)$
$(P_{26}, P_{25} = +),\ (P_{27}, P_{25} = - \vee P_{26} = +),\ (P_{28}, P_{12} = +)$
$(P_{29}, (P_{25} = - \vee P_{26} = +) \wedge (P_{25} = + \vee P_{27} = + \vee P_{28} = +)),\ (P_{30}, true),\ (P_{31}, true),\ (P_{32}, P_{31} = +)$
$(P_{33}, P_{32} = +),\ (P_{34}, true),\ (P_{35}, P_6 = + \wedge (P_{32} = + \vee P_{34} = +)),\ (P_{36}, P_{32} = + \wedge P_{14} = -)$
$(P_{37}, true),\ (P_{38}, P_{37} = +),\ (P_{39}, P_{31} = + \wedge P_{37} = +),\ (P_{40}, P_{39} = -)$
$(P_{41}, P_{31} = - \vee P_{39} = + \vee P_{40} = +),\ (P_{42}, true),\ (P_{43}, true),\ (P_{44}, P_{43} = -)$
$(P_{45}, P_{27} = + \wedge P_{44} = +),\ (P_{46}, P_{31} = + \wedge P_{44} = +),\ (P_{47}, P_{46} = +),\ (P_{48}, P_{47} = +),\ (P_{49}, P_{48} = +)$
$(P_{50}, P_{31} = - \vee P_{43} = + \vee P_{49} = +),\ (P_{51}, true),\ (P_{52}, P_{51} = +)$
$(P_{53}, (P_5 = - \vee P_6 = - \vee P_8 = +) \wedge (P_{22} = - \vee P_{51} = +)),\ (P_{54}, P_{30} = + \wedge (P_{45} = - \vee P_{46} = -))$
$(P_{55}, (P_{30} = - \vee P_{43} = + \vee P_{45} = + \vee P_{54} = -) \wedge (P_{31} = - \vee P_{32} = -) \wedge P_{42} = -)$
$(P_{56}, (P_5 = - \vee P_6 = - \vee P_8 = +) \wedge (P_{22} = - \vee P_{23} = - \vee P_{29} = -))$
$(P_{57}, (P_{30} = - \vee P_{43} = + \vee P_{45} = + \vee P_{54} = -) \wedge (P_{33} = + \vee P_{32} = - \vee P_{55} = +) \wedge P_{31} = + \wedge P_{42} = -)$
$(P_{58}, P_{46} = + \wedge (P_{47} = - \vee P_{48} = - \vee P_{49} = -) \wedge P_{57} = +),\ (P_{59}, P_{28} = -)$
$(P_{60}, P_{30} = - \vee P_{43} = + \vee P_{45} = +),\ (P_{61}, P_{45} = - \vee P_{46} = - \vee P_{47} = - \vee P_{48} = - \vee P_{13} = +)$
$(P_{62}, (P_5 = - \vee P_6 = - \vee P_{12} = -) \wedge (P_{13} = + \vee P_{14} = +))$

Table 1: Rules for the parameters in the PCM dataset [6, 1].

a large macro-parameter with 3916 possible values, and parameters 1, 2, 5, 6, 7, 8, 12, 13, 14, 21, 22 into a smaller macro-parameter with 168 values. These macro-parameters were constructed automatically, using a greedy heuristic. We started by merging parameters in the head of a selected rule and we iteratively merged more and more parameters by considering those appearing in the heads of complex rules of merged parameters. This procedure was stopped when a bound on the number of states of the macro-parameter was met. Intuitively, the problem with parameter merging is that the complex dependencies of Table 1 tend to produce a single large macro-parameter containing almost all parameters, hence we had to resort to heuristics to break this tendency.

The compact sampling structure has been constructed leaving out parameters 29, 35, 36, 53, 56, and 61 and fixing them uniformly in the set $\{+, -, 0\}$ in each attempt of the rejection sampling. On average, the rejection sampler takes slightly less that 65 trials to find an admissible language, and the average time to sample a single language is around 0.6 milliseconds. This allows one to generate 10 million languages, a reasonably large sample to extract meaningful properties of the language space, in about 2 hours. Experiments were performed on a laptop with a Core 2 Due T9300 CPU and 2 Gb of RAM.

The experimental results just presented suggest that this sampling method is feasible to deal with applications of the size of those required in [6]. Problems can arise if the number of parameters is increased and the rule structure continues to have the tendency of merging all parameters together. In such cases, we

may try to improve the sampler by using a Gibbs sampling scheme [2, 7], isolating subsets of parameters that are sufficiently (but not perfectly) separated from the others.

This sampling algorithm can be used to assess the statistical significance of real world data against background noise. For instance, we can use it to check if two specific languages are significantly more similar than two languages in a pair drawn at random. It can also be used to investigate the properties of the space of admissible languages, which can shed further light on the intrinsic structure of universal grammars. In general, this method allows the grounding of PCM on firmer statistical bases.

References

[1] *Linguistics laboratory of the University of Trieste*, http://www2.units.it/linglab/.
[2] S.P. Brooks, *Markov chain Monte Carlo method and its application*, J. Roy. Statist. Soc. D **47** (1998), 69–100.
[3] N. Chomsky, *Rules and representation*, Columbia University Press, New York (1980).
[4] J. Felsenstein, *Inferring phylogenies*, Sinauer, Sunderland (2004).
[5] C. Guardiano and G. Longobardi, *Parametric comparison and language taxonomy*, in M. Batllori, M.L. Hernanz, C. Picallo and F. Roca, *Grammaticalization and Parametric Variation*, Oxford University Press, Oxford (2005), pp. 149–174.
[6] G. Longobardi and C. Guardiano, *Evidence for syntax as a signal of historical relatedness*, Lingua **119** (2009), 1679–1706.
[7] D.J.C. MacKay, *Introduction to Monte Carlo methods*, in M.I. Jordan, *Learning in graphical models*, NATO Science Series, Kluwer, U.S.A. (1998), pp. 175–204.
[8] A.M.S. McMahon and R. McMahon, *Language classification by numbers*, Oxford University Press, Oxford (2005).

Authors' addresses:

Luca Bortolussi
Dipartimento di Matematica e Informatica
Università degli Studi di Trieste
Via Valerio 12/1, 34127 Trieste, Italy
E-mail: luca@dmi.units.it

Andrea Sgarro
Dipartimento di Matematica e Informatica
Università degli Studi di Trieste
Via Valerio 12/1, 34127 Trieste, Italy
E-mail: sgarro@units.it

Received May 31, 2011
Revised August 30, 2011