Journal of Logic, Language, and Information **5:** 115–155, 1996.

Identification in the Limit of Categorial Grammars

MAKOTO KANAZAWA
Department of Cognitive and Information Sciences, Faculty of Letters, Chiba University, 1–33 Yayoi-cho, Inage-ku, Chiba-shi, 263, Japan
kanazawa@cogsci.l.chiba-u.ac.jp

(Received 4 August 1993; in final form 22 January 1996)

Abstract. It is proved that for any k, the class of classical categorial grammars that assign at most k types to each symbol in the alphabet is learnable, in the Gold (1967) sense of identification in the limit from positive data. The proof crucially relies on the fact that the concept known as finite elasticity in the inductive inference literature is preserved under the inverse image of a finite-valued relation. The learning algorithm presented here incorporates Buszkowski and Penn's (1990) algorithm for determining categorial grammars from input consisting of functor-argument structures.

Key words: categorial grammar, finite elasticity, functor-argument struture, identification in the limit, inductive inference, learnability.

1. Introduction

One of the major goals of modern linguistic theory is to give an account of how language learning is possible (see, for example, Chomsky 1986). A child growing up in a linguistic community comes to possess a rule system, a grammar, for the language spoken by the community, on the basis of samples of speech presented to her without any explicit instruction. A remarkable fact is that a normal child is capable of learning any natural language, given adequate input: a child brought up in an English speaking environment comes to speak English, but the same child would come to speak Japanese if she were brought up in a Japanese speaking environment. Thus, whatever mechanism underlies first language acquisition must be able to deal with every possible natural language. This fact puts a substantial constraint on linguistic theory. The class of possible natural languages determined by an adequate linguistic theory must have the property of being *learnable* in the sense that there exists a single mechanism that has the capacity to learn any language in the class.

Learnability theory is an attempt to illuminate the concept of learnability using a mathematical model of learning. The approach adopted in this paper originates in the work of Gold (1967), for which first language acquisition was one of the primary motivations.*

* *Computational learning theory* (see, e.g., Kearns and Vazirani 1994), a branch of theoretical computer science that has developed since early 1980's, embraces various other approaches to

$$s_0, \quad s_1, \quad s_2, \quad \ldots, \quad s_i, \quad \ldots$$
$$G_0, \quad G_1, \quad G_2, \quad \ldots, \quad G_i, \quad \ldots$$

Figure 1. Language learning. G_i is the grammar produced by the learner on the basis of the sentences $s_0, s_1, s_2, \ldots, s_i$.

In Gold's model, language learning is an infinite process in which the learner is presented with an infinite stream of sentences of the target language, one sentence at a time. Every time the learner encounters a new sentence, she makes a guess as to the identity of the target language, on the basis of the (finitely many) sentences she has encountered so far. Each guess is made in the form of a grammar. As the learner receives more and more data, she makes successive guesses, each possibly different from the previous ones. Thus, corresponding to the infinite sequence of sentences presented to the learner, she produces an infinite sequence of grammars (Figure 1). Two assumptions are made about the infinite sequence of sentences given to the learner. Firstly, it is assumed that only grammatical sentences of the target language appear in the sequence. This assumption corresponds to the observation that no systematic negative evidence (information about ungrammatical sentences) is available to the child learning a first language.* The second assumption is that every sentence of the target language eventually appears in the infinite sequence. This is to ensure that the learner is provided with enough information to distinguish different languages. Now, in this model, learning is considered to be successful if there is a point beyond which the learner's guess always stays the same, and that guess is a correct grammar for the target language. Note that under this criterion, one can never tell at any finite stage whether successful learning has taken place, since the learner's guess might change at the next moment. Gold called this criterion of successful learning *identification in the limit.*

A class of languages is said to be learnable in this model if there exists a learner that successfully learns a correct grammar from any infinite sequence of sentences corresponding to a language in the class. It is important to emphasize that learnability is a property of a class of languages, not of an individual language. Since any mechanistic procedure for converting finite sequences of sentences to grammars counts as a learner, any single language would be trivially learnable—a correct grammar for the language could be simply wired into the learning mechanism. The fact that the learner must react differently to samples from different languages makes learning a class of languages a non-trivial task.

In the model sketched above, the information available to the learner about the target language at any point consists only of *positive data* about the language

---

learning, which are less relevant to the concerns about first language acquisition. For linguistically motivated work within the Gold paradigm, see Wexler and Culicover 1980.

* Also, the effects of ungrammatical intrusions are considered negligible. See Wexler and Culicover 1980 for some discussions of both issues.

(i.e., grammatical sentences of the language), mirroring the empirical fact about the environment in which first language acquisition takes place. Gold (1967) also considered learning from complete data, which includes both positive and *negative data* (sentences ungrammatical in the target language) marked as such. Naturally, complete data makes learning much easier (see Gold 1967). As complete data seems to have little relevance to first language acquisition and linguistic theory, this paper concerns only with learning from positive data.

Identification in the limit from positive data is interesting precisely because it makes the resulting notion of learnability quite restrictive. In his original 1967 paper studying learnability of formal languages, Gold revealed that none of the four levels of the Chomsky Hierarchy is learnable under this criterion. This result of Gold's had been widely taken to mean that learning from positive data alone is too difficult a task to be of much interest, until around 1980, when Angluin (1980a, 1980b, 1982) presented non-trivial learnable classes that cross-cut the Chomsky Hierarchy. An impressive result was subsequently obtained by Shinohara (1990a, 1990b) which says that placing any finite bound on the number of rules used in context-sensitive grammars results in a learnable class. Thus the initial pessimism about identification in the limit from positive data seems to have been misplaced.

In this paper, we prove a result similar to Shinohara's with respect to *categorial grammars*:* the class of categorial grammars that assign at most k types to each symbol in the alphabet is learnable, for any k. Our result builds on the work of Buszkowski and Penn (Buszkowski 1987a, 1987b, Buszkowski and Penn 1990), who describe natural algorithms for finding categorial grammars from data consisting of *functor-argument structures*, and makes essential use of the concept known as *finite elasticity* in the inductive inference literature, which is a property of language classes. Given Shinohara's theorem, our learnability result is not surprising; indeed, a straightforward reduction of our result to Shinohara's is possible (Appendix 7). Nevertheless, our method of proof may be of independent interest, as it suggests an extension not covered by Shinohara's theorem**; moreover, we provide a concrete learning algorithm based on Buszkowski's algorithm.

2. Preliminaries

2.1. IDENTIFICATION IN THE LIMIT

In this section, we introduce some basic concepts in learnability theory. As we explained in Section 1, we are only interested in Gold's notion of *identification in the limit from positive data*, which we will simply call *learning* in what follows.‡

* Categorial grammar originated in the work of Ajdukiewicz (1935) and was refined by the work of Bar-Hillel (1953) and Lambek (1958, 1961).

** See Section 9.2 of Kanazawa 1994a for an extension to a formalism like Montague grammar (Montague 1973).

‡ For introductions to different aspects of the field, see Angluin and Smith 1983, Osherson et al. 1986, Ohserson et al. 1994, Chapter 2 of Kanazawa 1994a, and de Jongh and Kanazawa 1995.

2.1.1. *Basic Definitions*

In order to formulate the question of learnability, we first need to specify three things:

- a set Ω ('hypothesis space'),
- a set $\mathfrak{S}$ ('sample space'),
- a function L that maps elements of Ω to subsets of $\mathfrak{S}$, i.e., $\mathrm{L}\colon \Omega \to \mathrm{pow}(\mathfrak{S})$.

Ω can be any class of finitary objects on which mechanical computation can be carried out. For instance, Ω could be the set of natural numbers, or it could consist of programs for a certain kind of abstract machine (e.g., Turing machine programs), or formal grammars of some kind (e.g., context-free grammars). Formally, we can identify Ω with a certain recursive set of strings over some finite alphabet. Elements of Ω are called *grammars*.

The set $\mathfrak{S}$ is a certain recursive subset of Σ^* for some fixed finite alphabet Σ. In many cases, one simply takes $\mathfrak{S} = \Sigma^*$. Elements of $\mathfrak{S}$ are called *sentences*, and subsets of $\mathfrak{S}$ are called *languages*.

If G is a grammar in Ω, then $\mathrm{L}(G)$ is called the *language generated by* G. We can think of G as a name for $\mathrm{L}(G)$, so we call L the *naming function*. The question whether $s \in \mathrm{L}(G)$ holds between $s \in \mathfrak{S}$ and $G \in \Omega$ is called the *universal membership problem*. The naming function is assumed to be such that the universal membership problem is at least semi-decidable (r.e.).

A triple $\langle \Omega, \mathfrak{S}, \mathrm{L} \rangle$ satisfying the above conditions is called a *grammar system*.

EXAMPLE 1. Let Σ be any finite alphabet, and let Γ be a countably infinite set of symbols disjoint from Σ. Let CFG be the set of all context-free grammars over Σ whose non-terminal symbols are in Γ. For every $G \in \mathrm{CFG}$, let $\mathrm{L}(G)$ be the language generated by G under the usual interpretation. Then $\langle \mathrm{CFG}, \Sigma^*, \mathrm{L} \rangle$ is a grammar system. (In this grammar system, the universal membership problem '$w \in \mathrm{L}(G)$?' is decidable in time bounded by a polynomial in the combined size of w and G.)

Let $\langle \Omega, \mathfrak{S}, \mathrm{L} \rangle$ be a grammar system. A *learning function* is a partial function φ that maps non-empty finite sequences of sentences to grammars, i.e.,

$$\varphi\colon \bigcup_{k \geq 1} \mathfrak{S}^k \rightharpoonup \Omega.$$

Usually, we are only interested in learning functions that are effectively computable. The term *learning algorithm* means an algorithm that computes a learning function, and will sometimes be used interchangeably with 'computable learning function'.

Let

$$\langle s_i \rangle_{i \in \mathbb{N}} = \langle s_0, s_1, s_2, \ldots \rangle$$

be an infinite sequence of sentences from $\mathfrak{S}$. Given $\langle s_i \rangle_{i \in \mathbb{N}}$, a learning function φ determines a grammar

$$G_i = \varphi(\langle s_0, \ldots, s_i \rangle)$$

for each $i \in \mathbb{N}$ such that φ is defined on $\langle s_0, \ldots, s_i \rangle$.* We say that φ *converges* to G on $\langle s_i \rangle_{i \in \mathbb{N}}$ if $G_i = \varphi(\langle s_0, \ldots, s_i \rangle)$ is defined and is equal to G for all but finitely many $i \in \mathbb{N}$—or, equivalently, if there exists an $n \in \mathbb{N}$ such that for all $i \geq n$, G_i is defined and is equal to G.

Let $\mathcal{G}$ be a class of grammars in Ω. $\mathcal{G}$ determines the corresponding class of languages, $\mathrm{L}(\mathcal{G}) = \{ \mathrm{L}(G) \mid G \in \mathcal{G} \}$.

DEFINITION 2. Let a grammar system $\langle \Omega, \mathfrak{S}, \mathrm{L} \rangle$ be given, and let $\mathcal{G} \subseteq \Omega$. A learning function φ is said to *learn* $\mathcal{G}$ if the following condition holds:**

for every language L in $\mathrm{L}(\mathcal{G})$,
for every infinite sequence $\langle s_i \rangle_{i \in \mathbb{N}}$ that enumerates the elements of L (i.e., $\{ s_i \mid i \in \mathbb{N} \} = L$),
there exists some G in $\mathcal{G}$ such that $\mathrm{L}(G) = L$ and
φ converges to G on $\langle s_i \rangle_{i \in \mathbb{N}}$.

A word about the above definition is in order. Let $L \in \mathrm{L}(\mathcal{G})$. If there is more than one grammar G in $\mathcal{G}$ such that $\mathrm{L}(G) = L$, a learning function φ that learns $\mathcal{G}$ is not required to converge to the same grammar on different infinite sequences that enumerate the elements of L; the grammar G that φ converges to can depend on the order in which the elements of L are enumerated.

A class $\mathcal{G}$ of grammars is said to be *learnable* if there is a computable learning function that learns $\mathcal{G}$.

EXAMPLE 3. Take the grammar system $\langle \mathrm{CFG}, \Sigma^*, \mathrm{L} \rangle$ of context-free grammars over alphabet Σ. Let $\mathcal{G}$ be the subclass of CFG consisting of grammars whose rules are all of the form

$$\mathrm{S} \rightarrow w,$$

where $w \in \Sigma^*$. Observe that $\mathrm{L}(\mathcal{G})$ is exactly the class of finite languages over Σ. We show that $\mathcal{G}$ is learnable. Define a computable learning function φ as follows:

$$\varphi(\langle s_0, \ldots, s_i \rangle) = \langle \Sigma, \{\mathrm{S}\}, \mathrm{S}, P \rangle,$$

where

$$P = \{ \mathrm{S} \rightarrow s_0, \ldots, \mathrm{S} \rightarrow s_i \}.$$

Let L be a finite language over Σ. If $\langle s_i \rangle_{i \in \mathbb{N}}$ enumerates L, then there exists an $n \in \mathbb{N}$ such that for all $i \geq n$, $\{s_0, \ldots, s_i\} = L$. This means that for all $i \geq n$, $\varphi(\langle s_0, \ldots, s_i \rangle) = G$, where $G = \langle \Sigma, \{\mathrm{S}\}, \mathrm{S}, \{ \mathrm{S} \rightarrow w \mid w \in L \} \rangle$. Clearly, $\mathrm{L}(G) = L$. Thus φ learns $\mathcal{G}$. In fact, in any grammar system $\langle \Omega, \mathfrak{S}, \mathrm{L} \rangle$, any class $\mathcal{G}$ of grammars such that $\mathrm{L}(\mathcal{G})$ consists of exactly the finite languages is learnable, if there exists a computable function ψ that maps each finite language $D \subseteq \mathfrak{S}$ to a grammar $\psi(D) \in \mathcal{G}$ such that $\mathrm{L}(\psi(D)) = D$.

* $\langle s_0, \ldots, s_i \rangle$ denotes a non-empty finite sequence. If $i = 0$, this is $\langle s_0 \rangle$.

** This definition is somewhat unorthodox in that what is learned is a class of grammars, rather than a class of languages in the sense of sets of sentences.

A class $\mathcal{G}$ of grammars is said to be *non-effectively learnable* if there is a learning function (not necessarily computable) that learns $\mathcal{G}$. Clearly, learnability implies non-effective learnability. We are primarily interested in (effective) learnability, but non-effective learnability is also a useful companion notion.*

2.1.2. *Finite Elasticity*

In Example 3, we noted that in a reasonable grammar system, a class $\mathcal{G}$ of grammars such that $\mathrm{L}(\mathcal{G})$ consists of exactly the finite languages is learnable. In a certain sense, this situation cannot be improved. In the original article that introduced identification in the limit, Gold (1967) proved the following important theorem:

THEOREM 4 (Gold). *In any grammar system, a class $\mathcal{G}$ of grammars is not (non-effectively) learnable if $\mathrm{L}(\mathcal{G})$ contains all finite languages and at least one infinite language.*

Let $\mathcal{G}$ be a class of grammars and let $\mathcal{L}$ be the corresponding class of languages, i.e., $\mathcal{L} = \mathrm{L}(\mathcal{G})$. As Gold's theorem illustrates, sometimes the fact that $\mathcal{L}$ has a certain property is enough to conclude that $\mathcal{G}$ is not (even non-effectively) learnable. (In fact, non-effective learnability is completely determined by $\mathcal{L}$.)

On the other hand, under certain conditions, a property of the class $\mathcal{L}$ of languages generated may imply learnability of the given grammar class $\mathcal{G}$. One such property is what Wright (1989) dubbed *finite elasticity*. Finite elasticity is defined as the negation of *infinite elasticity*, defined below.**

DEFINITION 5 (Infinite elasticity). A class $\mathcal{L}$ of languages is said to have *infinite elasticity* if there exist an infinite sequence $\langle s_n \rangle_{n \in \mathbb{N}}$ of sentences and an infinite sequence $\langle L_n \rangle_{n \in \mathbb{N}}$ of languages in $\mathcal{L}$ such that for all $n \in \mathbb{N}$,

$$s_n \notin L_n,$$

and

$$\{s_0, \ldots, s_n\} \subseteq L_{n+1}.$$

DEFINITION 6 (Finite elasticity). A class $\mathcal{L}$ of languages is said to have *finite elasticity* if it does not have infinite elasticity.

Building on Angluin's (1980b) work, Wright (1989) proves that for a class of grammars to be learnable, it is sufficient that the class of languages generated has finite elasticity, under certain provisions.

---

* See Osherson, Stob, and Weinstein 1986, p. 53 for this point.

** Wright's (1989) originial definition of infinite and finite elasticity was in error, and was later corrected by Motoki, Shinohara, and Wright (1991).

THEOREM 7 (Wright). *Let* $\langle \Omega, \mathfrak{S}, \mathrm{L} \rangle$ *be a grammar system for which the universal membership problem is decidable, and let* $\mathcal{G}$ *be an r.e. subset of* Ω. *If* $\mathrm{L}(\mathcal{G})$ *has finite elasticity, then* $\mathcal{G}$ *is learnable.*

Note that finite elasticity is far from a necessary condition for learnability (under the provisions in Theorem 7): Example 3 gave a learnable recursive class of grammars such that $\mathrm{L}(\mathcal{G}))$ has *in*finite elasticity. Nevertheless, finite elasticity is in practice a very useful condition, as it is often relatively easy to prove that a language class has finite elasticity. For example, Shinohara's theorem, mentioned in Section 1, was proved by showing that the class of languages generated by context-sensitive grammars with at most k rules has finite elasticity.*

2.2. Classical Categorial Grammars

In this section, we introduce the grammar system of classical categorial grammars, and state the main result of the paper.

2.2.1. *Basic Definitions*

In classical categorial grammar, each symbol in the alphabet is associated with a finite number of *types*. Types are constructed from *primitive types* by two type-forming operators, / and \. If we let Pr and Tp denote the set of primitive types and the set of types, respectively, Tp is defined to be the smallest set satisfying the following conditions:

- $\mathrm{Pr} \subseteq \mathrm{Tp}$,
- if $A \in \mathrm{Tp}$ and $B \in \mathrm{Tp}$, then $A\backslash B \in \mathrm{Tp}$.
- if $A \in \mathrm{Tp}$ and $B \in \mathrm{Tp}$, then $B/A \in \mathrm{Tp}$.

The letters A, B, C, possibly with subscripts, range over types. Type A is said to be a *subtype* of type B if A occurs in B. More precisely, A is a subtype of B if and only if either (i) $A = B$ or (ii) $B = B_1\backslash B_2$ or $B = B_2/B_1$ and A is a subtype of B_1 or B_2.

We assume that the set Pr is countably infinite** and no element of Pr is of the form $A\backslash B$ or B/A. One member t of Pr is singled out as the *distinguished type*. The members of Pr other than t are called *variables*, for reasons that will become clear later. The set of variables is denoted Var. Thus, $\mathrm{Pr} = \{t\} \cup \mathrm{Var}$. The letters x, y, z, possibly with subscripts, range over variables.

---

* To be accurate, Shinohara's (1990a) result was stated in terms of the related formalism of *length-bounded elementary formal systems*.

** Each grammar uses only finitely many primitive types. However, there is no bound on the number of primitive types a grammar can use, so it is necessary and sufficient to have a countably infinite supply of primitive types.

In categorial grammar, the combinatorial properties of types are completely determined by their shape. There are two modes of type combination, *Backward Application*:*

$$A,\ A\backslash B \Rightarrow B$$

and *Forward Application*:

$$B/A,\ A \Rightarrow B.$$

A non-empty sequence of types $A_1, \ldots, A_n$ is said to *derive* a type B, or, in symbols,

$$A_1,\ \ldots,\ A_n \Rightarrow B,$$

if repeated applications of the rules of Backward and Forward Application to the sequence $A_1, \ldots, A_n$ results in B. A formal definition follows:

DEFINITION 8. The relation $\Rightarrow \subseteq \mathrm{Tp}^+ \times \mathrm{Tp}$ is defined to be the smallest relation satisfying the following clauses:

- For all $A \in \mathrm{Tp}$, $A \Rightarrow A$.
- For all $\Gamma, \Delta \in \mathrm{Tp}^+$ and for all $A, B \in \mathrm{Tp}$,
  - if $\Gamma \Rightarrow A$ and $\Delta \Rightarrow A\backslash B$, then $\Gamma,\ \Delta \Rightarrow B$, and
  - if $\Gamma \Rightarrow B/A$ and $\Delta \Rightarrow A$, then $\Gamma,\ \Delta \Rightarrow B$.

Let Σ be a fixed alphabet. A *classical categorial grammar* over Σ is any finite relation G between Σ and Tp ($G \subset \Sigma \times \mathrm{Tp}$ and G is finite). For a symbol $c \in \Sigma$ and a type $A \in \mathrm{Tp}$, if $\langle c, A\rangle \in G$, we say that G *assigns* A *to* c, and often write $G\colon c \mapsto A$. (Henceforth, we will simply say 'grammar' to mean classical categorial grammar, as long as no confusion is likely to arise.)

Let G be a grammar over Σ. G *generates* a string $c_1 \ldots c_n \in \Sigma^+$ if and only if there are types $A_1, \ldots, A_n$ such that $G\colon c_i \mapsto A_i$ $(1 \leq i \leq n)$ and $A_1,\ \ldots,\ A_n \Rightarrow t$. The set of strings generated by G is called the *language* of G and is denoted $\mathrm{L}(G)$.

EXAMPLE 9. Let $\Sigma = \{\mathtt{a}, \mathtt{b}\}$. The following is a classical categorial grammar that generates a context-free language $\{\, \mathtt{a}^n\mathtt{b}^n \mid n \geq 1 \,\}$:

$$\begin{aligned} G_2\colon\ \mathtt{a} &\mapsto t/x,\ \ (t/x)/t, \\ \mathtt{b} &\mapsto x \end{aligned}$$

(We use notation like this to display all type assignments of a grammar. The above means $G_2 = \{\langle \mathtt{a}, t/x\rangle, \langle \mathtt{a}, (t/x)/t\rangle, \langle \mathtt{b}, x\rangle\}$.)

* Some people (e.g., Dowty (1988) and Steedman (1985, 1987, 1988)) write $B\backslash A$ for what we write $A\backslash B$. Our notation is that established by Lambek (1958) and followed by Bar-Hillel (1960).

We let CatG denote the class of all classical categorial grammars over Σ. Then $\langle \mathrm{CatG}, \Sigma^+, \mathrm{L} \rangle$ constitutes a grammar system. It is known that the class $\{ \mathrm{L}(G) \mid G \in \mathrm{CatG} \}$ of languages generated by a classical categorial grammar coincides with the class of ϵ-free context-free languages (Bar-Hillel, Gaifman, and Shamir 1960). As in the case of context-free grammars, the universal membership problem '$w \in \mathrm{L}(G)$?' for the grammar system of classical categorial grammars is decidable in time bounded by a polynomial in the combined size of w and G.

2.2.2. *Rigid and k-Valued Grammars*

If a grammar $G \subseteq \Sigma \times \mathrm{Tp}$ is a partial function from Σ to Tp, G is called a *rigid* grammar.* A rigid grammar assigns either zero or one type to each symbol in the alphabet. If a grammar G assigns at most k types to each symbol in the alphabet, that is, if $|\{ A \mid G\colon c \mapsto A \}| \leq k$ for all $c \in \Sigma$, we call G a *k-valued* grammar. (A rigid grammar is a one-valued grammar.)

EXAMPLE 10. The following grammar G_1 generates the regular language $\mathtt{a}^*\mathtt{b}$:

$$G_1\colon \mathtt{a} \mapsto t/t,$$
$$\mathtt{b} \mapsto t.$$

G_1 is a rigid grammar. (A fortiori, G_1 is a k-valued grammar for any $k \geq 1$.) The grammar G_2 in Example 9 is a 2-valued grammar, but not a rigid grammar.

Let $\mathcal{G}_{k\text{-valued}}$ denote the class of k-valued grammars, and let $\mathcal{L}_{k\text{-valued}} = \{ \mathrm{L}(G) \mid G \in \mathcal{G}_{k\text{-valued}} \}$. The classes $\mathcal{L}_{k\text{-valued}}$ form a strict hierarchy in the following sense:**

THEOREM 11 (Hierarchy Theorem). *For each* $k \in \mathbb{N}$, $\mathcal{L}_{k\text{-valued}} \subset \mathcal{L}_{k+1\text{-valued}}$.

We refer the reader to Kanazawa 1994a for proof.

The main result of this paper is the following theorem:

THEOREM 12 (Main Theorem). *For each* $k \in \mathbb{N}$, $\mathcal{G}_{k\text{-valued}}$ *is learnable.*

This is not a trivial result, because $\mathcal{L}_{k\text{-valued}}$ is always infinite if $k \geq 2$, and so is $\mathcal{L}_{1\text{-valued}}$ if $|\Sigma| \geq 2$.‡ The main theorem follows from the following lemma, but we will also present a concrete learning algorithm that learns $\mathcal{G}_{k\text{-valued}}$.

LEMMA 13. *For each* $k \in \mathbb{N}$, $\mathcal{L}_{k\text{-valued}}$ *has finite elasticity.*

We prove this lemma using a general theorem on finite elasticity. It is also possible to reduce it to Shinohara's result (Appendix 7), but our method of proof will be of independent interest.

---

* The term is due to Buszkowski.

** In this paper, we always use $\subset$ to mean *proper subset.*

‡ It is easy to see that any class of grammars that generate only finitely many languages is learnable.

3. Buszkowski's Algorithm

The question of learnability has not been addressed with respect to categorial grammars. There is very important work by Buszkowski (1987a, 1987b), however, in which he presents a simple intuitive algorithm that computes a classical categorial grammar from linguistic data. The input to the algorithm is a finite set of *functor-argument structures*, and the output is a rigid classical categorial grammar which is compatible with the input data (if there is one). The algorithm, which is similar to the one for the typing problem in lambda and combinatory calculi (see Barendregt 1992), is an interesting application of unification, and it enjoys some nice formal properties, as proved by Buszkowski and Penn (1990). Although Buszkowski does not consider the question of learnability, his algorithm can in fact be used in an algorithm that learns $\mathcal{G}_{1\text{-valued}}$.

3.1. Definitions of Basic Notions

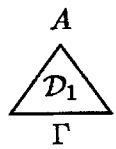
Before presenting Buszkowski's algorithm, we have to introduce some more basic definitions about categorial grammar.

3.1.1. *Derivations*

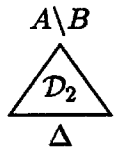
A *derivation* of a type B from a sequence of types $A_1, \ldots, A_n$ is a certain kind of binary branching tree that encodes a proof of $A_1, \ldots, A_n \Rightarrow B$. Each node of a derivation is labeled by a type, and each internal node has an additional label, which is either BA or FA. If $\mathcal{D}$ is a derivation of B from $A_1, \ldots, A_n$, the root node of $\mathcal{D}$ is labeled by B, and its leaf nodes are labeled by $A_1, \ldots, A_n$ in this order from left to right. The labels BA and FA stand for Backward Application and Forward Application, respectively, and indicate which of the two rules is used in each step of a derivation. The set of derivations is defined inductively as follows:

DEFINITION 14.

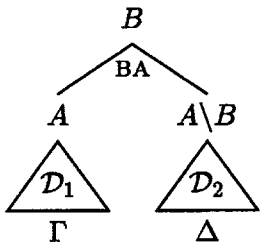
- If $A \in \mathrm{Tp}$, then A (the tree consisting of a single node labeled by A) is a derivation of A from A.
- **Backward Application.** If



is a derivation of A from Γ and

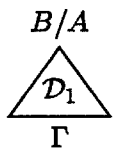


is a derivation of $A\backslash B$ from Δ, then

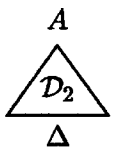


is a derivation of B from Γ, Δ.

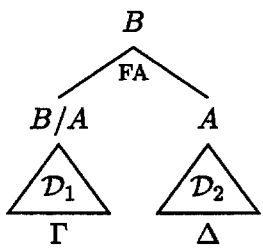
– **Forward Application.** If



is a derivation of B/A from Γ and

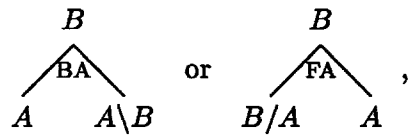


is a derivation of A from Δ, then



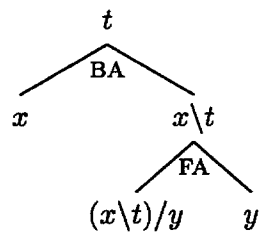
is a derivation of B from Γ, Δ.

In an instance of Backward or Forward Application



the node labeled by $A\backslash B$ or B/A is called the *functor*, and the node labeled by A is called the *argument*. The *ultimate functor* of a derivation is the leaf node that you arrive at by tracing the functor daughters starting from the root node.

EXAMPLE 15. The following is a derivation of t from x, $(x\backslash t)/y$, y:



In the only instance of Backward Application in this derivation, the node labeled by $x\backslash t$ is the functor and the node labeled by x is the argument. In the only instance of Forward Application, the node labeled by $(x\backslash t)/y$ is the functor, and the node labeled by y is the argument. The ultimate functor of this derivation is the node labeled by $(x\backslash t)/y$.

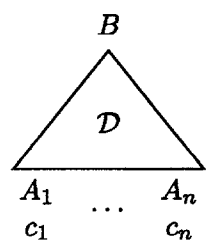
Clearly, we have $\Gamma \Rightarrow A$ if and only if there is a derivation of A from Γ.

3.1.2. *Parse Trees*

Just as a derivation encodes a proof of $A_1, \ldots, A_n \Rightarrow B$, a *parse tree* of G encodes a proof of $c_1 \ldots c_n \in \mathrm{L}(G)$. A parse tree of G is obtained from a derivation of t by attaching symbols to the leaf nodes in accordance with G's type assignments.

DEFINITION 16.

(i) If $\mathcal{D}$ is a derivation of B from $A_1, \ldots, A_n$, and $c_1, \ldots, c_n$ are symbols such that $G\colon c_i \mapsto A_i$ $(1 \leq i \leq n)$, the result of attaching $c_1, \ldots, c_n$, from left to right in this order, to the leaf nodes of $\mathcal{D}$ is a *partial parse tree* of G.



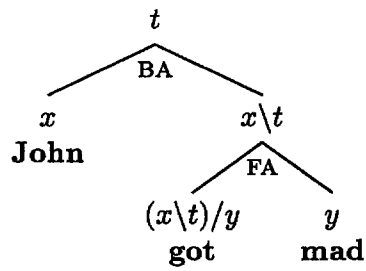
(ii) A *parse tree* of G is a partial parse tree of G whose root node is labeled by t.

If $c_1 \ldots c_n$ is the string of symbols attached to the leaf nodes of a partial parse tree $\mathcal{P}$, $c_1 \ldots c_n$ is said to be the *yield* of $\mathcal{P}$. Clearly, for every string $c_1 \ldots c_n \in \Sigma^+$, $c_1 \ldots c_n \in \mathrm{L}(G)$ if and only if there is a parse tree of G that yields $c_1 \ldots c_n$. If a parse tree $\mathcal{P}$ of G yields $c_1 \ldots c_n$, then $\mathcal{P}$ is called a *parse* of $c_1 \ldots c_n$ (in G).

EXAMPLE 17. Let $\{\mathbf{got}, \mathbf{John}, \mathbf{mad}\} \subseteq \Sigma$, and let

$$\{\langle \mathbf{got}, (x\backslash t)/y\rangle, \langle \mathbf{John}, x\rangle, \langle \mathbf{mad}, y\rangle\} \subseteq G.$$

Then the following is a parse tree of G:

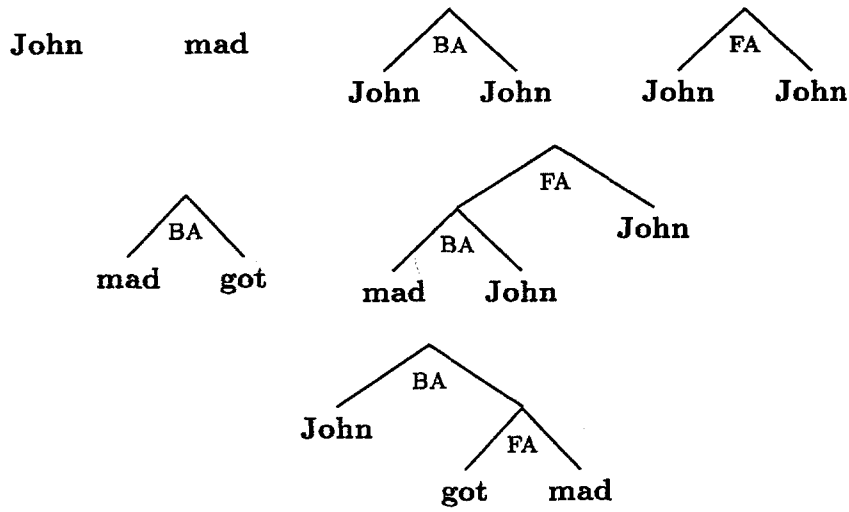


The yield of this parse tree is **John got mad**. Thus, **John got mad** $\in \mathrm{L}(G)$.

3.1.3. *Functor-Argument Structures and Structure Languages*

A *functor-argument structure* over alphabet Σ is a binary-branching tree whose leaf nodes are labeled by symbols in Σ and whose internal nodes are labeled by either BA or FA. As a special case, symbols in Σ are regarded as functor-argument structures of height 0. The set of functor-argument structures over Σ is denoted Σ^{F}. The letters T, U, possibly with subscripts, range over functor-argument structures. Often, we will simply say 'structure' to mean functor-argument structure. A set of functor-argument structures over Σ is called a *structure language* over Σ.

EXAMPLE 18. Let $\{\mathbf{got}, \mathbf{John}, \mathbf{mad}\} \subseteq \Sigma$. Then the following are examples of functor-argument structures over Σ:



The notions of *functor*, *argument*, and *ultimate functor* are defined for parse trees and functor-argument structures in the same way as for derivation trees. Take for example the second to last functor-argument structure in Example 18. In the only instance of Forward Application, the node labeled by BA is the functor, and the other node (labeled by **John**) is the argument. The ultimate functor of this structure is the first leaf node labeled by **John**.

Let G be a grammar, and let $\mathcal{P}$ be a partial parse tree of G. The result of stripping $\mathcal{P}$ of its type labels is a functor-argument structure, and we call this the *(functor-argument) structure* of $\mathcal{P}$. If T is the structure of a parse tree $\mathcal{P}$, we say that $\mathcal{P}$ is a *parse* of T. We say that a grammar G *generates* a structure T if and only if for some parse tree $\mathcal{P}$ of G, T is the structure of $\mathcal{P}$. The set of structures generated by G is called the *structure language* of G and is denoted $\mathrm{FL}(G)$.* In order to distinguish $\mathrm{L}(G)$, the language of G, from $\mathrm{FL}(G)$, its structure language, we often call the former the *string language* of G.

EXAMPLE 19. Let G be as in Example 17, and let T be the last structure in Example 18. Then the parse tree in Example 17 is a parse of T. Thus, $T \in \mathrm{FL}(G)$.

The *yield* of a functor-argument structure T is the string of symbols $c_1 \ldots c_n$ labeling the leaf nodes of T, from left to right in this order. The yield of T is denoted $\mathrm{yield}(T)$. It is easy to see that $\mathrm{L}(G) = \{\, \mathrm{yield}(T) \mid T \in \mathrm{FL}(G) \,\}$.

Since functor-argument structures over Σ are labeled trees, we can represent them as terms, by regarding BA and FA as function symbols, and symbols in Σ as constants. Thus, the structures in Example 18 can be represented as follows:

* The 'F' in $\mathrm{FL}(G)$ stands for 'functor-argument structures'. The notation is from Buszkowski 1988.

$$\mathbf{John},\ \mathbf{mad},\ \text{BA}(\mathbf{John},\mathbf{John}),\ \text{FA}(\mathbf{John},\mathbf{John}),$$
$$\text{BA}(\mathbf{mad},\mathbf{got}),\ \text{FA}(\text{BA}(\mathbf{mad},\mathbf{John}),\mathbf{John}),\ \text{BA}(\mathbf{John},\text{FA}(\mathbf{got},\mathbf{mad})).$$

Since parentheses and commas are not strictly necessary, functor-argument structures over Σ can be encoded as strings over $\Sigma \cup \{\text{BA}, \text{FA}\}$. Viewed this way, Σ^{F}, the set of functor-argument structures over Σ, is a context-free language over $\Sigma \cup \{\text{BA}, \text{FA}\}$. Note that the triple $\langle \text{CatG}, \Sigma^{\text{F}}, \text{FL}\rangle$ constitutes another grammar system.*

We let $\mathcal{FL}_{k\text{-valued}}$ denote $\{\, \text{FL}(G) \mid G \in \mathcal{G}_{k\text{-valued}} \,\}$. The elements of $\mathcal{FL}_{k\text{-valued}}$ are called *k-valued structure languages*, and the elements of $\mathcal{FL}_{1\text{-valued}}$ are called *rigid structure languages*. By contrast, we call the elements of $\{\, \text{L}(G) \mid G \in \mathcal{G}_{k\text{-valued}} \,\}$ *k-valued string languages* and the elements of $\{\, \text{L}(G) \mid G \in \mathcal{G}_{1\text{-valued}} \,\}$ *rigid string languages*.

3.1.4. *Substitutions*

Recall that $\text{Pr} = \{t\} \cup \text{Var}$. If we regard t as a constant, types can be regarded as terms in a first-order language where $\backslash$ and $/$ act as function symbols. Then, the standard notion of substitution of a term for a variable applies straightforwardly to types.

A *substitution* is a function $\sigma\colon \text{Var} \to \text{Tp}$ that maps variables to types. A substitution is extended to a function from types to types as follows:

DEFINITION 20. Let σ be a substitution. Then we set

$$\begin{aligned} \sigma(t) &= t, \\ \sigma(A\backslash B) &= \sigma(A)\backslash\sigma(B), \\ \sigma(B/A) &= \sigma(B)/\sigma(A), \end{aligned}$$

for all $A, B \in \text{Tp}$.**

We use the notation $\{x_1 \mapsto A_1, \ldots, x_n \mapsto A_n\}$ to denote the substitution σ such that $\sigma(x_1) = A_1, \ldots, \sigma(x_n) = A_n$ and $\sigma(y) = y$ for all other variables y.

EXAMPLE 21. Let $\sigma = \{x \mapsto x\backslash y,\ y \mapsto t,\ z \mapsto t/(t/x)\}$. Then $\sigma\Big((t/x)\backslash y\Big) = (t/(x\backslash y))\backslash t$ and $\sigma\Big(((t/x)\backslash y)/(x/z)\Big) = ((t/(x\backslash y))\backslash t)/((x\backslash y)/(t/(t/x)))$.

Next we extend the action of substitutions to grammars:

* An important fact about the structure languages of classical categorial grammars is that the inclusion problem '$\text{FL}(G_1) \subseteq \text{FL}(G_2)$?' is decidable. See Buszkowski 1987a.

** We use the prefix notation $\sigma(A)$ instead of the more common postfix notation $A\sigma$ for substitutions.

DEFINITION 22. Let σ be a substitution. Then $\sigma[G]$ denotes the grammar obtained by applying σ in the type assignments of G, that is:

$$\sigma[G] = \{ \langle c, \sigma(A) \rangle \mid \langle c, A \rangle \in G \}.$$

$\sigma[G]$ is called a *substitution instance* of G.

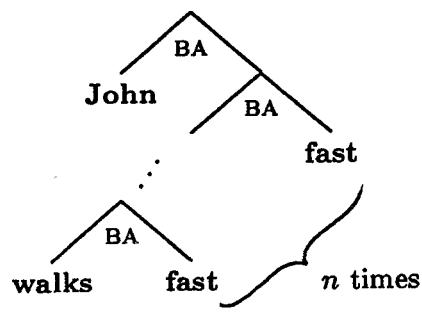
EXAMPLE 23. Let $\{\mathbf{fast}, \mathbf{John}, \mathbf{walks}\} \subseteq \Sigma$, and let

$$\begin{aligned} G_1\text{:}\quad \mathbf{fast} &\mapsto y\backslash(x\backslash t), \\ \mathbf{John} &\mapsto x, \\ \mathbf{walks} &\mapsto x\backslash t,\ y. \end{aligned}$$

and

$$\begin{aligned} G_2\text{:}\quad \mathbf{fast} &\mapsto (x\backslash t)\backslash(x\backslash t), \\ \mathbf{John} &\mapsto x, \\ \mathbf{walks} &\mapsto x\backslash t. \end{aligned}$$

Then $G_2 = \sigma[G_1]$, where $\sigma = \{y \mapsto x\backslash t\}$. Note that $\mathrm{FL}(G_1) = \{T_0, T_1\} \subset \{ T_n \mid n \in \mathbb{N} \} = \mathrm{FL}(G_2)$, where T_n is the following functor-argument structure:



If G_1 and G_2 are grammars, $G_1 \subseteq G_2$ expresses the fact that G_2 contains all type assignments of G_1, and possibly more. The following is a straightforward but important fact about substitution instances.

PROPOSITION 24 (Buszkowski and Penn). *If* $\sigma[G_1] \subseteq G_2$, *then* $\mathrm{FL}(G_1) \subseteq \mathrm{FL}(G_2)$.

Proof. Suppose $\sigma[G_1] \subseteq G_2$. Let $T \in \mathrm{FL}(G_1)$ and let $\mathcal{P}$ be a parse of T in G_1. Let $\sigma[\mathcal{P}]$ be the result of replacing each type label A of $\mathcal{P}$ by $\sigma(A)$. Then it is easy to see that $\sigma[\mathcal{P}]$ is a parse of T in G_2. Therefore, $T \in \mathrm{FL}(G_2)$.

Proposition 24 implies that if $\sigma[G_1] \subseteq G_2$, then $\mathrm{L}(G_1) \subseteq \mathrm{L}(G_2)$.

A substitution that is a one-to-one function from Var to Var is called a *variable renaming*. If σ is a variable renaming, then G and $\sigma[G]$ are called *alphabetic variants*. Clearly, grammars that are alphabetic variants have exactly the same shape and are identical for all intents and purposes. Therefore, it is convenient to adopt the following convention:

Convention Grammars that are alphabetic variants are treated as identical.

In other words, different finite relations between Σ and Tp that are alphabetic variants are considered different 'representations' of one and the same grammar.

Note that range(G) is the set of types assigned to some symbol by G. Let $\mathrm{Tp}(G) = \{ A \mid A$ is a subtypoe of some $B \in \mathrm{range}(G) \}$, and let $\mathrm{Var}(G) = \mathrm{Tp}(G) \cap \mathrm{Var}$. Var($G$) is the set of variables used in G.

PROPOSITION 25. *Suppose* $\sigma_1[G_1] = G_2$ *and* $\sigma_2[G_2] = G_1$. *Then* G_1 *and* G_2 *are alphabetic variants and are thus equal.*

Proof. First, note that for each symbol $c \in \Sigma$, σ_1 and σ_2 provide a one-one correspondence between $\{ A \mid G_1{:}c \mapsto A \}$ and $\{ A \mid G_2{:}c \mapsto A \}$. For, if $\{ \sigma_1(A) \mid G_1{:}c \mapsto A \} \subset \{ A \mid G_2{:}c \mapsto A \}$, $\sigma_2[\sigma_1[G_1]]$ cannot be equal to G_1, and likewise for σ_2. Then, it is easy to see that $\sigma_1 \restriction \mathrm{Var}(G_1)$ is a one-to-one function from Var(G_1) onto Var(G_2), and $\sigma_2 \restriction \mathrm{Var}(G_2) = (\sigma_1 \restriction \mathrm{Var}(G_1))^{-1}$. One can extend $\sigma_1 \restriction \mathrm{Var}(G_1)$ to a variable renaming σ. Then $\sigma[G_1] = \sigma_1[G_1] = G_2$.

3.1.5. *Most General Unifiers*

Let σ and τ be substitutions. The *composition* of σ with τ, denoted $\tau \circ \sigma$, is defined in the usual way: for all variables x, $\tau \circ \sigma(x) = \tau(\sigma(x))$. The composition of two substitutions acts as it should on all types: for all types A, $\tau \circ \sigma(A) = \tau(\sigma(A))$. The action of the composition of two substitutions on grammars is also as is expected: for all grammars G, $\tau \circ \sigma[G] = \tau[\sigma[G]]$.

Let σ_1 and σ_2 be substitutions. σ_1 is said to be *more general* than σ_2 if there is a substitution τ such that $\sigma_2 = \tau \circ \sigma_1$.

EXAMPLE 26. Let $\sigma_1 = \{x \mapsto x\backslash y,\ y \mapsto t,\ z \mapsto t/(t/x)\}$ and let $\sigma_2 = \{x \mapsto t\backslash y,\ y \mapsto t,\ z \mapsto t/(t/t)\}$. Then $\sigma_2 = \tau \circ \sigma_1$, where $\tau = \{x \mapsto t\}$. Thus σ_1 is more general than σ_2.

A substitution σ is said to *unify* a set $\mathbf{A}$ of types if for all $A_1, A_2 \in \mathbf{A}$, $\sigma(A_1) = \sigma(A_2)$. We say that σ unifies a family of sets of types, if σ unifies each set in the family. A substitution σ is a *most general unifier* of a family $\mathcal{A}$ of sets of types if and only if σ is a unifier of $\mathcal{A}$ and for every unifier σ' of $\mathcal{A}$, σ is more general than σ'.

EXAMPLE 27. Let $\mathcal{A}$ consist of the following sets:

$$A_1 = \{x_1/x_2, x_3/x_4\},$$
$$A_2 = \{x_5\backslash(x_3\backslash t)\},$$
$$A_3 = \{x_1\backslash t, x_5\}.$$

Then a most general unifier of $\mathcal{A}$ is:

$$\sigma = \{x_3 \mapsto x_1,\ x_4 \mapsto x_2,\ x_5 \mapsto x_1\backslash t\}.$$

A most general unifier is unique up to 'renaming of variables' in some suitable sense.*

DEFINITION 28. Let us fix a computable partial function mgu that maps a finite family $\mathcal{A}$ of finite sets of types to a most general unifier $\mathrm{mgu}(\mathcal{A})$ of $\mathcal{A}$, if there is one.

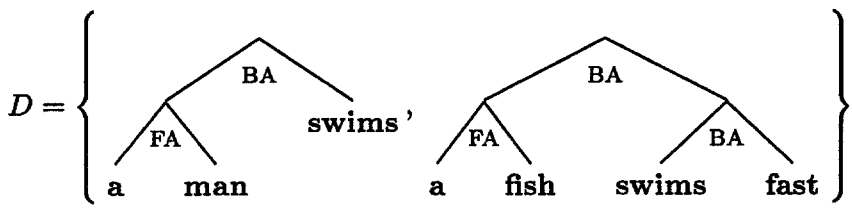
Since most general unifiers are unique up to renaming of variables, and we identify grammars that are alphabetic variants, it does not matter for our purposes which most general unifier mgu picks.

3.2. ALGORITHM RG

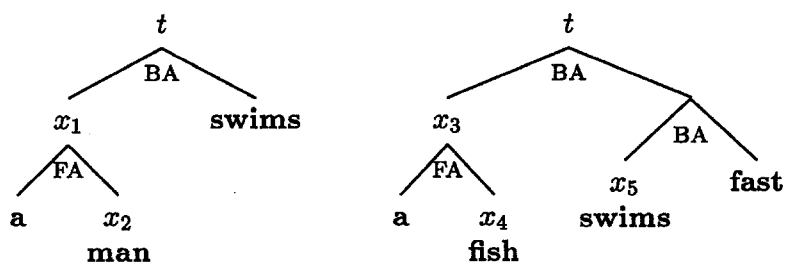
Buszkowski's (1987a, 1987b) algorithm, which we here call RG (for 'rigid grammars'), takes a finite set of functor-argument structures as input and returns a rigid grammar compatible with it as output, if there is one. This algorithm essentially relies on unification of a family of sets of types.

ALGORITHM RG.
- **input:** a finite set D of functor-argument structures.
- **output:** a rigid grammar G such that $D \subseteq \mathrm{FL}(G)$ (if there is one).

We illustrate the algorithm using the following example:

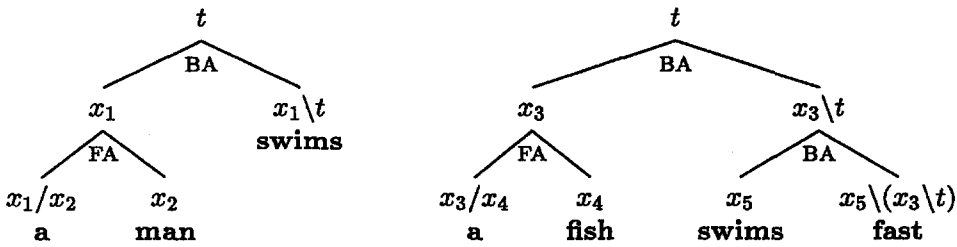


Step 1. Assign a type to each node of the structures in D as follows:
(a) Assign t to each root node.
(b) Assign distinct variables to the argument nodes.

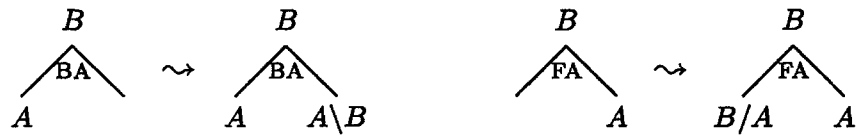


* See Lassez, Maher, and Marriott (1988) for discussions of the subtlety involved in the notion of most general unifier.

(c) Compute types for the functor nodes.



The general rules here are the following:



Step 2. Collect the types assigned to the leaf nodes into a grammar.

$$\begin{aligned}\mathrm{GF}(D)\colon \quad \mathbf{a} &\mapsto x_1/x_2,\ x_3/x_4,\\ \mathbf{fast} &\mapsto x_5\backslash(x_3\backslash t),\\ \mathbf{fish} &\mapsto x_4,\\ \mathbf{man} &\mapsto x_2,\\ \mathbf{swims} &\mapsto x_1\backslash t,\ x_5.\end{aligned}$$

This is the *general form* determined by D. In general, $\mathrm{GF}(D)\colon c \mapsto A$ if and only if the previous step assigns A to a leaf node labeled by c.

Step 3. Unify the types assigned to the same symbol. Let $\mathcal{A} = \{\, \{\, A \mid \mathrm{GF}(D)\colon c \mapsto A \,\} \mid c \in \mathrm{dom}(\mathrm{GF}(D)) \,\}$, and compute $\sigma = \mathrm{mgu}(\mathcal{A})$.

$$\sigma = \{x_3 \mapsto x_1,\ x_4 \mapsto x_2,\ x_5 \mapsto x_1\backslash t\}.$$

The algorithm fails if unification fails.

Step 4. Let $\mathrm{RG}(D) = \sigma[\mathrm{GF}(D)]$.

$$\begin{aligned}\mathrm{RG}(D)\colon \quad \mathbf{a} &\mapsto x_1/x_2,\\ \mathbf{fast} &\mapsto (x_1\backslash t)\backslash(x_1\backslash t),\\ \mathbf{fish} &\mapsto x_2,\\ \mathbf{man} &\mapsto x_2,\\ \mathbf{swims} &\mapsto x_1\backslash t.\end{aligned}$$

(Note the change in the types assigned to **fast** and **fish**.) This is the output of the algorithm.

3.3. PROPERTIES OF RG AND SOME CONSEQUENCES

Buszkowski and Penn (1990) noted some of the most important properties of the algorithm RG. In this section, we draw some consequences from Buszkowski and Penn's results. In later sections, we will use them to show that $\mathcal{FL}_{1\text{-valued}}$ has finite elasticity, and eventually, to show that $\mathcal{G}_{1\text{-valued}}$ is learnable.

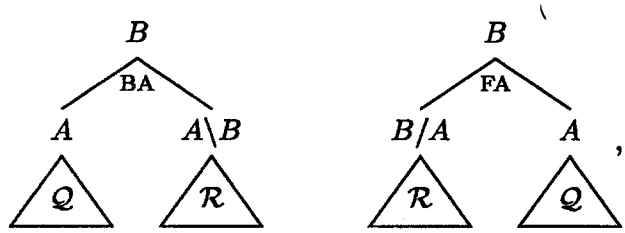
LEMMA 29 (Buszkowski and Penn). $\mathrm{FL}(\mathrm{GF}(D)) = D$.

Proof. Let $D = \{T_1, \ldots, T_n\}$. The labeling of the nodes of the structures in D that precedes the construction of $\mathrm{GF}(D)$ in fact forms a parse tree $\mathcal{P}_i$ of $\mathrm{GF}(D)$ for each structure T_i in D. This shows $D \subseteq \mathrm{FL}(\mathrm{GF}(D))$.

To show $\mathrm{FL}(\mathrm{GF}(D)) \subseteq D$, we prove that each partial parse tree $\mathcal{P}$ of $\mathrm{GF}(D)$ appears as a subtree in some $\mathcal{P}_i$. This is done by induction on the height h of $\mathcal{P}$.

INDUCTION BASIS. $h = 0$. $\mathcal{P}$ consists of just one node, labeled by symbol c and type A such that $\mathrm{GF}(D)\colon c \mapsto A$. Then, by the definition of $\mathrm{GF}(D)$, $\mathcal{P}$ appears as a subtree in some $\mathcal{P}_i$.

INDUCTION STEP. Let $\mathcal{P}$ be a partial parse tree of $\mathrm{GF}(D)$ of height $h > 0$. Then $\mathcal{P}$ must look like one of the following:



where $\mathcal{Q}$ and $\mathcal{R}$ are partial parse trees of $\mathrm{GF}(D)$ of height $\leq h - 1$. By induction hypothesis, $\mathcal{Q}$ appears as a subtree in some $\mathcal{P}_j$, and $\mathcal{R}$ appears as a subtree in some $\mathcal{P}_k$. By the construction, A must be a variable, and there is just one node in $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ labeled by A. Then there must be just one node in $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ labeled by $A\backslash B$ or B/A as well. These nodes, which are the root nodes of $\mathcal{Q}$ and $\mathcal{R}$, must be the functor node and the argument node, respectively, in some instance of Backward or Forward Application in $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$. Therefore, $\mathcal{P}$ occurs as a subtree in some $\mathcal{P}_i$.

LEMMA 30 (Buszkowski and Penn). *Let D be a finite set of functor-argument structures. Then, for any grammar G, the following are equivalent:*

(i) $D \subseteq \mathrm{FL}(G)$.

(ii) *There is a substitution σ such that $\sigma[\mathrm{GF}(D)] \subseteq G$.*

Proof. (ii) $\Rightarrow$ (i) follows from Lemma 29.

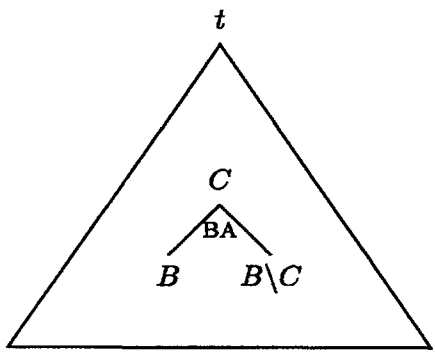
(i) $\Rightarrow$ (ii). Let $D = \{T_1, \ldots, T_n\}$ and let $\mathcal{P}_i$ be $\mathrm{GF}(D)$'s parse of T_i $(1 \leq i \leq n)$. Assume $D \subseteq \mathrm{FL}(G)$. Then G has a parse $\mathcal{Q}_i$ of each T_i. Define a substitution σ as follows. For each variable $x \in \mathrm{Var}(\mathrm{GF}(D))$, find a (unique) $\mathcal{P}_i$ that contains a

(unique) node labeled by x, and let $\sigma(x)$ be the type labeling the corresponding node of $\mathcal{Q}_i$. By induction on $A \in \mathrm{Tp}(\mathrm{GF}(D))$, we show that

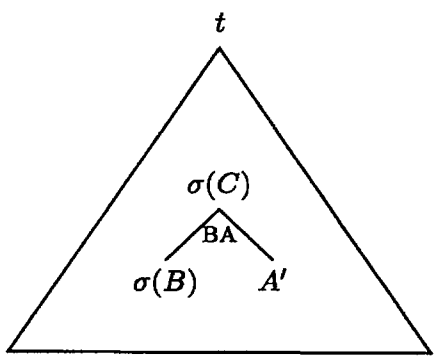
if A labels a node of some $\mathcal{P}_i$, then $\sigma(A)$ labels the corresponding node of $\mathcal{Q}_i$.

INDUCTION BASIS. If $A \in \mathrm{Var}$, this holds by definition. If $A = t$, then any node labeled by A in $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ is the root node of some $\mathcal{P}_i$. Since $\mathcal{Q}_i$ is a parse tree of G, the root node of $\mathcal{Q}_i$ must be labeled by t.

INDUCTION STEP. Let $A = B\backslash C$ labels a node of $\mathcal{P}_i$. Then the relevant part of $\mathcal{P}_i$ must look like the following:



By induction hypothesis, the corresponding part of $\mathcal{Q}_i$ looks like:



Then $A' = \sigma(B)\backslash\sigma(C) = \sigma(B\backslash C) = \sigma(A)$. The case $A = C/B$ is entirely similar, completing the induction.

It follows that if $\mathrm{GF}(D)$: $c \mapsto A$, then G: $c \mapsto \sigma(A)$. Therefore, $\sigma[\mathrm{GF}(D)] \subseteq G$.

PROPOSITION 31 (Buszkowski and Penn). *Let D be a finite set of functor-argument structures. Then, for any rigid grammar G, the following are equivalent:*
(i) $D \subseteq \mathrm{FL}(G)$.
(ii) $\mathrm{RG}(D)$ *exists and there is a substitution* τ *such that* $\tau[\mathrm{RG}(D)] \subseteq G$.

Proof. (ii) $\Rightarrow$ (i) follows from Lemma 30 and the fact that $\mathrm{RG}(D)$ is a substitution instance of $\mathrm{GF}(D)$.

(i) $\Rightarrow$ (ii). Assume that G is a rigid grammar such that $D \subseteq \mathrm{FL}(G)$. By Lemma 30, there is a substitution σ such that $\sigma[\mathrm{GF}(D)] \subseteq G$. Since G is a rigid grammar, $\sigma[\mathrm{GF}(D)]$ is also a rigid grammar. Then σ unifies the family $\mathcal{A} = \{ \{ A \mid \mathrm{GF}(D){:}c \mapsto A \} \mid c \in \mathrm{dom}(\mathrm{GF}(D)) \}$. This means that $\mathrm{RG}(D)$ exists and $\mathrm{RG}(D) = \sigma_0[\mathrm{GF}(D)]$, where $\sigma_0 = \mathrm{mgu}(\mathcal{A})$. Then there is a substitution τ such that $\sigma = \tau \circ \sigma_0$. Therefore, $\tau[\mathrm{RG}(D)] = \tau[\sigma_0[\mathrm{GF}(D)]] = (\tau \circ \sigma_0)[\mathrm{GF}(D)] = \sigma[\mathrm{GF}(D)]$. By assumption, $\sigma[\mathrm{GF}(D)] \subseteq G$, so $\tau[\mathrm{RG}(D)] \subseteq G$.

COROLLARY 32. *For every finite set D of functor-argument structures, $\{ L \in \mathcal{FL}_{\text{1-valued}} \mid D \subseteq L \}$, if non-empty, has a least element with respect to the ordering $\subseteq$.*

Proof. This is an easy consequence of Propositions 24 and 31. The least element of $\{ L \in \mathcal{FL}_{\text{1-valued}} \mid D \subseteq L \}$ is given by $\mathrm{FL}(\mathrm{RG}(D))$.

When G_1 and G_2 are rigid grammars, we write $G_1 \sqsubseteq G_2$ to mean there is a substitution σ such that $\sigma[G_1] \subseteq G_2$. Since we identify grammars that are alphabetic variants, it follows from Proposition 25 that $\sqsubseteq$ is a partial order. We write $G_1 \sqsubset G_2$ to mean $G_1 \sqsubseteq G_2$ and $G_2 \not\sqsubseteq G_1$. It is easy to see that for any rigid grammar G, $\{ G' \in \mathcal{G}_{\text{1-valued}} \mid G' \sqsubseteq G \}$ is finite.

The following is immediate from Proposition 31:

COROLLARY 33. *Let D_1 and D_2 be two finite sets of functor-argument structures such that $D_1 \subseteq D_2$. If $\mathrm{RG}(D_2)$ exists, $\mathrm{RG}(D_1)$ also exists and $\mathrm{RG}(D_1) \sqsubseteq \mathrm{RG}(D_2)$.*

DEFINITION 34. A rigid grammar G is said to be in *reduced form* if there is no grammar G' such that $G' \sqsubset G$ and $\mathrm{FL}(G') = \mathrm{FL}(G)$.

EXAMPLE 35. The following rigid grammar G is not in reduced form:

$$G\colon\ \begin{array}{l} \mathbf{mary} \mapsto t/x, \\ \mathbf{swims} \mapsto (t/x)\backslash t. \end{array}$$

An equivalent grammar in reduced form is

$$G'\colon\ \begin{array}{l} \mathbf{mary} \mapsto y, \\ \mathbf{swims} \mapsto y\backslash t. \end{array}$$

We have $\mathrm{FL}(G') = \mathrm{FL}(G)$ and $G' \sqsubset G$.

It should be clear that for any rigid grammar, there is one in reduced form that generates the same structure language.

COROLLARY 36. $\mathrm{RG}(D)$, *if it exists, is in reduced form.*

Proof. Immediate from Proposition 31 and the definition of reduced form.

We also have the converse of Corollary 36:*

PROPOSITION 37. *For every rigid grammar G in reduced form, there is a finite set D_G of functor-argument structures such that $G = \mathrm{RG}(D_G)$.*

Proof. D_G can be found by the following algorithm. Initially, set $D := \emptyset$. While $\mathrm{FL}(G) - \mathrm{FL}(\mathrm{RG}(D)) \neq \emptyset$, pick a functor-argument structure $T \in \mathrm{FL}(G) - \mathrm{FL}(\mathrm{RG}(D))$ and set $D := D \cup \{T\}$. The value of D when the algorithm terminates is the desired D_G.

The correctness of this algorithm can be seen as follows. Firstly, D is always a subset of $\mathrm{FL}(G)$, so by Proposition 31, $\mathrm{RG}(D)$ always exists and $\mathrm{RG}(D) \sqsubseteq G$ always holds. If we let G_i be the grammar $\mathrm{RG}(D)$ constructed at the ith stage of the algorithm, by Corollary 33, we have

$$G_0 \sqsubset G_1 \sqsubset G_2 \sqsubset \cdots \sqsubseteq G.$$

Since $\{ G' \in \mathcal{G}_{\text{1-valued}} \mid G' \sqsubseteq G \}$ is finite and a $\sqsubset$-chain cannot contain a cycle, the algorithm must terminate at some nth stage. Then we have $G_n \sqsubseteq G$ and $\mathrm{FL}(G) - \mathrm{FL}(G_n) = \emptyset$. Since $G_n \sqsubseteq G$ implies $\mathrm{FL}(G_n) \subseteq \mathrm{FL}(G)$, we get $\mathrm{FL}(G) = \mathrm{FL}(G_n)$. By the assumption that G is in reduced form, $G_n \not\sqsubset G$. Therefore, $G_n = G$.

Proposition 37 can be used to show that if we take $\langle \mathrm{CatG}, \Sigma^{\mathrm{F}}, \mathrm{FL} \rangle$ to be the grammar system instead of the standard $\langle \mathrm{CatG}, \Sigma^{+}, \mathrm{L} \rangle$, Buszkowski's algorithm essentially learns the class $\mathcal{G}_{\text{1-valued}}$.** More results of this kind about 'learning from structures', as opposed to 'learning from strings', which is our concern here, may be found in Kanazawa 1994a.

PROPOSITION 38. *Let G_1 be a rigid grammar in reduced form, and let G_2 be any rigid grammar. Then $G_1 \sqsubseteq G_2$ if and only if $\mathrm{FL}(G_1) \subseteq \mathrm{FL}(G_2)$.*

Proof. The 'only if' direction is given by Proposition 24. To prove the 'if' direction, let D_{G_1} be the finite subset of $\mathrm{FL}(G_1)$ such that $\mathrm{RG}(D_{G_1}) = G_1$, as given by Proposition 37. If $\mathrm{FL}(G_1) \subseteq \mathrm{FL}(G_2)$, then $D_{G_1} \subseteq \mathrm{FL}(G_2)$, so by Proposition 31, $\mathrm{RG}(D_{G_1}) \sqsubseteq G_2$.

4. Finite Elasticity of the k-Valued Languages

In this section, we prove our main theorem

* The acute reader will find that the next few results, although they are used to prove the main theorem of this paper, are not strictly necessary for that purpose. They may be of independent interest, however.

** I say 'essentially', since RG takes finite sets of functor-argument structures as input instead of finite sequences as required of learning algorithms.

THEOREM 12 (Main Theorem). *For each* $k \in \mathbb{N}$, $\mathcal{G}_{k\text{-valued}}$ *is learnable.*

by showing that $\mathcal{L}_{k\text{-valued}}$, the class of k-valued string languages, has finite elasticity. In the next section, we present a concrete learning algorithm that learns the class of k-valued grammars.

4.1. Finite Elasticity of the Rigid Structure Languages

To prove that $\mathcal{L}_{k\text{-valued}}$ has finite elasticity, we first prove that $\mathcal{FL}_{1\text{-valued}}$ has finite elasticity. The finite elasticity of $\mathcal{L}_{k\text{-valued}}$ then follows from it by a general theorem on finite elasticity to be proved in the next subsection.

We call a sequence $L_0, L_1, L_2, \ldots$ of languages an *ascending chain* if $L_0 \subset L_1 \subset L_2 \subset \cdots$. To begin with, we note that to prove the finite elasticity of $\mathcal{FL}_{1\text{-valued}}$, it is enough to show that $\mathcal{FL}_{1\text{-valued}}$ does not contain an infinite ascending chain.

LEMMA 39. *If* $\mathcal{FL}_{1\text{-valued}}$ *has infinite elasticity, then there is an infinite sequence* $\langle L_n \rangle_{n \in \mathbb{N}}$ *of languages in* $\mathcal{FL}_{1\text{-valued}}$ *such that for all* $n \in \mathbb{N}$, $L_n \subset L_{n+1}$.

Proof. Suppose that $\mathcal{FL}_{1\text{-valued}}$ has infinite elasticity, i.e., that there is an infinite sequence $\langle T_n \rangle_{n \in N}$ of functor-argument structures and an infinite sequence $\langle L'_n \rangle_{n' \in \mathbb{N}}$ of languages in $\mathcal{FL}_{1\text{-valued}}$ such that for all $n \in \mathbb{N}$,

$$T_n \notin L'_n$$

and

$$\{T_0, \ldots, T_n\} \subseteq L'_{n+1}.$$

By Corollay 32, let L_n be the least element of $\{ L \in \mathcal{FL}_{1\text{-valued}} \mid \{T_0, \ldots, T_{n-1}\} \subseteq L \}$. Then for all $n \in \mathbb{N}$, $L_n \subseteq L_{n+1}$. Moreover, $L_n \neq L_{n+1}$, since $T_n \notin L_n$ but $T_n \in L_{n+1}$. So for all $n \in \mathbb{N}$, $L_n \subset L_{n+1}$.

Note that the above proof shows that for any class of languages with the property in Corollay 32, infinite elasticity is equivalent to existence of an infinite ascending chain.*

We will prove that there is no infinite $\sqsubset$-chain of rigid grammars in reduced form (based on the same alphabet). Since for every rigid grammar there is a rigid grammar in reduced form that generates the same structure language, from this it follows by Proposition 38 that there is no infinite ascending chain of rigid structure languages.

We need some definitions.

DEFINITION 40. A type $A \in \mathrm{Tp}(G)$ is said to be *useless* in G if there is no parse tree of G that has a node labeled by A.

* Actually, a condition weaker than the one in Corollary 32 suffices, but this need not concern us here.

This definition is analogous to the definition of useless symbols in context-free grammars.

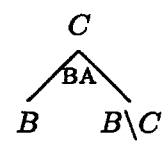
PROPOSITION 41. *If a rigid grammar G is in reduced form, then G has no useless type.*

Proof. We prove the contrapositive. Suppose that G is a rigid grammar with a useless type. Then there is a type $A \in \mathrm{Tp}(G)$ such that no parse tree of G has a node labeled by A.

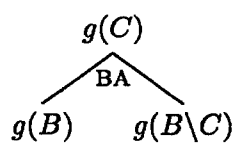
CASE 1. $A \in \mathrm{range}(G)$. Let $G' = G - \{\langle c, A\rangle\}$, where c is a symbol such that $G: c \mapsto A$. $G' \subset G$, and it is clear that $\mathrm{FL}(G') = \mathrm{FL}(G)$, so G is not in reduced form.

CASE 2. $A \notin \mathrm{range}(G)$. Then there is a type $A_1 \in \mathrm{Tp}(G)$ such that A is an immediate subtype of A_1 (i.e., $A_1 = A\backslash A_2$ or $A_2\backslash A$ or A_2/A or A/A_2). Since there is no parse tree of G that has a node labeled by A, there is no parse tree of G that has an instance of Backward or Forward Application in which the functor node is labeled by A_1.

Pick a variable $x \notin \mathrm{Var}(G)$ and let $\sigma(x) = A_1$. For each $B \in \mathrm{Tp}(G)$, let $g(B)$ be the result of replacing all occurrences of A_1 in B by x. This makes $\sigma(g(B)) = B$ for all types $B \in \mathrm{Tp}(G)$. Take the grammar $G' = \{\langle c, g(B)\rangle \mid \langle c, B\rangle \in G\}$. $G = \sigma[G']$, so $G' \sqsubseteq G$. Since σ maps x to a non-variable, $G' \neq G$. Thus, $G' \sqsubset G$. By Proposition 24, $\mathrm{FL}(G') \subseteq \mathrm{FL}(G)$, so it remains to show $\mathrm{FL}(G) \subseteq \mathrm{FL}(G')$. Suppose $T \in \mathrm{FL}(G)$. Let $\mathcal{P}$ be a parse of T in G. Let $\mathcal{P}'$ be the result of replacing each type label B of $\mathcal{P}$ by $g(B)$. We show that $\mathcal{P}'$ is a parse of T in G', to conclude $T \in \mathrm{FL}(G')$. That the root node of $\mathcal{P}'$ is labeled by t is obvious. If a leaf node of $\mathcal{P}'$ has type label B' and symbol label c, then by the construction of $\mathcal{P}'$, $B' = g(B)$ for some B such that $G: c \mapsto B$, which implies that $G': c \mapsto B'$. Now let



be an instance of Backward Application in $\mathcal{P}$. Then $\mathcal{P}'$ has



in the same position. The only way that this fails to be an instance of Backward Application is to have $g(B\backslash C) \neq g(B)\backslash g(C)$. This can be so only if $B\backslash C = A_1$, which is impossible. By symmetry, the same holds for Forward Application. Therefore, $\mathcal{P}'$ must be a parse of T in G'.

This proposition will be useful in showing that there is no infinite ascending chain of grammars without useless types.*

DEFINITION 42. Note that any type A can be written uniquely in the following form:

$$(\ldots(p|A_1)|\ldots)|A_n$$

where $B|C$ stands for either B/C or $C\backslash B$, and $p \in \mathrm{Pr}$. For $0 \leq i \leq n$, we call the subtype $(\ldots(p|A_1)|\ldots)|A_i$ of A (when $i = 0$, we take this to be p) a *head subtype* of A. p is the *head* of A and is denoted $\mathrm{head}(A)$. A_i's are called *argument subtypes* of A.

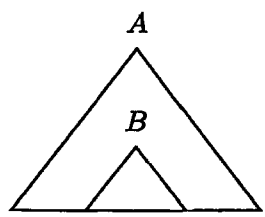
LEMMA 43. *If a type* $A \in \mathrm{Tp}(G)$ *is not useless in* G, *then* A *occurs as a head subtype of some type in* $\mathrm{range}(G)$.

Proof. Assume that A is not useless in G. Then there is a partial parse tree $\mathcal{P}$ whose root node is labeled by A. Then, by induction on the depth of the ultimate functor of $\mathcal{P}$, one can prove that A is a head subtype of the type labeling the ultimate functor of $\mathcal{P}$.

LEMMA 44. *If a variable* $x \in \mathrm{Var}(G)$ *is not useless in* G, *then*
(i) *there is a type* $B \in \mathrm{range}(G)$ *such that* $x = \mathrm{head}(B)$, *and*
(ii) *there is a type* $C \in \mathrm{range}(G)$ *such that* x *occurs as an argument subtype in* C.

Proof. Part (i) is just a special case of Lemma 43. To prove part (ii), let $\mathcal{P}$ be a parse tree of G that has a node labeled by x. Since $x \neq t$, x occurs in $\mathcal{P}$ as the label of the argument node of an instance of Backward or Forward Application. Then the accompanying functor node is labeled by a type B of the form $x\backslash A$ or A/x. Since B is not useless, by Lemma 43, there is a type $C \in \mathrm{range}(G)$ where B occurs as a head subtype. Then x occurs as an argument subtype in C.

DEFINITION 45. Let G be a grammar and let $A, B \in \mathrm{Tp}(G)$. We say that A *depends on* B (in G) if every partial parse tree $\mathcal{P}$ whose root node is labeled by A has a non-root node labeled by B.



* The converse of Proposition 41 also holds: if a rigid grammar has no useless type, it is in reduced form (Kanazawa 1994a).

LEMMA 46. *Let* $A, B \in \mathrm{Tp}(G)$ *and suppose that in every type* $C \in \mathrm{Tp}(G)$ *that has* A *as a head subtype,* B *occurs as an argument subtype outside* A.

$$C = A|\ldots|B|\ldots$$

Then A *depends on* B *in* G.

Proof. Let $\mathcal{P}$ be a partial parse tree of G whose root node is labeled by A. Then the ultimate functor of $\mathcal{P}$ is some type C of the form $(\ldots(A|B_1)|\ldots)|B_m$. By assumption, some $B_i = B$. Then somewhere on the path from the root to the ultimate functor of $\mathcal{P}$, there must be an instance of Backward or Forward Application where the argument node is labeled by B.

DEFINITION 47. For any grammar G, let $\mathrm{Head}(G)$ denote the set $\{\,\mathrm{head}(A) \mid A \in \mathrm{range}(G)\,\}$.

DEFINITION 48. Define the *degree* $d(G)$ of a rigid grammar G as follows:

$$d(G) = |\mathrm{dom}(G)| - |\mathrm{Var}(G)|.$$

We are now ready to prove one of the most crucial lemmas in this paper.

LEMMA 49 (Key Lemma). *Let* G_1 *and* G_2 *be rigid grammars that have no useless type. If* $G_1 \sqsubset G_2$, *then* $d(G_1) < d(G_2)$.

Proof. Suppose that G_1 and G_2 are rigid grammars without useless types such that $\sigma[G_1] \subseteq G_2$. By Lemma 44, $\mathrm{Var}(G_i) = \mathrm{Head}(G_i) - \{t\}$ for $i = 1, 2$. Then we have

$$\begin{aligned}
|\mathrm{Var}(G_2)| &= |\mathrm{Head}(G_2) - \{t\}| \\
&= |(\mathrm{Head}(\sigma[G_1]) - \{t\}) \cup (\mathrm{Head}(G_2 - \sigma[G_1]) - \{t\})| \\
&\leq |\mathrm{Head}(\sigma[G_1]) - \{t\}| + |\mathrm{Head}(G_2 - \sigma[G_1]) - \{t\}| && (1) \\
&\leq |\mathrm{Head}(\sigma[G_1]) - \{t\}| + |\mathrm{Head}(G_2 - \sigma[G_1])| && (2) \\
&\leq |\mathrm{Head}(\sigma[G_1]) - \{t\}| + |\mathrm{dom}(G_2) - \mathrm{dom}(G_1)| && (3) \\
&\leq |\mathrm{Head}(G_1) - \{t\}| + |\mathrm{dom}(G_2) - \mathrm{dom}(G_1)| && (4) \\
&= |\mathrm{Var}(G_1)| + |\mathrm{dom}(G_2) - \mathrm{dom}(G_1)|,
\end{aligned}$$

where equality holds for

(1) just in case $(\mathrm{Head}(\sigma[G_1]) - \{t\}) \cap (\mathrm{Head}(G_2 - \sigma[G_1]) - \{t\}) = \emptyset$;

(2) just in case $t \notin \mathrm{Head}(G_2 - \sigma[G_1])$;

(3) just in case for all $b, c \in \mathrm{dom}(G_2) - \mathrm{dom}(G_1)$, $b \neq c$ implies $\mathrm{head}(B) \neq \mathrm{head}(C)$, where B and C are the types such that $G_2{:}\, b \mapsto B$ and $G_2{:}\, c \mapsto C$;

(4) just in case for all $x \in \mathrm{Head}(G_1) - \{t\}$, $\mathrm{head}(\sigma(x)) \neq t$ and for all $x, y \in \mathrm{Head}(G_1) - \{t\}$, $x \neq y$ implies $\mathrm{head}(\sigma(x)) \neq \mathrm{head}(\sigma(y))$.

So $|\mathrm{Var}(G_2)| \leq |\mathrm{Var}(G_1)| + |\mathrm{dom}(G_2) - \mathrm{dom}(G_1)|$, which is equivalent to $d(G_1) \leq d(G_2)$.

Assume that $d(G_1) = d(G_2)$. Then the conditions (1)–(4) above must hold. Let

$$\begin{aligned} \{x_1, \ldots, x_m\} &= \mathrm{Head}(G_1) - \{t\}, \\ \{y_1, \ldots, y_l\} &= \mathrm{Head}(G_2 - \sigma[G_1]), \\ \mathrm{head}(\sigma(x_i)) &= z_i. \end{aligned}$$

$\{z_1, \ldots, z_m, y_1, \ldots, y_l\} = \mathrm{Head}(G_2) - \{t\}$ and $z_1, \ldots, z_m, y_1, \ldots, y_l$ are all distinct. We will show

$$\{y_1, \ldots, y_l\} = \emptyset, \tag{5}$$

$$\sigma(x_i) = z_i \text{ for } 1 \leq i \leq m. \tag{6}$$

(5) and (6) mean that $\sigma \restriction \mathrm{Var}(G_1)$ is a one-to-one function from $\mathrm{Var}(G_1)$ to $\mathrm{Var}(G_2)$, so if we can show (5) and (6), we can establish that G_1 and G_2 are alphabetic variants and are thus equal. Note that $y_i \notin \{\sigma(x_1), \ldots, \sigma(x_m)\}$, and if $\sigma(x_i) \neq z_i$, then $z_i \notin \{\sigma(x_1), \ldots, \sigma(x_m)\}$. Thus, if either (5) or (6) fails to hold,

$$\begin{aligned} &\text{there is some } w \in \{z_1, \ldots, z_m, y_1, \ldots, y_l\} \\ &\text{such that } w \notin \{\sigma(x_1), \ldots, \sigma(x_m)\}\ . \end{aligned} \tag{7}$$

Therefore, we assume (7), to derive a contradiction. Since G_2 has no useless type, by Lemma 44, w must occur as an argument subtype of some type A in $\mathrm{range}(G_2)$. There are two cases.

CASE 1. $A \in \mathrm{range}(G_2 - \sigma[G_1])$. Then A must look like

$$\ldots((\ldots(y_j|A_1)|\ldots)|w)|\ldots$$

Since y_j does not occur as the head of any $B \neq A$ in $\mathrm{range}(G_2)$, this implies that y_j depends on w.

CASE 2. $A \in \mathrm{range}(\sigma[G_1])$. A looks like

$$\ldots((\ldots(p|A_1)|\ldots)|w)|\ldots$$

$A = \sigma(B)$ for some $B \in \mathrm{range}(G_1)$, and the assumption $\sigma(x_i) \neq w$ for all x_i implies that $w \neq \sigma(C)$ for any $C \in \mathrm{Tp}(G_1)$. This means that w occurs as an argument subtype of $\sigma(x_j)$, where $x_j = \mathrm{head}(B)$. Then $z_j = \mathrm{head}(\sigma(x_j)) \neq \sigma(x_j)$. z_j must depend on w, since every $D \in \mathrm{range}(G_2)$ with $\mathrm{head}(D) = z_j$ has $\sigma(x_j)$ as a head subtype.

Thus we have found a $w' \in \{z_1, \ldots, z_m, y_1, \ldots, y_l\}$ such that $w' \notin \{\sigma(x_1), \ldots, \sigma(x_m)\}$ and w' depends on w. Repeating this argument, we find a cycle of dependency $w_0, w_1, \ldots, w_n$ ($n \geq 1$) such that $w_0 = w_n$ and w_i depends on w_{i-1} for $1 \leq i \leq n$. This is a contradiction, for no grammar without useless types can afford to have such a cycle of dependency.

So we have proved (5) and (6). This concludes the proof.

Note that, by the definition of $d(G)$, if G is a rigid grammar in reduced form, then $0 \leq d(G) \leq \mathrm{dom}(G)$.

LEMMA 50. *Let $G_0, G_1, \ldots, G_n$ be rigid grammars over Σ in reduced form such that $G_0 \sqsubset G_1 \sqsubset \cdots \sqsubset G_n$. Then $n \leq \Sigma$.*

Proof. By Proposition 41 and Lemma 49, $d(G_0) < d(G_1) < \cdots < d(G_n)$. Since $0 \leq d(G_0)$ and $d(G_n) \leq |\Sigma|$, it must be that $n \leq |\Sigma|$.

PROPOSITION 51. *Let $L_0, L_1, \ldots, L_n$ be rigid structure languages over alphabet Σ such that $L_0 \subset L_1 \subset \cdots \subset L_n$. Then $n \leq |\Sigma|$.*

Proof. For $0 \leq i \leq n$, let G_i be the rigid grammar in reduced form such that $\mathrm{FL}(G_i) = L_i$. Then, by Proposition 38, $G_0 \sqsubset G_1 \sqsubset \cdots \sqsubset G_n$. By Lemma 50, $n \leq |\Sigma|$.

Note that even though $\mathcal{FL}_{1\text{-valued}}$ does not contain an infinite ascending chain, it is not hard to see that for every $L \in \mathcal{FL}_{1\text{-valued}}$, the set $\{ L' \in \mathcal{FL}_{1\text{-valued}} \mid L \subset L' \}$ is infinite unless it is empty.

THEOREM 52. *$\mathcal{FL}_{1\text{-valued}}$ has finite elasticity.*

Proof. By Lemma 39 and Proposition 51.

4.2. A Theorem on Finite Elasticity

The theorem proved in this subsection generalizes the essence of a theorem obtained by Wright (1989). We will use it to show that $\mathcal{FL}_{k\text{-valued}}$ and $\mathcal{L}_{k\text{-valued}}$ have finite elasticity. The method of the proof below is essentially the same as the one used independently by Moriyama and Sato (1993), but these authors do not state their result in full generality. For various applications of the theorem not mentioned in the present paper, see Kanazawa 1994a or 1994b.

Let Σ and Υ be two (not necessarily distinct) alphabets. A relation $R \subseteq \Sigma^* \times \Upsilon^*$ is said to be *finite-valued* iff for every $s \in \Sigma^*$, there are at most finitely many $u \in \Upsilon^*$ such that Rsu. If M is a language over Υ, define a language $R^{-1}[M]$ over Σ by $R^{-1}[M] = \{ s \mid \exists u(Rsu \wedge u \in M) \}$.

THEOREM 53. *Let $\mathcal{M}$ be a class of languages over Υ that has finite elasticity, and let $R \subseteq \Sigma^* \times \Upsilon^*$ be a finite-valued relation. Then $\mathcal{L} = \{ R^{-1}[M] \mid M \in \mathcal{M} \}$ also has finite elasticity.*

Proof. Suppose that $\mathcal{L} = \{ R^{-1}[M] \mid M \in \mathcal{M} \}$ has *in*finite elasticity. Then there is an infinite sequence of strings $s_0, s_1, s_2, \ldots$ over Σ and an infinite sequence of languages $L_0, L_1, L_2, \ldots$ from $\mathcal{L}$ such that for each n, $s_n \notin L_n$ and $\{s_0, \ldots, s_n\} \subseteq L_{n+1}$. For each $n \in \mathbb{N}$, take an $M_n \in \mathcal{M}$ such that $L_n = R^{-1}[M_n]$. For each $k \in \mathbb{N}$, let

$$U_k = \{\, \langle u_0, \ldots, u_k \rangle \mid Rs_0u_0 \wedge \cdots \wedge Rs_ku_k \wedge \exists n(\{u_0, \ldots, u_k\} \subseteq M_n) \,\}.$$

Note that each U_k is non-empty, and U_i and U_j are disjoint if $i \neq j$. Let

$$U = \bigcup_{k \in \mathbb{N}} U_k.$$

By the preceding remarks, U is infinite. U has the form of a tree: the mother of $\langle u_0, \ldots, u_k, u_{k+1} \rangle \in U$ is $\langle u_0, \ldots, u_k \rangle$, which is also in U. Since R is finite-valued, U is finitely branching. Since U is an infinite tree, by König's Lemma, U has an infinite branch. Let $u_0, u_1, u_2, \ldots$ be an infinite sequence of strings over Υ that corresponds to an infinite branch of U; i.e., $\langle u_0 \rangle, \langle u_0, u_1 \rangle, \langle u_0, u_1, u_2 \rangle, \ldots$ are the nodes on this branch. Note that $s_n \notin L_n$ implies

$$u_n \notin M_n. \tag{8}$$

For each n, let $f(n)$ be such that $\{u_0, \ldots, u_n\} \subseteq M_{f(n)}$ and for all $j < f(n)$, $\{u_0, \ldots, u_n\} \not\subseteq M_j$. By (8), $n < f(n)$ for all n. For each n, let $g(n) = f^n(0) = \underbrace{f(\ldots(f}_{n \text{ times}}(0))\ldots)$. Note that g is monotone increasing. We claim that

$$u_{g(0)}, u_{g(1)}, \ldots, u_{g(n)}, \ldots$$

and

$$M_{g(0)}, M_{g(1)}, \ldots, M_{g(n)}, \ldots$$

witness the infinite elasticity of $\mathcal{M}$. We have (8), so it is enough to observe that by the definition of g,

$$\{u_{g(0)}, \ldots, u_{g(n)}\} \subseteq M_{g(n+1)}$$

for all $n \in \mathbb{N}$.

We can use Theorems 52 and 53 to show that $\mathcal{FL}_{k\text{-valued}}$ has finite elasticity. Let Σ be a fixed alphabet, and let k be a fixed natural number ≥ 2. We can associate with each k-valued grammar over Σ a rigid grammar over Υ, where Υ is an alphabet that contains k copies of each symbol in Σ, i.e.,

$$\Upsilon = \bigcup \{\, \{c_1, \ldots, c_k\} \mid c \in \Sigma \,\},$$

where all c_j's are assumed to be distinct. In order to determine the desired association uniquely, let us fix a total ordering $\prec$ on Tp.

DEFINITION 54. For each k-valued grammar G over Σ, we define the *rigid counterpart* $\mathrm{rc}(G)$ of G to be the following rigid grammar over Υ:

$$\mathrm{rc}(G) = \{\, \langle c_i, A_i \rangle \mid c \in \Sigma \text{ and } A_i \text{ is the } i\text{-th element of } \{\, A \mid \langle c, A \rangle \in G \,\} \,\}.$$

Here, 'i-th' refers to the ordering $\prec$.

DEFINITION 55. Let amb: $\Upsilon^{\mathrm{F}} \to \Sigma^{\mathrm{F}}$ be the homomorphism that maps each copy c_i of c to c. That is:

$$\begin{aligned} \mathrm{amb}(c_i) &= c && \text{for all } c \in \Sigma, \\ \mathrm{amb}(\mathrm{BA}(U_1, U_2)) &= \mathrm{BA}(\mathrm{amb}(U_1), \mathrm{amb}(U_2)) && \text{for all } U_1, U_2 \in \Upsilon^{\mathrm{F}}, \\ \mathrm{amb}(\mathrm{FA}(U_1, U_2)) &= \mathrm{FA}(\mathrm{amb}(U_1), \mathrm{amb}(U_2)) && \text{for all } U_1, U_2 \in \Upsilon^{\mathrm{F}}. \end{aligned}$$

(Here we are using the term representation of functor-argument structures.) If $M \subseteq \Upsilon^{\mathrm{F}}$, we let amb$[M]$ denote $\{ \mathrm{amb}(U) \mid U \in M \}$.

The following should be clear from definition.

LEMMA 56. *Let G be a k-valued grammar over Σ. Then* $\mathrm{FL}(G) = \mathrm{amb}[\mathrm{FL}(\mathrm{rc}(G))]$.

THEOREM 57. $\mathcal{FL}_{k\text{-valued}}$ *has finite elasticity.*

Proof. By Theorem 52, the class $\{ \mathrm{FL}(H) \mid H$ is a rigid grammar over $\Upsilon \}$ has finite elasticity. Then $\{ \mathrm{FL}(\mathrm{rc}(G)) \mid G$ is a k-valued grammar over $\Sigma \}$ also has finite elasticity, since the latter is a subset of the former. The relation

$$T = \mathrm{amb}[U]$$

between $T \in \Sigma^{\mathrm{F}}$ and $U \in \Upsilon^{\mathrm{F}}$ is clearly finite-valued. Then, by Theorem 53, the class $\{ \mathrm{amb}[\mathrm{FL}(\mathrm{rc}(G))] \mid G$ is a k-valued grammar over $\Sigma \}$ has finite elasticity. But this class equals $\{ \mathrm{FL}(G) \mid G$ is a k-valued grammar over $\Sigma \}$ by Lemma 56.

Another application of Theorem 53 proves the following:

THEOREM 58. $\mathcal{L}_{k\text{-valued}}$ *has finite elasticity.*

Proof. Recall that

$$\mathrm{L}(G) = \{ \mathrm{yield}(T) \mid T \in \mathrm{FL}(G) \}.$$

If $L \subseteq \Sigma^{\mathrm{F}}$, we write yield$[L]$ for $\{ \mathrm{yield}(T) \mid T \in L \}$. Then

$$\mathcal{L}_{k\text{-valued}} = \{ \mathrm{yield}[L] \mid L \in \mathcal{FL}_{k\text{-valued}} \}.$$

Note that the relation $R \subseteq \Sigma^{+} \times \Sigma^{\mathrm{F}}$ defined by

$$RsT \Leftrightarrow s = \mathrm{yield}(T)$$

is finite-valued; for, if RsT, then the number of nodes of T is exactly $2|s| - 1$, and there are only finitely many functor-argument structures with a given number of nodes. Since $\mathcal{FL}_{k\text{-valued}}$ has finite elasticity (Theorem 57), an application of Theorem 53 shows that $\mathcal{L}_{k\text{-valued}}$ also has finite elasticity.

By Theorem 7, Proposition 58 implies the main theorem (Theorem 12).

5. Learning Algorithms for k-Valued Grammars

There is a 'universal' learning algorithm that works for an arbitrary r.e. class $\mathcal{G}$ of grammars for which universal membership is decidable and whose corresponding language class has finite elasticity (see Wright 1989 and Kapur 1991). This algorithm of course can be used to learn $\mathcal{G}_{k\text{-valued}}$, but its simple 'enumerative' behavior does not bring any new insight. In this subsection, we present a concrete learning algorithm that learns $\mathcal{G}_{k\text{-valued}}$, for each k. We do this in stages. First, we treat the case $k = 1$, to illustrate the core idea of the algorithm. Next, we treat the general case using the notion of the rigid counterpart of a k-valued gramar defined in the previous section.

5.1. The Case $k = 1$

Our learning function for $\mathcal{G}_{1\text{-valued}}$ is defined in terms of two computable functions, $\Psi_{1\text{-valued}}$ and $\mu^{(2)}$. $\Psi_{1\text{-valued}}$ is a function that maps a finite set of strings to a finite set of rigid grammars, and $\mu^{(2)}$ is a function that takes two arguments, a finite set of grammars and a positive integer, and returns a member of the first argument.

DEFINITION 59.

$$\Psi_{1\text{-valued}}(\{s_0, \ldots, s_i\}) = \{\, \mathrm{RG}(\{T_0, \ldots, T_i\}) \mid s_j = \mathrm{yield}(T_j)\ (0 \leq j \leq i) \,\}.$$

Each grammar in the value of $\Psi_{1\text{-valued}}$ is the result of randomly guessing the functor-argument structure of each string in the set given as the argument to $\Psi_{1\text{-valued}}$ and then applying Buszkowski's algorithm to the resulting set of functor-argument structures.

EXAMPLE 60. Let

$$s_0 = \textbf{mary swims},$$
$$s_1 = \textbf{mary swims fast}.$$

There are two functor-argument structures that yield s_0, and eight that yield s_1, so there are 16 possible analyses of $\{s_0, s_1\}$. Buszkowski's algorithm RG succeeds on six of them, and we have $\Psi_{1\text{-valued}}(\{s_0, s_1\}) = \{G_1, G_2, G_3, G_4, G_5, G_6\}$, where

$$\begin{array}{lrcl}
G_1\text{:} & \textbf{fast} & \mapsto & (x\backslash t)\backslash(x\backslash t), \\
 & \textbf{mary} & \mapsto & x, \\
 & \textbf{swims} & \mapsto & x\backslash t, \\
G_2\text{:} & \textbf{fast} & \mapsto & t\backslash t, \\
 & \textbf{mary} & \mapsto & x, \\
 & \textbf{swims} & \mapsto & x\backslash t, \\
G_3\text{:} & \textbf{fast} & \mapsto & ((t/x)\backslash t)\backslash x, \\
 & \textbf{mary} & \mapsto & t/x,
\end{array}$$

$$
\begin{array}{rrcl}
 & \textbf{swims} & \mapsto & (t/x)\backslash t, \\
G_4\colon & \textbf{fast} & \mapsto & x\backslash((t/x)\backslash t), \\
 & \textbf{mary} & \mapsto & t/x, \\
 & \textbf{swims} & \mapsto & x, \\
G_5\colon & \textbf{fast} & \mapsto & t\backslash t, \\
 & \textbf{mary} & \mapsto & t/x, \\
 & \textbf{swims} & \mapsto & x, \\
G_6\colon & \textbf{fast} & \mapsto & x\backslash x, \\
 & \textbf{mary} & \mapsto & t/x, \\
 & \textbf{swims} & \mapsto & x.
\end{array}
$$

Note that $\mathrm{L}(G_1) = \mathrm{L}(G_2) = \mathrm{L}(G_5) = \mathrm{L}(G_6) = \{\, \textbf{mary swims fast}^n \mid n \in \mathbb{N} \,\}$ and $\mathrm{L}(G_3) = \mathrm{L}(G_4) = \{\textbf{mary swims}, \textbf{mary swims fast}\}$.

By Proposition 31, it is easy to see the following:

LEMMA 61. *If* $G \in \mathcal{G}_{\text{1-valued}}$ *and* $\{s_0, \ldots, s_i\} \subseteq \mathrm{L}(G)$, *then there is some* $G' \in \Psi_{\text{1-valued}}(\{s_0, \ldots, s_i\})$ *such that* $G' \sqsubseteq G$.

The above lemma implies the next two lemmas:

LEMMA 62. *If* $G \in \Psi_{\text{1-valued}}(\{s_0, \ldots, s_i\})$, *then* $\{s_0, \ldots, s_i\} \subseteq \mathrm{L}(G)$.

LEMMA 63. $\{\, \mathrm{L}(G) \mid G \in \Psi_{\text{1-valued}}(\{s_0, \ldots, s_i\}) \,\}$ *includes all minimal elements of* $\{\, L \in \mathcal{L}_{\text{1-valued}} \mid \{s_0, \ldots, s_i\} \subseteq L \,\}$.

The following simple fact will be important in proving the correctness of our learning algorithms:

LEMMA 64 (Kapur). *Let* $\mathcal{L}$ *be a language class with finite elasticity. Then for each language* $L \in \mathcal{L}$, *there is a finite set* $D_L \subseteq L$ *such that* L *is the unique smallest element of* $\{\, L' \in \mathcal{L} \mid D_L \subseteq L' \,\}$.

Proof. We prove the contrapositive. Let $\mathcal{L}$ be any language class, and suppose that for some $L \in \mathcal{L}$, for every finite set $D \subseteq L$, there is an $L' \in \mathcal{L}$ such that $D \subseteq L'$ and $L \not\subseteq L'$. Let $\langle s_i \rangle_{i \in \mathbb{N}}$ be an infinite sequence enumerating L. By assumption, for every $i \in \mathbb{N}$, there is an $L_i \in \mathcal{L}$ such that $\{s_0, \ldots, s_i\} \subseteq L_i$ and $L \not\subseteq L_i$. Let f be a function such that for all i, $s_{f(i)} \in L - L_i$. Note $f(i) > i$. Let $g(n) = f^n(0) = \underbrace{f(\ldots(f}_{n \text{ times}}(0))\ldots)$. Then

$$s_{g(1)}, s_{g(2)}, \ldots, s_{g(n+1)}, \ldots$$

and

$$L_{g(0)}, L_{g(1)}, \ldots, L_{g(n)}, \ldots$$

witness the infinite elasticity of $\mathcal{L}$.

Let $\mu_{\text{1-valued}}$ be a function that maps a non-empty finite set $\mathcal{G}$ of rigid grammars to an element G_0 of $\mathcal{G}$ such that $\mathrm{L}(G_0)$ is minimal in $\{\mathrm{L}(G) \mid G \in \mathcal{G}\}$. Then, by Lemmas 62 and 63, $\mathrm{L}(\mu_{\text{1-valued}}(\Psi_{\text{1-valued}}(\{s_0, \ldots, s_i\})))$ is always a minimal element of $\{L \in \mathcal{L}_{\text{1-valued}} \mid \{s_0, \ldots, s_i\} \subseteq L\}$. By Theorem 58 and Lemma 64, then, for every $L \in \mathcal{L}_{\text{1-valued}}$, there is a finite set D_L of strings such that if $D_L \subseteq \{s_0, \ldots, s_i\} \subseteq L$, $\mathrm{L}(\mu_{\text{1-valued}}(\Psi_{\text{1-valued}}(\{s_0, \ldots, s_i\}))) = L$. Define a learning function φ as follows:

$$\varphi(\langle s_0 \rangle) = \mu_{\text{1-valued}}(\Psi_{\text{1-valued}}(\{s_0\})),$$

$$\varphi(\langle s_0, \ldots, s_{i+1} \rangle) = \begin{cases} \varphi(\langle s_0, \ldots, s_i \rangle) \\ \qquad \text{if } s_{i+1} \in \mathrm{L}(\varphi(\langle s_0, \ldots, s_i \rangle)), \\ \mu_{\text{1-valued}}(\Psi_{\text{1-valued}}(\{s_0, \ldots, s_{i+1}\})) \\ \qquad \text{otherwise.} \end{cases}$$

It is now easy to see that φ learns $\mathcal{G}_{\text{1-valued}}$.* However, we do not know if there is a definition of $\mu_{\text{1-valued}}$ that makes it computable,** so the learning function φ defined thus may not be computable.

Although there may be no computable function like $\mu_{\text{1-valued}}$, there is one that is 'computable in the limit' in the following sense. Let $\prec$ be a computable well-order of all grammars, and let $\Sigma^{\leq n}$ be the set of strings over Σ of length $\leq n$.

DEFINITION 65. Let $\mu^{(2)}$ be a function that takes two arguments, a non-empty finite set $\mathcal{G}$ of grammars and a positive integer n, and returns the first member G_0 of $\mathcal{G}$ (under the ordering $\prec$) such that $\mathrm{L}(G_0) \cap \Sigma^{\leq n}$ is a minimal element of $\{\mathrm{L}(G) \cap \Sigma^{\leq n} \mid G \in \mathcal{G}\}$.

Since the universal membership problem is decidable for categorial grammars, $\mu^{(2)}$ is computable. It is easy to see the following:

LEMMA 66. *For every finite set $\mathcal{G}$ of grammars, there is an m such that for all $n \geq m$, $\mu^{(2)}(\mathcal{G}, m) = \mu^{(2)}(\mathcal{G}, n)$ and $\mathrm{L}(\mu^{(2)}(\mathcal{G}, m))$ is a minimal element of $\{\mathrm{L}(G) \mid G \in \mathcal{G}\}$.*

One can define a computable learning function $\psi_{\text{1-valued}}$ in terms of $\Psi_{\text{1-valued}}$ and $\mu^{(2)}$. For this, we first define a 'conservative' version of $\Psi_{\text{1-valued}}$:

---

* Moreover, φ has the desirable property of beging *conservative* (Angluin 1980b). See Kanazawa 1994a for a discussion of the question of whether $\mathcal{G}_{k\text{-valued}}$ is conservatively learnable.

** One can show that there is no computable function μ that takes a finite set $\mathcal{G}$ of arbitrary categorial grammars and returns an element G_0 of $\mathcal{G}$ such that $\mathrm{L}(G_0)$ is minimal in $\{\mathrm{L}(G) \mid G \in \mathcal{G}\}$ (Kanazawa 1994a). It is an open question whether there is any k such that a computable function $\mu_{k\text{-valued}}$ exists that maps a finite set $\mathcal{G}$ of k-valued grammars to an element G_0 of $\mathcal{G}$ such that $\mathrm{L}(G_0)$ is minimal in $\{\mathrm{L}(G) \mid G \in \mathcal{G}\}$.

DEFINITION 67.

$$\Psi^{\#}_{\text{1-valued}}(\langle s_0\rangle) = \Psi_{\text{1-valued}}(\{s_0\}),$$
$$\Psi^{\#}_{\text{1-valued}}(\langle s_0, \ldots, s_{i+1}\rangle) =$$
$$\begin{cases} \Psi^{\#}_{\text{1-valued}}(\langle s_0, \ldots, s_i\rangle) & \text{if } s_{i+1} \in \mathrm{L}(G) \text{ for every} \\ & \quad G \in \Psi^{\#}_{\text{1-valued}}(\langle s_0, \ldots, s_i\rangle), \\ \Psi_{\text{1-valued}}(\{s_0, \ldots, s_{i+1}\}) & \text{otherwise.} \end{cases}$$

It is easy to see the following:

LEMMA 68. *For every* $G \in \Psi^{\#}_{\text{1-valued}}(\langle s_0, \ldots, s_i\rangle)$, $\{s_0, \ldots, s_i\} \subseteq \mathrm{L}(G)$.

LEMMA 69. $\{\, \mathrm{L}(G) \mid G \in \Psi^{\#}_{\text{1-valued}}(\langle s_0, \ldots, s_i\rangle) \,\}$ *includes all minimal elements of* $\{\, L \in \mathcal{L}_{\text{1-valued}} \mid \{s_0, \ldots, s_i\} \subseteq L \,\}$.

DEFINITION 70.

$$\psi_{\text{1-valued}}(\langle s_0, \ldots, s_i\rangle) = \mu^{(2)}(\Psi^{\#}_{\text{1-valued}}(\langle s_0, \ldots, s_i\rangle), i+1)$$

($i+1$ is the length of the sequence $\langle s_0, \ldots, s_i\rangle$.)

EXAMPLE 71. Let s_0, s_1 and $G_1, \ldots, G_6$ be as in Example 60. Suppose that the well-ordering $\prec$ on all grammars is such that $G_2 \prec G_5 \prec G_6 \prec G_1 \prec G_4 \prec G_3$. Then

$$\Psi^{\#}_{\text{1-valued}}(\langle s_0, s_1\rangle) = \{G_1, G_2, G_3, G_4, G_5, G_6\},$$
$$\psi_{\text{1-valued}}(\langle s_0, s_1\rangle) = G_2.$$

At this point, $\psi_{\text{1-valued}}$ is picking a grammar that generates the bigger of the two languages involved. $\psi_{\text{1-valued}}(\langle s_0, s_1, s_1\rangle) = \psi_{\text{1-valued}}(\langle s_0, s_1\rangle)$, but we have

$$\Psi^{\#}_{\text{1-valued}}(\langle s_0, s_1, s_1, s_1\rangle) = \{G_1, G_2, G_3, G_4, G_5, G_6\},$$
$$\psi_{\text{1-valued}}(\langle s_0, s_1, s_1, s_1\rangle) = G_4,$$

since **mary swims fast fast** is in $\mathrm{L}(G_1)$, $\mathrm{L}(G_2)$, $\mathrm{L}(G_5)$, and $\mathrm{L}(G_6)$, but not in $\mathrm{L}(G_3)$ and $\mathrm{L}(G_4)$.

THEOREM 72. $\psi_{\text{1-valued}}$ *learns* $\mathcal{G}_{\text{1-valued}}$.

Proof. Let $L \in \mathcal{L}_{\text{1-valued}}$ and let $\langle s_i\rangle_{i\in\mathbb{N}}$ be an infinite sequence enumerating L. By Lemma 64, let D_L be a finite subset of L such that L is the unique smallest element of $\{\, L' \in \mathcal{L}_{\text{1-valued}} \mid D_L \subseteq L' \,\}$. Let l be the least such that $D_L \subseteq \{s_0, \ldots, s_l\}$. Then, by Lemmas 68 and 69, if $i \geq l$, L is the unique minimal element of $\{\, \mathrm{L}(G) \mid G \in \Psi^{\#}_{\text{1-valued}}(\langle s_0, \ldots, s_i\rangle) \,\}$. Also, by the definition of $\Psi^{\#}_{\text{1-valued}}$, this means that if $i \geq l$, $\Psi^{\#}_{\text{1-valued}}(\langle s_0, \ldots, s_i\rangle) = \Psi^{\#}_{\text{1-valued}}(\langle s_0, \ldots, s_l\rangle)$. Let $\mathcal{G} = \Psi^{\#}_{\text{1-valued}}(\langle s_0, \ldots, s_l\rangle)$. By Lemma 66, we can find an m such that for

every $n \geq m$, $\mu^{(2)}(\mathcal{G}, m) = \mu^{(2)}(\mathcal{G}, n)$ and $\mathrm{L}(\mu^{(2)}(\mathcal{G}, m)) = L$. Therefore, if $i \geq \max(l, m-1)$, $\psi_{1\text{-valued}}(\langle s_0, \ldots, s_i \rangle) = \psi_{1\text{-valued}}(\langle s_0, \ldots, s_{\max(l,m-1)} \rangle)$ and $\mathrm{L}(\psi_{1\text{-valued}}(\langle s_0, \ldots, s_i \rangle)) = L$.

5.2. The General Case

In the learning algorithm for $\mathcal{G}_{1\text{-valued}}$ presented in the previous subsection, a finite set of strings is 'analyzed' into a finite set of functor-argument structuers that yield the strings and Buszkowski's algorithm RG is applied to the result. In the learning algorithm for $\mathcal{G}_{k\text{-valued}}$ to be presented below, the input strings are first analyzed into functor-argument structures as before, but the structures thus produced are further 'disambiguated' into functor-argument structures over the extended alphabet Υ, which contains k copies of each symbol in the original alphabet Σ. Buszkowski's algorithm is then applied to those disambiguated functor-argument structures, and the rigid grammar over Υ that is output by the algorithm is 'ambiguated' back into a k-valued grammar over Σ.

Recall the definition of Υ, rc, and amb from Section 4.2. If H is a rigid grammar over Υ, $\mathrm{rc}^{-1}(H)$ is the k-valued grammar over Σ of which H is the rigid counterpart.

DEFINITION 73.
$\Psi_{k\text{-valued}}(\{s_0, \ldots, s_i\}) = \{\mathrm{rc}^{-1}(H) \mid$ for some $U_0, \ldots, U_i$ such that $\mathrm{yield}(\mathrm{amb}(U_j)) = s_j$ $(0 \leq j \leq i)$, $H = \mathrm{RG}(\{U_0, \ldots, U_i\})\}$.

LEMMA 74. *If* $G \in \mathcal{G}_{k\text{-valued}}$ *and* $\{s_0, \ldots, s_i\} \subseteq \mathrm{L}(G)$, *then there is some* $G' \in \Psi_{k\text{-valued}}(\{s_0, \ldots, s_i\})$ *such that* $\mathrm{rc}(G') \sqsubseteq \mathrm{rc}(G)$.

Proof. Since, by Lemma 56, $\mathrm{L}(G) = \mathrm{yield}(\mathrm{amb}(\mathrm{FL}(\mathrm{rc}(G))))$, if $\{s_0, \ldots, s_i\} \subseteq \mathrm{L}(G)$, then there must be some $U_0, \ldots, U_i$ such that for $0 \leq j \leq i$, $s_j = \mathrm{yield}(\mathrm{amb}(U_j))$ and $\{U_0, \ldots, U_i\} \subseteq \mathrm{FL}(\mathrm{rc}(G))$. Then, by Lemma 31, $\mathrm{RG}(\{U_0, \ldots, U_i\}) \sqsubseteq \mathrm{rc}(G)$.

LEMMA 75. *If* $G \in \Psi_{k\text{-valued}}(\{s_0, \ldots, s_i\})$, *then* $\{s_0, \ldots, s_i\} \subseteq \mathrm{L}(G)$.

LEMMA 76. $\{\mathrm{L}(G) \mid G \in \Psi_{k\text{-valued}}(\{s_0, \ldots, s_i\})\}$ *includes all minimal elements of* $\{L \in \mathcal{L}_{k\text{-valued}} \mid \{s_0, \ldots, s_i\} \subseteq L\}$.

Proof. Clear from Lemma 74, noting that $\mathrm{rc}(G') \sqsubseteq \mathrm{rc}(G)$ implies that $\mathrm{L}(G') \subseteq \mathrm{L}(G)$.

DEFINITION 77.

$$\Psi^{\#}_{k\text{-valued}}(\langle s_0 \rangle) = \Psi_{k\text{-valued}}(\{s_0, \ldots, s_i\}),$$

$$\Psi^{\#}_{k\text{-valued}}(\langle s_0, \ldots, s_i \rangle) = \begin{cases} \Psi^{\#}_{k\text{-valued}}(\langle s_0, \ldots, s_i \rangle) & \text{if } s_{i+1} \in \mathrm{L}(G) \text{ for every } G \in \Psi^{\#}_{k\text{-valued}}(\langle s_0, \ldots, s_i \rangle), \\ \Psi_{k\text{-valued}}(\{s_0, \ldots, s_{i+1}\}) & \text{otherwise.} \end{cases}$$

DEFINITION 78.

$$\psi_{k\text{-valued}}(\langle s_0, \ldots, s_i \rangle) = \mu^{(2)}(\Psi^{\#}_{k\text{-valued}}(\langle s_0, \ldots, s_i \rangle), i+1).$$

The proof of the correctness of $\psi_{k\text{-valued}}$ is exactly the same as in the case of $\psi_{1\text{-valued}}$.

6. Discussion: Learning with Additional Information

So far, we have assumed that a fixed finite alphabet is given. In fact, our algorithms $\psi_{1\text{-valued}}$ and $\psi_{k\text{-valued}}$ can work on an infinite alphabet, as long as each grammar (language) is based on a finite subalphabet of it. Let Σ be a fixed finite alphabet, and let $\Upsilon^{\infty} = \bigcup\{ \{ c_i \mid i \in \mathbb{N} \} \mid c \in \Sigma \}$. Assume that Υ^{∞} is coded in some finite alphabet, by, for example, regarding c_i as c followed by i strokes. Let $\mathcal{G}^{\infty}_{1\text{-valued}}$ be the class of all rigid grammars over some finite subalphabet of Υ^{∞}. Although $\{ \mathrm{L}(G) \mid G \in \mathcal{G}^{\infty}_{1\text{-valued}} \}$ now has infinite elasticity, it is clear that essentially the same algorithm as $\psi_{1\text{-valued}}$ learns $\mathcal{G}^{\infty}_{1\text{-valued}}$.

This fact has an interesting consequence. Consider the class CatG of all classical categorial grammars over Σ. Then $\mathcal{G}^{\infty}_{1\text{-valued}}$ includes all rigid counterparts of grammars in CatG.* That $\mathcal{G}^{\infty}_{1\text{-valued}}$ is learnable means that an arbitrary classical categorial grammar can be learned from data consisting of *disambiguated strings*, that is, strings in the language generated by the rigid counterpart of the target grammar. Disambiguated strings can be regarded as positive data about the target language augmented with certain 'intensional' information about the target grammar. With this additional information, the entire class of classical categorial grammars, which generate all ϵ-free context-free languages, becomes learnable.

To be more precise, $\langle \mathrm{CatG}, (\Upsilon^{\infty})^{+}, \mathrm{L}(\mathrm{rc}(\cdot)) \rangle$ constitutes a grammar system. In this grammar system, CatG is learnable.

This can be compared with Sakakibara's (1992) result about *reversible context-free grammars*. A context-free grammar is said to be *reversible* if no two distinct rules of G differ with respect to just one non-terminal; that is, if both $A \to \alpha$ and $B \to \alpha$ are rules of G, then $A = B$, and if both $A \to \alpha\ B\ \beta$ and $A \to \alpha\ C\ \beta$ are rules of G, then $B = C$. The class of reversible context-free grammars is a normal form for the class of all context-free grammars in the sense that for every context-free language L, there is a reversible context-free grammar G such that $L = \mathrm{L}(G)$. This means that the class of reversible context-free grammars is not learnable in the standard grammar system of context-free grammars. Sakakibara's (1992) result is that the class of reversible context-free grammars is learnable in the grammar system where the sentences *skeletal phrase structures* (Levy and Joshi 1978), rather than strings. A skeletal phrase structure over alphabet Σ is a tree whose leaf nodes are labeled by symbols in Σ but whose internal nodes have no labels. A context-free grammar G generates a skeletal phrase structure if it is the

* Note that the rigid counterpart $\mathrm{rc}(G)$ of a k-valued grammar G is uniquely determined irrespective of the choice of k, as long as $G \in \mathcal{G}_{k\text{-valued}}$.

result of stripping non-terminal symbols of a parse tree of G. Based on Angluin's (1982) work, Sakakibara describes a polynomial-time learning algorithm that learns the class of reversible context-free grammars from skeletal phrase structures. In his words (1992, p. 59), 'the assumption of examples in the form of structural descriptions strongly compensates for the lack of explicit negative information in positive samples and is helpful for efficient learning of context-free grammars.'*

While Sakakibara's result is interesting, the assumption that the learner is presented with the skeletal phrase structure of each string that she encounters is probably not very attractive as a model of first language acquisition. In contrast, the grammar system $\langle \mathrm{CatG}, \Upsilon^{\infty+}, \mathrm{L}(\mathrm{rc}(\cdot))\rangle$ correspondes to a model of language acquisition where the learner is presented with strings with additional information that signal syntactically different uses of a lexical ambiguous symbol, which might be more realistic.** Moreover, structures assigned by a reversible context-free grammar are sometimes rather unnatural, and they are rich enough that they can in effect encode informaiton about lexical ambiguity. To keep the balance, we do not have the kind of efficient algorithm for learning rigid classical categorial grammars from strings that Sakakibara has for learning reversible context-free grammars from skeletal phrase structures.

Appendix

7. Reduction to Shinohara's Theorem

We used the fact that $\mathcal{FL}_{1\text{-valued}}$ has finite elasticity (Theorem 52) and our Theorem 53 (twice) to prove Proposition 58. There is an alternative proof that reduces it to Shinohara's (1990a, 1990b) result, which says that the class of languages generated by context-sensitive grammars with at most k rules has finite elasticity. It can be shown that for any k-valued categorial grammar G over Σ, there is an ϵ-free context-free grammar $\widehat{G}$ (in general not in Chomsky normal form) with at most $2k|\Sigma| - 1$ rules such that $\mathrm{L}(G) = \mathrm{L}(\widehat{G})$. This shows that $\mathcal{L}_{k\text{-valued}}$ is included in a class known to have finite elasticity, which implies that $\mathcal{L}_{k\text{-valued}}$ itself has finite elasticity.

PROPOSITION 79. *For any classical categorial grammar G, there is an ϵ-free context-free grammar $\widehat{G}$ with no more than $2|G| - 1$ rules such that* $\mathrm{L}(G) = \mathrm{L}(\widehat{G})$.‡

Proof. Without loss of generality, we can assume that G has no useless type. Let $\mathrm{cf}(G) = \langle \Sigma, \mathrm{Tp}(G), t, P_{\mathrm{cf}(G)}\rangle$ be the obvious Chomsky normal form context-free

* There is some similarity between Sakakibara's (1992) algorithm and Buszkowski's algorithm RG. In Section 3.3, we noted in passing that with respect to the grammar system $\langle \mathrm{CatG}, \Sigma^{\mathrm{F}}, \mathrm{FL}\rangle$, Buszkowski's algorithm learns the class of rigid classical categorial grammars. The difference is that the rigid classical categorial grammars do not generate all (ϵ-free) context-free languages.

** Note that the notions of 'structure' and 'lexical ambiguity' are both dependent on the specific grammar that generates the given language. In particular, a classical categorial grammar may have to assign more than one type to a certain symbol c while an equivalent Chomsky normal form context-free grammar may have just one rule of the form $A \to c$.

‡ $|G|$ is the number of type assignments in G.

grammar corresponding to G, where

$$P_{\mathrm{cf}(G)} = \{ B \to A \;\; A\backslash B \mid A\backslash B \in \mathrm{Tp}(G) \} \cup \{ B \to B/A \;\; A \mid B/A \in \mathrm{Tp}(G) \} \cup \{ A \to c \mid G{:}\, c \mapsto A \}.$$

G has no useless non-terminal. Note that there is no bound on the number of rules in $P_{\mathrm{cf}(G)}$. Let

$$\mathcal{A} = \{ A \in \mathrm{Tp}(G) - \{t\} \mid \text{there is only one rule in } P_{\mathrm{cf}(G)} \text{ whose left-hand side is } A \}.$$

For $A \in \mathcal{A}$, let $\mathrm{right}(A)$ be such that $A \to \mathrm{right}(A)$ is a rule in $P_{\mathrm{cf}(G)}$. Let

$$P_1 = P_{\mathrm{cf}(G)} - \{ A \to \mathrm{right}(A) \mid A \in \mathcal{A} \}.$$

Now eliminate all occurrences of non-terminals in $\mathcal{A}$ from the right-hand side of rules in P_1 by repeatedly replacing such occurrences of $A \in \mathcal{A}$ by $\mathrm{right}(A)$. This process must terminate, for, if it does not, one can show using König's Lemma that there must be a cycle of non-terminals $A_0, A_1, \ldots, A_n = A_0$ $(n \geq 1)$ such that each A_i $(0 \leq i \leq n)$ is in $\mathcal{A}$ and A_{i+1} occurs in $\mathrm{right}(A_i)$ for $0 \leq i \leq n-1$, which implies that $A_0, A_1, \ldots, A_n$ are useless non-terminals in $\mathrm{cf}(G)$. Let P_2 be the result of applying this process to P_1. Let

$$\widehat{G} = \langle \Sigma, \mathrm{Tp}(G) - \mathcal{A}, t, P_2 \rangle.$$

It should be clear that $\mathrm{L}(\widehat{G}) = \mathrm{L}(\mathrm{cf}(G))$.

It remains to show that $|P_2| \leq 2|G| - 1$. This can be seen as follows. Define a binary relation $\prec$ on $\mathrm{Tp}(G) \cup G$ as follows:

$$\prec = \{ \langle B, A\backslash B \rangle \mid A\backslash B \in \mathrm{Tp}(G) \} \cup \{ \langle B, B/A \rangle \mid B/A \in \mathrm{Tp}(G) \} \cup \{ \langle A, \langle c, A \rangle \rangle \mid \langle c, A \rangle \in G \}.$$

There is a one-to-one correspondence between the pairs in $\prec$ and the rules in $P_{\mathrm{cf}(G)}$. The graph of $\prec$ consists of m rooted trees, where $m = |\mathrm{Var}(G) \cup \{t\}| = |\mathrm{Var}(G)| + 1$. The number of leaf nodes of this tree is $|G|$, and it is not difficult to see that the number of nodes in these trees which have more than one daughter is at most $|G| - m$. Let

$$R = \{ \langle A, X \rangle \in \prec \mid A \text{ has more than one daughter in the graph of } \prec \}.$$

Note that R corresponds one-to-one to the set consisting of the leaf nodes of the graph of $\prec$ plus the nodes with more than one daughter that are not highest. Then

$$\begin{aligned} |R| &\leq |G| + |G| - m - m \\ &= 2|G| - 2m. \end{aligned}$$

Note that by the definition of P_2, $|P_2| \leq |R| + 1$, so

$$\begin{aligned} |P_2| &\leq 2|G| - 2m + 1 \\ &\leq 2|G| - 1, \end{aligned}$$

which concludes the proof.*

References

Ajdukiewicz, K., 1935, "Die syntaktische Konnexität", *Studia Philosophica* **1**, 1–27.

Angluin, D., 1980a, "Finding patterns common to a set of strings", *Journal of Computer and System Sciences* **21**, 46–62.

Angluin, D., 1980b, "Inductive inference of formal languages from positive data", *Information and Control* **45**, 117–135.

Angluin, D., 1982, "Inference of reversible languages", *Journal of the Association for Computing Machinery* **29**, 741–765.

Angluin, D. and Smith, C. H., 1983, "Inductive inference: theory and methods", *Computing Surveys* **15**, 237–269.

Barendregt, H. P., 1992, "Lambda calculi with types", in *Handbook of Logic in Computer Science, Volume 2*, S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, eds., Oxford: Clarendon Press.

Bar-Hillel, Y., 1953, "A quasi-arithmetical notation for syntactic description", *Language* **29**, 47–58. Reprinted in Bar-Hillel 1964.

Bar-Hillel, Y., 1960, "Some linguistic obstacles to machine translation", reprinted in Bar-Hillel 1964.

Bar-Hillel, Y., 1964, *Language and Information*, Reading, MA: Addison-Wesley.

Bar-Hillel, Y., Gaifman, H., and Shamir, E., 1960, "On categorial and phrase structure grammars", *The Bulletin of the Research Council of Israel*, vol. 9F, 1–16. Reprinted in Bar-Hillel 1964.

Buszkowski, W., 1987a, "Solvable problems for classical categorial grammars", *Bulletin of the Polish Academy of Sciences: Mathematics* **35**, 373–382.

Buszkowski, W., 1987b, "Discovery procedures for categorial grammars", in *Categories, Polymorphism and Unification*, E. Klein and J. van Benthem, eds., University of Amsterdam.

Buszkowski, W., 1988, "Generative power of categorial grammars", in *Categorial Grammars and Natural Language Structures*, R. T. Oehrle, E. Bach, and D. Wheeler, eds., Dordrecht: Reidel.

Buszkowski, W. and Penn, G., 1990, "Categorial grammars determined from linguistic data by unification", *Studia Logica* **49**, 431–454.

Chomsky, N., 1986, *Knowledge of Language: Its Nature, Origin, and Use*, New York: Praeger.

Dowty, David. 1988. Type raising, functional composition, and non-constituent conjunction. In Richard T. Oehrle, Emmon Bach, and Deirdre Wheeler, eds., *Categorial Grammars and Natural Language Structures*. Dordrecht: Reidel.

Gold, M. E., 1967, "Language identification in the limit", *Information and Control* **10**, 447–474.

de Jongh, D. and Kanazawa, M., 1995, *Learnability Theory*, course material presented at the Seventh Summer School in Logic, Language and Information, University of Barcelona, August 1995.

Kanazawa, M., 1994a, *Learnable Classes of Categorial Grammars*, Ph.D. dissertation, Stanford University. Available as ILLC Dissertation Series 1994–8, Institute for Logic, Language, and Computation, University of Amsterdam (`illc@fwi.uva.nl`).

Kanazawa, M., 1994b, "A note on language classes with finite elasticity", Reprot CS–R9471, CWI, Amsterdam.

Kapur, S., 1991, *Computational Learning of Languages*, Ph.D. dissertation, Cornell University. Available as Technical Report 91–1234, Department of Computer Science, Cornell University.

Kearns, M. J. and Vazirani, U. V., 1994, *An Introduction to Computational Learning Theory*, Cambridge, Mass.: MIT Press.

* Equality holds when $|G| = 1$. Otherwise it can be shown that $|P_2| \leq 2|G| - 2$.

Lambek, J., 1958, "The mathematics of sentence structure", *American Mathematical Monthly* **65**, 154–170. Reprinted in *Categorial Grammars*, W. Buszkowski, W. Marciszewski, and J. van Benthem, eds., Amsterdam: John Benjamins, 1988.

Lambek, J., 1961, "On the calculus of syntactic types", in *Structure of Language and its Mathematical Aspects*, R. Jakobson, ed., Providence, R.I.: American Mathematical Society.

Lassez, J.-L., M. J. Maher, and K. Marriott. 1988. Unification revisited. In J. Minker, ed., *Foundations of Deductive Databases and Logic Programming*, pp. 587–625. Los Altos, Calif.: Morgan Kaufmann.

Levy, Leon S. and Aravind K. Joshi. 1978. Skeletal structural descriptions. *Information and Control* **39**, 192–211.

Montague, R., 1973, "The proper treatment of quantification in ordinary English", reprinted in *Formal Philosophy: Selected Papers of Richard Montague*, R. H. Thomason, ed., New Haven: Yale University Press, 1974.

Moriyama, T. and Sato, M, 1993, "Properties of language classes with finite elasticity", in *Algorithmic Learning Theory*, Proceedings, 1993, K. P. Jankte, S. Kobayashi, E. Tomita, and T. Yokomori, eds., Lecture Notes in Artificial Intelligence 744, Berlin: Springer.

Motoki, T., Shinohara, T., and Wright, K., 1991, "The correct definition of finite elasticity: Corrigendum to identification of unions", p. 375 in *The Fourth Annual Workshop on Computational Learning Theory*, San Mateo, CA: Morgan Kaufmann.

Osherson, D., Stob, M., and Weinstein, S., 1986, *Systems That Learn*, Cambridge, MA: MIT Press.

Osherson, D., Weinstein, S., de Jongh, D., and Martin, E., 1994, "Formal learning theory", to appear in *Handbook of Logic and Language*, J. van Benthem and A. ter Meulen, eds.

Sakakibara, Y., 1992, "Efficient learning of context-free grammars from positive structural examples", *Information and Computation* **97**, 23–60.

Shinohara, T., 1990a, "Inductive inference from positive data is powerful", pp. 97–110 in *The 1990 Workshop on Computational Learning Theory*, San Mateo, CA: Morgan Kaufmann.

Shinohara, T., 1990b, "Inductive inference of monotonic formal systems from positive data", pp. 339–351 in *Algorithmic Learning Theory*, S. Arikawa, S. Goto, S. Ohsuga, and T. Yokomori, eds., Tokyo: Ohmsha, and New York and Berlin: Springer.

Steedman, Mark J. 1985. Dependency and coordination in the grammar of Dutch and English. *Language* **61**, 523–568.

Steedman, Mark J. 1987. Combinatory grammars and parasitic gaps. *Natural Language and Linguistic Theory* **5**, 403–440.

Steedman, Mark J. 1988. Combinators and grammars. In Richard T. Oehrle, Emmon Bach, and Deirdre Wheeler, eds., *Categorial Grammars and Natural Language Structures*. Dordrecht: Reidel.

Wexler, K. and Culicover, P., 1980, *Formal Principles of Language Acquisition*, Cambridge, MA: MIT Press.

Wright, K., 1989, "Identification of unions of languages drawn from an identifiable class", pp. 328–333 in *The 1989 Workshop on Computational Learning Theory*, San Mateo, CA: Morgan Kaufmann.