Learning Categorial Grammars

Printed by PrintPartners Ipskamp, Enschede, the Netherlands
http://www.ppi.nl

ISBN 90-393-3544-3

Learning Categorial Grammars

Over de Leerbaarheid van Categoriale Grammatica's:

(met een samenvatting in het Nederlands)

Proefschrift

ter verkrijging van de graad van doctor
aan de Universiteit Utrecht
op gezag van de Rector Magnificus,
Prof. Dr. W. H. Gispen
Ingevolge het besluit van het College voor Promoties
in het openbaar te verdedigen
op 14 november 2003
des middags te 16:15 uur

door

Christophe Costa Florêncio

geboren op 20 december 1971 te Leiden

Promotors: Michael Moortgat
Wojciech Buszkowski

Contents

List of Figures

List of Tables

Acknowledgements

In a way this chapter was one of the hardest to write, because of the possible repercussions of any error or omission. I apologize in advance to anyone I might have left out by mistake. First of all I would of course like to thank my promotores Michael Moortgat and Wojciech Buszkowski.

This thesis is the result of an intellectual journey that began when I enrolled on the Artifical Intelligence and Cognitive Science program at the Faculty of Philosophy of Utrecht University, which at that time was the only place in the country that offered the opportunity of studying such an 'esoteric' subject. My time as a student has been an exciting and challenging experience. I thank Jan, Pepijn, Remco, Jeronimus and Frits especially for making this period in my life a colourful one indeed.

During this time I became increasingly interested in (machine) learning, and took courses on neural networks, genetic algorithms and the like. Though this was interesting enough, I found there was a lack of a firm theoretical basis for these approaches, and also felt that most of these techniques were applied to toy problems. I was looking for something more challenging as subject for a master's thesis and felt that learning grammars would fit the bill nicely. During this time Michael Moortgat lent me a copy of Kanazawa's thesis and sent me to Dick de Jongh, only then did things start to come together. I also was fortunate enough to be able to follow the course on learnability that Makoto Kanazawa taught at the Utrecht institute of Linguistics OTS in 1998, and I thank him for his many helpful comments. I'd also like to thank Jean-Jules Meyer, Wiebe van der Hoek and Cees Witteveen for helping me out when I was looking for a Ph.D. position.

During my time as a Ph.D. student I had the pleasure of meeting to the following people who have influenced me or were helpful in some way or other: Gerald Penn, Hans-Jörg Tiede, Glyn Morrill, Gerhard Jäger, Jens Michaelis, Yannick le Nir, Annie Foret, Dana Angluin, James Royer, Daniel N. Osherson (email), the ever helpful Paul Wielemaker (email), Robin Clark (email), Paul Dekker, Crit Cremers, Ton van der Wouden, Jacqueline van Kampen, Bill Philip, Paola Monachesi, Mark Steedman, Edward P. Stabler, Philippe de Groote, Christian Retoré, Johan van Benthem, Marc Tommasi, Isabelle Tellier, Colin de la Higuera, Joachim Lambek, Claudia Casadio and Kees Vermeulen,

in random order.

The following fellow Ph.D. students and/or officemates made working at UiL OTS a pleasant experience: Richard Moot, Balász Szurányi, Rick Nouwen, Berit Gehrke, Oele Koornwinder, Marijana Marelj, Anna Mlynarczyk, Sharon Unsworth, Ninke Stukker, Mike Huiskes, Evangelia Vlachou, Benjamin Spector, Anna Feldman, Heleen Hoekstra, Esther Kraak and Øystein Nilsen, again in random order.

Being a Ph.D. student offers many opportunities for travel and meeting people from other countries. Space restrictions do not allow me to mention them all, so I will just thank Roberto Bonato for his company and accomodating me during our stay in Venice, and Daniel Lefavre and Darko Sarenac for making ESSLLI'01 the most amusing summerschool I've ever attended.

There is more to life than science, and I thank some of the friends who kept reminding me of the fact: Richard Karsmakers, Christian Straman and all the other 'virusjes', my fellow squash gang members Jan van Rosmalen, Mark de Graaf and Patrick Lapre, and of course my family.

I thank Kathrin for her love, patience and support.

Chapter 1

Introduction

> It is not enough for a wise man to study nature and truth; he should dare state truth for the few who are willing and able to think. As for the rest, who are voluntarily slaves of prejudice, they can no more attain truth, than frogs can fly.
>
> Julien Offray de la Mettrie, *Man a Machine*

The science of linguistics is concerned with discovering and defining the form and structure of natural languages, and focuses on issues in the fields of e.g. syntax, semantics and pragmatics. In contrast, the study of *language acquisition* is concerned with the question of *how* and *when* children master the thing linguistics sets out to define. All normal children learn the language they are exposed to, the so called ambient language. The task of getting from a necessarily limited range of input to (implicit) knowledge of the complete adult grammar is known as the *projection problem* (Peters (1972)) or the *logical problem of language acquisition* (Hornstein and Lightfoot (1981); Baker and McCarthy (1981)).

Most linguists are convinced that the projection problem implies the existence of a substantial innate component of linguistic knowledge. They believe that the gap between the evidence available to the child and the linguistic competence the child ultimately achieves is so great that language acquisition can only be accounted for if we assume that children have access to some kind of linguistic knowledge. This innate knowledge is generally called *universal grammar*.[1]

Ever since Chomsky's review of Skinner's *Verbal Behavior* (Chomsky (1959)) the notion of *explanatory adequacy*, defined from the viewpoint of acquisition, has been a major goal in syntactic theory. Any syntactic theory is required in

[1] The role of universal grammar in language acquisition was discussed in the influential Chomsky (1965a).

the end to account for how acquisition is possible given the ambiguous relation between sentence and grammar. In other words, any proposed characterization of the class of all possible natural languages must have the property of being *learnable*.

Even though the term learnable has since then frequently been used in informal linguistic discourse, it is very seldom clear what exactly it is intended to signify. Instead vague terms like 'poverty of the stimulus', and biological metaphors like 'language organ growth' are used in generativist discourse to defend some kind of nativist theory of acquisition that presumably renders learnability issues irrelevant. Most critics of this approach use arguments on a similar informal level, emphasizing the existence of 'data-driven' and 'general' learning mechanisms, with either no or ad-hoc justifications for such claims, and quite often displaying complete ignorance of linguistic fact. It seems that proponents on both sides see themselves as defending either the rationalist or empiricist tradition in western philosophy, i.e. they are involved in a form of the age-old nature/nurture debate.

We will not concern ourselves with these issues in this thesis.[2] Instead, we will focus on the question of learnability from a formal point of view; we apply Gold's mathematical model of learning known as identification in the limit. More specifically, we will take as starting point the results presented in Kanazawa (1998), where Gold's was model applied to work by Wojciech Buszkowski and Gerald Penn. These results were obtained from a synthesis of formal learning theory, classical categorial grammar (CCG),[3] and learning algorithms based on unification.[4]

One can regard this type of research as belonging to the field of mathematical linguistics, which has two separate objects of study; the mathematical properties of natural language and the mathematical properties of *theories* about natural language. This thesis is clearly concerned with the latter kind of research, we will make no claims about psychological plausibility for example.

In the influential Gold (1967) the concept of identification in the limit was applied to the language classes in the Chomsky hierarchy.[5] In this model of (language)[6] learning a learning function receives an endless stream of sentences

[2]There are of course links between philosophy and formal learning theory. In Kelly et al. (1994) it is pointed out that Putnam invented computational learning theory in a critique of Carnap's confirmation theory, and in Kelly and Glymour (1992) it is argued (somewhat frivolously) that Plato was the first computational learning theorist.

[3]The abbreviation CCG is commonly used for *combinatory* categorial grammar. Since we will be discussing both classical and combinatory systems however, we will use notation from Kanazawa (1998). We reserve the acronym GCG, which is short for General Combinatory Grammars, for the combinatorial type of system. We trust this will not lead to confusion.

[4]Unification, in all its forms, seems to be a good starting point when designing algorithms for learning grammars in a lexicalized linguistic formalism, see Nicolas (1999).

[5]This is a good occasion to dismiss a common misconception: Gold did not introduce this concept. As is pointed out in the article itself, identification in the limit was discussed before in the pattern recognition literature, and Aizerman et al. (1964) is cited. To the best of the author's knowledge, the earliest (formal) work in this direction was Moore (1956).

[6]In this model languages are considered to be simply sets of sentences. In formal language

from the target language, called a *text*, and hypothesizes a grammar for the target language at each time-step.

A *class* of languages is called *learnable* if and only if there exists a learning function that guesses the right language on every text for every language from that class after the presentation of a finite number of sentences and does not deviate from this hypothesis. Note that the learner may not be aware of its success,[7] and that no claim is made about processes internal to the learner, only an *external* criterion for succes is given. However, using recursion theoretic methods it is possible to obtain results about these internal processes, given external criteria.

Gold's early work was and is widely known among linguists, it is cited in Chomsky (1981), and Pinker (1979), for example. References to later work in the field can be found in for example Chomsky (1986) (notes 86 and 89, Chapter 3), in Chomsky (1980), papers reprinted in Lasnik (1990), a chapter in the introductory book on acquisition Atkinson (1992), or Bertolo (2001).[8]

The most important result was considered to be the proposition that no superfinite class (any class containing all finite languages and at least one infinite language) is learnable from positive data. Since all non-trivial classes in the Chomsky-hierarchy are superfinite this result was interpreted as showing that identification in the limit is just 'too hard' and thus a trivial model of learning. It seems that this result has even been advanced as strong evidence for the existence of a nativist Universal Grammar. In hindsight this seems hard to understand, since few linguists would claim that there are finite natural languages, let alone that all finite languages are natural languages. It is also true that the Chomsky-hierarchy is just one of many different ways of classifying languages, there are other ways that are better suited to visualize properties related to learnability, as we shall see in Section 2.4.

It seems that formal learning theory has had little impact on (mainstream) linguistics.[9] In fact, it seems that most linguists are not familiar with learning theoretic work after Gold (1967). After Gold's (supposedly)[10] negative results,

---

theory sentences are often called *words*, and all the elements found in words for a given language form the *alphabet* of that language.

[7] A variant of this criterion has been studied where it is demanded that the learner signal its own convergence, this is known as *finite identification*. Gold cites Gill (1961), and there seems to be other work from the same period discussing this variant.

[8] The reader should exercise great caution when dealing with these informal discussions!

[9] The results by Wexler, Hamburger and others collected in Wexler and Culicover (1980) are a notable exception.

Work by Angluin, Angluin (1980b), has inspired the claim that the so-called subset-principle is a necessary condition for learnability in Berwick (1982, 1985). See Kapur et al. (1993) for a critical discussion.

[10] These results could also be considered positive, in the sense that they show that learnability considerations impose non-trivial constraints on what the class of human languages looks like. Learning theory could therefore be a very useful tool for any researcher interested in human language. To give one example, in Osherson et al. (1984) it was shown that the assumption that the data a child encounters is noisy (that is, contains a finite amount of ungrammatical sentences and (systematically) omits a finite number of grammatical sentences)

subsequent work showed that highly expressive non-trivial learnable classes exist. In the seventies (sometimes referred to as the Dark Age of formal learning theory) people like Bārzdiņš, Freivald, Wiehagen and Jantke worked mostly on characterizing learnable classes under different learning criteria, most of this work was published in German or Russian. Work by Angluin in the early eighties rekindled interest in the paradigm. She presented non-trivial learnable classes that cross-cut the Chomsky Hierarchy, demonstrating that the pessimism following Gold's earlier results was due to misinterpretation. To get an impression of the power of identification in the limit, see Subsection 2.3.1.

Although numerous impressive results were obtained during the eighties and nineties, deepening our understanding of identification in the limit, the impact of formal learning theory on linguistics was still almost negligable. The focus in formal learning theory is on general results within the highly theoretic framework of recursion theory, whereas most linguists seem to be after highly specific results applicable only to one particular type of formalism (c.f. Wexler and Culicover (1980)).

To bridge this gap, concrete examples of learnable classes are required that are expressed in linguistically relevant grammar formalisms. Some work in this direction has been done; in Shinohara (1990a,b) it was shown that the languages generated by context-sensitive grammars with a bounded number of rules can be identified from positive data. This result follows from more general results on monotonic systems[11] based on a formalism called Elementary Formal System (EFS, see Smullyan (1961)). These results suggest that monotonicity of the grammar formalism is a desirable property when dealing with learnability issues in linguistics. This paradigm will be discussed in Section 2.5.

Once a class has been shown to be learnable, the question arises how hard it is to learn languages from that class. This question is difficult to answer since there is no generally accepted definition of tractability for Gold's paradigm.[12] It turns out that this raises all sorts of interesting questions, which will be discussed in Section 2.9. We motivate the choice for a necessary condition for

restricts the space of learnable classes to finite classes. They also showed that the assumption that the class of all humanly learnable languages is of finite size (common in generative linguistics) does not by itself yield a learnable class: classes exist that contain just two languages that are not learnable with this type of noisy input. Thus the notion that assuming finite variation for natural languages trivializes learnability is shown to be a misconception.

[11] In a monotonic system it is always the case that for any two grammars G and G' such that $G \subseteq G'$, the language associated with G is a subset of the language associated with G'. Very informally this could be described as 'adding rules to a grammar will never block derivations that were previously accepted'.

[12] The learning paradigm known as Probably Approximately Correct (PAC) learning (introduced in Valiant (1984)) has the notion of efficiency built right into its definition of learnability, and (maybe) partly because of this it has been much more widely applied than Gold's paradigm. A thorough discussion of PAC learning is outside the scope of this thesis. The author feels that the PAC model is probably too restrictive to deal with the task of natural language learning. Even minor adaptations of this model potentially trivialize the learning task (see e.g. Parekh and Honavar (2000)), thus the paradigm is not flexible enough to cover a wider range of learning situations.

tractability, namely the tractability of coming up with a hypothesis consistent with the data. This definition is applied in Chapter 5 to the classes examined in Kanazawa (1998).

These classes will be detailed in Chapter 4. One unique[13] and interesting aspect of this approach is the emphasis on *structure languages*.[14] Most studies assume the input to consist just of strings and expect the final hypothesis to be a grammar that generates the right string language (note that the nature of data and the identification criterion are logically independent). When modeling language acquisition, however, it could be assumed that information about structure is accessible to the learner, be it from prosody (c.f. Wanner and Gleitman (1982)), semantics, or pragmatics (Snow (1977), Slobin (1977)).

Also, from a linguistic point of view it makes a lot of sense to use convergence on the right *structure language* as the identification criterion: one expects a learner to eventually *derive* and *analyze* sentences the same way as any other speaker, if only for the semantic implications. Since trees and tree languages play such an important role in this kind of research, background on regular tree languages and some known learnability results are presented in Chapter 7. Using these results an alternative and much shorter proof of a learnability result on CCG from Kanazawa (1998) is given.

We also turn our attention to a different kind of Categorial Grammar, namely the Lambek calculus (and its variants). Although this formalism is superficially very similar to CCG, its interpretation is very different, it is not a combinatory system but a form of positive intuitionistic linear logic. Although, like CCG, the Lambek calculus is context-free and thus not expressive enough to cover the full range of natural language phenomena, it is still relevant to linguistics because of the systematic way in which it relates syntactic derivations (proofs) to meaning. Surprisingly, the Lambek calculus has completely different learnability properties when compared to CCG as we will see in Chapter 9. Most of the results are negative, we give one positive result using results from Chapter 7.

The general moral of this chapter is that the paradigm of identification in the limit has been, and continues to be, relevant for linguists, especially for those working in acquisition (and also seems to be a promising approach for the field of Grammatical Inference). Hopefully it has also become clear that

[13]Learning (reversible) context-free languages from structured data was also investigated in Sakakibara (1992). The notion of structure used there is weaker than in Kanazawa (1998): the input consists of 'skeletal phrase structures', i.e., unlabeled trees. Also see Chapter 7 for work on learning from structures.

[14]The reader with a (generative) syntactic background may wonder just what is meant with structure language: constituent structure, head projections or licensing relations, for example. In this particular case the structures can be regarded as head projections. However, all of these notions are theory-specific; when dealing with associative Lambek grammars it can be misleading to talk about constituency, for example. Even the distinction of weak and strong generative capacity is not theory-neutral and is associated with Chomsky's original definition. Also see Miller (1999); Cornell and Rogers (to appear) for a discussions of strong generative capacity.indexstrong generative capacity

one cannot rely on informal discussions to learn about formal learning theory: there is no substitute for reading a standard work like Jain et al. (1999) (or perhaps still better, the original papers cited therein).

There are as yet few results for grammar formalisms that are actually used in linguistics: Kanazawa's classes of categorial grammars are restricted to the context-free realm, while natural language has been shown to go beyond this bound. Unfortunately our results for more expressive formalisms are all negative. It would be very interesting to see how much of the recent results can be adapted to frameworks like HPSG (see e.g. Pollard and Sag (1987)), for example. A natural approach would be to embed them in Elementary Formal Systems (EFS) and to place bounds on the size of the resulting logic programs, as in Shinohara (1990a,b), although results discussed in this thesis demonstrate that this is not always feasible.

We now give an overview of the structure of this thesis. First we will discuss the history and basis of formal learning theory in Chapter 2. Classical Categorial Grammar is discussed in Chapter 3. Chapter 4 gives a (condensed) presentation of work from Kanazawa (1998), and in Section 4.3 an open question from that book is answered.

New results are presented concerning the complexity of learning classes of Categorial Grammars in Chapter 5.

Chapter 6 takes a closer look at learnability aspects of *Combinatory* CG (a topic first addressed in Kanazawa (1998)), of collections of Generalized Quantifiers, extending work from Tiede (1999b), and of some severely restricted classes of Tree Adjoining Grammar (TAG), and of Minimalist Grammar (MG).

Our results for TAG are all negative, this is due to the optionality of adjoining, among other things. We conjecture that by restricting the use of adjoining it becomes possible to define learnable classes of TAGs by imposing numerical bounds on the complexity of a grammar. Our result for MG relies essentially on allowing the assignment of an unbounded number of licensee features to the same category, and, analogous to the TAG case, a restriction on such assignments may allow the existence of learnable classes of MGs of bounded complexity.

Chapter 7 presents results on the learnability of tree languages, mainly from Sakakibara (1990, 1992); Besombes and Marion (2001, 2002a).

Chapter 8 presents the Lambek calculus and variations, and the results from Chapter 7 are used in Chapter 9, where new results concerning Lambek grammars are presented, as well as work from Foret and Le Nir (2002a,b); Bechet and Foret (submitted).

Chapter 2

Formal Learning Theory

As was pointed out in the previous chapter, questions concerning learning are relevant to linguistic research, and thus a *precise* model of (language) learning is needed. Formal Learning Theory (or Learning Theory) can provide just such a model, or perhaps more accurately, a framework for formulating such models.

In this chapter[1] the formal concepts and terminology of learning theory are presented. We will mostly use notation from Kanazawa (1998). For a comprehensive overview and references, see Osherson et al. (1986) and Osherson et al. (1997). Note that, since we are only interested here in Gold's concept of learning, *learning* will be used only as abbreviation for *identification in the limit from positive data*.

We will discuss the origins and history of this field in the following section, the rest of the chapter will deal with formal definitions, known results and discussion of some advanced topics.

2.1 History

The aim of Formal Learning Theory is not necessarily providing us with learning *algorithms*, but rather characterizing the *a priori possibility* of learning a certain class (of languages) given a specification of the conditions under which learning has to take place. Most of the basic concepts of learning theory were introduced in the seminal paper Gold (1967).[2]

[1] Parts of this chapter have appeared in Costa Florêncio (2002b), and are reproduced with permission.

[2] Results concerning the possibility of learning the internal structure of finite state machines, in a different but comparable formal context, were first presented in Moore (1956), eleven years before Gold. This research was done with cryptographic applications in mind, however, and maybe for this reason the implications for linguistics and epistemology went largely unnoticed. In recent years this paper has been cited in work in the field of Grammatical Inference, especially in the context of learning regular languages.

In this model, the central concept is that of learning as an infinite process. To quote Gold (1967):

> A class of possible languages is specified, together with a method of presenting information to the learner about an unknown language, which is to be chosen from the class. The question is now asked, 'Is the information sufficient to determine which of the possible languages is the unknown language?' Many definitions of learnability are possible, but only the following is considered here. Time is quantized and has a finite starting time. At each time the learner receives a unit of information and is to make a guess as to the identity of the unknown language on the basis of the information received so far. This process continues forever. The class of languages will be considered *learnable* with respect to the specified method of information presentation if there is an algorithm that the learner can use to make his guesses, the algorithm having the following property: Given any language of the class, there is some finite time after which the guesses will all be the same and they will be correct.

So, a learner (learning function) is presented with an endless stream of sentences from the target language. Each time a sentence is presented, the learning function makes a guess as to (an index for) the target language. This guess consists of a conjectured grammar for the language, so the infinite sequence of sentences has an associated infinite sequence of (hypothesized) grammars.

As the body of presented sentences grows, the guesses may change. Two assumptions are made: first, we expect the sentences to be grammatical, i.e. they are all well formed and thus really belong to the target language. Second, we expect all possible sentences to eventually appear in the sequence. Such an infinite sequence of sentences is called a *text*. The set of all finite sequences of any length in any text is denoted SEQ, this can be thought of as the set of all possible evidential states.

In Gold's model, learning is successful if there exists a time after which the learning function's guess does not change (stability) and is identical to the grammar for the target language (veridicality). (Note that the learning function never knows if its guess is correct.)[3] This is exactly the definition of *identification in the limit*.

A class of languages is said to be *learnable* if a learning function exists that can identify any target language in that class from a sequence of sentences from that language. Learnability is therefore a property of a *class* of languages, not of any one individual language: a learning function that could only learn one

[3]Learning functions that signal their (successful) convergence, so called *self-monitoring learning functions*, have also been investigated. See Freivalds and Wiehagen (1979) and Osherson et al. (1986).

Learnability model[4]	Class of languages
Anomalous text	
	Recursively enumerable
	Recursive
Informant	
	Primitive recursive
	Context-sensitive
	Context-free
	Regular
	Superfinite
Text	
	Finite cardinality languages

Table 2.1: Dividing Lines between Learnability and Nonlearnability of Languages (from Gold (1967)).

particular language would not really be learning at all, since it could simply always hypothesize the (index for) the same language.

In this particular model only positive data is presented to the learner, i.e. the learning function only receives information about which sentences are in the target language. Gold also considered a model in which the learner is provided with *complete data*, that is, both positive and negative data. In this situation, an *oracle* or *informant* can be consulted by the learning function, telling it whether or not a sentence belongs to the target language. In this case, where complete data is available, learning turns out to be much easier. However, empirical evidence suggests that natural language is not acquired in this way. Children do not seem to respond to the correction of linguistic mistakes, and are generally not (overtly) corrected by their parents. (See, for example, Brown and Hanlon (1970) and Goodluck (1991).)

This restricted nature of the input (the paucity of positive evidence and the lack of negative evidence) is often referred to as the *poverty of the stimulus* and has frequently been invoked by Chomsky and others as a point in favour of a learning mechanism in which the child's innate knowledge of principles of grammar plays a major role in guiding development. (See Chomsky (1959), and e.g. Pullum and Scholz (2002) and papers in the same edition of the journal for a critique of this notion.)

As Table 2.1 shows, none of the four language classes in the Chomsky hierarchy is learnable from positive data. In fact, the only class that is learnable

[4]The classes of languages below the dividing line for a given model are learnable with respect to this model. The 'Informant' model refers to three variations of models that allow negative data in the input, they all yield the same learnability results.

The 'Anomalous Text' model refers to a model with primitive recursive text and the generator-naming relation.

from text in this table is completely trivial, since its members are all of finite cardinality. A learning function for this class could, after each presentation just store all the sentences presented so far and reproduce (an index for) its storage as its hypothesis.[5]

Gold's results have for a long time been taken to mean that identifying languages from positive data is just too hard. We cite Gold (1967):

> Those working in the field generally agree that most children are rarely informed when they make grammatical errors, and those that are informed take little heed. [...]
>
> However, the results presented in the last section show that only the most trivial class of languages considered is learnable [...]
>
> If one accepts identification in the limit as a model of learnability, then this conflict must lead to at least one of the following conclusions:
>
> 1. The class of possible natural languages is much smaller than one would expect from our present models of syntax. That is, even if English is context-sensitive, it is not true that any context-sensitive language can occur naturally. Equivalently, we may say that the child starts out with more information than that the language it will be presented is context-sensitive. In particular, the results on learnability from text imply the following: the class of possible natural languages, if it contains languages of finite cardinality, cannot contain all languages of finite cardinality.
> 2. The child receives negative instances by being corrected in a way we do not recognize. If we can assume that the child receives both positive and negative instances, then it is being presented information by an 'informant'. The class of primitive recursive languages, which includes the class of context-sensitive languages, is identifiable in the limit from an informant. The child may receive the equivalent of negative instances for the purpose of grammar acquisition when it does not get the desired response to an utterance. It is difficult to interpret the actual training program of a child in terms of the naive model of a language assumed here.
> 3. There is an a priori restriction on the class of texts which can occur, such as a restriction on the order of text presentation. The child may learn that a certain string is not acceptable by

[5] Even though this arguably makes learning this class trivial from a *learnability* point of view (this depends on the identification criterium, see Osherson et al. (1984)), it can be a tricky *algorithmic* problem.

> the fact that it never occurs in a certain context. This would constitute a negative instance.

In linguistics, it is generally assumed that the first conclusion holds, that is, the class of humanly learnable languages (commonly denoted **H**) is severely restricted.

Only around 1980 did papers by Angluin (Angluin (1980a,b, 1982)), in which non-trivial learnable classes were presented, show that the initial pessimism was premature. Later Shinohara showed that placing any finite bound on the number of rules used in context-sensitive grammars results in a learnable class (Shinohara (1990a,b)).

Although Gold's work was explicitly focused on (machine) acquisition of human languages, formal learnability theory seems to have had far more impact on e.g. epistemology than on linguistics. This may be due to the fact that most work in the field has been highly general and abstract, and little work has been done on linguistically relevant grammar formalisms.

Both Osherson et al. (1986) and Kanazawa (1998) work towards bridging this gap. In the former, additional constraints on learning were studied that are either linguistically or epistemologically motivated, in order to make the model more psychologically plausible. The latter applies this work to learning within a specific grammar system.

2.2 Basic Definitions

Formal learning theory offers a formal reconstruction of the following concepts:

- a set Ω (hypothesis space),
- a set **S** (sample space),
- a function L that maps elements of Ω to subsets of **S**.

Here, Ω can be any class of finitary objects, for example the set of natural numbers.[6] Members of Ω are called *grammars*.

S is a recursive subset of Σ^* for some fixed finite alphabet Σ. Elements of **S** are called *sentences*, subsets of **S** (which obviously are sets of sentences) are called *languages*.

If G is a grammar in Ω, then $\mathrm{L}(G)$ is called the *language generated by* G. L is also called the *naming function*. The question whether a sentence belongs to a language generated by a grammar is called the *universal membership problem*. Usually, the naming function is assumed to be such that the universal membership problem is decidable or at least semi-decidable (r.e.).

[6] See Higuera and Janodet (2001) for the inference of languages containing infinite words, so-called *ω-languages*, however in their model the learner sees only finite prefixes of these words. See Saoudi and Yokomori (1994) for learning ω-languages from presentations of *ultimately periodic* words.

A triple $\langle \Omega, \mathbf{S}, \mathrm{L} \rangle$ satisfying the above conditions is called a *grammar system*. A class of grammars is denoted $\mathcal{G}$, a class of languages is denoted $\mathcal{L}$.

2.3 Learning and Convergence

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

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 $\mathbf{S}$. 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 is defined and is equal to G for all but finitely many $i \in \mathbb{N}$.

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

Definition 2.1 ***Learning*** $\mathcal{G}$
Let a grammar system $\langle \Omega, \mathbf{S}, \mathrm{L} \rangle$ 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 $\mathrm{L}(\mathcal{G})$, and for every text for L, there exists some G in $\mathcal{G}$ such that $\mathrm{L}(G) = L$ and φ converges to G on $\langle s_i \rangle_{i \in \mathbb{N}}$.*

Note that a learning function φ is not required to converge to the same grammar on different infinite sequences for the same L, so the grammar that φ converges to can be dependent on the order of enumeration of elements from L.

Definition 2.2 ***Learnability of a class of grammars***
A class $\mathcal{G}$ of grammars is called non-effectively learnable *if a learning function exists that learns $\mathcal{G}$, and* not non-effectively learnable *if there is no such function.*

A class $\mathcal{G}$ of grammars is called (effectively) learnable *if a* computable *function exists that learns $\mathcal{G}$, and* nonlearnable *if there is no such function.*

Obviously learnability implies non-effective learnability.[7]

[7] In Osherson et al. (1986) a subsection is devoted to a discussion of the interest of non-recursive learning functions. It is claimed there that consideration of such functions often clarifies the respective roles of computational and information-theoretic factors in nonlearnability phenomena.

Definition 2.3 ***Existence of a limit point***
A class $\mathcal{L}$ of languages is said to have a limit point *if and only if there exists an infinite sequence $\langle L_n \rangle_{n \in \mathbb{N}}$ of languages in $\mathcal{L}$ such that* [8]

$$L_0 \subset L_1 \subset \ldots \subset L_n \subset \ldots$$

and there exists another language $L \in \mathcal{L}$ such that

$$L = \bigcup_{n \in \mathbb{N}} L_n.$$

The language L is called a limit point *of $\mathcal{L}$.*

Lemma 2.4 *If $\mathrm{L}(\mathcal{G})$ has a limit point, then $\mathcal{G}$ is not (non-effectively) learnable.*

Having a limit point is only a sufficient condition for a class to be not (non-effectively) learnable. A necessary and sufficient condition for non-effective nonlearnability is the existence of an *accumulation point*:[9]

Definition 2.5 ***Existence of an accumulation point (Kapur (1991))***
A class $\mathcal{L}$ of languages is said to have an accumulation point *if and only if there exists an infinite sequence $\langle S_n \rangle_{n \in \mathbb{N}}$ of sets such that*

$$S_0 \subseteq S_1 \subseteq \ldots \subseteq S_n \subseteq \ldots,$$

there exists a language $L \in \mathcal{L}$ such that

$$L = \bigcup_{n \in \mathbb{N}} S_n,$$

and for any $n \in \mathbb{N}$, there exists a language $L' \in \mathcal{L}$ such that $S_n \subseteq L'$ and $L' \subset L$. The language L is called an accumulation point *of $\mathcal{L}$.*

Definition 2.6 ***Locking sequence lemma (Blum and Blum (1975))*** *Suppose that a learning function φ converges on every infinite sequence that enumerates a language L. Then there is a finite sequence $\langle w_0, \ldots, w_l \rangle$ (called a* locking sequence *for φ and L) with the following properties:*

1. *$\{w_0, \ldots, w_l\} \subseteq L$,*
2. *for every finite sequence $\langle v_0, \ldots, v_m \rangle$, if $\{v_0, \ldots, v_m\} \subseteq L$, then $\varphi(\langle w_0, \ldots, w_l \rangle) = \varphi(\langle w_0, \ldots, w_l, v_0, \ldots, v_m \rangle)$.*

[8] The symbol $\subset$ denotes *proper* inclusion, that is $L_0 \subset L_1$ implies $L_1 - L_0 \neq \emptyset$. Inclusion will be notated as $\subseteq$. Likewise, we will make a distinction between *subset* and *proper subset*.

[9] As far as the author is aware, the first formulation of necessary conditions for non-effective learnability was presented in Wexler and Hamburger (1973), as part of an attempt to characterize effective learnability (text learnability in their terminology).

Proposition 2.7 *Suppose that φ learns $\mathcal{G}$. Then, for every $L \in \mathrm{L}(\mathcal{G})$, there exists a locking sequence $\langle w_0, \ldots, w_l \rangle$ for φ and L such that $\varphi(\langle w_0, \ldots, w_l \rangle) \in \mathcal{G}$ and $\mathrm{L}(\varphi(\langle w_0, \ldots, w_l \rangle)) = L$.*

The notion of locking sequences can be useful when proving (non)learnability, it is relatively easy for example to come up with a short alternative proof of Lemma 2.4 based on Proposition 2.7.

2.3.1 The Power of Identification in the Limit

The following proposition demonstrates the power of identification in the limit. Let **TxtEx** be the collection of all language classes that are effectively learnable, and let $\mathcal{E}$ be the class of all r.e. languages.[10]

Definition 2.8 *Let a class of languages $\mathcal{L} \subseteq \mathcal{E}$ be given.* $\mathcal{L}$ covers $\mathcal{E}$ *just in case for every $L \in \mathcal{E}$ there is an $L' \in \mathcal{E}$ such that L and L' are finite variants.*

Two languages L and L' are *finite* variants just in case both $L - L'$ and $L' - L$ are finite.

Proposition 2.9 *(Wiehagen (1978)) There is a class of languages $\mathcal{L} \in$ **TxtEx** such that $\mathcal{L}$ covers $\mathcal{E}$.*

In other words, nontrivial learnable classes exist that include languages with maximal expressive power, which demonstrates the relevance of Gold's paradigm. Finite variants share a lot of (computational) properties, for example if one is recursive so is the other. The same is true of degree of complexity. Thus $\mathcal{L}$ covers $\mathcal{E}$ implies that for any set of arbitrary complexity in $\mathcal{E}$ there is a set of equal complexity in $\mathcal{L}$.

2.3.2 The Weakness of Identification in the Limit

The next proposition from Fulk et al. (1994) is commonly known as the 'nonunion theorem'. It states that **TxtEx** is not closed under union:[11]

Proposition 2.10 *Let $\mathcal{L}_1 = \{L \in \mathcal{E} \mid L \text{ is finite}\}$ and $\mathcal{L}_2 = \{\mathbb{N}\}$. Then both $\mathcal{L}_1$ and $\mathcal{L}_2$ are in* **TxtEx**, *but $\mathcal{L}_1 \cup \mathcal{L}_2 \notin$* **TxtEx**.

This can be interpreted as a limitation on the kind of general purpose learning-devices we can construct, and has been used as motivation for the use of heuristics in AI. It also suggests a weaker identification criterion, where a *team* of n learners is set to the same task, and succes is achieved when m of the

[10] Noneffective identification in the limit from positive data is written as **Lang**, this (modern) notation was introduced by Case and Lynes (1982). By convention it is slightly abused and denotes both the paradigm and all classes learnable in this paradigm.

[11] There is also a non-union theorem for the paradigm of *function learning*, to be discussed in the next subsection.

n learners identify the same language. The fraction $\frac{n}{m}$ is known as the *success ratio*, the resulting paradigm is known as *team learning* or $\mathbf{Team^n_m TxtEx}$. What is the relation between $\mathbf{Team^n_m TxtEx}$ and **TxtEx**? It turns out that under sufficiently strict succes ratios the two paradigms are of the same power, the smallest such ratio is known as the *aggregation ratio* of the paradigm. The following two propositions are from Fulk et al. (1994):

Proposition 2.11 *Let n and m be such that $\frac{n}{m} > \frac{2}{3}$. Then $\mathbf{Team^n_m TxtEx} = \mathbf{TxtEx}$.*

Also consider the following surprising result, taken from Jain and Sharma (1996a):

Proposition 2.12 $\mathbf{Team^2_4 TxtEx} - \mathbf{Team^1_2 TxtEx} \neq \emptyset$.

This implies that something is to be gained by introducing redundancy into a team of learners! Contrast this with:

Proposition 2.13 *For all j,* $\mathbf{Team^{2j+1}_{4j+2} TxtEx} = \mathbf{Team^1_2 TxtEx}$.

2.3.3 Function Learning

It would be natural to consider the concept of learning a *function* rather than a set: empirical inquiry can be construed as learning from experiments, where Nature gives an output y in response to an experiment whose parameters are determined by x. The text thus includes any single-valued infinite sequence over $\mathbb{N} \times \mathbb{N}$, i.e. it consists of pairs $\langle x, y \rangle$, such that for each x the y is unique, since $f(x) = y$. It turns out that learning in the resulting paradigm (denoted by **Func** in the non-effective case, **Ex** in the effective case) is in some sense easier than in **TxtEx**:

Theorem 2.14 *The class of all recursive functions, $\mathcal{R}$, is* **Ex***-identifiable.*[12]

This implies that identifiability in this paradigm is a trivial matter, but note that memory limitation already restricts the space of learnable functions.

The paradigm of function learning may seem less relevant to linguistics than language learning, but consider the following; assume that the input to a **TxtEx**-learner φ is a *recursive text*, and let this paradigm be denoted by **RecTxtEx**. Then φ can behave like an **Func**-learner and learn the function $f(t) = e_t$, i.e. the function that generates the text:

Proposition 2.15 *There is a single noncomputable scientist that identifies each r.e. language from recursive texts.*

However, if we require the scientist to be computable, a different picture emerges:

[12] For a proof, see Jain et al. (1999), Theorem 3.43.

Proposition 2.16 $\mathbf{RecTxtEx} = \mathbf{TxtEx}$.

This result is a practical example of a situation in which the learner has access to information that could help it to learn, but is unable to take advantage of this because of computability issues.

Proposition 2.17 *(Pitt (1984); Pitt and Smith (1988))*
For all $j > 0$, $\mathbf{Team^{j}_{2j}Ex} = \mathbf{Team^{1}_{2}Ex}$.

In other words, in this case there is nothing to be gained by redundancy in the team. To end this section we mention an interesting correspondence between team and probabilistic function learning:

Proposition 2.18 *(Pitt (1984); Pitt and Smith (1988)) For each* $n \geq 1$ *and* $p \in (\frac{1}{n+1}, 1]$, $\mathbf{Prob^{p}Ex} \subseteq \mathbf{Team^{1}_{2}Ex}$.

This key correspondence has many interesting implications, again the reader is referred to Chapter 9 of Fulk et al. (1994). There are also related results for language learning.

2.4 Finite Elasticity

Proving learnability of a class of languages using Proposition 2.7 can be quite hard. Luckily there are some well-known sufficient conditions for learnability that can quite often easily shown to be satisfied. These conditions are expressed in terms of structural properties of classes to be learned, this section will discuss some of them.

If a class of languages has a limit point, there exists an *infinite ascending chain* of languages. This means that for $L_0, L_1, \ldots, L_n, \ldots$ in that class, $L_0 \subset L_1 \subset \ldots \subset L_n \subset \ldots$. This implies the weaker property known as *infinite elasticity*:

Definition 2.19 ***Infinite elasticity***
A class $\mathcal{L}$ *of languages is said to have* infinite elasticity *if there exists 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 2.20 ***Finite elasticity***
A class $\mathcal{L}$ *of languages is said to have* finite elasticity *if it does not have infinite elasticity.*

A related notion that is often useful is *finite thickness*; for any class with this property, any sentence can be a member of only a finite number of languages in that class. This obviously implies finite elasticity as well.

One other important property, much weaker than finite thickness, is finite fatness:

Definition 2.21 *A class of languages $\mathcal{L}$ is said to have* finite fatness *if every $L \in \mathcal{L}$ has a* strict telltale set *D in $\mathcal{L}$ (i.e., for all $L' \in \mathcal{L}$, $D \subseteq L'$ implies $L \subseteq L'$), and each $L \in \mathcal{L}$ is contained in only finitely many other languages in $\mathcal{L}$.*

Finite elasticity[13] is a sufficient condition for learnability[14] under two conditions, known as *Angluin's conditions*. These are quite natural conditions for linguistically plausible grammar formalisms, and very convenient when dealing with learnability issues. The first condition involves restricting ourselves to grammar systems for which universal membership is decidable (this obviously implies that all languages in the class are recursive):

Condition 2.22 ***Decidability of universal membership***
There is an algorithm for deciding whether $s \in \mathrm{L}(G)$, given $s \in \mathbf{S}$ and $G \in \Omega$.

This restriction is generally not problematic, since universal membership is almost always decidable for commonly used grammar systems (in fact it is generally assumed that all natural languages are context-sensitive).

The second condition restricts the class of grammars $\mathcal{G}$ to be learned. It requires that the question $G \in \mathcal{G}$ is at least semi-decidable.[15] This leads to the following condition:

Condition 2.23 ***Restriction to recursively enumerable classes***
The question of learnability has to be restricted to r.e. classes of grammars.

In Angluin (1980b)[16] learnability under these two restrictions was characterized:

Theorem 2.24 ***(Angluin)***
Let $\langle \Omega, \mathbf{S}, \mathrm{L} \rangle$ be a grammar system for which universal membership is decidable, and let $\mathcal{G}$ be an r.e. subset of Ω. Then $\mathcal{G}$ is learnable if and only if there is a computable partial function ψ: $\Omega \times \mathbf{N} \longmapsto \mathbf{S}$ with the following properties:

1. *For all $n \in \mathbf{N}$, $\psi(G, n)$ is defined if and only if $G \in \mathcal{G}$ and $\mathrm{L}(G) \neq \emptyset$.*

[13] The notions of both *finite* and *infinite elasticity* can be found in Wright (1989). The original definitions were incorrect, and were later corrected in Motoki et al. (1991).

[14] Note that this implies that *in*finite elasticity is a necessary condition for nonlearnability.

[15] To the best of the author's knowledge, up to now all concrete learnability results for grammar formalisms are for *recursive* classes, so in practice this condition is always met.

[16] In fact, Angluin (1980b) only considers the case were $\Omega = \mathcal{G}$, but her result easily generalizes to the present setting.

2. *For all* $G \in \mathcal{G}$, $T_G = \{\psi(G,n) \mid n \in \mathbf{N}\}$ *is a finite subset of* $\mathrm{L}(G)$, *called a* tell-tale subset.

3. *For all* $G, G' \in \mathcal{G}$, *if* $T_G \subseteq \mathrm{L}(G')$, *then* $\mathrm{L}(G') \not\subset \mathrm{L}(G)$.

In Wright (1989) Angluin's theorem[17] was used to prove the following:

Theorem 2.25 ***(Wright)***[18] *Let* $\langle \Omega, \mathbf{S}, \mathrm{L} \rangle$ *and* $\mathcal{G}$ *be as defined in Theorem 2.24. If* $\mathrm{L}(\mathcal{G})$ *has finite elasticity, then* $\mathcal{G}$ *is learnable.*

In de Jongh and Kanazawa (1996), Angluin's theorem is extended to deal with r.e. languages by introducing the concepts of *warning sets* and *unlocking sequences*.[19] This way Angluin's theorem is generalized to grammar classes with semi-decidable universal membership.

The study of learnability issues restricted to the setting just outlined is known as *learnability of indexed families*.[20]

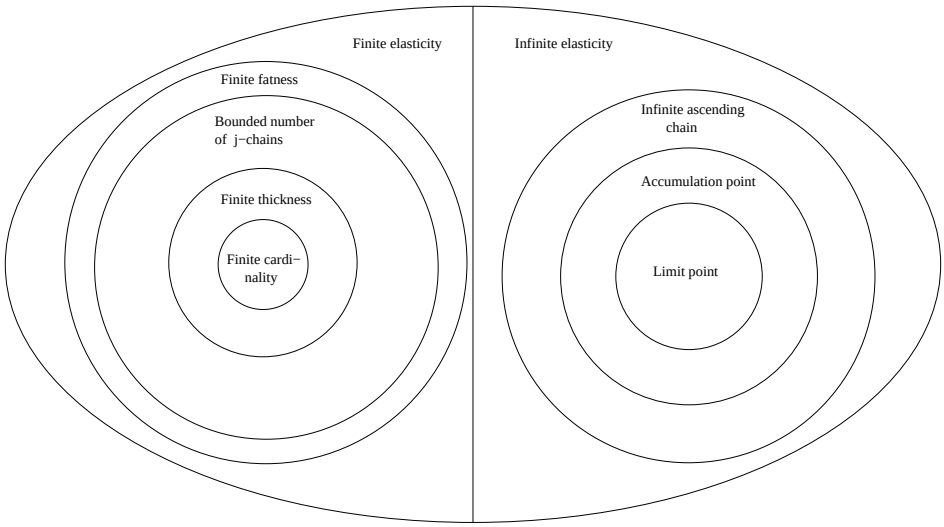


Figure 2.1: A learnability hierarchy for language classes.

Proposition 2.26 *(Kapur (1991))*
The language $L \in \mathcal{L}$ *has a tell-tale subset if and only if* L *is not an accumulation point.*

[17] In Osherson et al. (1986) a simple variant of this theorem is given that characterizes non-effective learnability by the existence of (non-effective) tell-tale sets.

[18] This is a generalization of a result from Angluin (1980b) which states that in this setting *finite thickness* is a sufficient condition for nonlearnability.

[19] This result is completely general in that every (effectively) learnable class is included in an (effectively) learnable uniformly r.e. class, see Fulk (1990a). This property is known as *r.e.-boundedness*.

[20] This is sometimes also referred to as the *Angluin/Wright paradigm*.

The following theorem is from Kanazawa (1994b), reproduced in Kanazawa (1998).[21] Let Σ and Υ be two alphabets, a relation $R \subseteq \Sigma^* \times \Upsilon^*$ is said to be *finite-valued* just if 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 2.27 *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.*

This theorem is very useful when dealing with the learnability of classes of grammars in some linguistic formalism with a clear and precise notion of derivation: for most formalisms the relation between string and possible derivation is finite-valued, and it is in general easier to prove finite elasticity of a class of derivation (structure) languages than of a class for string languages.

Figure 2.1 shows the relations between these important topological properties of language classes. The largest oval is the collection of all classes of (r.e.) languages, the vertical line divides this collection into classes with finite elasticity and classes with infinite elasticity. Note that there is no obvious relation between the Chomsky hierarchy and this learnability hierarchy, which graphically demonstrates the independence of properties pertaining to learnability and expressive power.

2.5 Elementary Formal Systems

As we have seen, Angluin's and Wright's work defines a learning theoretic framework that is suited for the learning of classes of formal languages. In order to develop a framework that is even more suited for this purpose the notion of learning a class of *elementary formal systems* (or *EFSs*), originally introduced in Smullyan (1961), was used in Arikawa et al. (1989). This framework was refined in, among others, Shinohara (1990a,b). EFS is basically a form of a (linear) logic programming language on Σ^+, and is a natural device for defining (language) formalisms. We will not go into too much detail, but some of the more striking results are presented in this section.

Definition 2.28 *A* definite clause *is a clause of the form $A \Leftarrow B_1, \ldots, B_n$, where $n \geq 0$, and $A, B_1, \ldots, B_n$ are atoms. We call atom A the* head *of a clause, and the sequence of atoms $B_1, \ldots, B_n$ the* body*.*

Definition 2.29 *A definite clause is called* variable bounded *if variables in $B_1, \ldots, B_n$ also appear in A.*

Let the length of an atom be the total length of the terms in it. We define length-boundedness as follows:

[21] This is in fact a generalization of a theorem from Wright (1989), which states that finite elasticity is closed under union.

Definition 2.30 *A definite clause is called* length-bounded *if the total length of $\sigma[B_1], \ldots, \sigma[B_n]$ does not exceed the length of $\sigma[A]$ for any substitution σ.*

Definition 2.31 *A clause C is* provable from *an EFS Γ, or $\Gamma \vdash C$, if C is obtained from Γ by finitely many applications of substitutions and modus ponens. More formally, the binary relation $\Gamma \vdash C$ is defined inductively as follows:*

1. *If $C \in \Gamma$ then $\Gamma \vdash C$.*
2. *If $\Gamma \vdash C$ then $\Gamma \vdash \sigma[C]$ for any substitution σ.*
3. *If $\Gamma \vdash A \Leftarrow B_1, \ldots, B_{n+1}$ and $\Gamma \vdash B_{n+1}$ then $\Gamma \vdash A \Leftarrow B_1, \ldots, B_n$*

2.5.1 Inductive Inference of Monotonic Formal Systems

One of the most interesting aspects of Shinohara's work is that it establishes a relation between learnability and *monotonicity* of the formalism used to represent the languages to be learned.

Definition 2.32 *A* concept defining framework *is a triple (U, E, M) of a universe U of objects, a universe E of expressions, and a semantic mapping M.*

Definition 2.33 *A class of concepts $C = R_1, R_2, \ldots$ is said to be an* indexed family of recursive concepts *if there exists a computable function*

$$f : \mathbb{N} \times U \to \{0,1\} \text{ such that } f(i,s) = \begin{cases} 1, & \text{if } s \in R_i, \\ 0, & \text{otherwise.} \end{cases}$$

Definition 2.34 *A semantic mapping M is* monotonic *if $\Gamma' \subseteq \Gamma \Rightarrow \mathrm{M}(\Gamma') \subseteq \mathrm{M}(\Gamma)$.*

Definition 2.35 *A formal system Γ is* reduced with respect to *a set $X \subseteq U$ if $X \subseteq M(\Gamma)$ but $X \not\subseteq M(\Gamma')$ for any $\Gamma' \subseteq \Gamma$.*

Definition 2.36 *A concept defining framework (U, E, M) has* bounded finite thickness *if M is monotonic, and* $\mathrm{card}(\{M(\Gamma) \mid \Gamma$ *is reduced with respect to* X, $\mathrm{card}(\Gamma) \leq n\}) < \infty$ *for any finite set $X \subseteq U$ and any $n \geq 0$.*

Note that finite thickness implies bounded finite thickness, but the converse does not hold. The next theorem is the main result of Shinohara (1990b):

Theorem 2.37 *Let a concept defining framework (U, E, M) have bounded finite thickness and $C^n = \{M(\Gamma) \mid \Gamma \subseteq E, |\Gamma| \leq n\}$. Then, the class C^n is inferable from positive data for any $n \geq 0$.*

Lemma 2.38 *Let $U = \Sigma^+$, E be the set of all length-bounded clauses, and $M(\Gamma) = L(\Gamma, p_0)$. Then (U, E, M) is a concept-defining framework that has bounded finite thickness.*

Corollary 2.39 *For any $n \geq 0$, the class of languages definable by length-bounded EFS's consisting of at most n clauses is inferable from positive data.*

The following is a result from Matsumoto et al. (1997) (Theorem 9):

Theorem 2.40 *Let $\mathcal{C}$ be a class. If $\mathcal{C}$ has finite thickness, and the membership problem and the minimal language problem for $\mathcal{C}$ are computable in polynomial time, then $\mathcal{C}$ is polynomial time inferable from positive data.*

2.5.2 Context-Sensitive Grammars

There is a straightforwarsd relation between EFS and Context-Sensitive Grammars (CSG). Since the latter is linguistically interesting, we will discuss this relation in some detail. Let Σ be a finite alphabet and V be a set disjoint from Γ. An element in V is called a *nonterminal symbol*. We assume that V contains a special nonterminal symbol S_0.

Definition 2.41 *A* production *is an expression of the form $\alpha \to \beta$, where $\alpha, \beta \in (\Sigma \cup V)^+$.*[22] *A* grammar *is a finite set of productions. A production $\alpha \to \beta$ is said to be* context-sensitive *if $|\alpha| \leq |\beta|$. A context-sensitive grammar is a grammar whose productions are all context-sensitive.*

Definition 2.42 *Let Σ be a grammar. We define a binary relation $\Rightarrow_\Gamma$ on $(\Sigma \cup V)^+$ by $\gamma\alpha\delta \Rightarrow_\Gamma \gamma\beta\delta$ if $\alpha \to \beta \in \Gamma$, where $\alpha, \beta \in (\Sigma \cup V)^+, \gamma, \delta \in (\Sigma \cup V)^*$. By $\Rightarrow_\Gamma{}^*$ we denote the reflexive transitive closure of $\Rightarrow_\Gamma$. The* language $\mathrm{L}(\Gamma)$ *of a grammar Γ is defined by $\mathrm{L}(\Gamma) = \{w \in \Sigma^+ \mid S_0 \Rightarrow_\Gamma{}^* w\}$, where S_0 is the start symbol of Γ.*

Given these definitions, it should be clear that the following is a direct consequence of Corollary 2.39:

Corollary 2.43 *For any $n \geq 0$, the class of languages definable by context-sensitive grammars consisting of at most n productions is inferable from positive data.*

2.5.3 Linear Prolog Programs

As a final demonstration of the usefulness of EFS in learning theory we briefly discuss Shinohara's results for Linear Prolog Programs, which have applications in the field of Inductive Logic Programming.

Definition 2.44 *A* term *is a variable, a constant symbol, or an expression of the form $f(t_1, \ldots, t_n)$, where f is a function symbol with arity $n \geq 1$ and $t_1, \ldots, t_n$ are terms. A* ground term *is a term that does not contain any variable.*

[22] Note that this implies that the righthand-side of a CSG is always non-empty, and thus that the emty string is not a context-sensitive language.

Definition 2.45 *The* length *of a term t, denoted by $|t|$, is defined inductively as follows:*

1. *If t is a variable or a constant symbol, then $|t| = 1$.*
2. *If $t = f(t_1, \ldots, t_n)$, then $|t| = |t_1| + \ldots + |t_n| + 1$.*

Definition 2.46 *A definite clause $A \rightarrow B_1, \ldots, B_n$ is* linear *if $|\sigma[A]| \geq |\sigma[B_i]|$ for any substitution σ and any $i = 1, \ldots, n$. A program Γ is* linear *if all clauses in Γ are linear.*

Given these definitions, it should be clear that the following is a direct consequence of Corollary 2.39 as well:

Corollary 2.47 *For any $n \geq 1$, the class of minimal models of linear programs consisting of at most n clauses is inferable from positive data.*

2.6 Constraints on Learning Functions

In the definition of learnability nothing is said about the behaviour of learning functions apart from convergence to a correct grammar, it grants learners total freedom in their behavior prior to convergence. Further constraints can be imposed, i.e. one can choose a certain *learning strategy*. Intuitively, a strategy refers to a *policy*, or *preference*, for choosing hypotheses (note that a strategy is not an algorithm!). Formally, a strategy can be analyzed as merely picking a subset (not necessarily proper) of possible learning functions, and thus computability and complexity measures can be regarded as constraints as well. A strategy is said to be *restrictive* if it constrains the class of learnable languages.

The strategies found in the literature can be roughly classified as constraints on the use of *resources* (computability, time complexity, memory limitation), constraints on potential conjectures (consistency), and constraints on the relation between conjectures (conservatism etc). Many different constraints have been proposed, we will only discuss some of the more relevant ones in this section.

The proof of Theorem 2.24 implies that in a grammar system where universal membership is decidable, a recursively enumerable class of grammars is learnable if and only if there is a computable learning function that learns it order-independently, prudently, and is responsive and consistent on this class.

Definition 2.48 ***Order-independent learning***
A learning function φ learns $\mathcal{G}$ order-independently *if for all $L \in \mathrm{L}(\mathcal{G})$, there exists a $G \in \mathcal{G}$ such that $\mathrm{L}(G) = L$ and for all infinite sequences $\langle s_i \rangle_{i \in \mathbb{N}}$ that enumerate L, φ converges on $\langle s_i \rangle_{i \in \mathbb{N}}$ to G.*

Intuitively this seems a reasonable strategy, there does not seem to be any a priori reason why the order of presentation should influence the final choice of hypothesis. In fact, it turns out this constraint (by itself) is not restrictive.

Definition 2.49 ***Exact learning***
A learning function φ learns $\mathcal{G}$ exactly *if for all $\mathcal{G}'$ such that φ learns $\mathcal{G}'$, $\mathrm{L}(\mathcal{G}') \subseteq \mathrm{L}(\mathcal{G})$.*

In other words, the learning function will respond succesfully to all languages in a given class and respond unsuccesfully to all other languages. Note that this is a constraint on the *relation* between a class of languages and a learning function.

The rationale for this constraint is the idea that natural languages form the *largest* collection of child-learnable languages. Thus, for every nonnatural language there must be some text on which children fail to converge to a correct index.

If we want to model human language learning, we want learning functions to learn a chosen class *exactly*. There seems to be empirical support for this idea. Some of it comes from studies of children raised in pidgin dialects (Sankoff and Brown (1976)), some from studies of sensory deprived children (Feldman et al. (1978)). It seems that children are not capable of learning very inexpressive languages.

Definition 2.50 ***Prudent learning***
A learning function φ learns $\mathcal{G}$ prudently *if φ learns $\mathcal{G}$ and $\mathrm{range}(\varphi) \subseteq \mathcal{G}$. Note that this implies exact learning.*

A prudent learning function only conjectures grammars for languages it is prepared to learn. This condition is not restrictive, also note that if φ learns $\mathcal{G}$ prudently, φ learns $\mathcal{G}$ exactly.

Definition 2.51 ***Responsive learning***
A learning function φ is responsive *on $\mathcal{G}$ if for any $L \in \mathrm{L}(\mathcal{G})$ and for any finite sequence $\langle s_0, \ldots, s_i \rangle$ of elements of L ($\{s_0, \ldots, s_i\} \subseteq L$), $\varphi(\langle s_0, \ldots, s_i \rangle)$ is defined.*

This constraint can be regarded as the complement of prudent learning: if all sentences found in the input are in a language in the class of languages learned, the learning function should always produce a hypothesis.

Definition 2.52 ***Consistent learning***[23]
A learning function φ is consistent *on $\mathcal{G}$ if for any $L \in \mathrm{L}(\mathcal{G})$ and for any finite sequence $\langle s_0, \ldots, s_i \rangle$ of elements of L, either $\varphi(\langle s_0, \ldots, s_i \rangle)$ is undefined or $\{s_0, \ldots, s_i\} \subseteq \mathrm{L}(\varphi(\langle s_0, \ldots, s_i \rangle))$.*

The idea behind this constraint is that all the data given should be explained by the chosen hypothesis. It should be self-evident that this is a desirable property. Indeed, one would almost expect it to be part of the definition of

[23]Note that consistency is only nonrestrictive for non-effective learning and for effective learning under Angluin's conditions.

learning. However, learning functions that are not consistent are not necessarily trivial. If, for example, the input is noisy, it would not be unreasonable for a learning function to ignore certain data because it deems it unreliable. It may also be reasonable to be give up consistency for efficiency, as will be discussed in Section 2.9.

Also, it is a well known fact that children do not learn languages consistently.

Definition 2.53 ***Set-driven learning***
A learning function φ learns $\mathcal{G}$ set-driven *if $\varphi(\langle s_0, \ldots, s_i \rangle)$ is determined by $\{s_0, \ldots, s_i\}$, or, more precisely, if the following holds: Whenever $\{s_0, \ldots, s_i\} = \{u_0, \ldots, u_j\}$, $\varphi(\langle s_0, \ldots, s_i \rangle)$ is defined if and only if $\varphi(\langle u_0, \ldots, u_i \rangle)$ is defined, and if they are defined, they are equal. (It is easy to see that set-drivenness implies order-independence.)*

The collection of classes that are learnable under this constraint is written as $[\mathbf{TxtEx}]^{\text{set-driven}}$.

Set-driven learning could be very loosely described as order-independent learning with the added bonus of ignoring 'doubles' in the input. It is obvious that this is a nice property for a learning function to have: one would not expect the choice of hypothesis to be influenced by repeated presentation of the same data.

The assumption here is that the order of presentation and the number of repetitions are essentially arbitrary, i.e. they carry no information that is of any use to the learning function. One can devise situations where this is not the case.

Note that in Osherson et al. (1986) it is shown that learnable families of infinite languages are always learnable by a set-driven learner.

Definition 2.54 ***Conservative learning***
A learning function φ is conservative *if for any finite sequence $\langle s_0, \ldots, s_i \rangle$ of sentences and for any sentence s_{i+1}, whenever $\varphi(\langle s_0, \ldots, s_i \rangle)$ is defined and $s_{i+1} \in \mathrm{L}(\varphi(\langle s_0, \ldots, s_i \rangle))$, $\varphi(\langle s_0, \ldots, s_i, s_{i+1} \rangle)$ is also defined and $\varphi(\langle s_0, \ldots, s_i \rangle) = \varphi(\langle s_0, \ldots, s_i, s_{i+1} \rangle)$.*

The collection of classes that are learnable under this constraint is written as $[\mathbf{TxtEx}]^{\text{cons}}$.

At first glance conservatism may seem a desirable property. Why change your hypothesis if there is no direct need for it? One could imagine cases, however, where it would not be unreasonable for a learning function to change its mind, even though the new data fits in the current hypothesis. Such a function could for example make reasonable but 'wild' guesses which it could later retract. The function could 'note' after a while that the inputs cover only a proper subset of its conjectured language. While such behaviour will sometimes result in temporarily 'overshooting', such a function could still be guaranteed to converge to the correct hypothesis in the limit.

Evidence that children are not conservative learners can be found in Mazurkewich and White (1984).

It is a common assumption in cognitive science that human cognitive processes can be simulated by computer. This would lead one to believe that children's learning functions are computable. The corresponding strategy is the set of all partial and total recursive functions.

Since this is only a subset of all possible functions, the computability strategy is a nontrivial hypothesis, but not necessarily a restrictive one. (For details, see Fulk (1988).)

The computability constraint interacts with consistency:

Proposition 2.55 *(See Fulk (1988)) There is a collection of languages that is identifiable by a computable learning function but by no consistent, computable learning function.*

The computability constraint also interacts with conservative learning:

Proposition 2.56 *(Angluin, 1980) There is a collection of languages that is identifiable by a computable learning function but by no conservative, computable learning function.*

Definition 2.57 ***Monotonicity***
The learning function φ is monotone increasing *if for all finite sequences $\langle s_0, \ldots, s_n\rangle$ and $\langle s_0, \ldots, s_{n+m}\rangle$, whenever $\varphi(\langle s_0, \ldots, s_n\rangle)$ and $\varphi(\langle s_0, \ldots, s_{n+m}\rangle)$ are defined,* $\mathrm{L}(\varphi(\langle s_0, \ldots, s_n\rangle)) \subseteq \mathrm{L}(\varphi(\langle s_0, \ldots, s_{n+m}\rangle))$.
This property is also known as strong monotonic ($[\mathbf{TxtEx}]^{\text{s-mon}}$).

Variants of strong monotonicity have been considered, among which:

- monotonicity ($[\mathbf{TxtEx}]^{\text{mon}}$) on $\mathcal{L}$, which requires that $\mathrm{L}(\varphi(\sigma)) \cap \mathrm{L}, \subseteq \mathrm{L}(\varphi(\sigma'))$, where $\mathrm{L} \in \mathcal{L}$,
- weak monotonicity ($[\mathbf{TxtEx}]^{\text{w-mon}}$), which requires that $\mathrm{range}(\sigma') \subseteq \mathrm{L}(\varphi(\sigma)) \Rightarrow \mathrm{L}(\varphi(\sigma)) \subseteq \mathrm{L}(\varphi(\sigma'))$,
- *duals* of these constraints: $[\mathbf{TxtEx}]^{\text{d-s-mon}}$ is dual strong monotonicity, $[\mathbf{TxtEx}]^{\text{d-w-mon}}$ dual weak monotonicity, etc, the definition of these duals are obtained by replacing all occurences of $\subset$ and $\subseteq$ in the definitions with $\supset$, $\supseteq$, respectively.

These monotonic strategies are all *generalization* strategies, their duals are *specialization* strategies. Note that $[\mathbf{TxtEx}]^{\text{d-w-mon}}$ is *not* restrictive, i.e. every identifiable collection of languages can be identified by a dual-weak-monotonic scientist, see Kinber and Stephan (1995b).

When a learning function that is monotone increasing changes its hypothesis, the language associated with the previous hypothesis will be (properly) included in the language associated with the new hypothesis. There seems to be little or no empirical support for such a constraint.

Definition 2.58 ***Incrementality***
Kanazawa *The learning function φ is* incremental *if there exists a computable function ψ such that $\varphi(\langle s_0, \ldots, s_{n+1}\rangle) \simeq \psi(\varphi(\langle s_0, \ldots, s_n\rangle), s_{n+1})$.*

An incremental learning function does not need to store previous data. All it needs is current input, s_n, and its previous hypothesis. A generalized form of this constraint, called *memory limitation*, limits access for a learning function to only n previous elements of the input sequence. This seems reasonable from an empirical point of view; it seems improbable that children (unconsciously) store all utterances they encounter.

Note that, on an infinite sequence enumerating language L in $\mathrm{L}(\mathcal{G})$, a conservative learning function φ learning $\mathcal{G}$ never outputs any grammar that generates a proper superset of L.

Let φ be a conservative and computable learning function that is responsive and consistent on $\mathcal{G}$, and learns $\mathcal{G}$ prudently. Then, whenever $\{s_0, \ldots, s_n\} \subseteq L$ for some $L \in \mathrm{L}(\mathcal{G})$, $\mathrm{L}(\varphi(\langle s_0, \ldots, s_n\rangle))$ must be a minimal element of $\{L \in \mathrm{L}(\mathcal{G}) \mid \{s_0, \ldots, s_n\} \subseteq L\}$. This implies the following:

Condition 2.59 *There is a computable partial function ψ that takes any finite set D of sentences and maps it to a grammar $\psi(D) \in \mathcal{G}$ such that $\mathrm{L}(\psi(D))$ is a minimal element (with respect to inclusion) of $\{L \in \mathrm{L}(\mathcal{G}) \mid D \subseteq L\}$, whenever the latter set is non-empty.*

Definition 2.60 *Let ψ be a computable function satisfying Condition 2.59. Define a learning function φ as follows:*

$$\varphi(\langle s_0\rangle) \simeq \psi(\{s_0\}),$$
$$\varphi(\langle s_0, \ldots, s_{i+1}\rangle) \simeq \begin{cases} \varphi(\langle s_0, \ldots, s_i\rangle) & \text{if } s_{i+1} \in \mathrm{L}(\varphi(\langle s_0, \ldots, s_i\rangle)), \\ \psi(\{s_0, \ldots, s_{i+1}\}) & \text{otherwise.} \end{cases}$$

Under certain conditions the function defined in 2.60 is guaranteed to learn $\mathcal{G}$, one such case is where $\mathrm{L}(\mathcal{G})$ has finite elasticity.

Proposition 2.61 *Let $\mathcal{G}$ be a class of grammars such that $\mathrm{L}(\mathcal{G})$ has finite elasticity, and a computable function ψ satisfying Condition 2.59 exists. Then the learning function φ defined in Definition 2.60 learns $\mathcal{G}$.*

The following constraints on learning functions are not really used in the rest of this paper, they are presented to get a 'feel' for the relations between various properties of learning functions.

Definition 2.62 ***Efficiency (Gold, 1967)***
The learning function φ_0 identifies L strictly faster *than φ_1 if and only if*

1. *both φ_0 and φ_1 identify L;*

2. *for every text t for every $L \in \mathrm{L}$, the convergence point for φ_0 on t is no greater than that for φ_1 on t;*

3. *for some text t for some $L \in \mathrm{L}$, the convergence point for φ_0 on t is smaller than that for φ_1 on t.*

The function φ_0 identifies L efficiently *if and only if φ_0 identifies L, and no learning function φ_1 identifies L strictly faster than φ_0.*

Text-efficiency is not restrictive, i.e. any learnable class is text-efficiently learnable. Note that any prudent, consistent and conservative learner is text-efficient (Proposition 8.22A in Osherson et al. (1986)), but the reverse does not hold. Also note that this property has nothing to do with the computational complexity of the learning functions, 'fast' is defined strictly in terms of length of text. It is also known as *text-efficiency*, we will postpone the discussion of efficiency to Section 2.9.

Definition 2.63 ***Totality***
The most natural constraint on a conjecture is that it exists. The corresponding strategy is the set of total learning functions.

Definition 2.64 ***Nontriviality***
The learning function φ is called nontrivial *if and only if for all $\sigma \in$ SEQ, $\mathrm{L}(\varphi(\sigma))$ is infinite.*

Linguists rightly emphasize the infinite quality of natural languages; apparently, no natural language includes a longest sentence. If this universal feature of natural language corresponds to an innate constraint on children's linguistic hypotheses, then children would be barred from conjecturing a grammar for a finite language. Such a constraint on potential conjectures amounts to a strategy[24].

It is easy to see that it is possible for a class $\mathcal{L}' = \bigcup\{L \in \mathcal{L} \mid L \text{ is infinite}\}$ to be learnable, even though $\mathcal{L}$ is not learnable. This is the case, for example, if $\mathcal{L}$ is not learnable only because it contains both an infinite ascending chain of finite languages and an infinite language L that is the union of all the languages in this chain.

Note that there are collections of infinite languages that are identifiable by recursive learning function but by no nontrivial, recursive learning function.

Definition 2.65 *Let $\sigma \in$ SEQ. The result of removing the last member of σ is denoted by σ^-; if $|\sigma| = 0$, then $\sigma^- = \sigma$.*

[24]There also exists epistemological motivation for disallowing finite languages as conjectures. The constraint called *accountability* (Osherson et al. (1986)) demands that all hypotheses of a learning function are always subject to further confirmation. It is easy to see that finite languages cannot be identified by accountable learners.

An analogous constraint, called *Popperian* learning, is defined in Case and Ngo-Manuelle (1979). For a discussion on the relation between these constraints, see Osherson et al. (1986).

For $n \in \mathbb{N}$, the result of removing all but the last n members of σ is denoted $\sigma^- n$; if $|\sigma| < n$, then $\sigma^- n = \sigma$.

Definition 2.66 ***Memory limitation (see Wexler and Culicover (1980))*** *For all $n \in \mathbb{N}$, φ is* n-memory limited *if and only if for all $\sigma, \tau \in$ SEQ, if $\sigma^- n = \tau^- n$ and $\varphi(\sigma^-) = \varphi(\tau^-)$, then $\varphi(\sigma) = \varphi(\tau)$. If φ is n-memory limited for some $n \in \mathbb{N}$, then φ is said to be* memory limited.

The collection of classes that are learnable under this constraint is written as $[\mathbf{TxtEx}]^{n\text{-lim}}$ (some $n \in \mathbb{N}$).

Thus φ is n-memory limited just in case its conjecture depends only on its last conjecture and the n latest members of σ.

Obviously, n memory limitation with $n = 1$ is equivalent to the definition of incrementality. It is called '1-memory limited' in Osherson et al. (1986).

In Kinber and Stephan (1995b) (or Kinber and Stephan (1995a)) it was shown that every class that is learnable by a set-driven function is learnable by means of a conservative function that uses an amount of memory that is linear in the amount of presented data.[25] The inclusion is strict, i.e. there are conservatively learnable classes that cannot be learned by a set-driven function. It was also shown that families learnable by a memory limited function are exactly those learnable by a set-driven function. Figure 2.2 gives an overview of known results,[26] where f and g are incomparable functions with $c \leq f, g \leq \mathrm{id}$:

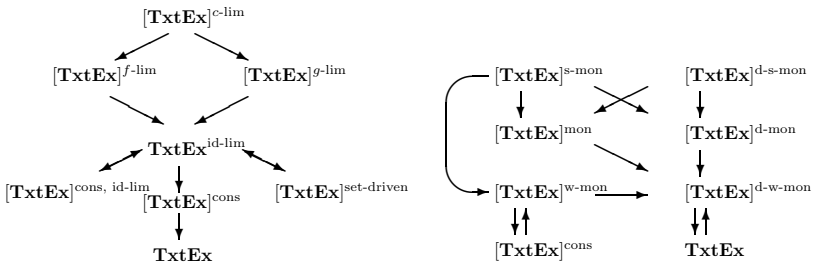


Figure 2.2: An overview of results from Kinber and Stephan (1995a).

Here $[\mathbf{TxtEx}]^{f\text{-lim}}$ denotes the language classes learnable with long term memory limited by $f, \forall x[f(x) \geq \mathrm{id}(x)]$, where id denotes the amount of memory needed to store range(σ) of any text σ.

Let $\mathcal{A}$ and $\mathcal{B}$ be two learning criteria. Then every $\mathcal{A}$ learnable (with k mindchanges) $\mathcal{L}$ is also $\mathcal{B}$ learnable (with k mindchanges) if and only if there is a transitive chain of arrows from $\mathcal{A}$ to $\mathcal{B}$ in the diagram.

[25] This kind of constraints on the use of computational resources, like memory-limited learning, are formally no different from constraints on the behaviour of a learning function: they specify a subset of all possible learners.

[26] Some of these results are taken from other sources or are generalizations of earlier results, see references cited in Kinber and Stephan (1995a).

If there is no such chain, then there is a class of sets which is $\mathcal{A}_k^{c\text{-lim}}$ learnable but not $\mathcal{B}$ learnable; there is a class of sets which is $\mathcal{A}_k^{c\text{-lim}}$ and $\mathcal{B}^{\text{id-lim}}$ learnable but not $\mathcal{B}^{f\text{-lim}}$ learnable for any $f \leq \text{id} - 2\text{-lim}$. It is always possible to take $k = 3$ mindchanges and $c = 2$ bits of long-term memory.

Also note that all classes learnable with constant long-term memory are also learnable with constantly many mindchanges (Theorem 5.2 in Kinber and Stephan (1995a)).

2.6.1 Constraints on Environments

It is possible to impose constraints on the learning environment as well as on learning functions. These constraints can be restrictive, but they can also *help* learning, i.e. *expand* the class of learnable languages:

Definition 2.67 ***Fat text***
A text t is fat *if and only if for all $x \in \text{content}(t), \{n \mid t(n) = x\}$ is infinite. The function φ identifies $\mathcal{L}$ on fat text if and only if for every fat text t for any $\text{L} \in \mathcal{L}$, φ identifies* L. *In this case, $\mathcal{L}$ is* identifiable on fat text.

Proposition 2.68 *Suppose that a collection $\mathcal{L}$ of languages is identifiable. Then some memory limited learning function identifies $\mathcal{L}$ on fat text.*

Thus, constraining the environment to fat text makes memory-limitedness a non-restrictive constraint.

Another constraint on environments is the introduction of *errors* in the text. Errors of two sorts can occur in text; on the one hand, ungrammatical strings may appear in the text, on the other hand, certain grammatical strings may never appear. This leads to the following definitions of *noisy text* and *incomplete text*:

Definition 2.69 *Let language L and text t be given.*

1. *t is a* noisy *text for L just in case there is finite $D \subset N$ such that t is a text for $L \bigcup D$.*

2. *t is an* incomplete *text for L just in case there is finite $D \subset N$ such that t is a text for $L - D$.*

3. *Learning function φ identifies L* on noisy text *just in case for every noisy text T for L, φ converges on t to an index for L. φ identifies collection* **L** of languages *on noisy text* just in case φ identifies every $L \in \mathbf{L}$ on noisy text.

4. Learning function φ identifies L *on incomplete text* just in case for every incomplete text T for L, φ converges on t to an index for L. φ identifies collection **L** of languages *on incomplete text* just in case φ identifies every $L \in \mathbf{L}$ on incomplete text.

Proposition 2.70 *There is a collection* **L** of languages with the following properties:

1. Every $L \in \mathbf{L}$ is infinite.

2. Every distinct pair of languages in **L** is disjoint.

3. Some computable learning function identifies **L** (on ordinary text).

4. No computable learning function identifies **L** on noisy text.

A parallel fact holds for incompletion. It is shown in Fulk et al. (1994) (Theorem 1) that incompletion is substantially more disruptive for identification than is noise.

2.7 Variations on Identification in the Limit

Identification proposes quite strict criteria of hypothesis stability and accuracy, so many liberalizations have been examined. For example, weaker criteria of stability might allow successful learners to switch indefinitely often among indices for the same language, or alternatively, to cycle among some finite set of them. (See Osherson et al. (1986), Jain et al. (1989).) Weaker criteria of accuracy might allow a finite number of errors into the final conjecture, or allow the final conjecture to 'approximate' the target in a variety of senses.

These and other liberalizations have been studied extensively, both separately and in combination. In this thesis we will stick to the standard concept of identifiability in the limit, without restrictions on the learning environment.

2.8 Algorithms for Learning

Since the paradigm **TxtEx** is restricted to computable learning functions it is natural to ask whether general learning algorithms exist.

Consider the strategy of *identification by enumeration*: the function has access to a fixed (well-ordered)[27] list of possible hypotheses, where this order respects inclusion[28], and at any time the current conjecture is the first on the list that is consistent with the current data. In Gold (1967) it was shown that each r.e. indexable class of computable functions can be identified this way, and Gold even conjectured that enumeration is a universal strategy for function

[27] A set S is said to be well-ordered by a relation R if R is a total order and every nonempty subset of S has a least element.

For example, $\mathbb{N} = \{0, 1, \ldots, \}$ is well-ordered, whereas $\mathbb{Z} = \{\ldots, -1, 0, 1, \ldots\}$ is not.

[28] An order $>$ respects inclusion just if $X > Y$ implies $Y \not\subset X$.

learning.[29] This conjecture can be refuted by example: Wiehagen's class of self-describing functions is identifiable, but not by an enumerator. A more sophisticated version of this conjecture was formulated by Bārzdiņš and was later refuted in Fulk (1990b). Interestingly, this refutation yields an algorithm that is strictly more general than enumeration. In Kapur (1991) it was shown that there is no general algorithm for identification from positive data of all learnable classes.

However, when the classes under consideration are restricted to indexed families of recursive languages, and some additional conditions are satisfied, a general (consistent and conservative) algorithm exists. It is based on choosing a grammar generating a minimal language from a collection of grammars consistent with the input, see Kapur (1991) for details.

2.8.1 Uniform Learning: Synthesizing Learners

Some work has been done on the problem of *universal* or *uniform* learners, that is functions that can generate a learner for a class $\mathcal{G}$ given a description of $\mathcal{G}$. In the approach of uniform learning one tries to solve a collection of such learning problems, i.e. the instance of the new problem is a set of descriptions for several problems of the classical type. The goal is to find a single "master" strategy simulating all the strategies for the classical problems referred to in the description set. For extensive discussion and recent results see e.g. Zilles (2001a, 2002, 2001b).

Though this subject may seem abstract and remote from the 'everyday concerns' of, for example, an acquisitionist, consider this: in Yang (1999) the nativist model of language learning is sketched as a function that takes data and some explicit description of the learning problem. This implies that, given another description, this function would be able to learn something else, perhaps not remotely resembling the class of natural languages. Therefore, under this interpretation, a nativist would claim that (very) general learning algorithms exist, which obviously goes against the general spirit of nativism.

The subject also holds interest from a practical point of view. Consider the context of parser-design, an algorithm designed for parsing just one particular language is in general not deemed to be very useful, instead one is more interested in *parser generators* that generate a parser from a description of a language (i.e. a grammar) which is necessarily restricted to some (given) class.

[29] One frequently encounters the misconception that enumeration is a universal strategy for *language* learning (cf Pinker (1979); Atkinson (1992)). It is easy to refute this claim: even the class COFIN $= \{\{n, n+1, n+2, \ldots\} \mid n \in \mathbb{N}\}$ is not identifiable from positive data by enumeration, since this class constitutes an *infinite descending chain*, and thus has no minimal language. Therefore no well-order over these languages can be given that respects the inclusion relation. The class is obviously learnable: if we take $n \in \mathbb{N}$ to be an index for the language $\{n, n+1, n+2, \ldots\}$, the function that yields the smallest number from a sequence of numbers learns COFIN. See Gold (1967) for conditions under which enumeration is an appropriate strategy.

Note that Gold credits Solomonoff (1964) for introducing identification by enumeration.

It has been shown that there is *no* algorithm for translating any class $\mathcal{L}$ consisting of just a *pair* of grammars into a learning algorithm which **TxtEx**-identifies $\mathcal{L}$, even though any finite class can be **TxtEx**-identified, see Osherson et al. (1988); Kapur (1991) for details.

There has been some research into synthesizing learners for noisy data, see Case and Jain (1998); Case et al. (1997a). For learning from noisy data also see Case et al. (1997b).

2.9 Time Complexity of Learning Functions

In formal learnability theory there are no a priori constraints on the computational resources required by the learning function. In Jain et al. (1999) a whole chapter has been devoted to complexity issues in identification, where it is noted that there is a close relationship between the complexity of learning and computational complexity of functionals and operators. Defining the latter is a complex problem and still an active area of research. It is therefore no surprise that only partial attempts have been made at modeling the complexity of the identification process. Some examples are given that are based on bounding the number of mind changes of a learner (The existence of an ordinal mind change bound gives a measure for the tractability of learning a class), or bounding the number of examples required before the onset of convergence. These definitions do not seem to be directly related to any 'computational' notion of complexity. Ideally, such a constraint would satisfy some obvious intuitions about what constitutes tractability: for example, in the worst case a learning function should converge to a correct solution in polynomial time with respect to the size of the input. Such definitions are not directly applicable, since the input is not guaranteed to be helpful, for example it can start with an unbounded number of presentations of the same sentence. In full generality there can never be a bound on the number of time-steps before convergence, so such a constraint poses no bounds on computation time whatsoever.

It turns out that giving a usable definition of the complexity of learning functions is not at all easy. In this subsection some proposals and their problems will be discussed, and the choice for one particular definition will be motivated.

In Daley and Smith (1986) a measure of learning complexity is developed that is based on an integral defining the amount of work a learning function needs to do to converge on a function. This approach is based on Blum measures, and may thus admit pathological cases.

In Gold (1967) a definition of efficiency for learning functions known as *text-efficiency* is given: a function φ identifies L *(text-)efficiently* just if there exists no other function that, for every language in L, given the same text, converges at the same point as φ or at an earlier point.

Formally this can be simply regarded as a constraint. Note that this property has nothing to do with the *computational* complexity of learning functions, 'faster' is defined strictly in terms of length of text.

Although the text-efficiency constraint seems to correspond to a rational learning strategy, by itself it is hardly restrictive. Every learnable class is text-efficiently learnable. Also, it is a *qualitative* rather than a *quantative* measure, and there is no direct connection between text-efficiency and time complexity. Text-efficiency seems to be of limited interest to the present discussion.[30]

Let the complexity of the update-time of some (computable) learning function φ be defined as the number of computing steps it takes to learn a language, with respect to $|\sigma|$, the size of the input sequence. In Pitt (1989) it was first noted that requiring the function to run in a time polynomial with respect to $|\sigma|$ does not constitute a significant constraint, since one can always define a learning function φ' that combines φ with a clock so that its amount of computing time is bounded by a polynomial over $|\sigma|$. Obviously, φ' learns the same class as φ, and it does so in polynomial update-time.[31]

The problem here is that without additional constraints on φ the 'burden of computation' can be shifted from the number of computations the function needs to perform to the amount of input data considered by the function.[32] Requiring the function to be consistent already constitutes a significant constraint when used in combination with a complexity restriction (see Bārzdiņš (1974)). In fact, this a common theme in the work of Wiehagen, who calls it the consistency effect.

Some monotone strategies seem to have the same effect. See Stein (1998) for a discussion of consistent polynomial-time identification.

Applying the notion of reduction from recursion theory to learning has recently received some attention. This approach, known as *intrinsic complexity*, was introduced in Freivalds et al. (1995) for function learning. The reductions are based on the idea that a class $\mathcal{L}$ is reducible to a class $\mathcal{L}'$ just if there is an enumeration operator that transforms any text t for $L \in \mathcal{L}$ to a text t' for $L' \in \mathcal{L}'$ and an enumeration operator that transforms admissible sequences for t' to admissible sequences for t. There is a close connection between structural properties and this notion of complexity: all classes that can be identified by a learner that can confirm its success can be reduced to the collection of singleton languages, all classes that can be identified with no more than n mind changes are reducible to FIN_{n+1} (the collection of all languages with cardinality less than or equal to $n+1$). It was also shown that a class $\mathcal{L}$ with k pairwise independent j-chains[33] cannot be (weakly) reduced to a class $\mathcal{L}'$ that contains only finite languages and has less than k pairwise independent j-chains. Intrinsic complexity has been proposed as a complexity measure

[30] In fact a whole section devoted to this subject in Osherson et al. (1986) has been completely omitted from the second edition Jain et al. (1999).

[31] To be more precise: in Daley and Smith (1986) it was shown that any unbounded monotone increasing update boundary is not by itself restrictive.

[32] Similar issues seem to be important in the field of *computational learning theory* (see Kearns and Vazirani (1994) for an introduction). The notion *sample complexity* from this field seems closely related to the notions of text- and data-efficiency. There also exists a parallel with our notion of (polynomial) update-time.

[33] See Figure 2.1 for the relation between this and other structural properties.

precisely because it abstracts away from computational aspects. This can be an advantage, depending on what the researcher is interested in. In the present case it is obviously a drawback.

In Angluin (1979), *consistent and conservative learning with polynomial time of updating conjectures* was proposed as a reasonable criterion for efficient learning. The consistency and conservatism requirements ensure that the update procedure really takes all input into account. It is interesting to note that a conservative (and prudent) learner that is consistent on its class is text-efficient.[34] This definition was applied in Arimura et al. (1992) to analyze the complexity of learning a subclass of context-free transformations. The following theorem gives a sufficient condition for efficient learnability. It has been taken from Matsumoto et al. (1997) where it was used to analyze the complexity of learning regular term tree languages (Theorem 9 which is attributed to Angluin (1980a) and Shinohara (1986)):

Theorem 2.71 *Let $\mathcal{L}$ be a class of languages. If $\mathcal{L}$ has finite thickness, and the membership problem and the minimal language problem for $\mathcal{L}$ are computable in polynomial time, then $\mathcal{L}$ is polynomial time inferable from positive data.*[35]

A closely related notion is due to Pitt (1989):

Definition 2.72 *A class of acceptors $\mathcal{M}$ is* polynomial-time learnable in the limit from positive data *if*

1. *$\mathcal{M}$ is learnable in the limit from positive data,*

2. *the learning algorithm for $\mathcal{M}$ satisfies the property that there exist polynomials p, q such that for any M of size $n, n \in \mathbb{N}^+$, and for any positive presentation of $\mathrm{L}(M)$, the time used by the algorithm between receiving the i-th example w_i and outputting the i-th conjectured acceptor M_i is at most $p(n, l_1 + \ldots + l_i)$, and the number of implicit errors of prediction made by the algorithm is at most $q(n)$, where $l_j = |(w_j), j \in \mathbb{N}^+$.*

To the best of our knowledge, no non-trivial class of acceptors or languages has been shown to be efficiently learnable according to this definition, even the class of all DFAs recognizing just finite languages does not meet the requirements (Angluin (1990)). The following was demonstrated in Yokomori (1995):

[34] It is interesting to note that a conservative (and prudent) learner that is consistent on its class is text-efficient (Proposition 8.2.2 A in Osherson et al. (1986)). Therefore, the conservative learning functions $\varphi_{k\text{-valued}}$, $\varphi_{\text{least-valued}}$ and $\varphi_{\text{least-card}}$ defined in Kanazawa (1998) that are consistent on their class are all text-efficient.

[35] Polynomial time inference is defined in Matsumoto et al. (1997) as 'there exists a consistently, responsively and conservatively working learning function that has polynomial update time'. This corresponds to the Angluin-style definition of efficient learning.

Theorem 2.73 *Let $\mathcal{M}$ be any class of acceptors that generate an* infinite descending chain*, i.e. there exists an infinite chain of acceptors $M_0, M_1, \ldots$ such that $\mathrm{L}(M_0) \supset \mathrm{L}(M_1) \supset \ldots$. Then $\mathcal{M}$ cannot be learned efficiently in the sense of Definition 2.72.*

Note that Yokomori (1995) suggests alternative definitions which seem to be more usable, due to space limitations we will not discuss them here.

There does not seem to be any generally accepted definition of what constitutes a tractable learning function. A serious problem with Angluin's approach is that it is not generally applicable to learning functions for any given class, since both consistency and (especially) conservatism are restrictive. I will therefore apply only the restrictions of consistency and polynomial update-time, since this seems to be the weakest combination of constraints that is restrictive and has an intuitive relation with standard notions of computational complexity.

This definition has at least one drawback: not all learnable classes can be learned by a learning function that is consistent on its class, so even this complexity measure cannot be generally applied.[36] There is also no guarantee that for a class that is learnable by a function consistent on that class characteristic samples (i.e. samples that justify convergence to the right grammar) can be given that are uniformly of a size polynomial in the size of their associated grammar.

See Wiehagen and Zeugmann (1994, 1995); Stein (1998) for discussions of the relation between the consistency constraint and complexity of learning functions.

For more applied work in a Grammatical Inference context, see Oncina and Garcia (1992); de la Higuera (1997); Oncina and de la Higuera (2002); Cascuberta and de la Higuera (2000); de la Higuera et al. (1996).

[36] As noted before, this is a recurrent theme in the work of Wiehagen. Our results in Chapter 5 give concrete evidence that the common approach in the field of machine learning, where only *consistent* algorithms are taken into account, may lead one to overlook feasible solutions to language learning problems.

Chapter 3

Classical Categorial Grammar

In this chapter the basic concepts of classical categorial grammar are introduced. Also, definitions of standardized forms of classical categorial grammar are presented. These have been mostly taken from Kanazawa (1998).

3.1 Basic Definitions

In classical categorial grammar each symbol in the alphabet Σ gets assigned a finite number of *types*. Types are constructed from *primitive type* by the operators $\backslash$ and $/$. We let Pr denote the set of primitive types.

Definition 3.1 *The set of types* Tp is the smallest set satisfying the following conditions:

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

Definition 3.2 *Type A is a* subtype *of B if and only if*

1. *$A = B$, or*
2. *$B = B_1\backslash B_2$ and A is a subtype of B_1 or B_2, or*
3. *$B = B_2/B_1$ and A is a subtype of B_1 or B_2.*

Definition 3.3 $\mathrm{Pr} = \{t\} \bigcup \mathrm{Var}$.

One member of Pr, the constant t, is said to be the *distinguished type*. The other members of Pr are called *variables*, this set is denoted by Var.[1]

Definition 3.4 Grammars *are finite relations on* $\Sigma \times \mathrm{Tp}$.

For a symbol $s \in \Sigma$ and a type $T \in \mathrm{Tp}$, if a grammar G assigns T to s we write $G : s \mapsto T$.

Definition 3.5 ***Domain and range***
The domain *and* range *of a categorial type are defined as*

$$\begin{aligned} \mathrm{dom}(A/B) = \mathrm{dom}(B\backslash A) = B; \\ \mathrm{ran}(A/B) = \mathrm{ran}(B\backslash A) = A. \end{aligned}$$

Definition 3.6 ***Domain subtypes and range subtypes***
A domain subtype *is a subtype that is in domain position, i.e. for the type* $((A/B)/C)$ *the domain subtypes are* B *and* C.
For the type $(C\backslash(B\backslash A))$ *the domain subtypes are* C *and* B.

A range subtype *is a subtype that is in range position, i.e. for the type* $((A/B)/C)$ *the range subtypes are* (A/B) *and* A.
For the type $(C\backslash(B\backslash A))$ *the range subtypes are* $(B\backslash A)$ *and* A.

Domain and range are sometimes called *argument* and *head* respectively.

Convention 3.7 *We write* $\cdot\backslash\cdot$ *and* $\cdot/\cdot$ *for types that have left application and right application, respectively, as main functor and some unspecified domain subtype and range subtype.*

Definition 3.8 *Note that any type* A *with head* p *can be written uniquely in the following form:*

$$(\ldots((p|A_1)|A_2)|\ldots)|A_n$$

where $A|B$ *stands for either* A/B *or* $B\backslash A$, *and* $p \in \mathrm{Pr}$.

This notation can be useful in situations where direction of application is not relevant.

Definition 3.9 [2] *The* degree *of a type is defined as*

$$\begin{aligned} \mathrm{degree}(A) &= 0, \textit{if } A \in Pr, \\ \mathrm{degree}(A/B) &= 1 + \mathrm{degree}(A) + \mathrm{degree}(B), \\ \mathrm{degree}(B\backslash A) &= 1 + \mathrm{degree}(A) + \mathrm{degree}(B), \\ \mathrm{degree}(B \bullet A) &= 1 + \mathrm{degree}(A) + \mathrm{degree}(B). \end{aligned}$$

[1] This notation is somewhat unusual. In the literature on categorial grammar t is commonly written as s. The other members of Pr are normally notated as constants. Here we will use Kanazawa's notation, which is useful in situations where substitutions over types are considered.

[2] Note that the $\bullet$-operator is not commonly used in CCG, however we include it here and in the definition of order since it *is* used in the Lambek calculus, to be defined in Chapter 8.

In other words, the degree of a type can be determined by counting the number of operators it contains.

Another useful notion is that of *order*:

Definition 3.10 *The* order *of a type is defined as*

$$\begin{array}{lcl} \text{order}(A) & = & 0, \textit{if } A \in Pr, \\ \text{order}(A/B) & = & \max(\text{order}(A), \text{order}(B)+1), \\ \text{order}(B\backslash A) & = & \max(\text{order}(A), \text{order}(B)+1), \\ \text{order}(A \bullet B) & = & \max(\text{order}(A), \text{order}(B)). \end{array}$$

Derivations in classical categorial grammar are composed of just *forward application*

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

and *backward application*:

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

Definition 3.11 *The relation* $\Rightarrow \subseteq \text{Tp}^+ \times \text{Tp}$ *is the smallest relation satisfying the following conditions:*

- *For all* $A \in \text{Tp}, A \Rightarrow A$.
- *For all* $\Gamma, \Delta \in \text{Tp}^+$ *and for all* $A, B \in \text{Tp}$,
  - *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* $\Gamma, \Delta \Rightarrow B$.

In other words, a sequence of types $A_1, \ldots, A_i$ is said to *derive* B, notated as

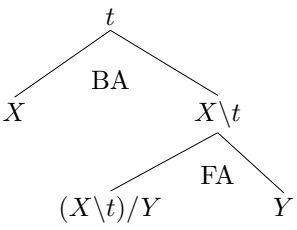
$$A_1, \ldots, A_i \Rightarrow B,$$

if (and only if) there exists a sequence of forward and backward applications such that, if it is applied to this sequence, the type B is obtained.

Definition 3.12 *A* derivation $\mathcal{D}$ *of* B *from* $A_1, \ldots, A_i$ *is a labeled binary branching tree that encodes a proof of* $A_1, \ldots, A_i \Rightarrow B$. *Each node is labeled by a type, and every internal node is also labeled with either* FA *or* BA, *denoting forward or backward application, respectively.*

The root node is labeled by B, *the leaf nodes are labeled by* $A_1, \ldots, A_i$ *in that order.*

Example 3.13 *A derivation of* t *from* $X, (X\backslash t)/Y, Y$:



Definition 3.14 *The node in a derivation $\mathcal{D}$ labeled by $A\backslash B$ or B/A is said to be the* functor*, the node labeled by A is called the* argument *of that application step.*

The ultimate functor *of a derivation is the leaf node arrived at by tracing the functor daughters starting at the root node.*

Definition 3.15 *A* functor-argument structure *over Σ is a binary branching tree that has all its internal nodes labeled by* BA *or* FA *and has all its leaves labeled by symbols in Σ.*

A functor-argument structure corresponding to *a derivation $\mathcal{D}$ in G is obtained by stripping a derivation $\mathcal{D}$ of its type-labels and replacing every type labeling of a leaf by an element of Σ that is assigned that type by G.*

A set of functor-argument structures is called a structure language*. The structure language generated by grammar G is denoted $\mathrm{FL}(G)$, it contains all and only the functor-argument structures corresponding to derivations in G.*

The yield *of a functor-argument structure T is denoted $\mathrm{yield}(T)$.*

The relation between string language and structure language is defined by $\mathrm{L}(G) = \{\mathrm{yield}(T) \mid T \in \mathrm{FL}(G)\}$.

The functions L and FL are called *naming functions*. Note that from a recursion-theoretic point of view, grammars are just indices for languages.

Definition 3.16 *A string is a* sentence *in $\mathrm{L}(G)$ if it is the yield of some functor-argument structure for which a corresponding derivation in G exists that derives t.*

3.2 Decidable and Undecidable Questions about Classical Categorial Grammars

Theorem 3.17 ***(Gaifman)*** *For any context-free language $L \subseteq \Sigma^*$, a categorial grammar G such that $\mathrm{L}(G) = L$ exists if and only if L is an ϵ-free language.*

Proposition 3.18 *The universal membership problem, '$s \in \mathrm{L}(G)$?', is decidable for classical categorial grammars.*

By using one of the numerous parsing algorithms for context-free grammars, this question can be answered in polynomial time.

The following follows directly from a well-known result for context-free languages:

Proposition 3.19 *The questions '$\mathrm{L}(G_1) = \mathrm{L}(G_2)$?' and '$\mathrm{L}(G_1) \subseteq \mathrm{L}(G_2)$?' are, in general, undecidable.*

Proposition 3.20 *Since $\mathrm{FL}(G_1) \subseteq \mathrm{FL}(G_2)$ if and only if $\mathrm{FL}(G_1) \bigcup \mathrm{FL}(G_2) = \mathrm{FL}(G_2)$, and from G_1 and G_2 one can effectively construct G_3 such that $\mathrm{FL}(G_3) = \mathrm{FL}(G_1) \bigcup \mathrm{FL}(G_2)$, the question '$\mathrm{FL}(G_1) \subseteq \mathrm{FL}(G_2)$?' is decidable.*

In Buszkowski (1987b) an algorithm for deciding 'FL(G_1) = FL(G_2)?', based on finding a mapping between two algebras is sketched which runs in at most exponential time. Since the classes of structure languages under discussion are collections of regular tree languages, this problem is closely related to the inclusion problem for regular tree languages. In Seidl (1989, 1990) it was shown that this problem is DEXPTIME-complete with respect to logspace reductions, and still even PSPACE-complete with respect to logspace reductions if the grammars in question only accept finite (structure) languages (also see Asveld and Nijholt (2000) for subclasses of the context-free languages for which inclusion of the corresponding structure languages is decidable, and Greibach and Freidman (1980) for subclasses of context-free tree languages for which inclusion is decidable).

The fact that 'L(G_1) $\subseteq$ L(G_2)?' is undecidable, in contrast to 'FL(G_1) $\subseteq$ FL(G_2)?', is another reason to focus on learning from structures rather than strings.

3.3 Substitutions and Standardizations

3.3.1 Substitutions

Types as defined in Definition 3.1 can be treated as terms where $\backslash$ and / are function symbols. The standard notion of substitution of a term for a variable applies to types.

Definition 3.21 *Let σ be a substitution defined as*

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

for all $A, B \in Tp$.

Substitution over types can be extended to substitution over grammars in a natural way:

Definition 3.22 *Let σ be a substitution. Then $\sigma[G]$ denotes the grammar obtained by applying σ to the type assignments of G:*

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

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

A substitution is called a *(variable) renaming* if and only if it is a one-to-one function from Var to Var. If σ is a variable renaming, then the terms T and $\sigma[T]$ are called *alphabetic variants*. We can of course apply this notion to grammars.

It should be clear that grammars that are alphabetic variants have the same structure and generate the same structure languages, hence to all intents and purposes they are equivalent. The following convention is therefore straightforward:

Convention 3.23 *Grammars that are alphabetic variants are treated as identical.*

Definition 3.24 *Let $<$ be a partial order over categorial types such that $T_1 < T_2$ if there exists a substitution σ such that $\sigma[T_1] = T_2$, $\sigma[T_2] \neq T_1$, and σ is not a renaming.*

This definition implies that T_1 is more general than T_2. Note that for all variables A, $A < t$.

Definition 3.25 *$G_1 \subseteq G_2$ is defined to mean that G_2 contains all of the type assignments of G_1, and possibly more. Obviously, $G_1 \subseteq G_2$ implies* $\mathrm{FL}(G_1) \subseteq \mathrm{FL}(G_2)$.

Proposition 3.26 *If $\sigma[G_1] \subseteq G_2$, then* $\mathrm{FL}(G_1) \subseteq \mathrm{FL}(G_2)$.

Corollary 3.27 *If $\sigma[G_1] \subseteq G_2$, then* $\mathrm{L}(G_1) \subseteq \mathrm{L}(G_2)$.

3.3.2 Grammars in Reduced Form and Grammars Without Useless Types

Definition 3.28 *A substitution σ is said to be* faithful *to a grammar G if the following holds:*
For all $c \in \mathrm{dom}(G)$, if $G_1\colon c \mapsto A, G_1\colon c \mapsto B$, and $A \neq B$, then $\sigma(A) \neq \sigma(B)$.

In other words, a substitution faithful to G does not unify types that are distinct.

Definition 3.29 *Let $\sqsubseteq$ be a binary relation on grammars such that $G_1 \sqsubseteq G_2$ if and only if there exists a substitution σ with the following properties:*

- *σ is faithful to G_1.*
- *$\sigma[G_1] \subseteq G_2$.*

The relation $\sqsubseteq$ is a partial order on grammars.
Let $G_1 \sqsubset G_2$ denote $G_1 \sqsubseteq G_2$ and $G_1 \neq G_2$.

Definition 3.30 *A grammar G is said to be in* reduced form *if there is no G' such that $G' \sqsubset G$ and $\mathrm{FL}(G') = \mathrm{FL}(G)$. This is a decidable property.*

Definition 3.31 *A type $A \in \mathrm{Tp}(G)$ is useless if no derivation in G that derives t contains a node labeled with type A.*

Proposition 3.32 *If a grammar G is in reduced form, then G has no useless type.*

Definition 3.33 *A grammar G is called* redundant *if there is a G' such that $G' \subset G$ and $\mathrm{FL}(G') = \mathrm{FL}(G)$.*

No grammar in reduced form is redundant. However, a grammar *can* be redundant without having a useless type.

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 $\varphi^{\flat}_{\mathrm{VG}_k}$ and $\varphi^{\flat}_{\mathrm{LVG}}$ 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 $\langle \mathrm{CatG}, \Sigma^{+}, \mathrm{L} \rangle$ and $\langle \mathrm{CatG}, \Sigma^{\mathrm{F}}, \mathrm{FL} \rangle$. 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 Σ. Using the latter system, the learning function receives as input sequences of functor-argument structures over Σ.

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 Σ) 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* Σ *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* Σ. *The class* $\{\text{FL}(G) \mid G \in \mathcal{G}_{\text{rigid}}\}$ *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.[1]

Algorithm RG

- **input:** a finite set D of functor-argument structures.
- **output:** a rigid grammar G such that $D \subseteq \text{FL}(G)$, 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 $D = \text{FL}(G)$.

$$\mathsf{c} \mapsto X \rightsquigarrow \left\{ \begin{array}{ll} \mathsf{c} & \mapsto X \end{array} \right\}$$

$$\text{ba}(\mathsf{c}_1, \mathsf{c}_2) \mapsto X \rightsquigarrow \left\{ \begin{array}{ll} \mathsf{c}_1 & \mapsto Y \\ \mathsf{c}_2 & \mapsto Y \backslash X \end{array} \right\}$$

$$\text{fa}(\mathsf{c}_2, \mathsf{c}_1) \mapsto X \rightsquigarrow \left\{ \begin{array}{l} c_1 \mapsto Y \\ \mathsf{c}_2 \mapsto X/Y \end{array} \right\}$$

[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 $\rightsquigarrow$ is taken to mean 'yields, after application of the function GF...'.

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

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

Lemma 4.3 $\mathrm{FL}(\mathrm{GF}(D)) = D$.

Proposition 4.4 $\mathrm{RG}(D)$, *if it exists, is in reduced form.*

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

Theorem 4.5 *The class* $\mathcal{FL}_{\mathrm{rigid}}$ *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* Σ *without useless types such that* $G_0, \sqsubset \ldots \sqsubset G_n$. *Then* $n \leq |\Sigma|$.

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

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

Theorem 4.8 φ_{RG} *learns* $\mathcal{G}_{\mathrm{rigid}}$ *from structures.*

Proposition 4.9 φ_{RG} *has the following desirable properties:*

- φ_{RG} *learns* $\mathcal{G}_{\mathrm{rigid}}$ *prudently.*
- φ_{RG} *is responsive and consistent on* $\mathcal{G}_{\mathrm{rigid}}$.
- φ_{RG} *is set-driven.*
- φ_{RG} *is conservative.*
- φ_{RG} *is monotone increasing.*
- φ_{RG} *is incremental.*
- φ_{RG} *runs in linear time.*[4]

[3] $X \simeq Y$ 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 occurences in the input) it runs in time linear in $|D|$, the size of the input.

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, φ_{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* Σ *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* Σ*. The class* $\{\mathrm{FL}(G) \mid G \in \mathcal{G}_{k\text{-valued}}\}$ *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* σ *is called a* k-partial unifier *of* $\mathcal{A}$ *if and only if for each* $\mathbf{A} \in \mathcal{A}, |\{\sigma(A) \mid A \in \mathbf{A}\}| \leq k$.

Definition 4.15 *Let* $\mathcal{A} = \{\mathbf{A}_1, \ldots, \mathbf{A}_n\}$ *be a family of sets of types.*

- *If* $\mathcal{B}_i = \{\mathbf{B}_{i,1}, \ldots, \mathbf{B}_{i,l_i}\}$ *is a partition of* $\mathbf{A}_i$ *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* σ *be a substitution. The equivalence relation*

$$\sigma(A) = \sigma(B)$$

  on types determines a partition $\mathcal{B}_i$ *of* $\mathbf{A}_i$ *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 σ.

Definition 4.16 *If* $\mathcal{A}$ *is a finite family of finite sets of types, let*

$$\mathrm{PU}_k(\mathcal{A}) = \{\mathrm{mgu}(\mathcal{B}) \mid \mathcal{B} \text{ is a } k\text{-partition of } \mathcal{A}\}.$$

The function PU_k is computable in exponential time.

Proposition 4.17 *Let $\mathcal{A}$ be a finite family of finite sets of types. Then, for every k-partial unifier σ of $\mathcal{A}$, there is some k-partial unifier σ_0 in $\mathrm{PU}_k(\mathcal{A})$ such that σ_0 is more general than σ.*

Algorithm VG_k.

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

Step 1. Algorithm RG: construct $\mathrm{GF}(D)$.

Step 2. Let $\mathcal{A} = \{\{A \mid \mathrm{GF}(D)\text{: } c \mapsto A\} \mid c \in \mathrm{dom}(\mathrm{GF}(D))\}$ and compute $\mathrm{PU}_k(\mathcal{A})$.

Step 3. Let $\mathrm{VG}_k(D) = \{\sigma[\mathrm{GF}(D)] \mid \sigma \in \mathrm{PU}_k(\mathcal{A})\}$. This is the output of the algorithm.

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

1. *$D \subseteq \mathrm{FL}(G)$.*
2. *There exists a grammar $G' \in \mathrm{VG}_k(D)$ such that $G' \sqsubseteq G$.*

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

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 $G \in \mathrm{VG}_k(D)$ and $T \in \mathrm{FL}(G)$, then $G \in \mathrm{VG}_k(D \cup \{T\})$.*

We could call this 'conservativity of grammar class'.

Since it is possible that $|\mathrm{VG}_k(D)| > 1$, we need a way to select a grammar from such a set. In fact, $\mathcal{G}_{\mathrm{rigid}}$ is unique in that $|\mathrm{RG}(D)| = 1$ 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 $\{L \in \mathrm{FL}_{k\text{-valued}} \mid D \subseteq L\}$ is included in $\{\mathrm{FL}(G) \mid G \in \mathrm{VG}_k(D)\}$.*

This, and the decidability of the question '$\mathrm{FL}(G_1) \subseteq \mathrm{FL}(G_2)$'?, makes it possible to find a grammar for a minimal element of $\{L \in \mathrm{FL}_{k\text{-valued}} \mid D \in L\}$ given any D as input.

4.2.1 Learning Functions Based on VG_k

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

Proposition 4.23 *For any finite set $D \subset \Sigma^{\mathrm{F}}$, if $\mu_{\mathrm{FL}}(\mathrm{VG}_k(D))$ is defined, then $\mathrm{FL}(\mu_{\mathrm{FL}}(\mathrm{VG}_k(D)))$ is a minimal element of $\{L \in \mathcal{FL}_{k\text{-valued}} \mid D \subseteq L\}$.*

Definition 4.24 *Let φ_{VG_k} be the learning function for $\langle \mathrm{CatG}, \Sigma^{\mathrm{F}}, \mathrm{FL}\rangle$ defined as follows:*

$$\varphi_{\mathrm{VG}_k}(\langle T_0\rangle) = \mu_{\mathrm{FL}}(\mathrm{VG}_k(\{T_0\})),$$
$$\varphi_{\mathrm{VG}_k}(\langle T_0, \ldots, T_{i+1}\rangle) = \begin{cases} \varphi_{\mathrm{VG}_k}(\langle T_0, \ldots, T_i\rangle) & \text{if } T_{i+1} \in \mathrm{FL}(\varphi_{\mathrm{VG}_k}(\langle T_0, \ldots, T_i\rangle)), \\ \mu_{\mathrm{FL}}(\mathrm{VG}_k(\{T_0, \ldots, T_{i+1}\})) & \text{otherwise.} \end{cases}$$

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 μ_{FL} and some class of grammars may not be *inherently* conservative.

Proposition 4.25

1. *φ_{VG_k} is responsive and consistent on $\mathcal{G}_{k\text{-valued}}$.*
2. *φ_{VG_k} is conservative.*
3. *φ_{VG_k} learns $\mathcal{G}_{k\text{-valued}}$ prudently.*

Theorem 4.26 *φ_{VG_k} learns $\mathcal{G}_{k\text{-valued}}$ from structures.*

The function φ_{VG_k} is not designed to be set-driven or even to learn order-independently. Kanazawa defines a set-driven learning function $\varphi^{\flat}_{\mathrm{VG}_k}$:

Definition 4.27

$$\varphi^{\flat}_{\mathrm{VG}_k}(\langle T_0, \ldots, T_i\rangle) = \mu_{\mathrm{FL}}(\mathrm{VG}_k(\{T_0, \ldots, T_i\})),$$

where μ_{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.

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

Here, $\prec$ is defined by:[6]

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

1. *size$(G_1) <$ size(G_2).*
2. *size$(G_1) =$ size(G_2) and $|\mathrm{Var}(G_1)| > |\mathrm{Var}(G_2)|$.*
3. *size$(G_1) =$ size(G_2) and $|\mathrm{Var}(G_1)| = |\mathrm{Var}(G_2)|$, 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,* size(G)*, as follows:*

$$\mathrm{size}(G) = \sum_{c \in \Sigma} \sum_{G:\, c \mapsto A} |A|$$

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

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

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

Unfortunately the reverse does not hold. If it would, $\sqsubset$ and $\prec$ would obviously be equivalent, greatly simplifying matters. We can, however, use $\prec$ instead of $\sqsubset$. 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 $\neg(G_2 \sqsubset G_1)$*

`Proof:` By lemma 4.32, $\neg(G_2 \prec G_1)$ implies $\neg(G_2 \sqsubset G_1)$. Since $\prec$ is a well-order, $G_1 \prec G_2$ implies $\neg(G_2 \prec G_1)$. □

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

There exists a learning strategy called *simplicity* (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 $\langle T_i \rangle_{i \in \mathbb{N}}$ be an infinite sequence enumerating some $L \in \mathcal{FL}_{k\text{-valued}}$. Then $\varphi^{\flat}_{\mathrm{VG}_k}$ converges on $\langle T_i \rangle_{i \in \mathbb{N}}$ to the first element of*

$$\mathcal{G}_L = \{G \in \mathcal{G}_{k\text{-valued}} \mid \mathrm{FL}(G) = L\}$$

under the ordering $\prec$.

4.3 $\varphi^{\flat}_{\mathrm{VG}_k}$ is not conservative

While $\varphi^{\flat}_{\mathrm{VG}_k}$ is set-driven, Kanazawa left it an open question whether it is conservative.

I have constructed a proof of non-conservativity of $\varphi^{\flat}_{\mathrm{VG}_k}$ that was inspired by a footnote on page 102 of Kanazawa (1998):

> Although I have not had time to prove that $\varphi^{\flat}_{\mathrm{VG}_k}$ is indeed nonconservative, it is conceivable that the following sort of situation can obtain: $G_0 = \mu_{\mathrm{FL}}(\mathrm{VG}_k(D))$. $G_1 \prec G_0$, $G_1 \in \mathrm{VG}_k(D)$, there is no $G'_1 \in \mathrm{VG}_k(D)$ such that $G'_1 \sqsubset G_1$, but $\mathrm{FL}(G_1)$ is not minimal in $\mathrm{FL}(\mathrm{VG}_k(D))$. $G_0, G_1 \in \mathrm{VG}_k(D \cup \{T\})$, and $\mathrm{FL}(G_1)$ is minimal in $\mathrm{FL}(\mathrm{VG}_k(D \cup \{T\}))$.

To summarize, the following conditions are sufficient conditions for $\varphi^{\flat}_{\mathrm{VG}_k}$ to be non-conservative:

1. $G_0, G_1 \in \mathrm{VG}_k(D)$, $\mathrm{FL}(G_0)$ is minimal, but $\mathrm{FL}(G_1)$ is not minimal in $\mathrm{FL}(\mathrm{VG}_k(D))$, $G_1 \prec G_0$, $\varphi^{\flat}_{\mathrm{VG}_k}(D) = G_0$.
2. $G_0, G_1 \in \mathrm{VG}_k(D \cup \{T\})$, $\mathrm{FL}(G_1)$ is minimal in $\mathrm{FL}(\mathrm{VG}_k(D \cup \{T\}))$, $\varphi^{\flat}_{\mathrm{VG}_k}(D \cup \{T\}) = G_1$.

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

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

Since $G_1 \prec G_0$ and $\mu_{\mathrm{FL}}(\mathrm{VG}_k(D)) = G_0$, G_1 is not minimal in $\mathrm{VG}_k(D)$, so there is a grammar $G_2 \mathrm{VG}_k(D)$ such that $\mathrm{FL}(G_2) \subset \mathrm{FL}(G_1)$ and $G_0 \prec G_2$.

Moreover, G_2 cannot be in $\mathrm{VG}_k(D \cup \{T\})$. If it would, μ_{FL} would choose G_2 over G_1, so condition 2 would be impossible. So we have $\neg(\mathrm{FL}(G_2) \subset \mathrm{FL}(G_0))$, and $G_0 \prec G_2$, so $G_1 \prec G_0 \prec G_2$.

By Proposition 4.20, since $G_2 \in \mathrm{VG}_k(D)$ and $G_2 \notin \mathrm{VG}_k(D \cup \{T\})$, $T \notin \mathrm{FL}(G_2)$.

Since for any $G \in \mathrm{VG}_k(D)$, $D \subseteq \mathrm{FL}(G)$, $\{T\} \subset \mathrm{FL}(G_0)$, $\{T\} \subseteq \mathrm{FL}(G_1)$.

We are now equipped to prove the following:[7]

[7] 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 verify 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 $\varphi^{\flat}_{\mathrm{VG}_k}$ is non-conservative.*

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

```
ba(fa(a,fa(b,fa(x,x))),g)                  g
ba(fa(fa(y,y),fa(fa(y,y),fa(x,x))),g)      ba(a,e)
ba(b,e)                                    e
ba(ba(fa(z,z),a),j)                        j
ba(ba(fa(z,z),fa(w,w)),j)                  ba(fa(y,y),c)
ba(fa(w,w),c)                              ba(d,e)
ba(ba(fa(z,z),d),j)                        ba(fa(a,fa(f,fa(x,x))),g)
ba(f,e)
```

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`.

$$G_1: \quad \begin{array}{lcl} \texttt{a} & \mapsto & B/B, D\backslash E \\ \texttt{b} & \mapsto & B/B \\ \texttt{c} & \mapsto & (B/B)\backslash t, (D\backslash E)\backslash t \\ \texttt{d} & \mapsto & B/B, D\backslash E \\ \texttt{e} & \mapsto & (B/B)\backslash t, t \\ \texttt{f} & \mapsto & B/B \\ \texttt{g} & \mapsto & B\backslash t, t \\ \texttt{j} & \mapsto & E\backslash t, t \\ \texttt{w} & \mapsto & (D\backslash E)/W, W \\ \texttt{x} & \mapsto & B/X, X \\ \texttt{y} & \mapsto & (B/B)/Y, Y \\ \texttt{z} & \mapsto & D/Z, Z \end{array}$$

$$G_2: \quad \begin{array}{lcl} \texttt{a} & \mapsto & B/C, D\backslash E \\ \texttt{b} & \mapsto & B/C, C/B \\ \texttt{c} & \mapsto & (B/B)\backslash t, (D\backslash E)\backslash t \\ \texttt{d} & \mapsto & B/C, D\backslash E \\ \texttt{e} & \mapsto & (B/C)\backslash t, t \\ \texttt{f} & \mapsto & B/C, C/B \\ \texttt{g} & \mapsto & B\backslash t, t \\ \texttt{j} & \mapsto & E\backslash t, t \\ \texttt{w} & \mapsto & (D\backslash E)/W, W \\ \texttt{x} & \mapsto & B/X, X \\ \texttt{y} & \mapsto & (B/B)/Y, Y \\ \texttt{z} & \mapsto & D/Z, Z \end{array}$$

$$G_3: \quad \begin{array}{lcl} \texttt{a} & \mapsto & B/C, D\backslash E \\ \texttt{b} & \mapsto & C/B, D\backslash E \\ \texttt{c} & \mapsto & (B/B)\backslash t, (D\backslash E)\backslash t \\ \texttt{d} & \mapsto & D\backslash E \\ \texttt{e} & \mapsto & (D\backslash E)\backslash t, t \\ \texttt{f} & \mapsto & C/B, D\backslash E \\ \texttt{g} & \mapsto & B\backslash t, t \\ \texttt{j} & \mapsto & E\backslash t, t \\ \texttt{w} & \mapsto & (D\backslash E)/W, W \\ \texttt{x} & \mapsto & B/X, X \\ \texttt{y} & \mapsto & (B/B)/Y, Y \\ \texttt{z} & \mapsto & D/Z, Z \end{array}$$

Note that G_1 is the result of unifying C with B in G_2, and $\mathrm{FL}(G_1)$ properly includes $\mathrm{FL}(G_2)$. $\mathrm{FL}(G_2)$ and $\mathrm{FL}(G_3)$ are incomparable and $\mathrm{size}(G_2) > \mathrm{size}(G_3)$, so the set-driven learning function based on $\prec$ picks G_3 for this sample.

Now consider adding

```
ba(b,c)
```

to the above sample and see what grammars are output:

G'_1 :

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

G'_2 :

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

G'_3 :

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

G'_4 :

a	$\mapsto$	$B/C, D\backslash E$
b	$\mapsto$	$C/B, D\backslash E$
c	$\mapsto$	$(B/B)\backslash t, (D\backslash E)\backslash t$
d	$\mapsto$	$D\backslash E$
e	$\mapsto$	$(D\backslash E)\backslash t, t$
f	$\mapsto$	$C/B, D\backslash E$
g	$\mapsto$	$B\backslash t, t$
j	$\mapsto$	$E\backslash t, t$
w	$\mapsto$	$(D\backslash E)/W, W$
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: $\mathtt{b} \mapsto D\backslash E$. So $\mathrm{FL}(G'_2)$ properly includes $\mathrm{FL}(G'_4)$ and $\mathrm{FL}(G'_3)$ properly includes $\mathrm{FL}(G'_1)$. $\mathrm{FL}(G'_1)$ and $\mathrm{FL}(G'_4)$ are incomparable. But $\mathrm{size}(G'_1) < \mathrm{size}(G'_4)$, so G'_1, not G'_4 ($= G_3$), is the grammar picked by the set-driven learning function for this expanded sample. □

4.4 Least-Valued Grammars

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

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

Definition 4.37 *If $G \in \mathrm{LVG}(D)$, 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 $\mathrm{FL}(G)$.*

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

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 $D \subseteq \mathrm{FL}(G)$ for the least k such that D is a subset of some $L \in \mathcal{FL}_{k\text{-valued}}$.*

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

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

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

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

Let φ_{LVG} be defined as φ_{VG_k} in Definition 4.24, with VG_k replaced by LVG.

Proposition 4.42 *The learning function φ_{LVG} learns $\mathcal{G}_{\text{least-valued}}$ from structures.*

- *φ_{LVG} is responsive and consistent on $\mathcal{G}_{\text{least-valued}}$.*
- *φ_{LVG} is conservative.*

It is possible to define an alternative learning function $\varphi^{\flat}_{\mathrm{LVG}}$ analogous to Definition 4.27 that is set-driven. It converges in a way entirely analogous to $\varphi^{\flat}_{\mathrm{VG}_k}$ as stated in Proposition 4.34.

Proposition 4.43 *The set-driven learning function $\varphi^{\flat}_{\mathrm{LVG}}$ 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 $\cdot/\cdot$ and a type of the form $\cdot\backslash\cdot$ are assigned to the symbol `l`. Clearly, these two types are not unifiable, so the least value for k for which $\mathrm{VG}_k(D)$ is defined is at least 2. In the proof, for all the grammars that have to be considered by μ (G'_1, G'_2 and G'_3), $k = 2$. Thus, the proof works for $\varphi^{\flat}_{\mathrm{LVG}}$ as well. □

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 $\mathcal{G}_{\mathrm{optimal}}$ to denote the class of optimal grammars over Σ. The class $\{\mathrm{FL}(G) \mid G \in \mathcal{G}_{\mathrm{optimal}}\}$ is denoted $\mathcal{FL}_{\mathrm{optimal}}$.*

Definition 4.45 *Let $\mathcal{A} = \{\mathbf{A}_1, \ldots, \mathbf{A}_n\}$ be a family of sets of types. A substitution σ is called an* optimal unifier *of $\mathcal{A}$ if the following holds:*

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

An optimal unifier of $\mathcal{A}$ unifies is a substitution that unifies $\mathcal{A}$ 'as much as possible'. Note that this means that no grammar G, $G \in \mathcal{G}_{\mathrm{optimal}}$ is redundant.

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

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

1. *$\mathcal{B}$ has a unifier*
2. *No partition $\mathcal{C}$ of $\mathcal{A}$ strictly coarser than $\mathcal{B}$ has a unifier.*

Proposition 4.48 *Let $\mathcal{A}$ be a family of sets of types. A substitution σ is an optimal unifier of $\mathcal{A}$ if and only if σ is a most general unifier of some optimal partition of $\mathcal{A}$.*

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

$$\mathrm{OU}(\mathcal{A}) = \{\mathrm{mgu}(\mathcal{B}) \mid \mathcal{B} \textit{ is an optimal partition of } \mathcal{A}\}.$$

Obviously, if $\mathcal{A}$ has an unifier, $\mathrm{OU}(\mathcal{A}) = \{\mathrm{mgu}(\mathcal{A})\}$. 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 $D \subseteq \mathrm{FL}(G)$.

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

Let $\mathcal{A} = \{\{A \mid \mathrm{GF}(D)\colon c \mapsto A\} \mid c \in \mathrm{dom}(\mathrm{GF}(D))\}$, and compute $\mathrm{OU}(\mathcal{A})$. Let $\mathrm{OG}(D) = \{\sigma[\mathrm{GF}(D)] \mid \sigma \in \mathrm{OU}(\mathcal{A})\}$.

Proposition 4.50 $\{G \in \mathcal{G}_{\mathrm{optimal}} \mid G \textit{ is in reduced form}\} \subset \bigcup \mathrm{range}(\mathrm{OG})$.

Proposition 4.51 $\mathcal{FL}_{\mathrm{OG}} \subset \mathcal{FL}_{\mathrm{optimal}}$.

Theorem 4.52 $\mathcal{FL}_{\mathrm{optimal}}$ *has a limit point.*

Corollary 4.53 $\mathcal{FL}_{\mathrm{OG}}$ *has a limit point.*

Corollary 4.54 *Neither* $\mathcal{G}_{\mathrm{optimal}}$ *nor* $\bigcup \mathrm{range}(\mathrm{OG})$ *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 $L \subseteq \Sigma^{\mathrm{F}}$. A grammar G is said to be* of least cardinality with respect to L *if $L \subseteq \mathrm{FL}(G)$ and there is no grammar G' such that $|G'| < |G|$ and $L \subseteq \mathrm{FL}(G')$.*

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

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

$$\mathrm{LCG}(D) = \{G \in \mathrm{OG}(D) \mid \forall G' \in \mathrm{OG}(D)(|G| \leq |G'|)\}.$$

Lemma 4.58 *if $G \in \mathrm{LCG}(D)$, 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* $\mathrm{FL}(G)$.

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

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

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

Proposition 4.63 $\mathcal{FL}_{\text{least-card}} \subset \mathcal{FL}_{\mathrm{OG}}$.

Let φ_{LCG} be defined as in Definition 4.24, with VG_k replaced by LCG.

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

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

φ_{LCG} can be shown to learn $\mathcal{G}_{\text{least-card}}$ order-independently. A set-driven learning function $\varphi^{\flat}_{\mathrm{LCG}}$ 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 Σ arranged in a fixed order. For each grammar G over Σ, let $v(G)$ be the vector defined as follows:*

$$v(G) = \langle n_1, \ldots, n_l \rangle,$$

where for $1 \leq j \leq l, n_j = |\{A \mid G\textit{:}\ c_j \mapsto A\}|$.

The partial order $\leq$ is defined on vectors in $\mathbb{N}^l$ in the natural way: $\langle n_1, \ldots, n_l \rangle \leq \langle m_1, \ldots, m_l \rangle$ if $n_j \leq m_j$ for all $j (1 \leq j \leq l)$. Also, if $v_1, v_2 \in \mathbb{N}^l$, $v_1 < v_2$ if and only if $v_1 \leq v_2$ and $v_1 \neq v_2$.

Definition 4.66 *If D is a finite subset of Σ^{F}, let*

$$\mathrm{MG}(D) = \{G \in \mathrm{OG}(D) \mid \neg \exists G' \in \mathrm{OG}(D)(v(G') < v(G))\}.$$

Thus, $\mathrm{MG}(D)$ consists of those elements of $\mathrm{OG}(D)$ of which the associated vector is minimal. Clearly, $\mathrm{MG}(D) \neq \emptyset$ for all D.

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

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

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

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

Corollary 4.70 *$G_1 \sqsubseteq G_2$ 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 $\{\mathrm{FL}(G) | v(G) \not> \langle n_1, \ldots, n_l \rangle\}$ does not have finite elasticity.*

Whether or not $\mathcal{G}_{\mathrm{minimal}}$ 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 $\mathcal{G}_{\mathrm{rigid}}$ 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 citetkanazawa94note 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 $\{L \cup M \mid L \in \mathcal{L} \wedge M \in \mathcal{M}\}$ also has finite elasticity).

Theorem 4.74 *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.*

From Definition 3.15, we have $\mathrm{L}(G) = \{\mathrm{yield}(T) \mid T \in \mathrm{FL}(G)\}$. If $L \subseteq \Sigma^{\mathrm{F}}$, we write $\mathrm{yield}[L]$ for $\{\mathrm{yield}(T) \mid T \in L\}$. Then $\mathcal{L}_{k\text{-valued}} = \{\mathrm{yield}[L] \mid L \in \mathcal{FL}_{k\text{-valued}}\}$. The relation $R \subseteq \Sigma^{+} \times \Sigma^{\mathrm{F}}$ defined by $RsT \Leftrightarrow s = \mathrm{yield}(T)$ is finite valued. Since $\mathcal{FL}_{k\text{-valued}}$ has finite elasticity, applying Theorem 4.74 shows that $\mathcal{L}_{k\text{-valued}}$ also has finite elasticity.

Using Proposition 4.73, the following can be shown:

Theorem 4.75 *For each $k \in \mathbb{N}$, $\mathcal{L}_{k\text{-valued}} \subset \mathcal{L}_{k+1\text{-valued}}$.*

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 μ_{L}. The function $\Psi_{k\text{-valued}}$ maps a finite set of strings to a finite set of k-valued grammars. The function μ_{L} takes two arguments, a finite set of grammars and a positive integer, and returns a member of the first argument.

Definition 4.76

$$\Psi_{k\text{-valued}}(\{s_0, \ldots, s_i\}) = \bigcup \{\mathrm{VG}_k(\{T_0, \ldots, T_i\}) \mid s_j = \mathrm{yield}(T_j)\}$$

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 $\{s_0, \ldots, s_i\} \subseteq \mathrm{L}(G)$.*

Given Proposition 4.18, the following lemma is straightforward:

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

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 $\mu_{\mathrm{L},k}$ to define a learning function as in Definition 2.60. The function $\mu_{\mathrm{L},k}$ 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.

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

It is not clear whether such a computable function $\mu_{\mathrm{L},k}$ exists.[9] If conservatism is not required, however, it is possible to define a computable learning function for $\mathcal{G}_{k\text{-valued}}$ using $\Psi_{k\text{-valued}}$. Recall the ordering $\prec$ from Definition 4.29:

Definition 4.80 *Let μ_{L} be a computable function that maps a non-empty finite set $\mathcal{G}$ of grammars and a positive integer n to the first element of the following set, under the ordering $\prec$, i.e.,*

$$\{G \in \mathcal{G} \mid \neg\exists G' \in \mathcal{G}(\mathrm{L}(G') \cap (\Sigma \cup \{\epsilon\})^n \subset \mathrm{L}(G) \cap (\Sigma \cup \{\epsilon\})^n)\}.$$

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

$$\mathcal{G}' = \{G \in \mathcal{G} \mid \neg\exists G' \in \mathcal{G}(\mathrm{L}(G') \subset \mathrm{L}(G))\},$$

under the ordering $\prec$.

Definition 4.82 *Define a learning function $\psi_{k\text{-valued}}$ for the grammar system $\langle \mathrm{CatG}, \Sigma^+, \mathrm{L}\rangle$ as follows:*

$$\psi_{k\text{-valued}}(\langle s_0, \ldots, s_i\rangle) = \mu_{\mathrm{L}}(\Psi_{k\text{-valued}}(\{s_0, \ldots, s_i\}), i+1),$$

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

Proposition 4.83 *The learning function $\psi_{k\text{-valued}}$ is responsive and consistent on $\mathcal{G}_{k\text{-valued}}$.*

Theorem 4.84 *The learning function $\psi_{k\text{-valued}}$ learns $\mathcal{G}_{k\text{-valued}}$ 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 $\psi^{\flat}_{k\text{-valued}}$ for the grammar system $\langle \mathrm{CatG}, \Sigma^+, \mathrm{L}\rangle$ as follows:*

$$\psi^{\flat}_{k\text{-valued}}(\langle s_0, \ldots, s_i\rangle)$$
$$= \begin{cases} \text{the first element of} \\ \{G \in \Psi_{k\text{-valued}}(\{s_0, \ldots, s_i\}) \mid \mathrm{L}(G) = \{s_0, \ldots, s_i\}\} & \text{if it exists,} \\ \mu_{\mathrm{L}}(\Psi_{k\text{-valued}}(\{s_0, \ldots, s_i\}), |\{s_0, \ldots, s_i\}|) & \text{otherwise.} \end{cases}$$

Note that reference to the length of $\langle s_0, \ldots, s_i\rangle$ 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* $\psi^{\flat}_{k\text{-valued}}$ *is responsive and consistent on* $\mathcal{G}_{k\text{-valued}}$ *and is set-driven.*

Proposition 4.87 *The learning function* $\psi^{\flat}_{k\text{-valued}}$ *learns* $\mathcal{G}_{k\text{-valued}}$ *from strings order-independently and prudently.*

4.9 Classes that are not Learnable from Strings

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

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

Corollary 4.90 *None of the classes*

$$\mathcal{G}_{\text{least-valued}}, \mathcal{G}_{\text{optimal}}, \mathcal{G}_{\text{least-card}}, \mathcal{G}_{\text{minimal}}$$

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 FL, is investigated:

Definition 4.91 *The classes* $\mathcal{G}^{\mathrm{L}}_{\text{least-valued}}$, $\mathcal{G}^{\mathrm{L}}_{\text{least-card}}$ *and* $\mathcal{G}^{\mathrm{L}}_{\text{minimal}}$ *are defined just like* $\mathcal{G}_{\text{least-valued}}$, $\mathcal{G}_{\text{least-card}}$ *and* $\mathcal{G}_{\text{minimal}}$ *except that reference to* FL *is replaced by reference to* L. *Let* $\mathcal{L}^{\mathrm{L}}_{\text{least-valued}} = \{\mathrm{L}(G) | G \in \mathcal{G}^{\mathrm{L}}_{\text{least-valued}}\}$, $\mathcal{L}^{\mathrm{L}}_{\text{least-card}} = \{\mathrm{L}(G) \mid G \in \mathcal{G}^{\mathrm{L}}_{\text{least-card}}\}$ *and* $\mathcal{L}^{\mathrm{L}}_{\text{minimal}} = \{\mathrm{L}(G) \mid G \in \mathcal{G}^{\mathrm{L}}_{\text{minimal}}\}$.

Corollary 4.92 *The following inclusions hold:*
$\mathcal{G}^{\mathrm{L}}_{\text{least-valued}} \subseteq \mathcal{G}_{\text{least-valued}}$, $\mathcal{G}^{\mathrm{L}}_{\text{least-card}} \subseteq \mathcal{G}_{\text{least-card}}$, *and* $\mathcal{G}^{\mathrm{L}}_{\text{minimal}} \subseteq \mathcal{G}_{\text{minimal}}$.

Corollary 4.93 $\mathcal{G}^{\mathrm{L}}_{\text{least-card}} \subseteq \mathcal{G}^{\mathrm{L}}_{\text{minimal}}$.

Proposition 4.94 *For* $\mathcal{G} = \mathcal{G}^{\mathrm{L}}_{\text{least-valued}} \subseteq \mathcal{G}_{\text{least-valued}}$, $\mathcal{G}^{\mathrm{L}}_{\text{least-card}} \subseteq \mathcal{G}_{\text{least-card}}$, $\mathcal{G}^{\mathrm{L}}_{\text{minimal}} \subseteq \mathcal{G}_{\text{minimal}}$, *the class* $\{G \in \mathcal{G} \mid$ *for no* $G' \sqsubset G, \mathrm{L}(G') = \mathrm{L}(G)\}$ *is r.e.*

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

Proposition 4.95 $\mathcal{L}^{\mathrm{L}}_{\text{minimal}}$ *contains an infinite ascending chain.*

Class	Learnable	Finite Elasticity
$\mathcal{G}_{\text{rigid}}$	yes	yes
$\mathcal{G}_{k\text{-valued}}$	yes	yes
$\mathcal{G}_{\text{optimal}}$	no	no
$\mathcal{G}_{\text{least-valued}}$	yes	no
$\mathcal{G}_{\text{least-card}}$	yes	no
$\mathcal{G}_{\text{minimal}}$	?	no

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

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

As a final remark, note that Seginer (2002a) shows that any subclass of $\mathcal{G}^{\text{L}}_{\text{rigid}}$ 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 occurences 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)).[10]

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 conservative 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).

Class	Learnable	Finite Elasticity
$\mathcal{G}_{\text{rigid}}$	yes	yes
$\mathcal{G}_{k\text{-valued}}$	yes	yes
$\mathcal{G}_{\text{least-valued}}$	no	no
$\mathcal{G}_{\text{optimal}}$	no	no
$\mathcal{G}_{\text{least-card}}$	no	no
$\mathcal{G}_{\text{minimal}}$	no	no
$\mathcal{G}^{\text{L}}_{\text{least-valued}}$	yes	yes
$\mathcal{G}^{\text{L}}_{\text{least-card}}$	yes	yes
$\mathcal{G}^{\text{L}}_{\text{minimal}}$	?	no

Table 4.2: Summary of Kanazawa's results for learning from strings.

Chapter 5

Complexity Issues

In this chapter[1] the complexity of learning the classes defined in Chapter 4 will be analyzed. As has been discussed at length in Section 2.9, at the present time there is no really satisfactory criterion to distinguish classes that are tractably (in a computational sense) learnable. However, it *is* possible to analyze the complexity of generating a hypothesis that is in the class specified and is consistent with the input. We will show that this problem is NP-hard for all the classes discussed so far, except for the rigid grammars.

First a closer look will be taken at the discovery procedures underlying these classes. More specifically, the next section will answer the question 'how many distinct hypotheses do they generate given a sample of a certain size?'.

5.1 An Avalanche of Hypotheses

The learning functions presented in Kanazawa (1998) are defined in terms of functions that produce finite sets of grammars given a sample of structures, the learning functions select one of the grammars that generate a minimal structure language (with respect to inclusion) from this set. Note that this is just a way of defining a function, an algorithm implementing such a function may use a completely different method to arrive at the intended result. However, these definitions suggest one particular implementation; in algorithmic terms they employ a 'generate-and-test' strategy, i.e., the space of possible solutions is traversed exhaustively, and after this a set of candidate solutions is evaluated. As we shall see, some of these sets of grammars grow exponentially in the size of the sample. This shows that it is impossible for any algorithm *based on such a strategy* to run in polynomial time, but obviously this does not prove anything about the existence of tractable algorithms that learn these classes.

[1] All results in this chapter were previously published, see Costa Florêncio (2001a, 2000b, 2001d, 2002a, To appear), all reproduced with permission.

We will prove this for the optimal grammar class, and the example samples will turn out to yield the same, or larger, sets of grammars when some of the other functions previously defined are applied to them.

Example 5.1 *Let G_{10} be the following general form in which A, $(B\backslash t)$ and (t/C) are types assigned to the same word.*

$$G_{10}: \quad \begin{array}{lcl} \mathtt{a} & \mapsto & A, B\backslash t, t/C \\ \mathtt{b} & \mapsto & B, C \\ \mathtt{c} & \mapsto & A\backslash t \end{array}$$

Then A has to be unified with either $(B\backslash t)$ or (t/C) to produce a grammar that is optimal:

$$G_{10\text{optimal}}: \quad \begin{array}{lcl} \mathtt{a} & \mapsto & B\backslash t, t/B \\ \mathtt{b} & \mapsto & B \\ \mathtt{c} & \mapsto & (B\backslash t)\backslash t \end{array}$$

or

$$G'_{10\text{optimal}}: \quad \begin{array}{lcl} \mathtt{a} & \mapsto & B\backslash t, t/B \\ \mathtt{b} & \mapsto & B \\ \mathtt{c} & \mapsto & (t/B)\backslash t \end{array}$$

Let G_{11} be a general form in which there are multiple copies of the A-type.

$$G_{11}: \quad \begin{array}{lcl} \mathtt{a} & \mapsto & A[1]\ldots A[n], B\backslash t, t/C \\ \mathtt{b} & \mapsto & B, C \\ \mathtt{c} & \mapsto & A[1]\backslash t, \ldots, A[n]\backslash t \end{array}$$

Now we have $A[1]\ldots A[n], (B\backslash t)$ and (t/C) assigned to the same word. What was true for A holds for $A[1]$ to $A[n]$ as well. These types can be distributed to unify with either $(B\backslash t)$ or (t/C) in 2^n ways. If we assume we can get all these combinations in distinct grammars that are in reduced form, the cardinality of $\mathrm{OG}(D_{11})$[2] *should be 2^n. In this case however most of these grammars turn out to be identical, in fact there are only three distinct ones:*

$$G_{11\text{optimal}}1: \quad \begin{array}{lcl} \mathtt{a} & \mapsto & B\backslash t, t/B \\ \mathtt{b} & \mapsto & B \\ \mathtt{c} & \mapsto & (B\backslash t)\backslash t \end{array}$$

[2]By convention, we will write D_k for $\mathrm{FL}(\mathrm{G}_k)$, see Lemma 4.3.

or

$$G_{11\text{optimal}}2: \quad \begin{array}{rcl} \texttt{a} & \mapsto & B\backslash t, t/B \\ \texttt{b} & \mapsto & B \\ \texttt{c} & \mapsto & (t/B)\backslash t \end{array}$$

or

$$G_{11\text{optimal}}3: \quad \begin{array}{rcl} \texttt{a} & \mapsto & B\backslash t, t/B \\ \texttt{b} & \mapsto & B \\ \texttt{c} & \mapsto & (t/B)\backslash t, (B\backslash t)\backslash t \end{array}$$

Note that even though these grammars are distinct, they generate the same structure language. Since the function μ_{FL} picks a grammar with a minimal structure language, this means that an algorithm may possibly exist that does not need to generate all of these grammars. As we will see, this is also true for the following grammars in this section.

We will need $A[1] \dots A[n]$ to be under obligation to unify only in the assignment to `a` to make all these grammars distinct. We can do this by assigning types $A[1]\backslash t \dots A[n]\backslash t$ to n pairwise distinct symbols. Let $G_{12} = \text{GF}(D_{12})$:[3]

$$G_{12}: \quad \begin{array}{rcl} \texttt{a} & \mapsto & A[1], \dots A[n], B\backslash t, t/C \\ \texttt{b} & \mapsto & B, C \\ \texttt{c[1]} & \mapsto & A[1]\backslash t \\ \texttt{c[2]} & \mapsto & A[2]\backslash t \\ \dots & & \\ \texttt{c[n]} & \mapsto & A[n]\backslash t \end{array}$$

This general form generates 2^n distinct optimal grammars:

$$G_{12\text{optimal}}1: \quad \begin{array}{rcl} \texttt{a} & \mapsto & B\backslash t, t/B \\ \texttt{b} & \mapsto & B \\ \texttt{c[1]} & \mapsto & (B\backslash t)\backslash t \\ \texttt{c[2]} & \mapsto & (t/B)\backslash t \\ \dots & & \\ \texttt{c[n]} & \mapsto & (t/B)\backslash t \end{array}$$

[3]We do not present D_{12} explicitly, but it can easily be derived from G_{12}.

$$
G_{12\text{optimal}}2: \quad
\begin{array}{rcl}
\texttt{a} & \mapsto & B\backslash t, t/B \\
\texttt{b} & \mapsto & B \\
\texttt{c[1]} & \mapsto & (t/B)\backslash t \\
\texttt{c[2]} & \mapsto & (B\backslash t)\backslash t \\
\ldots & & \\
\texttt{c[n]} & \mapsto & (t/B)\backslash t
\end{array}
$$

$$
G_{12\text{optimal}}3: \quad
\begin{array}{rcl}
\texttt{a} & \mapsto & B\backslash t, t/B \\
\texttt{b} & \mapsto & B \\
\texttt{c[1]} & \mapsto & (B\backslash t)\backslash t \\
\texttt{c[2]} & \mapsto & (B\backslash t)\backslash t \\
\ldots & & \\
\texttt{c[n]} & \mapsto & (t/B)\backslash t
\end{array}
$$

...

$$
G_{12\text{optimal}}2^n: \quad
\begin{array}{rcl}
\texttt{a} & \mapsto & B\backslash t, t/B \\
\texttt{b} & \mapsto & B \\
\texttt{c[1]} & \mapsto & (B\backslash t)\backslash t \\
\texttt{c[2]} & \mapsto & (B\backslash t)\backslash t \\
\ldots & & \\
\texttt{c[n]} & \mapsto & (B\backslash t)\backslash t
\end{array}
$$

Note that, like the grammars derived from G_{11}, they all generate the same structure language. Simply adding to G_{12} the assignments $\texttt{b} \mapsto D$ and $\texttt{d} \mapsto D\backslash t$ yields optimal grammars that have distinct structure languages.

The size of the sample needed to create such a general form is $2 * n + 4$. So n can be expressed in terms of the sample size as $n = (size - 4)\backslash 2$, thus a non-sharp worst-case upper bound for the cardinality of the set output by OG(D) is O($2^{(size)}$). Obviously, this bound is exponential.

The complexity is actually even worse than this. As the reader may already have noticed, we can generalize G_{12} so that the number of non-unifiable types assigned to `a` becomes variable, see Costa Florêncio (2001a).

Our example proves the following proposition:

Proposition 5.2 OG *has exponential output complexity.*

Proposition 5.3 LCG *has exponential output complexity.*

`Proof:` As the reader may verify, all the optimal grammars in OG(D_{12}) are of least cardinality. By definition (Definition 4.57), LCG(D) $\subseteq$ OG(D). Thus, LCG(D_{13}) = OG(D_{13}). □

Proposition 5.4 MG *has exponential output complexity.*

`Proof:` By Proposition 4.71, if a grammar $G \in \mathrm{LCG}(D)$, then $G \in \mathrm{MG}(D)$. This implies that $\mathrm{MG}(D_{12})$ has at least the same cardinality as $\mathrm{LCG}(D_{12})$. From this fact and Proposition 5.3 it follows that MG has exponential output complexity. □

Proposition 5.5 LVG *has exponential output complexity.*

`Proof:` We can use G_{12} to find an upper bound for the output complexity of $\mathrm{VG}_{\text{least-valued}}$. Since $B\backslash t$ and t/C are not unifiable, k (the maximum number of non-unifiable types assigned to one word) becomes at least 2.

Thus $\mathrm{OG}(D_{12}) \subseteq \mathrm{VG}_{\text{2-valued}}(D_{12})$. From this and Proposition 5.2 it follows that LVG has exponential output complexity. □

Proposition 5.6 VG_k *has exponential output complexity.*

`Proof:` In the case of $k = 1$, VG_k is equivalent to RG, so it will yield at most one grammar, it will be undefined in the case of G_{12}. For $k > 1$, there are at most k distinct types assigned to any word for all grammars in $\mathrm{VG}_k(D)$, so $k \geq 2$, and it will contain $\mathrm{OG}(D_{12})$. □

5.2 The Tractability of Producing Consistent Hypotheses

As discussed in Section 2.9, the complexity of producing hypotheses from a given class that are consistent with a given sample can be nontrivial and gives some indication of the 'difficulty of' learning a class. In this section one particular well-known NP-complete problem will be discussed, and in later sections it will be demonstrated that this problem can be reduced to consistency problems for the classes discussed in Chapter 4.

The following proposition (from Buszkowski and Penn (1990), page 442, Lemma 3) and corollary will be convenient for the NP-hardness proofs:

Proposition 5.7 *For every structure s, if $s \in \mathrm{FL}(G)$, then there exists a substitution τ such that $\tau[\mathrm{GF}(\{s\})] \subseteq G$.*

Corollary 5.8 *For every consistent learning function φ learning a subclass of* CatG *and every sequence σ for a language from that subclass there exists a substitution τ such that $\tau[\mathrm{GF}(\sigma)] \subseteq \varphi(\sigma)$, if $\varphi(\sigma)$ is defined.*

Thus, if $\mathrm{GF}(\sigma)$ assigns x different types to the same symbol that are pairwise not unifiable, the consistent learning function $\varphi(\sigma)$ assigns at least x different types to that same symbol.

5.2.1 The Node-Cover Problem

In order to prove NP-hardness of an algorithmic problem L, it suffices to show that there exists a polynomial-time reduction from an NP-complete problem L' to L.[4] We will present such a reduction using the node-cover (or vertex-cover) problem, a well-known NP-hard problem from the field of operations research that was first discussed in Karp (1972).

Definition 5.9 *Let $G = (V, E)$ be an undirected graph, where V is a set of nodes and E is a set of edges, represented as tuples of nodes. A* node cover *of G is a subset $V' \subseteq V$ such that if $(u, v) \in E$, then $u \in V'$ or $v \in V'$ (or both). That is, each node 'covers' its incident edges, and a node cover for G is a set of nodes that covers all the edges in E. The* size *of a node cover is the number of nodes in it.*

The node-cover problem *is the problem of finding a node cover of minimum size (called an* optimal node cover*) in a given graph.*

The node cover problem can be restated as a decision problem*: does a node cover of given size k exist for some given graph?*

Proposition 5.10 *The decision problem related to the node-cover problem is* NP*-complete.*

Proposition 5.11 *The node-cover problem is* NP*-hard, the decision problem related to the node-cover problem is* NP*-complete, and the production problem is* NP*-hard.*

This decision problem has been called one of the 'six basic NP-complete problems' by Garey and Johnson (Garey and Johnson (1979), Chapter 3), and is known to be approximable within $2 - \frac{\log\log|V|}{2log|V|}$ (Monien and Speckenmeyer (1985); Bar-Yehuda and Even (1985)) and $2 - \frac{2\ln\ln|V|}{\ln|V|}$ (Halperin (2000)). It is not approximable within 1.1666, see Håstad (1997). Also see Cormen et al. (1990) for a discussion of node-covers.

5.3 The Complexity of Learning $\mathcal{G}_{k\text{-valued}}$

Since the formal proof of Proposition 5.12 below will be somewhat complex an informal sketch of its structure will first be given. Let graph *Graph* be given. Construct an alphabet A and a sample D, that is, a set of structures $D = \{S_0, \ldots, S_n\}$, using A, following some recipe so that this sample represents *Graph*. A consistent learning function φ presented with D can only conjecture grammars whose associated languages contain D. Using Corollary 5.8 it will be shown that, in order for these grammars to be in φ's class, they have to

[4] This methodology of reductions was introduced in Karp (1972), and is also known as many-to-one reduction.

correspond to node covers for *Graph* of at most some given size. Therefore, computing the conjecture after the last element of D is input solves the decision problem related to the node-cover problem, which is NP-complete.[5] Note that the procedure that converts *Graph* to a sample constructs an alphabet with a size linear in the size of *Graph*. This limits the result to the case where there is no bound on the size of the alphabet.

Proposition 5.12 *Let φ be a learning function*[6] *that, given $k \in \mathbb{N}^+$, can learn any of the classes $\mathcal{G}_{k\text{-valued}}$ from structures, and that,* for each *k, is responsive and consistent on this class and learns this class prudently. Deciding whether φ is defined for an arbitrary D is an* NP*-hard problem (given that there is no bound on the size of the alphabet).*

`Proof:` The decision version of the node-cover problem can be transformed in polynomial time to the problem of learning a k-valued grammar from structures by means of a learning function consistent on that class. That is, given a bound on the size of the node cover, the function will yield a solution, or will be undefined if no node cover of that size exists.[7]

The transformation of the initial graph to an input sample will now be detailed. Edges are numbered $1, \ldots, e$ and nodes are numbered $1, \ldots, v$. First, for every edge i in E, we introduce in the input sample D the structure $\texttt{ba(e,e}_i\texttt{)}$.

Let $\Sigma_1, \Sigma_2, \ldots$ be shorthand for $\texttt{ba(x,v}_1\texttt{)}$, $\texttt{ba(x,ba(x,v}_2\texttt{))}$, ..., respectively. Let the type $X_0^i \backslash \Gamma_i$ be the type assigned to $\texttt{v}_i$ in $\mathrm{GF}(\{\Sigma_i\})$. Note that for any i, j, Γ_i and Γ_j are not unifiable when $i \neq j$.[8]

Add to the sample $\texttt{ba(x,}\Sigma_i\texttt{)}$ for all nodes $1 \leq i \leq v$. For the two nodes $j, k \in V$ incident on edge i, add $\texttt{ba(ba(x,v}_j\texttt{),e}_i\texttt{)}$, $\texttt{ba(ba(x,v}_k\texttt{),e}_i\texttt{)}$.[9]

Let the value of max, which is the size of the desired node cover, be assigned to k, the maximum number of types we want to assign to any single symbol in the final conjectured grammar. If $max = 1$, let k be 2. We add to D structures of the same kind as $\Sigma_1, \ldots$ such that some symbols in $\mathrm{GF}(D)$ get assigned a number of types that cannot be unified with any other type assigned to the same symbol. This can be done by using a variant on the procedure for creating Σ-types which uses only forward application instead of only backward

[5] In Kanazawa (1998), for each of the classes $\mathcal{G}_{\mathrm{VG}_k}$, $\mathcal{G}_{\mathrm{LVG}}$ and $\mathcal{G}_{\mathrm{LCG}}$ two learning functions are defined, one that is conservative and one that is set-driven. Both are responsive, prudent, and consistent on their class for all these classes, so the proof of Proposition 5.12 and its corollaries is directly applicable.

[6] To be more precise, φ is a *learner synhesizer* (recall Subsection 2.8.1): given a description of the class in the form of k it will behave like a learner for that class.

[7] Note that this does not mean that the function is not responsive, since it will only be undefined if the input is not from a language from this class.

[8] It is easy to see that, using this procedure for generating n such types, this will increase the size of D by a factor only polynomial in n.

[9] We can also allow a single node in this set, this would correspond with reflexive connections in the graph. We ignore this possibility for the sake of clarity, since it does not affect the proof in any way.

application. [10] To avoid cluttering the proof these types will be denoted by the (possibly empty) list $Filler$. Add to D structures such that that in GF(D), $max-2$ (if $max = 1$, let this number be 0) $Filler$-types are assigned to symbols $\mathtt{e}_1, \ldots, \mathtt{e}_e$, $max-1$ (if $max = 1$, let this number be 0) $Filler$-types are assigned to symbols $\mathtt{v}_1, \ldots, \mathtt{v}_v$, and 1 $Filler$-type is assigned to $\mathtt{e}$ just if $max = 1$.

To represent graphs in a generic way, some types have indices characteristic for the graph, and some constants characteristic for the graph are also required. Node j is connected to k_j edges, which are all edges which are numbered with some e such that $vf_1(e) = ef_j(x)$ or $vf_2(e) = ef_j(x)$, where $1 \leq x \leq k_j$.

Edge e is incident on the two nodes i, j for which $vf_1(e) = ef_i(y)$, for some $1 \leq y \leq k_i$, and $vf_2(e) = ef_j(z)$, for some $1 \leq z \leq k_j$.

Let $G = \mathrm{GF}(D)$:

$$
G: \quad
\begin{array}{rcl}
\mathtt{e}_1 & \mapsto & E_1\backslash t, A_{vf_1(1)}\backslash t, A_{vf_2(1)}\backslash t, Filler \\
\ldots & & \\
\mathtt{e}_e & \mapsto & E_e\backslash t, A_{vf_1(e)}\backslash t, A_{vf_2(e)}\backslash t, Filler \\
 & & \\
\mathtt{e} & \mapsto & E_1, \ldots, E_e, Filler \\
 & & \\
\mathtt{v}_1 & \mapsto & X_0^1\backslash\Gamma_1, X_1^1\backslash A_{ef_1(1)}, \ldots, \\
 & & X_{k_1}^1\backslash A_{ef_1(k_1)}, Filler \\
\mathtt{v}_2 & \mapsto & X_0^2\backslash\Gamma_2, X_1^2\backslash A_{ef_2(1)}, \ldots, \\
 & & X_{k_2}^2\backslash A_{ef_2(k_2)}, Filler \\
\ldots & & \\
\mathtt{v}_v & \mapsto & X_0^v\backslash\Gamma_v, X_1^v\backslash A_{ef_v(1)}, \ldots, \\
 & & X_{k_v}^v\backslash A_{ef_v(k_v)}, Filler \\
 & & \\
\mathtt{x} & \mapsto & X_0^1, \ldots, X_{k_1}^1, \\
 & & X_0^2, \ldots, X_{k_2}^2, \\
 & & \ldots, \ldots, \\
 & & X_0^v, \ldots, X_{k_v}^v
\end{array}
$$

Suppose this sample D is input for φ_{VG_k}, $k = max$.[11] Then, by Corollary 5.8, for each $i, 1 \leq i \leq v$, the type $X_0^i\backslash\Gamma_i$ assigned to $\mathtt{v}_i$ has to unify with the only types it *can* unify with, which are $X_1^i\backslash A_{ef_i(1)} \ldots X_{k_i}^i\backslash A_{ef_i(k_i)}$. For every such series of unification steps a substitution of the form $\{\Gamma_i \leftarrow A_{ef_i(1)}, \ldots, \Gamma_i \leftarrow A_{ef_i(k_i)}\}$ is obtained.

At this point an index function for the Γ-subtypes in the assignments to $\mathtt{e}_1$, $\ldots$, $\mathtt{e}_e$ is needed, since these unification steps are dependent on the original

[10] A proof based on types containing only operator \, or only operator / is desirable since it is more general than a proof based on types containing both operators; such a result would then also hold for unidirectional subclasses of these classes. Using the same procedure for creating the Σ- and $Filler$ types creates complications that I have not yet been able to solve.

[11] We show only GF(D) instead of D since D's properties that are relevant to this discussion are much more accessible in this form.

graph. For this purpose, let the functions $gf_1(i)$ and $gf_2(i)$ denote the two nodes connected to edge i.

These substitutions yield grammar G' (the X-variables are renumbered for readability):

$$
G' : \quad
\begin{array}{rcl}
\mathtt{e}_1 & \mapsto & E_1\backslash t, \Gamma_{gf_1(1)}\backslash t, \Gamma_{gf_2(1)}\backslash t, Filler \\
\ldots & & \\
\mathtt{e}_e & \mapsto & E_e\backslash t, \Gamma_{gf_1(e)}\backslash t, \Gamma_{gf_2(e)}\backslash t, Filler \\
\\
\mathtt{e} & \mapsto & E_1, \ldots, E_e, Filler \\
\\
\mathtt{v}_1 & \mapsto & X^1\backslash\Gamma_1, Filler \\
\ldots & & \\
\mathtt{v}_v & \mapsto & X^v\backslash\Gamma_v, Filler \\
\\
\mathtt{x} & \mapsto & X^1, \ldots, X^v
\end{array}
$$

Now, in order to obtain a grammar that is k-valued ($k = max$), we need to unify two of the types assigned to $\mathtt{e}_i$, for all i. Since the Γ-types are not unifiable, this means that either $E_i\backslash t$ and $\Gamma_{gf_1(i)}\backslash t$, or $E_i\backslash t$ and $\Gamma_{gf_2(i)}\backslash t$ have to be unified. This will result either in the substitution $\{\Gamma_{gf_1(i)} \leftarrow E_i\}$ or in the substitution $\{\Gamma_{gf_2(i)} \leftarrow E_i\}$. Since $\mathtt{e} \mapsto E_1, \ldots, E_e$, this results in the assignment of either $\Gamma_{gf_1(i)}$ or $\Gamma_{gf_2(i)}$ to $\mathtt{e}$.

This unification step is intended to correspond to including node $gf_1(i)$ or $gf_2(i)$ in the node-cover.

At this point another index function is needed, this time for the Γ-types assigned to $\mathtt{e}$. For this purpose, let the functions $gef(1), \ldots, gef(max)$ denote the nodes in the node cover.

The final output of φ_{VG_k}, if it is defined, is G'':

$$
G'' : \quad
\begin{array}{rcl}
\mathtt{e}_1 & \mapsto & \Gamma_{gf_1(1)}\backslash t, \Gamma_{gf_2(1)}\backslash t, Filler \\
\ldots & & \\
\mathtt{e}_e & \mapsto & \Gamma_{gf_1(e)}\backslash t, \Gamma_{gf_2(e)}\backslash t, Filler \\
\\
\mathtt{e} & \mapsto & \Gamma_{gef(1)}, \ldots, \Gamma_{gef(max)}, Filler \\
\\
\mathtt{v}_1 & \mapsto & X^1\backslash\Gamma_1, Filler \\
\ldots & & \\
\mathtt{v}_v & \mapsto & X^v\backslash\Gamma_v, Filler \\
\\
\mathtt{x} & \mapsto & X^1, \ldots, X^v
\end{array}
$$

Whether or not all types assigned to $\mathtt{x}$ are unified has no consequence for the structure language.

The resulting grammar can be read as a solution by taking the set S of all the Γ-types assigned to $\mathtt{e}$, and adding node v to the solution for each $\mathtt{v}_i$ that

has type $X^i \backslash \Gamma_i$, $\Gamma_i \in S$, assigned to it.

Any grammar output by a consistent learner that only has k-valued grammars as domain, where, $k = max$, will look like G''. Since such a grammar will correspond to a node cover any function that can learn any of these classes prudently and is responsive and consistent on that class will be able to solve the decision problem related to the node-cover problem after a polynomial-time reduction. □

Corollary 5.13 *(Of the proof)*
Let φ be a learning function that learns $\mathcal{G}_{\text{least-valued}}$ from structures, and that is responsive and consistent on this class and learns this class prudently. Deciding whether φ is defined for an arbitrary D is an NP*-hard problem (given that there is no bound on the size of the alphabet).*

Obviously, exactly the same proof works for learning $\mathcal{G}_{\text{least-valued}}$, since, because of the introduction of the *Filler*-types, no grammar can be obtained from D with $k < max$, so the least value for k is max.

Corollary 5.14 *(Of the proof)*
Let φ be a learning function that learns $\mathcal{G}_{\text{least-card}}$ from structures, and that is responsive and consistent on this class and learns this class prudently. Deciding whether φ is defined for an arbitrary D is an NP*-hard problem (given that there is no bound on the size of the alphabet).*

The proof works for learning $\mathcal{G}_{\text{least-card}}$, since the k-valued grammar obtained by learning $\mathcal{G}_{k\text{-valued}}$ is optimal (all symbols have k non-unifiable types assigned, recall the remark concerning symbol `x`), and it can easily be verified that all optimal grammars obtainable from D have the same cardinality.

The proof of Proposition 5.12 cannot be used for $\mathcal{G}_{\text{minimal}}$. However, the relation between $\mathcal{G}_{\text{minimal}}$ and $\mathcal{G}_{\text{least-card}}$ provides a different route for proving NP-hardness.

Let φ be a computable function for a class $\mathcal{L}$ that learns $\mathcal{L}$ consistently. Then the learning function φ' for a class $\mathcal{L}', \mathcal{L} \subseteq \mathcal{L}'$ that learns $\mathcal{L}'$ consistently has a time complexity that is the same as, or worse than, the time complexity of φ. From this and Proposition 4.71 the following proposition is straightforward:

Proposition 5.15 *Assume that there is a learning function φ that can learn $\mathcal{G}_{\text{minimal}}$ from structures, and that is responsive and consistent on this class and learns this class prudently.*[12] *Deciding whether φ is defined for an arbitrary D is an* NP*-hard problem (given that there is no bound on the size of the alphabet).*

[12] Recall that it is an open question whether or not this class is learnable.

5.4 The Complexity of Learning $\mathcal{G}_{k\text{-valued}}$ Revisited

As pointed out before, the proof of Theorem 5.12 requires an alphabet of unbounded size. We will now provide an alternative proof that works in the case $|\Sigma| = 3$:

Theorem 5.16 *Let φ be a learning function*[13] *that can learn any of the classes $\mathcal{G}_{k\text{-valued}}$ from structures, and that,* for each k*, is responsive and consistent on this class and learns this class prudently. Deciding whether φ is defined for an arbitrary D is an* NP*-hard problem.*

`Proof:` The decision version of the node-cover problem can be transformed in polynomial time to the problem of learning a k-valued grammar from structures by means of a learning function consistent on that class. That is, given a bound on the size of the node cover, the function will yield a solution, or will be undefined if no node cover of that size exists.

The transformation of the initial graph to an input sample will now be detailed. The initial graph consists of edges, which are numbered $1, \ldots, e$, and nodes, which are numbered $1, \ldots, v$. First, for every edge i in E, we produce a structure $\texttt{fa(a,}\underbrace{\texttt{ba(w,}\ \ldots, \texttt{ba(w,}}_{i\ \text{times}}\texttt{fa(g,w))}\ldots\texttt{))}$. Construct a sample D from all these structures for $1 \leq i \leq e$.

Inclusion of these structures results in the assignment of types $\Pi_i = (W_x \backslash \ldots (W_{x+i-1} \backslash (A_i / W_{x+i})) \ldots)$, for all $1 \leq i \leq e$, to symbol `g` in GF(D) (obviously no pair Π_i, Π_j can be unified unless $i = j$). Symbol `a` gets assigned just the types t/A_i for all $1 \leq i \leq e$. Symbol `w` gets assigned just some number w of primitive types $W_{1\ldots w}$ in GF(D).

For each node j, $1 \leq j \leq v$, create the structure $\Omega_j = \underbrace{\texttt{ba(w,}\ldots\ \texttt{ba(w,}}_{j\ \text{times}}\texttt{g)}\ldots\texttt{)}$. For each edge i, add the structures $\underbrace{\texttt{ba(w,}\ \texttt{ba(w,}\ldots,}_{i\ \text{times}}\texttt{fa(}\Omega_{fe(i,a)}\texttt{,w)}\ldots\texttt{))}$ and $\underbrace{\texttt{ba(w,}\ \texttt{ba(w,}\ldots,}_{i\ \text{times}}\texttt{fa(}\Omega_{fe(i,b)}\texttt{,w)}\ldots\texttt{))}$ to D. Here $fe(i,a)$ and $fe(i,b)$ give (indices for) the two nodes incident on edge i.[14] This results in the assignment of types $\Gamma_{(i,a)} = W_y \backslash (W_{y+1} \backslash \ldots\ (\Lambda_{fe(i,a)} / W_{y+i}) \ldots)$ and $\Gamma_{(i,b)} = W_q \backslash (W_{q+1} \backslash \ldots\ (\Lambda_{fe(i,b)} / W_{q+i}) \ldots)$ to symbol `g` in GF(D), where $\Lambda_{fe(i,a)} = (W_z \backslash \ldots (W_{z+fe(i,a)-1} \backslash t) \ldots)$ and $\Lambda_{fe(i,b)} = (W_r \backslash \ldots (W_{r+fe(i,b)-1} \backslash t) \ldots)$.

Let max be the size of the desired node cover, and let $d = 2e - max$. Obviously $d \geq 0$. The last step in constructing D consists of adding 'filler' types, to

[13] Again, φ is actually a learner synthesizer.

[14] The `fa`-label is used here as a separator between the 'outer' and 'inner' `ba`-labels in the structure. Obviously the corresponding Γ types can only be unified if they have the same number of 'outer' and 'inner' $\backslash$ operators. This idea can be generalized so that it can be used to encode sequences of arbitrary length of natural numbers in categorial types.

pad the number of types assigned to a. These will look like $(W_y \backslash t)/ \ldots /W_{y+l-1}$ for a given l. Let $Filler$ denote a (possibly empty) set containing d types.

Let $G = \mathrm{GF}(D)$:

$$G: \quad \begin{array}{lcl} \mathtt{g} & \mapsto & \Gamma_{(1,a)}, \Gamma_{(1,b)}, \Pi_1, \\ & & \ldots \\ & & \Gamma_{(e,a)}, \Gamma_{(e,b)}, \Pi_e, \\ \mathtt{a} & \mapsto & t/A_1, \ldots, t/A_e, Filler \\ \mathtt{w} & \mapsto & W_1, W_2, \ldots \end{array}$$

Suppose this sample D[15] is input for φ_{VG_k}, $k = 2e$. Then, by Corollary 5.8, for each $i, 1 \leq i \leq e$, the type Π_i assigned to g has to unify with one of the two types it *can* unify with, i.e. either $\Gamma_{(i,a)}$ or $\Gamma_{(i,b)}$. For every such unification step a substitution, either $\{\Lambda_{fe(i,a)} \leftarrow A_i\}$ or $\{\Lambda_{fe(i,b)} \leftarrow A_i\}$, is obtained. This unification step is intended to correspond to including node $fe(i,a)$ or $fe(i,b)$, $1 \leq i \leq e$, in the node-cover.

Applying these substitutions to G produces grammar G':

$$G': \quad \begin{array}{lcl} \mathtt{g} & \mapsto & \Gamma_{(1,a)}, \Gamma_{(1,b)}, \\ & & \ldots \\ & & \Gamma_{(e,a)}, \Gamma_{(e,b)}, \\ \mathtt{a} & \mapsto & t/\Lambda_{fe(1,fc(1))} \ldots, t/\Lambda_{fe(e,fc(e))}, Filler \\ \mathtt{w} & \mapsto & W_1, W_2, \ldots \end{array}$$

Here $fc(i)$ is the choice function from $i, 1 \leq i \leq e$ to the node (either a or b) chosen to cover edge i in the final node cover.

In order to obtain a grammar that is k-valued ($k = 2e$), there cannot be more than $2e$ types assigned to a. Since there are $d = 2e - max$ 'filler' types assigned to a, the rest of the types can only be at most max types of the form t/Λ_x (these obviously cannot unify with any type in $Filler$). Since the Λ-types are pairwise not unifiable, this means that only max different types of the form t/Λ_x are assigned to a.

Note that whether or not all types assigned to w are unified has no consequence for the structure language associated with G', and that there remain exactly $2e$ Γ types assigned to g.

The grammar G' is the final output, if $\varphi_{\mathrm{VG}_k}(D), k = 2e$ is defined. This output can be read as a solution by adding node j to the node cover for each of the types of the form t/Λ_j assigned to a. Since the function is responsive and prudent, its being undefined for D implies there is no $G, D \in \mathrm{FL}(G), G \in \mathcal{G}_{k\text{-valued}}$, which means that a node cover of size max does not exist for $Graph$.

Any grammar output by a consistent function that has only k-valued grammars in its domain, where $k = 2e$, will look like G'. Since such a grammar will correspond to a node cover any function that can learn any of these classes prudently and is responsive and consistent on that class will be able to solve

[15] Again for readability only $\mathrm{GF}(D)$ is shown.

the decision problem related to the node-cover problem after a polynomial-time reduction. □

Corollary 5.17 *(Of the proof)*
Let φ be a learning function that learns $\mathcal{G}_{\text{least-valued}}$ from structures, and that is responsive and consistent on this class and learns this class prudently. Deciding whether φ is defined for an arbitrary D is an NP*-hard problem*

Obviously, exactly the same proof works for learning $\mathcal{G}_{\text{least-valued}}$, since, because of the introduction of the 'filler'-types, there cannot be any grammars obtained from D with $k < v$, so the least value for k is v.

Corollary 5.18 *(Of the proof)*
Let φ be a learning function that learns $\mathcal{G}_{\text{least-card}}$ from structures, and that is responsive and consistent on this class and learns this class prudently. Deciding whether φ is defined for an arbitrary D is an NP*-hard problem.*

The proof works for learning $\mathcal{G}_{\text{least-card}}$, since the k-valued grammar obtained by learning $\mathcal{G}_{k\text{-valued}}$ is optimal (all symbols have k non-unifiable types assigned, recall the remark concerning symbol `x`), and all optimal grammars obtainable from D have the same cardinality.

The proof of Theorem 5.16 cannot be used for $\mathcal{G}_{\text{minimal}}$. However, the relation between $\mathcal{G}_{\text{minimal}}$ and $\mathcal{G}_{\text{least-card}}$ provides a different route for proving NP-hardness.

Let φ be a computable function for a class $\mathcal{L}$ that learns $\mathcal{L}$ consistently. Then the learning function φ' for a class $\mathcal{L}', \mathcal{L} \subseteq \mathcal{L}'$ that learns $\mathcal{L}'$ consistently has a time complexity that is the same as, or worse than, the time complexity of φ. From this and Proposition 4.71 the following proposition is straightforward:

Proposition 5.19 *Assume that there is a learning function φ that can learn $\mathcal{G}_{\text{minimal}}$ from structures, and that is responsive and consistent on this class and learns this class prudently. Deciding whether φ is defined for an arbitrary D is an* NP*-hard problem.*

5.5 The Complexity of Learning $\mathcal{G}_{2\text{-valued}}$

It follows from Lemma 5.8 that if $\text{GF}(\sigma)$ assigns x different types to the same symbol that are pairwise not unifiable, the consistent learning function $\varphi(\sigma)$ hypothesizes a grammar that assigns at least x different types to that same symbol.

Definition 5.20 *Let* $\mathrm{or}(p_1, p_2)$ *be the function that, given the two propositions* p_1 *and* p_2*, yields the following sample* D_{or} *for a language in* $\mathcal{L}_{\text{2-valued}}$:

$$D_{or} = \left\{ \begin{array}{l} \texttt{fa(fa(fa(d,fa(a,a)),fa(b,b)),c)} \\ \texttt{fa(fa(fa(d,fa(c,c)),c),fa(c,c))} \\ \texttt{fa(fa(fa(d,c),fa(c,c)),fa(r,r))} \\ \texttt{c} \\ \texttt{fa(c,c)} \\ \texttt{fa(fa(r,r),r)} \\ \texttt{a} \\ \texttt{b} \\ \texttt{c} \\ \texttt{r} \end{array} \right\}$$

Lemma 5.21 *Let* p *and* q *be two propositions. Then* $p \vee q$ *if and only if the following holds:*

For any learning function φ *that is responsive and consistent on* $\mathcal{G}_{\text{2-valued}}$ *and learns that class prudently (from structures) there exists a substitution* Θ *such that* $\Theta[\mathrm{GF}(\mathrm{or}(p,q))] \subseteq \varphi(\mathrm{or}(p,q))$.

`Proof:` The general form for D_{or} from Definition 5.20 is :

$$\mathrm{GF}(D_{or}): \quad \begin{array}{lcl} \texttt{a} & \mapsto & A/A_2, A_2, t \\ \texttt{b} & \mapsto & B/B_2, B_2, t \\ \texttt{c} & \mapsto & C, \\ & & D/D_2, D_2, \\ & & E, \\ & & F/F_2, F_2, \\ & & G, \\ & & H/H_2, H_2, \\ & & t, \\ & & t/J, J \end{array} \qquad \begin{array}{lcl} \texttt{d} & \mapsto & ((t/A)/B)/C, \\ & & ((t/D)/E)/F, \\ & & ((t/G)/H)/I \\ \texttt{r} & \mapsto & t, \\ & & I/I_2, \\ & & I_2, \\ & & (t/K)/L, \\ & & K, \\ & & L \end{array}$$

The symbol `r` has t and some complex types assigned to it. Since a constant and a complex term cannot be unified, the complex terms have to be unified. Since I_2, K and L occur in these complex types they have to unify with t. The same reasoning can be applied to the types assigned to symbols `a`, `b` and `c`, thus G' is obtained:

$$
\begin{array}{rcll}
 & \mathtt{a} & \mapsto & A/t, t \\
 & \mathtt{b} & \mapsto & B/t, t \\
 & \mathtt{c} & \mapsto & C, E, G, \\
 & & & t, t/t \\
G' : & \mathtt{d} & \mapsto & ((t/A)/B)/C, \\
 & & & ((t/t)/E)/t, \\
 & & & ((t/G)/t)/(t/t) \\
 & \mathtt{r} & \mapsto & t, \\
 & & & (t/t)/t
\end{array}
$$

Depending on p_1 being true or false, let $\mathtt{a} \mapsto t/t$ or $\mathtt{a} \mapsto (t/t)/t$, respectively. Depending on p_2 being true or false, let $\mathtt{b} \mapsto t/t$ or $\mathtt{b} \mapsto (t/t)/t$, respectively. If both p_1 and p_2 are false, A and B have to be substituted by (t/t), resulting in G'':

$$
\begin{array}{rcll}
 & \mathtt{a} & \mapsto & (t/t)/t, t \\
 & \mathtt{b} & \mapsto & (t/t)/t, t \\
 & \mathtt{c} & \mapsto & C, E, G, \\
 & & & t, t/t \\
G'' : & \mathtt{d} & \mapsto & ((t/(t/t))/(t/t))/C, \\
 & & & ((t/t)/E)/t, \\
 & & & ((t/G)/t)/(t/t) \\
 & \mathtt{r} & \mapsto & t, \\
 & & & (t/t)/t
\end{array}
$$

One glance at the three types assigned to `d` will show that none of them can be unified with any of the others, so there is no Θ such that $\Theta[G''] \in \mathcal{G}_{2\text{-valued}}$. Thus $\neg(p_1 \vee p_2)$ implies that there is no Θ such that $\Theta[\mathrm{GF}(\mathrm{or}(p_1, p_2))] \subseteq \varphi(\mathrm{or}(p, q))$, for any learning function φ that is responsive and consistent on $\mathcal{G}_{2\text{-valued}}$ and learns that class prudently (from structures).

In the other cases – i.e., either p_1 or p_2 or both are true – the required substitution exists, this can easily be checked by the reader. □

Note that the proof of both Lemma 5.21 and the following theorem use the right slash / exclusively. The subclass of $\mathcal{G}_{2\text{-valued}}$ with this property will be denoted $\mathcal{G}_{2\text{-valued}} \restriction \{/\}$.[16]

Theorem 5.22 *Let φ be a learning function that learns $\mathcal{G}_{2\text{-valued}} \restriction \{/\}$ from structures, and that is responsive and consistent on this class and learns this class prudently. Deciding whether φ is defined for an arbitrary D is an* NP*-hard problem (given that there is no bound on the size of the alphabet).*

`Proof:` It will be shown that the production version of the node-cover problem can be reduced in polynomial time to a learning problem for the class

[16] Unidirectional *rigid* grammars that only use / generate a subclass of the class of *simple languages*, which is a subclass of the deterministic context-free languages, see Harrison (1978).

$\mathcal{G}_{\text{2-valued}} \upharpoonright \{/\}$, thus showing NP-hardness. We proceed by demonstrating a procedure for rewriting a graph as a sample for this class and interpreting the output of a learning function as a node-cover.

Let $e = |E|, v = |V|$. The coding defined in Definition 5.20 can be used to enforce the constraint that, for every edge in the graph, at least one of the nodes incident on that edge is included in the cover. Doing this for every edge in the graph requires e versions of the sample in Definition 5.20, each with its own unique symbols (for convenience, these will be written as indexed versions of the original symbols). Since detailing this sample will unnecessarily clutter the proof it will simply be assumed that e2or -a function from E to sample D_{or} based on the function or - has the required properties.

What remains to be shown is how a bound on the size of the cover can be translated to the problem of learning a 2-valued grammar. Let b be this bound, and $size = v - b$.

Let D_v be the following sample:

$$\left\{\begin{array}{l}
\mathtt{x}_1 \\
\mathtt{fa(fa(s1,fa(g,f)),fa(e,f))} \\
\mathtt{fa(s1,fa(e,f))} \\
\mathtt{fa(x}_1\mathtt{,fa(s1,fa(v}_1\mathtt{,f)))} \\
\\
\text{For every } v_i \in V, v > 1: \\
\mathtt{x}_i \\
\mathtt{fa(x}_i\mathtt{(fa(fa(s}_i\mathtt{,fa(g,f)),fa(e,f)))} \\
\mathtt{fa(x}_{i-1}\mathtt{,fa(s}_i\mathtt{,fa(e,f)))} \\
\mathtt{fa(x}_i\mathtt{,fa(s}_i\mathtt{,fa(v}_i\mathtt{,f)))} \\
\\
\mathtt{e} \\
\mathtt{fa(e,f)} \\
\mathtt{fa(x}_v\mathtt{,fa(t,f))} \\
\mathtt{fa(fa(}\underbrace{\mathtt{...fa(fa(t,fa(e,f)),fa(e,f)),...}}_{size \text{ times}}\mathtt{),f)} \\
\mathtt{t} \\
\mathtt{fa(fa(g,fa(e,f)),fa(e,f))}
\end{array}\right\}$$

Note that this sample does not have symbols in common with D_{or}, which allows us to consider their general forms in isolation from one another. Eventually a relation between the two has to be established, so for this purpose certain symbols occurring in the two samples will be identified later.

Let $G = \text{GF}(D_v)$:

$$
G: \quad
\begin{array}{rcl@{\qquad}rcl}
\texttt{s1} & \mapsto & (t/E_0)/A_1, & \texttt{t} & \mapsto & T/G, \\
 & & t/E_1, & & & (\underbrace{(t/\ldots)/t}_{size \text{ times}})/I, \\
 & & X_{1,3}/V_1 & & & t \\
\texttt{s2} & \mapsto & (X_{2,1}/E_2)/A_2, & \texttt{e} & \mapsto & t, \\
 & & X_{2,2}/E_3, & & & t/F, \\
 & & X_{2,3}/V_2 & & & J_1/L_1, \\
\ldots & & & & & J_2/L_2, \\
\texttt{s}_v & \mapsto & (X_{v,1}/E_v)/A_v, & & & E_0/H_0, \\
 & & X_{v,2}/E_{v+1}, & & & E_1/H_1, \\
 & & X_{v,3}/V_v & & & E_2/H_2, \\
 & & & & & \ldots \\
\texttt{x}_1 & \mapsto & t, & \texttt{v}_1 & \mapsto & V_1/N_1 \\
 & & t/X_{1,3}, & \texttt{v}_2 & \mapsto & V_2/N_2 \\
 & & t/X_{2,1}, & \ldots & & \\
 & & t/X_{2,2} & \texttt{v}_v & \mapsto & V_v/N_v \\
\texttt{x}_2 & \mapsto & t, & \texttt{f} & \mapsto & F, G, I, L_1, L_2, \\
 & & t/X_{2,3} & & & H_0, H_1, H_2, \ldots, \\
\ldots & & & & & M_1, M_2, \ldots, \\
\texttt{x}_{v-1} & \mapsto & t, & & & N_1, N_2, \ldots \\
 & & t/X_{v-1,3}, & \texttt{g} & \mapsto & (t/J_1)/J_2, \\
 & & t/X_{v,1}, & & & A_1/M_1, \\
 & & t/X_{v,2} & & & A_2/M_2, \\
\texttt{x}_v & \mapsto & t, & & & \ldots \\
 & & t/X_{v,3}, & & & \\
 & & t/T & & &
\end{array}
$$

Unifying all complex types assigned to $\texttt{x}_i$, for any $1 \leq i \leq v$ yields grammar G':

$$
G' : \quad
\begin{array}{lcl}
\texttt{s1} & \mapsto & (t/E_0)/A_1, \\
 & & t/E_1, \\
 & & X_1/V_1 \\
\texttt{s2} & \mapsto & (X_1/t)/A_2, \\
 & & X_1/E_2, \\
 & & X_2/V_2 \\
\ldots & & \\
\texttt{s}_v & \mapsto & (X_{v-1}/t)/A_v, \\
 & & X_{v-1}/E_v, \\
 & & X_v/V_v \\
 & & \\
\texttt{x}_1 & \mapsto & t, \\
 & & t/X_1 \\
\ldots & & \\
\texttt{x}_{v-1} & \mapsto & t, \\
 & & t/X_{v-1} \\
\texttt{x}_v & \mapsto & t, \\
 & & t/X_v \\
 & & t/T
\end{array}
\qquad
\begin{array}{lcl}
\texttt{t} & \mapsto & T/G, \\
 & & (\underbrace{(t/\ldots)/t}_{size \text{ times}})/I, \\
 & & t \\
\texttt{e} & \mapsto & t, \\
 & & t/F, \\
 & & J_1/L_1, \\
 & & J_2/L_2, \\
 & & E_0/H_0, \\
 & & \ldots \\
\texttt{v}_1 & \mapsto & V_1/N_1 \\
\ldots & & \\
\texttt{v}_v & \mapsto & V_v/N_v \\
\texttt{f} & \mapsto & F, G, I, L_1, L_2, \\
 & & H_0, H_1, \ldots, \\
 & & M_1, M_2, \ldots, \\
 & & N_1, N_2, \ldots \\
\texttt{g} & \mapsto & (t/J_1)/J_2, \\
 & & A_1/M_1, \\
 & & A_2/M_2, \\
 & & \ldots
\end{array}
$$

Unifying all complex types assigned to `e`, `g` and `t` respectively yields G'':

$$
G'' : \quad
\begin{array}{lcl}
\texttt{s1} & \mapsto & (t/t)/(t/t), \\
 & & t/t, \\
 & & X_1/V_1 \\
\texttt{s2} & \mapsto & (X_1/t)/(t/t), \\
 & & X_1/t, \\
 & & X_2/V_2 \\
\ldots & & \\
\texttt{s}_v & \mapsto & (X_{v-1}/t)/(t/t), \\
 & & X_{v-1}/t, \\
 & & X_v/V_v \\
\texttt{x}_1 & \mapsto & t, \\
 & & t/X_1 \\
\ldots & & \\
\texttt{x}_v & \mapsto & t, \\
 & & t/X_v, \\
 & & t/\underbrace{(t/\ldots)/t}_{size \text{ times}}
\end{array}
\qquad
\begin{array}{lcl}
\texttt{t} & \mapsto & (\underbrace{(t/\ldots)/t}_{size \text{ times}})/G, \\
 & & t \\
\texttt{e} & \mapsto & t, \\
 & & t/F \\
\texttt{v}_1 & \mapsto & V_1/N_1 \\
\ldots & & \\
\texttt{v}_v & \mapsto & V_v/N_v \\
\texttt{f} & \mapsto & F, G, t, \\
 & & N_1, N_2, \ldots \\
\texttt{g} & \mapsto & (t/t)/t
\end{array}
$$

Since D_v is a sample for a language in $\mathcal{L}_{2\text{-valued}}$, for all $1 \leq i \leq v$, the type X_i/V_i assigned to $\mathtt{s}_i$ has to be unified with one of the two other types assigned to $\mathtt{s}_i$. Thus for any $G''' = \varphi(D_v)$, $G''' = \Theta[G'']$, where Θ unifies the type X_i assigned to symbol $\mathtt{x}i$ with either X_{i-1}/t or X_{i-1} (in the case $i = 1$, with either t/t or t). Thus $\Theta[X_v]$ will have a minimum degree of 0 and a maximum degree of v, and this degree is exactly the number of types X_i unified with types X_{i-1}/t.

The assignments to $\mathtt{x}_v$ necessitate the unification of t/X_v with $t/\underbrace{(t/\ldots)/t}_{size \text{ times}}$. This implies that X_v must have degree $size$, so exactly $size$ X_i types must have been unified with X_{i-1}/t types in the assignments to $\mathtt{s}i$. This implies that Θ has to unify a total number of $size$ V_i-types with t/t, and $v - size$ V_i-types with t. So $\Theta[GF(D_v)]$ will be defined if and only if the node-cover is of size b.

What remains to be shown is how D_{or} and D_v can be related to one another. As mentioned earlier certain symbols will be identified for this purpose.

Let $fe(i, a)$ and $fe(i, b)$ give indices for the two nodes incident on edge i. Recall that $G = \varphi(D_{or})$ contains for every $e_i \in E$ the following assignments:

$$\begin{array}{rcl} \mathtt{a}_i & \mapsto & A_i/t, t \\ \mathtt{b}_i & \mapsto & B_i/t, t \end{array}$$

Here A_i and B_i indicate (non)inclusion in the node-cover of nodes $v_{fe(i,a)}$ and $v_{fe(i,b)}$, respectively. Simply identifying the symbols $\mathtt{a}_i$ and $\mathtt{b}_i$ with the right symbols $\mathtt{v}_{fe(i,a)}$ and $\mathtt{v}_{fe(i,b)}$ will ensure that every variable $V_j, 1 \leq j \leq v$ is identified with all the A- and B-variables in $\mathrm{GF}(D_{or})$ that it should be identified with, given the graph. Since, given the constraints on the learning function and class, D_{or} only allows for assignments of t ('true', or 'assigned to cover') or t/t ('false', or 'not assigned to cover') to variables A_i and B_i, by this identification, the same is true for the V-variables.

Since the sample D_{or} ensures that all edges in E are covered, and D_v ensures that the cover contains only a given number of nodes, a learning function for the class $\mathcal{G}_{2\text{-valued}}$ that is responsive and consistent on this class and learns this class prudently can solve *any* (production version of the) node-cover problem, which is NP-hard. The computation time needed for the reduction from graph to sample is linear in e and v, as is the size of the alphabet, and the solution can be read from the assignments in G''' to symbols $\mathtt{v}_1 \ldots \mathtt{v}_v$ in linear time. Thus, updating the hypothesis for such a learning function is NP-hard in the size of the alphabet. □

Since the direction of the slash is immaterial for the proof of Theorem 5.22, the following is a corollary of this proof:

Corollary 5.23 *Let φ be a learning function that learns $\mathcal{G}_{2\text{-valued}} \restriction \{\backslash\}$ from structures, and that is responsive and consistent on this class and learns this class prudently. Deciding whether φ is defined for an arbitrary D is an* NP*-hard problem (given that there is no bound on the size of the alphabet).*

The proof of Theorem 5.22 can easily be adapted to the class $\mathcal{G}_{k\text{-valued}}$, for any given $k > 2$, by extending the sample with structures that result in the assignment to all $symbol \in \Sigma$ of extra types that are pairwise not unifiable. This can be done using one slash exclusively, by including the following structure in the sample for all symbols *symbol*, and for as many different x's as needed: $\texttt{fa(}\underbrace{\texttt{...fa(fa(}symbol\texttt{,fa(e,f)),fa(e,f)),...}}_{x \text{ times}}\texttt{)}$. This will result in the assignment $symbol \mapsto (\underbrace{(t/\ldots)/t}_{x \text{ times}})$ being included in any $G = \varphi(\mathrm{GF}(D)), G \in \mathcal{G}_{2\text{-valued}}$. Thus:

Corollary 5.24 *Let φ be a learning function that learns $\mathcal{G}_{k\text{-valued}} \restriction \{/\}, k \geq 2$ from structures, and that is responsive and consistent on this class and learns this class prudently. Deciding whether φ is defined for an arbitrary D is an* NP*-hard problem (given that there is no bound on the size of the alphabet).*

And since the direction of the slash is immaterial with respect to complexity, we of course also get:

Corollary 5.25 *Let φ be a learning function that learns $\mathcal{G}_{k\text{-valued}} \restriction \{\backslash\}, k \geq 2$ from structures, and that is responsive and consistent on this class and learns this class prudently. Deciding whether φ is defined for an arbitrary D is an* NP*-hard problem (given that there is no bound on the size of the alphabet).*

In general, proving NP-hardness of learning class $\mathcal{L}$ under a set of constraints C does not imply that the result holds for learning a class $\mathcal{L}', \mathcal{L} \subset \mathcal{L}'$ under C, since a learning function for $\mathcal{L}'$ would not in general be prudent with respect to $\mathcal{L}$. The function could therefore be able to 'postpone' certain conjectures by hypothesizing languages in $\mathcal{L}' - \mathcal{L}$, thus invalidating our type of proof. Given Lemma 5.8 however it should be clear that allowing slashes in both directions does not affect our proof at all, thus it follows that

Theorem 5.26 *Let φ be a learning function that learns $\mathcal{G}_{k\text{-valued}}, k \geq 2$ from structures, and that is responsive and consistent on this class and learns this class prudently. Deciding whether φ is defined for an arbitrary D is an* NP*-hard problem (given that there is no bound on the size of the alphabet).*

Considering these results it is natural to ask whether these problems are also NP-*complete*. In order to prove this it has to be shown that the problem is in NP, for example by showing that verification of solutions is in P. In this case the complexity of the verification problem would be related with the complexity of problems like deciding consistency of the conjecture with the input and membership of the conjecture in the class. However, the question whether the conjecture is *justified* given the input is problematic, since this notion is based on the concept of identification in the limit. It is not clear how this constraint can be expressed in terms of computational complexity. (Cf Section 2.9.)

5.6 Consistent Identification in the Limit of Rigid Grammars from Strings is NP-hard

Even though Lemma 5.8 is restricted to structure languages, it can also be used for dealing with string languages in special cases. Consider the following lemma that details a sample such that there is only one derivation consistent with each string, and the resulting type assignments only contain a constant as primitive type.

Lemma 5.27 *Let* $D = \{\texttt{t}, \texttt{te}, \texttt{jt}, \texttt{en}, \texttt{xee}, \texttt{kj}, \texttt{tjy}, \texttt{uyt}, \texttt{up}, \texttt{jhyt}, \texttt{jhp}, \texttt{kgu}, \texttt{gut}, \texttt{jijht}, \texttt{tjijhy}, \texttt{nb}, \texttt{ekz}, \texttt{cnn}, \texttt{tcn}, \texttt{zq}, \texttt{rz}, \texttt{onq}, \texttt{ron}, \texttt{ttf}, \texttt{nl}, \texttt{kmb}, \texttt{tmdb}, \texttt{stt}\}$*. Then, for any responsive and prudent* φ *that learns rigid grammars consistently,* $G = \varphi(D)$*:*

$$
G: \quad
\begin{array}{lcllcl}
\texttt{i} & \mapsto & ((t/t)\backslash(t/t))/((t/t)/((t/t)\backslash(t\backslash t))) & \texttt{t} & \mapsto & t \\
\texttt{e} & \mapsto & t\backslash t & \texttt{b} & \mapsto & A\backslash t \\
\texttt{j} & \mapsto & t/t & \texttt{z} & \mapsto & B\backslash A \\
\texttt{n} & \mapsto & A & \texttt{c} & \mapsto & (t\backslash t)/A \\
\texttt{x} & \mapsto & (t/(t\backslash t))/(t\backslash t) & \texttt{q} & \mapsto & (B\backslash A)\backslash t \\
\texttt{k} & \mapsto & B & \texttt{r} & \mapsto & t/(B\backslash A) \\
\texttt{y} & \mapsto & (t/t)\backslash(t\backslash t) & \texttt{o} & \mapsto & B\backslash(A/A) \\
\texttt{u} & \mapsto & (t/t)/((t/t)\backslash(t\backslash t)) & \texttt{f} & \mapsto & t\backslash(t\backslash t) \\
\texttt{p} & \mapsto & ((t/t)/((t/t)\backslash(t\backslash t)))\backslash t & \texttt{l} & \mapsto & A\backslash t \\
\texttt{h} & \mapsto & (t/t)\backslash((t/t)/((t/t)\backslash(t\backslash t))) & \texttt{m} & \mapsto & B\backslash A \\
\texttt{g} & \mapsto & (t/t)/((t/t)/((t/t)\backslash(t\backslash t))) & \texttt{d} & \mapsto & (B\backslash A)\backslash(t\backslash A) \\
\texttt{s} & \mapsto & (t/t)/t & & &
\end{array}
$$

Here $B = t/(t/t)$ *and* $A = (t\backslash t)\backslash t$*.*

The proof involves checking the possible derivations for these strings and is very tedious, the curious are referred to Section 5.8.

Lemma 5.28 *Let* p *and* q *be two propositions, and* f *be the function that maps true to type* t/t*, and false to type* $(t/t)/((t/t)\backslash(t\backslash t))$*. Then* $p \vee q$ *if and only if*

$$(t/t)/t, t, (t/t)/((t/t)/((t/t)\backslash(t\backslash t))), \underline{f(p)}, (t/t)\backslash((t/t)/((t/t)\backslash(t\backslash t))),$$
$$((t/t)\backslash(t/t))/((t/t)/((t/t)\backslash(t\backslash t))), \underline{f(q)}, (t/t)\backslash((t/t)/((t/t)\backslash(t\backslash t))), t \Rightarrow t.$$

`Proof:` Obviously $p \vee q$ is false if and only if both p and q are false, and is true otherwise, and the resulting sequents should reflect this. Simply checking these sequents shows that this holds, see Section 5.8 for a detailed proof. □

The type assignments obtained from D will be used in the other lemmas in this chapter.

The following lemma details a sample that will result in the possibility of choice for φ between two possible type assignments for a given symbol.

Lemma 5.29 *Let D be as in Lemma 5.27. For a given constant v, let sample $D' = D \cup \bigcup_{i=1}^{v} \{\mathtt{v}_i\mathtt{ab}, \mathtt{tcv}_i\mathtt{a}\}$. Let $G = \varphi(D')$. Then, for every $1 \leq i \leq v$, symbol $\mathtt{v}_i$ is assigned either $t/(t/t)$ or $((t\backslash t)\backslash t)/((t/(t/t))\backslash((t\backslash t)\backslash t))$ in G.*

`Proof:` For every string $\mathtt{v}_i\mathtt{ab}$ there are three possible corresponding sequents:

$$\begin{array}{llll} 1 & B, & B\backslash A, \quad A\backslash t & \Rightarrow t \\ 2 & A/(B\backslash A), & B\backslash A, \quad A\backslash t & \Rightarrow t \\ 3 & (t/(A\backslash t))/(B\backslash A), & B\backslash A, \quad A\backslash t & \Rightarrow t \end{array}$$

For every string $\mathtt{tcv}_i\mathtt{a}$ there are two possible corresponding sequents. The third option is out by the following table:

$$\begin{array}{llllll} 1 & t, & (t\backslash t)/A, & B, & B\backslash A & \Rightarrow t \\ 2 & t, & (t\backslash t)/A, & A/(B\backslash A), & B\backslash A & \Rightarrow t \\ 3 & t, & (t\backslash t)/A, & (t/(A\backslash t))/(B\backslash A), & B\backslash A & \Rightarrow t \end{array}$$

Thus, for every $1 \leq i \leq v$, symbol $\mathtt{v}_i$ is assigned either $t/(t/t)$ or $((t\backslash t)\backslash t)/((t/(t/t))\backslash((t\backslash t)\backslash t))$ in $\varphi(D')$. □

The following lemma shows how one can construct a sample that places a bound on the number of types that are assigned a particular type.

Lemma 5.30 *Let D' be as in Lemma 5.29, and let $D'' =$*

$$D' \cup \{\underbrace{\mathtt{z}\ldots\mathtt{z}}_{c \text{ times}}\mathtt{w}, \underbrace{\mathtt{ov}_1\mathtt{z}\ldots\mathtt{ov}_t\mathtt{z}}_{t \text{ times}}\mathtt{w}\}$$

for some given constants c and t, $c \geq t$.

Let $C = \{i | 1 \leq i \leq t, \mathtt{v}_i \mapsto A \in \varphi(D'')\}$ and $N = \{i | 1 \leq i \leq t, \mathtt{v}_i \mapsto A/(B\backslash A) \in \varphi(D'')\}$. Then $|C| = c - t$ and $|V| = 2t - c$.

`Proof:` The string $\underbrace{\mathtt{z}\ldots\mathtt{z}}_{c \text{ times}}\mathtt{w}$ implies $\mathtt{w} \mapsto \underbrace{(t/(t/t))\backslash((t\backslash t)\backslash t)\backslash(\ldots\backslash t)\ldots)}_{c \text{ times}}$.

Finally, consider string $\underbrace{\mathtt{ov}_1\mathtt{z}\ldots\mathtt{ov}_t\mathtt{z}}_{t \text{ times}}$. Each substring $\mathtt{ov}_x\mathtt{z}$ implies the antecedent $(B\backslash A)/A, \frac{A}{A/(B\backslash A)}, B\backslash A$, which reduces to $(B\backslash A)/A, (B\backslash A)/A$ or $(B\backslash A)/A$, depending on the second type being A or $A/(B\backslash A)$, respectively. Since the type assigned to $\mathtt{w}$ selects for c types of the form $A/(B\backslash A)$ to the left, the number of distinct symbols $\mathtt{v}_x$ that are assigned A plus twice the number of distinct symbols $\mathtt{v}_x$ that are assigned $A/(B\backslash A)$ has to be c. Since there are t distinct symbols $\mathtt{v}_x$ and $t > c$, the former number of symbols is $c - t$ and the latter $2t - c$. □

The following lemma presents a method for creating a specific relation between the types assigned to two distinct symbols.

Lemma 5.31 *Let $B = t/(t/t)$ and $A = (t\backslash t)\backslash t$. Let D'' be as in Lemma 5.30, and $D''' =$*

$$D'' \cup \{\mathtt{r}_1\mathtt{jtf}, \ldots, \mathtt{r}_v\mathtt{jtf}, \mathtt{r}_{f(1)}\mathtt{w}_1\mathtt{y}, \ldots, \mathtt{r}_{f(v)}\mathtt{w}_v\mathtt{y}, \mathtt{teer}_1\mathtt{ov}_1\mathtt{zdf}, \ldots, \mathtt{teer}_v\mathtt{ov}_v\mathtt{zdf}\}$$

where f is a function with domain $[1 \ldots v]$ and range $[1 \ldots w]$ for some $v, w \in \mathbb{N}$. Then $\mathtt{w}_i$ is assigned t/t iff $\mathtt{v}_i$ is assigned A, and $\mathtt{w}_i$ is assigned $(t/t)/((t/t)\backslash(t\backslash t))$ iff $\mathtt{v}_i$ is assigned $A/(B\backslash A)$.

`Proof:` Let D'' be as in Lemma 5.30. Every string $\mathtt{r}_i\mathtt{jtf}$ corresponding to sequent $R_i, t/t, t, t\backslash(t\backslash t) \Rightarrow t$ implies the assignment of either t, $t/(t/t)$, $(t/(t\backslash t))/(t/t)$, or $((t/(t\backslash(t\backslash t))/t)/(t/t)$ to $\mathtt{r}_i$.

Let p_i be the proposition stating that node i is in the cover. Then the string $\mathtt{r}_{f(i)}\mathtt{w}_i\mathtt{y}$ corresponding to sequent $R_{f(i)}, f(p_i), (t/t)\backslash(t\backslash t) \Rightarrow t$ implies that $\mathtt{r}_{f(i)}$ gets assigned t or $t/(t/t)$, depending on whether $\mathtt{w}_i$ is assigned t/t or $(t/t)/((t/t)\backslash(t\backslash t))$, respectively. Other type assignments are excluded by the string $\mathtt{r}_i\mathtt{jtf}$.

The string $\mathtt{teer}_i\mathtt{ov}_i\mathtt{zdf}$ corresponding to the sequent

$$t, t\backslash t, t\backslash t, \frac{t}{B}, (B\backslash A)/A, \frac{A}{A/(B\backslash A)}, B\backslash A, (B\backslash A)\backslash(t\backslash A), t\backslash(t\backslash t) \Rightarrow t$$

which reduces to

$$t, t\backslash t, t\backslash t, \frac{t}{B}, \frac{B\backslash A}{B\backslash A, B\backslash A}, (B\backslash A)\backslash(t\backslash A), t\backslash(t\backslash t) \Rightarrow t$$

then implies that $\mathtt{r}_i$ is assigned t iff $\mathtt{v}_i$ is assigned A, and $\mathtt{r}_i$ is assigned $t/(t/t)$ iff $\mathtt{v}_i$ is assigned $A/(B\backslash A)$. Thus $\mathtt{w}_i$ is assigned t/t iff $\mathtt{v}_i$ is assigned A, and $\mathtt{w}_i$ is assigned $(t/t)/((t/t)\backslash(t\backslash t))$ iff $\mathtt{v}_i$ is assigned $A/(B\backslash A)$. □

With the preliminaries taken care of we can now turn to the complexity result itself.

Theorem 5.32 *Let φ be a learning function that learns $\mathcal{G}_{1-\text{valued}}$ from strings, and that is responsive and consistent on this class and learns this class prudently. Deciding whether φ is defined for an arbitrary D is an* NP*-hard problem (given that there is no bound on the size of the alphabet).*

`Proof:` Let $G = (V, E)$ be a graph as in Definition 5.9, let $v = |V|$, let $e = |E|$, and let *size* be the desired size of the node-cover. Let D be as defined in Lemma 5.27 (which provides a sample containing the type assignments needed in the other lemmas).

A graph is coded in the following way: let D' be the sample consisting of all strings $\{\mathtt{v}_i\mathtt{ab}, \mathtt{tcv}_i\mathtt{a}\}, 1 \leq i \leq v$ as in Lemma 5.29. Each $\mathtt{v}_i$ is intended to represent the node $i \in V$. This sample provides the learning function with a choice for in- or exclusion in the cover.

Lemma 5.30 provides means for stating the desired size for the cover, just let $t = v$ and $c = v + size$.

Lemma 5.31 ties Lemmas 5.30 and 5.29 together, with the symbols $\mathtt{r}_1, \ldots, \mathtt{r}_v$ acting as 'intermediaries' between the symbols $\mathtt{v}_1, \ldots, \mathtt{v}_v$ and symbols $\mathtt{w}_1, \ldots, \mathtt{w}_{2e}$. The function f should reflect the structure of G, i.e. for every edge $i = (x, y) \in E$, $f(2i-1) = x$ and $f(2i) = y$. Thus every $\mathtt{v}_i$ represents a node in V and every $\mathtt{w}_j$ a *connection* of a node to an edge.

The coding defined in Lemma 5.28 can be used to enforce the constraint that at least one of the nodes incident on an edge is included in the cover. Let $D'' = \bigcup_{i=1}^{e} \{\mathtt{stgw}_{2i-1}\mathtt{hiw}_{2i}\mathtt{ht}\}$ for every edge $i \in E$.

Let $V' = \bigcup\{i | \mathtt{v}_i \mapsto (t\backslash t)\backslash t \in RG, 1 \leq i \leq v\}$, where RG is any rigid grammar consistent with $D''' = D \cup D' \cup D''$. By Lemma 5.30 there can only be *size* such type assignments in RG. By Lemma 5.28 at least one of the two symbols representing a node incident on a given edge is assigned t/t in RG, and thus, by Lemma 5.31, the symbol representing the corresponding node is assigned $(t\backslash t)\backslash t$. By Lemma 5.29 the learning function is free to assign any symbol $\mathtt{v}_i$ this type. Since we are assuming prudence $\varphi(D''')$ has to be a rigid grammar, and since we are assuming responsiveness $\varphi(D''')$ has to be defined if it exists. Therefore V' is a node cover for graph G of size *size*, if it exists. Since D''' can be constructed in polynomial time (relative to v and e) and V' can be constructed in polynomial time given RG[17], learning the class $\mathcal{G}_{1-\text{valued}}$ from strings by means of a function φ that is responsive and consistent on this class and learns this class prudently, is NP-hard in the size of the alphabet.[18] □

Note that this theorem does not imply that superclasses of this class are not learnable by a consistent function with polynomial update time. This may seem surprising since it can perform the same task as a learning function for rigid grammars, but it corresponds to dropping the prudency constraint: such a function could, before convergence, conjecture languages outside the class of rigid grammars and would thus not be forced to exhibit the behaviour needed for this proof.

However, it is possible to extend this proof for some superclasses. The proof of Lemma 5.27 demonstrates that it is quite easy to produce strings that yield type-assignments that are not pairwise unifiable. Doing this $k-1$ times for each symbol in a given alphabet, combined with the other strings mentioned in the lemmas, yields a proof of NP-hardness for deciding whether a consistent learning function for $\mathcal{G}_{k\text{-valued}}$ is defined given an arbitrary sample.

Our complexity result leaves open the question of an upper bound. This can be obtained using the Catalan numbers $C_n = \frac{1}{n+1}\binom{2n}{n}$, which yields the number of binary trees with $n+1$ leaves (see Stanley (1998)). Since these trees have

[17] Since RG can contain an arbitrary number of type assignments if Σ is not given this is not totally accurate, but in the case that the construction of V' is not bounded by some polynomial because $|RG|$ is not, the construction of RG from D''' obviously cannot be performed in polynomial time either, and in that case φ is not even in NP.

[18] The procedure requires v distinct symbols $\mathtt{v}$ and $2e$ distinct symbols $\mathtt{w}$ in Σ.

n internal nodes and every one of these can be labeled as forward or backward application, $2^{l-1}C_{l-1}$ gives the number of possible categorial derivations for a string of length l. Thus an algorithm that simply checks all combinations of these derivations for a sample D containing d sentences has to consider (at least) $\prod_{i=1}^{d} 2^{|s_i|-1} \cdot C_{|s_i|-1}$ grammars.

5.7 Conclusions and Further Research

It has been shown that learning any of the classes $\mathcal{G}_{\text{least-valued}}$, $\mathcal{G}_{\text{least-card}}$, and $\mathcal{G}_{\text{minimal}}$ from structures by means of a learning function that is consistent on its class is NP-hard in the size of the sample. The result for the classes $\mathcal{G}_{k\text{-valued}}$ is weaker: one function that can learn these classes *for each* k and is consistent on its class is NP-hard in the size of the sample. This leaves open the question whether there exist polynomial-time learning functions for $\mathcal{G}_{k\text{-valued}}$ for each k separately. Showing intractability for $k = 2$ would imply intractability for all $k > 1$, since $\mathcal{G}_{k\text{-valued}} \subseteq \mathcal{G}_{k+1\text{-valued}}$.

It has been shown that learning any class in $\mathcal{G}_{k\text{-valued}}, k \geq 2$ from structures by means of a learning function that is consistent on its class is NP-hard in the size of the *alphabet*. Note that these results hold just under the assumption that there is no bound on the size of the alphabet. It is an open question whether there exists a proof with an alphabet of some constant size. Complexity with respect to the size of the *sample* is still open, but in the light of our result this question seems of academic interest only. The problem does not scale well with the size of the alphabet (lexicon), which makes the problem intractable in practice.

It is a well-known fact that learning functions for any learnable class without consistency- and monotonicity constraints can be transformed to trivial learning functions that have polynomial update-time (see Subsection 2.9). It is an open question whether there exist 'intelligent' inconsistent learning functions that have polynomial update-time for the classes under discussion. In Lange and Wiehagen (1991) an example of such a function can be found that learns the class of all pattern languages (Angluin (1979)) and is computationally well-behaved given certain assumption about the distribution of the input.

Since the relation between structure language and string language is so clear-cut, it is in general easy to transfer results from one to the other. In Kanazawa (1998) some results concerning learnability of classes of structure languages were used to obtain learnability results for the corresponding classes of string languages. It might be possible to do the same with complexity results, i.e. obtain an NP-hardness result for learning $\mathcal{G}_{\text{least-valued}}$ from strings, for example.

Note that the proof of Proposition 5.15 nicely demonstrates that complexity results can be obtained even for classes for which learnability is still an open question.

The proof of Proposition 5.12, Proposition 5.16 and Theorem 5.32 rely on subclasses of languages that can all be identified with sequences that have a length polynomial in the size of their associated grammars. This is not necessarily true for any arbitrary language in the whole class, so data-complexity issues may make the complexity of learning these classes even worse than these results suggest.

Instead of investigating the complexity of learning for each distinct class on an individual basis, it would be nice to have insights into the direct relation between complexity and some structural properties of learnable classes of CCGs or related formalisms. This would be an interesting topic for future research.

Analyzing these classes in terms of intrinsic complexity (see Section 2.9) would yield insights into the relation between these and other classes, and into the structure of the complexity hierarchy of learnable classes in general.

It has been shown that learning any of the classes $\mathcal{G}_{k-\text{valued}}$, $k \geq 1$, from strings by means of a function φ that is responsive and consistent on its class and learns its class prudently, is NP-hard in the size of the alphabet. Note that this is a weaker result than NP-hardness in the size of the sample would be, it is an open question whether there exists a proof with an alphabet of some constant size.

A similar result for learning the class of pattern languages PAT consistently from informant[19] can be found in Wiehagen and Zeugmann (1994). It differs from ours in that it crucially depends on the membership test for PAT being NP-complete, whereas membership for (subclasses of) **AB** languages is known to be polynomial time.

The results in this chapter are of a technical nature and may seem to be of limited interest since thay apply just to the subclasses of CG defined by Kanazawa. However, they are also proof that there exist immediate practical consequences of the so-called consistency phenomenon as discussed in Section 2.9. Thus, the common approach of taking only *consistent* learning algorithms into account is dangerous in the sense that it may lead one to overlook efficient solutions to language learning problems.

5.8 Detailed Proofs

Proof of Lemma 5.27:
Let σ be the sequence $< \mathtt{t}, \mathtt{te}, \mathtt{jt}, \mathtt{en}, \mathtt{xee}, \mathtt{kj}, \mathtt{tjy} >$. It is obvious that φ's conjecture G has to include $\{\mathtt{t} \mapsto t, \mathtt{e} \mapsto t\backslash t, \mathtt{j} \mapsto t/t\}$. The assignments to `n`, `x`, `k` and `y` follow directly from these and the given strings.

Let σ' be the sequence $< \mathtt{uyt}, \mathtt{up} >$. The first string from this sequence corresponds to the sequent $U, (t/t)\backslash(t\backslash t), t \Rightarrow t$, so $\mathtt{u} \mapsto (t/t)/((t/t)\backslash(t\backslash t))$. Now consider the second string. It corresponds to the sequent $(t/t)/((t/t)\backslash(t\backslash t)), P \Rightarrow t$, so $\mathtt{p} \mapsto ((t/t)/((t/t)\backslash(t\backslash t)))\backslash t$.

[19] The result was conjectured to remain valid for learning from strings.

Let σ'' be the sequence $<$ jhyt, jhp $>$. The first string from this sequence corresponds to the sequent $t/t, H, (t/t)\backslash(t\backslash t), t \Rightarrow t$, so the type H can be:

1. $(t/t)\backslash((t/t)/((t/t)\backslash(t\backslash t)))$
2. $((t/t)\backslash(t/t))/((t/t)\backslash(t\backslash t))$
3. $(((t\backslash t)\backslash t)/t)/((t/t)\backslash(t\backslash t))$
4. $(t/t)/((t/t)\backslash(t\backslash t))$.

Now consider the second string. Given the possible assignments to the symbols it contains, it has to correspond to the sequent $t/t, H, ((t/t)/((t/t)\backslash(t\backslash t)))\backslash t \Rightarrow t$. Trying the four possible assignments to h will show that only the first, $(t/t)\backslash((t/t)/((t/t)\backslash(t\backslash t)))$ is possible.

Let σ''' be the sequence $<$ kgu, gut $>$. The first string from this sequence corresponds to the sequent $t/(t/t), G, (t/t)/((t/t)\backslash(t\backslash t)) \Rightarrow t$, so the type G can be:

1. $(t/t)/((t/t)/((t/t)\backslash(t\backslash t)))$
2. $((t/(t/t))\backslash t)/((t/t)/((t/t)\backslash(t\backslash t)))$
3. $(t/(t/t))\backslash(t/((t/t)/((t/t)\backslash(t\backslash t)))$

Now consider the second string. Given the possible assignments to the symbols it contains, it has to correspond to the sequent $G, (t/t)/((t/t)\backslash(t\backslash t)), t \Rightarrow t$. Only the first type, $(t/t)/((t/t)/((t/t)\backslash(t\backslash t)))$, is possible.

Let σ'''' be the sequence $\langle$jiut, tjiuy$\rangle$. The first string from this sequence corresponds to the sequent $t/t, I, (t/t)/((t/t)\backslash(t\backslash t)), t \Rightarrow t$, so the type I can be:

1. $((t/t)\backslash(t/t))/((t/t)/((t/t)\backslash(t\backslash t)))$
2. $(((t/t)\backslash t)/t)/((t/t)/((t/t)\backslash(t\backslash t)))$
3. $(t/t)/((t/t)/((t/t)\backslash(t\backslash t)))$

Now consider the second string. Given the possible assignments to the symbols it contains, it has to correspond to the sequent

$t, t/t, I, (t/t)/((t/t)\backslash(t\backslash t)), (t/t)\backslash(t\backslash t) \Rightarrow t$.

Only the first type, $((t/t)\backslash(t/t))/((t/t)/((t/t)\backslash(t\backslash t)))$, is possible.

From nb it follows immediately that b$\mapsto (t\backslash t)\backslash t)\backslash t$. The string ekz corresponds to sequent $t\backslash t, t/(t/t), Z \Rightarrow t$, from which follows immediately that z$\mapsto (t/(t/t))\backslash((t\backslash t)\backslash t)$. The string cnn corresponds to the sequent $C, (t\backslash t)\backslash t, (t\backslash t)\backslash t \Rightarrow t$. There are two possible types for C, namely $(t/((t\backslash t)\backslash t))/((t\backslash t)\backslash t)$ and $(t\backslash t)/((t\backslash t)\backslash t)$. The string tcn corresponds to the sequent $t, C, (t\backslash t)\backslash t \Rightarrow t$. Only the first type applies here, so c$\mapsto (t\backslash t)/((t\backslash t)\backslash t)$.

The string zq immediately implies q$\mapsto ((t/(t/t))\backslash((t\backslash t)\backslash t))\backslash t$. The string rz immediately implies r$\mapsto t/((t/(t/t))\backslash((t\backslash t)\backslash t))$. The string onq corresponds

to the sequent $t/(t/t), O, (t\backslash t)\backslash t, ((t\backslash t)\backslash t)\backslash t \Rightarrow t$. This implies that O is either $(t/(t/t))\backslash((t\backslash t)\backslash t)/((t\backslash t)\backslash t)$ or $(t/((t/(t/t))\backslash((t\backslash t)\backslash t)\backslash t))/((t\backslash t)\backslash t)$. The string `ron` corresponds to the sequent $t/(t/(t/t))\backslash((t\backslash t)\backslash t), O, (t\backslash t)\backslash t \Rightarrow t$. Only the first possibility for O works here, so $\mathtt{o}\mapsto (t/(t/t))\backslash((t\backslash t)\backslash t)/((t\backslash t)\backslash t)$.

By Theorem 5.8, $\Theta[\mathrm{GF}(\sigma)] \subseteq \varphi(\sigma)$, if $\varphi(\sigma)$ is defined. Since $\mathrm{GF}(D)$ only contains ground types for this sample, Θ must be the empty substitution, so any consistent hypothesis should contain $\mathrm{GF}(D)$. This also shows that the order of presentation of D is irrelevant.

Since φ is restricted to the class of rigid grammars no other types can be assigned to the symbols occurring in G. □

Proof of Lemma 5.28:
The truth table for $\vee$ shows that $p \vee q$ is false if and only if both p and q are false, and is true otherwise. Simply checking the four possible combinations for the categorial derivation yields:

1. Both p and q true implies $p \vee q$ true, so t should be derived:

   $(t/t)/t, t, (t/t)/((t/t)/((t/t)\backslash(t\backslash t))), \underline{t/t}, (t/t)\backslash((t/t)/((t/t)\backslash(t\backslash t))),$
   $((t/t)\backslash(t/t))/((t/t)/((t/t)\backslash(t\backslash t))), \overline{t/t}, (t/t)\backslash((t/t)/((t/t)\backslash(t\backslash t))), t \Rightarrow t$
   $(t/t)/t, t, (t/t)/((t/t)/((t/t)\backslash(t\backslash t))), \overline{(t/t)}, (t/t)/((t/t)\backslash(t\backslash t)),$
   $((t/t)\backslash(t/t))/((t/t)/((t/t)\backslash(t\backslash t))), t/t, (t/t)\backslash((t/t)/((t/t)\backslash(t\backslash t))), t \Rightarrow t$
   $(t/t)/t, t, t/t, ((t/t)\backslash(t/t))/((t/t)/((t/t)\backslash(t\backslash t))), t/t,$
   $(t/t)\backslash((t/t)/((t/t)\backslash(t\backslash t))), t \Rightarrow t$
   $(t/t)/t, t, t/t, ((t/t)\backslash(t/t))/((t/t)/((t/t)\backslash(t\backslash t))), (t/t)/((t/t)\backslash(t\backslash t)), t \Rightarrow t$
   $(t/t)/t, t, t/t, (t/t)\backslash(t/t), t \Rightarrow t$
   $(t/t)/t, t, t/t, t \Rightarrow t$
   $(t/t)/t, t, t \Rightarrow t$
   $t/t, t \Rightarrow t.$

2. Proposition p true, q false implies $p \vee q$ true, so t should be derived:

   $(t/t)/t, t, (t/t)/((t/t)/((t/t)\backslash(t\backslash t))), \underline{t/t}, (t/t)\backslash((t/t)/((t/t)\backslash(t\backslash t))),$
   $((t/t)\backslash(t\backslash t))/((t/t)/((t/t)\backslash(t\backslash t))), \underline{(t/t)/((t/t)\backslash(t\backslash t))},$
   $(t/t)\backslash((t/t)/((t/t)\backslash(t\backslash t))), t \Rightarrow t$
   $(t/t)/t, t, (t/t)/((t/t)/((t/t)\backslash(t\backslash t))), (t/t)/((t/t)\backslash(t\backslash t)),$
   $((t/t)\backslash(t\backslash t))/((t/t)/((t/t)\backslash(t\backslash t))), (t/t)/((t/t)\backslash(t\backslash t)),$
   $(t/t)\backslash((t/t)/((t/t)\backslash(t\backslash t))), t \Rightarrow t$
   $(t/t)/t, t, (t/t)/((t/t)/((t/t)\backslash(t\backslash t))), (t/t)/((t/t)\backslash(t\backslash t)), ((t/t)\backslash(t\backslash t)),$
   $(t/t)\backslash((t/t)/((t/t)\backslash(t\backslash t))), t \Rightarrow t$
   $(t/t)/t, t, (t/t)/((t/t)/((t/t)\backslash(t\backslash t))), t/t, (t/t)\backslash((t/t)/((t/t)\backslash(t\backslash t))), t \Rightarrow t$
   $(t/t)/t, t, (t/t)/((t/t)/((t/t)\backslash(t\backslash t))), (t/t)/((t/t)\backslash(t\backslash t)), t \Rightarrow t$
   $(t/t)/t, t, t/t, t \Rightarrow t$
   $(t/t)/t, t, t \Rightarrow t$
   $t/t, t \Rightarrow t.$

3. Proposition p false, q true implies $p \vee q$ true, so t should be derived:

$$(t/t)/t, t, (t/t)/((t/t)/((t/t)\backslash(t\backslash t))), \underline{(t/t)/((t/t)\backslash(t\backslash t))},$$
$$(t/t)\backslash((t/t)/((t/t)\backslash(t\backslash t))), ((t/t)\backslash(t\backslash t))/((t/t)/((t/t)\backslash(t\backslash t))), \underline{t/t},$$
$$(t/t)\backslash((t/t)/((t/t)\backslash(t\backslash t))), t \Rightarrow t$$
$$(t/t)/t, t, (t/t)/((t/t)/((t/t)\backslash(t\backslash t))), (t/t)/((t/t)\backslash(t\backslash t)),$$
$$(t/t)\backslash((t/t)/((t/t)\backslash(t\backslash t))), ((t/t)\backslash(t\backslash t))/((t/t)/((t/t)\backslash(t\backslash t))),$$
$$(t/t)/((t/t)\backslash(t\backslash t)), t \Rightarrow t$$
$$(t/t)/t, t, (t/t)/((t/t)/((t/t)\backslash(t\backslash t))), (t/t)/((t/t)\backslash(t\backslash t)),$$
$$(t/t)\backslash((t/t)/((t/t)\backslash(t\backslash t))), (t/t)\backslash(t\backslash t), t \Rightarrow t$$
$$(t/t)/t, t, t/t, (t/t)\backslash((t/t)/((t/t)\backslash(t\backslash t))), (t/t)\backslash(t\backslash t), t \Rightarrow t$$
$$(t/t)/t, t, (t/t)/((t/t)\backslash(t\backslash t)), (t/t)\backslash(t\backslash t), t \Rightarrow t$$
$$(t/t)/t, t, t/t, t \Rightarrow t$$
$$(t/t)/t, t, t \Rightarrow t$$
$$t/t, t \Rightarrow t.$$

4. Both propositions false implies $p \vee q$ false, so t should not be derivable:

$$(t/t)/t, t, (t/t)/((t/t)/((t/t)\backslash(t\backslash t))), \underline{(t/t)/((t/t)\backslash(t\backslash t))},$$
$$(t/t)\backslash((t/t)/((t/t)\backslash(t\backslash t))), ((t/t)\backslash(t\backslash t))/((t/t)/((t/t)\backslash(t\backslash t))),$$
$$\underline{(t/t)/((t/t)\backslash(t\backslash t))}, (t/t)\backslash((t/t)/((t/t)\backslash(t\backslash t))), t \not\Rightarrow t$$

   Consider the second to last type in the sequent. Since it selects for $(t/t)\backslash(t\backslash t)$ to the right and there is only t to the right of it, a type to the left of it must have $X/((t/t)\backslash((t/t)/((t/t)\backslash(t\backslash t))))$ or $X/(t/t)/((t/t)\backslash(t\backslash t))$ as its range (here X is an arbitrary type). There is no candidate for the former, there are two for the latter: the third and the sixth type in the sequent. We proceed by case analysis:

   (a) For the third type, $(t/t)/((t/t)/((t/t)\backslash(t\backslash t)))$, to combine with $(t/t)/((t/t)\backslash(t\backslash t)))$, the sequent between the third and eighth type must derive t/t. Thus it needs to be shown that

$$(t/t)/((t/t)\backslash(t\backslash t)), (t/t)\backslash((t/t)/((t/t)\backslash(t\backslash t))),$$
$$((t/t)\backslash(t\backslash t))/((t/t)/((t/t)\backslash(t\backslash t))), (t/t)/((t/t)\backslash(t\backslash t)) \not\Rightarrow t/t$$

   There is only one possible application step, which yields:
   $(t/t)/((t/t)\backslash(t\backslash t)), (t/t)\backslash((t/t)/((t/t)\backslash(t\backslash t))), (t/t)\backslash(t\backslash t) \not\Rightarrow t/t$
   This sequent cannot be further reduced, so t/t indeed cannot be derived.

   (b) For the sixth type, $((t/t)\backslash(t\backslash t))/((t/t)/((t/t)\backslash(t\backslash t)))$ to combine with $(t/t)\backslash((t/t)/((t/t)\backslash(t\backslash t)))$, the sequent between them must derive t/t. Since this sequent consists just of the type $(t/t)/((t/t)\backslash(t\backslash t))$, t/t cannot be derived.

□

Chapter 6

Miscellaneous

Most of the proofs from Kanazawa (1998) involving CCGs are independent of the actual formalism used. Some of these results have been adapted in a straightforward way to other domains, most notably to General Combinatory Grammars (GCGs). Similar techniques have been used in Tiede (1999b) to obtain learnability results for classes of Generalized Quantifiers. We will discuss both in this chapter,[1] and we will conclude with some negative results on learning classes of TAG.

6.1 General Combinatory Grammars

The formalism GCG employs additional combinatory rules in addition to the application rules of CCG.[2] Our main motivation for studying GCG is that this formalism is a (linguistically motivated) extension of the (weakly) context-free formalism CCG. Its expressive power is strictly greater than that of CCG; it was shown in Vijay-Shanker and Weir (1990, 1994) to be weakly equivalent to the mildly context-sensitive formalism known as linear indexed grammar (LIG), and in Weir and Joshi (1988) it was shown that Combinatory Categorial Grammars with directional categories, forward and backward application, and a generalized form of forward and backward function composition are weakly equivalent to TAGs. Indexed grammar was introduced in Aho (1968), see Gazdar (1988); Michaelis and Wartena (1997, 1998) for a discussion of variations on the IG-formalism and its relevance for the study of natural language.

We mention some of the numerous combinatory rules that have been proposed:

[1] Some of the results in this chapter have been previously published in Costa Florêncio (2001b, 2002c), reproduced with permission.

[2] For linguistic applications of this system see for example Steedman (1987), Szabolcsi (1987) and most recently Steedman (2000); Baldridge (2002)

Forward Composition:	$C/B, B/A$	$\Rightarrow$	C/A
Backward Composition:	$A\backslash B, B\backslash C$	$\Rightarrow$	$A\backslash C$
Forward Crossing Composition:	$C/B, A\backslash B$	$\Rightarrow$	$A\backslash C$
Backward Crossing Composition:	$B/A, B\backslash C$	$\Rightarrow$	C/A
(Forward) Lifting:	A	$\Rightarrow$	$B/(A\backslash B)$
(Backward) Lifting:	A	$\Rightarrow$	$(B/A)\backslash B$
Forward Substitution:	$(C/B)/A, B/A$	$\Rightarrow$	C/A
Backward Substitution:	$A\backslash B, A\backslash(B\backslash C)$	$\Rightarrow$	$A\backslash C$
Forward Crossing Substitution:	$C\backslash(A/B), C\backslash B$	$\Rightarrow$	$C\backslash A$
Backward Crossing Substitution:	$B/C, (B\backslash A)/C$	$\Rightarrow$	A/C

By extending the labeling scheme for functor-argument structures with the names of these rules, the notion of structure language can be extended. For example, we can use FS for 'Forward Substitution' etc.

6.1.1 Previous Results

This subsection reviews results from Kanazawa (1998) concerning GCGs.

Proposition 6.1 *The class of rigid grammars remains learnable from structures in the system of combinatory grammars.*

`Proof:` Let $\mathcal{S}$ be any finite set of function symbols. The set $\mathrm{Tp}_{\mathcal{S}}$ of *$\mathcal{S}$-types* is the set of terms constructed from $\mathrm{Pr} = \{t\} \cup \mathrm{Var}$ using function symbols from $\mathcal{S}$. For instance, if f and g are n-ary and m-ary function symbols in $\mathcal{S}$,

$$f(g(x_1, \ldots, x_m), y_2, \ldots, y_n) \in \mathrm{Tp}_{\mathcal{S}}.$$

The notions of substitution and unification apply directly to $\mathcal{S}$-types. Let Σ be any alphabet, then an *$\mathcal{S}$-grammar over* Σ is any finite relation between Σ and $\mathrm{Tp}_{\mathcal{S}}$.

An $\mathcal{S}$-rule is any expression of the form

$$A_1, \ldots, A_n \Rightarrow A_{n+1},$$

where $A_1, \ldots, A_n, A_{n+1} \in \mathrm{Tp}_{\mathcal{S}}$. Let $\mathcal{R}$ be any finite set of $\mathcal{S}$-rules. Then an $\mathcal{R}$-structure over Σ is a tree where each leaf node is labeled by some symbol in Σ and each internal node having n daughters is labeled by some rule $R = A_1, \ldots, A_n \Rightarrow A_{n+1} \in \mathcal{R}$. Formally, the set $\Sigma^{\mathcal{R}}$ or *$\mathcal{R}$-structures* over Σ is defined as follows:

1. $\Sigma \subseteq \Sigma^{\mathcal{R}}$.
2. If $T_1, \ldots, T_n \in \Sigma^{\mathcal{R}}$, and $R = A_1, \ldots, A_n \Rightarrow A_{n+1}$ is an $\mathcal{S}$-rule in $\mathcal{R}$, then

$$R(T_1, \ldots, T_n) \in \Sigma^{\mathcal{R}}$$

Here $R(T_1, \ldots, T_n)$ is a term representation of the tree that is the result of attaching $T_1, \ldots, T_n$ to a node labeled by r. The notion of $\mathcal{R}$-structure is a generalization of the notion of functor- argument structure.

An $\mathcal{R}$-derivation is a generalized notion of derivation. Formally, a tree $\mathcal{D}$ is an $\mathcal{R}$-derivation if and only if the following conditions hold:

1. Each node of $\mathcal{D}$ is labeled by an $\mathcal{S}$-type.

2. Each internal node of $\mathcal{D}$ is in addition labeled by an $\mathcal{S}$-rule in $\mathcal{R}$.

3. If an internal node v_{n+1} has n daughters $v_1, \ldots, v_n$, then $v_1, \ldots, v_n, v_{n+1}$ are labeled by some $\mathcal{S}$-types $B_1, \ldots, B_n, B_{n+1}$ and v_{n+1} is (in addition) labeled by an $\mathcal{S}$-rule $R = A_1, \ldots, A_n \Rightarrow A_{n+1} \in \mathcal{R}$ such that for some σ, $\sigma(A_1) = B_1, \ldots, v(A_{n+1}) = B_{n+1}$.

If $\mathcal{D}$ is a derivation whose root node is labeled by B and whose leaf nodes are labeled by $A_1, \ldots, A_n$, $\mathcal{D}$ is called an *$\mathcal{R}$-derivation of B from $A_1, \ldots, A_n$*.

If G is an $\mathcal{S}$-grammar, an *$\mathcal{R}$-parse tree* of G is obtained from an $\mathcal{R}$-derivation $\mathcal{D}$ of t from $A_1, \ldots, A_n$ by decorating the leaf nodes of $\mathcal{D}$ with symbols $c_1, \ldots, c_n$ in Σ such that G: $c_i \mapsto A_i$ for $1 \leq i \leq n$. *$G\mathcal{R}$-generates* an $\mathcal{R}$-structure T if T is the result of stripping an $\mathcal{R}$-parse tree of G of its type labels. The set of $\mathcal{R}$-structures $\mathcal{R}$-generated by G is called the *$\mathcal{R}$-structure language* of G and is denoted $\mathcal{FL}_{\mathcal{R}}(G)$.

If Σ, R, and $\mathcal{R}$ are a finite alphabet, a finite set of function symbols, and a finite set of $\mathcal{S}$-rules, respectively, then the triple $\langle \mathrm{CatG}_{\mathcal{S}}, \Sigma^{\mathcal{R}}, \mathrm{FL}_{\mathcal{R}} \rangle$ constitutes a grammar system. Such a grammar system is called a *system of general combinatory grammars*. Note that if $\mathcal{S} = \{\backslash, /\}$ and $\mathcal{R} = \{\mathrm{BA}, \mathrm{FA}\}$, where $\mathrm{BA} = x, x\backslash y \Rightarrow y$ and $\mathrm{FA} = y/x, x \Rightarrow y$, then the grammar system $\langle \mathrm{CatG}_{\mathcal{S}}, \Sigma^{\mathcal{R}}, \mathrm{FL}_{\mathcal{S}} \rangle$ is the familiar grammar system of classical categorial grammar.

Theorem 6.2 *In any system $\langle \mathrm{CatG}_{\mathcal{S}}, \Sigma^{\mathcal{R}}, \mathrm{FL}_{\mathcal{R}} \rangle$ of general combinatory grammars, the class of rigid $\mathcal{S}$-grammars is learnable.*

Lemma 6.3 *Let σ: $\mathrm{Var} \rightarrow \mathrm{Tp}_{\mathcal{S}}$ be a substitution. Then $\sigma[G_1] \subseteq G_2$ implies $\mathrm{FL}(G_1) \subseteq \mathrm{FL}(G_2)$.*

Definition 6.4 *Let D be a finite set of $\mathcal{R}$-structures. we construct $\mathcal{S}$-grammar $\mathrm{GF}_{\mathcal{S}(D)}$, the* general form determined by *D, by the following algorithm:*

1. Assign one of two types to each node of the structures in D as follows:

   (a) Assign t to each root node.

   (b) For each internal node v_{n+1} with daughters $v_1, \ldots, v_n$ that is labeled by an $\mathcal{S}$-rule $R = A_1, \ldots, A_n \Rightarrow A_{n+1} \in \mathcal{R}$, do the following:

i. If $x_1, \ldots, x_m$ are the variables that occur in R, let

$$\tau = \{x_1 \mapsto z_1, \ldots, x_m \mapsto z_m\},$$

where $z_1, \ldots, z_m$ are distinct *fresh* variables.

ii. Assign $\tau(A_1), \ldots, \tau(A_n), \tau(A_{n+1})$ to $v_1, \ldots, v_n, v_{n+1}$, respectively.

In this step, each leaf node in D is assigned one type, and each internal node in D is assigned two types.

2. Collect the types assigned to the leaf nodes in D into a grammar:

$$G = \{\langle c, A\rangle \mid A \text{ is assigned to a leaf node labeled by } c\}.$$

3. For each internal node v in D, let $A_{v,1}, A_{v,2}$ be the two types assigned to v. Let

$$\mathcal{A} = \{\{A_{v,1}, A_{v,2}\} \mid v \text{ is an internal node in } D\}.$$

and compute $\sigma = \mathrm{mgu}(\mathcal{A})$.

4. Let $\mathrm{GF}_{\mathcal{S}}(D) = \sigma[G]$, this is the output of the algorithm.

Lemma 6.5 *Let D be any finite set of $\mathcal{R}$-structures. Then for any $G \in \mathrm{CatG}_{\mathcal{S}}$, the following are equivalent:*

1. $D \subseteq \mathrm{FL}_{\mathcal{R}}(G)$.
2. $\mathrm{GF}_{\mathcal{S}}(D)$ *exists and there is a substitution σ such that* $\sigma[\mathrm{GF}_{\mathcal{S}}(D)] \subseteq G$.

Definition 6.6 *For a finite set D of $\mathcal{R}$-structures, we define $\mathrm{RG}_{\mathcal{S}}(D)$, the rigid $\mathcal{S}$- grammar determined by D, to be the output of the following algorithm:*

1. Compute $\mathrm{GF}_{\mathcal{S}}(D)$.
2. Let

$$\mathcal{A} = \{\{A \mid \mathrm{GF}_{\mathcal{S}}(D)\colon\ c \mapsto A\} \mid c \in \mathrm{dom}(\mathrm{GF}_{\mathcal{S}}(D))\}$$

and compute $\sigma = \mathrm{mgu}(\mathcal{A})$.

3. Let $\mathrm{RG}_{\mathcal{S}}(D) = \sigma[\mathrm{GF}_{\mathcal{S}}(D)]$.

Proposition 6.7 *Let D be any finite set of $\mathcal{R}$-structures. Then for any rigid $\mathcal{S}$-grammar G, the following are equivalent:*

1. $D \subseteq \mathrm{FL}_{\mathcal{S}}(G)$.
2. $\mathrm{RG}_{\mathcal{S}}(D)$ *exists and* $\mathrm{RG}_{\mathcal{S}}(D) \sqsubseteq G$.

Define the learning function $\varphi_{\mathrm{RG}_{\mathcal{S}}}$ for $\langle \mathrm{CatG}_{\mathcal{S}}, \Sigma^{\mathcal{R}}, \mathrm{FL}_{\mathcal{R}}\rangle$ by

$$\varphi_{\mathrm{RG}_{\mathcal{S}}}(\langle T_0, \ldots, T_i\rangle) = \mathrm{RG}_{\mathcal{S}}(\{T_0, \ldots, T_i\}).$$

Proposition 6.8 *The function $\varphi_{\mathrm{RG}_\mathcal{S}}$ has the following properties:*

1. *$\varphi_{\mathrm{RG}_\mathcal{S}}$ learns the class of rigid $\mathcal{S}$-grammars from $\mathcal{R}$-structures prudently.*
2. *$\varphi_{\mathrm{RG}_\mathcal{S}}$ is responsive and consistent on the class of rigid $\mathcal{S}$-grammars.*
3. *$\varphi_{\mathrm{RG}_\mathcal{S}}$ is set-driven.*
4. *$\varphi_{\mathrm{RG}_\mathcal{S}}$ is conservative.*
5. *$\varphi_{\mathrm{RG}_\mathcal{S}}$ is monotone increasing.*
6. *$\varphi_{\mathrm{RG}_\mathcal{S}}$ incremental.*
7. *$\varphi_{\mathrm{RG}_\mathcal{S}}$ can be implemented to run in linear time.*

6.1.2 Restricting Combinatory Rules for Finite Elasticity

The learnability from structures of the class of rigid grammars is easy to prove in any system of general combinatory grammars. Things are different when it comes to finite elasticity, however. The fact that application is the only rule used in the grammar system is crucial in Kanazawa's proof of finite elasticity for this class.

Since finite elasticity was essential in the proof of learnability of the class of k-valued classical categorial grammars from structures and from strings, these results do not generalize to learning rigid grammars from strings under an arbitrary set of combinatory rules. In fact it is quite easy to come up with a set of combinatory rules under which not even rigid grammars are learnable:

Example 6.9 *The class of rigid grammars using the combinatory rules*

$$X/X, X/X \Rightarrow X, Y/(X/X) \Rightarrow Y$$

has a limit point and is thus not (non-effectively) learnable from strings.[3]

`Proof:` For any $n \in \mathbb{N}^+$, Let

$$G_n: \begin{array}{lcl} \mathtt{c} & \mapsto & ((S/\underbrace{(Y/Y))/(X/X)}_{n \text{ times}} \\ \mathtt{a} & \mapsto & (X/X) \\ \mathtt{b} & \mapsto & (Y/Y) \end{array} \qquad G_*: \begin{array}{lcl} \mathtt{c} & \mapsto & S/(X/X) \\ \mathtt{a} & \mapsto & (X/X) \\ \mathtt{b} & \mapsto & (X/X) \end{array}$$

[3]Note that these two rules are axioms in the system $\mathbf{L}_\emptyset$ (see Chapters 8 and 9) and that the first rule is implied by Composition. The second rule is unlikely to ever be used by any syntactician in this unrestricted form, however. The set of rules mentioned at the beginning of Section 6.1 is more natural and also yields non-learnability results. We will not demonstrate this in detail but it easily follows from the fact that in this setting GCG is weakly equivalent to TAGs and there are strong non-learnability results for 'small' classes of TAGs, as will be discussed in Section 6.3.

It is easy to see that $\mathrm{L}(G_n) = c\{a^*, b^*\}^i, 0 \leq i \leq n$, so $\mathrm{L}(G_0) \subset \mathrm{L}(G_1) \subset \ldots$, thus the class has an infinite ascending chain. It is also obvious that $\mathrm{L}(G_*) = c\{a^*, b^*\}^* = \cup_{i \in \mathbb{N}} c\{a^*, b^*\}^i$, which is a limit point for the class.[4]. □

It should be clear that elasticity of subclasses of GCGs is important and non-trivial. This subsection provides some sufficient conditions for finite elasticity in the form of restrictions over combinatory rules.

Recall the combinatorial rules mentioned at the beginning of this section. A given grammar G interpreted under these rules can be compiled into a grammar G' interpreted under just (a generalized version of) the application rules without affecting the functor language. The naming function for this interpretation will be written as $\mathrm{FL}\_$.

Let the regular slashes be labeled with *fa* and *ba* (depending on their direction), and let the introduced slashes be indexed with labels for the rules they are compiled for. Combinatory rules are assumed to take the (binary) form of either $\mathbf{Functor}, \mathbf{Argument} \Rightarrow \mathbf{Result}$ or $\mathbf{Argument}, \mathbf{Functor} \Rightarrow \mathbf{Result}$[5], where **Argument** is not a variable type, their corresponding rewrite rules are $(\mathbf{Argument}|\ldots) \mapsto (n|\ldots), (\mathbf{Functor}|\ldots) \mapsto (\mathbf{Result}/_{label} n|\ldots)$[6] or $(\mathbf{Argument}|\ldots) \mapsto (n|\ldots), (\mathbf{Functor}|\ldots) \mapsto (n\backslash_{label}\mathbf{Result})|\ldots$, respectively.

The combinatory rules mentioned above result in the following rewrite rules which can be (recursively) applied to the types in a rigid grammar G, yielding *copies* of these types. Application terminates when none of these rules can be applied anymore. Assigning these copies to the same symbols as their originals were assigned to, combined with the assignments in G, yields G', which can be interpreted with just application:[7]

$$
\begin{array}{lcl}
\text{Forward Composition:} & \langle & \\
(B/A)|\ldots & \mapsto & n|\ldots, \\
(C/B)|\ldots & \mapsto & ((C/A)/_{fc} n)|\ldots\rangle \\
\text{Backward Composition:} & \langle & \\
(A\backslash B)|\ldots & \mapsto & n|\ldots, \\
(B\backslash C)|\ldots & \mapsto & (n\backslash_{bc}(A\backslash C))|\ldots\rangle \\
\text{Forward Substitution:} & \langle & \\
(B/A)|\ldots & \mapsto & n|\ldots, \\
((C/B)/A)|\ldots & \mapsto & (C/A)/_{fs} n|\ldots\rangle \\
\text{Backward Substitution:} & \langle & \\
(A\backslash B)|\ldots & \mapsto & n|\ldots, \\
(A\backslash(B\backslash C))|\ldots & \mapsto & (n\backslash_{bs}(A\backslash C))|\ldots\rangle
\end{array}
$$

[4] This proof was inspired by a (far more) elaborate proof of non-learnability of rigid Lambek grammars in Foret and Le Nir (2002a) which will be discussed in Chapter 9.

[5] As required by The Principle of Consistency, see (Steedman, 2000, page 54).

[6] Here $A|B$ denotes either A/B or $B\backslash A$, and $\ldots$ denotes the rest of the categorial type, which is assumed to be identical on both sides of $\mapsto$. Let n denote some freshly introduced (unique) primitive type.

[7] The operators $\backslash$ and $/$ are now shorthand for $\backslash_{ba}$ and $/_{fa}$, respectively.

Let *application degree* be defined as in Definition 3.9, restricted to slashes with index *fa* or *ba* and other slashes not counted.

Definition 6.10 *A* conservative combinatory rule (CCR) *is a combinatory rule in which the resulting type has a degree that is lower than the degree of the functor type and does not introduce new variable types.*

A conservative tuple of rewrite rules (CTRR) *is a tuple of rewrite rules corresponding to a conservative combinatory rule.*

Note that our restrictions disallow composition, but it is not clear at the present time whether the class of structure languages generated by rigid grammars with just application and composition has finite elasticity.

The first rule in a CTRR rewrites the argument type, its result type is always of lower degree than the argument type. The result type of the second rule in a CTRR has an application degree that is lower than the application degree of the functor type: an extra slash in the result type is never labeled with *fa* or *ba*, thus this slash does not add to the application degree of this type. We will slightly abuse notation and let $ctrr[G]$ denote the grammar obtained by applying some CTRRs to grammar G. Then the following should be obvious:

Proposition 6.11 *For any finite grammar G, $G' = ctrr[G]$ is finite.*

`Proof:` (sketch) Let R be the set of CTRRs corresponding to a given set of CCRs. It is clear that all rewrite rules in R can only be applied a finite number of times to any grammar G, since all types in G have a finite degree and these rules reduce the degree of the types they rewrite. Thus only a finite number of new types can be assigned in G', the rewritten version of G interpreted under generalized application. □

The following defines the relation dcom, and, implicitly, structure language for CCG. Note that the application labels are included in the alphabet:

Definition 6.12 *Let* dcom*: $\Upsilon^{\mathrm{F}} \rightarrow \Sigma^{\mathrm{F}}$ be the homomorphism that maps each indexed application label to a non-indexed application label:*

$$\begin{aligned} \mathrm{dcom}(c) &= c, \\ \mathrm{dcom}(\mathrm{ba}(label, S_1, S_2)) &= \mathrm{ba}(\mathrm{dcom}(S_1), \mathrm{dcom}(S_2)), \\ \mathrm{dcom}(\mathrm{fa}(label, S_2, S_1)) &= \mathrm{fa}(\mathrm{dcom}(S_2), \mathrm{dcom}(S_1)), \end{aligned}$$

for all $c \in \Sigma$, and $label, S_1, S_2 \in \Upsilon^{\mathrm{F}}$.

The following proposition states that, under the proper interpretation, $ctrr[G]$ is strongly equivalent to G:

Proposition 6.13 *Let G be a grammar under the set of CCRs R. Then $\mathcal{FL}_{\mathcal{R}}(G) = \mathrm{FL}_{\_}(ctrr[G])$, where ctrr denotes application of the CTRRs corresponding to $\bar{R}$.*

Lemma 6.14 *Let G be a rigid grammar under the set of CCRs R. Then there is a k-valued grammar G'' such that* $\mathrm{FL}(G'') = \mathrm{dcom}(\mathrm{FL}_{\_}(ctrr[G]))$*, where ctrr denotes application of the CTRRs corresponding to R.*

The value of k depends on both R and (the degree of) the types in G (which is finite but not bounded). Clearly dcom defines a finite-valued relation.

Lemma 6.15 *Kanazawa (1998)* $\mathcal{FL}_{k\text{-valued}}$ *has finite elasticity.*

Theorem 6.16 *The class $\mathcal{G}$ of rigid* CCG*s interpreted under any set of CCRs has finite elasticity.*

`Proof:` (sketch) Let R be the set of CTRRs corresponding to a given set of CCRs. By Lemma 6.14, these rewrite rules constitute a finite-valued relation between $\mathcal{FL}_{\mathrm{rigid}}$ and $\mathcal{FL}_{k\text{-valued}}$, for some constant k. Using Theorem 2.27 and Lemma 6.15, it can be shown that under any set of conservative combinatory rules the class of rigid grammars has finite elasticity. □

It follows directly that $\mathcal{FL}_{\mathcal{R}\,k\text{-valued}}$, $\mathcal{L}_{\mathcal{R}\,\mathrm{rigid}}$ and $\mathcal{L}_{\mathcal{R}\,k\text{-valued}}$, under CCRs, also have finite elasticity.

Restricting grammar systems to CCRs seems to severely limit their expressive power, it precludes Lifting, for example. However, Lifting is used in CCG under the restriction that the resulting range type is a parametrically licensed category for the language. This is assumed to restrict it to a finite set of categories (see Steedman (2000), page 44).[8] Thus the Lifting rules are actually shorthand for (finite) lists of rules in which A and T are replaced by types that only contain primitive types. Adding such lists to a collection of CCRs will probably not affect finite elasticity.

What is *not* allowed by our restrictions are rules of the form $C/B, B/A \Rightarrow A/C$, since type A is moved from a domain to a range position, so that the degree of the resulting type may be higher than that of the argument or functor type. This type of rule does not seem to be linguistically plausible and would violate polarity, so this does not really pose a problem.

The rich formalism of Combinatory Categorial Grammar has natural subclasses that have the pleasant property of finite elasticity. This implies that there exist algorithms that learn these classes while conforming to consistency and conservativity constraints on their behavior prior to convergence. It also implies that union with other classes with finite elasticity yields classes with the same property.

6.2 Learning Generalized Quantifiers

This section will address the issue of learning generalized quantifiers. This topic was first taken up in van Benthem (1986a), and, although it has been

[8] Restrictions on Lifting were already proposed in Lambek (1958): only primitive types could be lifted, with a lexical type as argument.

mentioned since then (cf Keenan (1996)), it has received little attention.

Gold's paradigm was intended as a model of acquisition of syntax, so most of the applications of formal learning theory to linguistics have been restricted to syntax. It seems only natural to apply the paradigm to semantics, which certainly is a non-trivial enterprise.[9] Naive associationist theories for example have a hard time dealing with quantifiers, since it may not be clear what quantified expressions refer to.

It is also a challenge from a formal viewpoint, since the class of all generalized quantifiers is superfinite and thus not learnable. Therefore (nontrivial) learnable subclasses should be investigated, and fortunately formal learning theory provides some tools for doing this. Conceptually these subclasses correspond to constraints on quantifiers, so this kind of research may in the future provide motivation for new or existing 'natural' constraints like, for example, conservativity.

In Clark (1996) the problem of learning first-order generalized quantifiers with a *minimally adequate teacher* was considered. In this paradigm the learner is not only provided with positive data but also has access to an oracle that answers *membership queries* and gives *counterexamples* in response to the learner's incorrect conjectures. In van Benthem (1986b) first-order generalized quantifiers were interpreted as regular sets, and it was shown that the associated formalism of finite automata yields natural interpretations for these quantifiers. In Angluin (1987) an algorithm was given that learns regular sets under such conditions.

The model from Clark (1996) assumes that in the real world, parents somehow have access to the conjecture of a child, which seems to be an unrealistic abstraction. However, evidence from Brown and Hanlon (1970) suggests that the learner has access to some negative data in the case of semantic learning.

In Tiede (1999b) a different approach was taken, obtaining some stronger results:

Proposition 6.17 *Tiede (1999b) Define left upward monotonicity as* $\uparrow$MON : QAB *and* $A \subseteq A' \Rightarrow QA'B$. *The set of left upward monotone quantifiers is in* **Lang**.

This confirms the conjecture from Barwise and Cooper (1981) that monotonicity of quantifiers facilitates learning. Note that nothing is said about *effective* learnability. We will now refine this result and see under what other natural restrictions learnability of this class is preserved.

6.2.1 Beyond First Order Generalized Quantifiers

Let a (binary) generalized quantifier, Q, on a domain E be a (binary) relation between subsets of E, i.e. $Q_E \subseteq \wp(E \times E)$. Restricting our attention to finite

[9] Some recent interesting learnability results concerming learning and semantic knowledge can be found in Stephan and Ventsov (2001), this work falls outside the scope of the present discussion.

structures results in the representation in the tree of numbers:

$$\begin{array}{ccccccc} & & & (0,0) & & & \\ & & (1,0) & & (0,1) & & \\ & (2,0) & & (1,1) & & (0,2) & \\ (3,0) & & (2,1) & & (1,2) & & (0,3) \\ & & & \vdots & & & \end{array}$$

Here each pair (a, b) is such that $a = |A - B|, b = |A \cap B|$. Every quantifier is thus a subset of $\mathbb{N} \times \mathbb{N}$. Thus, quantifiers can be identified with formal languages: given $Q \subseteq \mathbb{N}^2, (n, m) \in \mathbb{N}^2$ can be represented as $\{w \in \{a, b\}^* | w$ contains n a's and m b's$\}$.

It was shown in van Benthem (1986b) that every first order definable quantifier is accepted by an acyclic, permutation-invariant finite automaton. Another result by van Benthem states that the quantifiers that can be computed by pushdown- automata are exactly those that can be defined in Presburger arithmetic. In Ginsburg and Spanier (1966) it was shown that sets and relations definable in Presburger arithmetic are precisely the semi-linear sets. These results allow us to confine ourselves to the numeric representation of quantifiers.

Theorem 6.18 *Abe (1989) Every linear set $A \subseteq \mathbb{N}^2$ is equivalent to a finite union of linear sets, each of which is generated by at most 3 elements, that is a set of the form* $\left\{ v_0 + \sum_{i=1}^{2} m_i \times v_i \mid m_i \in \mathbb{N} \right\}$.

Theorem 6.19 *Tiede (1999b) In the above normal form we can take m_1 and m_2 to be nonzero.*

Proposition 6.20 *Tiede (1999b) The set*

$$\mathcal{B} = \left\{ \left\{ v_0 + \sum_{i=1}^{2} m_i \times v_i \mid m_1, m_2 > 0 \right\} \mid v_0, v_1, v_2 \in \mathbb{N}^2 \right\}$$

has finite thickness.

Definition 6.21 *Tiede (1999b) Let*

$$\begin{array}{lll} \mathcal{B}_0 & =_{def} & \mathcal{B}, \\ \mathcal{B}_{n+1} & =_{def} & \mathcal{B}_0 \tilde{\cup} \mathcal{B}_n = \{b \cup b' \mid b \in \mathcal{B}_n, b' \in \mathcal{B}\}. \end{array}$$

Corollary 6.22 *Tiede (1999b) For any n, $\mathcal{B}_n$ has finite thickness and is therefore learnable.*

The first refinement of this result will be a complexity analysis. The notion of *intrinsic complexity* is based on reductions between classes of languages,

which define complexity hierarchies. Such a reduction is defined as a transformation for input sequences for a language from class $\mathcal{L}_1$ to sequences for a language from $\mathcal{L}_2$, and a transformation for acceptable output sequences for $\mathcal{L}_2$ to acceptable output sequences for $\mathcal{L}_1$. Weak reductions are defined with an initial transformation that may yield the same sequence given two different sequences.

The following is from Jain and Sharma (1996b) (Theorem 3):

Theorem 6.23 *Suppose $\mathcal{L}$ has finite thickness. Then* FIN $\not\leq^{\mathbf{TxtEx}}_{\text{weak}}$ $\mathcal{L}$.

Thus it is a simple corollary that for any n, FIN $\not\leq^{\mathbf{TxtEx}}_{\text{weak}}$ $\mathcal{B}_n$.

Using techniques borrowed from Kanazawa (1998) and Shinohara (1994) based on length-bounded elementary formal systems a hierarchy was defined of learnable subclasses of quantifiers definable in Presburger arithmetic. Each of these classes has finite thickness.

Definition 6.24 *Tiede (1999b) Let $\mathbb{B}$ be the set of all formulas of the form*

$$\begin{aligned}\exists x_1 \exists x_2 \quad (y_1 = \underline{m_0} \underline{+} (\underline{m_1} \underline{\times} x_1) \underline{+} (\underline{m_2} \underline{\times} x_2) \wedge \\ y_2 = \underline{n_0} \underline{+} (\underline{n_1} \underline{\times} x_1) \underline{+} (\underline{n_2} \underline{\times} x_2) \wedge \\ x_1 \neq \underline{0} \wedge x_2 \neq \underline{0}).\end{aligned}$$

Then

$$\begin{aligned}\mathbb{B}_0 &=_{def} \mathbb{B}, \\ \mathbb{B}_{n+1} &=_{def} \{\varphi \vee \psi \mid \varphi \in \mathbb{B}_n, \psi \in \mathbb{B}\}.\end{aligned}$$

Lemma 6.25 *Tiede (1999b) The set of quantifiers definable in Presburger arithmetic by formulas of the form $\bigvee_{i=1}^{k} \varphi_i(y_1, y_2)$ with $\varphi_i(y_1, y_2) \in \mathbb{B}$ is identifiable for all k.*

Obviously these classes are highly expressive. Wright's and Shinohara's results apply directly, yielding the following:

Theorem 6.26 *The set of quantifiers definable in Presburger arithmetic by formulas of the form $\bigvee_{i=1}^{k} \varphi_i(y_1, y_2)$ with $\varphi_i(y_1, y_2) \in \mathbb{B}$ is effectively identifiable by means of a consistent and conservative learning function for all k.*

I will leave open for now the question of polynomial update time. Note however that a sufficient condition for the non-restrictiveness of this constraint would be that generating an index for a minimal[10] language given a finite input (ψ from Condition 2.59) takes only polynomial time (obviously membership testing is in P). See Shinohara (1986) for details.

Recall that learnable families of infinite languages are always learnable by a set-driven learner (Chapter 2). This leads us to conclude the following:

[10] With respect to the class to be learned.

Theorem 6.27 *The set of quantifiers definable in Presburger arithmetic by formulas of the form* $\bigvee_{i=1}^{k} \varphi_i(y_1, y_2)$ *with* $\varphi_i(y_1, y_2) \in \mathbb{B}$ *minus the set of all quantifiers associated with a finite language is effectively identifiable by means of a set-driven learning function for all* k*.*

It is not obvious whether leaving out these quantifiers is strictly necessary for set-driven learning, but it seems natural to exclude them.

Recall the relation between set-driven and conservative learning functions as mentioned in Section 2.6. We have an easy corollary of Theorem 6.27:

Corollary 6.28 *The set of quantifiers definable in Presburger arithmetic by formulas of the form* $\bigvee_{i=1}^{k} \varphi_i(y_1, y_2)$ *with* $\varphi_i(y_1, y_2) \in \mathbb{B}$ *minus the set of all quantifiers associated with a finite language is effectively identifiable by means of a conservative learning function with linear memory for all* k*.*

One interesting restriction that has not yet been applied is that of *ordinal mind change complexity*:

Theorem 6.29 *Ambainis et al. (1997) Let* $\mathcal{L}'$ *be an indexed family of recursive languages with finite elasticity, assume that* $\mathcal{L}$ *is learnable by a conservative function with respect to hypothesis space* $\mathcal{L}'$*. Then* $\mathcal{L} \in \mathbf{TxtEx}_\alpha$ *with respect to* $\mathcal{L}'$*, for some constructive ordinal* α*.*

Corollary 6.30 *The set of quantifiers definable in Presburger arithmetic by formulas of the form* $\bigvee_{i=1}^{k} \varphi_i(y_1, y_2)$ *with* $\varphi_i(y_1, y_2) \in \mathbb{B}$ *is in* $\mathbf{TxtEx}_\alpha$*, for some constructive ordinal* α*.*

6.3 Tree Adjoining Grammars

In this section we will discuss a few new learnability results for classes of Tree Adjoining Grammars (TAGs). This formalism was introduced in Joshi et al. (1975). We will not go into too much detail, see the survey Joshi and Schabes (1996) for motivation, formal results, algorithmic aspects and the like.

Definition 6.31 *A tree-adjoining grammar (TAG) is a quintuple* $\langle \Sigma, NT, I, A, S \rangle$*, where*

1. Σ *is a finite set of terminal symbols;*
2. NT *is a finite set of non-terminal symbols:* $\Sigma \cap NT = \emptyset$*;*
3. S *is a distinguished non-terminal symbol:* $S \in NT$*;*
4. I *is a finite set of finite trees, called* initial trees*, characterized as follows:*

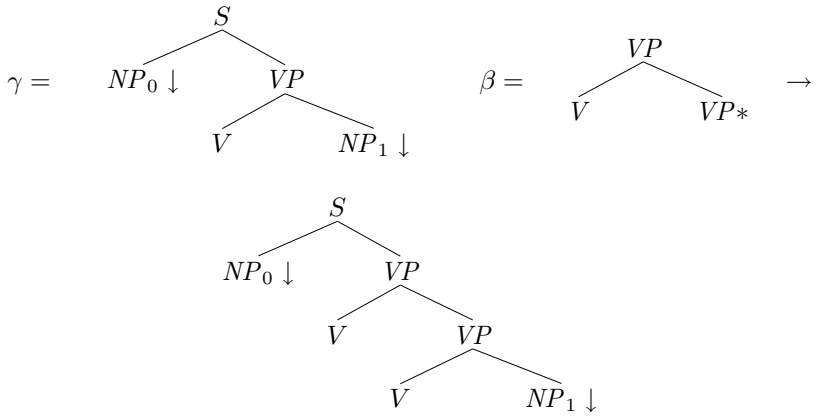


Figure 6.1: Adjoining.

- *interior nodes are labeled by non-terminal symbols;*
- *the nodes on the frontier of initial trees are labeled by terminals or non-terminals; non-terminal symbols on the frontier of the trees in I are marked for substitution; by convention, we annotate nodes to be substituted with ↓;*

5. *A is a finite set of finite trees, called* auxiliary trees*, characterized as follows:*

   - *interior nodes are labeled by non-terminal symbols;*
   - *the nodes on the frontier of auxiliary trees are labeled by terminals or non-terminals; non-terminal symbols on the frontier of the trees in A are marked for substitution except for one node, called the* foot node*; by convention, we annotate the foot node with ∗; the label of the foot node must be identical to the label of the root node.*

In TAG two composition operations are used: *adjoining* (Figure 6.1) and *substitution* (Figure 6.2). Constraints on adjoining can be specified, see Joshi (1987).

It has been shown in Vijay-Shanker (1987) that the class of Tree Adjoining Languages (TALs) is weakly equivalent to the class of languages generated by Linear Indexed Grammars (LIGS) or IG(1)s, that is IGs whose rules limit the inheritance and manipulation of index sequences to a single nonterminal daughter on the right side of the rule.

Note that the tree obtained by derivation from a TAG, the *derived tree* does not give enough information to determine how it was constructed. The *derivation tree* is an object that specifies how a derived tree was constructed. There

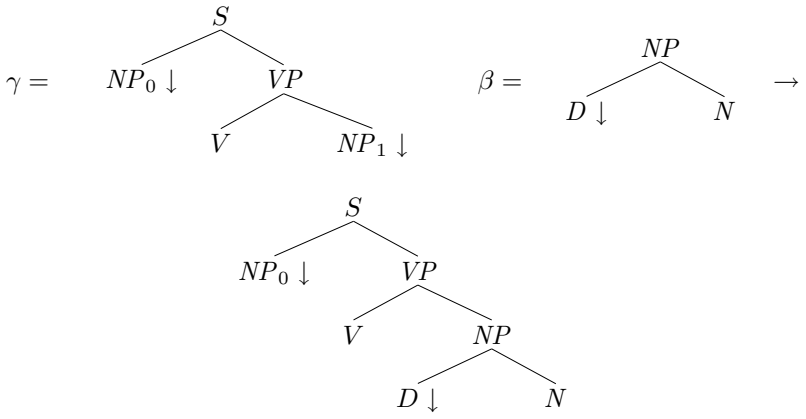


Figure 6.2: Substitution.

are (obvious) relations between the two, however: for example, the derived tree grows monotonically with each step in the derivation tree.

There are some variants of TAG about and they are often presented in an informal fashion. Since we want to analyze learnability in a formal way, we will use one particular precise definition from Vijay-Shanker and Weir (1994), the expressive power of which is known exactly. It is also a more or less restricted version, making negative results more general. It is restricted in the sense that it is (strongly) lexicalized,[11] and it does not use substitution, since this operation can be emulated with adjoining provided ε is allowed as leaf.[12]

Tree Adjoining Grammars manipulate trees containing only nodes that are labeled either by terminal symbols or by triples of the form $\langle A, \text{OA}, \text{NA}\rangle$, where A is a nonterminal symbol, SA is the set of tree labels (that determines which of the trees of the grammar can be adjoined at that node), and OA is either `true` (indicating that adjunction is obligatory) or `false` (indicating that adjunction is optional). We call SA and OA the adjunction constraints at that node. A node at which the value of $\text{SA} = \emptyset$ is said to have an NA constraint.

Let V_N be a set of nonterminal symbols, let V_T be a set of terminal symbols, let V_L be a set of tree labels, and let $V_\text{T}^\varepsilon = V_\text{T} \cup \{\varepsilon\}$.

Initial Trees. For each $A \in V_\text{N}$, $init(V_\text{N}, V_\text{T}, V_\text{L}, A)$ is the set of trees $\alpha : D_\alpha \to V_\text{T}^\varepsilon \cup (V_\text{N} \times 2^{V_\text{L}} \times \{\texttt{true}, \texttt{false}\})$, where D_α is a tree domain and the following hold:

[11] In *lexicalized TAG* (LTAG) at least one terminal symbol (the *anchor*) appears at the frontier of all initial and auxiliary trees, this restricts the weak expressive power to *finitely ambiguous* TALs.

[12] A normal form for TAG is also presented in the same paper, which relies heavily on the use of ε.

- The root of α is labeled $\langle A, sa, oa\rangle$ for some $sa \subseteq V_{\mathrm{L}}$ and $oa \in \{\texttt{true}, \texttt{false}\}$.
- All internal nodes of α are labeled $\langle B, sa, oa\rangle$ for some $B \in V_{\mathrm{N}}$, $sa \subseteq V_{\mathrm{L}}$, and $oa \in \{\texttt{true}, \texttt{false}\}$.
- All leaf nodes of α are labeled by some $u \in V_{\mathrm{T}}^{\varepsilon}$.

Auxiliary Trees. For each $A \in V_{\mathrm{N}}$, $aux(V_{\mathrm{N}}, V_{\mathrm{T}}, V_{\mathrm{L}}, A)$ is the set of trees $\beta : D_\beta \to V_{\mathrm{T}}^{\varepsilon} \cup (V_{\mathrm{N}} \times 2^{V_{\mathrm{L}}} \times \{\texttt{true}, \texttt{false}\})$, where D_β is a tree domain and the following hold:

- The root of β is labeled $\langle A, sa, oa\rangle$ for some $sa \subseteq V_{\mathrm{L}}$ and $oa \in \{\texttt{true}, \texttt{false}\}$.
- All internal nodes of β are labeled $\langle B, sa, oa\rangle$ for some $B \in V_{\mathrm{N}}$, $sa \subseteq V_{\mathrm{L}}$, and $oa \in \{\texttt{true}, \texttt{false}\}$.
- All leaf nodes of β except one are labeled by some $u \in V_{\mathrm{T}}^{\varepsilon}$. The remaining leaf node is called the foot node and is labeled $\langle A, sa, oa\rangle$ for some $sa \subseteq V_{\mathrm{L}}$ and $oa \in \{\texttt{true}, \texttt{false}\}$. The address of the foot node of β is denoted $ft(\beta)$.

Let $aux(V_{\mathrm{N}}, V_{\mathrm{T}}, V_{\mathrm{L}}) = \bigcup_{A \in V_{\mathrm{N}}} aux(V_{\mathrm{N}}, V_{\mathrm{T}}, V_{\mathrm{L}}, A)$.
Elementary Trees. $elem(V_{\mathrm{N}}, V_{\mathrm{T}}, V_{\mathrm{L}}, A) = init(V_{\mathrm{N}}, V_{\mathrm{T}}, V_{\mathrm{L}}, A) \cup aux(V_{\mathrm{N}}, V_{\mathrm{T}}, V_{\mathrm{L}})$.
Tree adjunction:

$$\nabla : elem(V_{\mathrm{N}}, V_{\mathrm{T}}, V_{\mathrm{L}}, A) \times aux(V_{\mathrm{N}}, V_{\mathrm{T}}, V_{\mathrm{L}}) \times \mathcal{N}_{+}^{*} \to elem(V_{\mathrm{N}}, V_{\mathrm{T}}, V_{\mathrm{L}}, A)$$

for every $A \in V_{\mathrm{N}}$.

6.3.1 TAGs with the Empty String

Since, as noted, the empty string is commonly used when working with TAGs, it is necessary to discuss its effect on learnability. For most formal systems (CCG, CFG, CSG, ...) it is quite easy to show that even very restricted learnable classes that exclude ε from the alphabet become non-learnable as soon as it is included. For TAGs this is slightly less obvious, so we will demonstrate it explicitly.

Consider the grammars G_n in Figure 6.3 and grammar G_* in Figure 6.4. It is clear that $\mathrm{yield}(G_n) = \mathtt{a}^{i+1}, 0 \leq i \leq n$ and $\mathrm{yield}(G_*) = \mathtt{a}^{+}$. (Note that $|V_{\mathrm{T}}^{\varepsilon}| = 2$ (since $V_{\mathrm{T}}^{\varepsilon} = \{a, \varepsilon\}$).) Thus for all $i \in \mathbb{N}, \mathrm{yield}(G_i) \subset \mathrm{yield}(G_{i+1})$, which constitutes an infinite ascending chain, and $\mathrm{yield}(G_*) = \cup_{i \in \mathbb{N}} \mathrm{yield}(G_i)$, which constitutes a limit point for this class.

Therefore, from this point on we will let V_{T}, not $V_{\mathrm{T}}^{\varepsilon}$, denote the set of nonterminals.

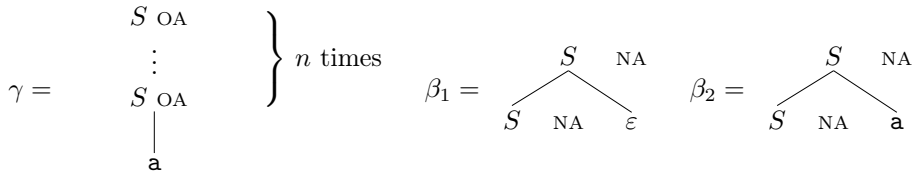


Figure 6.3: Grammar G_n.

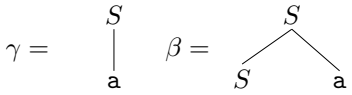


Figure 6.4: Grammar G_*.

6.3.2 The Class of Rigid TAGs is Not Learnable

At first glance there are two obvious options for defining rigidity for TAGs; any $s \in V_\mathrm{T}$ occurs at most once as leaf in $elem(G)$ for any given $G \in \mathcal{G}_\mathrm{rigid}$, or alternatively, any $s \in V_\mathrm{T}$ occurs at most once as leaf in $init(G)$ for any given $G \in \mathcal{G}^\flat_\mathrm{rigid}$. Note that the latter is more permissive than the former, ie, $\mathcal{L}_\mathrm{rigid} \subset \mathcal{L}^\flat_\mathrm{rigid}$. The latter definition allows the existence of a limit point, which is easy to show; consider grammars G_n and G_* as shown in Figure 6.5 and Figure 6.6, respectively.

Obviously, $\mathrm{yield}(G_n) = \mathtt{a}(\mathtt{ba}^{j_i})^*, 1 \leq j_i \leq n, i \in \mathbb{N}$, thus for all $i \in \mathbb{N}, \mathrm{yield}(G_i) \subset \mathrm{yield}(G_{i+1})$, constituting an infinite ascending chain. It is clear that $\mathrm{yield}(G_*) = \mathtt{a}(\mathtt{ba}^+)^*$, which is equivalent to $\cup_{i\in\mathbb{N}}\mathrm{yield}(G_i)$, so G_* is the index for a limit point for this class, even with the restriction $|V_\mathrm{T}| = 2$.

Note that in this particular case even the class of derived tree languages for which these grammars are indices has a limit point.

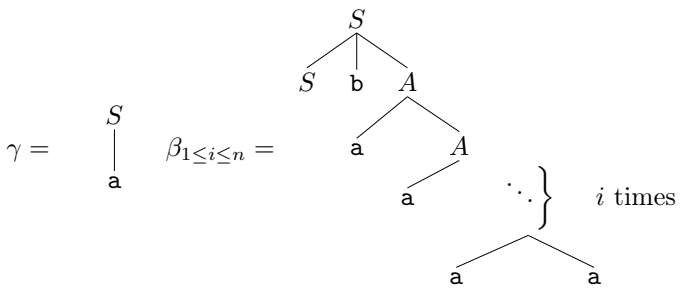


Figure 6.5: Grammar G_n.

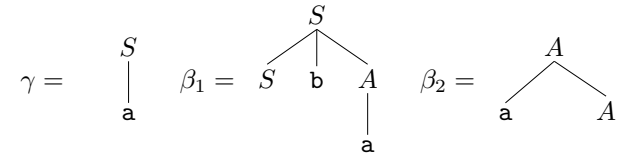


Figure 6.6: Grammar G_*.

Given the stricter, notion of rigidity for TAGs, it is possible to show that even under this definition the class of rigid TAGs allows a limit point.

Consider the set of grammars G_n defined in Figure 6.7. Any of the internal nodes of initial tree γ allows for adjunction with just the auxiliary tree β. Since foot and header of this tree are labeled NA, such an adjunction can only take place once for every internal node A, and since these internal nodes are not labeled OA, such an adjunction does not *have* to take place in order to derive a sentence. Thus, for a grammar G_n, the number of adjunctions i for any derivation is restricted by $0 \leq i \leq n$. It is easy to see that this implies $\text{yield}(G_n) = \cup_{0 \leq i \leq n} \mathtt{a}^i\mathtt{b}$, thus for all $i \in \mathbb{N}, \text{yield}(G_i) \subset \text{yield}(G_{i+1})$, therefore this collection of grammars constitutes an infinite ascending chain.

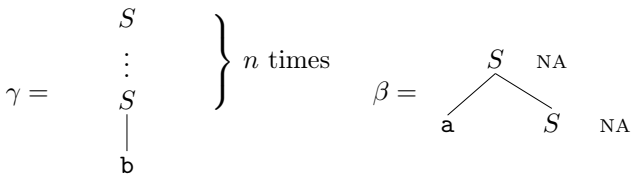


Figure 6.7: Grammar G_n.

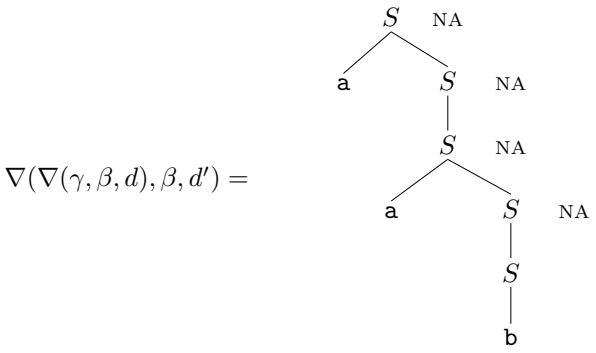


Figure 6.8: Example derived tree for aab using grammar G_3.

In order to show existence of a limit point we now need to define a grammar G_* such that $\text{yield}(G_*) = \cup_{i\in\mathbb{N}} G_i$. Such a grammar is shown in Figure 6.9, and differs from any G_n in that its one initial tree has no internal nodes, and its single auxiliary tree β has no NA restrictions on its nodes. It is obvious that an unbounded number of adjunctions can take place during a derivation, and any of these (optional) adjunctions adds a symbol a as leaf of the derived tree, somewhere to the left of the rightmost leaf b. Thus $\text{yield}(G_*) = \cup_{i\in\mathbb{N}} \mathtt{a}^i\mathtt{b} = \cup_{i\in\mathbb{N}} \text{yield}(G_i)$ and is therefore a limit point for the class.

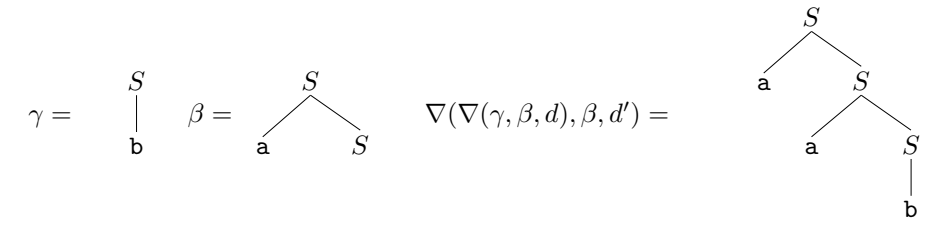


Figure 6.9: Grammar G_* with example derived tree.

Theorem 6.32 *The class of rigid Tree Adjoining Grammars with an alphabet of two symbols neither of which is the empty string, allowing unary and binary branchings, and allowing labeling with* NA*, has a limit point and is therefore not (non-effectively) learnable.*

6.4 Minimalist Grammars

The Minimalist approach to syntax was introduced in Chomsky (1992, 1995, 1998) and is seen as a continuation of the tradition of transformational grammar. We will not go deeply into the motivation of this approach here,the interested reader is referred to this literature.

Some formalisations have been proposed for Minimalist Grammar (MG), see e.g. Stabler (1997, 1998); Retoré and Stabler (1999); Stabler (2001), see Lecomte and Retoré (2001); Cornell (1999) for connections with multimodal logic, and see Stabler and Keenan (2003) for a succint reformulation.

MG has been shown independently to be weakly equivalent to linear context-free rewriting systems, and hence to MCTAGS, in Harkema (2001); Michaelis (2001), also see Michaelis (1998).[13]

[13]Ed Stabler reports results for learning rigid Minimalist Grammars from structures (Stabler (2002), also presented at the ESSLLI 2002 workshop "Learning Algorithms for Lexicalized Grammars", Trento, Italy). This paper is still in the manuscript stage at the time of writing, so we will not discuss it here.

In the Minimalist grammars formalism languages are defined as closures of a (finite) lexicon under some structure building operations, where the lexical elements consist of a (finite) sequence of features of the following form:

Features		Examples
`f`	categories	(`n`, `v`, ...)
`=f`	selectors	(`=n`, `=v`, ...)
`+f`	licensors	(`+case`, `+wh`, ...)
`-f`	licensees	(`-case`, `-wh`, ...)
`g`	non-syntactic	('`some`', '`every`', ...)

The two structure building operations are known as *merge* and *move*. The operation *merge* applies to a pair of trees when the head of the first expression is labeled with a sequence that starts with a selection feature `=f` and the head of the second expression is labeled with a sequence that starts with a category feature `f`. Both these features are removed.

When the first argument is a lexical expression (i.e. consists of just one node) the second tree is attached to its right as a *complement*. When the first argument is complex (i.e. is not lexical) the second argument is attached to its left as its *specifier*.

The second structure building operation *move* applies to a single tree, the head of which is labeled with a sequence beginning with some licensor feature `+f`, and that also contains exactly one leaf that is labeled with a sequence beginning with licensee feature `-f`. It moves the maximal projection (a maximal subtree with a given head) of the `-f` head to the specifier position of the head of the expression.

These two partial functions are formally defined as follows:

1. $merge : (exp \times exp) \rightarrow exp$:

$$merge(t_1[\texttt{=c}], t_2 c[\texttt{c}]) = \begin{cases} \overset{<}{\underset{t_1 \quad t_2}{\wedge}} & \text{if } t_1 \in Lex \\ \overset{>}{\underset{t_1 \quad t_2}{\wedge}} & \text{otherwise} \end{cases}$$

   where $t[f]$ is the result of prefixing feature f to the sequence of features at the head of t, and for all trees t_1, t_2, all `c` are categories.

2. $move : exp \rightarrow exp$:

$$move(t_1[\texttt{+f}]) = \overset{>}{\underset{t_2^{>} \quad t_1}{\wedge}} \quad \{t_2[\texttt{-f}]^{>}/\lambda\}$$

where $t^{>}$ is the maximal projection of t, and $t_1[\texttt{+f}]$ is any tree which contains exactly one node with first feature `-f`.

6.4.1 The Class of 2-Valued MGs with Empty Categories is Not Learnable

MG grammars exist for fragments of a number of natural languages. Generally speaking such fragments rely heavily on the use of phonologically empty categories, for example for complementizers (`=t +wh c`) or for case (`=acc +case t`).[14]

As we have seen in the last paragraph, grammar systems that allow inclusion of ε in their lexicons generally speaking have bad learnability properties. Since phonologically empty categories seem essential for the use of MG in linguistics, one might conjecture that classes of MGs restricted only by numerical bounds on their complexity are not learnable. This conjecture is valid, as we shall demonstrate by constructing a limit point for the class of 2-valued MGs that allow phonologically empty categories only if they contain a licensor feature.

Note that

1. 2-valued is defined here somewhat arbitrarily as 'any phonological feature occurs at most twice in the lexicon'. As in the case of TAGs, slightly different definitions are possible, but it is felt that any reasonable definition would allow this construction,

2. the lexicon contains one phonologically empty category that *doesn't* contain a licensor feature, but this is the special case where the category is the start-category `c`. Obviously this category is present in *any* MG that generates a non-empty language, and can be safely ignored,

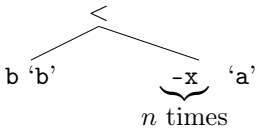
3. our construction requires that there is no bound on the number of licensee features assigned to any one category.

Definition 6.33 *For any $n \in \mathbb{N}$, let G_n be the following grammar:*

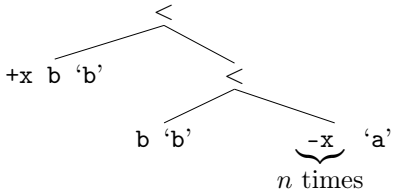
(complementizer, phonologically empty)	$\mapsto$	`=b`	`c`	
(complementizer, phonologically empty)	$\mapsto$	`=b`	`+x b`	
a	$\mapsto$	`a`	n times `-x`	'a'
b	$\mapsto$	`=a`	`b`	'b'
b	$\mapsto$	`=b`	`+x b`	'b'

It is easy to see that $\mathrm{L}(G_n) = \{\texttt{ab}^i \mid 1 \leq i \leq n+1, i \in \mathbb{N}^+\}$. We will not prove this rigourously, but it will be made clear by the following derivation. First, by combining lexical item `b` with `a` we obtain:

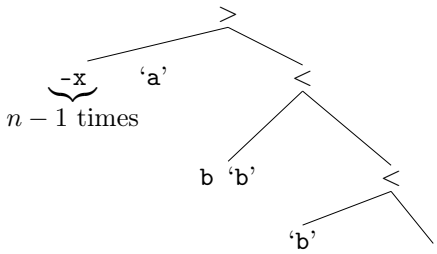
[14] Note that this kind of category seems to always contain licensor features.



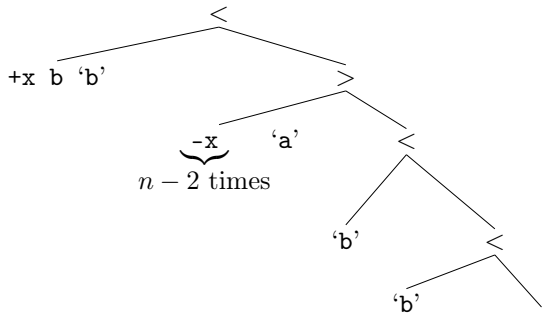
This tree can only be selected by a lexical item with category b, so we can obtain:



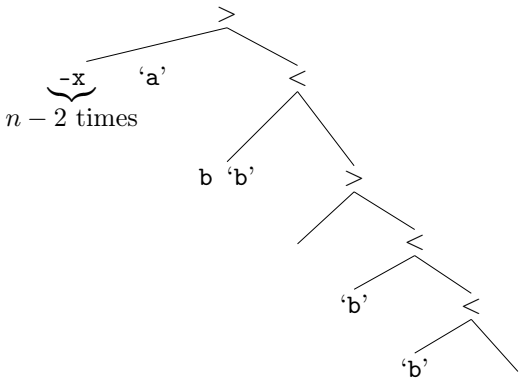
This tree has a movement trigger on its head, so we apply movement:



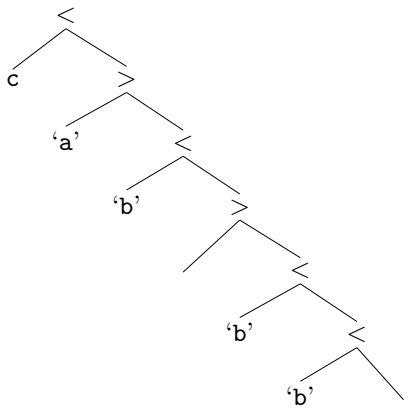
This tree can only be selected by a lexical item with category b, so we can obtain:



This tree has a movement trigger on its head, so applying movement we obtain the following:



And so on. Assuming that $n = 2$, this tree can be selected by the simple lexical item with category c, yielding a tree that has no outstanding syntactic features except the start-category c:



It should be obvious that, wherever the lexical item b that selects for category b is used, the phonologically empty category that selects for category b can be substituted. The only difference any such substitution makes is that the derived tree will have one less 'b' as leaf. Since there is always at least one such leaf (because of the way category a is assigned in the lexicon), and since the number of additional leaves labeled with 'b' in the derived tree is bounded by n, the string language generated by G_n is $\{\mathtt{ab}^i \mid 1 \leq i \leq n+1, i \in \mathbb{N}^+\}$.

Corollary 6.34 *Let G_n be as defined in Definition 6.33. Since $\mathrm{L}(G_n) = \{\mathtt{ab}^i \mid 1 \leq i \leq n+1, i \in \mathbb{N}^+\}$, $\mathrm{L}(G_0) \subset \mathrm{L}(G_1) \subset \ldots$, which constitutes an infinite ascending chain.*

Definition 6.35 *Let G_* be the following grammar:*

a ↦ =b ‘a’ c
b ↦ b
b ↦ =b ‘b’

It is easy to see that $\mathrm{L}(G_*) = \{\mathtt{ab}^i \mid i \in \mathbb{N}^+\}$.

Theorem 6.36 *Let G_n be as defined in Definition 6.33 and G_* as defined in Definition 6.35. Let $\mathcal{G}$ be any class of minimalist grammars that contains both G_* and G_n for any $n \in \mathbb{N}$. Then $\mathcal{G}$ is not (non-effectively) learnable from strings.*

Proof: Since $\mathrm{L}(G_n) = \{\mathtt{ab}^i \mid 1 \leq i \leq n+1, i \in \mathbb{N}^+\}$ and $\mathrm{L}(G_*) = \{\mathtt{ab}^i \mid i \in \mathbb{N}^+\}$, $\mathrm{L}(G_*) = \cup_{n\in\mathbb{N}}\mathrm{L}(G_n)$. Thus G_* is a limit point for any class that contains both G_* and G_n for all $n \in \mathbb{N}$. □

Note that imposing an additional numerical bound on the number of licensee features assigned to any one category does not allow the existence of this particular limit point. It may be that under such restrictions the classes of k-valued MGs are learnable, we will leave this an open question.

6.5 Conclusions and Future Work

This chapter presents and improves on results from Kanazawa (1998) concerning learnability of classes of Combinatorial CG, an extension of Classical CG. It is demonstrated that, given certain sets of combinators, the class of rigid grammars is not to learnable from strings. Constraints over combinatory rules are specified that guarantee preservation of finite elasticity of the class of structure languages generated by rigid grammars under these rules, and given such a class it is easy to show that the corresponding class of string languages has finite elasticity as well.

We also discuss results from Tiede (1999b) concerning the learnability of generalized quantifiers. In Tiede (1999b) a number of subclasses of the generalized quantifiers were shown to be learnable, and using techniques from the field of formal learning theory these results have been generalized; it has been shown that Tiede's classes are learnable under psychologically plausible restrictions on the learner.

The possibility of consistency with polynomial update time has been left open; the existence of an algorithm for producing a quantifier with a minimal denotation that is consistent with some positive data would be a sufficient condition.

The tree of numbers-approach to generalized quantifiers suggests that algorithms for learning geometric forms (thoroughly investigated within the PAC paradigm) might be applied straightforwardly. In Jain and Kinber (1999) **TxtEx** learning of open semi-hulls is considered. This work might be relevant although at present it is unclear if these geometric forms correspond to ‘natural’ quantifiers.

In van Benthem (1986a) it was suggested that tree automata may be useful for dealing with conditionals. Results on learnability of tree automata from Besombes and Marion (2002a) (to be discussed in Chapter 7) might be applicable to this subject.

Tree Adjoining Grammar is also discussed in this chapter. We obtained negative results for learning from strings for three very restricted classes of TAG, one allowing the empty string and two that are rigid (in two slightly different senses). The more restricted one of the latter two is not even learnable with the restrictions that the alphabet contain just two symbols (neither of which is the empty string) and that the grammar contain just one initial and one auxiliary tree.

The proofs for these results crucially rely on the optionality of adjunction and on the restrictions on adjunction that can be specified for each individual node. I conjecture that rigid TAGs are learnable under some restrictions on optionality, for example by marking all nodes in the grammar so that adjunction is either obligatory or prohibited at any given node. An alternative approach, possibly of more interest, could be to consider learning TAGs from *derivation* trees rather than from derived trees or strings. Derivation trees of TAGs can be interpreted as dependency structures (see e.g. Rambow and Joshi (1997); Dras et al. (to appear); Schuler et al. (2000)), so this setting has a natural linguistic interpretation. Learning from dependency trees will be discussed in more detail in the next chapter.

Minimalist Grammar (MG) is the last formalism discussed in this chapter. Again, a very restricted class turns out not to be learnable from strings, namely the class of 2-valued MGs with $|\Sigma| = 2$, allowing the assignment of an unbounded number of licensee features to the same category, and allowing phonologically empty categories.

Chapter 7

Learning Regular Tree Languages

7.1 Introduction

As noted in eg Fernau (2002, 2000), tree language induction is an important subject in the fields of (applied) Formal Learning Theory and Grammar Induction. We have already discussed the motivation from linguistics; one expects a learner to not just act as a characteristic function for a set of strings, but also to assign the right derivations (and thus meaning) to them.[1]

The work by Buszkowski and Penn and Kanazawa previously discussed has taken this approach, but without explicitly using a tree formalism. In this chapter the benefits of using the tree automata formalism (a generalization of finite state automata) will be shown by discussing results from Angluin (1982), Sakakibara (1992) and Besombes and Marion (2001, 2002a,b). The latter offers the following results:

1. Reset-free context-free grammars are identifiable from parse tree presentations. See Theorem 7.25.

2. Reversible dependency tree grammars are identifiable from positive examples. See Theorem 7.29.

3. Rigid Classical categorial grammars are identifiable from unlabeled derivation trees. See Theorem 7.30 This was already established by Kanazawa in Kanazawa (1998), but the proof is much shorter and conceptually simpler.

[1] There are other uses as well, Bernard and de la Higuera (1999) proposes it as a tool in Inductive Logic Programming, and there is also motivation from bioinformatics.

Related work: There are several papers on tree grammar identification, Angluin's and Sakakibara's results have already been mentioned. Other papers (Gonzalez et al. (1976); Levine (1981); Kamata (1984); Fukuda and Kamata (1984)) are based on the idea of k-tail inference for regular word languages from Biermann and Feldman (1972) and has been expounded in Knuutila and Steinby (1994). More recently, Fernau (2001) has applied this approach to XML grammars. In Carrasco et al. (1998) it was shown that regular tree languages are identifiable with probabilistic samples. The reader interested in context-free language inference may consult the surveys [Lee, Sakakibara1997, Mak97].

7.2 Regular Tree Languages

7.2.1 Trees are Terms

Background on regular tree languages can be found in the survey book Comon et al. (1997). A ranked alphabet is a tuple $(\mathcal{F}, arity)$ where $\mathcal{F}$ is a finite set of symbols and $arity$ is a function from $\mathcal{F}$ to $\mathbb{N}$, which indicates the arity of a symbol. Given a set $\mathcal{X}$ of variables, terms are inductively defined: a symbol of arity 0 is a term, a variable of $\mathcal{X}$ is a term, and if f is a symbol of arity n and $t_1, \ldots, t_n$ are terms, then $f(t_1, \ldots, t_n)$ is a term. The set of all terms is denoted by $\mathcal{T}(\mathcal{F}, \mathcal{X})$, the set of ground terms is dentoted by $\mathcal{T}(\mathcal{F})$. Throughout, labeled ordered trees are represented by terms.

A *context* is a term $C[\diamond]$ containing a special variable $\diamond$ which occurs just once in that term, it marks an empty place. Throughout, the substitution of $\diamond$ by a term u is written $C[u]$, its states that u is an occurrence of the term $C[u]$.

7.2.2 Tree Automata

A (nondeterministic) finite tree automaton (NFTA) is a quadruplet $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_F, \underset{\mathcal{A}}{\longrightarrow} \rangle$ such that $\mathcal{Q}$ is a finite set of states, $\mathcal{Q}_F \subseteq \mathcal{Q}$ is the set of final states, and $\underset{\mathcal{A}}{\longrightarrow}$ is the set of transitions. A state q of a deterministic tree automaton A is *useful* if and only if there exists a tree t and some node $x \in \Delta_t$ such that $\delta(t/x) = q$ and $\delta(t) \in F$. A deterministic automaton containing only useful states is called *stripped*. A transition is a ground rewrite rule of the form $f(q_1, \ldots, q_n) \underset{\mathcal{A}}{\longrightarrow} q$ where q and $q_1, \ldots, q_n$ are states of $\mathcal{Q}$, and f is a symbol of arity n, in the case that $n = 0$ the transition is just of the form $a \underset{\mathcal{A}}{\longrightarrow} q$.

A finite tree automaton is a *deterministic* finite tree automaton (DFTA) if it contains no rules sharing the same left hand side. In this case $\underset{\mathcal{A}}{\longrightarrow}$ represents a mapping which is not necessarily defined for all entries, so this means that we shall consider incomplete automata. The single derivation relation $\underset{\mathcal{A}}{\longrightarrow}$ is

defined so that $t \underset{\mathcal{A}}{\longrightarrow} u$ if and only if there is a transition $f(q_1, \ldots, q_n) \underset{\mathcal{A}}{\longrightarrow} q$ such that $t = v[f(q_1, \ldots, q_n)]$ and $u = v[q]$. Note that $\underset{\mathcal{A}}{\longrightarrow} \subseteq \mathcal{T}(\mathcal{F}, \mathcal{Q}) \times \mathcal{T}(\mathcal{F}, \mathcal{Q})$ where states of $\mathcal{Q}$ are 0-ary symbols. The derivation relation $\overset{*}{\underset{\mathcal{A}}{\longrightarrow}}$ is the reflexive and transitive closure of $\underset{\mathcal{A}}{\longrightarrow}$. The tree language *recognized* by $\mathcal{A}$ is $L_{\mathcal{A}} = \{t \in \mathcal{T}(\mathcal{F}) \mid t \overset{*}{\underset{\mathcal{A}}{\longrightarrow}} q_f \text{ and } q_f \in \mathcal{Q}_F\}$.

7.2.3 Reversible Regular Tree Languages

Definition 7.1 *A* DFTA $\mathcal{A}$ *is* reversible *if and only if*

1. *There are no two rules with left hand sides that differ by just one symbol. That is, there are no transitions* $f(p_1, \ldots, p_n, q, p_{n+1}, \ldots p_m) \underset{\mathcal{A}}{\longrightarrow} p$ *and* $f(p_1, \ldots, p_n, q', p_{n+1}, \ldots, p_m) \underset{\mathcal{A}}{\longrightarrow} p$ *where* $q \neq q'$.
2. $\mathcal{A}$ *has one final state.*

A tree language is reversible if it is recognised by a reversible DFTA. Note that a symbol of arity n and $n-1$ states determine at most one transition.

7.2.4 Reversible Regular Tree Grammars

A *regular tree grammar* (RTG) is a quadruplet $\Gamma = \langle \mathcal{F}, \mathcal{X}, \underset{\Gamma}{\longrightarrow}, S \rangle$ where $S \in \mathcal{X}$ is the start variable. Each production is of the form $X \underset{\Gamma}{\longrightarrow} t$ where X is a variable of $\mathcal{X}$ called the head, and t is a term of $\mathcal{T}(\mathcal{F}, \mathcal{X})$. Throughout no productions of the form $X \underset{\Gamma}{\longrightarrow} Y$, where Y is a variable of $\mathcal{X}$, are allowed. Define $t \underset{\Gamma}{\longrightarrow} u$ if and only if there is a production $X \underset{\Gamma}{\longrightarrow} v$ such that $u = t[X \leftarrow v]$. The derivation relation $\overset{*}{\underset{\Gamma}{\longrightarrow}}$ is the reflexive and transitive closure of $\underset{\Gamma}{\longrightarrow}$. The language *produced* by Γ is $L_\Gamma = \{t \in \mathcal{T}(\mathcal{F}) \mid S \overset{*}{\underset{\Gamma}{\longrightarrow}} t\}$.

Definition 7.2 *A grammar* Γ *is* reversible *if and only if*

1. *there are no productions* $X \underset{\Gamma}{\longrightarrow} C[Y]$ *and* $X \underset{\Gamma}{\longrightarrow} C[Z]$ *starting with the same head* X *such that* $Y \neq Z$,
2. *there are no productions* $X \underset{\Gamma}{\longrightarrow} t$ *and* $Y \underset{\Gamma}{\longrightarrow} t$ *with the same right hand-side such that* $X \neq Y$.

Definition 7.3 *An* RTG Γ *is* normal *if each production is of the form* $X \underset{\Gamma}{\longrightarrow} a$ *or of the form* $X \underset{\Gamma}{\longrightarrow} f(X_1, \ldots, X_n)$.

Theorem 7.4 *A language* L *is reversible if and only if the language* L *is produced by a reversible and normal* RTG.

7.2.5 Identifification of Reversible Tree Languages

A *positive presentation* of a language L is a sequence $t_1, \ldots, t_n, \ldots$ which enumerates all elements of L. Let Ω be a given class of automata (or grammars). An inference algorithm A takes as input a finite segment $t_1, \ldots, t_n$ of a positive presentation of L and guesses an automaton $A(t_1, \ldots, t_n)$. The inference algorithm A converges to L if there is a stage N such that for all $n \geq N$, the language provided by $A(t_1, \ldots, t_n)$ is exactly L. A class of languages $\mathcal{L}$ is *identifiable* if and only if there is an inference algorithm A such that for each positive presentation of a language L, A converges to L.

Theorem 7.5 *The class of reversible tree languages is identifiable.*

We shall demonstrate the theorem above in the next section. The main notions are now set, the reader may skip the proofs and follow most discussions about applications in the last section of this chapter.

7.3 Identification of Reversible Tree Languages

7.3.1 An Algebraic Characterization of Reversible Tree Languages

An equivalence relation $\equiv$ is *closed under context* if for all terms t and u in $\mathcal{T}(\mathcal{F})$, $t \equiv u$ if for every context $C[\diamond]$, $C[t] \equiv C[u]$. A *congruence* $\equiv$ on $\mathcal{T}(\mathcal{F})$ is an equivalence relation which is closed under context.

Given a DFTA $\mathcal{A}$, the equivalence relation $\equiv_{\mathcal{A}}$ is defined as: $t \equiv_{\mathcal{A}} u$ if $t \xrightarrow[\mathcal{A}]{*} q$ and $u \xrightarrow[\mathcal{A}]{*} q$. It is easy to see that $\equiv_{\mathcal{A}}$ is closed under context, so it is a congruence.

Lemma 7.6 *Let $\mathcal{A}$ be a reversible* DFTA. *For every context $C[\diamond]$ and for every term t and u such that $C[t]$ and $C[u]$ are in $L_{\mathcal{A}}$, $t \equiv_{\mathcal{A}} u$.*

`Proof:` By induction on the size of the context. Basis: Suppose that $C[\diamond] = \diamond$. By Lemma assumption, t and u are in $L_{\mathcal{A}}$. Since $\mathcal{A}$ is reversible, there is only one final state q_f, therefore we have $t \xrightarrow[\mathcal{A}]{*} q_f$ and $u \xrightarrow[\mathcal{A}]{*} q_f$. We conclude that $t \equiv_{\mathcal{A}} u$.

Inductive step: Suppose that $C[\diamond] = C'[f(t_1, \ldots, t_n, \diamond, t_{n+1}, \ldots, t_m)]$. By the lemma assumption, $t_i \xrightarrow[\mathcal{A}]{*} q_i$ for some state q_i. Since $\mathcal{A}$ is reversible the states $t_1, \ldots, t_m$ determine a unique transition whose left hand side is $C'[f(q_1, \ldots, q_n, q, q_{n+1}, \ldots, q_m)]$. This implies that $t \xrightarrow[\mathcal{A}]{*} q$ and $u \xrightarrow[\mathcal{A}]{*} q$. Therefore $t \equiv_{\mathcal{A}} u$. □

Given a tree language L, the congruence $\equiv_L$ is defined as: $t \equiv_L u$ if for every context $C[\diamond]$, $C[t] \in L$ if and only if $C[u] \in L$. Following the Myhill-Nerode Theorem, the index of $\equiv_L$ is lower or equal than the index of $\equiv_{\mathcal{A}}$ for any automaton $\mathcal{A}$ which recognises L.

As a consequence, the minimal DFTA (up to a renaming of states) $\mathcal{A}_L = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_F, \underset{L}{\longrightarrow} \rangle$ which recognises L is defined as follows: let $[t]$ be the equivalence class of t with respect to $\equiv_L$ and $\mathrm{sub}(L)$ be the set of subterms of L. Let $\mathcal{Q} = \{[t] | t \in \mathrm{sub}(L)\}$ and $\mathcal{Q}_F = \{[t] | t \in L\}$. For every state $[t_1], \ldots, [t_n]$ and for each n-ary symbol $f \in \mathcal{F}$, there is a transition $f([t_1], \ldots, [t_n]) \underset{L}{\longrightarrow} [f(t_1, \ldots, t_n)]$.

Theorem 7.7 *A tree language L is* reversible *if and only if for every context $C[\diamond]$ and for every term t and every term u, if $C[t]$ and $C[u]$ are in L, then $t \equiv_{\mathcal{L}} u$.*

`Proof:` There is a reversible DFTA $\mathcal{A}$ which recognises L. Following the Myhill-Nerode Theorem, $\equiv_{\mathcal{A}}$ refines $\equiv_L$, that is if $t \equiv_{\mathcal{A}} u$ then $t \equiv_L u$. Therefore, we conclude by Lemma 7.6.

Conversely, consider the minimal DFTA $\mathcal{A}_L$ recognising L. This DFTA has only one final state because the lemma assumption says that each tree of L belongs to the same equivalence class. Next, suppose that we have two transitions of the form $f(p_1, \ldots, p_n, q, p_{n+1}, \ldots, p_m) \underset{L}{\longrightarrow} q''$ and $f(p_1, \ldots, p_n, q', p_{n+1}, \ldots, p_m) \underset{L}{\longrightarrow} q''$. Since $\mathcal{A}_L$ is minimal, there are terms $t_1, \ldots, t_m$ such that $t_i \underset{L}{\overset{*}{\longrightarrow}} p_i$. This leads us to consider a context $C[\diamond] = C'[f(t_1, \ldots, t_n, \diamond, t_{n+1}, \ldots, t_m)]$ such that $C[q] \underset{L}{\overset{*}{\longrightarrow}} q_f$ and $C[q'] \underset{L}{\overset{*}{\longrightarrow}} q_f$ where q_f is the final state of $\mathcal{A}_L$. However, the lemma assumption implies that $q = q'$. We conclude that $\mathcal{A}_L$ is reversible, and so L is reversible. □

This proof has the following consequence:

Corollary 7.8 *A tree language L is reversible if and only if the minimal* DFTA *$\mathcal{A}_L$ is reversible.*

Example 7.9 *Let $L = \{f(g^n(a)) \mid n \geq 0\} \cup \{g(f^n(a)) \mid n \geq 0\}$. The tree language L is recognised by the* DFTA *$\mathcal{A}$ whose transitions are*

$$\begin{array}{rr} a \underset{\mathcal{A}}{\longrightarrow} A & f(A) \underset{\mathcal{A}}{\longrightarrow} C \\ g(A) \underset{\mathcal{A}}{\longrightarrow} B & f(C) \underset{\mathcal{A}}{\longrightarrow} S \\ g(B) \underset{\mathcal{A}}{\longrightarrow} G & f(F) \underset{\mathcal{A}}{\longrightarrow} F \\ g(G) \underset{\mathcal{A}}{\longrightarrow} G & f(G) \underset{\mathcal{A}}{\longrightarrow} S \\ g(F) \underset{\mathcal{A}}{\longrightarrow} S & \end{array}$$

The final states are S,B and C.

The DFTA $\mathcal{A}$ is minimal and is therefore isomorphic to $\mathcal{A}_L$. As a consequence of Corollary 7.8 the language L is not a reversible tree language, although it is the union of two reversible tree languages.

7.3.2 Characteristic Samples

Let L be a reversible language recognised by the minimal automaton $\mathcal{A}_L$. Each state q has a term $\mathrm{rt}(q)$ of $\mathcal{T}(\mathcal{F})$ associated with it such that $\mathrm{rt}(q) \xrightarrow[L]{*} q$ and a minimal context $C_q[\diamond]$, with respect to the size, such that $C_q[q] \xrightarrow[L]{*} q_f$.

The set of *characteristic samples* $\mathrm{CS}(L)$ is the smallest set which contains

1. the term $C_q[\mathrm{rt}(q)]$ for each state q,
2. the term $C_q[f(\mathrm{rt}(q_1), \ldots, \mathrm{rt}(q_n))]$ for each transition $f(q_1, \ldots, q_n) \xrightarrow[L]{} q$.

It follows immediately that $\mathrm{CS}(L) \subseteq L$. Also note that $C_{q_f}[\diamond] = \diamond$ where q_f is a final state.

Theorem 7.10 *Let L_1 and L_2 be two reversible languages over $\mathcal{T}(\mathcal{F})$, and assume $\mathrm{CS}(L_1) \subseteq L_2$. Then $L_1 \subseteq L_2$.*

`Proof:` Suppose that $\mathcal{A}_1 = \langle \mathcal{F}, \mathcal{Q}_1, \{q_{f_1}\}, \xrightarrow[1]{} \rangle$ and $\mathcal{A}_2 = \langle \mathcal{F}, \mathcal{Q}_2, \{q_{f_2}\}, \xrightarrow[2]{} \rangle$ are the minimal DFTAs which recognize L_1 and L_2 respectively.

For each state $q \in \mathcal{Q}_1$, $C_q[\mathrm{rt}(q)] \in \mathrm{CS}(L_1) \subseteq L_2$. Thus $\mathrm{rt}(q) \in \mathrm{sub}(L_2)$. It follows that there is a unique function $\theta : \mathcal{Q}_1 \to \mathcal{Q}_2$ such that $\mathrm{rt}(q) \xrightarrow[2]{*} \theta(q)$, because $\mathcal{A}_2$ is deterministic.

We now state an important observation which is a consequence of Theorem 7.7. For each term $t \in \mathcal{T}(\mathcal{F})$, if $C_q[t]$ is in L_2, then $t \xrightarrow[2]{*} \in \theta(q)$.

By induction on the size of the term $t \in \mathcal{T}(\mathcal{F})$ it can be shown that for each state $q \in \mathcal{Q}_1$, if $C_q[t] \in L_1$ then $C_q[t] \in L_2$. It follows that if $t \in L_1$ then $t \in L_2$, because $C_{q_{f_1}}[t] = t$.

Basis: Suppose that t is a symbol of arity 0. There is a transition $t \xrightarrow[1]{} q$ and so $C_q[t] \in \mathrm{CS}(L_1) \subseteq L_2$.

Inductive step: Suppose that $t = f(t_1, \ldots, t_n)$. By the lemma hypothesis, there is a state q_i in $\mathcal{Q}_1$ such that $t_i \xrightarrow[1]{*} q_i$. Because $C_q i[t_i] \in L_1$, the induction hypothesis is applied to obtain $C_{q_i}[t_i] \in L_2$. Following the observation above, $t_i \xrightarrow[2]{*} \theta(q_i)$. We have

$$C_q[f(t_1, \ldots, t_n)] \xrightarrow[2]{*} C_q[f(\theta(q_1), \ldots, \theta(q_n))] \tag{7.1}$$

On the other hand, since $C_q[f(\mathrm{rt}(q_1), \ldots, \mathrm{rt}(q_n))] \in L_2$, we have

$$C_q[f(\mathrm{rt}(q_1), \ldots, \mathrm{rt}(q_n))] \xrightarrow[2]{*} C_q[f(\theta(q_1), \ldots, \theta(q_n))] \tag{7.2}$$

$$\xrightarrow[2]{*} q_{f_2} \tag{7.3}$$

By combining 7.1 and 7.3,

$$C_q[f(t_1, \ldots, t_n)] \xrightarrow[2]{*} q_{f_2} \tag{7.4}$$

so $C_q[t] \in L_2$. □

Based on Angluin (1982), the characteristic samples of reversible tree languages are telltale sets. Consequently, the class of reversible tree languages is identifiable.

7.3.3 An Efficient Learning Algorithm

The learning algorithm works as follows: The input is a finite set S of positive examples, that is terms of a (regular) target language. Let us first define the prefix tree automaton $\mathrm{PTA}(S) = \langle \mathcal{F}, \mathcal{Q}_0, \mathcal{Q}_{F_0}, \underset{0}{\rightarrow} \rangle$ as follows. For each subterm t of $\mathrm{sub}(S)$, there is a state written $[t]$ in $\mathcal{Q}_0$. The set of final states $\mathcal{Q}_{F_0}$ contains each state $[t]$ where $t \in S$. For each subterm $f(t_1, \ldots, t_n)$ of $\mathrm{sub}(S)$, we have a transition $f([t_1], \ldots, [t_n]) \underset{0}{\rightarrow} [f(t_1, \ldots, t_n)]$. It follows directly that $L_{\mathrm{PTA}(S)} = S$.

Next a succession of NFTAs $\mathcal{A}_0 = \mathrm{PTA}(S), \mathcal{A}_1, \ldots, A_n$ is computed by repeatedly applying one of the reduction rules described in Figure 7.1 until we find two identical NFTAs. That is, A_{i+1} is obtained from $\mathcal{A}_i$ by merging two states following one of the rules. The process terminates after n reduction steps since each step decreases the number of states, so $\mathrm{nf}(S) = \mathcal{A}_n$.

Lemma 7.11 *The* DFTA $\mathrm{nf}(S)$ *is a reversible* DFTA.

`Proof:` Since no rules are applicable, $\mathrm{nf}(S)$ is necessarily reversible. □

An *automaton homomorphism* between the NFTA $\mathcal{A}_1 = \langle \mathcal{F}, \mathcal{Q}_1, Q_{F_1}, \underset{1}{\rightarrow} \rangle$ and $\mathcal{A}_2 = \langle \mathcal{F}, \mathcal{Q}_2, Q_{F_2}, \underset{2}{\rightarrow} \rangle$ is a function θ which maps $\mathcal{Q}_1$ to $\mathcal{Q}_2$ and which has the property that if $f(q_1, \ldots, q_n) \underset{1}{\rightarrow} q$ then $f(\theta(q_1), \ldots, \theta(q_n)) \underset{2}{\rightarrow} \theta(q)$, for each transition of $\mathcal{A}_1$. We say that $\mathcal{A}_1 \sqsubseteq \mathcal{A}_2$ if there is some automaton homomorphism between them. Note that if $\mathcal{A}_1 \sqsubseteq \mathcal{A}_2$ then $L_{\mathcal{A}_1} \subseteq L_{\mathcal{A}_2}$.

Lemma 7.12 *If* $S \subseteq L$ *then* $\mathrm{PTA}(S) = \mathcal{A}_0 \sqsubseteq \mathcal{A}_L$.

`Proof:` Define θ as $\theta([t])$ is the equivalence class of t. □

Lemma 7.13 *Let* $\mathcal{A}$ *be a reversible* DFTA. *Assume that* $\mathcal{A}_{i+1}$ *is obtained by applying either* **R1**, **R2** *or* **R3** *to* $\mathcal{A}_i$. *If* $\mathcal{A}_i \sqsubseteq \mathcal{A}$ *then* $\mathcal{A}_{i+1} \sqsubseteq \mathcal{A}$.

1. **R1** If $f(p_1, \ldots, p_n, q, p_{n+1}, \ldots, p_m) \to p$ and $f(p_1, \ldots, p_n, q', p_{n+1}, \ldots, p_m) \to p$ then $q = q'$.
2. **R2** if $f(q_1, \ldots, q_n) \to q$ and $f(q_1, \ldots, q_n) \to q'$ then $q = q'$.
3. **R3** if q_{f_1} and q_{f_2} are both final states then $q_{f_1} = q_{f_2}$.

Figure 7.1: Reduction rules

`Proof:` Let θ be the automaton homomorphism from $\mathcal{A}_i$ to $\mathcal{A}$. We prove the lemma by inspecting the three rules which can be used to construct $\mathcal{A}_{i+1}$. Rule **R1** is applied. Since $\mathcal{A}$ is reversible, $\theta(q) = \theta(q')$. Rule **R2** is applied. Since $\mathcal{A}$ is deterministic, $\theta(q) = \theta(q')$. Rule **R3** is applied. Since $\mathcal{A}$ has only one final state, $\theta(q_{f_1}) = \theta(q_{f_2})$. Thus θ is an automaton homomorphism from $\mathcal{A}_{i+1}$ to $\mathcal{A}$. □

Lemma 7.14 *For each reversible language L, if $S \subseteq L$ then $L_{\mathrm{nf}(S)} \subseteq L$.*

`Proof:` Follows directly from Lemma 7.12 and Lemma 7.13. □

Example 7.15 *Suppose we have the entry $\langle f(a,b), f(g(a),b), f(g(g(a)),b)\rangle$. The algorithm first calculates the automaton $\mathrm{PTA}(S) = \langle \mathcal{F}, \mathcal{Q}_0, \mathcal{Q}_{F_0}, \underset{0}{\to}\rangle$, where*

- *$\mathcal{F}$ is the set $\{f, g, a, b\}$,*
- *$\mathcal{Q}_0$ is the set $\{[a], [b], [f(a,b)], [g(a)], [f(g(a),b)], [g(g(a))], [f(g(g(a)),b)]\}$,*
- *$\mathcal{Q}_{F_0}$ is the set $\{[f(a,b)], [f(g(a),b)], [f(g(g(a)),b)]\}$,*
- *$\underset{0}{\to}$ is the set $\{a \underset{0}{\to} [a], b \underset{0}{\to} [b], f([a],[b]) \underset{0}{\to} [f(a,b)], g([a]) \underset{0}{\to} [g(a)]$, $f([g(a)],[b]) \underset{0}{\to} [f(g(a),b)], g([g(a)]) \underset{0}{\to} [g(g(a))], f([g(g(a))],[b]) \underset{0}{\to}$ $[f(g(g(a)),b)]\}$.*

*Applying the rule **R3** three times we get $[f(a,b)] = [f(g(a),b)] = [f(g(g(a)),b)]$ and the new set $\langle a \to [a], b \to [b], f([a],[b]) \to [f(a,b)], g([a]) \to [g(a)], f([g(a)],[b]) \to [f(a,b)], g([g(a)]) \to [g(g(a))], f([g(g(a))],[b]) \to [f(a,b)]\rangle$. Now the rule **R1** can be applied twice to obtain $[a] = [g(a)] = [g(g(a))]$, so the output is the automaton $\langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_F, \to\rangle$, where*

- *$\mathcal{F}$ is the set $\langle f, g, a, b\rangle$,*
- *$\mathcal{Q}$ is the set $\langle [a], [b], [f(a,b)]\rangle$,*
- *$\mathcal{Q}_F$ is the set $\langle [(f(a,b)]\rangle$,*

- $\rightarrow$ *is the set* $\langle a \rightarrow [a], b \rightarrow [b], f([a],[b]) \rightarrow [f(a,b)], g([a]) \rightarrow [a] \rangle$.

Theorem 7.16 *(Proof of Theorem 7.5). Assume that L is a reversible tree language. For each positive presentation $t_1, \ldots, t_n, \ldots$ of L, there is a step N such that for all $n > N$,* $\mathrm{nf}(t_1, \ldots, t_n)$ *is the minimal reversible* DFTA *which recognizes L. In other words, the class of reversible tree languages is identifiable.*

`Proof:` There is a step N such that for all $n > N$, $t_1, \ldots, t_n$ contains $\mathrm{CS}(L)$. By Lemma 7.14, $L_{\mathrm{nf}(t_1,\ldots,t_n)} \subseteq L$. Now, $L_{\mathrm{nf}(t_1,\ldots,t_n)} \subseteq L$ is a reversible tree language. We apply Theorem 7.10, and we conclude that $L \subseteq L_{\mathrm{nf}(t_1,\ldots,t_n)}$. Therefore, we have $L = L_{\mathrm{nf}(t_1,\ldots,t_n)}$. □

The learning algorithm is incremental and runs in quadratic time in the size of the examples.

7.4 Learning with Structural Examples

We now discuss the identification in the limit of word languages and we give some applications as a way of illustrating the learning method developed in the previous sections.

7.4.1 Context-Free Word Languages

Let us recall briefly the definition of context-free grammars (CFG). A context-free grammar G is a quadruple $\langle \Sigma, \mathcal{N}, \underset{G}{\longrightarrow}, S \rangle$ where Σ is the alphabet, $\mathcal{N}$ is the set of non-terminal symbols, and $S \in \mathcal{N}$ is the start symbol. The production rules are defined by $\underset{G}{\longrightarrow}$ and are of the form $X \underset{G}{\longrightarrow} w$ where $X \in \mathcal{N}$ and $w \in (\Sigma \cup \mathcal{N})^*$. The language L_G defined by G is $L_G = \langle w \mid S \overset{*}{\underset{G}{\longrightarrow}} w \rangle$. Define $\mathcal{D}(G)$ as the set of derivation trees of G.

There is a close relationship between context-free word languages and regular tree grammars. To see this, let Σ be the set of symbol of arity 0 of $\mathcal{F}$. We define the yield function yield from $\mathcal{T}(\mathcal{F})$ to Σ as follows:

$$\begin{aligned} \mathrm{yield}(a) &= a \\ \mathrm{yield}(f(t_1, \ldots, t_n)) &= \mathrm{yield}(t_1) \ldots \mathrm{yield}(t_n) \end{aligned}$$

If $L \subseteq \mathcal{T}(\mathcal{F})$ is a tree language, then $\mathrm{yield}(L) = \langle \mathrm{yield}(t) \mid t \in L \rangle$.

Theorem 7.17 *If L is a regular tree language then* $\mathrm{yield}(L)$ *is a context-free language.*

The second thing is that the set of derivation trees of context-free grammars is a regular tree language.

Theorem 7.18 *Assume that G is a context-free grammar. Then, the set of derivation trees $\mathcal{D}(G)$ is a regular tree language.*

Let us look at the construction of a reversible and normal RTG $\Gamma = \langle \mathcal{F}, \mathcal{X}, \underset{\Gamma}{\longrightarrow}, S \rangle$ from a grammar $G = \langle \Sigma, \mathcal{N}, \underset{G}{\longrightarrow}, S \rangle$. Every letter in Σ is a 0-ary symbol in $\mathcal{F}$. For each rule $X \underset{G}{\longrightarrow} w$ where w is a word of length n, there is a symbol X_n of arity n in $\mathcal{F}$. Every non-terminal of $\mathcal{N}$ is a variable of $\mathcal{X}$. For each letter $a \in \Sigma$, there is a variable $[a]$. For each production $X \underset{G}{\longrightarrow} w_1, \ldots, w_n$ there is a production $X \underset{\Gamma}{\longrightarrow} X_n(X_1, \ldots, X_n)$ where $X_i = w_i$ if w_i is in $\mathcal{N}$, otherwise $X_i = [w_i]$. Lastly, for each letter $a \in \Sigma$, we have $[a] \underset{\Gamma}{\longrightarrow} a$.

7.4.2 Structural Examples

In the case of context-free languages it is in general hard to prove learnability of string languages (or coming up with sufficient conditions for such classes to be learnable). Consequently several authors have suggested to learn classes of grammars, like context-free grammars Sakakibara (1992) or categorial grammars Kanazawa (1998), where positive examples are annotated by additional information and are called *structural examples*. We present two cases to illustrate this idea.

1. The full presentation of a CFG G is a sequence $t_1, t_2, \ldots$ of all parse trees of G.

2. A delabeling sk is function defined by:

$$\begin{aligned} \mathrm{sk}(a) &= a \\ \mathrm{sk}(f(t_1, \ldots, t_n)) &= \sigma_n(\mathrm{sk}(t_1), \ldots, \mathrm{sk}(t_n)) \end{aligned}$$

   The skeleton presentation of a grammar G is a sequence $\mathrm{sk}(t_1), \mathrm{sk}(t_2), \ldots$ of all delabeled parse trees of G.

The available data consists of a regular tree language which is a homomorphic image of derivation trees of some CFG. These observations lead to the consideration of h-presentations of a CFG G defined as follows: assume that h is a tree homomorphism, then an h-presentation of a CFG G is a sequence $h(t_1), \ldots, h(t_n), \ldots$ where $t_1, \ldots, t_n, \ldots$ is an enumeration of all parse trees of G.

Note that h being a linear homomorphism is a sufficient condition for an h-presentation to be a regular tree language.

7.4.3 Grammar Identification

Let $h(\mathcal{D}(G)) = \{h(t) | t \in \mathcal{D}(G)\}$. Let Ω be a given class of grammars and h be a tree homomorphism. Given a h-presentation $h(t_1), \ldots, h(t_n), \ldots$ the inference

algorithm A *converges to* $G \in \Omega$ if there is a stage N such that for all $n > N$, the language provided by $A(h(t_1), \ldots, h(t_n))$ is exactly $h(\mathcal{D}(G))$.

A class of grammars Ω is *identifiable from h-presentations* if and only if there is an inference algorithm A such that for each h-presentation of a grammar G, A converges to G.

7.4.4 Sakakibara's Approach

Sakakibara demonstrated that the class of reversible context-free grammars is identifiable from skeleton presentations.

Definition 7.19 *A* CFG *is reversible if and only if*

1. **(Reset-free)** *there are no productions* $X \underset{G}{\longrightarrow} wYv$ *and* $X \underset{G}{\longrightarrow} wZv$ *such that* Y *and* Z *are non-terminals and* $Y \neq Z$,
2. **(Deterministic)** *there are no productions* $X \underset{G}{\longrightarrow} t$ *and* $Y \underset{G}{\longrightarrow} t$ *with the same right hand-side and such that* $X \neq Y$.

A reversible context-free language is a language produced by a reversible CFG.

Theorem 7.20 *(Sakakibara (1992)). The class of reversible context-free grammars is identifiable from skeleton presentations.*

`Proof:` The skeleton presentation of a reversible CFG is a reversible tree language, so Theorem 7.5 implies that the class of skeleton presentations is identifiable. □

7.4.5 Identification of Reversible Context-Free Word Languages

Definition 7.21 *The class* $\mathcal{R}$ *of reversible context-free word languages is the smallest class such that for every reversible tree language* L_1 *there is a language* L *in* $\mathcal{R}$ *such that* $L = \text{yield}(L_1)$. *We call* L_1 *a* reversible tree presentation *of* L.

Again from Theorem 7.5 the following result is obtained:

Theorem 7.22 *The class* $\mathcal{R}$ *is identifiable from reversible tree presentations.*

In order to make this result clear and to compare it with Sakakibara's work, we shall illustrate it by discussing some of its consequences.

Example 7.23 *The grammar* Γ *defined below is reversible:*

$$\begin{array}{ll} S \underset{\Gamma}{\rightarrow} f(X,Y) & X \underset{\Gamma}{\rightarrow} c \\ X \underset{\Gamma}{\rightarrow} g(A,X,B) & Y \underset{\Gamma}{\rightarrow} d \\ Y \underset{\Gamma}{\rightarrow} g(A,Y,B) & B \underset{\Gamma}{\rightarrow} b \\ A \underset{\Gamma}{\rightarrow} a & \end{array}$$

By Theorem 7.5, L_Γ is identifiable. Theorem 7.22 states that the context-free language yield(L_Γ) is identifiable from any enumeration of L_Γ which constitutes a reversible tree presentation. However, L_Γ is not the set of parse trees of a context-free language.

Example 7.24 *This example shows that learning is facilitated by a full presentation of parse trees. The following* CFG G *is not reversible because it is not deterministic:*

$$\begin{array}{lll} S & \underset{G}{\rightarrow} & ab \\ S & \underset{G}{\rightarrow} & aXb \\ X & \underset{G}{\rightarrow} & ab \end{array}$$

The set $\mathcal{D}(G)$ *of parse trees is a reversible tree language and is generated by the following (reversible and minimal) tree grammar:*

$$\begin{array}{lll} S & \underset{G}{\rightarrow} & S_2([a],[b]) \\ S & \underset{G}{\rightarrow} & S_3([a],X,[b]) \\ X & \underset{G}{\rightarrow} & X_2([a],[b]) \\ [a] & \underset{G}{\rightarrow} & a \\ [b] & \underset{G}{\rightarrow} & b \end{array}$$

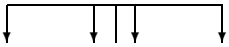
Theorem 7.25 *The class of reset-free context-free languages is identifiable from full presentations.*

7.5 Lexical Dependency Tree Languages

Following Dikovsky and Modina (2000),[2] we present a class of projective dependency grammars that was introduced in Hays (1961) and Gaifman (1965).

[2]Also see Dikovsky (2001), for details on a dependency tree grammar formalism that can generate non-semilinear context-sensitive languages.

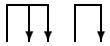
A lexical dependency grammar (LDG) is a quadruplet $\langle \Sigma, N, \underset{\Gamma}{\rightarrow}, S \rangle$, where Σ is the alphabet, N is the set of non-terminal symbols, and $S \in N$ is the start symbol. Each production is of the form



$$X \rightarrow U_1, \ldots, U_p a V_1, \ldots, V_q,$$

where $X \in N$ and all U_i and V_j are in $\Sigma \cup N$. The right-hand side can be interpreted either as a labeled ordered directed tree of depth 1 whose head is a, or as the word $U_1, \ldots, U_p a V_1, \ldots, V_q$. Thus there is a total order on the tree nodes.

Example 7.26 *The grammar* $\Gamma_0 = \langle \{a, b\}, \{S\}, P, S \rangle$ *where* P *consists of:*



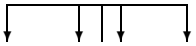
$$S \rightarrow a\ S\ b \mid a\ b$$

We define partial dependency trees *recursively as follows:*

1. *S is a partial dependency tree generated by Γ.*
2. *If*

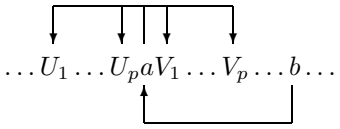
$$\ldots X \ldots b \ldots$$

is a partial dependency tree generated by Γ, and if



$$X \rightarrow U_1 \ldots U_p a V_1 \ldots V_q$$

is a production of Γ, then

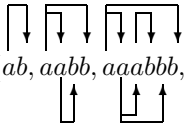


$$\ldots U_1 \ldots U_p a V_1 \ldots V_p \ldots b \ldots$$

is a partial dependency tree generated by Γ.

A *dependency tree* generated by an LDG Γ is a partial dependency tree of Γ in which all nodes are terminal symbols. The language $\mathcal{D}(\Gamma)$ is the set of all dependency trees generated by Γ.

Example 7.27 *The language generated by* Γ_0 *is*

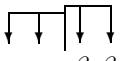


$$\mathcal{D}(\Gamma_0) = \{ab, aabb, aaabbb, \ldots\}$$

Without dependencies, we recognize the context-free language $\{a^n b^n \mid n > 0\}$. *Without the linear order on letters, the regular tree language is*

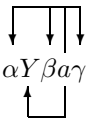
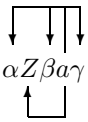
$$\{a(b), (a(a(b), b)), a(a(a(b), b), b), \ldots\}.$$

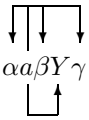
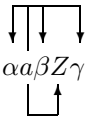
Note that the arity of a *is 1 or 2, but this is not problematic. We write*  $\alpha a \beta$


for $\alpha_1 \alpha_p a \beta_1 \beta_q$.

Definition 7.28 *An* LDG *grammar is* reversible *if and only if the following three conditions are satisfied:*

1. *if*  $X \to UaV$ *and if*  $Y \to UaV$ *, then* $X = Y$.

2. *If*  $X \to \alpha Y \beta a \gamma$ *and if*  $X \to \alpha Z \beta a \gamma$ *, then* $Y = Z$*, where* $Y, Z \in N$.

3. *If*  $X \to \alpha a \beta Y \gamma$ *and if*  $X \to \alpha a \beta Z \gamma$ *, then* $Y = Z$*, where* $Y, Z \in N$.

The class of *reversible dependency tree languages* is the class of languages generated by reversible LDG grammars.

Theorem 7.29 *The class of reversible dependency trees is identifiable.*

`Proof:` The algorithm described in Section 7.3.3 is easily adapted to take the node ordering into account and to handle symbols with variable arities. It is not diffcult to see that a presentation is a reversible tree language. □

7.6 Classical Categorial Grammar

We are now ready to present Besombes and Marion's alternative proof of Kanazawa's theorem on learning $\mathcal{AL}_{\text{rigid}}$ from structures. We feel that its conciseness demonstrates the power of the tree automata-approach to learning:

Theorem 7.30 *(Kanazawa). The class of rigid grammars is identifiable from unlabeled derivation tree presentations.*

`Proof:` The set of unlabeled derivation trees is a reversible regular language. To see this, construct a normal RTG Γ such that for each subtype A of a type assigned to a symbol there is a state $[A]$. For each symbol u of Σ there is a corresponding production $[Lex(u)] \underset{\Gamma}{\rightarrow} u$. For each possible subtype A of such a type we have the following productions (known as *functional* productions):

$$[B] \underset{\Gamma}{\rightarrow} \backslash([A], [A \backslash B])$$
$$[A] \underset{\Gamma}{\rightarrow} /([A/B], [B])$$

Note that functional productions are reversible. Since the grammar is rigid, Γ is reversible. Thus, by Theorem 7.5, this class is learnable. □

Chapter 8

The Lambek Calculus

8.1 Introduction

This chapter describes the logical approach in categorial grammar, a tradition that originated in Ajdukiewicz (1935) and was further developed in Bar-Hillel (1953) and Lambek (1958). This chapter will only give a brief introduction, Casadio (1988) gives an account of the historical roots of this field, while Moortgat (1997) and Buszkowski (1997) offer a more comprehensive overview.

General combinatory grammar is essentially a rule based approach to linguistic analysis, where a finite collection of unary type transitions and binary type combinators are postulated as *primitive* rule schemata, some examples of these rules were given in Chapter 6.

In contrast to GCG, Lambek systems do not need explicit rules defining grammatical composition. Instead, they rely on a fixed 'logical' component and a variable 'structural' component. The pure logic of residuation **NL** ('non-associative Lambek') captures the fixed logical component:

Definition 8.1 *The pure logic of residuation* **NL** *(Lambek (1961)).*

$$\begin{array}{rl} \text{(REFL)} & A \to A, \\ \text{(TRANS)} & \textit{if } A \to B \textit{ and } B \to C\textit{, then } A \to C, \\ \text{(RES)} & A \to C/B \textit{ if and only if } A \bullet B \to C \textit{ if and only if } B \to A\backslash C. \end{array}$$

This version of categorial grammar is known as the *deductive* approach. Among the reasons in favor of this approach is systematicity of the relation between syntax and semantics.[1] Semantics in Montague's tradition associates each syntactic rule with a semantic one; in the deductive approach this correspondence is more strict than the usual notion of compositionality (see Janssen

[1] Note that Lambek calculi have two kinds of semantics; the 'meaning' kind of semantics discussed here, and a *relational* semantics that yields soundness and completeness of the calculi with respect to Kripke-style relational models, see e.g. Došen (1992).

(1997)), it makes use of the Curry-Howard correspondence between proofs and types.[2] Also note that Tiede (1998) argues for the necessity of introduction rules for any compositional treatment of natural language along these lines.

By relaxing sensitivity of **NL** in a number of (linguistically relevant) dimensions one can obtain other categorial type logics. By adding to the pure logic of residuation a postulate licensing commutative resource management, one obtains freedom in the dimension of linear precedence. The resulting logic is called **NLP**. Adding a postulate licensing associative resource management yields the logic **L**, which provides freedom in the dimension of immediate dominance. Combining these postulates yields the logic **LP**, which obviously licenses both commutative and associative resource management.

Definition 8.2 ***Associativity and Commutativity Postulates***

$$\begin{array}{rl} \text{(ASS)} & (A \bullet B) \bullet C \leftrightarrow A \bullet (B \bullet C), \\ \text{(COMM)} & A \bullet B \rightarrow B \bullet A \end{array}$$

By necessity, GCG systems are only approximations of logics such as **L** and **LP**. These logics have been shown not to be *finitely axiomatizable* (Zielonka (1989, 1981)), this means no finite number of combinators together with Modus Ponens can equal their deductive strength.

As noted in Lambek (1988a), Lifting is a closure operation as it enjoys the following properties (we write A^B for either $B/(A\backslash B)$ or $(B/A)\backslash B$):

$$\begin{gathered} A \rightarrow A^B, \\ (A^B)^B \rightarrow A^B, \\ A \rightarrow C, \text{ implies } A^B \rightarrow C^B. \end{gathered}$$

Note that in general $A^B \not\rightarrow A$, which implies that, during a derivation, once a primitive type is lifted it cannot be lowered anymore. By convention, the typeraising of A to $X/(A\backslash X)$ may be written as $A^{l,X}$.

The type-raising laws can be used to lift the proper noun category (n) to the nominal phrase category ($s/(n\backslash s)$), as in Montague (1974) and van Benthem (1983, 1984), also see Dowty (1988) for other examples in linguistic analysis.

The calculus **LP** was introduced in van Benthem (1986a) because of its natural relation with a fragment of the lambda calculus, but there is also linguistic motivation for introducing commutativity. Also see van Benthem (1987).

All permutation closures of context-free languages are recognizable in **LP** (van Benthem (1991)). Also note that the languages expressible in **NL** are precisely the context-free languages (Buszkowski (1986), also see Kandulski (1988)), the same holds for **L** (Pentus (1993b)). These formalisms do not have the necessary expressive power to capture natural languages (which require at

[2]The Curry-Howard correspondence originated in the influential Howard (1980) and Curry and Feys (1958). The former, and a section from the latter, have been reprinted in the collection de Groote and Lamarche (1995), which is highly recommended to anyone interested in this topic. Also see Sørensen and Urzyczyn (1998).

1	Application:	$A/B \bullet B \to A,\ B \bullet B\backslash A \to A$
2	Co-application:	$A \to (A \bullet B)/B,\ A \to B\backslash(B \bullet A)$
3	Monotonicity of $\bullet$:	if $A \to B$ and $C \to D$, then $A \bullet C \to B \bullet D$
4	Isotonicity of $\cdot/C$, $C\backslash\cdot$:	if $A \to B$, then $A/C \to B/C$
		if $A \to B$, then $C\backslash A \to C\backslash B$
5	Antitonicity of $C/\cdot$, $\cdot\backslash C$:	if $A \to B$, then $C/B \to C/A$
		if $A \to B$, then $B\backslash C \to A\backslash C$
6	Lifting:	$A \to B/(A\backslash B),\ A \to (B/A)\backslash B$
7	Geach (main functor):	$A/B \to (A/C)/(B/C)$,
		$B\backslash A \to (C\backslash B)\backslash(C\backslash A)$
8	Geach (secondary functor):	$B/C \to (A/B)\backslash(A/C)$,
		$C\backslash B \to (C\backslash A)/(B\backslash A)$
9	Composition:	$A/B \bullet B/C \to A/C,\ C\backslash B \bullet B\backslash A \to C\backslash A$
10	Restucturing:	$(A\backslash B)/C \leftrightarrow A\backslash(B/C)$
11	(De)Currying:	$A/(B \bullet C) \leftrightarrow (A/C)/B$,
		$(A \bullet B)\backslash C \leftrightarrow B\backslash(A\backslash C)$
12	Permutation:	if $A \to B\backslash C$ then $B \to A\backslash C$
13	Exchange:	$A/B \leftrightarrow B\backslash A$
14	Preposing/Postposing:	$A \to B/(B/A),\ A \to (A\backslash B)\backslash B$
15	Mixed Composition:	$A/B \bullet C\backslash B \to C\backslash A,\ B/C \bullet B\backslash A \to A/C$

Figure 8.1: Characteristic theorems and derived inference rules for **NL** (1-6); **L** (1-11); **NLP** (1-6, 12-14); **LP** (1-15).

least mild context-sensitivity). Therefore more expressive variants have been proposed, for example the multi-modal variant (MMCG) where applicability of postulates is controlled through the use of modal operators in the lexicon. This variant, without restrictions on postulates, is a Turing-complete system (Carpenter (1999)).[3] Recently some restrictions on postulates have been proposed that restrict expressive power to (mild) context-sensitivity, see Moot (2002), we will discuss these in Subsection 8.5.

The presentation of **LP** used here is due to Kurtonina and Moortgat (1997), it takes **NL**$\Diamond$, which is the system **NL** (Figure 8.2) extended with unary connectives (Figure 8.3) and modalities for the binary connectives, as the 'base logic'[4] and adds associativity and commutativity postulates (Figure 8.4).

[3]The proof of this fact is based on an embedding of the full system of multiplicative linear logic in MMCG, which necessitates the introduction of copying and deletion postulates. Such postulates have little linguistic use (see also the discussion in Section 9.6) and do not fit into the program of exploring the *substructural* landscape between the base logic **NL** and **LP**. A remark pertaining to this issue can be found in Lambek (1993).

[4]Note that, unless otherwise stated, the empty sequent is not allowed, i.e. $\vdash A$ may not occur in any derivation. Lambek variants which allow the empty sequent have $\emptyset$ added as subscript, for example **NL** with empty sequent is written as $\mathbf{NL}_\emptyset$. Allowing the empty sequent permits clearly undesirable derivations such as

A	very	book	
$np/n,$	$(n/n)/(n/n),$	n	$\vdash_{\mathbf{L}_\emptyset} np$

however, these calculi are sometimes used in proofs for technical reasons.

$$\frac{}{A \vdash A} \quad \frac{\Delta \vdash A \qquad \Gamma[A] \vdash C}{\Gamma[\Delta] \vdash C}\,[Cut]$$

$$[/I]\,\frac{(\Gamma, B) \vdash A}{\Gamma \vdash A/B} \quad \frac{\Gamma \vdash A/B \qquad \Delta \vdash B}{(\Gamma, \Delta) \vdash A}\,[/E]$$

$$[\backslash I]\,\frac{(B, \Gamma) \vdash A}{\Gamma \vdash B\backslash A} \quad \frac{\Gamma \vdash B \qquad \Delta \vdash B\backslash A}{(\Gamma, \Delta) \vdash A}\,[\backslash E]$$

$$[\bullet I]\,\frac{\Gamma \vdash A \qquad \Delta \vdash B}{(\Gamma, \Delta) \vdash A \bullet B} \quad \frac{\Delta \vdash A \bullet B \qquad \Gamma[(A, B)] \vdash C}{\Gamma[\Delta] \vdash C}\,[\bullet E]$$

Figure 8.2: Sequent-style presentation of the natural deduction rules for **NL** (with product).

$$[L\Diamond_i]\,\frac{\Gamma[\langle A\rangle^i] \vdash C}{\Gamma[\Diamond_i A] \vdash C} \quad \frac{\Gamma \vdash C}{\langle\Gamma\rangle^i \vdash \Diamond_i C}\,[R\Diamond_i]$$

$$[L\Box_i^{\downarrow}]\,\frac{\Gamma[A] \vdash C}{\Gamma[\langle\Box_i^{\downarrow} A\rangle^i] \vdash C} \quad \frac{\langle\Gamma\rangle^i \vdash C}{\Gamma \vdash \Box_i^{\downarrow} C}\,[R\Box_i^{\downarrow}]$$

Figure 8.3: Unary connectives.

This notation facilitates some of the steps in our (syntactic) proofs of non-learnability in Chapter 9, and makes the derivations more explicit.

Each of the systems **NL**, **L**, **LP** and **NLP** have their virtues in linguistic analysis, but in isolation none of them provides a basis for a plausible theory of grammar. By combining properties of these systems in a controlled way, mixed, *multimodal* systems are obtained that overcome the limitations of these simple systems. This multimodal style of reasoning was developed in Moortgat and Morrill (1991), Moortgat and Oehrle (1993), Moortgat and Oehrle (1994) and Hepple (1994), among others.

While the logical approach to categorial grammar is perhaps not really part of the linguistic mainstream, it has however influenced the development of other grammar formalisms: Johnson (1999) gave a reinterpretation of Lexical

$$[comm]\,\frac{(\Gamma, \Delta) \vdash A}{(\Delta, \Gamma) \vdash A} \quad \frac{((\Gamma, \Delta), \Theta) \vdash A}{(\Gamma, (\Delta, \Theta)) \vdash A}\,[ass]$$

Figure 8.4: Postulates for **LP**.

Functional Grammar as a type-logic, Partial Proof Tree Grammars (Joshi and Kulick (1997)), and a multi-modal interpretation of Derivational Minimalism (Stabler (1997)) in Lecomte (2001). There are also formalisms strongly related to Lambek systems, like pregroup grammars (Lambek (1999); Casadio and Lambek (2002)), pomset-logic (Lecomte and Retoré (1995); Schena (1997)), linear logic and proof nets, these are all outside the scope of this thesis and ware only mentioned for completeness.[5]

8.2 Models for the Lambek Calculus

Algebraic interpretations of the Lambek calculi will be used in Chapter 9, in this section the definitions we are interested in[6] are given. Two kinds of models are defined: free groups and powerset residuated groupoids (or semi-groups), a special case of residuated groupoids (see Buszkowski (1997) for details).

Free Group Interpretation. Let FG denote the free group with generators Pr, operation . and neutral element I. We associate with each formula C an element in FG written $[C]$ as follows: $[p] = p$ for p atomic, $[C_1 \backslash C_2] = [C_1]^{-1}.[C_2]$, $[C_1/C_2] = [C_1].[C_2]^{-1}$, $[C_1 \bullet C_2] = [C_1].[C_2]$. We extend the notation to sequents by $[C_1, C_2, \ldots, C_n] = [C_1].[C_2]. \ldots .[C_n]$. The free group FG provides models for L: if $\Gamma \vdash_L C$ then $[\Gamma] =_{FG} [C]$.

Powerset Residuated Groupoids and Semigroups. Let $(M, .)$ be a groupoid, and let $\mathcal{P}(M)$ denote the powerset of M. A *powerset residuated groupoid* over $(M, .)$ is the structure $(\mathcal{P}(M), \circ, \Rightarrow, \Leftarrow, \subseteq)$ such that for $X, Y \subseteq M$:

$$\begin{array}{lcl} X \circ Y & = & \{x.y \mid x \in X, y \in Y\} \\ X \Rightarrow Y & = & \{y \in M \mid (\forall x \in X) x.y \in Y\} \\ Y \Leftarrow X & = & \{y \in M \mid (\forall x \in X) y.x \in Y\} \end{array}$$

If $(M, .)$ is a semi-group (. is associative), then this structure is a *powerset residuated semi-group*. If $(M, .)$ has a unit I (i.e. $\forall x \in M, I.x = x.I = x$) this structure is a *powerset residuated groupoid with unit* (with $\{I\}$ as unit).

Interpretation. Given a powerset residuated groupoid $(\mathcal{P}(M), \circ, \Rightarrow, \Leftarrow, \subseteq)$, an *interpretation* is a map from primitive types p to elements $[\![p]\!]$ in $\mathcal{P}(M)$ that is extended to types and sequences in a natural way:

$$\begin{array}{rcl} [\![C_1 \backslash C_2]\!] & = & [\![C_1]\!] \Rightarrow [\![C_2]\!] \\ [\![C_1/C_2]\!] & = & [\![C_1]\!] \Leftarrow [\![C_2]\!] \\ [\![C_1 \bullet C_2]\!] & = & [\![C_1]\!] \circ [\![C_2]\!] \\ [\![C_1, C_2, \ldots, C_n]\!] & = & [\![C_1]\!] \circ [\![C_2]\!] \circ \ldots \circ [\![C_n]\!] \end{array}$$

[5] Negative learnability results for classes of pregroup grammars can be found in Bechet and Foret (submitted).

[6] There are other models available, most notably the relational model based on Kripke frames, see Kurtonina (1995). These are outside the scope of this thesis and will not be discussed in any detail.

If $(M, .)$ is a groupoid with an identity I, we add $[\![\Lambda]\!] = \{I\}$ for the empty sequence Λ and get a model property for $\mathbf{NL}_\emptyset$: if $\Gamma \vdash_{\mathbf{NL}_\emptyset} C$ then $[\![\Gamma]\!] \subseteq [\![C]\!]$. If $(M, .)$ is a semi-group, we have a similar model property for $\mathbf{L}$: if $\Gamma \vdash_{\mathbf{L}} C$ then $[\![\Gamma]\!] \subseteq [\![C]\!]$.

8.3 Substitution in the Lambek Calculus

It would be natural to use Kanazawa's approach to learnability questions for CCG grammars to Lambek grammars, since these frameworks seem to have so much in common. In order to do this, precise notions of substitution (unification) as well as a notion of structure language need to be defined. The latter will be dealt with in the next section. This section addresses the former, by discussing work from Foret (2001a,b).

Definition 8.3 *The* join-equivalence*, written $\sim$, is defined as*

$$t \sim t' \text{ if and only if } \exists t_1, \ldots, t_n \text{ such that } t = t_1, t' = t_n, (t_i \vdash t_{i+1} \vee t_{i+1} \vdash t_i),$$

for $i < n$. Types $t_1, \ldots, t_n$ are said to be conjoinable *whenever there is a type t such that $t_i \vdash t$, for each $i \leq n$. In this case t is said to be a* join *for $t_1, \ldots, t_n$.*

The following proposition, defining what is known as the *Diamond property*, is from Lambek (1958):[7].

Proposition 8.4 *Let t_1 and t_2 be two categorial types. The following statements are equivalent:*

1. *t_1 and t_2 are conjoinable.*
2. *there exists a type t' such that $t' \vdash t_1$ and $t' \vdash t_2$.*

Proposition 8.5 *Let t_1 and t_2 be two categorial types. The statement $t_1 \sim t_2$ is equivalent to 1 and 2 of Proposition 8.4.*

We will use this proposition combined with the following completeness result by Pentus which characterizes $\sim$ by groups:

Theorem 8.6 *For any two types t, t', $t \sim [\![t']\!] \leftrightarrow =_{FG} [\![t']\!]$.*

Definition 8.7 *The relation $\|=$ on types t_1, t_2 is defined by:*

$$t_1 \|= t_2 \text{ if and only if there is a substitution } \sigma \text{ such that } t_1 \vdash_{\mathbf{L}} \sigma[t_2].$$

Note that $\|=$ is reflexive and transitive.

[7] An alternative proof of this proposition can be found in Pentus (1993a).

8.3.1 $\|=$-Unifiability

Definition 8.8 *Two types t_1, t_2 are said to be* $\|=$-unifiable *whenever there exists a type t and substitution σ such that $t \vdash \sigma[t_1]$ and $t \vdash \sigma[t_2]$.*

The substitution σ is said to be a $\|=$-unifier *of t_1 and t_2 and t is a* $\|=$-unificand *of t_1 and t_2.*

There are strong links between $\|=$-unification and E-unification, i.e. unification under an equational theory.[8] Depending on the nature of the equational theory, E-unification of two terms can yield an infinite number of mgu's (for example under associativity), and even when there are only finitely many mgu's, finding them may be a computationally difficult task.

Therefore, a naive application of Kanazawa's approach to classes of Lambek grammars yields a host of problems. In fact, as we shall see in Chapter 9, even the rigid subclasses of **L**, **NL** and **LP** are not learnable (from strings).[9]

8.4 The Structure Languages of Non-Associative Lambek Grammars

In Tiede (1999a, 1998) a notion of structure language (i.e., strong generative capacity or SGP) for both **L** and **NL** was proposed. This is a highly non-trivial matter, since in the context of a logical grammar formalism derivations are *proofs*, and proofs are generally (spuriously) nondeterministic.[10]

His approach differs from the one found in Buszkowski (1997) which defines functor-argument structures (so-called f-structures) and phrase structures (p-structures). In the case of Lambek grammars this definition has undesirable consequences, most notably the structural completeness theorem: any (associative) Lambek grammar that generates string s can assign any binary tree that has $|s|$ (non-ε) leaves as structural description for s.[11] This fact has caused some researchers to turn their attention to **NL**-grammars, since these

[8] E-unification plays an important role in automated theorem provers with 'built-in' theories (see e.g., Plotkin (1972)), Stickel (1985)) and in logic programming with equality (see e.g., Jaffar et al. (1984)). See Baader and Siekmann (1993) for a comprehensive overview.

[9] In this context we'd also like to mention the exploratory work in Moortgat (2001), where an algorithm is proposed that takes the unification approach 'as far as possible' and, in the case that no rigid grammar can be obtained this way, invents postulates that equate non-unifiable types assigned to the same symbol. There are as of yet no learnability results or independent characterization of learnable classes in relation tho this approach.

[10] One way of dealing with this spurious non-determinism is the use of *proofnets*, see eg. Moot (2002). However, for the present purposes proofnets are not very useful: depending on the calculus one proofnet may be applicable to several proofs that are non-equivalent, for example when product is used.

They also do not offer a (tree) automata-theoretic perspective, since they produce graphs, and, to the best of the author's knowledge, there exists little work on learning graph languages.

[11] This is easy to see; lifting allows reversal of the direction of application, and associativity allows any rearrangement of the derivation tree.

are not structurally complete. However, Buszkowski (1997) also shows that **NL**-grammars cannot extend the strong generative capacity of context-free grammars, which is unfortunate given the current trend in computational linguistics to 'squeeze more strong power out of a formal system without increasing its weak generative power'.

In Tiede (1999a, 1998) it was shown that considering (the normal forms of) *natural deduction proof trees* as the structures assigned by Lambek grammars yields a notion of SGP that goes beyond that of context-free grammars[12] (and may allow characterization of *crossing dependencies*), but does not suffer from a collapse into structural completeness.[13] Another pleasant property is that this notion of SGP distinguishes between proofs if and only if the semantic term they produce differs (cf Hendriks (1993)), just like proofnets.

A direct definition of natural deduction proof trees for **L** would involve either defining proof trees to contain parenthesised structures or giving a complicated list of side conditions for the introduction rules. The tree format will therefore be defined *indirectly* by defining a translation from SND to to natural deduction trees. Well-formed trees for **NL** will be defined as a subset of well-formed trees for **L**.

Definition 8.9 *Let t be a proof tree of the associative Lambek calculus. We define its translation into a proof in SND, t^*, as follows:*

1. *if $t = A$, i.e. t is an instance of $[ID]$, then $t^* = A \vdash A$,*

2. *if* $t = \dfrac{t_1 \quad t_2}{A}\ [/E]$ *where* $t_1 = \dfrac{\vdots}{A/B}$ *and* $t_2 = \dfrac{\vdots}{B}$.

   We have $t_1^* = \dfrac{\vdots}{\Gamma \vdash A/B}$ *and* $t_2^* = \dfrac{\vdots}{\Delta \vdash B}$, *where* Γ *and* Δ *are the uncancelled assumptions of* t_1 *and* t_2, *respectively, in the order in which they occur. We define* $t^* = \dfrac{t_1^* \quad t_2^*}{(\Gamma, \Delta) \vdash A}\ [/E]$.

3. *If* $t = \dfrac{t_1}{A/B}\ [/I]$, *where* $t_1 = \dfrac{\begin{array}{c} B \\ \vdots \\ A \end{array}}{}$, *then consider* $t_1^* = \dfrac{\vdots}{(\Gamma, B) \vdash A}$. *We define* $t^* = \dfrac{t_1^*}{\Gamma \vdash A/B}\ [/I]$.

4. $[\backslash E]$ *and* $[\backslash I]$ *are defined analogously.*

[12] In van Benthem (1991) the possibility of extended SGP was mentioned as a motivation for studying Lambek grammars; the *weak* generative capacity of CCG, **L** and **NL** is exactly that of context-free grammars.

[13] But see Joshi (2002) for a critique; the crossing dependencies allowed by this definition of SGP are 'very degenerate', i.e. they can only connect a lexical item with a lexically empty element.

Definition 8.10 *A natural deduction proof tree t is a well-formed proof of* **NL** *if its translation t^* (according to Definition 8.9) is a well-formed proof according to Figure 8.2.*

Since there are strong non-learnability results for **L**, we will only be concerned with **NL** in Chapter 9.

Definition 8.11 *A natural deduction proof tree is* in (β-η-)normal form *(or* (β-η-)normal*) if none of its subtrees are of the form of any of the following trees:*

$$\dfrac{\dfrac{\begin{array}{c}[B]\\ \vdots\\ A\end{array}}{A/B}\,[/I] \quad B}{A}\,[/E] \qquad \dfrac{B \quad \dfrac{\begin{array}{c}[B]\\ \vdots\\ A\end{array}}{B\backslash A}\,[\backslash I]}{A}\,[\backslash E] \qquad \dfrac{\dfrac{A/B \quad [B]}{A}\,[/E]}{A/B}\,[/I] \qquad \dfrac{[B] \quad \dfrac{B\backslash A \quad [B]}{A}\,[\backslash E]}{B\backslash A}\,[\backslash I]$$

Note that every proof tree can be converted into a unique normal form proof tree. The following proposition concerns the very important *subformula property*:

Proposition 8.12 ***Subformula property*** *Every formula that occurs in a normal form natural deduction proof or cut-free sequent calculus proof is either a subformula of the uncanceled assumptions or of the conclusion.*

Definition 8.13 *A* track *of a proof tree T is a sequence of formulae $A_0, \ldots, A_n$ such that*

1. *A_0 is a leaf of T,*
2. *for $0 \leq i < n$,*
   (a) *A_{i+1} is immediately below A_i,*
   (b) *A_i is not the minor premise of an $[\backslash E]$ or $[/E]$ application,*
3. *T is maximal, i.e. not a proper part of another track.*

Note that in a normal proof every formula occurence belongs to a track.

The following result has a well-known analogy in the field of linear logic:

Proposition 8.14 *(Also see Prop 2.28 in Tiede (1999a)) Every track that is part of an* **L** *or* **NL** *normal form proof tree begins*[14] *with an E-part, i.e. $A_0, \ldots, A_{i-1}$, has a minimal formula A_i after the E-part and may have an I-part after that, $A_{i+1}, \ldots, A_n$.*

[14] When read top-down.

Proposition 8.15 *The number of different tracks occuring in normal form proofs for a given* **L** *or* **NL** *grammar is finite.*

Definition 8.16 *An* incomplete proof tree *is either*

1. *a track, or*
2. *an incomplete proof tree with a track t inserted such that t starts with formula A and the insertion point is the minor premise of an application of $[/E]$ or $[\backslash E]$, where the major premise is B/A or $A\backslash B$, respectively.*

Definition 8.17 *An incomplete proof tree is* saturated *just if for every occurence of $[/I]$ or $[\backslash I]$ in the tree has a corresponding leaf in the tree that it cancels.*

Definition 8.18 *A saturated incomplete proof tree (or* **NL***-track) is* minimal *just if*

1. *it is a track that contains no introductions, or*
2. *a saturated incomplete proof tree T that contains introduction rules, such that removing any subtree that is an incomplete proof tree would cause T to not be saturated.*

Proposition 8.19 *The number of minimal saturated incomplete proof trees to which a track in a normal form proof tree for an* **NL** *grammar can be extended is finite.*

Since **NL**-tracks behave just like tracks in classical categorial grammar, Kanazawa's approach can be applied in a straightforward way, as we shall see in Chapter 9.

8.5 Multimodal Systems

As was noted in the beginning of this chapter, it is possible to control structural relaxation with the use of modal operators. However, the resulting calculi can be overly expressive,[15] to the point of undecidability. In Moot (2002) a natural restriction on structural rules is proposed that turns out to restrict expressive power to more linguistically plausible levels.

The restriction requires that the left-hand side of a structural conversion contains at least as many unary connectives as the right-hand side, the resulting postulates are called *non-expanding*. First 'length' is defined in terms of unary connectives:

[15] See Kurtonina and Moortgat (1997) for the relations between the different systems obtained by the use of such operators.

Aso see Jäger (1998) for results on the *strong* generative capacity of multimodal systems.

Definition 8.20 *Given an antecedent Ξ, its length is defined as*

$$\begin{array}{lcl} length(\Delta_1 \circ \Delta_2) & = & length(\Delta_1) + length(\Delta_2) + 2 \\ length(\langle\Delta\rangle^i) & = & length(\Delta) + 1 \\ length(\Delta) & = & 0 \end{array}$$

This definition can be generalized over n-ary configurations: $length(*(\Delta_1, \ldots, \Delta_n)) = length(\Delta_1) + \ldots + length(\Delta_n) + n$.

Definition 8.21 *the logic $\mathbf{NL}\Diamond_{\mathcal{R}-}$ is the logic $\mathbf{NL}\Diamond_{\mathcal{R}}$ where for every structural rule $R \in \mathcal{R}$*

$$\frac{\Gamma[\Xi'[\Delta_1, \ldots, \Delta_n]] \vdash C}{\Gamma[\Xi[\Delta_{\pi_1}, \ldots, \Delta_{\pi_n}]] \vdash C}\ [R]$$

the following holds:

$$length(\Xi[\Delta_{\pi_1}, \ldots, \Delta_{\pi_n}]) \leq length(\Xi'[\Delta_1, \ldots, \Delta_n])$$

Structural rules with this property are called non-expanding.

Note that in the case of binary modalities this restricts postulates to be *linear*, and that a structural rule for -for example- strong distributivity is non-expanding according to this definition.

An embedding for lexicalized context-sensitive grammars is offered in Moot (2002):

Definition 8.22 *From a lexicalized context-sensitive grammar G we generate the corresponding multimodal Lambek calculus $\mathcal{M}(G)$ as follows: $\mathcal{M}(G)$ has one unary mode for every nonterminal of G and a single binary mode.*

Every lexicalization rule $A \mapsto \beta$ correspond to a lexical entry of the form

$$lex(\beta) = a \backslash \Box^{\downarrow}_{A} a$$

The goal formula of $\mathcal{M}(G)$ is $a\backslash\Box^{\downarrow}_{S}a$ where S is the mode corresponding to the start symbol of G. The calculus $\mathcal{M}(G)$ has a structural rule for every grammar rule $A_1 \ldots A_n \mapsto_{R1} B_1 \ldots B_m$ of G.

$$\frac{\Gamma[\langle \ldots \langle\Delta\rangle^{A_n} \ldots\rangle^{A_1}] \vdash C}{\Gamma[\langle \ldots \langle\Delta\rangle^{B_m} \ldots\rangle^{B_1}] \vdash C}\ [R1]$$

Because $m > 0$ and $n \leq m$, ths is a valid $\mathbf{NL}\Diamond_{\mathcal{R}-}$ rule.

Furthermore, the structural rule component of $\mathcal{M}(G)$ contains one of the structural rules for associativity for the single binary mode:

$$\frac{\Gamma[\Delta_1 \circ (\Delta_2 \circ \Delta_3)] \vdash C}{\Gamma[(\Delta_1 \circ \Delta_2) \circ \Delta_3] \vdash C}\ [Ass2]$$

and the structural rule of $[K1]$ for every mode $A \in N$:

$$\frac{\Gamma[\langle\Delta_1\rangle^A \circ \Delta_2] \vdash C}{\Gamma[\langle\Delta_1 \circ \Delta_2\rangle^A] \vdash C}\ [K1]$$

Lemma 8.23 *Given a lexicalized context-sensitive grammar G, the corresponding (by Definition 8.22) multimodal Lambek calculus $\mathcal{M}(G)$ generates the same language as G.*

Theorem 8.24 *The parsing problem for $\mathbf{NL}\Diamond_{\mathcal{R}-}$ is quivalent to the parsing problem for context-sensitive grammars.*

Note that, although these are obviously interesting results, the grammars obtained through this embedding are in some sense unnatural. They basically abuse postulates by treating them as rules, resulting in a rather circuitous way of using a rule-based system. We will use this embedding in the next chapter to define a learnable class of multimodal CG, but this should be regarded as a quite trivial result.

Identity

$$\frac{}{A \vdash A}\ [Ax] \qquad \frac{\Gamma[B] \vdash C \quad \Delta \vdash B}{\Gamma[\Delta] \vdash C}\ [Cut]$$

Unary Connectives

$$\frac{\Gamma[\langle A \rangle^i] \vdash C}{\Gamma[\Diamond_i A] \vdash \Diamond_i C}\ [L\Diamond_i] \qquad \frac{\Gamma \vdash C}{\langle \Gamma \rangle^i \vdash C}\ [R\Diamond_i]$$

$$\frac{\Gamma[A] \vdash C}{\Gamma[\langle \Box_i A \rangle^i] \vdash C}\ [L\Box_i^{\downarrow}] \qquad \frac{\langle \Gamma \rangle^i \vdash C}{\Gamma \vdash \Box_i C}\ [R\Box_i^{\downarrow}]$$

Binary Connectives

$$\frac{\Gamma[A \circ_i B] \vdash C}{\Gamma[A \bullet_i B] \vdash C}\ [L\bullet_i] \qquad \frac{\Gamma \vdash A \quad \Delta \vdash B}{\Gamma \circ_i \Delta \vdash A \bullet_i B}\ [R\bullet_i]$$

$$\frac{\Delta \vdash B \quad \Gamma[A] \vdash C}{\Gamma[A/_i B \circ_i \Delta] \vdash C}\ [L/_i] \qquad \frac{\Gamma \circ_i B \vdash A}{\Gamma \vdash A/_i B}\ [R/_i]$$

$$\frac{\Delta \vdash B \quad \Gamma[A] \vdash C}{\Gamma[\Delta \circ_i B \backslash_i A] \vdash C}\ [L\backslash_i] \qquad \frac{B \circ_i \Gamma \vdash A}{\Gamma \vdash B \backslash_i A}\ [R\backslash_i]$$

Structural Rules

$$\frac{\Gamma[\Xi'[\Delta_1, \ldots, \Delta_n]] \vdash C}{\Gamma[\Xi[\Delta_{\pi_1}, \ldots, \Delta_{\pi_n}]] \vdash C}\ [SR]$$

Figure 8.5: The sequent calculus $\mathbf{NL}\Diamond_{\mathcal{R}}$.

Chapter 9

Learning Lambek Grammars

9.1 Introduction

This chapter will discuss learnability results for a number of classes of Lambek grammars.[1] Even though Lambek systems have a superficial resemblance to CCG grammars, and contrary to what Shinohara's results on EFSs suggest, they turn out to have completely different learnability properties: in Foret and Le Nir (2002a,b) it was shown that the classes of $\mathbf{L}$, $\mathbf{L}_\emptyset$ and $\mathbf{NL}_\emptyset$ (and some variants) k-valued grammars are not learnable.[2] Restating these strong results in their full generality:

Proposition 9.1 *The classes of (string)languages generated by k-valued* $\mathbf{L}$ *grammars with $k \geq 1$, $|\Sigma| \geq 3$, for all types T in these grammars,* $\mathrm{order}(T) \geq 2$*, and the use of operators restricted to either $(/, \bullet)$ or $(\backslash, \bullet)$, have a limit point and are thus not non-effectively learnable from strings.*

Proposition 9.2 *The classes of (string)languages generated by k-valued* $\mathbf{L}_\emptyset$ *grammars with $k \geq 1$, $|\Sigma| \geq 3$, and the use of operators restricted to either $/$ or $\backslash$, have a limit point and are thus not non-effectively learnable from strings.*

We will provide their complete proofs for these two propositions, and give our own proofs for non-learnability of rigid $\mathbf{LP}$ and $\mathbf{LP}_\emptyset$ grammars, originally published in Costa Florêncio (2003).

This chapter is organized as follows: first, Subsection 9.2.1 defines some learnable subclasses of $\mathcal{G}\mathbf{NL}$. We then present two main results on variants of Lambek calculus: Section 9.3 gives a construction and a proof of the existence

[1] Some of the material in this chapter has previously appeared as Costa Florêncio (2003), it is reproduced with permission.

[2] In Bechet and Foret (submitted) a limit point construction for rigid $\mathbf{NL}$ grammars is given. However, since this proof has not yet been published it cannot be discussed here in detail.

of a limit point for rigid non-associative Lambek grammars allowing empty sequences. Section 9.4 adresses the construction for Lambek grammars without product and without empty sequences. In Section 9.5 the nonlearnability of rigid **LP** and $\mathbf{LP}_\emptyset$ will be demonstrated. Section 9.6 discusses learnability in the context of the Lambek calculus extended with structural rules like contraction, Section 9.7 does the same for a particular generalization of the Lambek calculus, and Section 9.8 adresses learnability of classes of Lambek grammars that contain polymorphic types. Section 9.9 concludes this chapter.

9.2 Learning NL-Grammars from Structures?

As we have seen in Section 8.4, it is possible to give a useful definition of structure language for **NL**-grammars. As suggested in Tiede (1999a) it may be used to investigate the learnability of classes of **NL**-grammars from structures. For example, it is easy to adapt the notion of general form to **NL**:[3]

Definition 9.3 *The* general form *of a structure is defined inductively as*

1. $\mathrm{GF}([ID](\texttt{symbol}) \Rightarrow T)$ *implies* $\texttt{symbol} \mapsto T$, *(note that* $\mathrm{GF}([ID](\epsilon) \Rightarrow T)$ *is acceptable but will not lead to any assignments),*
2. $\mathrm{GF}([\backslash E](Minor, Major) \Rightarrow T)$ *implies* $\mathrm{GF}(Minor \Rightarrow T_{minor})$, $\mathrm{GF}(Major \Rightarrow T_{major})$, *and* $T = T_{minor} \backslash T_{major}$,
3. $\mathrm{GF}([/E](Major, Minor) \Rightarrow T)$ *implies* $\mathrm{GF}(Minor \Rightarrow T_{minor})$, $\mathrm{GF}(Major \Rightarrow T_{major})$, *and* $T = T_{major}/T_{minor}$,
4. $\mathrm{GF}([\backslash I](Premise) \Rightarrow T)$ *implies* $T = T_{hyp} \backslash T_{new}$, $\mathrm{GF}(Premise \Rightarrow T_{new})$,
5. $\mathrm{GF}([/I](Premise) \Rightarrow T)$ *implies* $T = T_{new}/T_{hyp}$, $\mathrm{GF}(Premise \Rightarrow T_{new})$

Lemma 9.4 $\mathrm{FL}(\mathrm{GF}(D)) = D$.

`Proof:` By the definition of GF, each E-step has a unique premise and conclusion. The I-steps unify the unique premise of the succeeding step with $T_1|T_2$, where T_2 is a unique type (introduced as hypothesis), and T_1 is the unique premise of that I-step. Since the construction of the types in $\mathrm{GF}(D)$ is determined by the premise/conclusion pairs of all the steps of the proof trees in D, all the range and domain occurences of these types are unique. □

Note that the Lambek version of $\mathrm{GF}(D)$, unlike the classical version, may assign types that have a complex domain subtype.

Lemma 9.5 $D \subseteq \mathrm{FL}(G)$ *is equivalent to: there is a substitution* σ *such that* $\sigma[\mathrm{GF}(D)] \subseteq G$.

[3] Note that this version of GF may just as well be used for *non* normal form proofs, however.

`Proof:`

1. $D \subseteq \mathrm{FL}(G)$ implies that there is a substitution σ such that $\sigma[\mathrm{GF}(D)] \subseteq G$:

   Since $D \subseteq \mathrm{FL}(G)$, every tree in D is accepted by G. The formula-labeling imposed by G may differ from the (implicit) formula-labeling in D, but only in the sense that it is a substitution instance of the latter. Thus there must be a substitution over $\mathrm{GF}(D)$ that assigns the same formula-labeling as G does.

2. The existence of a substitution σ such that $\sigma[\mathrm{GF}(D)] \subseteq G$ implies $D \subseteq \mathrm{FL}(G)$:

   Since $\mathrm{FL}(G) \subseteq \mathrm{FL}(\sigma[G])$, $\sigma[\mathrm{GF}(D)] \subseteq G$ implies $\mathrm{FL}(\sigma[\mathrm{GF}(D)]) \subseteq \mathrm{FL}(G)$. Since $D \subseteq \mathrm{FL}(\sigma[\mathrm{GF}(D)])$, $D \subseteq \mathrm{FL}(G)$ follows.

□

As we have seen in Section 8.3, unification of types in **L** or **NL** is much more complicated than the Robinson-style unification that can be applied straightforwardly to CCG-types. Coming up with algorithms that learn Lambek grammars from structures is therefore expected to be much more challenging. Even when a collection of such grammars is shown to be learnable, finite elasticity may yet be another matter, which makes it much harder to extend such a result to a wider class or to learning from strings (using Theorem 2.27). We will therefore examine a number of possible approaches in the following subsection.

A positive result for learning $\mathcal{G}_{\mathrm{rigid}}\mathbf{NL}$ from *structures* is hinted at in Bonato and Retoré (2001). Other learnability results can be obtained for a subclass of $\mathcal{G}_{\mathbf{NL}}$: by bringing any CFG into Chomsky normal form, the resulting grammar can be interpreted as a CCG, an **NL** and even an **L** grammar, because all lexical types are types with order of at most 1, so the construction is the same for all kinds of CGs. This fact can be used to easily come up with different kinds of learnable classes of **NL**-grammars, but we will not pursue this line of inquiry further in this chapter.

9.2.1 Learnable Classes of NL-Grammars

Perhaps surprisingly, the class $\mathcal{G}_{\mathrm{rigid}}\mathbf{NL}$ is known to have a limit point and thus to be not (non-effectively) learnable, as mentioned before. However, given the results discussed in previous chapters it is quite easy to come up with a learnable subclass of $\mathcal{G}\mathbf{NL}$. First let prune be (somewhat informally) defined as the function that, given a set of normal form derivations, yields a set comprised of trees that correspond to these derivations, but without node labeling, without the (unary) introduction nodes and without paths that lead to introduced types (corresponding to ε in the yield of the derivation). Then, we define this subclass analogous to Definition 7.21:

Definition 9.6 *Let $\mathcal{G}_{\mathcal{R}}\mathbf{NL}$ be the smallest class of* **NL***-grammars such that for every reversible tree language L_1 there is a language $L = \mathrm{FL}(G), G \in \mathcal{G}_{\mathcal{R}}\mathbf{NL}$ such that $L_1 = \mathrm{prune}(L)$.*[4]

And, again by Theorem 7.5, the following result is obtained:

Proposition 9.7 *The class $\mathcal{G}_{\mathcal{R}}\mathbf{NL}$ is learnable from structures.*

This class is linguistically not very interesting, however. As pointed out in Kanazawa (1998) (page 130, footnote 4), the class of CCGs whose associated context-free grammars are reversible is properly included by $\mathcal{G}_{\mathrm{rigid}}$ and properly includes the class of unidirectional rigid CCGs, and thus does not include languages of even remotely sufficient expressive power to model natural language.

There are other ways of 'engineering' learnable classes of **NL**-grammars, a trivial example would be bounding the (maximum) degree of the types occuring in the grammars, since this would yield a finite class. A more interesting approach would be to bound the number of distinct states in the tree automata corresponding to the pruned(normal) tree languages of **NL**-grammars, for example by the number of symbols occuring in the grammar, i.e. $|\Sigma|$. This way one can 'artificially' impose structural properties on subclasses of $\mathcal{G}\mathbf{NL}$ as those that $\mathcal{FL}_{\mathrm{rigid}}$ has been demonstrated to have, namely bounded elasticity (See Theorem 4.5 and Lemma 4.6). Let the associated classes be denoted by $\mathcal{FL}_{\mathrm{bounded}}$ and $\mathcal{G}_{\mathrm{bounded}}$.

Proposition 9.8 *The class $\mathcal{FL}_{\mathrm{bounded}}$ has finite elasticity.*

Since the relation between string language and pruned normal tree language is finite-valued, Theorem 2.27 can be applied,

Proposition 9.9 *The class $\mathcal{G}_{\mathrm{bounded}}$ is learnable from strings.*

We mention one more approach, this time for learning classes of multi-modal CG. The result follows trivially, and is really no more than a cheap trick: since we have Theorem 2.43, Shinohara's result on length-bounded context-sensitive grammars, the embedding from Definition 8.22 easily yields a learnability result. Since this embedding is bijective (modulo alphabetic variation), it fulfills the conditions of Theorem 2.27, therefore

Corollary 9.10 *The class of rigid $\mathbf{NL}\Diamond_{\mathcal{R}-}$ grammars that assign just $a\backslash\Box^{\downarrow}_{A}a$ to every symbol in their alphabet, have a bounded number of rules of the form*

$$\frac{\Gamma[\langle \ldots \langle\Delta\rangle^{A_n} \ldots\rangle^{A_1}] \vdash C}{\Gamma[\langle \ldots \langle\Delta\rangle^{B_m} \ldots\rangle^{B_1}] \vdash C}\ [R1]$$

[4] An alternative (equivalent) definition could be given along the lines of 'the largest class of **NL**-grammars such that for every grammar it contains, the set of all the derivations allowed by its corresponding ε-free (Greibach normal-form) context-free grammar forms a reversible tree language'.

and

$$\frac{\Gamma[\Delta_1 \circ (\Delta_2 \circ \Delta_3)] \vdash C}{\Gamma[(\Delta_1 \circ \Delta_2) \circ \Delta_3] \vdash C}\ [Ass2]$$

and the structural rule of $[K1]$ *for every mode* $A \in N$*:*

$$\frac{\Gamma[\langle \Delta_1 \rangle^A \circ \Delta_2] \vdash C}{\Gamma[\langle \Delta_1 \circ \Delta_2 \rangle^A] \vdash C}\ [K1]$$

has finite elasticity and is therefore learnable from strings.

9.3 A Limit Point for Rigid $\mathbf{NL}_\emptyset$ Grammars

9.3.1 Construction Overview

Definition 9.11 *Let* $G_{\langle 1,n\rangle} = \{a \mapsto p/p, c \mapsto D_{\langle 1,n\rangle}\}$*, where* $D_{\langle 1,0\rangle} = S$ *and* $D_{\langle 1,n\rangle} = D_{\langle 1,n-1\rangle}/(p/p)$*, and* $G_{\langle 1,*\rangle} = \{a \mapsto p/p, c \mapsto S/(p/p)\}$*, where* p *and* S *are primitive types.*

Language. We get (see proof) $\mathrm{L}(G_{\langle 1,n\rangle}) = \{ca^k (0 \leq k \leq n)\}$ and $\mathrm{L}(G_{\langle 1,*\rangle})$ $= ca^*$.

Notation. Let $\tau_{\langle 1,n\rangle}$ ($\tau_{\langle 1,*\rangle}$) denote the type assignment by $G_{\langle 1,n\rangle}$ ($G_{\langle 1,*\rangle}$, respectively), on $\{a, c\}$ extended to $\{a, c\}^*$. We write $\tau = \tau_{\langle 1,n\rangle}$ on $\{a\}^*$ (independent of $n \geq 0$).

Key Points. Tautologies of the Lambek calculus are used that allow empty sequences that ensure one way of type-derivability ($D_{\langle 1,n\rangle} \vdash D_{\langle 1,n-1\rangle}$). Note that in contrast to Foret and Le Nir (2002a)'s treatment for the associative calculus **L**, an alternation effect is not needed: non-associativity is enough to block derivations such as ($D_{\langle 1,n-1\rangle} \nvdash D_{\langle 1,n\rangle}$). We thus provide a strictly infinite chain of types for $\mathbf{NL}_\emptyset$ with respect to $\vdash$.

9.3.2 Corollaries

For the Class of Rigid $\mathbf{NL}_\emptyset$-grammars. This class has a limit point ($c\{a\}^*$) which entails that this class is not learnable from strings.

The same results hold if we restrict to a bounded order. In fact order($G_{\langle 1,n\rangle}$) in this construction is not greater than 2. This result also holds for the subclass of unidirectional grammars.

9.3.3 Details of Proofs

The proof is based on syntactic reasoning on both derivations and on models.

Proposition 9.12 *(Language description)* $\mathrm{L}(G_{\langle 1,n\rangle}) = \{ca^k \mid 0 \leq k \leq n\}$ *and* $\mathrm{L}(G_{\langle 1,*\rangle}) = ca^*$.

For ease of proof, we introduce the following operations: for a word $w = c_1c_2c_3\dots c_{k-1}c_k$, where c_i denote symbols, $l(w)$ is the left bracketed version of w, i.e. $l(w) = ((\dots((c_1c_2)c_3)\dots c_{k-1})c_k)$; the same applies to sequences of types $\Gamma = (A_1, A_2, A_3\dots, A_{k-1}, A_k) : l(\Gamma) = ((\dots((A_1, A_2), A_3)\dots, A_{k-1}), A_k)$, i.e. the left bracketed version of the sequence. We define r on words $(r(w))$ and type sequences($r(\Gamma)$) for the right bracketed versions of these structures.

`Proof:` of $\{ca^k \,|\, 0 \leq k \leq n\} \subseteq \mathrm{L}(G_{\langle 1,n\rangle})$: we show the following left bracketed version of this property, by induction on n: $\forall k, 0 \leq k \leq n, l(\tau_{\langle 1,n\rangle}(ca^k)) \vdash S$. For $n = 0$ this is an axiom: $\tau_{\langle 1,0\rangle}(c) = S \vdash S$.

Suppose $n > 0$ and $w' = c.w$ with $w \in \{a^*\}$ and $l(\tau_{\langle 1,n-1\rangle}(cw)) \vdash S$. First it will be shown that $l(\tau_{\langle 1,n\rangle}(c.a.w)) \vdash S$:

$$\underbrace{\dfrac{\dfrac{\vdots}{l(D_{\langle 1,n-1\rangle}, \tau(w)) \vdash S} \quad p/p \vdash p/p}{l(\underbrace{(D_{\langle 1,n-1\rangle}/(p/p)), (p/p)}, \tau(w)) \vdash S}}_{=l(\tau_{\langle 1,n\rangle}(c.a.w))}$$

-we easily get $l(\tau_{\langle 1,n\rangle}(c.w)) \vdash S$: first we have $D_{\langle 1,n\rangle} \vdash D_{\langle 1,n-1\rangle}$ in $\mathbf{NL}_\emptyset$ for $n > 0$:

$$\dfrac{\dfrac{p \vdash p}{\emptyset \vdash p/p} \quad D_{\langle 1,n-1\rangle} \vdash D_{\langle 1,n-1\rangle}}{D_{\langle 1,n-1\rangle}/(p/p) \vdash D_{\langle 1,n-1\rangle}}$$

applying the Cut rule to $D_{\langle 1,n-1\rangle}$: $\underbrace{l(\tau_{\langle 1,n\rangle}(c.w))}_{=l(D_{\langle 1,n\rangle}, \tau(w))} \vdash S$ from $\underbrace{l(_{\langle 1,n-1\rangle}(c.w))}_{=l(D_{\langle 1,n-1\rangle}, \tau(w))} \vdash S$

`-Proof:` of $\mathrm{L}(G_{\langle 1,n\rangle}) \subseteq \{ca^k / 0 \leq k \leq n\}$ (main part):

We consider a powerset residuated groupoid $(\mathcal{P}(M), \circ, \Rightarrow, \Leftarrow, \subseteq)$ over the groupoid $(M, .)$ where . is the concatenation operation and M is the set of bracketed strings over the alphabet $V = \{a, c\}$ with unit ε (empty word). Let us fix n (arbitrarily), we define an interpretation as follows: $[\![S]\!] = \{l(ca^k) \,|\, k \leq n\}$, $[\![p]\!] = \{r(a^k) \mid 0 \leq k\}$.

Note that $[\![p/p]\!] = \{\varepsilon, a\}$ (since $[\![p/p]\!] = \{z \in M \mid \forall x \in [\![p]\!], (z.x) \in [\![p]\!]\} = \{z \in M \mid \forall j, (z.r(a^j)) \in \{r(a^k) \mid 0 \leq k\}\}$).

By induction on i it can be shown that $\forall i, 0 \leq i \leq n, [\![D_{\langle 1,i\rangle}]\!] = \{l(c.a^k) \,|\, k \leq (n-i)\}$

case $i = 0 \leq n$ holds since $[\![D_{\langle 1,0\rangle}]\!] = [\![S]\!] = \{l(c.a^k) \mid 0 \leq k \leq n\}$

case$(0 < i \leq n)$: $[\![D_{\langle 1,i-1\rangle}/(p/p)]\!] =$
$\{z \in M \mid \forall x \in [\![p/p]\!], (z.x) \in [\![D_{\langle 1,i-1\rangle}]\!]\} =_{ind.} \{z \in M \mid \forall x \in \{\varepsilon, a\}, (z.x) \in \{l(c.a^k) \mid 0 \leq k \leq (n-(i-1))\}\} = \{z \in M \,|\, z \in \{l(c.a^k) \mid k \leq (n-i+1)\}$ and $(z.a) \in \{l(c.a^k) \mid k \leq (n-i+1)\}\} = \{l(c.a^k) \mid k \leq (n-i)\}$ □

This shows that $[\![D_{\langle 1,n\rangle}]\!] = \{c\}$.

Remark. Note that for each $w \in \{a, c\}$, $l(w) \in [\![l(\tau_{\langle 1,n\rangle}\tau(w))]\!]$. This holds for all atomic words since $l(a) = a \in [\![l(\tau_{\langle 1,n\rangle}\tau(w))]\!] = [\![l(p/p)]\!] =$

$[\![p/p]\!] = \{\varepsilon, a\}$ and $l(c) = c \in [\![l(\tau_{\langle 1,n\rangle}(c))]\!] = [\![D_{\langle 1,n\rangle}]\!] = \{c\}$. For compound types we have $[\![l(\tau_{\langle 1,n\rangle}(c_1 \dots c_n))]\!] = [\![l(\tau_{\langle 1,n\rangle}(c_1)) \dots \tau_{\langle 1,n\rangle}(c_k))]\!]$ which includes $l(c_1, c_2 \dots c_k) = ((c_1 \cdot 2) \dots c_k)$ (where all c_i are atomic).

Let us suppose $\Gamma \vdash S$, where Γ is a bracketed version of $\tau_{\langle 1,n\rangle}(w)$. By models, we have $[\![\Gamma]\!] \subseteq [\![S]\!]$. Since $[\![S]\!] = \{l(c.a^k) \mid k \leq n\}$ has only left bracketed words, Γ must be the left bracketed version of $\tau_{\langle 1,n\rangle}(w)$. Therefore $[\![l(\tau_{\langle 1,n\rangle}\tau(w))]\!] \subseteq [\![S]\!]$, hence $l(w) \in [\![S]\!]$. This corresponds to $w \in \{c.a^k \,|\, k \leq n\}$ □

`Proof:` of $c\{a\}^* \subseteq \mathrm{L}(G_{\langle 1,*\rangle})$:
We have $c \in \mathrm{L}(G_{\langle 1,*\rangle})$ since in $\mathbf{NL}_\emptyset$:

$$\frac{\dfrac{p \vdash p}{\emptyset \vdash p/p} \quad S \vdash S}{S/(p/p) \vdash S}$$

Let $\Gamma_0 = (p/p), \Gamma_k = ((p/p)/(p/p), \Gamma_{k-1})$. By induction on k we get $\Gamma_k \vdash (p/p)$ in $\mathbf{NL}_\emptyset$:

$$\frac{\begin{array}{c}\vdots\\ \Gamma_{k-1} \vdash p/p\end{array} \quad p/p \vdash p/p}{\underbrace{((p/p)/(p/p), \Gamma_{k-1})}_{=\Gamma_k} \vdash p/p}$$

therefore

$$\frac{\begin{array}{c}\vdots\\ \Gamma_k \vdash p/p\end{array} \quad S \vdash S}{(S/(p/p), \Gamma_k) \vdash S}$$

which shows $c.a^k$ is in the language of $G_{\langle 1,*\rangle}$.

`Proof:` of $\mathrm{L}(G_{\langle 1,*\rangle}) \subseteq c\{a\}^*$:

We consider the powerset residuated groupoid $(\mathcal{P}(M), \circ, \Rightarrow, \Leftarrow, \subseteq)$ as above but with the following (similar) interpretation: $[\![S]\!] = \{l(c.a^k) \mid 0 \leq k\}, [\![p]\!] = \{r(a^k)/0 \leq k\}$.

Let us suppose $\Gamma \vdash_{\mathbf{NL}_\emptyset} S$ where Γ is a bracketed version of $\tau_{\langle 1,*\rangle}(w)$. By models, we have $[\![\Gamma]\!] \subseteq [\![S]\!]$.

As before, $[\![p/p]\!] = \{\varepsilon, a\}$. Thus $[\![S/(p/p)]\!] = \{l(c.a^k) \mid 0 \leq k\} = [\![S]\!] (= \{z \in M \mid \forall x \in [\![p/p]\!], z.x \in [\![S]\!]\} = \{z \in M \mid z \in \{l(c.a^k) \mid 0 \leq k\}$ and $(z.a) \in \{l(c.a^k)/0 \leq k\}\})$.

Therefore, if $[\![l(\tau_{\langle 1,*\rangle}(w))]\!] \subseteq [\![S]\!] = \{l(c.a^k) \mid 0 \leq k\}$, this implies that $w = cw'$ with $w' \in \{a\}^*$. □

9.4 A Limit Point for Rigid L Grammars

9.4.1 Construction Overview

Definition 9.13 *We define the following types and assignments $\tau_{\langle 2,n\rangle}$, where $A = p\backslash p, B = q\backslash q$ and p, q are primitive types: $\{a \mapsto A; b \mapsto B; c \mapsto D_{\langle 2,n\rangle}\}$, where $D_{\langle 2,0\rangle} = S$ and $D_{\langle 2,n\rangle} = (S/p) \bullet ((p/q) \bullet (q/p))^{n-1} \bullet (p/q) \bullet q$ if $n > 0$. We write $D'_{\langle 2,n\rangle} = S/p \bullet (p/q \bullet q/p)^n \bullet p$. Let $G_{\langle 2,n\rangle}$ denote the grammar defined by $\tau_{\langle 2,n\rangle}$ with alphabet $\{a, b, c\}$.*

Language. We get (see proof) $\mathrm{L}(G_{\langle 2,n\rangle}) = c(b^*a^*)^n$.

9.4.2 Details of Proofs

The proof is based on syntax and models:

Lemma 9.14 *1. if $\tau_{\langle 2,n\rangle}(w) \vdash S$ is derivable in $\mathbf{L}$ then w has exactly one occurrence of c,*

2. if $p, \tau_{\langle 2,n\rangle}(w) \vdash p$ in $\mathbf{L}$ then $w \in a^*$,

(a) if $q, \tau_{\langle 2,n\rangle}(w) \vdash q$ in $\mathbf{L}$ then $w \in b^*$,

3. $\tau_{\langle 2,n\rangle}(w_1), S, \tau_{\langle 2,n\rangle}(w_2) \not\vdash S$ (where $w_1.w_2 \in \{a, b\}^+$).

`Proof:` of 1 direct by interpretation in the free group ($[\tau_{\langle 2,n\rangle}(c)] = S$ and $[\tau_{\langle 2,n\rangle}(a)] = [\tau_{\langle 2,n\rangle}(b)] = I$).

`Proof:` of 2, 2a, 3 we consider the powerset residuated semi-group over $M = \{p, q, t, I\}$ equipped with the associative operator $\circ$:

$\circ$	p	q	t	I
p	p	t	t	p
q	t	q	t	q
t	t	t	t	t
I	p	q	t	I

We now define $[\![p]\!] = \{p\}$, $[\![q]\!] = \{q\}$, $[\![S]\!] = I$ and get $[\![A]\!] = \{p, I\}$ and $[\![B]\!] = \{q, I\}$. Suppose $\tau_{\langle 2,n\rangle}(w) \vdash A$, then by models $[\![\tau_{\langle 2,n\rangle}(w)]\!] \subseteq [\![A]\!]$; this is impossible if w has an occurrence of b, since we would have $[\![\tau_{\langle 2,n\rangle}(w)]\!] \ni q$ or $[\![\tau_{\langle 2,n\rangle}(w)]\!] \ni t$ whereas $[\![A]\!] = \{I, p\}$.

Sublemma 2a can be demonstrated in a similar way.

To show 3, we just have to consider $w_1.w_2 \in \{a, b\}^+$ to obtain $[\![\tau_{\langle 2,n\rangle}(w_1), S, \tau_{\langle 2,n\rangle}(w_2)]\!] \subseteq \{p, q, t\}$, whereas $[\![S]\!] = \{I\}$. □

Lemma 9.15 *1. (i) if $(p/q, q/p)^m, p, \tau(w) \vdash p$ in L then $w \in a^*(b^*a^*)^m$,*

2. (ii) if $(q/p, p/q)^m, q, \tau(w) \vdash q$ in L then $w \in b^*(a^*b^*)^m$,

3. *(iii) if* $q/p, (p/q, q/p)^m, p, \tau(w) \vdash q$ *in* L *then* $w \in (a^*b^*)^{m+1}$,

4. *(iv) if* $p/q, (q/p, p/q)^m, q, \tau(w) \vdash p$ *in* L *then* $w \in (b^*a^*)^{m+1}$.

`Proof:` This lemma is established by reasoning on the possible derivations in **L**, using induction on m and the length of w, with the help of free group interpretation of the sequents in the derivations.

Note that if $n > 0$, then $\tau(w_1), D_{\langle 2,n\rangle}, \tau(w_2) \vdash S \Leftrightarrow$
$\tau(w_1), S/p, (p/q, q/p)^{n-1}, p/q, q, \tau(w_2) \vdash S$, since $C_1, \ldots, C_n \vdash C_{n+1} \Leftrightarrow C_1 \bullet \ldots \bullet C_n \vdash C_{n+1}$.

Proposition 9.16 *(Language description)* $\mathrm{L}(G_{\langle 2,n\rangle}) = c(b^*a^*)^n$.

`Proof:` of $c(b^*a^*)^n \subseteq \mathrm{L}(G_{\langle 2,n\rangle})$
For $n = 0$ this is an axiom: $\tau_{\langle 2,n\rangle}(c) = S \vdash S$.
For $n = 1$, we have the deduction

$$\dfrac{S \vdash S \quad \dfrac{p, A \vdash p \quad q, B \vdash q}{p/q, q, B, A \vdash p}}{S/p, p/q, q, B, A \vdash S}$$

Suppose $n > 1$ and $w' = cw = c(b^*a^*)^{n-1} \in \mathrm{L}(G_{\langle 2,n-1\rangle})$. First it is shown that $c.b.a.w \in \mathrm{L}(G_{\langle 2,n\rangle})$:

$$\dfrac{\dfrac{\overbrace{(S/p), ((p/q), (q/p))^{n-2}, (p/q), q}^{\Leftrightarrow D_{\langle 2,n-1\rangle}}, \tau(w) \vdash S \quad p, A \vdash p}{(S/p), ((p/q), (q/p))^{n-1}, p, A, \tau(w) \vdash S}\, /l \quad q, B \vdash q}{\underbrace{(S/p), ((p/q), (q/p))^{n-1}, (p/q), q}_{=D_{\langle 2,n\rangle}}, \underbrace{B, A, \tau(w)}_{=\tau(b.a.w)} \vdash S}\, /l$$

-we then easily get $c.w \in \mathrm{L}(G_{\langle 2,n\rangle})$ since $D_{\langle 2,n\rangle} \vdash D_{\langle 2,n-1\rangle}$ in **L** for $n > 0$.
-we also get $c.a.w \in \mathrm{L}(G_{\langle 2,n\rangle})$ from $D_{\langle 2,n\rangle}, A \vdash D_{\langle 2,n-1\rangle}$
-we get $c.b.w \in \mathrm{L}(G_{\langle 2,n\rangle})$ from $D_{\langle 2,n\rangle}, B \vdash D_{\langle 2,n-1\rangle}$
-since $\tau(a), \tau(a) \vdash \tau(a)$ and $\tau(b), \tau(b) \vdash \tau(b)$, this can be extended to repetitions of each letter a or b separately, which concludes the proof.

`Proof:` of $\mathrm{L}(G_{\langle 2,n\rangle}) \subseteq c(b^*a^*)^n$
We have to show: if $\tau(w_1), D_{\langle 2,n\rangle}, \tau(w_2) \vdash S$ in **L** then w_1 is empty and $w_2 \in (b^*a^*)^n$.
We show by joined lexicographical induction on n and $s = |w_2|$ that
(i) if $\tau(w_1), D_{\langle 2,n\rangle}, \tau(w_2) \vdash S$ in **L** then w_1 is empty and $w_2 \in (b^*a^*)^n$
(ii) if $\tau(w_1), D'_{\langle 2,n\rangle}, \tau(w_2) \vdash S$ in **L** then w_1 is empty and $w_2 \in a^*(b^*a^*)^n$

9.4.3 New Types Without Product

We now transform the type-assignments to obtain product-free types by currying. Type raising properties are used to apply the previous result on $\tau_{\langle 2,n\rangle}$.

Definition 9.17 . *We define the following type-assignments where $A = p\backslash p$, $B = q\backslash q$ and p, q are primitive types: $\tau_{\langle 3,n\rangle} : \{a \mapsto A; b \mapsto B; c \mapsto D_{\langle 3,n\rangle}\}$, where $D_{\langle 3,n\rangle} = D^{l,S}_{\langle 2,n\rangle}$.*

Let $G_{\langle 3,n\rangle}$ denote the grammar defined by $\tau_{\langle 3,n\rangle}$ with alphabet $\{a, b, c\}$ and distinguished type S.

Let $G_{\langle 3,*\rangle}$ denote the grammar (with type-assignment $\tau_{\langle 3,*\rangle}$) $\{a \mapsto A; b \mapsto A; c \mapsto S/A\}$.

Proposition 9.18 *(Language description)* $\mathrm{L}(G_{\langle 3,n\rangle}) = c(b^*a^*)^n - \{c\}$ *in* $\mathbf{L}$ *for* $n \geq 0$ *and* $\mathrm{L}(G_{\langle 3,*\rangle}) = c\{a, b\}^+$ *in* $\mathbf{L}$.

`Proof:` of $c(b^*a^*)^n \subseteq \mathrm{L}(G_{\langle 3,n\rangle})$ We know from Proposition 9.16 that for $n \geq 0, D_{\langle 2,n\rangle}, \tau(w) \vdash S$ where $w \in (b^*a^*)^n$. Moreover,

$$\dfrac{S \vdash S \quad \dfrac{D_{\langle 2,n\rangle}, \tau(w) \vdash S}{\tau(w) \vdash D_{\langle 2,n\rangle}\backslash S}\ [\backslash I]\ (\text{if } \tau(w) \neq \varepsilon)}{S/(D_{\langle 2,n\rangle}\backslash S), \tau(w) \vdash S}\ [/E]$$

thus $c(b^*a^*)^n - \{c\} \subseteq \mathrm{L}(G_{\langle 3,n\rangle})$.

`Proof:` of $\mathrm{L}(G_{\langle 3,n\rangle}) \subseteq c(b^*a^*)^n - \{c\}$ in $\mathbf{L}$.

Using free group interpretation it can be shown that there is exactly one c in every word of $\mathrm{L}(G_{\langle 3,n\rangle})$, $\tau_{\langle 3,n\rangle}(w) \vdash S$ is then equivalent to $\tau_{\langle 3,n\rangle}(w_1), D_{\langle 3,n\rangle}, \tau_{\langle 3,n\rangle}(w_2) \vdash S$ where $w = w_1.c.w_2$.

By definition, $\tau_{\langle 3,n\rangle}(w_1), D^{l,S}_{\langle 2,n\rangle}, \tau_{\langle 3,n\rangle}(w_2) \vdash S$ thus $\tau_{\langle 3,n\rangle}(w_1), D_{\langle 2,n\rangle}, \tau_{\langle 3,n\rangle}(w_2) \vdash S$ and $\tau_{\langle 2,n\rangle}(w_1), D_{\langle 2,n\rangle}, \tau_{\langle 2,n\rangle}(w_2) \vdash S$.

By Proposition 9.16, w_1 is empty and $w_2 \in (b^*a^*)^n$. Moreover, for $n \geq 0, \tau_{\langle 3,n\rangle}(c) \not\vdash S$ in $\mathbf{L}$ since $C^{l,S} \not\vdash S$ in $\mathbf{L}$ for any formula C. We have $\mathrm{L}(G_{\langle 3,n\rangle}) \subseteq c(b^*a^*)n - \{c\}$ as desired.

By currying $D_{\langle 3,n\rangle}$ is equivalent to a type without product:

$$D_{\langle 3,n\rangle} \equiv S/(q\backslash((p/q)\backslash\overbrace{((q/p)\backslash\ldots\backslash((p/q)}^{n-1 \text{ times}})\backslash((S/p)\backslash S))\ldots)))$$

The previous property is then true for $\mathbf{L}$ without product.

We now have to prove that $\mathrm{L}(G_{\langle 3,*\rangle}) = c\{a, b\}^+$ to obtain a limit point.

`Proof:` of $c\{a, b\}^+ \subseteq \mathrm{L}(G_{\langle 3,*\rangle})$; We have $ca, cb \in \mathrm{L}(G_{\langle 3,*\rangle})$ since $S/A, A \vdash S$ Thus $c\{a, b\}^+ \subseteq \mathrm{L}(G_{\langle 3,*\rangle})$ since $A, A \vdash A$.

`Proof:` of $\mathrm{L}(G_{\langle 3,*\rangle}) \subseteq c\{a, b\}^+$; We consider the powerset residuated semigroup $(\mathcal{P}(V^+), \circ, \Rightarrow, \Leftarrow, \subseteq)$ with the interpretation $[\![S]\!] = c\{a, b\}^*$ and $[\![p]\!] = a^*$, thus $[\![A]\!] = a^*$ and $[\![S/A]\!] = c\{a, b\}^*$. Therefore if $[\![\tau_{\langle 3,*\rangle}(w)]\!] \subseteq [\![S]\!] = c\{a, b\}^*$,

which implies $w = cw'$ with $w' \in \{a, b\}^*$. Moreover $S/A \not\vdash S$ in **L**, thus $w' \in \{a, b\}^+$ □

From these constructions we get the following propositions as corollaries:

Proposition 9.19 *(nonlearnability) The class of languages of rigid (or k-valued for an arbitrary k) Non-associative Lambek grammars with empty sequence contains a limit point; the class of rigid (or k-valued for an arbitrary k) Non-associative Lambek grammars with empty sequence is not learnable from strings.*

Proposition 9.20 *(nonlearnability) The class of languages of rigid (or k-valued for an arbitrary k) Lambek grammars without product and without empty sequence contains a limit point; the class of rigid (or k-valued for an arbitrary k) Lambek grammars without product and without empty sequence is not learnable from strings.*

9.5 The Classes of Rigid LP and $\mathbf{LP}_\emptyset$ Grammars are Not Learnable

The proofs from Foret and Le Nir (2002a) as discussed in the previous section rely on the fact that in **L** the axioms $A/A, A/A \rightarrow A/A$ (and in $\mathbf{L}_\emptyset$ the axiom $B/(A/A) \rightarrow B$) hold. These axioms cause contraction-like phenomena that allow the existence of limit points even in a class of (string) languages generated by rigid grammars. They defined rigid grammars $G_n, n \in \mathbb{N}$ and G_* such that $\mathrm{L}(G_n) = c(b^*a^*)^n$ and $\mathrm{L}(G_n) = c\{a, b\}^*$. For G_n the number of alternations between a sequence of a's and a sequence of b's, (both of unbounded length) is bounded. This approach is not readily applicable to either **LP** or $\mathbf{LP}_\emptyset$ grammars, since commutativity removes the bound on the number of alterations in $\mathrm{L}(G_n)$. Instead we exploit an asymmetry inherent in the Lifting operation to show that rigid **LP** and $\mathbf{LP}_\emptyset$ grammars are not learnable.

9.5.1 The Construction of a Limit Point for LP and $\mathbf{LP}_\emptyset$

Definition 9.21 *For $n = 0$, let G_n be defined as*

$$G_0 : \begin{array}{lcl} \mathtt{s} & \mapsto & (s/a)/c \\ \mathtt{a} & \mapsto & a \\ \mathtt{c} & \mapsto & c \end{array}$$

and for any $n \in \mathbb{N}^+$, let G_n be defined as

$$G_n : \begin{array}{lcl} \mathtt{s} & \mapsto & (s/\underbrace{a^a \bullet a^a \dots a^a}_{n \text{ times}})/(a\backslash a^a) \\ \mathtt{a} & \mapsto & \underbrace{a \bullet a \dots a}_{n \text{ times}} \\ \mathtt{c} & \mapsto & a\backslash a^a \end{array}$$

and let G_+ be defined as

$$G_+ : \begin{array}{rcl} \mathtt{s} & \mapsto & (s/a)/(c/c) \\ \mathtt{a} & \mapsto & a \\ \mathtt{c} & \mapsto & c/c. \end{array}$$

A final word on notation: $\sigma, \sigma', \tau \ldots$ denote strings, and σ^{perm} is the function that yields the set of all permutations of σ.[5] Concatenation of strings will be denoted by $+$, and $\vdash$ will be taken to mean $\vdash_{\mathbf{LP}}$.

Lemma 9.22 *The language generated by any G_n, $n \in \mathbb{N}$, is $\bigcup\{\langle \mathtt{s}, \mathtt{a}, \mathtt{c}^{i+1}\rangle^{\mathrm{perm}} \mid 0 \leq i \leq n\}$.*

It may be helpful to look at some example derivations to get a feel for the kind of derivations our construction allows. We will then turn to the actual proof.

The following list of derivations was obtained using Grail.[6]

The list exhaustively enumerates all (normal form) derivations and corresponding lambda terms for the string `sac` given the grammar G_2 and calculus $\mathbf{LP}_\emptyset$.

$$\dfrac{\mathtt{a} \vdash a \bullet a \qquad \dfrac{\mathtt{s} \vdash s/(a/(a\backslash a)) \bullet (a/(a\backslash a)) \qquad \dfrac{(\mathtt{p}_2 \circ \mathtt{c}) \circ \mathtt{s}_1 \vdash (a/(a\backslash a)) \bullet (a/(a\backslash a))}{\mathtt{p}_2 \circ \mathtt{c} \vdash a/(a\backslash a)}\,[\backslash E] \qquad \dfrac{\dfrac{[\mathtt{s}_1 \vdash a]^3 \quad [\mathtt{r}_0 \vdash a\backslash a]^1}{\mathtt{s}_1 \circ \mathtt{r}_0 \vdash a}\,[/I]^1}{\mathtt{s}_1 \vdash a/(a\backslash a)}\,[\bullet I]}{(\mathtt{p}_2 \circ \mathtt{c}) \circ \mathtt{s}_1 \vdash (a/(a\backslash a)) \bullet (a/(a\backslash a))}\,[/E]}{\dfrac{\mathtt{s} \circ ((\mathtt{p}_2 \circ \mathtt{c}) \circ \mathtt{s}_1) \vdash s}{\dfrac{\mathtt{s} \circ (\mathtt{s}_1 \circ (\mathtt{p}_2 \circ \mathtt{c})) \vdash s}{\mathtt{s} \circ ((\mathtt{s}_1 \circ \mathtt{p}_2) \circ \mathtt{c}) \vdash s}\,[ass]}\,[comm]}{\mathtt{s} \circ (\mathtt{a} \circ \mathtt{c}) \vdash s}\,[\bullet E]^{3,4}$$

1. $(\mathbf{1}\ \langle(\mathbf{4}\ \pi^2\mathbf{2}), \lambda z_0.(z_0\ \pi^1\mathbf{2})\rangle)$

$$\dfrac{\mathtt{a} \vdash a \bullet a \qquad \dfrac{\mathtt{s} \vdash s/(a/(a\backslash a)) \bullet (a/(a\backslash a)) \qquad \mathtt{s}_1 \circ (\mathtt{p}_2 \circ \mathtt{c}) \vdash (a/(a\backslash a)) \bullet (a/(a\backslash a))}{\mathtt{s}_1 \circ (\mathtt{p}_2 \circ \mathtt{c}) \vdash (a/(a\backslash a)) \bullet (a/(a\backslash a))}\,[/E] \qquad \dfrac{\dfrac{[\mathtt{s}_1 \vdash a]^3 \quad [\mathtt{p}_1 \vdash a\backslash a]^2}{\mathtt{s}_1 \circ \mathtt{p}_1 \vdash a}\,[\backslash E]}{\mathtt{s}_1 \vdash a/(a\backslash a)}\,[/I]^2 \qquad \dfrac{[\mathtt{p}_2 \vdash a]^4 \quad \mathtt{c} \vdash a\backslash(a/(a\backslash a))}{\mathtt{p}_2 \circ \mathtt{c} \vdash a/(a\backslash a)}\,[\backslash E]}{\mathtt{s}_1 \circ (\mathtt{p}_2 \circ \mathtt{c}) \vdash (a/(a\backslash a)) \bullet (a/(a\backslash a))}\,[\bullet I]}{\dfrac{\mathtt{s} \circ (\mathtt{s}_1 \circ (\mathtt{p}_2 \circ \mathtt{c})) \vdash s}{\mathtt{s} \circ ((\mathtt{s}_1 \circ \mathtt{p}_2) \circ \mathtt{c}) \vdash s}\,[ass]}{\mathtt{s} \circ (\mathtt{a} \circ \mathtt{c}) \vdash s}\,[\bullet E]^{3,4}$$

2. $(\mathbf{1}\ \langle\lambda y_1.(y_1\ \pi^1\mathbf{2}), (\mathbf{4}\ \pi^2\mathbf{2})\rangle)$

[5] We will slightly abuse this notation by letting it denote *any* permutation of σ, we trust this will not lead to confusion.

[6] Grail is an automated theorem prover, written by Richard Moot, designed to aid in the development and prototyping of grammar fragments for categorial logics, see Moot (2002).

$$\dfrac{\mathtt{a} \vdash a \bullet a \qquad \dfrac{\dfrac{\mathtt{s} \vdash s/(a/(a\backslash a)) \bullet (a/(a\backslash a)) \qquad \dfrac{\dfrac{[\mathtt{s}_1 \vdash a]^3 \quad \mathtt{c} \vdash a\backslash(a/(a\backslash a))}{\mathtt{s}_1 \circ \mathtt{c} \vdash a/(a\backslash a)}\,[\backslash E] \qquad \dfrac{\dfrac{[\mathrm{p}_2 \vdash a]^4 \quad [\mathrm{r}_0 \vdash a\backslash a]^1}{\mathrm{p}_2 \circ \mathrm{r}_0 \vdash a}\,[\backslash E]}{\mathrm{p}_2 \vdash a/(a\backslash a)}\,[/I]^1}{(\mathtt{s}_1 \circ \mathtt{c}) \circ \mathrm{p}_2 \vdash (a/(a\backslash a)) \bullet (a/(a\backslash a))}\,[\bullet I]}{\mathtt{s} \circ ((\mathtt{s}_1 \circ \mathtt{c}) \circ \mathrm{p}_2) \vdash s}\,[/E]}{\mathtt{s} \circ (\mathrm{p}_2 \circ (\mathtt{s}_1 \circ \mathtt{c})) \vdash s}\,[comm]}{\mathtt{s} \circ ((\mathrm{p}_2 \circ \mathtt{s}_1) \circ \mathtt{c}) \vdash s}\,[ass]}{\mathtt{s} \circ ((\mathtt{s}_1 \circ \mathrm{p}_2) \circ \mathtt{c}) \vdash s}\,[comm]}{\mathtt{s} \circ (\mathtt{a} \circ \mathtt{c}) \vdash s}\,[\bullet E]^{3,4}$$

3. $(\mathbf{1}\ \langle(\mathbf{4}\ \pi^1\mathbf{2}), \lambda z_0.(z_0\ \pi^2\mathbf{2})\rangle)$

$$\dfrac{\mathtt{a} \vdash a \bullet a \qquad \dfrac{\dfrac{\dfrac{\mathtt{s} \vdash s/(a/(a\backslash a)) \bullet (a/(a\backslash a)) \qquad \dfrac{\dfrac{[\mathrm{p}_2 \vdash a]^4 \quad [\mathrm{p}_1 \vdash a\backslash a]^2}{\mathrm{p}_2 \circ \mathrm{p}_1 \vdash a}\,[\backslash E]}{\mathrm{p}_2 \vdash a/(a\backslash a)}\,[/I]^2 \qquad \dfrac{[\mathtt{s}_1 \vdash a]^3 \quad \mathtt{c} \vdash a\backslash(a/(a\backslash a))}{\mathtt{s}_1 \circ \mathtt{c} \vdash a/(a\backslash a)}\,[\backslash E]}{\mathrm{p}_2 \circ (\mathtt{s}_1 \circ \mathtt{c}) \vdash (a/(a\backslash a)) \bullet (a/(a\backslash a))}\,[\bullet I]}{\mathtt{s} \circ (\mathrm{p}_2 \circ (\mathtt{s}_1 \circ \mathtt{c})) \vdash s}\,[/E]}{\mathtt{s} \circ ((\mathrm{p}_2 \circ \mathtt{s}_1) \circ \mathtt{c}) \vdash s}\,[ass]}{\mathtt{s} \circ ((\mathtt{s}_1 \circ \mathrm{p}_2) \circ \mathtt{c}) \vdash s}\,[comm]}{\mathtt{s} \circ (\mathtt{a} \circ \mathtt{c}) \vdash s}\,[\bullet E]^{3,4}$$

4. $(\mathbf{1}\ \langle\lambda y_1.(y_1\ \pi^2\mathbf{2}), (\mathbf{4}\ \pi^1\mathbf{2})\rangle)$

`Proof:`

1. It is trivial to show that $\langle \mathtt{s}, \mathtt{a}, \mathtt{c}\rangle^{\mathrm{perm}} \subseteq \mathrm{L}(G_0)$.

   We prove that for any $n \in \mathbb{N}^+$, $\bigcup\{\langle \mathtt{s}, \mathtt{a}, \mathtt{c}^{i+1}\rangle^{\mathrm{perm}} \mid 0 \leq i \leq n\} \subseteq \mathrm{L}(G_n)$: Grammar G_n assigns $(s/\underbrace{a^a \bullet a^a \ldots a^a}_{n \text{ times}})/(a\backslash a^a)$ to $\mathtt{s}$, and $a\backslash a^a$ to $\mathtt{c}$. With right-elimination we get $\mathtt{s} \circ \mathtt{c} \vdash s/\underbrace{a^a \bullet a^a \ldots a^a}_{n \text{ times}}$ (and by commutation $\mathtt{c} \circ \mathtt{s} \vdash s/\underbrace{a^a \bullet a^a \ldots a^a}_{n \text{ times}}$).

   Grammar G_n assigns $\underbrace{a \bullet a \ldots a}_{n \text{ times}}$ to $\mathtt{a}$. Now, the derivation *TreeLift* =

   $$\dfrac{\dfrac{[\mathrm{hypo}_1 \vdash a]^1 \quad [\mathrm{hypo}_2 \vdash a\backslash a]^2}{\mathrm{hypo}_1 \circ \mathrm{hypo}_2 \vdash a}\,[\backslash E]}{\mathrm{hypo}_1 \vdash a/(a\backslash a)}\,[/I]^2$$

   can be combined into derivation $\mathit{TreeLift}_n$ through n times dot-introduction to yield $\mathrm{hypo}_1 \circ \ldots \circ \mathrm{hypo}_n \vdash \underbrace{a^a \bullet a^a \ldots a^a}_{n \text{ times}}$. Using $\mathit{TreeLift}_n$ as an argument for right-elimination, with $(\mathtt{s} \circ \mathtt{c})^{\mathrm{perm}} \vdash s/\underbrace{a^a \bullet a^a \ldots a^a}_{n \text{ times}}$ as functor, we get $(\mathtt{s} \circ \mathtt{c})^{\mathrm{perm}} \circ (\mathrm{hypo}_1 \circ \ldots \circ \mathrm{hypo}_n) \vdash s$. With n times dot-elimination, the last of which takes $a \vdash \underbrace{a \bullet a \ldots a}_{n \text{ times}}$ as argument, the hypotheses 1 through n can be eliminated, yielding $(\mathtt{s} \circ \mathtt{c})^{\mathrm{perm}} \circ \mathtt{a} \vdash s$. Using commutation and association we also get $\mathtt{a} \circ (\mathtt{s} \circ \mathtt{c})^{\mathrm{perm}} \vdash s$, etc, so $\bigcup\{\langle \mathtt{s}, \mathtt{a}, \mathtt{c}^{i+1}\rangle^{\mathrm{perm}} \mid i = 0\} \subseteq \mathrm{L}(G_n)$.

Grammar G_n assigns $a\backslash a^a$ to $\mathtt{c}$, so the derivation *TreeCElim* =

$$\frac{[\text{hypo} \vdash a]^1 \quad \mathtt{c} \vdash a\backslash(a/(a\backslash a))}{\text{hypo} \circ \mathtt{c} \vdash a/(a\backslash a)}\ [\backslash E]$$

derives the same type as *TreeLift* does. Since i $(0 \leq i \leq n)$ *TreeLift* deductions can occur in a derivation for G_n, by replacing them with *TreeCElim* we get $i+1$ times $\mathtt{c}$ in the yield of the complete deduction.

With application of associativity and commutativity rules the resulting sequent can be rearranged so that all hypotheses occur in one minimal subsequent (for example, $\mathtt{s} \circ (((\text{hypo}_1 \circ \mathtt{c}) \circ \text{hypo}_2) \circ ((\mathtt{c} \circ \text{hypo}_3) \circ \mathtt{c})) \vdash s$ becomes $\mathtt{s} \circ ((\text{hypo}_1 \circ (\text{hypo}_2 \circ \text{hypo}_3)) \circ (\mathtt{c} \circ (\mathtt{c} \circ \mathtt{c}))) \vdash s$), which can then be replaced through dot-elimination by $\mathtt{a}$. Thus $(\mathtt{s} \circ \mathtt{c})^{\text{perm}} \circ \mathtt{c}(i \text{ times}) \circ \mathtt{a} \vdash s$ is obtained, and any permutation of this as well, by commutativity and associativity. Thus $\bigcup\{\langle \mathtt{s}, \mathtt{a}, \mathtt{c}^{i+1}\rangle^{\text{perm}} \mid 1 \leq i \leq n\} \subseteq \mathrm{L}(G_n)$, for any $n \in \mathbb{N}^+$.

Together with the result for $\mathrm{L}(G_0)$, this shows that $\bigcup\{\langle \mathtt{s}, \mathtt{a}, \mathtt{c}^{i+1}\rangle^{\text{perm}} | 0 \leq i \leq n\} \subseteq \mathrm{L}(G_n)$, for any $n \in \mathbb{N}$.

2. It is trivial to show that $\mathrm{L}(G_0) \subseteq \langle \mathtt{s}, \mathtt{a}, \mathtt{c}\rangle^{\text{perm}}$.

   We prove that for any $n \in \mathbb{N}^+$, $\mathrm{L}(G_n) \subseteq \bigcup\{\langle \mathtt{s}, \mathtt{a}, \mathtt{c}^{i+1}\rangle^{\text{perm}} \mid 0 \leq i \leq n\}$: For a string σ to be included in a language generated by an **LP** grammar G, G must assign a type T_n to a symbol in σ that has s as range subtype. For any n, G_n assigns such a type only to the symbol $\mathtt{s}$. Furthermore, s occurs only once, as range subtype, in this type. Hence $\mathtt{s}$ must occur (only) once in every sentence in $\mathrm{L}(G_n)$. All derivations for a string in $\mathrm{L}(G_{i\geq 1})$ will start with

$$\dfrac{\dfrac{\mathtt{s} \vdash (s/TD_n^1)/TD_n^2 \quad \dfrac{\dfrac{\underline{\textit{Tree}_a}}{\vdots}}{\sigma \vdash TD_n^2}\ ass, comm}{\mathtt{s} \circ \sigma \vdash s/TD_n^1}\ [/E] \qquad \dfrac{\textit{Tree}_b}{\sigma' \vdash TD_n^1}}{\dfrac{(\mathtt{s} \circ \sigma) \circ \sigma' \vdash s}{\dfrac{\vdots}{\sigma'' \circ \mathtt{s} \circ \sigma''' \vdash s}\ ass, comm, [\bullet E]}}\ [/E]$$

   where $\sigma + \sigma'$ is some permutation of $\sigma'' + \sigma'''$ (either σ'' or σ''' may be empty). Since T_n has as domain subtype $TD_n^2 = a\backslash(a^a)$, $\textit{Tree}_a$ must yield $a\backslash(a^a)$. This tree can begin with a sequence of applications of the *ass* and *comm* rules (which only makes sense if σ is not a single symbol), there are some possibilities after this:

   (a) since $G_n, n \geq 1$ assigns this type to $\mathtt{c}$, $\sigma = \mathtt{c}$,

(b) use of $[\backslash I]^1$. This implies that the type a^a is derived from the sequent one step up. This type is a range type only of TD^2_n out of all types in $G_{n\geq 1}$. Therefore this derivation can end in

$$\frac{[\text{hypo} \vdash a]^1 \quad \mathtt{c} \vdash a\backslash(a^a)}{\text{hypo} \circ \mathtt{c} \vdash a^a}\ [\backslash E],$$

which, as far as string language is concerned, is equivalent to 2a.[7] The type a^a can be interpreted as either $a/(a\backslash a)$ or $(a/a)\backslash a$, so more introduction rules can appear. All possibilities lead to some range subtype unique to TD^2_n (with respect to the types found in G_n), therefore $\mathtt{c} \vdash a\backslash(a^a)$ must be in Tree_a. All the other types found in this tree must be introduced by hypotheses, and all the hypotheses introduced have to be eliminated within Tree_a, and all these cases are in fact equivalent to 2a.

Since T_n has only one other domain subtype $TD^1_n = \underbrace{a^a \bullet a^a \ldots a^a}_{n \text{ times}}$, every sentence in $\mathrm{L}(G_n)$ must contain at least one symbol to which G_n assigns a type with a as range subtype, the only symbols that qualify are $\mathtt{a}$ and $\mathtt{c}$. Given that there are no range subtypes TD^1_n to be found in G_n, Tree_b must be of the form[8]

$$\dfrac{\dfrac{\mathit{Tree}_1}{\tau_1 \vdash a^a} \quad \dfrac{\dfrac{\mathit{Tree}_2}{\tau_2 \vdash a^a} \quad \dfrac{\dfrac{\mathit{Tree}_{n-1}}{\tau_{n-1} \vdash a^a} \quad \dfrac{\mathit{Tree}_n}{\tau_n \vdash a^a}\ [\bullet I]}{\vdots}}{\tau_2 \circ \ldots \circ \tau_n \vdash a^a \bullet a^a \ldots a^a (n-1 \text{ times})}\ [\bullet I]}{\sigma' \vdash a^a \bullet a^a \ldots a^a (n \text{ times})}\ [\bullet I]$$

where $\sigma' = \tau_1 + \ldots + \tau_n$. Symbol $\mathtt{a}$ is assigned $\underbrace{a \bullet a \ldots a}_{n \text{ times}}$, using hypothetical reasoning and applying the Lifting rule n times this derives TD_n, hence it can be shown that $L' = \bigcup\{\langle \mathtt{s}, \mathtt{a}, \mathtt{c}^i \rangle^{\text{perm}} \mid i = 1\}$ is a subset of the language. This case corresponds with all trees $\mathit{Tree}_1 \ldots \mathit{Tree}_n$ being of the form *TreeLift* where the hypothesis hypo is cancelled (together with $n-1$ other hypotheses) lower in the tree by n times application of $[\bullet I]$ where the last application has argument $\mathtt{a} \vdash \underbrace{a \bullet a \ldots a}_{n \text{ times}}$.

Since $a^a = a/(a\backslash a)$ (the case $a^a = (a/a)\backslash a$ can be dealt with in similar

[7] Note however that this derivation is not in normal form as defined in Tiede (1998).

[8] This is actually a normal form for Tree_b, it could also be left-branching, for example. All the other possible configurations are equivalent, however, since **LP** is associative.

fashion), any $Tree_i$ is either of the form

$$\dfrac{\dfrac{\dfrac{\ldots \quad [\mathrm{r}_0 \vdash a\backslash a]^1}{\vdots}}{\tau_i' \vdash a}}{\dfrac{\tau_i' \circ \mathrm{r}_0 \vdash a/(a\backslash a)}{\vdots}\ [/I]^1}$$
$$\dfrac{}{\tau_i \vdash a/(a\backslash a)}\ ass, comm, [\bullet E]$$

which given the type-assignments in $G_{n\geq 1}$ can only be a (non-normal form) variant of *TreeLift*, or

$$\mathrm{symbol} \vdash a/(a\backslash a)$$

which, given the type-assignments in $G_{n\geq 1}$, is only compatible with the derivation *TreeCElim*. Using hypothetical reasoning and applying the Right Elimination rule $i \leq n$ times, we can obtain i times the type a^a. All remaining a's can be lifted to obtain n a^a's.

Thus, for any $n \in \mathbb{N}^+$, $\bigcup\{\langle \mathtt{s}, \mathtt{a}, \mathtt{c}^{i+1}\rangle^{\mathrm{perm}} \mid 0 \leq i \leq n\} \subseteq \mathrm{L}(G_n)$, and with the result for $\mathrm{L}(G_0)$, it follows that for any $n \in \mathbb{N}$, $\bigcup\{\langle \mathtt{s}, \mathtt{a}, \mathtt{c}^{i+1}\rangle^{\mathrm{perm}} \mid 0 \leq i \leq n\} \subseteq \mathrm{L}(G_n)$.

Taken together, 1 and 2 imply that for any $n \in \mathbb{N}$, $\mathrm{L}(G_n) = \bigcup\{\langle \mathtt{s}, \mathtt{a}, \mathtt{c}^{i+1}\rangle^{\mathrm{perm}} | 0 \leq i \leq n\}$. □

Lemma 9.23 *The language generated by* G_+ *is* $\langle \mathtt{s}, \mathtt{a}, \mathtt{c}^+\rangle^{\mathrm{perm}}$.

`Proof:`

1. We show that $\langle \mathtt{s}, \mathtt{a}, \mathtt{c}^+\rangle^{\mathrm{perm}} \subseteq \mathrm{L}(G_+)$: Grammar G_+ assigns $(s/a)/(c/c)$ to $\mathtt{s}$, and c/c to $\mathtt{c}$. Since in **LP** the axiom $A/A, A/A \rightarrow A/A$ holds, it follows immediately that $\mathtt{c} \circ \ldots \mathtt{c} \vdash c/c$, thus with right-elimination we get $\mathtt{s} \circ \mathtt{c}^+ \vdash s/a$. Grammar G_+ assigns a to $\mathtt{a}$, thus $(\mathtt{s} \circ \mathtt{c}^+) \circ \mathtt{a} \vdash s$. By associativity and commutativity any permutation of this sequent will also derive s, thus any string in $\langle \mathtt{s}, \mathtt{a}, \mathtt{c}^+\rangle^{\mathrm{perm}}$ can be derived.

2. We show that $\mathrm{L}(G_+) \subseteq \langle \mathtt{s}, \mathtt{a}, \mathtt{c}^+\rangle^{\mathrm{perm}}$: For a string σ to be included in a language generated by an **LP** grammar G, G must assign a type T_+ to a symbol in σ that has s as subtype. Grammar G_+ assigns such a type only to the symbol $\mathtt{s}$. Furthermore, s occurs only once, as range subtype, in this type. Hence $\mathtt{s}$ must occur (only) once in every sentence in $\mathrm{L}(G_+)$. Since T_+ has only two domain subtypes $TD^1_+ = a$ and $TD^2_+ = c/c$, every sentence in $\mathrm{L}(G_+)$ must contain at least one symbol to which G_+ assigns

a type with a as range subtype, the only symbol that qualifies is $\mathtt{a}$. Thus all derivations for a string in this language must start with

$$\dfrac{\dfrac{\dfrac{\mathtt{s} \vdash (s/a)/(c/c) \quad \dfrac{\mathit{Tree}_+}{\sigma' \vdash c/c}}{\mathtt{s} \circ (\sigma') \vdash s/a}\,[/E] \qquad \mathtt{a} \vdash a}{(\mathtt{s} \circ (\sigma')) \circ \mathtt{a} \vdash s}\,[/E]}{\dfrac{\vdots}{\sigma'' \circ \mathtt{s} \circ \sigma''' \vdash s}\; ass, comm, [\bullet E]}$$

where $\sigma' \circ \mathtt{a}$ is some permutation of $\sigma'' + \sigma'''$ (σ'' and σ''' may be empty).

Grammar G_+ assigns TD^2_+ as range subtypes to $\mathtt{c}$, so Tree_+ can simply be $\mathtt{c} \vdash c/c$. Some reflection will show that other possibilities must be of the (normal) form:

$$\dfrac{\dfrac{\mathtt{c}_1 \vdash c/c \quad \dfrac{\mathtt{c}_2 \vdash c/c \qquad \vdots}{\mathtt{c}_2 \circ \ldots \circ \mathtt{c}_i \vdash c}\,[/E]}{\mathtt{c}_1 \circ \ldots \circ \mathtt{c}_i \vdash c}\,[/E]}{\mathtt{c}_1 \circ \ldots \circ \mathtt{c}_i \vdash c/c}\,[/I]^1 \qquad \dfrac{\mathtt{c}_i \vdash c/c \quad [c]^1}{\mathtt{c} \vdash c}\,[/E]$$

This shows that there must be one or more $\mathtt{c}$'s in every sentence in $\mathrm{L}(G_+)$. Thus the language generated by G_+ is $\langle \mathtt{s}, \mathtt{a}, \mathtt{c}^+ \rangle^{\mathrm{perm}}$. □

Theorem 9.24 *The class of rigid* **LP** *grammars has a limit point.*

`Proof:` From Lemma 9.22 it follows that the languages $\mathrm{L}(G_0) \subset \mathrm{L}(G_1) \subset \ldots$ form an infinite ascending chain.

By Lemma 9.23 $\mathrm{L}(G_+) = \langle \mathtt{s}, \mathtt{a}, \mathtt{c}^+ \rangle^{\mathrm{perm}}$ and for any $n \in \mathbb{N}$ and $0 \leq i \leq n$, $\mathrm{L}(G_n) = \langle \mathtt{s}, \mathtt{a}, \mathtt{c}^{i+1} \rangle^{\mathrm{perm}}$, $\mathrm{L}(G_+) = \cup_{n \in \mathbb{N}} \mathrm{L}(G_n)$, thus $\mathrm{L}(G_+)$ is a limit point for the class of rigid **LP** grammars. □

Corollary 9.25 *The class of rigid* **LP** *grammars is not (non-effectively) learnable from strings.*

In contrast to Foret and Le Nir's results, it is still an open question whether the class of *unidirectional* rigid **LP** grammars is learnable; the class under consideration is bi-directional, but only because lifting is necessary for the construction to work.

Also note that the construction depends on the presence of introduction and elimination rules for the product, and cannot be (easily) adapted for a product-free version of **LP**.

In the case of $\mathbf{LP}_\emptyset$, i.e. **LP** allowing empty sequent, things are slightly less complicated, since the axiom $B/(A/A) \to B$ holds. Consider the following construction:

Definition 9.26 *For any $n \in \mathbb{N}$, let G_n be defined as*

$$G_n : \begin{array}{rcl} \mathtt{s} & \mapsto & s/\underbrace{a^a \bullet a^a \dots a^a}_{n \text{ times}} \\ \mathtt{a} & \mapsto & \underbrace{a \bullet a \dots a}_{n \text{ times}} \\ \mathtt{c} & \mapsto & a\backslash a^a \end{array}$$

and let G_* be defined as

$$G_* : \begin{array}{rcl} \mathtt{s} & \mapsto & (s/a)/(c/c) \\ \mathtt{a} & \mapsto & a \\ \mathtt{c} & \mapsto & c/c. \end{array}$$

Lemma 9.27 *The language generated by any G_n, $n \in \mathbb{N}$, is $\bigcup\{\langle \mathtt{s}, \mathtt{a}, \mathtt{c}^i \rangle^{\mathrm{perm}} \mid 0 \leq i \leq n\}$.*

The proof is very similar to the proof of Lemma 9.22.

Lemma 9.28 *The language generated by G_* is $\langle \mathtt{s}, \mathtt{a}, \mathtt{c}^* \rangle^{\mathrm{perm}}$.*

The proof is very similar to the proof of Lemma 9.23.

Theorem 9.29 *The class of rigid $\mathbf{LP}_\emptyset$ grammars has a limit point.*

The proof is similar to the proof of Theorem 9.24; Lemmas 9.27 and 9.28 imply the existence of a limit point.

Corollary 9.30 *The class of rigid $\mathbf{LP}_\emptyset$ grammars is not (non-effectively) learnable from strings.*

9.6 Gentzen's Structural Rules

(Re)introduction to **NL** of (some of) Gentzen's structural rules has been proposed in the literature, see eg van Benthem (1991). The rules for contraction (C), expansion (E) and weakening (also known as *thinning*) (W) are as follows:

$$\frac{X[Y \circ Y] \Rightarrow A}{X[Y] \Rightarrow A}\ [C] \quad \frac{X[Y] \Rightarrow A}{X[Y \circ Y] \Rightarrow A}\ [E] \quad \frac{X[Y] \Rightarrow A}{X[Y \circ Z] \Rightarrow A}\ [W]$$

To the best of the author's knowledge there has been little linguistic motivation for (pure versions of) these rules. In Buszkowski (To appear) a limited usage of contraction has been proposed to be able to derive $S\backslash(S/S) \Rightarrow VP\backslash(VP/VP)$, where $VP = NP\backslash S$, i.e., lifting from sentence conjunction to verb phrase conjunction. See Morrill (1994) for the use of contraction for an analysis of parasitic gapping. In Jäger (2001) rules that incorporate some aspects of contraction were used to deal with anaphora, and in Moortgat (1997) the same was done to deal with certain coordination-phenomena.

Proposition 9.31 *The class $\mathcal{G}_{\text{rigid}}\mathbf{NL} + C$ has an infinite descending chain.*

`Proof:` Let $G_n, n \in \mathbb{N}$ be defined as $\{\mathtt{s} \mapsto s/\underbrace{(a \bullet a) \dots (a \bullet a)}_{n \text{ times}}, \mathtt{a} \mapsto a\}$.

1. It is easy to see that $\langle \mathtt{s}, \mathtt{a}^i \rangle, i \geq n \subseteq \mathrm{L}(G_n)$: to derive s, it is sufficient to show that sequences associated with words $\mathtt{a} \dots \mathtt{a}$ of a certain length derive $\underbrace{(a \bullet a) \dots (a \bullet a)}_{n \text{ times}}$. Obviously words of length n fulfill this condition, since the associated sequence consists of just n times a, and by repeatedly applying the $[\bullet I]$-rule the desired type can be obtained. Thus $\langle \mathtt{s}, \mathtt{a}^i \rangle, i = n \subseteq \mathrm{L}(G_n)$.

   By applying C we obtain the (sub)derivation $\dfrac{(a \bullet a) \vdash a}{a \vdash A}\ [C]$. Applying this rule i times at the appropriate places during a derivation we can 'get rid of' i excessive a's, so words of length $n+i$ fulfill the condition as well.

2. Showing that $\mathrm{L}(G_n) \subseteq \langle \mathtt{s}, \mathtt{a}^i \rangle, i \geq n$ is easy but tedious, we leave this to the reader.

Thus $\mathrm{L}(G_1) \supset \mathrm{L}(G_2) \supset \mathrm{L}(G_3) \dots$, and since G_n is rigid for any $n \in \mathbb{N}$, $\mathcal{G}_{\text{rigid}}\mathbf{NL} + C$ has an infinite descending chain. □

Note that this implies that $\mathcal{G}_{\text{rigid}}\mathbf{NL} + C$ is not learnable by enumeration, and that it is not efficiently learnable according to Definition 2.72 (recall Theorem 2.73). We now turn to the case of allowing expansion:

Proposition 9.32 *The class $\mathcal{G}_{\text{rigid}}\mathbf{NL} + E$ has an infinite ascending chain.*

`Proof:` Let $G_n, n \in \mathbb{N}$ be defined as $\{\mathtt{s} \mapsto s/\underbrace{(a \bullet a) \dots (a \bullet a)}_{n \text{ times}}, \mathtt{a} \mapsto a\}$.

1. It is easy to see that $\langle \mathtt{s}, \mathtt{a}^i \rangle, i \leq n \subseteq \mathrm{L}(G_n)$: to derive s, it is sufficient to show that sequences associated with words $\mathtt{a} \dots \mathtt{a}$ of a certain length derive $\underbrace{(a \bullet a) \dots (a \bullet a)}_{n \text{ times}}$. Obviously words of length n fulfill this condition, since the associated sequence consists of just n times a, and by repeatedly

applying the $[\bullet I]$-rule the desired type can be obtained. Thus $\langle \mathtt{s}, \mathtt{a}^i \rangle, i = n \subseteq \mathrm{L}(G_n)$.

By applying E we obtain the (sub)derivation $\dfrac{a \vdash a}{(a \bullet a) \vdash a}\ [E]$. Applying this rule i times at the appropriate places during a derivation we can 'introduce' i extra a's, so words of length $n - i$ fulfill the condition as well.

2. Showing that $\mathrm{L}(G_n) \subseteq \langle \mathtt{s}, \mathtt{a}^i \rangle, i \leq n$ is easy but tedious, we leave this to the reader.

Thus $\mathrm{L}(G_1) \subset \mathrm{L}(G_2) \subset \mathrm{L}(G_3) \ldots$, and since G_n is rigid for any $n \in \mathbb{N}$, $\mathcal{G}_{\text{rigid}}\mathbf{NL} + E$ has an infinite ascending chain. □

Since the E rule implies W the same proof can be used to show the following:

Corollary 9.33 *The class $\mathcal{G}_{\text{rigid}}\mathbf{NL} + W$ has an infinite ascending chain.*

When we allow to types to be assigned to the same symbol the latter two results can be strengthened:

Proposition 9.34 *The class $\mathcal{G}_{\text{2-valued}}\mathbf{NL} + E$ has a limit point.*

`Proof:` Let $G_n, n \in \mathbb{N}$ be defined as in the proof of Proposition 9.32.

Let $G_* = \{\mathtt{s} \mapsto s/a, \mathtt{a} \mapsto a, a/a\}$. It is easy to see that $\mathrm{L}(G_*) = \mathtt{sa}^*$, thus $\mathrm{L}(G_*) = \bigcup_{i=1}^{\infty} \mathrm{L}(G_i)$, so $\mathcal{G}_{\text{2-valued}}\mathbf{NL} + E$ has a limit point. □

Since the E rule implies W the same proof can be used to show the following:

Corollary 9.35 *The class $\mathcal{G}_{\text{2-valued}}\mathbf{NL} + W$ has a limit point.*

Note that these two results hold for the unidirectional case.

9.7 Generalizations of the Lambek Calculus

Residuated logical connectives are easily generalized to operators of arbitrary arity, see for example Moortgat (1996) and Kandulski (2002).

A *multimodal logic of pure residuation* (**LPR**) is characterized by a family of modes $\mathcal{M}$ and a function δ that assigns each mode an arity. If $f \in \mathcal{M}$ is a mode of arity $\delta(f) = m$, it defines an m-ary product $f_\bullet$ and m implications $\{f^i_{\rightarrow} \mid 1 \leq i \leq m\}$. The laws for binary and unary operators are generalized in the following way: $f_\bullet(A_1, \ldots, A_{\delta(f)}) \rightarrow f^i_{\rightarrow}(A_1, \ldots, A_{i-1}, B, A_{i+1}, \ldots, A_{\delta(f)})_i$, where $\forall f \forall i \leq \delta(f)$.

A similar calculus known as *Generalized Lambek Calculus* (**GLC**) was introduced in Buszkowski (1989) and analysed in Kandulski (1997) and Kołowska–Gawiejnowicz (1997).

The class of **LPR**-grammars is known to be context-free, see Jäger (To appear), where **LPR** extended with structural rules were also considered. It should be obvious that the class of rigid **LPR**-grammars contains the class of rigid **NL**, so non-learnability follows immediately. Adding the rule for associativity will also result in a nonlearnable class, even for rigid **LPR**-grammars, since these will contain the class of rigid **L**-grammars. Analogously, adding associativity- and commutativity rules will yield a nonlearnable class. However, it may be possible to characterize non-trivial learnable collections of **LPR**-grammars in a way analogous to $\mathcal{G}_{\text{bounded}}$ (recall Section 9.2.1).

9.8 Second Order Polymorphism

Consider the so-called *chameleon words* like the coordinating particles "and" and "or" (cited as examples in Lambek (1958)), negation, generalized quantifiers or relative pronouns.[9] Rather than assigning multiple types like $(np\backslash np)/np$ and $(s\backslash s)/s$ it would be desirable to capture the obvious generalization by some sort of type schemata. The use of *second order polymorphism* offers this kind of flexibility, it makes possible the assignment of type $\forall X.(X\backslash X)/X$ to "and". The second order polymorphic variant of **L** is written as **L2**, the second order polymorphic variant of **NL** as **NL2** etc. See Emms (1993a,b) for discussions of **L2**, a PSPACE-complete fragment of MALL2 (a logic related to **L2**) that allows analysis ofat least some of the discussed phenomena is presented in Perrier (1999).

Obviously rigid **L2** and rigid **NL2** are not learnable, given the results for rigid **L** and rigid **NL**. Still, it is interesting to see if there is a direct relation between polymorphism and learnability, if only to extend learnable subclasses of $\mathcal{G}$**L** or $\mathcal{G}$**NL**.

Note that in **L2** (as well as in **NL2**) the structural rules of Expansion, Weakening and Contraction can be (re)introduced with types $\forall X.X\backslash(X \bullet X)$, $\forall X \forall Y.X\backslash(X \bullet Y)$ (or $\forall X \forall Y.X\backslash(Y \bullet X)$), and $\forall X.(X \bullet X)\backslash X$, respectively.[10] We will use these facts to show non-learnability of a subclass of $\mathcal{G}$**NL2**, but their application is far from straightforward.

We will now investigate the learnability of a subclass of **NL2**-grammars. Note that scope and alphabetic variation may be implicit, the intended reading will be clear from context.

[9] It seems that all of these categories except the coordinating particles, can also be *monomorphically*typed.

[10] These types were used in Lincoln et al. (1995) (in the guise of **IMLL2** formulæ) to show that the (undecidable) system of second-order linear logic (**LJ2**) can be embedded in **IMLL2** (which is (almost) equivalent to **LP2**). Note that it is also possible to express other structural rules, for example Permutation with second-order type $\forall X \forall Y.((X \bullet Y)/X)/Y$.

$$\frac{\Gamma, A[B/X], \Delta \vdash C}{\Gamma, \forall X.A, \Delta \vdash C}\ [\forall L] \quad \frac{\Gamma \vdash A}{\Gamma \vdash \forall X.A[X/Y]}\ [\forall R(*)]$$

$$\frac{\Gamma \vdash A[B/X]}{\Gamma \vdash \exists X.A}\ [\exists R] \quad \frac{\Gamma, A, \Delta \vdash C}{\Gamma, \exists X.A[X/Y], \Delta \vdash B}\ [\exists L(*)]$$

Figure 9.1: Rules of inference for **L2**.

Definition 9.36 *For any $n \in \mathbb{N}^+$, let G_n be defined as*

$$G_n : \begin{array}{rcl} \mathtt{s} & \mapsto & s/\underbrace{\forall A \forall B \ldots.((A\backslash A) \bullet (A\backslash A)) \bullet ((B/B) \bullet (B/B)) \bullet \ldots}_{n \text{ times}} \\ \mathtt{a} & \mapsto & \forall A.(A\backslash A)\backslash(A\backslash A), \forall A \forall B \forall C.(B \bullet C)/(A\backslash A) \\ \mathtt{b} & \mapsto & \forall B.(B/B)/(B/B), \forall A \forall B \forall C.(B/B)\backslash(A \bullet C) \end{array}$$

and let G_* be defined as

$$G_* : \begin{array}{rcl} \mathtt{s} & \mapsto & s/b \\ \mathtt{a} & \mapsto & b/c, c/c, c, c/b \\ \mathtt{b} & \mapsto & b/d, d/d, d, d/b \end{array}$$

Proposition 9.37 *Let G_n be as described in Definition 9.36. For any $n \in \mathbb{N}^+$, $\mathrm{L}(G_n) = \mathtt{s}\mathtt{a}^{i_1}\mathtt{b}^{j_1} \ldots \mathtt{a}^{i_k}\mathtt{b}^{j_k}$, where $1 \leq k \leq n$, and for all i_t, j_t, $1 \leq t \leq k$, either*

1. *$i_t, j_t \geq 2$,*
2. *$i_t = 0$ and $j_t \geq 2$, or*
3. *$i_t \geq 2$ and $j_t = 0$.*

`Proof:`

1. `Proof:` of $\mathtt{s}\mathtt{a}^{i_1}\mathtt{b}^{j_1} \ldots \mathtt{a}^{i_k}\mathtt{b}^{j_k} \subseteq \mathrm{L}(G_n)$ (with restrictions on k and all $i_t, j_t, 1 \leq t \leq k$ as specified):

   Since the type that G_n assigns to `s` is $s/\underbrace{\forall A \forall B \ldots.((A\backslash A) \bullet (A\backslash A)) \bullet ((B/B) \bullet (B/B)) \bullet \ldots}_{n \text{ times}}$, it is sufficient to show that we can associate a sequent with $\mathtt{a}^{i_1}\mathtt{b}^{j_1} \ldots \mathtt{a}^{i_k}\mathtt{b}^{j_k}$ such that it derives $\underbrace{\forall A \forall B \ldots.((A\backslash A) \bullet (A\backslash A)) \bullet ((B/B) \bullet (B/B)) \bullet \ldots}_{n \text{ times}}$

   (a) For the case $i_t, j_t \geq 2$: assume that `s` is followed by $k \geq 2$ `a`'s, all associated with type $\forall A.(A\backslash A)\backslash(A\backslash A)$. Using $[\backslash E]$, the derivation $\forall A.(A\backslash A)\backslash(A\backslash A), \forall A.(A\backslash A)\backslash(A\backslash A) \vdash \forall A.(A\backslash A)\backslash(A\backslash A)$ can be obtained. This step can be applied $k-1$ times, then, followed by $[\bullet I]$, the type $\forall A.(A\backslash A)\backslash(A\backslash A) \bullet \forall A.(A\backslash A)\backslash(A\backslash A)$ is obtained.

Assume this is followed by $k \geq 2$ b's, all associated with type $\forall B.(B/B)/(B/B)$. Using $[/E]$, the derivation $\forall B.(B/B)/(B/B), \forall B.(B/B)/(B/B) \vdash \forall B.(B/B)/(B/B)$ can be obtained. This step can be applied $k-1$ times, then, followed by $[\bullet I]$, the type $\forall B.(B/B)/(B/B) \bullet \forall B.(B/B)/(B/B)$ is obtained. Applying another $[\bullet I]$, finally the type $(\forall A.(A\backslash A)\backslash(A\backslash A) \bullet \forall A.(A\backslash A)\backslash(A\backslash A)) \bullet (\forall B.(B/B)/(B/B) \bullet \forall B.(B/B)/(B/B))$ is obtained. This type has the same structure as any one of the n domain subtypes of
$s/\underbrace{\forall A \forall B \ldots.((A\backslash A) \bullet (A\backslash A)) \bullet ((B/B) \bullet (B/B)) \bullet \ldots.}_{n \text{ times}}$

(b) For the case $i_t = 0$ and $j_t \geq 2$: assume that s is followed by $k \geq 2$ b's, the first $k-1$ associated with type $\forall B.(B/B)/(B/B)$, the last one associated with $\forall A \forall B \forall C.(B/B)\backslash(A \bullet C)$. As we have seen, the sequent containing the first $k-1$ types can, with application of $[/E]$, derive type $\forall B.(B/B)/(B/B)$. By applying $[/E]$ to the resulting sequent we get $\forall B.(B/B)/(B/B), \forall A \forall B \forall C.(B/B)\backslash(A \bullet C) \vdash \forall A \forall C A \bullet C$. This type has the same structure as any one of the n domain subtypes of
$s/\underbrace{\forall A \forall B \ldots.((A\backslash A) \bullet (A\backslash A)) \bullet ((B/B) \bullet (B/B)) \bullet \ldots.}_{n \text{ times}}$

(c) For the case $i_t \geq 2$ and $j_t = 0$: assume that s is followed by $k \geq 2$ a's, the first one associated with $\forall A \forall B \forall C.(B \bullet C)/(A\backslash A)$, the last $k-1$ associated with type $\forall A.(A\backslash A)\backslash(A\backslash A)$. As we have seen, the sequent containing the last $k-1$ types can, with application of $[\backslash E]$, derive type $\forall A.(A\backslash A)\backslash(A\backslash A)$. By applying $[\backslash E]$ to the resulting sequent we get $\forall A \forall B \forall C.(B \bullet C)/(A\backslash A), \forall A.(A\backslash A)\backslash(A\backslash A) \vdash \forall B \forall C.(B \bullet C)$. This type has the same structure as any one of the n domain subtypes of
$s/\underbrace{\forall A \forall B \ldots.((A\backslash A) \bullet (A\backslash A)) \bullet ((B/B) \bullet (B/B)) \bullet \ldots.}_{n \text{ times}}$

2. Proof: of $\mathrm{L}(G_n) \subseteq \mathtt{s}\mathtt{a}^{i_1}\mathtt{b}^{j_1} \ldots \mathtt{a}^{i_k}\mathtt{b}^{j_k}$:

   Left to the reader.

□

Proposition 9.38 *Let G_* be as described in Definition 9.36. Then* $\mathrm{L}(G_*) = \cup \mathtt{s}\mathtt{a}^{i_1}\mathtt{b}^{j_1} \ldots \mathtt{a}^{i_k}\mathtt{b}^{j_k}$ *where $k \in \mathbb{N}^+$ and for all i_t, j_t, $1 \leq t \leq k$, either*

1. *$i_t, j_t \geq 2$,*
2. *$i_t = 0$ and $j_t \geq 2$, or*

3. $i_t \geq 2$ *and* $j_t = 0$.

`Proof:` None of the types assigned in G_* contains a variable, so its interpretation under **NL** results in the same language as under **NL2**. Since the degree of none of the types assigned in G_* is greater than 1 we need not concern ourselves with hypothetical reasoning; the interpretation under CCG rules yields the same result. So, the only relevant derivation subtrees for G_n are the ones shown in Figure 9.2.

$$Tree(s) = \dfrac{\mathtt{s} : s/b \qquad \dfrac{Tree(b)}{b}}{s}\,[/E]$$

$$Tree(b)_1 = \dfrac{\mathtt{a} : b/c \qquad \dfrac{Tree(c)}{c}}{b}\,[/E] \qquad Tree(c)_1 = \dfrac{\mathtt{a} : c/c \qquad \dfrac{Tree(c)}{c}}{c}\,[/E] \qquad Tree(d)_1 = \dfrac{\mathtt{b} : d/d \qquad \dfrac{Tree(d)}{d}}{d}\,[/E]$$

$$Tree(c)_2 = \dfrac{\mathtt{a} : c}{c} \qquad Tree(d)_2 = \dfrac{\mathtt{b} : d}{d}$$

$$Tree(b)_2 = \dfrac{\mathtt{b} : b/d \qquad \dfrac{Tree(d)}{d}}{b}\,[/E] \qquad Tree(c)_3 = \dfrac{\mathtt{a} : c/b \qquad \dfrac{Tree(b)}{b}}{c}\,[/E] \qquad Tree(d)_3 = \dfrac{\mathtt{b} : d/b \qquad \dfrac{Tree(b)}{b}}{d}\,[/E]$$

Figure 9.2: Subtrees for G_*.

Note that all operators found in the types assigned in G_n are $/$, so all symbols produced during derivation are attached to the right of the string to be generated.

Since any sentence must correspond to a sequent deriving s, $Tree(s)$ is the only starting option, wich yields the initial string `s`. The next subtree needed to complete the derivation is $Tree(b)_x$, where $1 \leq x \leq 2$. We will show by induction that only strings of the form $\mathtt{a}^{i_1}\mathtt{b}^{j_1} \ldots \mathtt{a}^{i_k}\mathtt{b}^{j_k}$ can derive b (again, $k \in \mathbb{N}^+$ and for all i_t, j_t, $1 \leq t \leq k$, either $i_t, j_t \geq 2$, or $i_t = 0$ and $j_t \geq 2$, or $i_t \geq 2$ and $j_t = 0$).

Consider $Tree(b)_x$, where $1 \leq x \leq 2$.

1. If we choose $x = 1$, the symbol `a` is added to the string, and the next subtree must be $Tree(c)_y, 1 \leq y \leq 3$:

   (a) (base) In the case of $y = 2$, another `a` is added to the string, and the derivation is terminated.

   (b) (Induction step) In the case of $y = 1$, another `a` is added to the string, and another $Tree(c)$ is added to the derivation.

   (c) (Induction step) In the case of $y = 3$, another `a` is added to the string, and a $Tree(b)$ is added to the derivation.

   Thus the string $\mathtt{a}^i, i \geq 2$ is added, and the derivation either terminates or continues with $Tree(b)$. In the former case, we obtain the string $\sigma + \mathtt{a}^{i_t}\mathtt{b}^{j_t}$, $i_t \geq 2$ and $j_t = 0$. In the latter case either more `a`'s are added ($x = 1$),

or a string of b's is appended ($x = 2$), which corresponds to $\mathtt{a}^{i_t}\mathtt{b}^{j_t}\mathtt{a}^{i_{t+1}}$, $i_t, i_{t+1} \geq 2$ and $j_t = 0$, and $\mathtt{a}^{i_t}\mathtt{b}^{j_t}$, $i_t, j_t \geq 2$, respectively.

2. If we choose $x = 2$, the symbol b is added to the string, and the next subtree must be $\mathit{Tree}(d)_y, 1 \leq y \leq 3$:

   (a) (base) In the case of $y = 2$, another b is added to the string, and the derivation is terminated.

   (b) (Induction step) In the case of $y = 1$, another b is added to the string, and another $\mathit{Tree}(d)$ is added to the derivation.

   (c) (Induction step) In the case of $y = 3$, another b is added to the string, and a $\mathit{Tree}(b)$ is added to the derivation.

   Thus the string $\mathtt{b}^j, j \geq 2$ is added, and the derivation either terminates or continues with $\mathit{Tree}(b)$. In the former case, we obtain the string $\sigma + \mathtt{a}^{i_t}\mathtt{b}^{j_t}$, $i_t = 0$ and $j_t \geq 2$. In the latter case either more b's are added ($x = 2$), or a string of a's is appended ($x = 1$), which corresponds to $\mathtt{b}^{j_t}\mathtt{a}^{i_{t+1}}\mathtt{a}^{j_{t+1}}$, $j_t, j_{t+1} \geq 2$ and $i_{t+1} = 0$, and $\mathtt{b}^{j_t}\mathtt{a}^{i_{t+1}}$, $j_t, i_{t+1} \geq 2$, respectively.

This shows that $\mathrm{L}(G_*) = \cup \mathtt{s}\mathtt{a}^{i_1}\mathtt{b}^{j_1} \ldots \mathtt{a}^{i_k}\mathtt{b}^{j_k}$ where $k \in \mathbb{N}^+$ and for all i_t, j_t, $1 \leq t \leq k$, either

1. $i_t, j_t \geq 2$,
2. $i_t = 0$ and $j_t \geq 2$, or
3. $i_t \geq 2$ and $j_t = 0$.

□

Theorem 9.39 *The class* $\mathcal{G}_{4\text{-valued}}\mathbf{NL2}$ *has a limit point and is thus not (non-effectively) learnable.*

Proof: From Proposition 9.37 we can conclude that $\mathrm{L}(G_1) \subset \mathrm{L}(G_2) \subset \ldots$, so the class contains an infinite ascending chain. From Proposition 9.38 it follows that $\mathrm{L}(G_*) = \bigcup_{i=1}^{\infty} \mathrm{L}(G_i)$, so $\mathrm{L}(G_*)$ is a limit point for the class. □

Note that the proof only holds for the bi-directional case with product.

We conjecture that there is a version of G_* that assigns less types to any given symbol, so the result probably holds for $\mathcal{G}_{k\text{-valued}}\mathbf{NL2}$ with $k < 4$.

9.9 Concluding Remarks

In this chapter we have shown that the classes of rigid **LP** and $\mathbf{LP}_\emptyset$ grammars have limit points and are thus not learnable from strings. These results, as well as the negative results from Foret and Le Nir (2002a), Foret and Le Nir (2002b)

and Bechet and Foret (submitted) are quite surprising in the light of some of Shinohara's general results. In Kanazawa (1998) it was suggested that placing a numerical bound on the complexity of a grammar will *typically* lead to a non-trivial learnable class. The negative results for Lambek(like) systems show that this is not always the case. Even placing bounds on the complexity of the types appearing in the grammar may not help: rigid $\mathbf{L}$ is not even learnable when the order of types is bounded to 2.

A final thought concerns the claim in Foret and Le Nir (2002a) and Foret and Le Nir (2002b) that these results demonstrate the paucity of 'flat' strings as input for a learner. They suggest that enriched input (i.e. some kind of bracketing or additional semantic information) may overcome this problem, which is certainly an interesting approach. However, one could also take another approach to constructing classes learnable from strings within some Lambek(like) calculus by restricting the use of postulates. The multimodal approach (see for example Moortgat and Morrill (1991)) offers a way of doing this in the lexicon. The viability of this approach is of course dependent on the existence of learnable subclasses of the class $\mathcal{G}\mathbf{NL}$, though even given such a class it would be a highly nontrivial enterprise. Thus, we feel that these results offer no evidence either against or in favour of the claim that semantic information or any other kind of enriched input is used during language acquisition.

Table 9.1 sums up the results discussed in this chapter. These results are negative, but one could also interpret them positively: they show that Lambek systems are in some sense far more 'concise' than CCG systems. A negative interpretation may also be premature in the sense that there may be 'natural' learnable classes of Lambek grammars that bear no resemblance whatsoever to classes defined in terms of rigidity, k-valuedness etc. In other words, lots of open questions remain.

We conjecture $\mathcal{G}_{\text{rigid}}\mathbf{NLP}$, $\mathcal{G}_{\text{rigid}}\mathbf{NLP}_{\emptyset}$, $\mathcal{G}_{\text{rigid}}\mathbf{NL} + C$ (unidirectional, $\bullet$), $\mathcal{G}_{\text{rigid}}\mathbf{NL} + E$ (unidirectional, $\bullet$) and $\mathcal{G}_{\text{rigid}}\mathbf{NL} + W$ (unidirectional, $\bullet$) not to be learnable. We also conjecture that $\mathcal{G}_{\text{rigid}}\mathbf{LP}, \mathcal{G}_{\text{rigid}}\mathbf{LP}_{\emptyset}$ without $\bullet$ are not learnable.

Note that there are some positive results: see Proposition 9.9 for a learnable class of $\mathbf{NL}$-grammars and Corollary 9.10 for a learnable class of rigid $\mathbf{NL}\diamond_{\mathcal{R}-}$-grammars, but these results are in itself not very interesting. Also note that for subclasses of $\mathcal{G}_{\text{rigid}}\mathbf{NL}$ that are restricted so that for any type T occuring in any grammar in such a subclass $1 < \text{order}(T) < 5$, the question of learnability is still open.

The negative results seem to warrant the conclusion that 'natural' classes of type-logical systems are too powerful to be learnable, since an important axioms like Lifting gives rise to the existence of limit points. It is possible that restricting the use of such axioms[12] in combinatory systems permits the existence of non-trivial learnable classes. Such restrictions on the use of axioms may seem ad-hoc, but they have been proposed on purely linguistic grounds

[12]Note that Lifting is already subject to certain restrictions in GCG.

Class[11]	Learnable (strings)	Finite Elasticity
$\mathcal{G}_{\text{rigid}}\mathbf{L}$ (unidirectional)	no	no, limit point
$\mathcal{G}_{\text{rigid}}\mathbf{L}_{\emptyset}$ (unidirectional)	no	no
$\mathcal{G}_{\text{rigid}}\mathbf{LP}$ (bidirectional, $\bullet$)	no	no
$\mathcal{G}_{\text{rigid}}\mathbf{LP}_{\emptyset}$ (bidirectional, $\bullet$)	no	no, limit point
$\mathcal{G}_{\text{rigid}}\mathbf{NL}$ (bidirectional)	no	no, limit point
$\mathcal{G}_{\text{rigid}}\mathbf{NL}_{\emptyset}$ (unidirectional)	no	no, limit point
$\mathcal{G}_{\text{rigid}}\mathbf{NLP}$	?	?
$\mathcal{G}_{\text{rigid}}\mathbf{NLP}_{\emptyset}$	?	?
$\mathcal{G}_{\text{rigid}}\mathbf{NL}+C$ (unidirectional, $\bullet$)	?	no, i.a.c.
$\mathcal{G}_{\text{rigid}}\mathbf{NL}+E$ (unidirectional, $\bullet$)	?	no, i.a.c.
$\mathcal{G}_{\text{rigid}}\mathbf{NL}+W$ (unidirectional, $\bullet$)	?	no, i.a.c.
$\mathcal{G}_{\text{2-valued}}\mathbf{NL}+E$ (unidirectional, $\bullet$)	no	no, limit point
$\mathcal{G}_{\text{2-valued}}\mathbf{NL}+W$ (unidirectional, $\bullet$)	no	no, limit point
$\mathcal{G}_{\text{4-valued}}\mathbf{NL2}$ (bidirectional, $\bullet$)	no	no, limit point

Table 9.1: Summary of known learnability results for rigid classes of Lambek grammars.

(Steedman (2000), for the analysis of Dutch, for example). A more elegant way of imposing such restrictions has been proposed in Baldridge and Kruijff (2003), where a multimodal variant of combinatory CG is defined.

[12]The calculi do not use product ($\bullet$), unless stated otherwise.

Chapter 10

Conclusions

The specific technical results discussed in the preceding chapters suggest some general conclusions:

1. *Formal Learning Theory is relevant to linguistics, particularly to language acquisition research. Applying Formal Learning Theory to linguistics is also a highly non-trivial enterprise.* Contrary to what is often suggested in linguistic literature, results from Gold (1967) did *not* show that identification in the limit is not feasible for non-trivial classes, nor did developments in this field stop after the publication of this paper.

   Research has steadily continued since the late sixties, and blossomed after some impressive results were obtained in the early eighties. It has lead to deep and often surprising insights that cannot be ignored by any cognitive scientist interested in the nature of learning.

2. *Imposing a numerical bound on the complexity of a grammar* can *lead to a learnable class, but this completely depends on both the formalism and the specific notion of complexity.* To quote Kanazawa (1998), page 159:

   > *Placing a numerical bound on the complexity of a grammar can lead to a non-trivial learnable class.* [...] Together with Shinohara's (Shinohara (1990a), Shinohara (1990b)) earlier result [a class of context-sensitive grammars having at most k rules is learnable], this suggests that something like this may in fact turn out to be typical in learnability theory.

   The results discussed in this thesis certainly show that there are 'natural' counterexamples to this conjecture: rigid combinatory CG may not be learnable, depending on the combinatory rules used. The class of rigid TAGs isn't learnable either, nor are subclasses of this class that fulfill severe additional constraints. The same is true of Minimalist Grammars and of the Lambek-systems **NL**, **L**, **LP** and their variants, including

pregroup grammars. In the case of **NL** and **LP** the nonlearnability is due to properties of Lifting, an operation that seems to be essential for any strongly compositional system.

3. *Learning is hard work.* Many different definitions of complexity of learning can be found in the literature, though none of them seem to capture an intuitive notion of 'learnable by a feasibly computable function'. This is due to the fact that the paradigm of identification in the limit imposes only weak restrictions on the nature of the input, see Section 2.9 for a discussion.

   However, we have obtained some negative complexity results for Kanazawa's classes of categorial grammar: in Section 5.1 a proof is given of the existence of an exponential bound on the number of candidate grammars generated by the algorithms presented in Chapter 4, and in Chapter 5 results are discussed concerning the complexity of producing a conjecture consistent with the given data while falling within a specified class. These results suggest that learning is hard even given some class that has only a moderately rich structure and contains just modestly expressive languages.

4. *A good, coherent notion of strong generative capacity (derivation- or dependency structure or otherwise) is important and useful when dealing with learnability issues of a class of grammars.* Kanazawa's theorem on infinite elasticity as first published in Kanazawa (1994b) (Theorem 2.27 in this thesis) implies that if a tree language has finite elasticity, then so does the string language that is its yield, provided that there is only a finite number of possible trees that have any given string as its yield.

   This seems a perfectly reasonable condition in a linguistic context, but when dealing with Lambek grammars for example things are more complicated than they may appear at first sight. Explicit descriptions of trees have no place in categorial grammar, since strings are obtained from *proofs*, not derivations. There is a (spurious) non-determinism inherent in the standard notion of categorial proof, so any 'naive' notion of strong generative capacity is structurally complete and therefore useless for our purposes. In order to be able to apply Kanazawa's theorem in this context we need a normal form for such proofs, such as provided by Tiede (1999a) for proof trees of **L** and **NL** grammars. However, the tree languages corresponding to these normal forms have not yet been explicitly characterized.

   From a more technical perspective, thsi thesis offers good evidence for the idea that tree automata-techniques offer a promising approach for dealing with learnability of linguistic formalisms, and especially for designing algorithms.

5. *Structural properties of classes of languages, like having an infinite ascending chain, a limit- or accumulation point, and especially having (in-) finite elasticity, play an important role in Formal Learning Theory.* Almost all the learnability results discussed in this thesis are based to some degree on these structural properties. Finite elasticity is an especially useful concept. It is a natural property and easy to preserve, and, as already mentioned, it can be used to establish a link between structure languages and string languages.

Index

Bibliography

Naoki Abe. Polynomial Learnability of Semilinear Sets. In *The 1989 Workshop on Computational Learning Theory*, pages 25–40. San Mateo: Morgan Kaufman, 1989.

P. Adriaans, H. Fernau, and M. van Zaanen, editors. *Grammatical Inference: Algorithms and Applications, 6th International Colloquium, ICGI 2002, Amsterdam, The Netherlands, September 2000, Proceedings*, volume 2484 of *Lecture Notes in Artificial Intelligence*, Amsterdam, The Netherlands, September 23–25 2002. Springer-Verlag.

Alfred V. Aho. Indexed grammars. *Journal of the Association for Computing Machinery*, 15:647–671, 1968.

M. A. Aizerman, E. M. Braverman, and L. I. Rozonoer. Theoretical foundations of the potential function method in pattern recognition. *Automation and Remote Control*, 25:821–837 and 1175–1190, 1964.

Kazimierz Ajdukiewicz. Die syntaktische Konnexität. *Stud. Philos.*, 1:1–27, 1935.

Andris Ambainis, Sanjay Jain, and Arun Sharma. Ordinal mind change complexity of language identification. In S. Ben-David, editor, *Third European Conference on Computational Learning Theory*, volume 1208 of *Lecture Notes in Artificial Intelligence*, pages 301–315. Springer-Verlag, 1997.

Dana Angluin. Finding common patterns to a set of strings. In *Proceedings of the 11th Annual Symposium on Theory of Computing*, pages 130–141, 1979.

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

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

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

Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75:87–106, 1987.

Dana Angluin. Negative results for equivalence queries. *Machine Learning*, 5: 121–150, 1990.

Setsuo Arikawa, Takeshi Shinohara, and Akihiro Yamamoto. Elementary formal system as a unifying framework for language learning. In *Proceedings of the Second Annual Workshop on Computational Learning Theory*, pages 312–327, University of California, Santa Cruz, 31 July–2 August 1989. ACM Press.

Hiroki Arimura, Hiroki Ishizaka, and Takeshi Shinohara. Polynomial time inference of a subclass of context-free transformations. In *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*, pages 136–143, Pittsburgh, Pennsylvania, 27–29 July 1992. ACM Press.

Peter R. J. Asveld and Anton Nijholt. The inclusion problem for some subclasses of context-free languages. *Theoretical Computer Science*, 230:247–256, 2000.

Martin Atkinson. *Children's Syntax. An Introduction to Principles and Parameters Theory.* Oxford: Blackwell, 1992.

Franz Baader and Jörg H. Siekmann. Unification theory. In *Gabbay et al. (1997)*. Oxford University Press, 1993.

C. L. Baker and J. J. McCarthy, editors. *The Logical Problem of Language Acquisition.* Cambridge, Mass.: MIT Press, 1981.

Jason Baldridge. *Lexically Specified Derivational Control in Combinatory Categorial Grammar.* PhD thesis, University of Edinburgh, 2002.

Jason Baldridge and Geert-Jan M. Kruijff. Multi-modal Combinatory Categorial Grammar. In Ann Copestake and Jan Hajič, editors, *Proceedings of EACL2003, Tenth Conference of the European Chapter of the Association for Computational Linguistics*, pages 211–218, Agro Hotel, Budapest, April 12–17 2003.

Yehoshua Bar-Hillel. A quasi-arithmetical notation for syntactic description. *Language*, 29:47–58, 1953.

Reuven Bar-Yehuda and Shimon Even. A local-ratio theorem for approximating the weighted vertex cover problem. In *Analysis and Design of Algorithms for Combinatorial Problems*, volume 25 of *Annals of Disc. Math.*, pages 27–46. Elsevier science publishing company, Amsterdam, 1985.

John Barwise and Robin Cooper. Generalized quantifiers and natural language. *Linguistics and Philosophy*, 4(2):159–219, 1981.

Janis Bārzdiņš. Inductive inference of automata, functions and programs. In *Proceedings International Congress of Math.*, pages 455–460, Vancouver, 1974.

Denis Bechet and Annie Foret. k-valued non-associative Lambek categorial grammars are not learnable from strings, submitted.

Marc Bernard and Colin de la Higuera. GIFT: Grammatical inference for terms. Late Breaking Paper at the International Conference on Inductive Logic Programming ILP, 1999. French version: Bernard and de la Higuera (2001).

Marc Bernard and Colin de la Higuera. Apprentisage de programmes logiques par inférence grammaticale. *Revue d'Intelligence Artificielle*, 14:375–396, 2001.

Stefano Bertolo. *A brief overview of learnability*, chapter 1, pages 1–14. Cambridge University Press, Cambridge, UK, 2001.

Robert Berwick. *Locality Principles and the Acquisition of Syntactic Knowledge*. PhD thesis, Department of Electrical Engineering and Computer Science, MIT, 1982. unpublished doctoral dissertation.

Robert Berwick. *The Acquisition of Syntatic Knowledge*. MIT Press, Cambridge, Massachusetts, 1985.

Jérôme Besombes and Jean-Yves Marion. Identification of reversible dependency tree languages. In Popelínský and Nepil (2001), pages 11–22. Technical report FIMU-RS-2001-08, 66 Pages, available from `http://www.fi.muni.cz/ilpnet2/LLL2001/proceedings.ps`.

Jérôme Besombes and Jean-Yves Marion. Efficient learning of regular tree languages and applications. Submitted, available as `http://www.loria.fr/~marionjy/Recherche/Papers/BesombesMarion.ps`, 4 February 2002a.

Jérôme Besombes and Jean-Yves Marion. Learning languages from positive examples with dependencies. In Frank (2002), pages 25–29.

Alan W. Biermann and Jerome A. Feldman. On the synthesis of finite state machines from samples of their behavior. In *IEEE Trans. on Computers*, pages 592–597, 1972.

Leonore Blum and Manuel Blum. Inductive inference: A recursion theoretic approach. In *FOCS 1973, IEEE Symposium on Foundations of Computer Science: 14th Annual Symposium on Switching and Automata Theory*, pages 200–208, 1973.

Leonore Blum and Manuel Blum. Toward a mathematical theory of inductive inference. *Information and Control*, 28(2):125–155, 1975.

Roberto Bonato and Christian Retoré. Learning rigid Lambek grammars and Minimalist Grammars from structured sentences. In Popelínský and Nepil (2001), pages 23–34. Technical report FIMU-RS-2001-08, 66 Pages, available from `http://www.fi.muni.cz/ilpnet2/LLL2001/proceedings.ps`.

Roger Brown and Camille Hanlon. Derivational complexity and order of acquisition in child speech. In *Hayes (1970)*. New York: Wiley, 1970.

Wojciech Buszkowski. Generative capacity of nonassociative Lambek calculus. *Bulletin of the Polish Academy of Science and Mathematics*, 35:507–516, 1986.

Wojciech Buszkowski. Discovery procedures for categorial grammars. In E. Klein and J. van Benthem, editors, *Categories, Polymorphism and Unification*. University of Amsterdam, 1987a.

Wojciech Buszkowski. Solvable problems for classical categorial grammars. *Bulletin of the Polish Academy of Sciences: Mathematics*, 34:373–382, 1987b.

Wojciech Buszkowski. *Logical Foundations of Ajdukiewicz and Lambek Categorial Grammars*. Polish Scientific Publishers, Warsaw, 1989. in Polish.

Wojciech Buszkowski. Categorial grammars with negative information. In H. Wansing, editor, *Negation. A notion in focus*. de Gruyter, Berlin, 1995.

Wojciech Buszkowski. Mathematical linguistics and proof theory. In van Benthem and ter Meulen (1997), chapter 12, pages 683–736.

Wojciech Buszkowski. Lambek calculus with nonlogical axioms. In C. Casadio, R. Seely, and P. J. Scott, editors, *A Festschrift for Joachim Lambek*. submitted to CSLI Lecture Notes, To appear.

Wojciech Buszkowski and Gerald Penn. Categorial grammars determined from linguistic data by unification. *Studia Logica*, 49:431–454, 1990.

Bob Carpenter. The Turing Completeness of Multimodal Categorial Grammars. In Jelle Gerbrandy, Maarten Marx, Maarten de Rijke, and Yde Venema, editors, *JFAK. Essays Dedicated to Johan van Benthem on the Occasion of his 50th Birthday*, Vossiuspers. Amsterdam University Press, Amsterdam, 1999.

Rafael C. Carrasco, Jose Oncina, and Jorge Calera. Stochastic inference of regular tree languages. In *Grammatical Inference, 4th International Colloquium, ICGI-98, Ames, Iowa, USA, July 1998, Proceedings*, volume 1433 of *Lecture Notes in Artificial Intelligence*, pages 187–198. Springer, 1998.

Claudia Casadio. Semantic categories and the development of categorial grammars. In Oehrle et al. (1988), pages 95–124.

Claudia Casadio and Joachim Lambek. A tale of four grammars. *Studia Logica*, 71(3):315–329, 2002.

Francisco Cascuberta and Colin de la Higuera. Computational complexity of problems on probabilistic grammars and transducers. In *ICGI 2000 (International Colloquium on Grammatical Inference), proceedings*, volume 1891 of *Lecture Notes in Artificial Intelligence*, pages 141–156. Springer-Verlag, 2000.

John Case and Sanjay Jain. Synthesizing learners tolerating computable noisy data. In *Algorithmic Learning Theory, 9th International Conference, ALT '98, Otzenhausen, Germany, October 1998, Proceedings*, volume 1501 of *Lecture Notes in Artificial Intelligence*, pages 205–219. Springer, 1998.

John Case, Sanjay Jain, and Arun Sharma. Synthesizing noise-tolerant language learners. In Li and Maruoka (1997), pages 228–243.

John Case, Sanjay Jain, and Frank Stephan. Vacillatory and BC learning on noisy data. In *Theoretical Computer Science, Special Issue for ALT'96*, 1997b. accepted 1997.

John Case and Christopher Lynes. Machine inductive inference and language identification. In M. Nielsen and E. M. Schmidt, editors, *Proceedings of the 9th International Colloquium on Automata, Languages and Programming* , volume 140 of *Lecture Notes in Computer Science*, pages 107–115. Springer-Verlag, 1982.

John Case and S. Ngo-Manuelle. Refinements of inductive inference by popperian machines. Technical report, Department of Computer Science, SUNY, Buffalo, 1979.

Noam Chomsky. Review of Skinner, *Verbal Behaviour*. *Language*, 35:26–58, 1959.

Noam Chomsky. *Aspects of the Theory of Syntax*. MIT Press, Cambridge, Mass., 1965a.

Noam Chomsky. *Aspects of the Theory of Syntax*. MIT Press, Cambridge, MA, 1965b.

Noam Chomsky. Rules and representations. *Behavioral and Brain Sciences*, 3 (1):1–15, 1980.

Noam Chomsky. *Lectures on Government and Binding*, volume 9 of *Studies in Generative Grammar*. Foris Publications, Dordrecht, 1981.

Noam Chomsky. *Knowledge of Language*. Praeger Publications, New York, 1986.

Noam Chomsky. A minimalist program for linguistic theory. MIT working papers in linguistics, MIT, Cambridge, MA, 1992.

Noam Chomsky. *The Minimalist Program*. MIT Press, Cambridge, MA, 1995.

Noam Chomsky. Minimalist inquiries: the framework. MIT Occasional Papers in Linguistics 15, MIT, Cambridge, MA, 1998.

Robin Clark. Learning First Order Quantifier Denotations: An Essay in Semantic Learnability. `http://www.ling.upenn.edu/~rclark/acq.html`, 1996.

Hubert Comon, Max Dauchet, Remi Gilleron, Florent Jacquemard, Denis Lugiez, Sophie Tison, and Marc Tommasi. Tree automata techniques and applications. Available on: `http://www.grappa.univ-lille3.fr/tata`, 1997.

Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, Mass., eighteenth edition, 1990.

Thomas Cornell and James Rogers. Model theoretic syntax. In *The Glot International State of the Article Book I*. Holland Academic Graphics, to appear.

Thomas L. Cornell. Lambek calculus for transformational grammar. In *Proceedings of the Workshop on Resource Logics and Minimalist Grammars, ESSLLI'99*, Utrecht, 1999.

Christophe Costa Florêncio. Efficient learning algorithms for some classes of categorial grammars. Master's thesis, Universiteit Utrecht, 1998a.

Christophe Costa Florêncio. Rats on film. Rudolf Magnus Institute, Utrecht University, 1998b.

Christophe Costa Florêncio. Consistent identification in the limit of some of Penn and Buszkowski's classes is NP-hard. In Paola Monachesi, editor, *Computational Linguistics in the Netherlands 1999*, pages 1–12, 2000a.

Christophe Costa Florêncio. On the complexity of consistent identification of some classes of structure languages. In Arlindo L. Oliveira, editor, *Grammatical Inference: Algorithms and Applications*, volume 1891 of *Lecture Notes in Artificial Intelligence*, pages 89–102. Springer-Verlag, 2000b.

Christophe Costa Florêncio. An avalanche of hypotheses. In notes of Workshop on Logic and Learning, affiliated with the IEEE, Symposium on Logic in Computer Science, LICS 2001, June 19–20 2001a.

Christophe Costa Florêncio. Combinatory categorial grammars and finite elasticity. In Véronique Hoste and Guy De Pauw, editors, *Proceedings of the Eleventh Belgian-Dutch Conference on Machine Learning*, pages 13–18. University of Antwerp, 2001b.

Christophe Costa Florêncio. Conservative vs set-driven learning functions for the class k-valued. In Mariet Theune, Anton Nijholt, and Hendri Hondorp, editors, *Proceedings of Computational Linguistics in the Netherlands, Selected Papers from the Twelfth CLIN Meeting*, volume 45 of *Language and Computers: Studies in Practical Linguistics*, pages 38–46. Amsterdam, New York, 2001c. Series edited by Jan Aarts and Willem Meijs.

Christophe Costa Florêncio. Consistent Identification in the Limit of the Class k-valued is NP-hard. In de Groote et al. (2001), pages 125–138.

Christophe Costa Florêncio. Consistent Identification in the Limit of Rigid Grammars from Strings is NP-hard. In Adriaans et al. (2002), pages 49–62.

Christophe Costa Florêncio. Language, learning and complexity. In Sergio Baauw, Mike Huiskes, and Maaike Schoorlemmer, editors, *Utrecht Institute of Linguistics OTS Yearbook 2002*, pages 1–16. 2002b.

Christophe Costa Florêncio. Learning generalized quantifiers. In M. Nissim, editor, *Proceedings of the ESSLLI02 Student Session*, pages 31–40. University of Trento, 2002c.

Christophe Costa Florêncio. A Limit Point for Rigid Associative-Commutative Lambek Grammars. In Ann Copestake and Jan Hajič, editors, *Proceedings of EACL2003, Tenth Conference of the European Chapter of the Association for Computational Linguistics*, pages 75–82, Agro Hotel, Budapest, April 12–17 2003.

Christophe Costa Florêncio. Complexity results for learning Categorial Grammars - an overview. To appear.

Haskell Brookes Curry and Robert Feys. *Combinatory Logic, Volume I.* Studies in Logic. North-Holland,Amsterdam, the Netherlands, 1958.

Robert P. Daley and Carl H. Smith. On the complexity of inductive inference. *Information and Control*, 69:12–40, 1986.

Philippe de Groote. The non-associative Lambek calculus with product in polynomial time. *Lecture Notes in Artificial Intelligence*, 1617:128–139, 1999.

Philippe de Groote and François Lamarche, editors. *The Curry-Howard isomorphism*, volume 8 of *Cahiers du Centre de Logique*. Academia, Louvain-La-Neuve, 1995. Université catholique de Louvain, Département de Philosophie.

Philippe de Groote, Glyn Morrill, and Christian Retoré, editors. *Logical Aspects of Computational Linguistics, 4th International Conference, LACL 2001, Le Croisic, France, June 27–29, 2001, Proceedings*, volume 2099 of *Lecture Notes in Computer Science*. Springer, 2001.

Dick de Jongh and Makoto Kanazawa. Angluin's theorem for indexed families of r.e. sets and applications. In *Proceedings of the Ninth Annual Conference on Computational Learning Theory, COLT '96*, pages 193–204. The Association for Computing Machinery, New York, 1996.

Colin de la Higuera. Characteristic sets for polynomial grammatical inference. *Machine Learning*, 27:125–137, 1997.

Colin de la Higuera, J. Oncina, and E. Vidal. Identification of DFA: datadependent versus data-independant algorithms. In *ICGI-96*, pages 311–325. Springer-Verlag, 1996.

Alexander Ja. Dikovsky. Polarized non-projective dependency grammars. In de Groote et al. (2001), pages 139–157.

Alexander Ja. Dikovsky and Larisa S. Modina. Dependencies on the other side of the curtain. *Traitement automatique des langues*, 41(1):67–96, 2000.

Kosta Došen. A brief survey of frames for the Lambek calculus. *Zeitschrift für mathematischen Logik und Grundlagen der Mathematik*, 38:179Ŭ–187, 1992.

David Dowty. Type raising, functional composition, and non-constituent conjunction. In Oehrle et al. (1988), pages 153–197.

Mark Dras, David Chiang, and William Schuler. A multi-level TAG approach to dependency. *Journal of Language and Computation*, to appear.

Martin Emms. Parsing with polymorphism. In *Proceedings of the Sixth Conference of the European ACL*, pages 120–129, Utrecht, 1993a.

Martin Emms. Some applications of categorial polymorphism. In M. Moortgat, editor, *Polymorphic Treatments*, pages 1–52. ILLC, University of Amsterdam, 1993b.

Heidi Feldman, Susan Goldin-Meadow, and Lila Gleitman. Beyond Herodotus: The creation of language by linguistically deprived deaf children. In A. Lock, editor, *Action, Symbol, and Gesture: The Emergence of Language*. Academic Press, San Diego, 1978.

Henning Fernau. On learning function distinguishable languages. Technical Report WSI–2000–13, Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, 2000.

Henning Fernau. Learning XML grammars. *Machine Learning and Data Mining in Pattern Recognition*, pages 73–87, 2001.

Henning Fernau. Learning tree languages from text. In *15th Annual Conference on Computational Learning Theory, COLT 2002, Sydney, Australia, July 2002, Proceedings*, volume 2375 of *Lecture Notes in Artificial Intelligence*, pages 153–168. Springer, 2002.

Annie Foret. Conjoinability and unification in Lambek categorial grammars. In *V Roma Workshop, proceedings to be published in Papers in Formal Linguistics and Logic series*, 2001a.

Annie Foret. On mixing deduction and substitution in Lambek Categorial Grammars. In de Groote et al. (2001), pages 158–174.

Annie Foret and Yannick Le Nir. Lambek rigid grammars are not learnable from strings. In *Proceedings of the 19th International Conference on Computational Linguistics (COLING 2002), Taipei, Republic of China (Taiwan)*. Morgan Kaufmann Publishers and ACL, 2002a. available as `http://perso.ifsic.univ-rennes1.fr/foret/learn.ps`.

Annie Foret and Yannick Le Nir. On limit points for some variants of rigid Lambek grammars. In Adriaans et al. (2002), pages 49–62.

Robert Frank, editor. *Proceedings of the Sixth International Workshop on Tree Adjoining Grammars and Related Frameworks, TAG+6*, 20–23 May 2002.

Rūsiņš Freivalds, Efim Kinber, and Carl H. Smith. On the intrinsic complexity of learning. In Paul Vitányi, editor, *Second European Conference on Computational Learning Theory*, volume 904 of *Lecture Notes in Artificial Intelligence*, pages 154–168. Springer-Verlag, 1995.

Rūsiņš Freivalds and R. Wiehagen. Inductive inference with additional information. *Elektronische Informationsverarbeitung und Kybernetik*, 15:179–185, 1979.

H. Fukuda and K. Kamata. Inference of tree automata from sample set of trees. *International Journal of Computer and Information Sciences*, 13(3): 177–196, 1984.

Mark Fulk. Saving the phenomena: Requirements that inductive machines not contradict known data. *Information and Computation*, 79(3):193–209, 1988.

Mark Fulk. Prudence and other conditions on formal language learning. *Information and Computation*, 85:1–11, 1990a.

Mark Fulk. Robust separations in inductive inference. In *31st Annual IEEE Symposium on Foundations of Computer Science*, pages 405–410, 1990b.

Mark Fulk, Sanjay Jain, and Daniel Osherson. Open problems in systems that learn. *J. Comput. System Sci.*, 49(3):589–604, 1994.

Dov M. Gabbay, C. J. Hogger, and J. A. Robinson, editors. *Handbook of Logic in Artificial Intelligence and Logic Programming*. Oxford: Oxford University Press, 1997.

Haim Gaifman. Dependency systems and phrase structure systems. *Information and Control*, 8(3):304–337, 1965.

Michael R. Garey and David S. Johnson. *Computers and Intractability. A Guide to the Theory of NP-completeness*. Freeman, New York, 1979.

Gerald Gazdar. Applicability of indexed grammars to natural languages. In Uwe Reyle and Christian Rohrer, editors, *Natural Language Parsing and Linguistic Theories*, pages 69–94. Dordrecht: Reidel, 1988.

Arthur Gill. State-identification experiments in finite automata. *Information and Control*, 4:132–154, 1961.

Seymour Ginsburg and Edwin H. Spanier. Semigroups, Presburger formulas, and languages. *Pacific Journal of Mathematics*, 16:285–296, 1966.

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

E. Mark Gold. Complexity of automaton identification from given data. *Information and Control*, 37(3):302–320, June 1978.

R. C. Gonzalez, J. J. Edwards, and M. G. Thomason. An algorithm for the inference of tree grammars. *Internat. J. Comput. Info. Sci.*, 5(2):145–164, 1976.

Helen Goodluck. *Language Acquisition: A Linguistic Introduction*. Blackwell Publishers, Oxford, UK, 1991.

Ronald Lewis Graham, Donald Ervin Knuth, and Oren Potashnik. *Concrete Mathematics*. Addison-Wesley, second edition, 1994.

S. A. Greibach and E. P. Freidman. Superdeterministic PDAs: A subcase with a decidable inclusion problem. *Journal of the ACM*, 27(4):675–700, October 1980.

Eran Halperin. Improved approximation algorithms for the vertex cover problem in graphs and hypergraphs. In *Proc. 11th Ann. ACM-SIAM Symp. on Discrete Algorithms*. ACM-SIAM, 2000.

Henk Harkema. A characterization of minimalist languages. In de Groote et al. (2001), pages 193–211.

Michael A. Harrison. *Introduction to Formal Language Theory*. Reading, Mass.: Addison Wesley, 1978.

Johan Håstad. Some optimal inapproximability results. In *Proc. 29th Ann. ACM Symp. on Theory of Comp.*, pages 1–10. ACM, 1997.

John R. Hayes, editor. *Cognition and the Development of Language*. New York: Wiley, 1970.

David G. Hays. Grouping and dependency theories. In *National symp. on machine translation*, 1961.

Herman Hendriks. *Studied Flexibility; Categories and Types in Syntex and Semantics*. PhD thesis, ILLC, Amsterdam, 1993.

Mark Hepple. Labelled deduction and discontinuous constituency. In M. Abrusci, C. Casadio, and M. Moortgat, editors, *Linear Logic and Lambek Calculus*, pages 123–150. ILLC, Amsterdam, 1994.

Colin De La Higuera and Jean-Christophe Janodet. Inference of ω-languages from prefixes. In *Algorithmic Learning Theory*, pages 364–378, 2001.

N. Hornstein and D. Lightfoot, editors. *Explanation in Linguistics: The Logical Problem of Language Acquisition*. London: Longman, 1981.

William A. Howard. The formulæ-as-types notion of construction. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 479–490. Academic ress, London, 1980. Reprint of a manuscript first published in 1969.

Joxan Jaffar, Jean-Louis Lassez, and Michael J. Maher. A theory of complete logic programs with equality. *Journal of Logic Programming*, 1, 1984.

Gerhard Jäger. The generative capacity of multi-modal categorial grammars. Technical report, Technical Report IRCS-98-26, IRCS, University of Pennsylvania, 1998, 1998.

Gerhard Jäger. Anaphora and type logical grammar. Habilitationsschrift, Humboldt University Berlin, published as UIL-OTS Working Papers 01004-CL/TL, Utrecht Institute of Linguistics (OTS), University of Utrecht, 2001.

Gerhard Jäger. Residuation, structural rules and context freeness. *Journal of Logic, Language and Information*, To appear.

Sanjay Jain and Efim Kinber. On intrinsic complexity of learning geometrical concepts from texts. Technical Report TRB6/99, The National University of Singapore, School of Computing, June 1999.

Sanjay Jain, Daniel N. Osherson, James Royer, and Arun Sharma. *Systems that Learn: An Introduction to Learning Theory*. The MIT Press, Cambridge, MA., second edition, 1999.

Sanjay Jain and Arun Sharma. Computational limits on team identification of languages. *Information and Computation*, 130(1):19–60, 10 October 1996a.

Sanjay Jain and Arun Sharma. The intrinsic complexity of language identification. *Journal of Computer and System Sciences*, 52(3):393–402, June 1996b.

Sanjay Jain, Arun Sharma, and John Case. Convergence to nearly minimal size grammars by vacillating learning machines. In R. Rivest, D. Haussler, and M. Warmuth, editors, *Proceedings of the 2nd Annual Workshop on Computational Learning Theory*, pages 189–199. Morgan Kaufmann, San Mateo, CA, 1989.

Theo M. V. Janssen. Compositionality. In van Benthem and ter Meulen (1997), chapter 5, pages 417–473.

Mark Johnson. A resouce sensitive interpretation of Lexical Functional Grammar. *Journal of Logic, Language and Information*, 8(1):45–81, 1999.

Aravind K. Joshi. An introduction to tree adjoining grammars. In A. Manaster-Ramer, editor, *Mathematics of Language*. John Benjamins, Amsterdam, 1987.

Aravind K. Joshi. Relationship between strong and weak generative power of formal systems. In Frank (2002), pages 159–162.

Aravind K. Joshi and Seth Kulick. Partial proof trees as building blocks for a categorial grammar. *Linguistics and Philosophy*, 20:637–667, 1997.

Aravind K. Joshi, L. S. Levy, and M. Takahashi. Tree adjunct grammars. *Journal of Computer and System Sciences*, 1(10), 1975.

Aravind K. Joshi and Yves Schabes. Tree-adjoining grammars. In Grzegorz Rosenberg and Arto Salomaa, editors, *Handbook of Formal Languages*, volume 3, pages 69–123. Springer-Verlag, New York, 1996.

K. Kamata. Inference methods for tree automata from sample set of trees. In *IEEE International Conference on Systems, Man and Cybernetics*, pages 490–493, 1984.

Makoto Kanazawa. *Learnable Classes of Categorial Grammars*. PhD thesis, Stanford University, 1994a.

Makoto Kanazawa. A note on language classes with finite elasticity. Technical Report CS-R9471, CWI, Amsterdam, 1994b.

Makoto Kanazawa. *Learnable Classes of Categorial Grammars*. CSLI Publications, Stanford University, distributed by Cambridge University Press., 1998.

Maciej Kandulski. The equivalence of nonassociative Lambek categorial grammars and context-free grammars. *Zeischrift für Mathematische Logik und Grundlagne der Mathematik*, 34:41–52, 1988.

Maciej Kandulski. On generalized Ajdukiewicz and Lambek calculi and grammars. *Fundamenta Informaticæ*, 30:169–181, 1997.

Maciej Kandulski. On generalized Ajdukiewicz and Lambek calculi. manuscript, Poznań University, 2002.

Shyam Kapur. *Computational Learning of Languages*. Available as technical report 91-1234, Department of Computer Science, Cornell University, 1991.

Shyam Kapur, Barbara Lust, Wayne E. Harbert, and Gita Martohardjono. Universal grammar and learnability theory: The case of binding domains and the "subset principle". In E. Reuland and W. Abraham, editors, *Knowledge and Language (Vol. I): From Orwell's Problem to Plato's Problem*, pages 185–216. Kluwer, Dordrecht, 1993.

Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Complexity of Computer Computations*. Plenum Press, 1972.

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

Edward L. Keenan. The semantics of determiners. In S. Lappin, editor, *The Handbook of Contemporary Semantic Theory*, pages 41–63. Oxford: Blackwell, 1996.

Kevin T. Kelly and Clark Glymour. Thoroughly modern Meno. In John Earman, editor, *Inference, Explanation, and other Frustrations*, pages 3–23. University of California Press, 1992.

Kevin T. Kelly, Cory Juhl, and Clark Glymour. Reliability, realism, and relativism. In P. Clark, editor, *Reading Putnam*, pages 98–161. London: Blackwell, 1994.

Efim Kinber and Frank Stephan. Language learning from texts: Mindchanges, limited memory, and monotonicity (extended abstract). In *Proceedings of the Eighth Annual Conference on Computational Learning Theory*, pages 182–189, Santa Cruz, California, 5–8 July 1995a. ACM Press.

Efim Kinber and Frank Stephan. Language learning from texts: Mindchanges, limited memory, and monotonicity. *Information and Computation*, 123(2): 224–241, December 1995b.

Timo Knuutila and Magnus Steinby. The inference of tree languages from finite samples: an algebraic approach. *Theoretical Computer Science*, 129:337–367, 1994.

Miroslawa Kołowska–Gawiejnowicz. Powerset residuated algebras and generalized Lambek calculus. *Mathematical Logic Quarterly*, 43:60–72, 1997.

Natasha Kurtonina. *Frames and Labels.A Modal Analysis of Categorial Inference.* PhD thesis, OTS Utrecht, ILLC Amsterdam, 1995.

Natasha Kurtonina and Michael Moortgat. Structural control. In Patrick Blackburn and Maarten de Rijke, editors, *Specifying syntactic structures*, Studies in Logic, Language and Information. CSLI Publications, Stanford, 1997.

Joachim Lambek. The mathematics of sentence structure. *Amer. Math. Monthly*, 65:154–170, 1958. Reprinted as Lambek (1988b).

Joachim Lambek. On the calculus of syntactic types. In R. Jakobson, editor, *Structure of Language and Its Mathematical Aspects.* Providence, RI, 1961.

Joachim Lambek. Categorial and categorical grammars. In Oehrle et al. (1988), pages 297–317.

Joachim Lambek. The mathematics of sentence structure. In Wojciech Buszkowski, Witold Marciszewski, and Johan van Benthem, editors, *Categorial Grammar*, volume 25 of *Linguistic and Literary Studies in Eastern Europe*, pages 153–172. 1988b.

Joachim Lambek. From categorial grammar to bilinear logic. In Kosta Došen and Peter Schröder-Heister, editors, *Substructural Logics*, pages 207–237. Clarendon Press, Oxford, 1993.

Joachim Lambek. Type grammars revisited. In Alain Lecomte, François Lamarche, and Guy Perrier, editors, *LACL99*, volume 1582 of *Lecture Notes in Computer Science*, pages 1–27. Springer, 1999.

Kevin J. Lang. Random DFA's can be approximately learned from sparse uniform examples. In *5th ACM Workshop on Computational Learning Theory*, pages 45–52, 1992.

Kevin J. Lang, B. A. Pearlmutter, and R. A. Price. Results of the Abbadingo One DFA learning competition and a new evidence-driven state merging algorithm. In Vasant Honavar and Giora Slutzki, editors, *ICGI*, volume 1433 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 1998.

Steffen Lange and Rolf Wiehagen. Polynomial time inference of arbitrary pattern languages. *New Generation Computing*, 8:361–370, 1991.

Howard Lasnik. *Essays on restrictiveness and learnability: Studies in natural language & linguistic theory.* Studies in natural language and linguistic theory; vol. 20. Kluwer, Dordrecht, 1990.

Alain Lecomte. Categorial minimalism. In de Groote et al. (2001), pages 143–158.

Alain Lecomte and Christian Retoré. Pomset logic as an alternative categorial grammar. In Glyn Morrill and Richard T. Oehrle, editors, *Formal Grammar*, pages 181–196, Barcelona, August 1995. FoLLI.

Alain Lecomte and Christian Retoré. Extending Lambek grammars: a logical account of minimalist grammars. In *Proceedings of the 39th meeting of the Association for Computational Linguistics, ACL 2001, Toulouse*, pages 354–361, july 2001.

Barry Levine. Derivatives of tree sets with applications to grammatical inference. *IEEE Trans. Pattern Anal. Mach. Intell.*, PAMI-3:285–293, 1981.

Ming Li and Akira Maruoka, editors. *Algorithmic Learning Theory, 8th International Workshop, ALT '97, Sendai, Japan, October 1997, Proceedings*, volume 1316 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, Berlin, October 1997.

Patrick Lincoln, A. Scedrov, and N. Shankar. Decision problems for second-order linear logic. In *Proceedings of the Tenth Annual IEEE Symposiumm on Logic in Computer Science*, 1995.

Jacek Marciniec. Learning categorial grammar by unification with negative constraints. *Journal of Applied Non-Classical Logics*, 4(2), 1994.

Jacek Marciniec. Connected sets of types and categorial consequence. In Christian Retoré, editor, *Logical Aspects of Computational Linguistics*, volume 1328 of *Lecture Notes in Computer Science*, pages 292–309. Springer, 1996.

Jacek Marciniec. Infinite set unification with application to categorial grammar. *Studia Logica*, 58(3), 1997.

Satoshi Matsumoto, Yukiko Hayashi, and Takayoshi Shoudai. Polynomial time inductive inference of regular term tree languages from positive data. In Li and Maruoka (1997), pages 212–227.

Irene Mazurkewich and Lydia White. The acquisition of dative-alternation: Unlearning overgeneralizations. *Cognition*, 16(3):261–283, 1984.

Jens Michaelis. Derivational minimalism is mildy context-sensitive. In *Logical Aspects of Computational Linguistics (LACL '98)*, pages 179–198, Grenoble, December 1998.

Jens Michaelis. Transforming linear context-free rewriting systems into Minimalist Grammars. In de Groote et al. (2001), pages 228–244.

Jens Michaelis and Christian Wartena. How linguistic constraints on movement conspire to yield languages analyzable with a restricted form of LIGs. In G. J. M. Kruijff, G. V. Morrill, and R. T. Oehrle, editors, *Formal Grammar 1997. Linguistic Aspects of Logical and Computational Perspectives on Language. Proceedings of the Conference*, pages 158–168, Aix-en-Provence, August 9–10 1997.

Jens Michaelis and Christian Wartena. Unidirectional inheritance of indices. A weakly context free facet of LIGs. In G. Bouma, G. J. M. Kruijff, and R. T. Oehrle, editors, *Proceedings of the FHCG'98, Joint Conference on Formal Grammar, Head-Driven Phrase Structure Grammar and Categorial Grammar*, pages 258–267, Saarbrücken, August 14–16 1998.

Philip H. Miller. *Strong Generative Capacity: The Semantics of Linguistic Formalism*. CSLI Publications, 1999.

Burkhard Monien and Ewald Speckenmeyer. Ramsey numbers and an approximation algorithm for the vertex cover problem. *Acta Informatica*, 22: 115–123, 1985.

Richard Montague. *Formal Philosophy*. New Haven: Yale University Press, 1974. R. Thomason, editor.

Edward F. Moore. Gedanken-experiments on sequential machines. In C. E. Shannon and J. McCarthy, editors, *Automata Studies*, pages 129–153. Princeton, NJ: Princeton University Press, 1956.

Michael Moortgat. Multimodal linguistic inference. *Bulletin of IGPL*, 3:371–401, 1995.

Michael Moortgat. Multimodal linguistic inference. *Journal of Language, Logic and Information*, 5(3-4):349–385, 1996. First published as Moortgat (1995).

Michael Moortgat. Categorial type logics. In van Benthem and ter Meulen (1997), pages 93–177. Chapter 2.

Michael Moortgat. Structural equations in language learning. In de Groote et al. (2001), pages 1–16.

Michael Moortgat and Glyn Morrill. Heads and phrases. Type calculus for dependency and constituent structure, 1991. Manuscript.

Michael Moortgat and Richard Oehrle. Logical parameters and linguistic variation. Lecture notes on categorial grammar. In *Fifth European Summer School in Logic, Language and Information*, Lisbon, 1993.

Michael Moortgat and Richard Oehrle. Adjacency, dependency and order. In P. Dekker and M. Stokhof, editors, *Proceedings Ninth Amsterdam Colloquium*, pages 447–466, Amsterdam, 1994.

Richard Moot. *Proof Nets for Linguistic Analysis*. PhD thesis, Utrecht Institute of Linguistics OTS, Utrecht University, 2002.

Glyn Morrill. *Type Logical Grammar*. Kluwer Academic Publishers, Dordrecht, 1994.

Tatsuya Motoki, Takeshi Shinohara, and Keith Wright. The correct definition of finite elasticity: Corrigendum to identification of unions. In *The Fourth Workshop on Computational Learning Theory*. San Mateo, Calif.: Morgan Kaufmann, 1991.

Jacques Nicolas. Grammatical inference as unification. Technical Report RR-3632, INRIA, 1999. `http://www.inria.fr/RRRT/publicati0nseng.html`.

R. T. Oehrle, E. Bach, and D. Wheeler, editors. *Categorial Grammars and Natural Language Structures*. Reidel, Dordrecht, 1988.

José Oncina and Colin de la Higuera. On sufficient conditions to identify in the limit classes of grammars from polynomial time and data. In Adriaans et al. (2002).

José Oncina and Pedro Garcia. Inferring regular languages in polynomial update time. In Perez, Sanfeliu, and Vidal, editors, *Pattern Recognition and Image Analysis*, pages 49–61. World Scientific, 1992.

Daniel N. Osherson, Dick de Jongh, Eric Martin, and Scott Weinstein. Formal learning theory. In van Benthem and ter Meulen (1997).

Daniel N. Osherson, Michael Stob, and Scott Weinstein. Learning theory and natural language. *Cognition*, 17:1–28, 1984.

Daniel N. Osherson, Michael Stob, and Scott Weinstein. *Systems that Learn: An Introduction to Learning Theory for Cognitive and Computer Scientists*. MIT Press, Cambridge, MA., 1986.

Daniel N. Osherson, Michael Stob, and Scott Weinstein. Synthesizing inductive expertise. *Information and Computation*, pages 138–161, 1988.

Rajes Parekh and Vasant Honavar. On the relationship between models for learning in helpful environments. In Arlindo L. Oliveira, editor, *Grammatical Inference: Algorithms and Applications*, volume 1891 of *Lecture Notes in Artificial Intelligence*, pages 207–220. Springer-Verlag, 2000.

Mati Pentus. The conjoinability relation in Lambek calculus and linear logic. ILLC Prepublication Series ML–93–03, Institute for Logic, Language and Computation, University of Amsterdam, 1993a.

Mati Pentus. Lambek grammars are context free. In *Proceedings of the 8th Annual IEEE Symposium on Logic in Computer Science*, pages 429–433, Los Alamitos, California, 1993b. IEEE Computer Society Press.

Mati Pentus. Product-free Lambek calculus and context-free grammars. *Journal of Symbolic Logic*, 62(2):648–660, 1997.

Mati Pentus. Lambek calculus is NP-complete. Technical Report TR-2003005, City University of New York Graduate Center, May 12 2003.

G. Perrier. A PSPACE-complete fragment of second order linear logic. *Theoretical Computer Science*, 222(1–2):267–289, 1999.

Paul Stanley Peters. The projection problem: how is a grammar to be selected? In S. Peters, editor, *Goals of Linguistic Theory*. NJ: Prentice-Hall, Englewood Cliffs, 1972.

Steven Pinker. Formal models of language learning. *Cognition*, 7:217–283, 1979.

Leonard Pitt. *A characterization of probabilistic inference*. PhD thesis, Yale University, 1984.

Leonard Pitt. Inductive inference, DFAs, and computational complexity. In K. P. Jantke, editor, *Proceedings of International Workshop on Analogical and Inductive Inference*, volume 397 of *Lecture Notes in Computer Science*, pages 18–44, 1989.

Leonard Pitt and Carl H. Smith. Probability and plurality for aggregations of learning machines. *Information and Computation*, 77:77–92, 1988.

Gordon D. Plotkin. Building in equational theories. *Machine Intelligence*, 7, 1972.

Carl Pollard and Ivan A. Sag. *Information-Based Syntax and Semantics, Volume 1: Fundamentals*. CSLI Lecture Notes. London: Springer Verlag, 1987.

Luboš Popelínský and Miloslav Nepil, editors. *Proceedings of the Third Learning Language in Logic (LLL) Workshop*, Strasbourg, France, 8–9 September 2001. Faculty of Informatics, Masaryk University, Brno, Czech Republic. Technical report FIMU-RS-2001-08, 66 Pages, available from `http://www.fi.muni.cz/ilpnet2/LLL2001/proceedings.ps`.

Geoffrey K. Pullum and Barbara C. Scholz. Empirical assessment of stimulus poverty arguments. *The Linguistic Review*, 19(1–2):9–50, 2002.

Owen Rambow and Aravind K. Joshi. A formal look at Dependency Grammars and Phrase-Structure grammars, with special consideration of word-order phenomena. In Leo Wanner, editor, *Current Issues in Meaning-Text Theory*. Pinter, London, 1997.

Christian Retoré and Edward Stabler. Resource logics and minimalist grammars. In Christian Retoré and Edward Stabler, editors, *Resource Logics and Minimalist Grammars. Workshop held at the 11th European Summer School in Logic, Language and Information (ESSLLI)*, Utrecht University, 1999.

Yasubumi Sakakibara. Learning context-free grammars from structural data in polynomial time. *Theoretical Computer Science*, 76(2–3):223–242, 21 November 1990.

Yasubumi Sakakibara. Efficient learning of context-free grammars from positive structural examples. *Information and Computation*, 97(1):23–60, March 1992.

Gillian Sankoff and Penelope Brown. The origins of syntax in discourse: A case study of tok pisin relatives. *Language*, 52:631–666, 1976.

Ahmed Saoudi and Takashi Yokomori. Learning local and recognizable ω-languages and monadic logic programs. In John Shawe-Taylor and Martin Anthony, editors, *Proceedings of the First European Conference on Computational Learning Theory, EuroCOLT'93*, London, UK, December 20–22 1994. Oxford University Press.

Irene Schena. Pomset logic and variants in natural languages. *Lecture Notes in Computer Science*, 1328:386–405, 1997.

William Schuler, David Chiang, and Mark Dras. Multi-component TAG and notions of formal power. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics (ACL'00)*, Hong Kong, China, 2000.

Yoav Seginer. Fast learning from strings of 2-letter rigid grammars. In Adriaans et al. (2002), pages 213–224.

Yoav Seginer. Rigid categorial grammars over two letters, September 18 2002b. 82 pages, unpublished manuscript.

Helmut Seidl. Deciding equivalence of finite tree automata. In *Theor. Aspects Comput. Sci. (Proc. STACS88)*, volume 349 of *Lecture Notes in Computer Science*, pages 480–492, Berlin, 1989. Springer-Verlag.

Helmut Seidl. Deciding equivalence of finite tree automata. *SIAM Journal on Computing*, 19(3):424–437, June 1990.

Takeshi Shinohara. *Studies on Inductive Inference from Positive Data*. PhD thesis, Kyushu University, 1986.

Takeshi Shinohara. Inductive inference from positive data is powerful. In *The 1990 Workshop on Computational Learning Theory*, pages 97–101, San Mateo, Calif., 1990a. Morgan-Kaufmann.

Takeshi Shinohara. Inductive inference of monotonic formal systems from positive data. In S. Arikawa, S. Goto, S. Ohsuga, and T. Yokomori, editors, *Algorithmic Learning Theory*, pages 339–351. Springer, New York and Berlin, 1990b.

Takeshi Shinohara. Rich classes inferable from positive data: Length-bounded elementary formal systems. *Information and Computation*, 108(2):175–186, 1 February 1994.

Dan I. Slobin. Language change in childhood and in history. In J. Macnamara, editor, *Language Learning and Thought*. New York: Academic Press, 1977.

Raymond M. Smullyan. *Theory of Formal Systems*. Princeton Univ. Press, 1961.

Catherine E. Snow. Mothers' speech research: from input to interaction. In C. E. Snow and C. A. Ferguson, editors, *Talking to Children: Language Input and Acquisition*. Cambridge: Cambridge University Press, 1977.

Ray J. Solomonoff. A formal theory of inductive inference. *Information and Control*, pages 1–22 and 224–254, 1964.

Morten Heine B. Sørensen and Paweł Urzyczyn. Lectures on the Curry-Howard isomorphism. Available as DIKU Rapport 98/14, 1998.

Edward P. Stabler. Derivational minimalism. In Christian Retoré, editor, *Proceedings of the First International Conference on Logical Aspects of Computational Linguistics*, Lecture Notes in Artificial Intelligence, pages 68–95. Springer-Verlag, 1997.

Edward P. Stabler. Acquiring languages with movement. *Syntax*, 1:72–97, 1998.

Edward P. Stabler. Recognizing head movement. In de Groote et al. (2001), pages 245–260.

Edward P. Stabler. Identifying minimalist languages from dependency structures. Manuscript, April 2002.

Edward P. Stabler and Edward Keenan. Structural similarity. *Theoretical Computer Science*, 293:345–363, 2003. Revised version.

Richard P. Stanley. Excercises on Catalan and Related Numbers, June 23rd 1998. Excerpted from Stanley (1999), available from `http://www-math.mit.edu/~rstan/ec/catalan.ps.gz`.

Richard P. Stanley. *Enumerative Combinatorics*, volume 2. Cambridge University Press, 1999.

Mark Steedman. Combinatory grammars and parasitic gaps. *Natural Language and Linguistic Theory*, 5:403–439, 1987.

Mark Steedman. *The Syntactic Process*. MIT Press/Bradford Books, 2000.

Werner Stein. Consistent polynomial identification in the limit. In *Algorithmic Learning Theory (ALT)*, volume 1501 of *Lecture Notes in Computer Science*, pages 424–438, Berlin, 1998. Springer-Verlag.

Frank Stephan and Yuri Ventsov. Learning algebraic structures from text. *Theoretical Computer Science*, 268(2):221–273, 2001.

Mark E. Stickel. Automated deduction by theory resolution. *Journal of Automated Reasoning*, 1:333–356, 1985.

Anna Szabolcsi. Bound variables in syntax. In *Proceedings of the Sixth Amsterdam Colloquium*, pages 331–351, Institute for Language, Logic and Information, 1987. Universiteit van Amsterdam.

Hans-Jörg Tiede. Lambek calculus proofs and tree automata. In Michael Moortgat, editor, *Logical Aspects of Computational Linguistics Third International Conference, LACL'98, Selected Papers*, volume 2014 of *Lecture Notes in Artificial Intelligence*, pages 251–265, Grenoble, France, December 1998. Springer-Verlag.

Hans-Jörg Tiede. *Deductive Systems and Grammars: Proofs as Grammatical Structures*. PhD thesis, Indiana University, 1999a.

Hans-Jörg Tiede. Identifiability in the limit of context-free generalized quantifiers. *Journal of Language and Computation*, 1(1):93–102, 1999b.

Leslie G. Valiant. A theory of the learnable. *Communications of the ACM*, 27 (11):1134–1142, 1984.

Johan van Benthem. *The semantics of variety in categorial grammar*. Simon Fraser University, Burnaby, 1983.

Johan van Benthem. Questions about quantifiers. *Journal of Symbolic Logic*, 49(2):443–466, 1984.

Johan van Benthem. *Essays in Logical Semantics*. Reidel, Dordrecht, 1986a.

Johan van Benthem. Semantic automata. In Jeroen Groenendijk, Dick de Jongh, and Martin Stokhof, editors, *Studies in Discourse, Representation Theory and the Theory of Generalized Quantifiers*, chapter 1, pages 1–26. Foris Publications - Dordrecht, 1986b.

Johan van Benthem. Categorial grammar and Lambda calculus. In D. Skordev, editor, *Mathematical Logic and Its Applications*. Plenum Press, New York, 1987.

Johan van Benthem. *Language in Action: Categories, Lambdas and Dynamic Logic*, volume 130 of *Studies in Logic*. North-Holland, Amsterdam, 1991.

Johan van Benthem and Alice ter Meulen, editors. *Handbook of Logic and Language*. Elsevier Science Publishing, 1997.

Krishnamurti Vijay-Shanker. *A Study of Tree Adjoining Grammars*. PhD thesis, University of Pennsylvania, 1987.

Krishnamurti Vijay-Shanker and David J. Weir. Polynomial time parsing of combinatory categorial grammars. In *Proc. of the 28 th ACL*, pages 1–8, Pittsburgh, 1990.

Krishnamurti Vijay-Shanker and David J. Weir. The equivalence of four extensions of context-free grammars. *Mathematical Systems Theory*, 27:511–546, 1994.

Eric Wanner and Lila R. Gleitman, editors. *Language Acquisition: The State of the Art*. Cambridge: Cambridge University Press, 1982.

David J. Weir and Aravind K. Joshi. Combinatory categorial grammars: generative power and relationship to linear context-free rewriting systems. In *Proceedings of the 26th Meeting of the Association for Computational Linguistics*, 1988.

Kenneth Wexler and Peter W. Culicover. *Formal Principles of Language Acquisition*. MIT Press, Cambridge, MA., 1980.

Kenneth Wexler and H. Hamburger. On the insufficiency of surface data for the learning of transformational languages. In K. J. J. Hintikka, J. M. E. Moravcsik, and P. Suppes, editors, *Approaches to Natural Language*. Dordrecht, Holland: D. Reidel, 1973.

Rolf Wiehagen. Identification of formal languages. In *Mathematical Foundations of Computer Science, Proceedings, 6th Symposium, Tatranska Lomnica*, volume 53 of *Lecture Notes in Computer Science*, pages 571–579. Springer, 1977.

Rolf Wiehagen. Characterization problems in the theory of inductive inference. In *Proceedings of the 5th International Colloquium on Automata, Languages and Programming*, volume 62 of *Lecture Notes in Computer Science*, pages 494–508. Springer, 1978.

Rolf Wiehagen and Thomas Zeugmann. Ignoring data may be the only way to learn efficiently. *Journal of Experimental and Theoretical Artificial Intelligence*, 6:131–144, 1994.

Rolf Wiehagen and Thomas Zeugmann. Learning and consistency. In K. P. Jantke and S. Lange, editors, *Algorithmic Learning for Knowledge-Based Systems*, volume 961 of *Lecture Notes in Artificial Intelligence*, pages 1–24. Springer-Verlag, 1995.

Keith Wright. Identification of unions of languages drawn from an identifiable class. In *The 1989 Workshop on Computational Learning Theory*, pages 328–333. San Mateo, Calif.: Morgan Kaufmann, 1989.

Charles D. Yang. *Knowledge and Learning in Natural Language*. Csl 26, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1999. unpublished.

Takashi Yokomori. On polynomial-time learning in the limit of strictly deterministic automata. *Machine Learning*, 19(2):153–182, 1995.

Wojciech Zielonka. Axiomatizability of Adjukiewicz-Lambek calculus by means of cancellation scheme. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 27:215–224, 1981.

Wojciech Zielonka. A simple and general method of solving the finite axiomatizability problems for Lambek's syntactic calculi. *Studia Logica*, 48:35–39, 1989.

Sandra Zilles. On the comparison of inductive inference criteria for uniform learning of finite classes. In *Algorithmic Learning Theory, 12th International Conference, ALT 2001, Washington, DC, USA, November 25–28, 2001, Proceedings*, volume 2225 of *Lecture Notes in Artificial Intelligence*, pages 251–266. Springer, 2001a.

Sandra Zilles. On the synthesis of strategies identifying recursive functions. In *14th Annual Conference on Computational Learning Theory, COLT 2001 and 5th European Conference on Computational Learning Theory, EuroCOLT 2001, Amsterdam, The Netherlands, July 2001, Proceedings*, volume 2111 of *Lecture Notes in Artificial Intelligence*, pages 160–176. Springer, 2001b.

Sandra Zilles. Merging uniform inductive learners. In *15th Annual Conference on Computational Learning Theory, COLT 2002, Sydney, Australia, July 2002, Proceedings*, volume 2375 of *Lecture Notes in Artificial Intelligence*, pages 201–215. Springer, 2002.

Samenvatting in het Nederlands

In deze dissertatie worden verschillende taalkundige toepassingen van Formele Leertheorie onderzocht. Het verschijnen van Gold (1967) wordt algemeen beschouwd als de geboorte van dit vakgebied, hoewel er meerdere voorlopers kunnen worden aangewezen, de meesten met een sterk (wetenschaps) filosofische inslag (zie referenties in onder andere Kelly et al. (1994); Kelly and Glymour (1992)).

In dit paper werd een aantal (deels) linguïstisch gemotiveerde formele modellen van leerbaarheid –collectief aangeduid als 'identificatie in de limiet'– toegepast op de taalklassen uit de Chomsky-hiërarchie. De theorie rond deze modellen werd in de daarop volgende jaren verfijnd en uitgebeid (oa Blum and Blum (1973, 1975); Gold (1978); Wiehagen (1977, 1978)). Dit leidde tot voortschrijdende theoretische inzichten, echter nauwelijks tot echt praktisch toepasbare resultaten. Pas sinds het begin van de jaren '80, nadat Dana Angluin een volledige en bruikbare karakterisering gaf van (effectief) leerbare verzamelingen van talen (Angluin (1980a,b)), was het vakgebied volwassen genoeg om toegepast te worden in de context van de theorie van formele talen (en dus ook binnen de wiskundige en computationele taalkunde), dit (deel)vakgebied wordt soms aangeduid als Inductive Inference.

Dit werk van Angluin, en van onder andere Wright (Motoki et al. (1991)) toonde het belang aan van een begrip als eindige elasticiteit. Ook het proefschrift Kapur (1991) biedt bruikbaar gereedschap, met name de toepassing van de uit de topologie afkomstige concepten limiet- en accumulatie punt in de formele leertheorie blijkt erg nuttig.

In het werk van Shinohara (oa Shinohara (1986); Arikawa et al. (1989); Shinohara (1990b)) is deze lijn van onderzoek nog verder uitgewerkt, door de eindige elasticiteit aan te tonen van allerlei klassen van 'optimale' of 'begrensde' Elementary Formal Systems (EFS, Smullyan (1961)). Dit generieke logische systeem biedt de mogelijkheid andere formalismen te emuleren en wordt ook wel beschouwd als een (lineair) logische programmeertaal. Een voor de linguïstiek interessant gevolg van zijn resultaten is dat alle context-gevoelige grammatica's met een begrensd aantal regels kunnen worden geëmuleerd binnen een leerbare

klasse van EFS en zelf dus ook een leerbare klasse vormen.

In Kanazawa (1998) worden een aantal klassen van (klassieke) Categoriale Grammatica's onderzocht op leerbaarheid. Deze klassen zijn gebaseerd op criteria voor 'optimale' of 'begrensde' grammatica's en men zou op grond van Shinohara's resultaten dus verwachten dat deze klassen leerbaar zijn, dit bleek inderdaad het geval.

Voor deze leerbare klassen zijn leeralgoritmen ontwikkeld, deze algoritmen zijn naïef. Dit wil zeggen dat het 'directe' implementaties van een theorie zijn, en niet per se optimaal wat efficiëntie betreft. Een van de doelen van deze dissertatie is het onderzoeken van de complexiteit van de leerproblemen die door deze algoritmen worden opgelost. Er wordt onder andere aangetoond dat, voor elk van de onderzochte klassen, het vormen van een hypothese die consistent is met alle gegeven data en binnen de klasse valt een NP-moeilijk probleem is. Dit onderbouwt het advies van Rolf Wiehagen dat men zich bij het ontwerpen van leeralgoritmen niet blind moet staren op consistentie, omdat men op die manier efficiënte maar niet consistente algoritmen over het hoofd zou kunnen zien.

Een andere benadering van Categoriale Grammatica gebruikt een compleet *logisch* systeem om grammaticale theorieën in uit te drukken, de zogenaamde Lambek Calculus. In de praktijk is gebleken dat de analyse van de formele eigenschappen van deze en aanverwante systemen zeer moeilijk is. Zelfs voor de hand liggende en belangrijke vragen omtrent expressieve kracht en complexiteit zijn pas zeer recent, en dan nog maar deels, beantwoord (Pentus (1993b, 1997); de Groote (1999); Pentus (2003)). Ook leerbaarheidsvraagstukken bleken in deze context moeilijk te beantwoorden. Het eerste resultaat op dit gebied is zeer recent; de klasse van rigide (dat wil zeggen: elk woord heeft maar één syntactische categorie) associatieve Lambek grammatica's bleek niet leerbaar aan de hand van strings (Foret and Le Nir (2002a)). Resultaten voor andere Lambek varianten volgden binnen korte tijd (Foret and Le Nir (2002b); Bechet and Foret (submitted); Costa Florêncio (2003)). Deze (negatieve) resultaten zijn een direkt gevolg van de geldigheid in deze calculi van een axioma zoals Lifting. Aangezien deze essentieel lijkt voor elke sterk compositionele benadering van (natuurlijke) taal is het te verwachten dat de 'Shinohara benadering' onbruikbaar is voor al deze soort systemen.

De zogenaamde Tree Adjoining Grammars (TAGs) spelen een belangrijke rol in de (computationele) linguïstiek; ze maken efficiënt parseren mogelijk, lijken ongeveer de juiste (zwakke) expressieve kracht te hebben voor het uitdrukken van natuurlijke taal, en zijn min of meer uniek in de manier waarop ze bomen (derivaties) kunnen manipuleren. Helaas blijken TAGs wat leerbaarheid betreft minder aantrekkelijk: zelfs hele beperkte (rigide) klassen blijken niet leerbaar, en dus blijkt de 'Shinohara benadering' weer niet zonder meer toepasbaar. De bewijzen van deze negatieve resultaten maken gebruik van de optionaliteit van de adjunctie-operatie en van het mechanisme dat TAGs bieden om controle uit te oefenen over deze optionaliteit.

Minimalist Grammar (MG), een formalisering van de meest recente vorm van transformationele syntax, is een in bepaalde opzichten aan TAG verwant systeem. Het is daarom misschien niet verrassend dat de 'Shinohara benadering' ook hier niet zomaar toepasbaar is, in ieder geval niet als het is toegestaan een onbeperkt aantal zogenaamde licensee features aan dezelfde syntactische categorie toe te kennen.

Er is geen enkele reden om de toepassing van formele leertheorie in de linguïstiek te beperken tot syntax. Een voorbeeld van toepassing binnen de semantiek is het leren van Gegeneraliseerde Kwantoren zoals onderzocht in (Tiede (1999b)). Deze benadering wordt besproken, en Tiede's resultaten worden hier ook enigszins verfijnd. Tevens wordt gekeken naar de leerbaarheid van subklassen van *combinatoriële* categoriale grammatica's, een expressievere variant van klassieke CG.

In Kanazawa (1998) wordt er op gewezen dat zijn resultaten betreffende een aantal CG klassen volgen uit Shinohara's resultaat voor context-gevoelige grammatica's. Het vermoeden werd geuit dat soortgelijke resultaten ook gelden voor andere formalismen. Een belangrijke conclusie die uit de besproken formele resultaten getrokken kan worden is dat de situatie in werkelijkheid een stuk gecompliceerder blijkt; voor allerlei aan de Lambek calculus verwante systemen bijvoorbeeld zijn de klassen van rigide en k-waardige grammatica's niet leerbaar, zelfs niet non-effectief, en zelfs niet met allerlei bijkomende beperkingen op de grootte van het gebruikte alfabet, de orde van de gebruikte lexicale typen et cetera.

Een ander beeld wat naar voren komt is het belang van *derivaties*, en dus in de context van formele talen van boom-talen, boom-automaten en context-vrije boom-grammatica's als hulpmiddelen bij het redeneren over leerbaarheids-vraagstukken. Hiervoor bestaat linguïstische motivatie – Chomsky (1965b) pleit al voor een dergelijke benadering – maar zeker ook technische; een voor leerbaarheid belangrijke probleem als 'is taal L_1 een deelverzameling van L_2' en het verwante 'welke grammatica in een gegeven verzameling is genereert een minimale taal?' zijn niet beslisbaar voor bijvoorbeeld context-vrije (string) talen maar wel voor reguliere *boom*-talen, die de *derivaties* vormen van context-vrije talen. Een ander voordeel is dat er redelijk algemene voldoende voorwaarden te geven zijn voor de leerbaarheid van boom-talen, zoals omkeerbaarheid. Dit maakt het bewijzen van leerbaarheid aanzienlijk simpeler, zoals in Hoofdstuk 7 wordt gedemonstreerd aan de hand van het werk van Besombes en Marion. Zij geven een alternatief en elegant bewijs voor Kanazawa's stelling dat rigide (klassieke) categoriale grammatica's een leerbare klasse vormen.

Curriculum Vitæ

Christophe Costa Florêncio was born in Leiden, the Netherlands, on December 20, 1971.

He studied at the Gymnasium Haganum in The Hague from to 1984 to 1991, and, starting in 1991, studied Cognitive Science and Artifical Intelligence at Utrecht University, and also Philosophy during 1992–1993. He switched from the specializations Cognition and Representation to Knowledge-Based Systems to Computational Linguistics and Logic, following an extra year of courses in total. He was teaching-assistent for several courses taught at the Faculty of Arts and the Faculty of Philosophy.

He was intern at the Rudolf Magnus Institute during 1997–1998, applying neural network techniques to visual classification of rat behaviour, resulting in the report Costa Florêncio (1998b).

He graduated, with the final specialization Free Variant, in August 1998 with the master's thesis Costa Florêncio (1998a), written under the supervision of dr. Dick de Jongh (UvA/ILLC) and prof. dr. Michael Moortgat (UU/UiL OTS).

During 1998–1999 he was employed as a Scientific-Technical programmer at Syllogic (now Perot Systems Netherlands), working on Information Retrieval amongst other things.

During 1999–2003 he was employed as a PhD-student at the Utrecht Institute of Linguistics. The present thesis is the result of his research he carried out there.

In addition to his PhD work he assisted at the summerschool ESSLLI'99 and was editor of the Colibri electronic newsletter.