Chapter 4

Learning Classes of Categorial Grammars

In this chapter various classes of classical categorial grammars are presented, and the results from Kanazawa (1998) concerning learnability of these classes will be discussed. Also, an open question from this book is answered: the set-driven learning functions $\varphi_{\text{VG}}$ and $\varphi_{\text{LVG}}$ are shown not to be conservative in Section 4.3.

In Chapter 3 the class of classical categorial grammars was associated with the grammar systems $(\text{CatG}, \Sigma^+, L)$ and $(\text{CatG}, \Sigma^F, \text{FL})$. These systems correspond to two different models for learning categorial grammars. Using the former grammar system, the learning function receives as input sequences of (non-empty) strings over $\Sigma$. Using the latter system, the learning function receives as input sequences of functor-argument structures over $\Sigma$.

A class $\mathcal{G}$ of grammars is called learnable from strings if it is learnable with respect to the former grammar system. If it is learnable with respect to the latter grammar system it is called learnable from structures.

Since a functor-argument structure provides more information than its yield, learnability from structures may seem trivial. This is not the case, however: Gold’s theorem implies that the class CatG of all categorial grammars (over a given $\Sigma$) is learnable neither from strings nor from structures. The notions of learnability from strings and learnability from structures are, in principle, logically independent. When learning from structures, the criteria for successful learning are more strict than when learning from strings: the learning function is required to converge to a grammar $G$ that generates exactly the structures that appear in the input sequence. This is more strict than the requirement that the string language of $G$ contain exactly the yields of the structures in the input sequence. Therefore, learnability from strings does not a priori imply learnability from structures. Learning from strings will be discussed in Section 4.8.
4.1 Learning Rigid Grammars: the Algorithm RG

Definition 4.1 A rigid grammar is a partial function from $\Sigma$ to $Tp$. It assigns either zero or one type to each symbol in the alphabet.

Definition 4.2 We write $\mathcal{G}_{\text{rigid}}$ to denote the class of rigid grammars over $\Sigma$. The class $\{FL(G) \mid G \in \mathcal{G}_{\text{rigid}}\}$ is denoted $\mathcal{FL}_{\text{rigid}}$.

Buszkowski’s algorithm for learning rigid grammars (see Buszkowski (1987a), Buszkowski and Penn (1990)), which Kanazawa calls RG, takes a finite set of functor-argument structures ($D$) as input and yields either a rigid grammar or the empty set, in case $D$ is no sublanguage of any language generated by a rigid grammar. This algorithm relies on unification of types.

Algorithm RG

- **input:** a finite set $D$ of functor-argument structures.
- **output:** a rigid grammar $G$ such that $D \subseteq FL(G)$, if it exists.

The first step of this algorithm is called GF. It maps a structure to a finite set of type assignments. For this, the following rules are applied$^2$:

Algorithm GF

- **input:** a finite set $D$ of functor-argument structures.
- **output:** a grammar $G$ such that $D = FL(G)$.

$$c \mapsto X \leadsto \{c \mapsto X\}$$

$$\text{ba}(c_1, c_2) \mapsto X \leadsto \begin{cases} c_1 \mapsto Y \\ c_2 \mapsto Y\backslash X \end{cases}$$

$$\text{fa}(c_2, c_1) \mapsto X \leadsto \begin{cases} c_1 \mapsto Y \\ c_2 \mapsto X/Y \end{cases}$$

$^1$For a comprehensive overview of unification theory the reader is referred to Baader and Siekmann (1993). Generalizations of unification of types have been considered in the context of learning categorial grammars, see Buszkowski (1995); Marciniec (1994, 1997, 1996) for e.g. unification with negative constraints and unification of infinite sets of types.

As noted in Chapter 3, the structure languages under discussion are regular tree languages. These are generalizations of regular languages, and they are commonly represented by tree automata, which are generalizations of finite state automata. It should not come as a surprise then that unification in this context is a generalization of a technique for inducing finite state automata known as state merging. It is known to be an efficient technique, see e.g. Angluin (1982); Oncina and Garcia (1992); Lang (1992); Lang et al. (1998).

$^2$Here the symbol $\leadsto$ is taken to mean ‘yields, after application of the function GF…’.
4.1. LEARNING RIGID GRAMMARS: THE ALGORITHM RG

The type assignments obtained in this way are collected into a grammar. This grammar is the general form determined by $D$. Note that this grammar can be redundant. To obtain a rigid grammar, the types assigned to the same symbol must be unified.

Let $A = \{ \{ A \mid GF(D): c \mapsto A \} \mid c \in \text{dom}(GF(D)) \}$, and compute $\sigma = \text{mgu}(A)$. If unification fails, the algorithm also fails. The substitution is then applied to the general form, so $RG(D) = \sigma[GF(D)]$.

In Buszkowski and Penn (1990) some of the more important properties of RG have been investigated:

Lemma 4.3 $FL(GF(D)) = D$.

Proposition 4.4 $RG(D)$, if it exists, is in reduced form.

Kanazawa’s results concerning the learnability properties of RG are the following:

Theorem 4.5 The class $\mathcal{F}_{\text{rigid}}$ has finite elasticity.

In fact, Kanazawa’s proof of this Theorem shows that the length of any ‘elasticity chain’ in this class is bounded. One of the key arguments in his proof is stated by the following lemma:

Lemma 4.6 Let $G_0, \ldots, G_n$ be rigid grammars over $\Sigma$ without useless types such that $G_0 \sqsupset \ldots \sqsupset G_n$. Then $n \leq |\Sigma|$.

Definition 4.7 Let $\varphi_{RG}$ be the learning function for the grammar system \langle CatG, $\Sigma^F$, FL $\rangle$ defined as follows $^3$:

$\varphi_{RG}(\langle T_0, \ldots, T_n \rangle) \simeq RG(\{T_0, \ldots, T_n\})$.

Theorem 4.8 $\varphi_{RG}$ learns $G_{\text{rigid}}$ from structures.

Proposition 4.9 $\varphi_{RG}$ has the following desirable properties:

- $\varphi_{RG}$ learns $G_{\text{rigid}}$ prudently.
- $\varphi_{RG}$ is responsive and consistent on $G_{\text{rigid}}$.
- $\varphi_{RG}$ is set-driven.
- $\varphi_{RG}$ is conservative.
- $\varphi_{RG}$ is monotone increasing.
- $\varphi_{RG}$ is incremental.
- $\varphi_{RG}$ runs in linear time.$^4$

$^3$X $\approx$ Y means ‘X and Y are both defined and equal, or both are undefined’.

$^4$Note that Kanazawa does not claim this for a function, but for a particular algorithm. Since RG can be implemented by an algorithm that unifies at most $n$ types (where $n$ is the number of symbol occurrences in the input) it runs in time linear in $|D|$, the size of the input.
48 CHAPTER 4. LEARNING CLASSES OF CATEGORIAL GRAMMARS

4.2 Learning $k$-Valued Grammars

Restricting a learning function to the class of rigid grammars seems rather unfortunate: if the input enumerates a structure language of a grammar that assigns more than one type to a symbol, $\mathcal{RG}$ either fails or overgeneralizes.

RG can easily be modified to deal with these cases. This modified RG can be used in an algorithm that learns the class $\mathcal{G}_{k\text{-valued}}$ of $k$-valued grammars.

**Definition 4.10** A $k$-valued grammar is a partial function from $\Sigma$ to the powerset of $Tp$. It assigns at most $k$ types to each symbol in the alphabet.

**Definition 4.11** We write $\mathcal{G}_{k\text{-valued}}$ to denote the class of $k$-valued grammars over $\Sigma$. The class $\{\text{FL}(G) \mid G \in \mathcal{G}_{k\text{-valued}}\}$ is denoted $\mathcal{FL}_{k\text{-valued}}$.

**Theorem 4.12** (Hierarchy Theorem)
For each $k \in \mathbb{N}, \mathcal{L}_{k\text{-valued}} \subset \mathcal{L}_{k+1\text{-valued}}$.

**Corollary 4.13** For each $k \in \mathbb{N}, \mathcal{FL}_{k\text{-valued}} \subset \mathcal{FL}_{k+1\text{-valued}}$.

The algorithm is based on a generalization of unification that is called $k$-partial unification.

**Definition 4.14** let $\mathcal{A}$ be a family of sets of types. A substitution $\sigma$ is called a $k$-partial unifier of $\mathcal{A}$ if and only if for each $A \in \mathcal{A}, |\sigma(A) \mid A \in \mathcal{A}| \leq k$.

**Definition 4.15** Let $\mathcal{A} = \{A_1, \ldots, A_n\}$ be a family of sets of types.

- If $\mathcal{B}_i = \{B_{i,1}, \ldots, B_{i,l_i}\}$ is a partition of $A_i$ for $1 \leq i \leq n$, then the family
  $$\mathcal{B} = \bigcup \{\mathcal{B}_i \mid 1 \leq i \leq n\}$$
  is called a partition of $\mathcal{A}$; $\mathcal{B}$ is called a $k$-partition of $\mathcal{A}$ if $l_i \leq k$ for each $i$.

- Let $\sigma$ be a substitution. The equivalence relation
  $$\sigma(A) = \sigma(B)$$
  on types determines a partition $\mathcal{B}_i$ of $A_i$ for each $i$. then
  $$\mathcal{B} = \bigcup \{\mathcal{B}_i \mid 1 \leq i \leq n\}$$
  is called the partition of $\mathcal{A}$ induced by $\sigma$.

**Definition 4.16** If $\mathcal{A}$ is a finite family of finite sets of types, let
  $$\text{PU}_k(\mathcal{A}) = \{\text{mgu}(\mathcal{B}) \mid \mathcal{B} \text{ is a } k\text{-partition of } \mathcal{A}\}.$$
4.2. LEARNING K-VALUED GRAMMARS

The function $PU_k$ is computable in exponential time.

**Proposition 4.17** Let $A$ be a finite family of finite sets of types. Then, for every $k$-partial unifier $\sigma$ of $A$, there is some $k$-partial unifier $\sigma_0$ in $PU_k(A)$ such that $\sigma_0$ is more general than $\sigma$.

**Algorithm VG$_k$.**

- **Input:** A finite set $D$ of functor-argument structures.
- **Output:** A finite set $G$ of $k$-valued grammars such that for each $G \in G$, $D \subseteq \text{FL}(G)$.

**Step 1.** Algorithm RG: construct $GF(D)$.

**Step 2.** Let $A = \{ \{A \mid GF(D): c \mapsto A \} \mid c \in \text{dom}(GF(D)) \}$ and compute $PU_k(A)$.

**Step 3.** Let $VG_k(D) = \{ \sigma[GF(D)] \mid \sigma \in PU_k(A) \}$. This is the output of the algorithm.

**Proposition 4.18** Let $G$ be a $k$-valued grammar. Then the following are equivalent:

1. $D \subseteq \text{FL}(G)$.
2. There exists a grammar $G' \in VG_k(D)$ such that $G' \sqsubseteq G$.

**Proposition 4.19** For $k \geq 2, \{G \in \mathcal{G}_{k-\text{valued}} \mid G \text{ is in reduced form} \} \subseteq \bigcup \text{range}(VG_k)$.

In other words, VG$_k$ is ‘messy’: it may produce grammars that are not in reduced form. When it does, it will also produce the reduced version of such a grammar, since a grammar is of the same size, or larger than, its reduced version.

**Proposition 4.20** *(Stated without proof in Kanazawa (1998))* If $G \in VG_k(D)$ and $T \in \text{FL}(G)$, then $G \in VG_k(D \cup \{T\})$.

We could call this ‘conservativity of grammar class’.

Since it is possible that $|VG_k(D)| > 1$, we need a way to select a grammar from such a set. In fact, $G_{\text{rigid}}$ is unique in that $|RG(D)| = 1$ if defined. A learning function based on any of the other classes we will examine needs a selection function.

**Proposition 4.21** Let $D$ be a finite set of functor-argument structures. Then the set of minimal elements of $\{L \in \text{FL}_{k-\text{valued}} \mid D \subseteq L \}$ is included in $\{\text{FL}(G) \mid G \in VG_k(D)\}$. 
This, and the decidability of the question \( \text{FL}(G_1) \subseteq \text{FL}(G_2) \)?, makes it possible to find a grammar for a minimal element of \( \{ L \in \mathcal{F}_k \text{-valued} | D \in L \} \) given any \( D \) as input.

4.2.1 Learning Functions Based on \( \text{VG}_k \)

**Definition 4.22** Let \( \mu_{\text{FL}} \) be a (computable) function that maps a non-empty finite set \( G \) of grammars to a grammar \( G \in \mathcal{G} \) such that \( \text{FL}(G) \) is a minimal element of \( \{ \text{FL}(G) | G \in \mathcal{G} \} \).

**Proposition 4.23** For any finite set \( D \subseteq \Sigma_F \), if \( \mu_{\text{FL}}(\text{VG}_k(D)) \) is defined, then \( \text{FL}(\mu_{\text{FL}}(\text{VG}_k(D))) \) is a minimal element of \( \{ L \in \mathcal{F}_k \text{-valued} | D \subseteq L \} \).

**Definition 4.24** Let \( \varphi_{\text{VG}_k} \) be the learning function for \( (\text{CatG}, \Sigma_F, \text{FL}) \) defined as follows:

\[
\begin{align*}
\varphi_{\text{VG}_k}((T_0)) &= \mu_{\text{FL}}(\text{VG}_k(\{T_0\})), \\
\varphi_{\text{VG}_k}((T_0, \ldots, T_{i+1})) &= \begin{cases} \\
\varphi_{\text{VG}_k}((T_0, \ldots, T_i)) & \text{if } T_{i+1} \in \text{FL}(\varphi_{\text{VG}_k}((T_0, \ldots, T_i))), \\
\mu_{\text{FL}}(\text{VG}_k(\{T_0, \ldots, T_{i+1}\})) & \text{otherwise}. \\
\end{cases}
\end{align*}
\]

This is a construction that is guaranteed to be conservative: it ignores input that fits into the current hypothesis\(^5\). Only if input is not compatible with the current hypothesis (i.e., is not in the structure language of the former output grammar), a new hypothesis is considered. The learning function based on \( \mu_{\text{FL}} \) and some class of grammars may not be inherently conservative.

**Proposition 4.25**

1. \( \varphi_{\text{VG}_k} \) is responsive and consistent on \( \mathcal{G}_k \text{-valued} \).
2. \( \varphi_{\text{VG}_k} \) is conservative.
3. \( \varphi_{\text{VG}_k} \) learns \( \mathcal{G}_k \text{-valued} \) prudently.

**Theorem 4.26** \( \varphi_{\text{VG}_k} \) learns \( \mathcal{G}_k \text{-valued} \) from structures.

The function \( \varphi_{\text{VG}_k} \) is not designed to be set-driven or even to learn order-independently. Kanazawa defines a set-driven learning function \( \varphi_{\text{VG}_i}^\delta \):

**Definition 4.27**

\[
\varphi_{\text{VG}_i}^\delta((T_0, \ldots, T_i)) = \mu_{\text{FL}}(\text{VG}_i(\{T_0, \ldots, T_i\})),
\]

where \( \mu_{\text{FL}} \) is defined as follows:

\(^5\)Note that it is only ignored ‘locally’. Once input does not fit, the input that was formerly ignored is taken into account when constructing a new hypothesis.
4.2. LEARNING K-VALUED GRAMMARS

**Definition 4.28** Let $\mu_{FL}$ be a computable total function that maps a finite set $G$ of grammars to the first element of 
\{ $G \in G$ | FL($G$) is a minimal element of FL($G$) \} under the ordering $\prec$.

Here, $\prec$ is defined by:

**Definition 4.29** Let $\prec$ be a computable well-order on Cat$G$ such that $G_1 \prec G_2$ whenever one of the following conditions holds:

1. $\text{size}(G_1) < \text{size}(G_2)$.
2. $\text{size}(G_1) = \text{size}(G_2)$ and $|\text{Var}(G_1)| > |\text{Var}(G_2)|$.
3. $\text{size}(G_1) = \text{size}(G_2)$ and $|\text{Var}(G_1)| = |\text{Var}(G_2)|$, then $G_1 \prec G_2$ by some arbitrary lexicographic ordering of grammars.

The size of a grammar is defined as:

**Definition 4.30** For any grammar $G$, define the size of $G$, $\text{size}(G)$, as follows:

$$\text{size}(G) = \sum_{c \in \Sigma} \sum_{c \rightarrow A} |A|$$

where for each type $A$, $|A|$ is the number of symbol occurrences in $A$.

**Lemma 4.31** If $G_1 \subseteq G_2$, then $\text{size}(G_1) \leq \text{size}(G_2)$.

**Lemma 4.32** $G_1 \subseteq G_2$ implies $G_1 \prec G_2$.

Unfortunately the reverse does not hold. If it would, $\subseteq$ and $\prec$ would obviously be equivalent, greatly simplifying matters. We can, however, use $\prec$ instead of $\subseteq$. Kanazawa leaves this implicit since it is quite easy to see, but we have decided to show this because it is an important point. The following is an easy consequence of lemma 4.32:

**Corollary 4.33** $G_1 \prec G_2$ implies $\neg(G_2 \subseteq G_1)$

**Proof:** By lemma 4.32, $\neg(G_2 \prec G_1)$ implies $\neg(G_2 \subseteq G_1)$. Since $\prec$ is a well-order, $G_1 \prec G_2$ implies $\neg(G_2 \prec G_1)$. $\square$

---

6Even though it is not clear why Kanazawa chose this particular ordering, this definition suggests this adapted version of $\mu_{FL}$ is intended to pick the ‘simplest’ (in an informal sense) grammar from a set.

There exists a learning strategy called simplcity (see Osherson et al. (1986)) that constrains learning functions so as not to conjecture grammars that are arbitrarily more complex than simpler alternatives for the same language. For any size measure and any bound on the size difference between conjecture and simpler alternative, this strategy severely restricts the class of learnable languages, in the recursive case.
Proposition 4.34 Let \((T_i)_{i \in \mathbb{N}}\) be an infinite sequence enumerating some \(L \in \mathcal{H}_{k,\text{valued}}\). Then \(\varphi^{\perp}_{\mathbb{V}G_k}\) converges on \((T_i)_{i \in \mathbb{N}}\) to the first element of
\[
\mathcal{G}_L = \{G \in \mathbb{G}_{k,\text{valued}} \mid \text{FL}(G) = L\}
\]
under the ordering \(\prec\).

4.3 \(\varphi^{\perp}_{\mathbb{V}G_k}\) is not conservative

While \(\varphi^{\perp}_{\mathbb{V}G_k}\) is set-driven, Kanazawa left it an open question whether it is conservative.

I have constructed a proof of non-conservativity of \(\varphi^{\perp}_{\mathbb{V}G_k}\) that was inspired by a footnote on page 102 of Kanazawa (1998):

Although I have not had time to prove that \(\varphi^{\perp}_{\mathbb{V}G_k}\) is indeed nonconservative, it is conceivable that the following sort of situation can occur, and implies the following:

To summarize, the following conditions are sufficient conditions for \(\varphi^{\perp}_{\mathbb{V}G_k}\) to be non-conservative:

1. \(G_0, G_1 \in \mathbb{V}G_k(D), \text{FL}(G_0)\) is minimal, but \(\text{FL}(G_1)\) is not minimal in \(\text{FL}(\mathbb{V}G_k(D))\), \(G_1 \prec G_0\), \(\varphi^{\perp}_{\mathbb{V}G_k}(D) = G_0\).

2. \(G_0, G_1 \in \mathbb{V}G_k(D \cup \{T\}), \text{FL}(G_1)\) is minimal in \(\text{FL}(\mathbb{V}G_k(D \cup \{T\}))\), \(\varphi^{\perp}_{\mathbb{V}G_k}(D \cup \{T\}) = G_1\).

It turns out that such a situation can occur, and implies the following:

There is a finite (possibly empty) set of grammars \(FG \subset \mathbb{V}G_k(D)\) such that all \(G \in FG\), \(\text{FL}(G) \not\subseteq \text{FL}(G_0)\) and \(\text{FL}(G_0) \not\subseteq \text{FL}(G)\), and \(G_0 \prec G\). For all \(G \in \mathbb{V}G_k(D) - FG\), \(\text{FL}(G_0) \not\subseteq \text{FL}(G)\).

Since \(G_1 \prec G_0\) and \(\mu_{\text{FL}}(\mathbb{V}G_k(D)) = G_0\), \(G_1\) is not minimal in \(\mathbb{V}G_k(D)\), so there is a grammar \(G_2 \mathbb{V}G_k(D)\) such that \(\text{FL}(G_2) \subset \text{FL}(G_1)\) and \(G_0 \prec G_2\).

Moreover, \(G_2\) cannot be in \(\mathbb{V}G_k(D \cup \{T\})\). If it would, \(\mu_{\text{FL}}\) would choose \(G_2\) over \(G_1\), so condition 2 would be impossible. So we have \(\lnot(\text{FL}(G_2) \subset \text{FL}(G_0))\), and \(G_0 \prec G_2\), so \(G_1 \prec G_0 \prec G_2\).

By Proposition 4.20, since \(G_2 \in \mathbb{V}G_k(D)\) and \(G_2 \not\in \mathbb{V}G_k(D \cup \{T\})\), \(T \not\in \text{FL}({G_2})\).

Since for any \(G \in \mathbb{V}G_k(D)\), \(D \subset \text{FL}(G)\), \(\{T\} \subset \text{FL}(G_0)\), \(\{T\} \not\subset \text{FL}(G_1)\).\footnote{An earlier version of this proof first appeared in Costa Florêncio (2001c). Makoto Kanazawa has pointed out (personal communication) that this proof could be simplified, making it easier to be verified by hand, and making it possible to be verified by the Prolog implementation as given in the appendix of Kanazawa (1998). The proof presented here is thus an alternative version and is due to Kanazawa.}
Proposition 4.35 The set-driven learning function $\varphi_{VG_k}$ is non-conservative.

Proof: By example: The initial sample to be considered is the set consisting of the following functor-argument structures:

\[
\begin{align*}
&ba(fa(a,fa(b,fa(x,x))),g) \quad g \\
&ba(fa(fa(y,y),fa(fa(y,y),fa(x,x))),g) \quad ba(a,e) \\
&ba(b,a) \quad e \\
&ba(fa(z,z),a) \quad j \\
&ba(fa(z,z),fa(w,w)) \quad ba(fa(y,y),c) \\
&ba(fa(w,w),c) \quad ba(d,e) \\
&ba(fa(z,z),d) \quad ba(fa(a,fa(f,fa(x,x))),g) \\
&ba(f,e)
\end{align*}
\]

It’s useful to note that with a 2-valued grammar, a structure of the form $fa(x,x)$ (or $ba(x,x)$, for that matter) must be assigned the same type wherever it occurs. The Prolog implementation outputs the following three grammars. Note that they differ only in the types assigned to $a$, $b$, $d$, $e$, and $f$.

\[
G_1:
\begin{align*}
a & \mapsto B/B, D\setminus E \\
b & \mapsto B/B \\
c & \mapsto (B/B)\setminus t, (D\setminus E)\setminus t \\
d & \mapsto B/B, D\setminus E \\
e & \mapsto (B/B)\setminus t, t \\
f & \mapsto B/B \\
g & \mapsto B/t, t \\
j & \mapsto E\setminus t, t \\
w & \mapsto (D\setminus E)/W,W \\
x & \mapsto B/X, X \\
y & \mapsto (B/B)/Y, Y \\
z & \mapsto D/Z, Z
\end{align*}
\]

\[
G_2:
\begin{align*}
a & \mapsto B/C, D\setminus E \\
b & \mapsto B/C, C/B \\
c & \mapsto (B/B)\setminus t, (D\setminus E)\setminus t \\
d & \mapsto B/C, D\setminus E \\
e & \mapsto (B/C)\setminus t, t \\
f & \mapsto B/C, C/B \\
g & \mapsto B/t, t \\
j & \mapsto E\setminus t, t \\
w & \mapsto (D\setminus E)/W,W \\
x & \mapsto B/X, X \\
y & \mapsto (B/B)/Y, Y \\
z & \mapsto D/Z, Z
\end{align*}
\]

\[
G_3:
\begin{align*}
a & \mapsto B/C, D\setminus E \\
b & \mapsto C/B, D\setminus E \\
c & \mapsto (B/B)\setminus t, (D\setminus E)\setminus t \\
d & \mapsto D\setminus E \\
e & \mapsto (D\setminus E)\setminus t, t \\
f & \mapsto C/B, D\setminus E \\
g & \mapsto B/t, t \\
j & \mapsto E\setminus t, t \\
w & \mapsto (D\setminus E)/W,W \\
x & \mapsto B/X, X \\
y & \mapsto (B/B)/Y, Y \\
z & \mapsto D/Z, Z
\end{align*}
\]
Note that $G_1$ is the result of unifying $C$ with $B$ in $G_2$, and $FL(G_1)$ properly includes $FL(G_2)$. $FL(G_2)$ and $FL(G_3)$ are incomparable and $size(G_2) > size(G_3)$, so the set-driven learning function based on $<$ picks $G_3$ for this sample.

Now consider adding $\text{ba}(b,c)$ to the above sample and see what grammars are output:

$$G'_1:$$

- $a \mapsto B/B, D\setminus E$
- $b \mapsto B/B$
- $c \mapsto (B/B)\setminus t, (D\setminus E)\setminus t$
- $d \mapsto B/B, D\setminus E$
- $e \mapsto (B/B)\setminus t, t$
- $f \mapsto B/B$
- $g \mapsto B\setminus t, t$
- $j \mapsto E\setminus t, t$
- $w \mapsto (D\setminus E)/W, W$
- $x \mapsto B/X, X$
- $y \mapsto (B/B)/Y, Y$
- $z \mapsto D/Z, Z$

$$G'_2:$$

- $a \mapsto B/B, D\setminus E$
- $b \mapsto B/B, D\setminus E$
- $c \mapsto (B/B)\setminus t, (D\setminus E)\setminus t$
- $d \mapsto B/B, D\setminus E$
- $e \mapsto (B/B)\setminus t, t$
- $f \mapsto B/B$
- $g \mapsto B\setminus t, t$
- $j \mapsto E\setminus t, t$
- $w \mapsto (D\setminus E)/W, W$
- $x \mapsto B/X, X$
- $y \mapsto (B/B)/Y, Y$
- $z \mapsto D/Z, Z$

$$G'_3:$$

- $a \mapsto B/B, D\setminus E$
- $b \mapsto B/B, D\setminus E$
- $c \mapsto (B/B)\setminus t, (D\setminus E)\setminus t$
- $d \mapsto B/B, D\setminus E$
- $e \mapsto (B/B)\setminus t, t$
- $f \mapsto B/B$
- $g \mapsto B\setminus t, t$
- $j \mapsto E\setminus t, t$
- $w \mapsto (D\setminus E)/W, W$
- $x \mapsto B/X, X$
- $y \mapsto (B/B)/Y, Y$
- $z \mapsto D/Z, Z$

$$G'_4:$$

- $a \mapsto B/C, D\setminus E$
- $b \mapsto C/B, D\setminus E$
- $c \mapsto (B/B)\setminus t, (D\setminus E)\setminus t$
- $d \mapsto B/B, D\setminus E$
- $e \mapsto (D\setminus E)\setminus t, t$
- $f \mapsto C/B, D\setminus E$
- $g \mapsto B\setminus t, t$
- $j \mapsto E\setminus t, t$
- $w \mapsto (D\setminus E)/W, W$
- $x \mapsto B/X, X$
- $y \mapsto (B/B)/Y, Y$
- $z \mapsto D/Z, Z$

Note that $G'_1$ is the same as $G_1$, and $G'_4$ is the same as $G_3$. $G'_2$ is the result of unifying $C$ with $B$ in $G'_4$, and $G'_3$ is $G'_1$ plus one additional type assignment: $b \mapsto D\setminus E$. So $FL(G'_2)$ properly includes $FL(G'_4)$ and $FL(G'_3)$ properly includes $FL(G'_4)$. $FL(G'_1)$ and $FL(G'_4)$ are incomparable. But $size(G'_1) < size(G'_4)$, so $G'_1$, not $G'_4 (= G_3)$, is the grammar picked by the set-driven learning function for this expanded sample. \[\square\]
4.4 LEAST-VALUED GRAMMARS

4.4 Least-Valued Grammars

The class of \(k\)-valued grammars suffers from a major problem; \(\text{VG}_k(D)\) is not defined for all \(D\). There is a simple way to solve this problem: define a class based on \(\text{VG}_k\) where \(k\) is always the minimal \(k\) for which \(\text{VG}_k(D)\) is defined. Let us call this the class of least-valued grammars.

Definition 4.36 (Definition 6.49 from Kanazawa (1998)) Let \(L \subseteq \Sigma^p\). A grammar \(G \in \mathcal{G}_{k+1}\)-valued \(-\mathcal{G}_k\)-valued is called least-valued with respect to \(L\) if \(L \subseteq \text{FL}(G)\) and there is no \(G' \in \mathcal{G}_k\)-valued such that \(L \subseteq \text{FL}(G')\).

Definition 4.37 If \(G \in \text{LVG}(D)\), then \(G\) is least-valued with respect to \(D\).

Definition 4.38 A grammar \(G\) is called a least-valued grammar if it is least-valued with respect to \(\text{FL}(G)\).

Definition 4.39 We write \(\mathcal{G}_{\text{least-valued}}\) to denote the class of least-valued grammars over \(\Sigma\). The class \(\{\text{FL}(G) \mid G \in \mathcal{G}_{\text{least-valued}}\}\) is denoted \(\mathcal{F}_{\text{least-valued}}\).

Definition 4.40 Algorithm for LVG

- **input:** A finite set \(D\) of functor-argument structures.
- **output:** A finite set of \(k\)-valued grammars \(G\) such that \(D \subseteq \text{FL}(G)\) for the least \(k\) such that \(D\) is a subset of some \(L \in \mathcal{F}_{k\text{-valued}}\).

Set \(k := 0\).
Input \(D\).
While \(\text{VG}_k(D) = \emptyset\) do

Set \(k := k + 1\).
Let \(\text{LVG}(D) = \text{VG}_k(D)\).

Thus, \(\text{LVG}\) finds the least \(k\) such that \(\text{VG}_k(D) \neq \emptyset\) and outputs \(\text{VG}_k(D)\). This makes \(\text{VG}_k(D) \neq \emptyset\) for any \(D\).

Proposition 4.41 \(\{G \in \mathcal{G}_{\text{least-valued}} \mid G\ is\ in\ reduced\ form\} \subset \text{range}(\text{LVG})\).

Let \(\varphi_{\text{LVG}}\) be defined as \(\varphi_{\text{VG}_k}\) in Definition 4.24, with \(\text{VG}_k\) replaced by \(\text{LVG}\).

Proposition 4.42 The learning function \(\varphi_{\text{LVG}}\) learns \(\mathcal{G}_{\text{least-valued}}\) from structures.

- \(\varphi_{\text{LVG}}\) is responsive and consistent on \(\mathcal{G}_{\text{least-valued}}\).
- \(\varphi_{\text{LVG}}\) is conservative.
It is possible to define an alternative learning function $\varphi_{LVG}^k$ analogous to Definition 4.27 that is set-driven. It converges in a way entirely analogous to $\varphi_{VG}^k$ as stated in Proposition 4.34.

**Proposition 4.43** The set-driven learning function $\varphi_{LVG}^k$ is non-conservative.

**Proof:** In the proof of 4.35, a grammar $G$ is presented that is the general form of sample $D$. In this grammar, both a type of the form $\cdot \cdot \cdot \cdot$ and a type of the form $\cdot \cdot \cdot \cdot \cdot$ are assigned to the symbol $\cdot$. Clearly, these two types are not unifiable, so the least value for $k$ for which $\varphi_{VG}^k(D)$ is defined is at least 2. In the proof, for all the grammars that have to be considered by $\mu (G_1', G_2'$ and $G_3')$, $k = 2$. Thus, the proof works for $\varphi_{LVG}^k$ as well. \(\square\)

### 4.5 Optimal Grammars

Another extension of RG proposed by Buszkowski and Penn is the class of optimal grammars. The algorithm for generating this class, OG, is based on a generalization of unification called optimal unification.

**Definition 4.44** We write $G_{\text{optimal}}$ to denote the class of optimal grammars over $\Sigma$. The class $\{\text{FL}(G) \mid G \in G_{\text{optimal}}\}$ is denoted $\mathcal{R}_{\text{optimal}}$.

**Definition 4.45** Let $A = \{A_1, \ldots, A_n\}$ be a family of sets of types. A substitution $\sigma$ is called an optimal unifier of $A$ if the following holds:

1. $\sigma$ is a most general unifier of the partition of $A$ induced by $\sigma$.
2. For all $A_i \in A$ and for all $A, B \in A_i$, if $\sigma(A) \neq \sigma(B)$, then $\{\sigma(A), \sigma(B)\}$ has no unifier.

An optimal unifier of $A$ unifies a substitution that unifies $A$ ‘as much as possible’. Note that this means that no grammar $G, G \in G_{\text{optimal}}$ is redundant.

**Definition 4.46** Let $B$ and $C$ be partitions of $A$. $B$ is said to be coarser than $C$ if $C$ is a partition of $B$. We say that $B$ is strictly coarser than $C$ if $B$ is coarser than $C$ but not vice versa.

**Definition 4.47** Let $A$ be a family of sets of types. A partition $B$ of $A$ is said to be optimal if the following conditions hold:

1. $B$ has a unifier
2. No partition $C$ of $A$ strictly coarser than $B$ has a unifier.

**Proposition 4.48** Let $A$ be a family of sets of types. A substitution $\sigma$ is an optimal unifier of $A$ if and only if $\sigma$ is a most general unifier of some optimal partition of $A$. 
4.6. LEAST CARDINALITY GRAMMARS

**Definition 4.49** If \( \mathcal{A} \) is a finite family of finite sets of types, define

\[
\text{OU}(\mathcal{A}) = \{ \text{mgu}(\mathcal{B}) \mid \mathcal{B} \text{ is an optimal partition of } \mathcal{A} \}.
\]

Obviously, if \( \mathcal{A} \) has an unifier, \( \text{OU}(\mathcal{A}) = \{ \text{mgu}(\mathcal{A}) \} \). The algorithm for computing a set of optimal grammars is as follows:

**Algorithm OG**

- **input:** a finite set \( D \) of functor-argument structures.
- **output:** a finite set of optimal grammars \( G \) such that \( D \subseteq \text{FL}(G) \).

First the algorithm RG is invoked to compute the general form.

Let \( \mathcal{A} = \{ \{ A \mid \text{GF}(D); c \mapsto A \} \mid c \in \text{dom}(\text{GF}(D)) \} \), and compute \( \text{OU}(\mathcal{A}) \).

Let \( \text{OG}(D) = \{ \sigma[\text{GF}(D)] \mid \sigma \in \text{OU}(\mathcal{A}) \} \).

**Proposition 4.50** \( \{ G \in \mathcal{G}_{\text{optimal}} \mid G \text{ is in reduced form} \} \subseteq \bigcup \text{range}(\text{OG}). \)

**Proposition 4.51** \( \mathcal{F}_{\text{OG}} \subseteq \mathcal{F}_{\text{optimal}}. \)

**Theorem 4.52** \( \mathcal{F}_{\text{optimal}} \) has a limit point.

**Corollary 4.53** \( \mathcal{F}_{\text{OG}} \) has a limit point.

**Corollary 4.54** Neither \( \mathcal{G}_{\text{optimal}} \) nor \( \bigcup \text{range}(\text{OG}) \) is learnable from structures.

Even though the class of optimal grammars is not learnable, it is still interesting, since it can be used as basis for defining other classes.

### 4.6 Least Cardinality Grammars

The class of least cardinality grammars is a variation of optimal grammars. Constraining the cardinality of hypothesized grammars leads to the definition of a learnable subclass of optimal grammars.

**Definition 4.55** Let \( L \subseteq \Sigma^F \). A grammar \( G \) is said to be of least cardinality with respect to \( L \) if \( L \subseteq \text{FL}(G) \) and there is no grammar \( G' \) such that \( |G'| < |G| \) and \( L \subseteq \text{FL}(G') \).

**Definition 4.56** We write \( \mathcal{G}_{\text{least-card}} \) to denote the class of least cardinality grammars over \( \Sigma \). The class \( \{ \text{FL}(G) \mid G \in \mathcal{G}_{\text{least-card}} \} \) is denoted \( \mathcal{F}_{\text{least-card}} \).

**Definition 4.57** If \( D \) is a finite set of functor-argument structures, let

\[
\text{LCG}(D) = \{ G \in \text{OG}(D) \mid \forall G' \in \text{OG}(D) (|G| \leq |G'|) \}.
\]
Lemma 4.58 if $G \in \text{LCG}(D)$, then $G$ is of least cardinality with respect to $D$.

Definition 4.59 A grammar $G$ is called a least cardinality grammar if $G$ is of least cardinality with respect to $\text{FL}(G)$.

Proposition 4.60 $\bigcup \text{range}(\text{LCG}) = (\bigcup \text{range}(\text{OG})) \cup \mathcal{G}_{\text{least-card}}$.

Proposition 4.61 $\{G \in \mathcal{G}_{\text{least-card}} \mid G$ is in reduced form$\} \subseteq \bigcup \text{range}(\text{LCG})$.

Corollary 4.62 $\mathcal{F}_{\text{least-card}} = \{\text{FL}(G) \mid G \in \bigcup \text{range}(\text{LCG})\}$.

Proposition 4.63 $\mathcal{F}_{\text{least-card}} \subseteq \mathcal{F}_{\text{OG}}$.

Let $\varphi_{\text{LCG}}$ be defined as in Definition 4.24, with $\text{VG}_k$ replaced by $\text{LCG}$.

Theorem 4.64 $\varphi_{\text{LCG}}$ learns $\mathcal{G}_{\text{least-card}}$ from structures.

- $\varphi_{\text{LCG}}$ is responsive and consistent on $\mathcal{G}_{\text{least-card}}$.
- $\varphi_{\text{LCG}}$ is conservative.

$\varphi_{\text{LCG}}$ can be shown to learn $\mathcal{G}_{\text{least-card}}$ order-independently. A set-driven learning function $\varphi^\ast_{\text{LCG}}$ can be defined, analogous to Definition 4.27. Whether this function is also conservative is an open question; the proof of Proposition 4.35 does not work for (subclasses of) optimal grammars.

4.7 Minimal Grammars

Like least cardinality grammars, the class of minimal grammars is a subclass of optimal grammars. Hypothesized grammars are required to be minimal according to a certain partial ordering, in addition to being optimal.

Definition 4.65 Let $l = |\Sigma|$, and let $c_1, \ldots, c_l$ be the elements of $\Sigma$ arranged in a fixed order. For each grammar $G$ over $\Sigma$, let $v(G)$ be the vector defined as follows:

$$v(G) = \langle n_{1}, \ldots, n_{l} \rangle,$$

where for $1 \leq j \leq l, n_{j} = |\{A \mid G : c_{j} \rightarrow A\}|$.

The partial order $\preceq$ is defined on vectors in $\mathbb{N}^l$ in the natural way: $\langle n_{1}, \ldots, n_{l} \rangle \preceq \langle m_{1}, \ldots, m_{l} \rangle$ if $n_{j} \leq m_{j}$ for all $1 \leq j \leq l$. Also, if $v_{1}, v_{2} \in \mathbb{N}^l$, $v_{1} < v_{2}$ if and only if $v_{1} \preceq v_{2}$ and $v_{1} \neq v_{2}$.

Definition 4.66 If $D$ is a finite subset of $\Sigma^*$, let

$$\text{MG}(D) = \{G \in \text{OG}(D) \mid \exists G' \in \text{OG}(D)(v(G') < v(G))\}.$$
Thus, \( MG(\mathcal{D}) \) consists of those elements of \( \mathcal{OG}(\mathcal{D}) \) of which the associated vector is minimal. Clearly, \( MG(\mathcal{D}) \neq \emptyset \) for all \( \mathcal{D} \).

In Buszkowski and Penn (1990) the elements of \( MG(\mathcal{D}) \) were characterized in terms of minimality with respect to \( \mathcal{D} \).

**Definition 4.67** Let \( L \subseteq \Sigma^F \). A grammar \( G \) is said to be minimal with respect to \( L \) if \( L \subseteq FL(G) \) and there is no grammar \( G' \) such that \( v(G') < v(G) \) and \( L \subseteq FL(G') \).

**Definition 4.68** A grammar \( G \) is said to be minimal if \( G \) is minimal with respect to \( FL(G) \).

**Definition 4.69** We write \( \mathcal{G}_{\text{minimal}} \) to denote the class of minimal grammars over \( \Sigma \). The class \( \{FL(G) \mid G \in \mathcal{G}_{\text{minimal}}\} \) is denoted \( \mathcal{F}_{\text{minimal}} \).

**Corollary 4.70** \( G_1 \subseteq G_2 \) implies \( v(G_1) \leq v(G_2) \).

**Proposition 4.71** If a grammar \( G \) is of least cardinality with respect to \( L \), then \( G \) is minimal with respect to \( L \).

**Proposition 4.72** The class \( \{FL(G) \mid v(G) \neq \langle n_1, \ldots, n_k \rangle \} \) does not have finite elasticity.

Whether or not \( \mathcal{G}_{\text{minimal}} \) is learnable from structures is, as far as we know, still an open question. Kanazawa conjectures it is learnable.

### 4.8 Learning \( k \)-valued Grammars from Strings

As we have seen, an algorithm for learning \( \mathcal{G}_{k\text{-valued}} \) from strings was presented in Kanazawa (1998), and it was shown that, since \( \mathcal{L}_{k\text{-valued}} \) has finite elasticity, this class is learnable from strings. Note that \( \mathcal{G}_{\text{rigid}} \) is just a special case of \( \mathcal{G}_{k\text{-valued}} \), so this class is learnable as well.

**Proposition 4.73** The class \( \mathcal{L}_{k\text{-valued}} \) has finite elasticity.

In the proof of this proposition a theorem from citekanazawa94note was used that is a generalization of a theorem by Wright (Wright (1989)) (Wright’s theorem states that if two language classes \( \mathcal{L} \) and \( \mathcal{M} \) have finite elasticity, then the class \( \{L \cup M \mid L \in \mathcal{L} \land M \in \mathcal{M}\} \) also has finite elasticity).

**Theorem 4.74** Let \( \mathcal{M} \) be a class of languages over \( \Upsilon \) 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.
From Definition 3.15, we have \( L(G) = \{ \text{yield}(T) \mid T \in FL(G) \} \). If \( L \subseteq \Sigma^F \), we write \( \text{yield}(L) \) for \( \{ \text{yield}(T) \mid T \in L \} \). Then \( L_{k\text{-valued}} = \{ \text{yield}(L) \mid L \in \mathcal{L}_{k\text{-valued}} \} \). The relation \( R \subseteq \Sigma^+ \times \Sigma^F \) defined by \( RsT \iff s = \text{yield}(T) \) is finite valued. Since \( \mathcal{L}_{k\text{-valued}} \) has finite elasticity, applying Theorem 4.74 shows that \( L_{k\text{-valued}} \) also has finite elasticity.

Using Proposition 4.73, the following can be shown:

**Theorem 4.75** For each \( k \in \mathbb{N} \), \( L_{k\text{-valued}} \subseteq L_{k+1\text{-valued}} \).

### 4.8.1 Algorithms for Learning \( k \)-Valued Grammars from Strings

Kanazawa’s algorithm is based on two computable functions, \( \Psi_{k\text{-valued}} \) and \( \mu_L \). The function \( \Psi_{k\text{-valued}} \) maps a finite set of strings to a finite set of \( k \)-valued grammars. The function \( \mu_L \) takes two arguments, a finite set of grammars and a positive integer, and returns a member of the first argument.

**Definition 4.76**

\[
\Psi_{k\text{-valued}}(\{s_0, \ldots, s_i\}) = \bigcup \{ VG_k(\{T_0, \ldots, T_i\}) \mid s_j = \text{yield}(T_j) \}
\]

where \( 0 \leq j \leq i \).

This function applies \( VG_k \) to all possible functor-argument structures of the strings in the input\(^8\). The value of \( \Psi_{k\text{-valued}} \) is always a finite set of \( k \)-valued grammars.

**Lemma 4.77** If \( G \in \Psi_{k\text{-valued}}(\{s_0, \ldots, s_i\}) \), then \( \{s_0, \ldots, s_i\} \subseteq L(G) \).

Given Proposition 4.18, the following lemma is straightforward:

**Lemma 4.78** If \( G \in G_{k\text{-valued}}(\{s_0, \ldots, s_i\}) \subseteq L(G) \), then there exists some \( G' \in \Psi_{k\text{-valued}}(\{s_0, \ldots, s_i\}) \) such that \( G' \subseteq G \).

This implies the following:

**Proposition 4.79** \( \Psi_{k\text{-valued}}(\{s_0, \ldots, s_i\}) \) includes all minimal elements of \( \{ L \in \mathcal{L}_{k\text{-valued}} \mid \{s_0, \ldots, s_i\} \subseteq L \} \).

We could use this function together with a function \( \mu_{L,k} \) to define a learning function as in Definition 2.60. The function \( \mu_{L,k} \) would always have to choose a grammar of which the (string) language is a minimal element of

---

\(^8\)Note that the number of possible functor-argument structures associated with a sentence is exponential in the length of that sentence. Also note that, given sufficient strings of sufficient length, samples of the form found in the proof of Proposition 5.2 will be generated. Since it follows from this proposition that an exponential number of \( k \)-valued grammars can be generated, both size and time complexity of this algorithm are exponential.
4.8. LEARNING K-VALUED GRAMMARS FROM STRINGS

\{L \in L_{k, \text{valued}} \mid \{s_0, \ldots, s_i\} \subseteq L\}. By Proposition 2.61, we would be able to define a conservative learning function that learns \(G_{k, \text{valued}}\) from strings prudently and is responsive and consistent on this class.

It is not clear whether such a computable function \(\mu_{L,k}\) exists.\(^9\) If conservatism is not required, however, it is possible to define a computable learning function for \(G_{k, \text{valued}}\) using \(\Psi_{k, \text{valued}}\). Recall the ordering \(\prec\) from Definition 4.29:

**Definition 4.80** Let \(\mu_L\) be a computable function that maps a non-empty finite set \(G\) of grammars and a positive integer \(n\) to the first element of the following set, under the ordering \(\prec\), i.e.,

\[
\{G \in G \mid \neg \exists G' \in G(L(G') \cap (\Sigma \cup \{e\})^n) \subseteq L(G) \cap (\Sigma \cup \{e\})^n\}\]

**Lemma 4.81** Let \(G\) be a finite set of grammars. Then there is an \(m \in \mathbb{N}\) such that for all \(n \geq m\), \(\mu_L(G, n)\) is the first element of the following set,

\[
G' = \{G \in G \mid \neg \exists G' \in G(L(G') \subseteq L(G))\},
\]

under the ordering \(\prec\).

**Definition 4.82** Define a learning function \(\psi_{k, \text{valued}}\) for the grammar system \((\text{CatG}, \Sigma^+, L)\) as follows:

\[
\psi_{k, \text{valued}}([s_0, \ldots, s_i]) = \mu_L(\Psi_{k, \text{valued}}([s_0, \ldots, s_i]), i + 1),
\]

where \(i + 1\) is the length of \([s_0, \ldots, s_i]\).

**Proposition 4.83** The learning function \(\psi_{k, \text{valued}}\) is responsive and consistent on \(G_{k, \text{valued}}\).

**Theorem 4.84** The learning function \(\psi_{k, \text{valued}}\) learns \(G_{k, \text{valued}}\) from strings order-independently and prudently.

The function \(\psi_{k, \text{valued}}\) is not set-driven, since it refers to the length of its argument. However, a simple variation on the definition of \(\psi_{k, \text{valued}}\) makes it set-driven:

**Definition 4.85** Define a learning function \(\psi_{k, \text{valued}}^\sharp\) for the grammar system \((\text{CatG}, \Sigma^+, L)\) as follows:

\[
\psi_{k, \text{valued}}^\sharp([s_0, \ldots, s_i]) = \begin{cases} 
\text{the first element of} & \\
\{G \in G_{k, \text{valued}}([s_0, \ldots, s_i]) \mid L(G) = \{s_0, \ldots, s_i\}\} & \text{if it exists,} \\
\mu_L(\Psi_{k, \text{valued}}([s_0, \ldots, s_i]), \{s_0, \ldots, s_i\}) & \text{otherwise.}
\end{cases}
\]

Note that reference to the length of \([s_0, \ldots, s_i]\) is replaced by reference to the cardinality of \([s_0, \ldots, s_i]\).

\(^9\)For a discussion of this question, and of the relation between finite elasticity and undecidability of the inclusion problem, see Kanazawa (1998), page 138.
Proposition 4.86 The learning function $\psi_k^{k}$-valued is responsive and consistent on $G_k$-valued and is set-driven.

Proposition 4.87 The learning function $\psi_k^{k}$-valued learns $G_k$-valued from strings order-independently and prudently.

4.9 Classes that are not Learnable from Strings

Definition 4.88 Let $L_{\text{least-valued}}$, $L_{\text{optimal}}$, $L_{\text{least-card}}$, and $L_{\text{minimal}}$ denote the classes $\{L(G) \mid G \in G_{\text{least-valued}}\}$, $\{L(G) \mid G \in G_{\text{optimal}}\}$, $\{L(G) \mid G \in G_{\text{least-card}}\}$, and $\{L(G) \mid G \in G_{\text{minimal}}\}$, respectively.

Proposition 4.89 Each of the classes $L_{\text{least-valued}}$, $L_{\text{optimal}}$, $L_{\text{least-card}}$, and $L_{\text{minimal}}$ has a limit point.

Corollary 4.90 None of the classes $G_{\text{least-valued}}$, $G_{\text{optimal}}$, $G_{\text{least-card}}$, and $G_{\text{minimal}}$ is learnable from strings.

In Kanazawa (1998), an alternative characterization of these classes, that are defined in reference to the naming function $L$ instead of $F_L$, is investigated:

Definition 4.91 The classes $G_{L_{\text{least-valued}}}$, $G_{L_{\text{least-card}}}$, and $G_{L_{\text{minimal}}}$ are defined just like $G_{\text{least-valued}}$, $G_{\text{least-card}}$, and $G_{\text{minimal}}$ except that reference to $F_L$ is replaced by reference to $L$. Let $L_{\text{least-valued}} = \{L(G) \mid G \in G_{L_{\text{least-valued}}}\}$, $L_{\text{least-card}} = \{L(G) \mid G \in G_{L_{\text{least-valued}}}\}$, and $L_{\text{minimal}} = \{L(G) \mid G \in G_{L_{\text{minimal}}}\}$.

Corollary 4.92 The following inclusions hold:

$G_{L_{\text{least-valued}}} \subseteq G_{L_{\text{least-valued}}}$, $G_{L_{\text{least-card}}} \subseteq G_{L_{\text{least-card}}}$, and $G_{L_{\text{minimal}}} \subseteq G_{L_{\text{minimal}}}$.

Corollary 4.93 $G_{L_{\text{least-card}}} \subseteq G_{L_{\text{minimal}}}$.

Proposition 4.94 For $G = G_{L_{\text{least-valued}}} \subseteq G_{L_{\text{least-valued}}}$, $G_{L_{\text{least-card}}} \subseteq G_{L_{\text{least-card}}}$, $G_{L_{\text{minimal}}} \subseteq G_{L_{\text{minimal}}}$, the class $\{G \in G \mid \text{for no } G' \supseteq G, L(G') = L(G)\}$ is r.e.

It turns out that $G_{L_{\text{least-valued}}}$ and $G_{L_{\text{least-card}}}$ are learnable from strings. This is not as interesting a result as it may seem at first sight, however. Since $\Sigma^+ \in L_{2}$-valued, $G_{L_{\text{least-valued}}}$ is included in $G_{L_{2}}$-valued. Similarly, $G_{L_{\text{least-card}}}$ is included in some $G_{k}$-valued.

Proposition 4.95 $L_{\text{minimal}}$ contains an infinite ascending chain.
4.10. SUMMARY

<table>
<thead>
<tr>
<th>Class</th>
<th>Learnable</th>
<th>Finite Elasticity</th>
</tr>
</thead>
<tbody>
<tr>
<td>$G_{\text{rigid}}$</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>$G_{\text{k-valued}}$</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>$G_{\text{optimal}}$</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>$G_{\text{least-valued}}$</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>$G_{\text{least-card}}$</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>$G_{\text{minimal}}$</td>
<td>?</td>
<td>no</td>
</tr>
</tbody>
</table>

Table 4.1: Summary of Kanazawa’s results for learning from structures.

It is not clear whether $G_{\text{minimal}}^{L}$ is learnable from strings.

As a final remark, note that Seginer (2002a) shows that any subclass of
$G_{\text{rigid}}^{L}$ that has an alphabet restricted to just two letters is efficiently learnable.
An algorithm is presented that exploits properties of the string languages in
this class (restrictions on the ratio of the number of occurrences of a’s and b’s in
a sentence, among others things) to build a certain type of graph that is very
similar to the Stern-Brocot tree (see Graham et al. (1994)).

4.10 Summary

Tables 4.1 and 4.2 sum up Kanazawa’s results concerning learnability from
structures and strings, respectively, of the classes of grammars referred to in
this chapter.

- ‘Learnable’ here means prudently learnable from structures, by a conser-
  vative learning function that is responsive and consistent on the class.

- ‘Finite elasticity’ refers to the finite elasticity of the structure language
  associated with the class.

- The existence of efficient algorithms for learning these classes will be
discussed in Chapter 5.

---

[10] The presentation of the algorithm in Seginer (2002a) is somewhat incomplete, a full
description and detailed proofs of correctness can be found in Seginer (2002b).
Table 4.2: Summary of Kanazawa’s results for learning from strings.

<table>
<thead>
<tr>
<th>Class</th>
<th>Learnable</th>
<th>Finite Elasticity</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{G}_{\text{rigid}}$</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>$\mathcal{G}_{k\text{-valued}}$</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>$\mathcal{G}_{\text{least-valued}}$</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>$\mathcal{G}_{\text{optimal}}$</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>$\mathcal{G}_{\text{least-card}}$</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>$\mathcal{G}_{\text{minimal}}$</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>$\mathcal{G}^{L}_{\text{least-valued}}$</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>$\mathcal{G}^{L}_{\text{least-card}}$</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>$\mathcal{G}^{L}_{\text{minimal}}$</td>
<td>?</td>
<td>no</td>
</tr>
</tbody>
</table>