The Mathematics of Language

Marcus Kracht
Department of Linguistics
UCLA
PO Box 951543
450 Hilgard Avenue
Los Angeles, CA 90095–1543
USA
`kracht@humnet.ucla.edu`

Printed Version

September 16, 2003

Was dann nachher so schön fliegt . . .
wie lange ist darauf rumgebrütet worden.

Peter Rühmkorf: *Phönix voran*

Preface

The present book developed out of lectures and seminars held over many years at the Department of Mathematics of the Freie Universität Berlin, the Department of Linguistics of the Universität Potsdam and the Department of Linguistics at UCLA. I wish to thank in particular the Department of Mathematics at the Freie Universität Berlin as well as the Freie Universität Berlin for their support and the always favourable conditions under which I was allowed to work. Additionally, I thank the DFG for providing me with a Heisenberg–Stipendium, a grant that allowed me to continue this project in between various paid positions.

I have had the privilege of support by Hans–Martin Gärtner, Ed Keenan, Hap Kolb and Uwe Mönnich. Without them I would not have had the energy to pursue this work and fill so many pages with symbols that create so much headache. They always encouraged me to go on.

Lumme Erilt, Greg Kobele and Jens Michaelis have given me invaluable help by scrupulously reading earlier versions of this manuscript. Further, I wish to thank Helmut Alt, Christian Ebert, Benjamin Fabian, Stefanie Gehrke, Timo Hanke, Wilfrid Hodges, Gerhard Jäger, Makoto Kanazawa, Franz Koniecny, Thomas Kosiol, Ying Lin, Zsuzsanna Lipták, István Németi, Terry Parsons, Alexis–Manaster Ramer, Jason Riggle, Stefan Salinger, Ed Stabler, Harald Stamm, Peter Staudacher, Wolfgang Sternefeld and Ngassa Tchao for their help.

Los Angeles and Berlin, September 2003 *Marcus Kracht*

Introduction

This book is — as the title suggests — a book about the mathematical study of language, that is, about the description of language and languages with mathematical methods. It is intended for students of mathematics, linguistics, computer science, and computational linguistics, and also for all those who need or wish to understand the formal structure of language. It is a mathematical book; it cannot and does not intend to replace a genuine introduction to linguistics. For those who are not acquainted with general linguistics we recommend [**?**], which is a bit outdated but still worth its while. For a more recent book see [**?**]. No linguistic theory is discussed here in detail. This text only provides the mathematical background that will enable the reader to fully grasp the implications of these theories and understand them more thoroughly than before. Several topics of mathematical character have been omitted: there is for example no statistics, no learning theory, and no optimality theory. All these topics probably merit a book of their own. On the linguistic side the emphasis is on syntax and formal semantics, though morphology and phonology do play a role. These omissions are mainly due to my limited knowledge. However, this book is already longer than I intended it to be. No more material could be fitted into it.

The main mathematical background is algebra and logic on the semantic side and strings on the syntactic side. In contrast to most introductions to formal semantics we do not start with logic — we start with strings and develop the logical apparatus as we go along. This is only a pedagogical decision. Otherwise, the book would start with a massive theoretical preamble after which the reader is kindly allowed to see some worked examples. Thus we have decided to introduce logical tools only when needed, not as overarching concepts.

We do not distinguish between natural and formal languages. These two types of languages are treated completely alike. I believe that it should not matter in principle whether what we have is a natural or an artificial product. Chemistry applies to naturally occurring substances as well as artificially produced ones. All I will do here is study the structure of language. Noam Chomsky has repeatedly claimed that there is a fundamental difference between natural and nonnatural languages. Up to this moment, conclusive evidence for this claim is missing. Even if this were true, this difference should

not matter for this book. To the contrary, the methods established here might serve as a tool in identifying what the difference is or might be. The present book also is not an introduction to the theory of formal languages; rather, it is an introduction to the mathematical theory of linguistics. The reader will therefore miss a few topics that are treated in depth in books on formal languages on the grounds that they are rather insignificant in linguistic theory. On the other hand, this book does treat subjects that are hardly found anywhere else in this form. The main characteristic of our approach is that we do not treat languages as sets of strings but as algebras of signs. This is much closer to the linguistic reality. We shall briefly sketch this approach, which will be introduced in detail in Chapter 3.

A **sign** σ is defined here as a triple $\langle e, c, m\rangle$, where e is the **exponent of** σ, which typically is a string, c the (**syntactic**) **category of** σ, and m its **meaning**. By this convention a string is connected via the language with a set of meanings. Given a set Σ of signs, e **means** m **in** Σ if and only if (= iff) there is a category c such that $\langle e, c, m\rangle \in \Sigma$. Seen this way, the task of language theory is not only to say which are the legitimate exponents of signs (as we find in the theory of formal languages as well as many treatises on generative linguistics which generously define language to be just syntax) but it must also say which string can have what meaning. The heart of the discussion is formed by the principle of compositionality, which in its weakest formulation says that the meaning of a string (or other exponent) is found by homomorphically mapping its analysis into the semantics. Compositionality shall be introduced in Chapter 3 and we shall discuss at length its various ramifications. We shall also deal with Montague Semantics, which arguably was the first to implement this principle. Once again, the discussion will be rather abstract, focusing on mathematical tools rather than the actual formulation of the theory. Anyhow, there are good introductions to the subject which eliminate the need to include details. One such book is [**?**] and the book by the collective of authors [**?**]. A **system of signs** is a partial algebra of signs. This means that it is a pair $\langle \Sigma, M\rangle$, where Σ is a set of signs and M a finite set, the set of so–called **modes** (**of composition**). Standardly, one assumes M to have only one nonconstant mode, a binary function $\bullet$, which allows one to form a sign $\sigma_1 \bullet \sigma_2$ from two signs σ_1 and σ_2. The modes are generally partial operations. The action of $\bullet$ is explained by defining its action on the three components of the respective signs. We give a simple example. Suppose we

have the following signs.

$$\begin{array}{lcl} \texttt{`runs'} & = & \langle \texttt{runs}, v, \rho \rangle \\ \texttt{`Paul'} & = & \langle \texttt{Paul}, n, \pi \rangle \end{array}$$

Here, v and n are the syntactic categories *(intransitive) verb* and *proper name*, respectively. π is a constant, which denotes an individual, namely Paul, and ρ is a function from individuals to the set of truth values, which typically is the set $\{0,1\}$. (Furthermore, $\rho(x) = 1$ if and only if x is running.) On the level of exponents we choose word concatenation, which is string concatenation (denoted by $^\frown$) with an intervening blank. (Perfectionists will also add the period at the end...) On the level of meanings we choose function application. Finally, let $\circ$ be a partial function which is only defined if the first argument is n and the second is v and which in this case yields the value t. Now we put

$$\langle e_1, c_1, m_1 \rangle \bullet \langle e_2, c_2, m_2 \rangle := \langle e_1 {}^\frown \Box {}^\frown e_2, c_1 \circ c_2, m_2(m_1) \rangle$$

Then `'Paul'` $\bullet$ `'runs'` is a sign, and it has the following form.

$$\texttt{`Paul'} \bullet \texttt{`runs'} := \langle \texttt{Paul runs}, t, \rho(\pi) \rangle$$

We shall say that this sentence is true if and only if $\rho(\pi) = 1$; otherwise we say that it is false. We hasten to add that `'Paul'` $\bullet$ `'Paul'` is *not* a sign. So, $\bullet$ is indeed a partial operation.

The key construct is the free algebra generated by the constant modes alone. This algebra is called the **algebra of structure terms**. The structure terms can be generated by a simple context free grammar. However, not every structure term names a sign. Since the algebras of exponents, categories and meanings are partial algebras, it is in general not possible to define a homomorphism from the algebra of structure terms into the algebra of signs. All we can get is a partial homomorphism. In addition, the exponents are not always strings and the operations between them not only concatenation. Hence the defined languages can be very complex (indeed, every recursively enumerable language Σ can be so generated).

Before one can understand all this in full detail it is necessary to start off with an introduction into classical formal language theory using semi Thue systems and grammars in the usual sense. This is what we shall do in Chapter 1. It constitutes the absolute minimum one must know about these matters. Furthermore, we have added some sections containing basics from algebra,

set theory, computability and linguistics. In Chapter 2 we study regular and context free languages in detail. We shall deal with the recognizability of these languages by means of automata, recognition and analysis problems, parsing, complexity, and ambiguity. At the end we shall discuss semilinear languages and Parikh's Theorem.

In Chapter 3 we shall begin to study languages as systems of signs. Systems of signs and grammars of signs are defined in the first section. Then we shall concentrate on the system of categories and the so–called categorial grammars. We shall introduce both the Ajdukiewicz–Bar Hillel Calculus and the Lambek–Calculus. We shall show that both can generate exactly the context free string languages. For the Lambek–Calculus, this was for a long time an open problem, which was solved in the early 1990s by Mati Pentus.

Chapter 4 deals with formal semantics. We shall develop some basic concepts of algebraic logic, and then deal with boolean semantics. Next we shall provide a completeness theorem for simple type theory and discuss various possible algebraizations. Then we turn to the possibilities and limitations of Montague Semantics. Then follows a section on partiality and presupposition.

In the fifth chapter we shall treat so–called PTIME languages. These are languages for which the parsing problem is decidable deterministically in polynomial time. The question whether or not natural languages are context free was considered settled negatively until the 1980s. However, it was shown that most of the arguments were based on errors, and it seemed that none of them was actually tenable. Unfortunately, the conclusion that natural languages are actually all context free turned out to be premature again. It now seems that natural languages, at least some of them, are not context free. However, all known languages seem to be PTIME languages. Moreover, the so–called weakly context sensitive languages also belong to this class. A characterization of this class in terms of a generating device was established by William Rounds, and in a different way by Annius Groenink, who introduced the notion of a literal movement grammar. We shall study these types of grammars in depth. In the final two sections we shall return to the question of compositionality in the light of Leibniz' Principle, and then propose a new kind of grammars, de Saussure grammars, which eliminate the duplication of typing information found in categorial grammar.

The sixth chapter is devoted to the logical description of language. This approach has been introduced in the 1980s and is currently enjoying a revival. The close connection between this approach and the so–called constraint–programming is not accidental. It was proposed to view grammars not as

generating devices but as theories of correct syntactic descriptions. This is very far away from the tradition of generative grammar advocated by Chomsky, who always insisted that language contains a generating device (though on the other hand he characterizes this as a theory of competence). However, it turns out that there is a method to convert descriptions of syntactic structures into syntactic rules. This goes back to ideas by Büchi, Wright as well as Thatcher and Doner on theories of strings and theories of trees in monadic second order logic. However, the reverse problem, extracting principles out of rules, is actually very hard, and its solvability depends on the strength of the description language. This opens the way into a logically based language hierarchy, which indirectly also reflects a complexity hierarchy. Chapter 6 ends with an overview of the major syntactic theories that have been introduced in the last 25 years.

NOTATION. Some words concerning our notational conventions. We use typewriter font for true characters in print. For example: `Maus` is the German word for 'mouse'. Its English counterpart appears in (English) texts either as `mouse` or as `Mouse`, depending on whether or not it occurs at the beginning of a sentence. Standard books on formal linguistics often ignore these points, but since strings are integral parts of signs we cannot afford this here. In between true characters in print we also use so–called *metavariables* (placeholders) such as a (which denotes a single letter) and $\vec{x}$ (which denotes a string). The notation $\mathtt{c}_i$ is also used, which is short for the true letter `c` followed by the binary code of i (written with the help of appropriately chosen characters, mostly `0` and `1`). When defining languages as sets of strings we distinguish between brackets that appear in print (these are `(` and `)`) and those which are just used to help the eye. People are used to employ abbreviatory conventions, for example `5+7+4` in place of `(5+(7+4))`. Similarly, in logic one uses $\mathtt{p}_0\wedge(\neg\mathtt{p}_1)$ or even $\mathtt{p}_0\wedge\neg\mathtt{p}_1$ in place of $(\mathtt{p}_0\wedge(\neg\mathtt{p}_1))$. We shall follow that usage when the material shape of the formula is immaterial, but in that case we avoid using the true function symbols and the true brackets '`(`' and '`)`', and use '(' and ')' instead. For $\mathtt{p}_0\wedge(\neg\mathtt{p}_1)$ is actually *not* the same as $(\mathtt{p}_0\wedge(\neg\mathtt{p}_1))$. To the reader our notation may appear overly pedantic. However, since the character of the representation is part of what we are studying, notational issues become syntactic issues, and syntactical issues simply cannot be ignored. Notice that '$\langle$' and '$\rangle$' are truly metalinguistic symbols that are used to define sequences. We also use sans serife fonts for terms in formalized and computer languages, and attach a prime to refer to its denotation (or meaning). For example, the computer code for a while–loop is written

semi–formally as $\mathsf{while}\ i < 100\ \mathsf{do}\ x := x \times (x + i)\ \mathsf{od}$. This is just a string of symbols. However, the notation $\mathsf{see}'(\mathsf{john}', \mathsf{paul}')$ denotes the proposition that John sees Paul, not the sentence expressing that.

Contents

1	**Fundamental Structures**	1
1	Algebras and Structures	1
2	Semigroups and Strings	16
3	Fundamentals of Linguistics	29
4	Trees	42
5	Rewriting Systems	52
6	Grammar and Structure	65
7	Turing machines	79
2	**Context Free Languages**	93
1	Regular Languages	93
2	Normal Forms	101
3	Recognition and Analysis	115
4	Ambiguity, Transparency and Parsing Strategies	130
5	Semilinear Languages	145
6	Parikh's Theorem	158
7	Are Natural Languages Context Free?	163
3	**Categorial Grammar and Formal Semantics**	175
1	Languages as Systems of Signs	175
2	Propositional Logic	189
3	Basics of λ–Calculus and Combinatory Logic	205
4	The Syntactic Calculus of Categories	222
5	The AB–Calculus	237
6	The Lambek–Calculus	247
7	Pentus' Theorem	255
8	Montague Semantics I	266
4	**Semantics**	279
1	The Nature of Semantical Representations	279
2	Boolean Semantics	294
3	Intensionality	306
4	Binding and Quantification	321
5	Algebraization	330

6	Montague Semantics II	341
7	Partiality and Discourse Dynamics	352
5	**PTIME Languages**	365
1	Mildly–Context Sensitive Languages	365
2	Literal Movement Grammars	379
3	Interpreted LMGs	391
4	Discontinuity	399
5	Adjunction Grammars	412
6	Index Grammars	421
7	Compositionality and Constituent Structure	431
8	de Saussure Grammars	445
6	**The Model Theory of Linguistic Structures**	459
1	Categories	459
2	Axiomatic Classes I: Strings	468
3	Categorization and Phonology	483
4	Axiomatic Classes II: Exhaustively Ordered Trees	503
5	Transformational Grammar	513
6	GPSG and HPSG	527
7	Formal Structures of GB	538

Chapter 1
Fundamental Structures

1. Algebras and Structures

In this section we shall provide definitions of basic terms and structures which we shall need throughout this book. Among them are the notions of *algebra* and *structure*. Readers for whom these are entirely new are advised to read this section only cursorily and return to it only when they hit upon something for which they need background information.

We presuppose some familiarity with mathematical thinking, in particular some knowledge of elementary set theory and proof techniques such as induction. For basic concepts in set theory see [**?**] or [**?**; **?**]; for background in logic see [**?**]. Concepts from algebra (especially universal algebra) can be found in [**?**] and [**?**], and in [**?**] and [**?**] for partial algebras; for general background on lattices and orderings see [**?**] and [**?**].

We use the symbols $\cup$ for the union, $\cap$ for the intersection of two sets. Instead of the difference symbol $M \backslash N$ we use $M - N$. $\varnothing$ denotes the empty set. $\wp(M)$ denotes the set of subsets of M, $\wp_{fin}(M)$ the set of finite subsets of M. Sometimes it is necessary to take the union of two sets that does not identify the common symbols from the different sets. In that case one uses $+$. We define $M + N := M \times \{0\} \cup N \times \{1\}$ ($\times$ is defined below). This is called the **disjoint union**. For reference, we fix the background theory of sets that we are using. This is the theory ZFC (Zermelo Fraenkel Set Theory with Choice). It is essentially a first order theory with only two two place relation symbols, $\in$ and $=$. (See Section 3.8 for a definition of first order logic.) We define $x \subseteq y$ by $(\forall z)(z \in x \to x \in y)$. Its axioms are as follows.

1. *Singleton Set Axiom.* $(\forall x)(\exists y)(\forall z)(z \in y \leftrightarrow z = x)$.
   This makes sure that for every x we have a set $\{x\}$.

2. *Powerset Axiom.* $(\forall x)(\exists y)(\forall z)(z \in y \leftrightarrow z \subseteq x)$.
   This ensures that for every x the power set $\wp(x)$ of x exists.

3. *Set Union.* $(\forall x)(\exists y)(\forall z)(z \in y \leftrightarrow (\exists u)(z \in u \wedge u \in x))$.
   u is denoted by $\bigcup_{z \in x} z$ or simply by $\bigcup x$. The axiom guarantees its existence.

4. *Extensionality.* $(\forall x)(\forall y)(x = y \leftrightarrow (\forall z)(z \in x \leftrightarrow z \in y))$.

5. *Replacement.* If f is a function with domain x then the direct image of x under f is a set. (See below for a definition of *function*.)

6. *Weak Foundation.*

$$(\forall x)(x \neq \varnothing \to (\exists y)(y \in x \wedge (\forall z)(z \in x \to z \notin y)))$$

   This says that in every set there exists an element that is minimal with respect to $\in$.

7. *Comprehension.* If x is a set and φ a first order formula with only y occurring free, then $\{y : y \in x \wedge \varphi(y)\}$ also is a set.

8. *Axiom of Infinity.* There exists an x and an injective function $f : x \to x$ such that the direct image of x under f is not equal to x.

9. *Axiom of Choice.* For every set of sets x there is a function $f : x \to \bigcup x$ with $f(y) \in y$ for all $y \in x$.

We remark here that in everyday discourse, comprehension is generally applied to all collections of sets, not just elementarily definable ones. This difference will hardly matter here; we only mention that in monadic second order logic this stronger from of comprehension is expressible and also the axiom of foundation.

Full Comprehension. For every class P and every set x, $\{y : y \in x \text{ and } x \in P\}$ is a set.

Foundation is usually defined as follows

Foundation. There is no infinite chain $x_0 \ni x_1 \ni x_2 \ni \cdots$.

In mathematical usage, one often forms certain collections of sets that can be shown not to be sets themselves. One example is the collection of all finite sets. The reason that it is not a set is that for every set x, $\{x\}$ also is a set. The function $x \mapsto \{x\}$ is injective (by extensionality), and so there are as many finite sets as there are sets. If the collection of finite sets were a set, say y, its powerset has strictly more elements than y by a theorem of Cantor. But this is impossible, since y has the size of the universe. Nevertheless, mathematicians

do use these collections (for example, the collection of Ω–algebras). This is not a problem, if the following is observed. A collection of sets is called a **class**. A class is a set iff it is contained in a set as an element. (We use 'iff' to abbreviate 'if and only if'.)

In set theory, numbers are defined as follows.

$$\begin{aligned} 0 &:= \varnothing \\ n+1 &:= \{k : k < n\} = \{0,1,2,\ldots,n-1\} \end{aligned} \tag{1.1}$$

The set of so–constructed numbers is denoted by ω. It is the set of **natural numbers**. In general, an **ordinal** (**number**) is a set that is transitively and linearly ordered by $\in$. (See below for these concepts.) For two ordinals κ and λ, either $\kappa \in \lambda$ (for which we also write $\kappa < \lambda$) or $\kappa = \lambda$ or $\lambda \in \kappa$.

Theorem 1.1 *For every set x there exists an ordinal κ and a bijective function $f\colon \kappa \to x$.*

f is also referred to as a **well–ordering** of x. The finite ordinals are exactly the natural numbers defined above. A **cardinal** (**number**) is an ordinal κ such that for every ordinal $\lambda < \kappa$ there is no onto map $f\colon \lambda \to \kappa$. It is not hard to see that every set can be well–ordered by a cardinal number, and this cardinal is unique. It is denoted by $|M|$ and called the **cardinality of** M. The smallest infinite cardinal is denoted by $\aleph_0$. The following is of fundamental importance.

Theorem 1.2 *For two sets x, y exactly one of the following holds: $|x| < |y|$, $|x| = |y|$ or $|x| > |y|$.*

By definition, $\aleph_0$ is actually identical to ω so that it is not really necessary to distinguish the two. However, we shall do so here for reasons of clarity. (For example, infinite cardinals have a different arithmetic than ordinals.) If M is finite, its cardinality is a natural number. If $|M| = \aleph_0$, M is called **countable**; it is **uncountable** otherwise. If M has cardinality κ, the cardinality of $\wp(M)$ is denoted by 2^κ. $2^{\aleph_0}$ is the cardinality of the set of all real numbers. $2^{\aleph_0}$ is strictly greater than $\aleph_0$ (but need not be the smallest uncountable cardinal). We remark here that the set of finite sets of natural numbers is countable.

If M is a set, a **partition** of M is a set $P \subseteq \wp(M)$ such that every member of P is nonempty, $\bigcup P = M$ and for all $A, B \in P$ such that $A \neq B$, $A \cap B = \varnothing$. If M and N are sets, $M \times N$ denotes the set of all pairs $\langle x, y\rangle$, where $x \in M$ and

$y \in N$. A definition of $\langle x,y\rangle$, which goes back to Kuratowski and Wiener, is as follows.

$$\langle x,y\rangle := \{x,\{x,y\}\} \tag{1.2}$$

Lemma 1.3 $\langle x,y\rangle = \langle u,v\rangle$ *iff* $x = u$ *and* $y = v$.

Proof. By extensionality, if $x = u$ and $y = v$ then $\langle x,y\rangle = \langle u,v\rangle$. Now assume that $\langle x,y\rangle = \langle u,v\rangle$. Then either $x = u$ or $x = \{u,v\}$, and $\{x,y\} = u$ or $\{x,y\} = \{u,v\}$. Assume that $x = u$. If $u = \{x,y\}$ then $x = \{x,y\}$, whence $x \in x$, in violation to foundation. Hence we have $\{x,y\} = \{u,v\}$. Since $x = u$, we must have $y = v$. This finishes the first case. Now assume that $x = \{u,v\}$. Then $\{x,y\} = u$ cannot hold, for then $u = \{\{u,v\},y\}$, whence $u \in \{u,v\} \in u$. So, we must have $\{x,y\} = \{u,v\}$. However, this gives $x = \{x,y\}$, once again a contradiction. So, $x = u$ and $y = v$, as promised. $\square$

With these definitions, $M \times N$ is a set if M and N are sets. A **relation** from M to N is a subset of $M \times N$. We write xRy if $\langle x,y\rangle \in R$. Particularly interesting is the case $M = N$. A relation $R \subseteq M \times M$ is called **reflexive** if xRx for all $x \in M$; **symmetric** if from xRy follows that yRx. R is called **transitive** if from xRy and yRz follows xRz. An **equivalence relation** on M is a reflexive, symmetric and transitive relation on M. A pair $\langle M,<\rangle$ is called an **ordered set** if M is a set and $<$ a transitive, irreflexive binary relation on M. $<$ is then called a (**strict**) **ordering on** M and M is then called **ordered by** $<$. $<$ is **linear** if for any two elements $x,y \in M$ either $x < y$ or $x = y$ or $y < x$. A **partial ordering** is a relation which is reflexive, transitive and antisymmetric; the latter means that from xRy and yRx follows $x = y$.

If $R \subseteq M \times N$ is a relation, we write $R^{\smile} := \{\langle x,y\rangle : yRx\}$ for the so–called **converse of** R. This is a relation from N to M. If $S \subseteq N \times P$ and $T \subseteq M \times N$ are relations, put

$$\begin{aligned} R \circ S &:= \{\langle x,y\rangle : \text{ for some } z\colon xRzSy\} \\ R \cup T &:= \{\langle x,y\rangle : xRy \text{ or } xTy\} \end{aligned} \tag{1.3}$$

We have $R \circ S \subseteq M \times P$ and $R \cup T \subseteq M \times N$. In case $M = N$ we still make further definitions. We put $\Delta_M := \{\langle x,x\rangle : x \in M\}$ and call this set the **diagonal on** M. Now put

$$\begin{aligned} R^0 &:= \Delta_M & R^{n+1} &:= R \circ R^n \\ R^+ &:= \bigcup_{0<i\in\omega} R^i & R^* &:= \bigcup_{i\in\omega} R^i \end{aligned} \tag{1.4}$$

R^+ is the smallest transitive relation which contains R. It is therefore called the **transitive closure of** R. R^* is the smallest reflexive and transitive relation containing R.

A **partial function** from M to N is a relation $f \subseteq M \times N$ such that if $x f y$ and $x f z$ then $y = z$. f is a **function** if for every x there is a y such that $x\ f\ y$. We write $y = f(x)$ to say that $x f y$ and $f\colon M \to N$ to say that f is a function from M to N. If $P \subseteq M$ then $f \restriction P := f \cap (P \times N)$. Further, $f\colon M \twoheadrightarrow N$ abbreviates that f is a **surjective** function, that is, every $y \in N$ is of the form $y = f(x)$ for some $x \in M$. And we write $f\colon M \rightarrowtail N$ to say that f is **injective**, that is, for all $x, x' \in M$, if $f(x) = f(x')$ then $x = x'$. f is **bijective** if it is injective as well as surjective. Finally, we write $f\colon x \mapsto y$ if $y = f(x)$. If $X \subseteq M$ then $f[X] := \{f(x) : x \in X\}$ is the so–called **direct image of** X **under** f. We warn the reader of the difference between $f(X)$ and $f[X]$. For example, let $\mathrm{suc}\colon \omega \to \omega\colon x \mapsto x+1$. Then according to the definition of natural numbers above we have $\mathrm{suc}(4) = 5$ and $\mathrm{suc}[4] = \{1,2,3,4\}$, since $4 = \{0,1,2,3\}$. Let M be an arbitrary set. There is a bijection between the set of subsets of M and the set of functions from M to $2 = \{0,1\}$, which is defined as follows. For $N \subseteq M$ we call $\chi_N\colon M \to \{0,1\}$ the **characteristic function** of N if $\chi_N(x) = 1$ iff $x \in N$. Let $y \in N$ and $Y \subseteq N$; then put $f^{-1}(y) := \{x : f(x) = y\}$ and $f^{-1}[Y] := \{x : f(x) \in Y\}$. If f is injective, $f^{-1}(y)$ denotes the unique x such that $f(x) = y$ (if that exists). We shall see to it that this overload in notation does not give rise to confusions.

M^n, $n \in \omega$, denotes the set of n–tuples of elements from M.

$$M^1 := M \qquad\qquad M^{n+1} := M^n \times M \tag{1.5}$$

In addition, $M^0 := 1 (= \{\varnothing\})$. Then an n–tuple of elements from M is an element of M^n. Depending on need we shall write $\langle x_i : i < n\rangle$ or $\langle x_0, x_1, \ldots, x_{n-1}\rangle$ for a member of M^n.

An n–**ary relation** on M is a subset of M^n, an n–**ary function** on M is a function $f\colon M^n \to M$. $n = 0$ is admitted. A 0–ary relation is a subset of 1, hence it is either the empty set or the set 1 itself. A 0–ary function on M is a function $c\colon 1 \to M$. We also call it a **constant**. The **value** of this constant is the element $c(\varnothing)$. Let R be an n–ary relation and $\vec{x} \in M^n$. Then we write $R(\vec{x})$ in place of $\vec{x} \in R$.

Now let F be a set and $\Omega\colon F \to \omega$. The pair $\langle F, \Omega\rangle$, also denoted by Ω alone, is called a **signature** and F the set of **function symbols**.

Definition 1.4 *Let $\Omega\colon F \to \omega$ be a signature and A a nonempty set. Further, let Π be a mapping which assigns to every $f \in F$ an $\Omega(f)$–ary function on A. Then we call the pair $\mathfrak{A} := \langle A, \Pi\rangle$ an* ***Ω–algebra****. Ω–algebras are in general denoted by upper case German letters.*

In order not to get drowned in notation we write $f^{\mathfrak{A}}$ for the function $\Pi(f)$. In place of denoting $\mathfrak{A}$ by the pair $\langle A, \Pi\rangle$ we shall denote it somewhat ambiguously by $\langle A, \{f^{\mathfrak{A}} : f \in F\}\rangle$. We warn the reader that the latter notation may give rise to confusion since functions of the same arity can be associated with different function symbols. However, this problem shall not arise.

The set of Ω–terms is the smallest set Tm_Ω such that if $f \in F$ and $t_i \in \mathrm{Tm}_\Omega$, $i < \Omega(f)$, also $f(t_0, \ldots, t_{\Omega(f)-1}) \in \mathrm{Tm}_\Omega$. Terms are abstract entities; they are not to be equated with functions nor with the strings by which we denote them. To begin we define the **level** of a term. If $\Omega(f) = 0$, then $f()$ is a term of level 0, which we also denote by 'f'. If t_i, $i < \Omega(f)$, are terms of level n_i, then $f(t_0, \ldots, t_{\Omega(f)-1})$ is a term of level $1 + \max\{n_i : i < \Omega(f)\}$. Many proofs run by induction on the level of terms, we therefore speak about *induction on the construction of the term*. Two terms u and v are equal, in symbols $u = v$, if they have identical level and either they are both of level 0 and there is an $f \in F$ such $u = v = f()$ or there is an $f \in F$, and terms s_i, t_i, $i < \Omega(f)$, such that $u = f(s_0, \ldots, s_{\Omega(f)-1})$ and $v = f(t_0, \ldots, t_{\Omega(f)-1})$ as well as $s_i = t_i$ for all $i < \Omega(f)$.

An important example of an Ω–algebra is the so–called *term algebra*. We choose an arbitrary set X of symbols, which must be disjoint from F. The signature is extended to $F \cup X$ such that the symbols of X have arity 0. The terms over this new signature are called **Ω–terms over** X. The set of Ω–terms over X is denoted by $\mathrm{Tm}_\Omega(X)$. Then we have $\mathrm{Tm}_\Omega = \mathrm{Tm}_\Omega(\varnothing)$. For many purposes (indeed most of the purposes of this book) the terms Tm_Ω are sufficient. For we can always resort to the following trick. For each $x \in X$ add a 0–ary function symbol $\underline{x}$ to F. This gives a new signature Ω_X, also called the **constant expansion of** Ω by X. Then Tm_{Ω_X} can be canonically identified with $\mathrm{Tm}_\Omega(X)$.

There is an algebra which has as its objects the terms and which interprets the function symbols as follows.

$$\Pi(f)\colon \langle t_i : i < \Omega(f)\rangle \mapsto f(t_0, \ldots, t_{\Omega(f)-1}) \tag{1.6}$$

Then $\mathfrak{Tm}_\Omega(X) := \langle \mathrm{Tm}_\Omega(X), \Pi\rangle$ is an Ω–algebra, called the **term algebra generated by** X. It has the following property. For any Ω–algebra $\mathfrak{A}$ and any

map $v\colon X \to A$ there is exactly one homomorphism $\overline{v}\colon \mathrm{Tm}_\Omega(X) \to \mathfrak{A}$ such that $\overline{v} \restriction X = v$. This will be restated in Proposition 1.6.

Definition 1.5 *Let $\mathfrak{A}$ be an Ω–algebra and $X \subseteq A$. We say that X* ***generates*** *$\mathfrak{A}$ if A is the smallest subset which contains X and which is closed under all functions $f^{\mathfrak{A}}$. If $|X| = \kappa$ we say that $\mathfrak{A}$ is* ***κ–generated****. Let $\mathcal{K}$ be a class of Ω–algebras and $\mathfrak{A} \in \mathcal{K}$. We say that $\mathfrak{A}$ is* ***freely generated by*** *X* ***in*** *$\mathcal{K}$ if for every $\mathfrak{B} \in \mathcal{K}$ and maps $v\colon X \to B$ there is exactly one homomorphism $\overline{v}\colon \mathfrak{A} \to \mathfrak{B}$ such that $\overline{v} \restriction X = v$. If $|X| = \kappa$ we say that $\mathfrak{A}$ is* ***freely κ–generated in*** *$\mathcal{K}$.*

Proposition 1.6 *Let Ω be a signature, and let X be disjoint from F. Then the term algebra over X, $\mathfrak{Tm}_\Omega(X)$, is freely generated by X in the class of all Ω–algebras.*

The following is left as an exercise. It is the justification for writing $\mathfrak{Fr}_{\mathcal{K}}(\kappa)$ for the (up to isomorphism unique) freely κ–generated algebra of $\mathcal{K}$. In varieties such an algebra always exists.

Proposition 1.7 *Let $\mathcal{K}$ be a class of Ω–algebras and κ a cardinal number. If $\mathfrak{A}$ and $\mathfrak{B}$ are both freely κ–generated in $\mathcal{K}$ they are isomorphic.*

Maps of the form $\sigma\colon X \to \mathrm{Tm}_\Omega(X)$, as well as their homomorphic extensions are called **substitutions**. If t is a term over X, we also write $\sigma(t)$ in place of $\overline{\sigma}(t)$. Another notation, frequently employed in this book, is as follows. Given terms s_i, $i < n$, we write $[s_i/x_i\colon i < n]t$ in place of $\sigma(t)$, where σ is defined as follows.

$$\sigma(y) := \begin{cases} s_i & \text{if } y = x_i, \\ y & \text{else.} \end{cases} \tag{1.7}$$

(Most authors write $t[s_i/x_i\colon i < n]$, but this notation will cause confusion with other notation that we use.)

Terms induce term functions on a given Ω–algebra $\mathfrak{A}$. Let t be a term with variables x_i, $i < n$. (None of these variables has to occur in the term.) Then $t^{\mathfrak{A}}\colon A^n \to A$ is defined inductively as follows (with $\vec{a} = \langle a_i : i < \Omega(f)\rangle$).

$$\begin{aligned} x_i^{\mathfrak{A}}(\vec{a}) &:= a_i \\ (f(t_0, \ldots, t_{\Omega(f)-1}))^{\mathfrak{A}}(\vec{a}) &:= f^{\mathfrak{A}}(t_0^{\mathfrak{A}}(\vec{a}), \ldots, t_{\Omega(f)-1}^{\mathfrak{A}}(\vec{a})) \end{aligned} \tag{1.8}$$

We denote by $\mathrm{Clo}_n(\mathfrak{A})$ the set of n–ary term functions on $\mathfrak{A}$. This set is also called the **clone of n–ary term functions of** $\mathfrak{A}$. A **polynomial of** $\mathfrak{A}$ is a

term function over an algebra that is like $\mathfrak{A}$ but additionally has a constant for each element of A. (So, we form the constant expansion of the signature with every $a \in A$. Moreover, $\underline{a}$ (more exactly, $\underline{a}()$) shall have value a in A.) The clone of n–ary term functions of this algebra is denoted by $\mathrm{Pol}_n(\mathfrak{A})$. For example, $((x_0 + x_1) \cdot x_0)$ is a term and denotes a binary term function in an algebra for the signature containing only $\cdot$ and $+$. However, $(2 + (x_0 \cdot x_0))$ is a polynomial but not a term. Suppose that we add a constant $\mathbf{1}$ to the signature, with denotation 1 in the natural numbers. Then $(2 + (x_0 \cdot x_0))$ is still not a term of the expanded language (it lacks the symbol 2), but the associated function actually is a term function, since it is identical with the function induced by the term $((\mathbf{1} + \mathbf{1}) + (x_0 \cdot x_0))$.

Definition 1.8 *Let $\mathfrak{A} = \langle A, \{f^{\mathfrak{A}} \colon f \in F\}\rangle$ and $\mathfrak{B} = \langle B, \{f^{\mathfrak{B}} \colon f \in F\}\rangle$ be Ω–algebras and $h \colon A \to B$. h is called a **homomorphism** if for every $f \in F$ and every $\Omega(f)$–tuple $\vec{x} \in A^{\Omega(f)}$ we have*

$$h(f^{\mathfrak{A}}(\vec{x})) = f^{\mathfrak{B}}(h(x_0), h(x_1), \ldots, h(x_{\Omega(f)-1})) \tag{1.9}$$

*We write $h \colon \mathfrak{A} \to \mathfrak{B}$ if h is a homomorphism from $\mathfrak{A}$ to $\mathfrak{B}$. Further, we write $h \colon \mathfrak{A} \twoheadrightarrow \mathfrak{B}$ if h is a surjective homomorphism and $h \colon \mathfrak{A} \rightarrowtail \mathfrak{B}$ if h is an injective homomorphism. h is an **isomorphism** if h is injective as well as surjective. $\mathfrak{B}$ is called **isomorphic** to $\mathfrak{A}$, in symbols $\mathfrak{A} \cong \mathfrak{B}$ if there is an isomorphism from $\mathfrak{A}$ to $\mathfrak{B}$. If $\mathfrak{A} = \mathfrak{B}$ we call h an **endomorphism of** $\mathfrak{A}$; if h is additionally bijective then h is called an **automorphism** of $\mathfrak{A}$.*

If $h \colon A \to B$ is an isomorphism from $\mathfrak{A}$ to $\mathfrak{B}$ then $h^{-1} \colon B \to A$ is an isomorphism from $\mathfrak{B}$ to $\mathfrak{A}$.

Definition 1.9 *Let $\mathfrak{A}$ be an Ω–algebra and Θ a binary relation on A. Θ is called a **congruence relation on** $\mathfrak{A}$ if Θ is an equivalence relation and for all $f \in F$ and all $\vec{x}, \vec{y} \in A^{\Omega(f)}$ we have:*

$$\text{If } x_i \,\Theta\, y_i \text{ for all } i < \Omega(f) \text{ then } f^{\mathfrak{A}}(\vec{x}) \,\Theta\, f^{\mathfrak{A}}(\vec{y}). \tag{1.10}$$

We also write $\vec{x} \,\Theta\, \vec{y}$ in place of '$x_i \,\Theta\, y_i$ for all $i < \Omega(f)$'. If Θ is an equivalence relation put

$$[x]\Theta := \{y : x \,\Theta\, y\} \tag{1.11}$$

We call $[x]\Theta$ the **equivalence class of** x. Then for all x and y we have either $[x]\Theta = [y]\Theta$ or $[x]\Theta \cap [y]\Theta = \varnothing$. Further, we always have $x \in [x]\Theta$. If Θ

additionally is a congruence relation then the following holds: if $y_i \in [x_i]\Theta$ for all $i < \Omega(f)$ then $f^{\mathfrak{A}}(\vec{y}) \in [f^{\mathfrak{A}}(\vec{x})]\Theta$. Therefore the following definition is independent of representatives.

$$[f^{\mathfrak{A}}]\Theta([x_0]\Theta, [x_1]\Theta, \ldots, [x_{\Omega(f)-1}]\Theta) := [f^{\mathfrak{A}}(\vec{x})]\Theta \tag{1.12}$$

Namely, let $y_0 \in [x_0]\Theta, \ldots, y_{\Omega(f)-1} \in [x_{\Omega(f)-1}]\Theta$. Then $y_i \,\Theta\, x_i$ for all $i < \Omega(f)$. Then because of (1.10) we immediately have $f^{\mathfrak{A}}(\vec{y}) \;\Theta\; f^{\mathfrak{A}}(\vec{x})$. This simply means $f^{\mathfrak{A}}(\vec{y}) \in [f^{\mathfrak{A}}(\vec{x})]\Theta$. Put

$$A/\Theta := \{[x]\Theta : x \in A\} \tag{1.13}$$

$$\mathfrak{A}/\Theta := \langle A/\Theta, \{[f^{\mathfrak{A}}]\Theta : f \in F\}\rangle \tag{1.14}$$

We call $\mathfrak{A}/\Theta$ the **factorization of** $\mathfrak{A}$ **by** Θ. The map $h_\Theta\colon x \mapsto [x]\Theta$ is easily proved to be a homomorphism.

Conversely, let $h\colon \mathfrak{A} \to \mathfrak{B}$ be a homomorphism. Then put

$$\ker(h) := \{\langle x, y\rangle \in A^2 : h(x) = h(y)\} \tag{1.15}$$

$\ker(h)$ is a congruence relation on $\mathfrak{A}$. Furthermore, $\mathfrak{A}/\ker(h)$ is isomorphic to $\mathfrak{B}$ if h is surjective. A set $B \subseteq A$ is **closed under** $f \in F$ if for all $\vec{x} \in B^{\Omega(f)}$ we have $f^{\mathfrak{A}}(\vec{x}) \in B$.

Definition 1.10 *Let $\langle A, \{f^{\mathfrak{A}} : f \in F\}\rangle$ be an Ω–algebra and $B \subseteq A$ closed under all $f \in F$. Put $f^{\mathfrak{B}} := f^{\mathfrak{A}} \restriction B^{\Omega(f)}$. The pair $\langle B, \{f^{\mathfrak{B}} : f \in F\}\rangle$ is called a* ***subalgebra*** *of $\mathfrak{A}$.*

The product of the algebras $\mathfrak{A}_i$, $i \in I$, is defined as follows. The carrier set is the set of functions $\alpha\colon I \to \bigcup_{i\in I} A_i$ such that $\alpha(i) \in A_i$ for all $i \in I$. Call this set P. For an n–ary function symbol f put

$$\begin{aligned} f^{\mathfrak{P}}(\alpha_0, \ldots, \alpha_{n-1})(i) \\ := \langle f^{\mathfrak{A}_i}(\alpha_0(i)), f^{\mathfrak{A}_i}(\alpha_1(i)), \ldots, f^{\mathfrak{A}_i}(\alpha_{n-1}(i))\rangle \end{aligned} \tag{1.16}$$

The resulting algebra is denoted by $\prod_{i\in I} \mathfrak{A}_i$. One also defines the product $\mathfrak{A} \times \mathfrak{B}$ in the following way. The carrier set is $A \times B$ and for an n–ary function symbol f we put

$$\begin{aligned} f^{\mathfrak{A}\times\mathfrak{B}}(\langle a_0, b_0\rangle, \ldots, \langle a_{n-1}, b_{n-1}\rangle) \\ := \langle f^{\mathfrak{A}}(a_0, \ldots, a_{n-1}), f^{\mathfrak{B}}(b_0, \ldots, b_{n-1})\rangle \end{aligned} \tag{1.17}$$

The algebra $\mathfrak{A} \times \mathfrak{B}$ is isomorphic to the algebra $\prod_{i \in 2} \mathfrak{A}_i$, where $\mathfrak{A}_0 := \mathfrak{A}$, $\mathfrak{A}_1 := \mathfrak{B}$. However, the two algebras are not identical. (Can you verify this?)

A particularly important concept is that of a *variety* or *equationally definable class of algebras*.

Definition 1.11 *Let Ω be a signature. A class of Ω–algebras is called a* ***variety*** *if it is closed under isomorphic copies, subalgebras, homomorphic images, and (possibly infinite) products.*

Let $V := \{x_i : i \in \omega\}$ be the set of variables. An **equation** is a pair $\langle s,t \rangle$ of Ω–terms (involving variables from V). We introduce a formal symbol '$=$' and write $s = t$ for this pair. An algebra $\mathfrak{A}$ satisfies the equation $s = t$ iff for all maps $v : V \to A$, $\overline{v}(s) = \overline{v}(t)$. We then write $\mathfrak{A} \vDash s = t$. A class $\mathcal{K}$ of Ω–algebras satisfies this equation if every algebra of $\mathcal{K}$ satisfies it. We write $\mathcal{K} \vDash s = t$.

Proposition 1.12 *The following holds for all classes $\mathcal{K}$ of Ω–algebras.*

① $\mathcal{K} \vDash s = s$.

② *If $\mathcal{K} \vDash s = t$ then $\mathcal{K} \vDash t = s$.*

③ *If $\mathcal{K} \vDash s = t; t = u$ then $\mathcal{K} \vDash s = u$.*

④ *If $\mathcal{K} \vDash s_i = t_i$ for all $i < \Omega(f)$ then $\mathcal{K} \vDash f(\vec{s}) = f(\vec{t})$.*

⑤ *If $\mathcal{K} \vDash s = t$ and $\sigma \colon V \to \mathrm{Tm}_\Omega(V)$ is a substitution, then $\mathcal{K} \vDash \sigma(s) = \sigma(t)$.*

The verification of this is routine. It follows from the first three facts that equality is an equivalence relation on the algebra $\mathfrak{Tm}_\Omega(V)$, and together with the fourth that the set of equations valid in $\mathcal{K}$ form a congruence on $\mathfrak{Tm}_\Omega(V)$. There is a bit more we can say. Call a congruence Θ on $\mathfrak{A}$ **fully invariant** if for all endomorphisms $h \colon \mathfrak{A} \to \mathfrak{A}$: if $x \,\Theta\, y$ then $h(x) \,\Theta\, h(y)$. The next theorem follows immediately once we observe that the endomorphisms of $\mathfrak{Tm}_\Omega(V)$ are exactly the substitution maps. To this end, let $h \colon \mathfrak{Tm}_\Omega(V) \to \mathfrak{Tm}_\Omega(V)$. Then h is uniquely determined by $h \restriction V$, since $\mathfrak{Tm}_\Omega(V)$ is freely generated by V. It is easily computed that h is the substitution defined by $h \restriction V$. Moreover, every map $v \colon V \to \mathfrak{Tm}_\Omega(V)$ induces a homomorphism $\overline{v} \colon \mathfrak{Tm}_\Omega(V) \to \mathfrak{Tm}_\Omega(V)$, which is unique. Now write $\mathrm{Eq}(\mathcal{K}) := \{\langle s,t \rangle : \mathcal{K} \vDash s = t\}$.

Corollary 1.13 *Let $\mathcal{K}$ be a class of Ω–algebras. Then* $\mathrm{Eq}(\mathcal{K})$ *is a fully invariant congruence on* $\mathfrak{Tm}_\Omega(V)$.

Let E be a set of equations. Then put

$$\mathsf{Alg}(E) := \{\mathfrak{A} : \text{ for all } \langle s,t\rangle \in E : \mathfrak{A} \vDash s = t\} \tag{1.18}$$

This is a class of Ω–algebras. Classes of Ω–algebras that have the form $\mathsf{Alg}(E)$ for some E are called **equationally definable**.

Proposition 1.14 *Let E be a set of equations. Then $\mathsf{Alg}(E)$ is a variety.*

We state without proof the following result.

Theorem 1.15 (Birkhoff) *Every variety is an equationally definable class. Furthermore, there is a biunique correspondence between varieties and fully invariant congruences on the algebra $\mathfrak{Tm}_\Omega(\aleph_0)$.*

The idea for the proof is as follows. It can be shown that every variety has free algebras. For every cardinal number κ, $\mathfrak{Fr}_\mathcal{K}(\kappa)$ exists. Moreover, a variety is uniquely characterized by $\mathfrak{Fr}_\mathcal{K}(\aleph_0)$. In fact, every algebra is a subalgebra of a direct image of some product of $\mathfrak{Fr}_\mathcal{K}(\aleph_0)$. Thus, we need to investigate the equations that hold in the latter algebra. The other algebras will satisfy these equations, too. The free algebra is the image of $\mathfrak{Tm}_\Omega(V)$ under the map $x_i \mapsto i$. The induced congruence is fully invariant, by the freeness of $\mathfrak{Fr}_\mathcal{K}(\aleph_0)$. Hence, this congruence simply *is* the set of equations valid in the free algebra, hence in the whole variety. Finally, if E is a set of equations, we write $E \vDash t = u$ if $\mathfrak{A} \vDash t = u$ for all $\mathfrak{A} \in \mathsf{Alg}(E)$.

Theorem 1.16 (Birkhoff) *$E \vDash t = u$ iff $t = u$ can be derived from E by means of the rules given in Proposition 1.12.*

The notion of an algebra can be extended into two directions, both of which shall be relevant for us. The first is the concept of a many–sorted algebra.

Definition 1.17 *A **sorted signature** is a triple $\langle F, \mathcal{S}, \Omega\rangle$, where F and $\mathcal{S}$ are sets, the set of **function symbols** and of **sorts**, respectively, and $\Omega\colon F \to \mathcal{S}^+$ a function assigning to each element of F its so–called **signature**. We shall denote the signature by the letter Ω, as in the unsorted case.*

So, the signature of a function is a (nonempty) sequence of sorts. The last member of that sequence tells us what sort the result has, while the others tell us what sort the individual arguments of that function symbol have.

Definition 1.18 *A **(sorted) Ω–algebra** is a pair $\mathfrak{A} = \langle \{A_\sigma : \sigma \in \mathcal{S}\}, \Pi \rangle$ such that for every $\sigma \in \mathcal{S}$ A_σ is a set and for every $f \in F$ such that $\Omega(f) = \langle \sigma_i : i < n+1 \rangle$*

$$\Pi(f) \colon A_{\sigma_0} \times A_{\sigma_1} \times \cdots \times A_{\sigma_{n-1}} \to A_{\sigma_n} \tag{1.19}$$

*If $\mathfrak{B} = \langle \{B_\sigma : \sigma \in \mathcal{S}\}, \Sigma \rangle$ is another Ω–algebra, a **(sorted) homomorphism from $\mathfrak{A}$ to $\mathfrak{B}$** is a set $\{h_\sigma \colon A_\sigma \to B_\sigma : \sigma \in \mathcal{S}\}$ of functions such that for each $f \in F$ with signature $\langle \sigma_i : i < n+1 \rangle$:*

$$h_{\sigma_n}(f^{\mathfrak{A}}(a_0, \ldots, a_{n-1})) = f^{\mathfrak{B}}(h_{\sigma_0}(a_0), \ldots, h_{\sigma_{n-1}}(a_{n-1})) \tag{1.20}$$

*A **many–sorted algebra** is an Ω–algebra of some signature Ω.*

Evidently, if $\mathcal{S} = \{\sigma\}$ for some σ, then the notions coincide (modulo trivial adaptations) with those of unsorted algebras. Terms are defined as before, but now they are sorted. First, for each sort we assume a countably infinite set V_σ of variables. Moreover, $V_\sigma \cap V_\tau = \varnothing$ whenever $\sigma \neq \tau$. Now, every term is given a unique sort in the following way.

① If $x \in V_\sigma$, then x has sort σ.

② $f(t_0, \ldots, t_{n-1})$ has sort σ_n, if $\Omega(f) = \langle \sigma_i : i < n+1 \rangle$ and t_i has sort σ_i for all $i < n$.

The set of terms over V is denoted by $\mathrm{Tm}_\Omega(V)$. This can be turned into a sorted Ω–algebra; simply let $\mathrm{Tm}_\Omega(V)_\sigma$ be the set of terms of sort σ. Again, given a map v that assigns to a variable of sort σ an element of A_σ, there is a unique homomorphism $\overline{v}$ from the Ω–algebra of terms into $\mathfrak{A}$. If t has sort σ, then $\overline{v}(t) \in A_\sigma$. A **sorted equation** is a pair $\langle s, t \rangle$, where s and t are of equal sort. We denote this pair by $s = t$. We write $\mathfrak{A} \vDash s = t$ if for all maps v into $\mathfrak{A}$, $\overline{v}(s) = \overline{v}(t)$. The Birkhoff Theorems have direct analogues for the many sorted algebras, and can be proved in the same way.

Sorted algebras are one way of introducing partiality. To be able to compare the two approaches, we first have to introduce partial algebras. We shall now return to the unsorted notions, although it is possible — even though not really desirable — to introduce partial many–sorted algebras as well.

Definition 1.19 *Let Ω be an unsorted signature. A **partial Ω–algebra** is a pair $\langle A, \Pi \rangle$, where A is a set and for each $f \in F$: $\Pi(f)$ is a partial function from $A^{\Omega(f)}$ to A.*

The definitions of canonical terms split into different notions in the partial case.

Definition 1.20 *Let $\mathfrak{A}$ and $\mathfrak{B}$ be partial Ω–algebras, and $h\colon A \to B$ a map. h is a **weak homomorphism from** $\mathfrak{A}$ **to** $\mathfrak{B}$ if for every $\vec{a} \in A^{\Omega(f)}$ we have $h(f^{\mathfrak{A}}(\vec{a})) = f^{\mathfrak{B}}(h(\vec{a}))$ if both sides are defined. h is a **homomorphism** if it is a weak homomorphism and for every $\vec{a} \in A^{\Omega(f)}$ if $h(f^{\mathfrak{A}}(\vec{a}))$ is defined then so is $f^{\mathfrak{B}}(h(\vec{a}))$. Finally, h is a **strong homomorphism** if it is a homomorphism and $h(f^{\mathfrak{A}}(\vec{a}))$ is defined iff $f^{\mathfrak{B}}(h(\vec{a}))$ is. $\mathfrak{A}$ is a **strong subalgebra of** $\mathfrak{B}$ if $A \subseteq B$ and the identity map is a strong homomorphism.*

Definition 1.21 *An equivalence relation Θ on A is called a **weak congruence of** $\mathfrak{A}$ if for every $f \in F$ and every $\vec{a}, \vec{c} \in A^{\Omega(f)}$ if $\vec{a}\ \Theta\ \vec{c}$ and $f^{\mathfrak{A}}(\vec{a})$, $f^{\mathfrak{A}}(\vec{c})$ are both defined, then $f^{\mathfrak{A}}(\vec{a})\ \Theta\ f^{\mathfrak{A}}(\vec{c})$. Θ is **strong** if in addition $f^{\mathfrak{A}}(\vec{a})$ is defined iff $f^{\mathfrak{A}}(\vec{c})$ is.*

It can be shown that the equivalence relation induced by a weak (strong) homomorphism is a weak (strong) congruence, and that every weak (strong) congruence defines a surjective weak (strong) homomorphism.

Let $v\colon V \to A$ be a function, $t = f(s_0, \ldots, s_{\Omega(f)-1})$ a term. Then $\overline{v}(t)$ is defined iff (a) $\overline{v}(s_i)$ is defined for every $i < \Omega(f)$ and (b) $f^{\mathfrak{A}}$ is defined on $\langle \overline{v}(s_i) : i < n \rangle$. Now, we write $\langle \mathfrak{A}, v \rangle \vDash^w s = t$ if $\overline{v}(s) = \overline{v}(t)$ in case both are defined and equal; $\langle \mathfrak{A}, v \rangle \vDash^s s = t$ if $\overline{v}(s)$ is defined iff $\overline{v}(t)$ is and if one is defined the two are equal. An equation $s = t$ is said to hold in $\mathfrak{A}$ in the **weak** (**strong**) **sense**, if $\langle \mathfrak{A}, v \rangle \vDash^w s = t$ ($\langle \mathfrak{A}, v \rangle \vDash^s s = t$) for all $v\colon V \to A$. Proposition 1.12 holds with respect to $\vDash^s$ but not with respect to $\vDash^w$. Also, algebras satisfying an equation in the strong sense are closed under products, strong homomorphic images and under strong subalgebras.

The relation between classes of algebras and sets of equations is called a **Galois correspondence**. It is useful to know a few facts about such correspondences. Let A, B be sets and $R \subseteq A \times B$ (A and B may in fact also be classes). The triple $\langle A, B, R \rangle$ is called a **context**. Now define the following operators:

$$^{\uparrow}\colon \wp(A) \to \wp(B)\colon O \mapsto \{y \in B : \text{ for all } x \in O : x\,R\,y\} \tag{1.21}$$

$$^{\downarrow}\colon \wp(B) \to \wp(A)\colon P \mapsto \{x \in A : \text{ for all } y \in P : x\,R\,y\} \tag{1.22}$$

One calls $O^{\uparrow}$ the **intent of** $O \subseteq A$ and $P^{\downarrow}$ the **extent** of $P \subseteq B$.

Theorem 1.22 *Let $\langle A,B,R\rangle$ be a context. Then the following holds for all $O,O^* \subseteq A$ and all $P,P^* \subseteq B$.*

① $O \subseteq P^{\downarrow}$ *iff* $O^{\uparrow} \supseteq P$.

② *If* $O \subseteq O^*$ *then* $O^{\uparrow} \supseteq O^{*\uparrow}$.

③ *If* $P \subseteq P^*$ *then* $P^{\downarrow} \supseteq P^{*\downarrow}$.

④ $O \subseteq O^{\uparrow\downarrow}$.

⑤ $P \subseteq P^{\downarrow\uparrow}$.

Proof. Notice that if $\langle A,B,R\rangle$ is a context, $\langle B,A,R^{\smile}\rangle$ also is a context, and so we only need to show ①, ② and ④. ①. $O \subseteq P^{\downarrow}$ iff every $x \in O$ stands in relation R to every member of P iff $P \subseteq O^{\uparrow}$. ②. If $O \subseteq O^*$ and $y \in O^{*\uparrow}$, then for every $x \in O^*$: $x\,R\,y$. This means that for every $x \in O$: $x\,R\,y$, which is the same as $y \in O^{\uparrow}$. ④. Notice that $O^{\uparrow} \supseteq O^{\uparrow}$ by ① implies $O \subseteq O^{\uparrow\downarrow}$. □

Definition 1.23 *Let M be a set and $H\colon \wp(M) \to \wp(M)$ a function. H is called a* ***closure operator on*** *M if for all $X,Y \subseteq M$ the following holds.*

① $X \subseteq H(X)$.

② *If* $X \subseteq Y$ *then* $H(X) \subseteq H(Y)$.

③ $H(X) = H(H(X))$.

A set X is called ***closed*** *if $X = H(X)$.*

Proposition 1.24 *Let $\langle A,B,R\rangle$ be a context. Then $O \mapsto O^{\uparrow\downarrow}$ and $P \mapsto P^{\downarrow\uparrow}$ are closure operators on A and B, respectively. The closed sets are the sets of the form $P^{\downarrow}$ for the first, and $O^{\uparrow}$ for the second operator.*

Proof. We have $O \subseteq O^{\uparrow\downarrow}$, from which $O^{\uparrow} \supseteq O^{\uparrow\downarrow\uparrow}$. On the other hand, $O^{\uparrow} \subseteq O^{\uparrow\downarrow\uparrow}$, so that we get $O^{\uparrow} = O^{\uparrow\downarrow\uparrow}$. Likewise, $P^{\downarrow} = P^{\downarrow\uparrow\downarrow}$ is shown. The claims now follow easily. □

Definition 1.25 *Let $\langle A,B,R\rangle$ be a context. A pair $\langle O,P\rangle \in \wp(A) \times \wp(B)$ is called a* ***concept*** *if $O = P^{\downarrow}$ and $P = O^{\uparrow}$.*

Theorem 1.26 *Let $\langle A,B,R\rangle$ be a context. The concepts are exactly the pairs of the form $\langle P^{\downarrow}, P^{\downarrow\uparrow}\rangle$, $P \subseteq B$, or, alternatively, the pairs of the form $\langle O^{\uparrow\downarrow}, O^{\uparrow}\rangle$, $O \subseteq A$.*

As a particular application we look again at the connection between classes of Ω–algebras and sets of equations over Ω–terms. (It suffices to take the set of Ω–algebras of size $< \kappa$ for a suitable κ to make this work.) Let Alg_Ω denote the class of Ω–algebras, Eq_Ω the set of equations. The triple $\langle \mathsf{Alg}_\Omega, \mathsf{Eq}_\Omega, \models \rangle$ is a context, and the map $^\uparrow$ is nothing but Eq and the map $^\downarrow$ nothing but Alg. The classes $\mathsf{Alg}(E)$ are the equationally definable classes, $\mathsf{Eq}(\mathcal{K})$ the equations valid in $\mathcal{K}$. Concepts are pairs $\langle \mathcal{K}, E \rangle$ such that $\mathcal{K} = \mathsf{Alg}(E)$ and $E = \mathsf{Eq}(\mathcal{K})$.

Often we shall deal with structures in which there are also relations in addition to functions. The definitions, insofar as they still make sense, are carried over analogously. However, the notation becomes more clumsy.

Definition 1.27 *Let F and G be disjoint sets and $\Omega\colon F \to \omega$ as well as $\Xi\colon G \to \omega$ functions. A pair $\mathfrak{A} = \langle A, \mathfrak{I} \rangle$ is called an $\langle \Omega, \Xi \rangle$–**structure** if for all $f \in F$ $\mathfrak{I}(f)$ is an $\Omega(f)$–ary function on A and for each $g \in G$ $\mathfrak{I}(g)$ is a $\Xi(g)$–ary relation on A. Ω is called the **functional signature**, Ξ the **relational signature** of $\mathfrak{A}$.*

Whenever we can afford it we shall drop the qualification '$\langle \Omega, \Xi \rangle$' and simply talk of 'structures'. If $\langle A, \mathfrak{I} \rangle$ is an $\langle \Omega, \Xi \rangle$–structure, then $\langle A, \mathfrak{I} \restriction F \rangle$ is an Ω–algebra. An Ω–algebra can be thought of in a natural way as a $\langle \Omega, \varnothing \rangle$–structure, where $\varnothing$ is the empty relational signature. We use a convention similar to that of algebras. Furthermore, we denote relations by upper case Roman letters such as R, S and so on. Let $\mathfrak{A} = \langle A, \{f^{\mathfrak{A}} : f \in F\}, \{R^{\mathfrak{A}} : R \in G\} \rangle$ and $\mathfrak{B} = \langle B, \{f^{\mathfrak{B}} : f \in F\}, \{R^{\mathfrak{B}} : R \in G\} \rangle$ be structures of the same signature. A map $h\colon A \to B$ is called an **isomorphism** from $\mathfrak{A}$ to $\mathfrak{B}$, if h is bijective and for all $f \in F$ and all $\vec{x} \in A^{\Omega(f)}$ we have

$$h(f^{\mathfrak{A}}(\vec{x})) = f^{\mathfrak{B}}(h(\vec{x})) \tag{1.23}$$

as well as for all $R \in G$ and all $\vec{x} \in A^{\Xi(R)}$

$$R^{\mathfrak{A}}(\vec{x}) \quad \Leftrightarrow \quad R^{\mathfrak{B}}(h(x_0), h(x_1), \ldots, h(x_{\Xi(R)-1})) \tag{1.24}$$

Exercise 1. Since $y \mapsto \{y\}$ is an embedding of x into $\wp(x)$, we have $|x| \leq |\wp(x)|$. Show that $|\wp(x)| > |x|$ for every set. *Hint.* Let $f : x \to \wp(x)$ be any function. Look at the set $\{y : y \notin f(y)\} \subseteq x$. Show that it is not in $\mathrm{im}(f)$.

Exercise 2. Let $f\colon M \to N$ and $g\colon N \to P$. Show that if $g \circ f$ is surjective, g is surjective, and that if $g \circ f$ is injective, f is injective. Give in each case an

example that the converse fails.

Exercise 3. In set theory, one writes ${}^{N}M$ for the set of functions from N to M. Show that if $|N| = n$ and $|M| = m$, then $|{}^{N}M| = m^n$. Deduce that $|{}^{N}M| = |M^n|$. Can you find a bijection between these sets?

Exercise 4. Show that for relations $R, R' \subseteq M \times N$, $S, S' \subseteq N \times P$ we have

$$(R \cup R') \circ S = (R \circ S) \cup (R' \circ S) \tag{1.25a}$$

$$R \circ (S \cup S') = (R \circ S) \cup (R \circ S') \tag{1.25b}$$

Show by giving an example that analogous laws for $\cap$ do not hold.

Exercise 5. Let $\mathfrak{A}$ and $\mathfrak{B}$ be Ω–algebras for some signature Ω. Show that if $h\colon \mathfrak{A} \twoheadrightarrow \mathfrak{B}$ is a surjective homomorphism then $\mathfrak{B}$ is isomorphic to $\mathfrak{A}/\Theta$ with $x \,\Theta\, y$ iff $h(x) = h(y)$.

Exercise 6. Show that every Ω–algebra $\mathfrak{A}$ is the homomorphic image of a term algebra. *Hint.* Take X to be the set underlying $\mathfrak{A}$.

Exercise 7. Show that $\mathfrak{A} \times \mathfrak{B}$ is isomorphic to $\prod_{i \in \{0,1\}} \mathfrak{A}_i$, where $\mathfrak{A}_0 = \mathfrak{A}$, $\mathfrak{A}_1 = \mathfrak{B}$. Show also that $(\mathfrak{A} \times \mathfrak{B}) \times \mathfrak{C}$ is isomorphic to $\mathfrak{A} \times (\mathfrak{B} \times \mathfrak{C})$.

Exercise 8. Prove Proposition 1.7.

2. Semigroups and Strings

In formal language theory, languages are sets of strings over some alphabet. We assume throughout that an alphabet is a finite, nonempty set, usually called A. It has no further structure (but see Section 1.3), it only defines the material of primitive letters. We do not make any further assumptions on the size of A. The Latin alphabet consists of 26 letters, which actually exist in two variants (upper and lower case), and we also use a few punctuation marks and symbols as well as the blank. On the other hand, the Chinese 'alphabet' consists of several thousand letters!

Strings are very fundamental structures. Without a proper understanding of their workings one could not read this book, for example. A string over A is nothing but the result of successively placing elements of A after each other. It is not necessary to always use a fresh letter. If, for example, $A = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}$, then `abb`, `bac`, `caaba` are strings over A. We agree to use typewriter font to mark actual symbols (= pieces of ink), while letters in different font are

only proxy for letters (technically, they are variables for letters). Strings are denoted by a vector arrow, for example $\vec{w}$, $\vec{x}$, $\vec{y}$ and so on, to distinguish them from individual letters. Since paper is of bounded length, strings are not really written down in a continuous line, but rather in several lines, and on several pieces of paper, depending on need. The way a string is cut up into lines and pages is actually immaterial for its abstract constitution (unless we speak of paragraphs and similar textual divisions). We wish to abstract from these details. Therefore we define strings formally as follows.

Definition 1.28 *Let A be a set. A* ***string over*** *A is a function $\vec{x}\colon n \to A$ for some natural number n. n is called the* ***length of*** *$\vec{x}$ and is denoted by $|\vec{x}|$. $\vec{x}(i)$, $i < n$, is called the* ***ith segment*** *or the* ***ith letter of*** *$\vec{x}$. The unique string of length 0 is denoted by ε. If $\vec{x}\colon m \to A$ and $\vec{y}\colon n \to A$ are strings over A then $\vec{x}^\frown\vec{y}$ denotes the unique string of length $m+n$ for which the following holds:*

$$(\vec{x}^\frown\vec{y})(j) := \begin{cases} \vec{x}(j) & \text{if } j < m, \\ \vec{y}(j-m) & \text{else.} \end{cases} \tag{1.26}$$

We often write $\vec{x}\vec{y}$ in place of $\vec{x}^\frown\vec{y}$. In connection with this definition the set A is called the ***alphabet****, an element of A is also referred to as a* ***letter****. Unless stated otherwise, A is finite and nonempty.*

So, a string may also be written using simple concatenation. Hence we have $\texttt{abc}^\frown\texttt{baca} = \texttt{abcbaca}$. Note that there no blank is inserted between the two strings; for the blank is a *letter*. We denote it by $\square$. Two words of a language are usually separated by a blank possibly using additional punctuation marks. That the blank is a symbol is felt more clearly when we use a typewriter. If we want to have a blank, we need to press down a key in order to get it. For purely formal reasons we have added the empty string to the set of strings. It is not visible (unlike the blank). Hence, we need a special symbol for it, which is ε, in some other books also λ. We have

$$\vec{x}^\frown\varepsilon = \varepsilon^\frown\vec{x} = \vec{x} \tag{1.27}$$

We say, the empty string is the **unit** with respect to concatenation. For any triple of strings $\vec{x}$, $\vec{y}$ and $\vec{z}$ we have

$$\vec{x}^\frown(\vec{y}^\frown\vec{z}) = (\vec{x}^\frown\vec{y})^\frown\vec{z} \tag{1.28}$$

We therefore say that concatenation, $^\frown$, is **associative**. More on that below. We define the notation $\vec{x}^i$ by induction on i.

$$\begin{aligned} \vec{x}^0 &:= \varepsilon \\ \vec{x}^{i+1} &:= \vec{x}^i{}^\frown\vec{x} \end{aligned} \tag{1.29}$$

Furthermore, we define $\prod_{i<n}\vec{x}_i$ as follows.

$$\prod_{i<0}\vec{x}_i := \varepsilon, \qquad \prod_{i<n+1} := (\prod_{i<n}\vec{x}_i)^\frown\vec{x}_n \tag{1.30}$$

Note that the letter a is technically distinct from the string $\vec{x}\colon 1 \to A\colon 0 \mapsto \mathtt{a}$. They are nevertheless written in the same way, namely a. If $\vec{x}$ is a string over A and $A \subseteq B$, then $\vec{x}$ is a string over B. The set of all strings over A is denoted by A^*.

Let $<$ be a linear order on A. We define the so–called **lexicographical ordering** (**with respect to** $<$) as follows. Put $\vec{x} <_L \vec{y}$ if there exist $\vec{u}$, $\vec{v}$ and $\vec{w}$ as well as a and b such that $\vec{x} = \vec{u}^\frown a^\frown\vec{v}$, $\vec{y} = \vec{u}^\frown b^\frown\vec{w}$ and $a < b$. Notice that $\vec{x} <_L \vec{y}$ can obtain even if $\vec{x}$ is longer than $\vec{y}$. Another important ordering is the following one. Let $\mu(a) := k$ if a is the kth symbol of A in the ordering $<$. Further, put $n := |A|$. For $\vec{x} = x_0x_1\cdots x_{p-1}$ we associate the following number.

$$Z(\vec{x}) := \sum_{i=0}^{p-1}(\mu(x_i)+1)(n+1)^{p-i-1} \tag{1.31}$$

Now put $\vec{x} <_N \vec{y}$ if and only if $Z(\vec{x}) < Z(\vec{y})$. This ordering we call the **numerical ordering**. Notice that both orderings depend on the choice of $<$. We shall illustrate these orderings with $A := \{\mathtt{a},\mathtt{b}\}$ and $\mathtt{a} < \mathtt{b}$. Then the numerical ordering is as follows.

$\vec{x}$	ε	a	b	aa	ab	ba	bb	aaa	aab	aba	...
$Z(\vec{x})$	0	1	2	4	5	7	8	13	14	16	

This ordering is linear. The map sending $i \in \omega$ to the ith element in this sequence is known as the **dyadic representation** of the numbers. In the dyadic representation, 0 is represented by the empty string, 1 by a, 2 by b, 3 by aa and so on. (Actually, if one wants to avoid using the empty string here, one may start with a instead.)

The lexicographical ordering is somewhat more complex. We illustrate it for words with at most four letters.

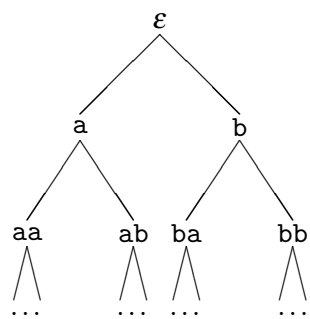


Figure 1. The Tree A^*

ε,	a,	aa,	aaa,	aaaa,	aaab,
aab,	aaba,	aabb,	ab,	aba,	abaa,
abab,	abb,	abba,	abbb,	b,	ba,
baa,	baaa,	baab,	bab,	baba,	babb,
bb,	bba,	bbaa,	bbab,	bbb,	bbba,
bbbb					

In the lexicographical as well as the numerical ordering ε is the smallest element. Now look at the ordered tree based on the set A^*. It is a tree in which every node is n–ary branching (cf. Section 1.4). Then the lexicographical ordering corresponds to the linearization obtained by depth–first search in this tree, while the numerical ordering corresponds to the linearization obtained by breadth–first search (see Section 2.2).

A **monoid** is a triple $\mathfrak{M} = \langle M, 1, \circ \rangle$ where $\circ$ is a binary operation on M and 1 an element such that for all $x, y, z \in M$ the following holds.

(1.32a) $$x \circ 1 = x$$

(1.32b) $$1 \circ x = x$$

(1.32c) $$x \circ (y \circ z) = (x \circ y) \circ z$$

A monoid is therefore an algebra with signature $\Omega\colon 1 \mapsto 0, \cdot \mapsto 2$, which in addition satisfies the above equations. An example is the algebra $\langle 4, 0, \max \rangle$ (recall that $4 = \{0, 1, 2, 3\}$), or $\langle \omega, 0, + \rangle$.

Proposition 1.29 *Let* $\mathfrak{Z}(A) := \langle A^*, \varepsilon, \cdot \rangle$. *Then* $\mathfrak{Z}(A)$ *is a monoid.*

The function which assigns to each string its length is a homomorphism from $\mathfrak{Z}(A)$ onto the monoid $\langle \omega, 0, + \rangle$. It is surjective, since A is always assumed to be nonempty. $\mathfrak{Z}(A)$ are special monoids:

Proposition 1.30 *The monoid $\mathfrak{Z}(A)$ is freely generated by A.*

Proof. Let $\mathfrak{N} = \langle N, 1, \circ \rangle$ be a monoid and $v\colon A \to N$ an arbitrary map. Then we define a map $\overline{v}$ as follows.

$$\begin{aligned} \overline{v}(\varepsilon) &:= 1 \\ \overline{v}(\vec{x}^\frown a) &:= \overline{v}(\vec{x}) \circ v(a) \end{aligned} \tag{1.33}$$

This map is surely well defined. For the defining clauses are mutually exclusive. Now we must show that this map is a homomorphism. To this end, let $\vec{x}$ and $\vec{y}$ be words. We shall show that

$$\overline{v}(\vec{x}^\frown \vec{y}) = \overline{v}(\vec{x}) \circ \overline{v}(\vec{y}) \tag{1.34}$$

This will be established by induction on the length of $\vec{y}$. If it is 0, the claim is evidently true. For we have $\vec{y} = \varepsilon$, and hence $\overline{v}(\vec{x}^\frown \vec{y}) = \overline{v}(\vec{x}) = \overline{v}(\vec{x}) \circ 1 = \overline{v}(\vec{x}) \circ \overline{v}(\vec{y})$. Now let $|\vec{y}| > 0$. Then $\vec{y} = \vec{w}^\frown a$ for some $a \in A$.

$$\begin{aligned} \overline{v}(\vec{x}^\frown \vec{y}) &= \overline{v}(\vec{x}^\frown \vec{w}^\frown a) \\ &= \overline{v}(\vec{x}^\frown \vec{w}) \circ v(a) \\ &= (\overline{v}(\vec{x}) \circ \overline{v}(\vec{w})) \circ v(a) \\ &= \overline{v}(\vec{x}) \circ (\overline{v}(\vec{w}) \circ v(a)) \\ &= \overline{v}(\vec{x}) \circ \overline{v}(\vec{y}) \end{aligned} \tag{1.35}$$

This shows the claim. □

The set A is the only set that generates $\mathfrak{Z}(A)$ freely. For a letter cannot be produced from anything longer than a letter. The empty string is always dispensable, since it occurs anyway in the signature. Hence any generating set must contain A, and since A generates A^* it is the only minimal set that does so. A non–minimal generating set can never freely generate a monoid. For example, let $X = \{\mathtt{a}, \mathtt{b}, \mathtt{bba}\}$. X generates $\mathfrak{Z}(A)$, but it is not minimal. Hence it does not generate $\mathfrak{Z}(A)$ freely. For example, let $v\colon \mathtt{a} \mapsto \mathtt{a}, \mathtt{b} \mapsto \mathtt{b}, \mathtt{bba} \mapsto \mathtt{a}$. Then there is no homomorphism that extends v to A^*. For then on the one hand $\overline{v}(\mathtt{bba}) = \mathtt{a}$, on the other $\overline{v}(\mathtt{bba}) = v(\mathtt{b})^\frown v(\mathtt{b})^\frown v(\mathtt{a}) = \mathtt{bba}$.

The fact that A generates $\mathfrak{Z}(A)$ freely has various noteworthy consequences. First, a homomorphism from $\mathfrak{Z}(A)$ into an arbitrary monoid need only be

fixed on A in order to be defined. Moreover, *any* such map can be extended to a homomorphism into the target monoid. As a particular application we get that every map $v\colon A \to B^*$ can be extended to a homomorphism from $\mathfrak{Z}(A)$ to $\mathfrak{Z}(B)$. Furthermore, we get the following result, which shows that the monoids $\mathfrak{Z}(A)$ are up to isomorphism the only freely generated monoids (allowing infinite alphabets). They reader may note that the proof works for algebras of any signature.

Theorem 1.31 *Let $\mathfrak{M} = \langle M, \circ, 1\rangle$ and $\mathfrak{N} = \langle N, \circ, 1\rangle$ be freely generated monoids. Then either ① or ② obtains.*

① *There is an injective homomorphism $i\colon \mathfrak{M} \rightarrowtail \mathfrak{N}$ and a surjective homomorphism $h\colon \mathfrak{N} \twoheadrightarrow \mathfrak{M}$ such that $h \circ i = 1_M$.*

② *There exists an injective homomorphism $i\colon \mathfrak{N} \rightarrowtail \mathfrak{M}$ and a surjective homomorphism $h\colon \mathfrak{M} \twoheadrightarrow \mathfrak{N}$ such that $h \circ i = 1_N$.*

Proof. Let $\mathfrak{M}$ be freely generated by X, $\mathfrak{N}$ freely generated by Y. Then either $|X| \leq |Y|$ or $|Y| \leq |X|$. Without loss of generality we assume the first. Then there is an injective map $p\colon X \rightarrowtail Y$ and a surjective map $q\colon Y \twoheadrightarrow X$ such that $q \circ p = 1_X$. Since X generates $\mathfrak{M}$ freely, there is a homomorphism $\overline{p}\colon \mathfrak{M} \to \mathfrak{N}$ with $\overline{p} \restriction X = p$. Likewise, there is a homomorphism $\overline{q}\colon \mathfrak{N} \to \mathfrak{M}$ such that $\overline{q} \restriction Y = q$, since $\mathfrak{N}$ is freely generated by Y. The restriction of $\overline{q} \circ \overline{p}$ to X is the identity. (For if $x \in X$ then $\overline{q} \circ \overline{p}(x) = \overline{q}(p(x)) = q(p(x)) = x$.) Since X freely generates $\mathfrak{M}$, there is only one homomorphism which extends 1_X on $\mathfrak{M}$ and this is the identity. Hence $\overline{q} \circ \overline{p} = 1_M$. It immediately follows that $\overline{q}$ is surjective and $\overline{p}$ injective. Hence ① obtains. If $|Y| \leq |X|$ holds, ② is shown in the same way. □

Theorem 1.32 *In $\mathfrak{Z}(A)$ the following cancellation laws hold.*

① *If $\vec{x}^\frown\vec{u} = \vec{y}^\frown\vec{u}$, then $\vec{x} = \vec{y}$.*

② *If $\vec{u}^\frown\vec{x} = \vec{u}^\frown\vec{y}$, then $\vec{x} = \vec{y}$.*

$\vec{x}^T$ is defined as follows.

$$\left(\prod_{i<n} x_i\right)^T := \prod_{i<n} x_{n-1-i} \tag{1.36}$$

$\vec{x}^T$ is called the **mirror string** of $\vec{x}$. It is easy to see that $(\vec{x}^T)^T = \vec{x}$. The reader is asked to convince himself that the map $\vec{x} \mapsto \vec{x}^T$ is *not* a homomorphism if $|A| > 1$.

Definition 1.33 *Let $\vec{x}, \vec{y} \in A^*$. Then $\vec{x}$ is a* ***prefix of*** *$\vec{y}$ if $\vec{y} = \vec{x}^\frown\vec{u}$ for some $\vec{u} \in A^*$. $\vec{x}$ is called a* ***postfix*** *or* ***suffix of*** *$\vec{y}$ if $\vec{y} = \vec{u}^\frown\vec{x}$ for some $\vec{u} \in A^*$. $\vec{x}$ is called a* ***substring of*** *$\vec{y}$ if $\vec{y} = \vec{u}^\frown\vec{x}^\frown\vec{v}$ for some $\vec{u}, \vec{v} \in A^*$.*

It is easy to see that $\vec{x}$ is a prefix of $\vec{y}$ exactly if $\vec{x}^T$ is a postfix of $\vec{y}^T$. Notice that a given string can have several occurrences in another string. For example, `aa` occurs four times is `aaaaa`. The occurrences are in addition not always disjoint. An occurrence of $\vec{x}$ in $\vec{y}$ can be defined in several ways. We may for example assign *positions* to each letters. In a string $x_0x_1 \ldots x_{n-1}$ the numbers $< n+1$ are called **positions**. The positions are actually thought of as the spaces between the letters. The ith letter, x_i, occurs between the position i and the position $i+1$. The substring $\prod_{i \leq j < k} x_i$ occurs between the positions i and k. The reason for doing it this way is that it allows us to define occurrences of the empty string as well. For each i, there is an occurrence of ε between position i and position i. We may interpret positions as time points in between which certain events take place, here the utterance of a given sound. Another definition of an occurrence is via the context in which the substring occurs.

Definition 1.34 *A* ***context*** *is a pair $C = \langle \vec{y}, \vec{z} \rangle$ of strings. The* ***substitution of*** *$\vec{x}$* ***into*** *C, in symbols $C(\vec{x})$, is defined to be the string $\vec{y}^\frown\vec{x}^\frown\vec{z}$. We say that $\vec{x}$* ***occurs in*** *$\vec{v}$* ***in the context*** *C if $\vec{v} = C(\vec{x})$. Every occurrence of $\vec{x}$ in a string $\vec{v}$ is uniquely defined by its context. We call C a* ***substring occurrence of*** *$\vec{x}$* ***in*** *$\vec{v}$.*

Actually, given $\vec{x}$ and $\vec{v}$, only one half of the context defines the other. However, as will become clear, contexts defined in this way allow for rather concise statements of facts in many cases. Now consider two substring occurrences C, D in a given word $\vec{z}$. Then there are various ways in which the substrings may be related with respect to each other.

Definition 1.35 *Let $C = \langle \vec{u}_1, \vec{u}_2 \rangle$ and $D = \langle \vec{v}_1, \vec{v}_2 \rangle$ be occurrences in $\vec{z}$ of the strings $\vec{x}$ and $\vec{y}$, respectively. We say that C* ***precedes*** *D if $\vec{u}_1^\frown\vec{x}$ is a prefix of $\vec{v}_1$. C and D* ***overlap*** *if C does not precede D and D does not precede C. C* ***is contained in*** *D if $\vec{v}_1$ is a prefix of $\vec{u}_1$ and $\vec{v}_2$ is a suffix of $\vec{u}_2$.*

Notice that if $\vec{x}$ is a substring of $\vec{y}$ then every occurrence of $\vec{y}$ contains an occurrence of $\vec{x}$; but not every occurrence of $\vec{x}$ is contained in a given occurrence of $\vec{y}$.

Definition 1.36 *A* ***(string) language over the alphabet*** *A is a subset of A^*.*

This definition admits that $L = \varnothing$ and that $L = A^*$. Moreover, $\varepsilon \in L$ also may occur. The admission of ε is often done for technical reasons (like the introduction of a zero).

Theorem 1.37 *Suppose A is not empty, and $|A| \leq \aleph_0$. Then there are exactly $2^{\aleph_0}$ languages.*

Proof. This is a standard counting argument. We establish that $|A^*| = \aleph_0$. The claim then follows since there are as many languages as there are subsets of $\aleph_0$, namely $2^{\aleph_0}$. If A is finite, we can enumerate A^* by enumerating the strings of length 0, the strings of length 1, the strings of length 2, and so on. If A is infinite, we have to use cardinal arithmetic: the set of strings of length k of any finite k is countable, and A^* is therefore the countable union of countable sets, again countable. □

One can prove the previous result directly using the following argument. (The argument works even when C is countably infinite.)

Theorem 1.38 *Let $C = \{c_i : i < p\}$, $p > 2$, be an arbitrary alphabet and $A = \{\mathtt{a}, \mathtt{b}\}$. Further, let $\overline{v}$ be the homomorphic extension of $v\colon c_i \mapsto \mathtt{a}^i{}^\frown\mathtt{b}$. The map $S \mapsto \overline{v}[S]\colon \wp(C^*) \to \wp(A^*)$ defined by $V(S) = \overline{v}[S]$ is a bijection between $\wp(C^*)$ and those languages which are contained in the direct image of $\overline{v}$.*

The proof is an exercise. The set of all languages over A is closed under $\cap$, $\cup$, and $-$, the relative complement with respect to A^*. Furthermore, we can define the following operations on languages.

(1.37a) $\quad L \cdot M := \{\vec{x}{}^\frown\vec{y} : \vec{x} \in L, \vec{y} \in M\}$

(1.37b) $\quad L^0 := \{\varepsilon\}$

(1.37c) $\quad L^{n+1} := L^n \cdot L$

(1.37d) $\quad L^* := \bigcup_{n \in \omega} L^n$

(1.37e) $\quad L^+ := \bigcup_{0 < n \in \omega} L^n$

(1.37f) $\quad L/M := \{\vec{y} \in A^* : (\exists \vec{x} \in M)(\vec{y}{}^\frown\vec{x} \in L)\}$

(1.37g) $\quad M\backslash L := \{\vec{y} \in A^* : (\exists \vec{x} \in M)(\vec{x}{}^\frown\vec{y} \in L)\}$

* is called the **Kleene star**. For example, L/A^* is the set of all strings which can be extended to members of L; this is exactly the set of prefixes of members of L. We call this set the **prefix closure** of L, in symbols L^P. Analogously,

$L^S := A^*\backslash L$ is the **suffix** or **postfix closure** of L. It follows that $(L^P)^S$ is nothing but the substring closure of L.

In what is to follow, we shall often encounter string languages with a special distinguished symbol, the **blank**, typically written $\Box$. Then we use the abbreviation

$$\vec{x} \diamond \vec{y} := \vec{x}^\frown \Box^\frown \vec{y} \qquad\qquad L \diamond M := \{\vec{x} \diamond \vec{y} : \vec{x} \in L, \vec{y} \in M\} \tag{1.38}$$

Let L be a language over A, $C = \langle \vec{x}, \vec{y} \rangle$ a context and $\vec{u}$ a string. We say that C **accepts** $\vec{u}$ **in** L, and write $\vec{u} \dashv_L C$, if $C(\vec{u}) \in L$. The triple $\langle A^*, A^* \times A^*, \dashv_L \rangle$ is a context in the sense of the previous section. Let $M \subseteq A^*$ and $P \subseteq A^* \times A^*$. Then denote by $C_L(M)$ the set of all C which accept all strings from M in L (intent); and denote by $Z_L(P)$ the set of all strings which are accepted by all contexts from P in L (extent). We call M (L–)**closed** if $M = Z_L(C_L(M))$. The closed sets form the so–called **distribution classes** of strings in a language. $Z_L(C_L(M))$ is called the **Sestier–closure** of M and the map $S_L \colon M \mapsto Z_L(C_L(M))$ the **Sestier–operator**. From Proposition 1.24 we immediately get this result.

Proposition 1.39 *The Sestier–operator is a closure operator.*

For various reasons, identifying terms with strings that represent them is a dangerous affair. As is well–known, conventions for writing down terms can be misleading, since they might be ambiguous. Therefore we defined the term as an entity in itself. The string by which we denote the term is only as a representative of that term.

Definition 1.40 *Let Ω be a signature. A **representation of terms (by means of strings over** A**)** is a relation $R \subseteq \mathrm{Tm}_\Omega \times A^*$ such that for each term t there exists a string $\vec{x}$ with $\langle t, \vec{x} \rangle \in R$. $\vec{x}$ is called a **representative** or **representing string** of t **with respect to** R. $\vec{x}$ is called **unambiguous** if from $\langle t, \vec{x} \rangle, \langle u, \vec{x} \rangle \in R$ it follows that $t = u$. R is called **unique** or **uniquely readable** if every $\vec{x} \in A^*$ is unambiguous.*

R is uniquely readable iff it is an injective function from Tm_Ω to A^* (and therefore its converse a partial injective function). We leave it to the reader to verify that the representation defined in the previous section is actually uniquely readable. This is not self evident. It could be that a term possesses several representing strings. Our usual way of denoting terms is in fact not uniquely readable. For example, one writes $2+3+4$ even though this could

be a representative of the term $+(+(2,3),4)$ or of the term $+(2,+(3,4))$. This hardly matters, since the two terms denote the same number, but nevertheless they are different terms.

There are many more conventions for writing down terms. We give a few examples. (a) A binary symbol is typically written in between its arguments (this is called the **infix notation**). So, we do not write `+(2,3)` but `(2+3)`. (b) Outermost brackets may be omitted: `(2+3)` denotes the same term as `2+3`. (c) The multiplication sign binds stronger than `+`. So, the following strings all denote the same term.

(1.39) `(2+(3*5))` `2+(3*5)` `(2+3*5)` `2+3*5`

In logic, it was customary to use dots in place of brackets. In this notation, `p∧q.→.p` means the same as the more common `(p∧q)→p`. The dots are placed to the left or right (sometimes both) of the operation sign. Ambiguity is resolved by using more than one dot, for example '`:`'. (See [**?**] on this notation.) Also, let $\circ$ be a binary operation symbol, written in infix notation. Suppose that ℓ defines a string for every term in the following way.

$$
\begin{aligned}
\ell(x) &:= x && x \text{ basic} \\
\ell(\circ(x,y)) &:= \ell(x) \circ y && y \text{ basic} \\
\ell(\circ(x,t)) &:= \ell(x) \circ (\ell(t)) && t \text{ complex}
\end{aligned}
\tag{1.40}
$$

If $\ell(t)$ represents t, we say that $\circ$ is **left–associative**. If on the other hand $\rho(t)$ represents the term t, $\circ$ is said to be **right–associative**.

$$
\begin{aligned}
\rho(y) &:= x && y \text{ basic} \\
\rho(\circ(x,y)) &:= x \circ \rho(y) && x \text{ basic} \\
\ell(\circ(t,y)) &:= (\rho(t)) \circ \rho(y) && t \text{ complex}
\end{aligned}
\tag{1.41}
$$

Since the string `(2+3)*5` represents a different term than `2+3*5` (and both have a different value) the brackets cannot be omitted. That we can do without brackets is an insight we owe to the Polish logician Jan Łukasiewicz. In his notation, which is also called **Polish Notation** (**PN**), the function symbol is always placed in front of its arguments. Alternatively, the function symbol may be consistently placed behind its arguments (this is the so–called **Reverse Polish Notation**, **RPN**). There are some calculators (in addition to the programming language FORTH) which have implemented RPN. In place of

the (optional) brackets there is a key called '`enter`'. It is needed to separate two successive operands. For in RPN, the two arguments of a function follow each other immediately. If nothing is put in between them, both the terms $+(13,5)$ and $+(1,35)$ would both be written `135+`. To prevent this, '`enter`' is used to separate the first from the second input string. You therefore need to enter into the computer `13`$\boxed{\texttt{enter}}$`5+`. (Here, the box is the usual way in computer handbooks to turn a sequence into a 'key'. In Chapter 3 we shall deal again with the problem of writing down numbers.) Notice that in practice (i.e. as far as the tacit conventions go) the choice between Polish and Reverse Polish Notation only affects the position of the function symbol, and not the way in which arguments are placed with respect to each other. For example, suppose there is a key $\boxed{\texttt{exp}}$ for the exponential function. Then to get the result of 2^3, you enter `2`$\boxed{\texttt{enter}}$`3`$\boxed{\texttt{exp}}$ on a machine using RPN and $\boxed{\texttt{exp}}$`2`$\boxed{\texttt{enter}}$`3=` on a machine using PN. Hence, the relative order between base (`2`) and exponent (`3`) remains. (Notice incidentally the need for typing in `=` or something else that indicates the end of the second operand in PN!) This effect is also noted in natural languages: the subject precedes the object in the overwhelming majority of languages irrespective of the place of the verb. The mirror image of an VSO language is an SOV language, not OSV.

Now we shall show that Polish Notation is uniquely readable. Let F be a set of symbols and Ω a signature over F. Each symbol $f \in F$ is assigned an arity $\Omega(f)$. Next, we define a set of strings over F, which we assign to the various terms of Tm_Ω. PN_Ω is the smallest set M of strings over F for which the following holds.

$$\begin{gathered}\text{For all } f \in F \text{ and for all } \vec{x}_i \in M,\ i < \Omega(f):\\ f^\frown \vec{x}_0{}^\frown \cdots {}^\frown \vec{x}_{\Omega(f)-1} \in M.\end{gathered}$$

(Notice the special case $n = 0$. Further, notice that no special treatment is needed for variables, by the remarks of the preceding section.) This defines the set PN_Ω, members of which are called **well–formed strings**. Next we shall define which string represents which term. The string 'f', $\Omega(f) = 0$, represents the term 'f'. If $\vec{x}_i$ represents t_i, $i < \Omega(f)$, then $f^\frown \vec{x}_0{}^\frown \cdots {}^\frown \vec{x}_{\Omega(f)-1}$ represents $f(t_0, \ldots, t_{\Omega(f)-1})$. We shall now show that this relation is bijective. (A different proof than the one used here can be found in Section 2.4, proof of Theorem 2.61.) Here we use an important principle, namely induction over the length of the string. The following is for example proved by induction on $|\vec{x}|$.

① No proper prefix of $|\vec{x}|$ is a well–formed string.

② If $\vec{x}$ is a well–formed string then $\vec{x}$ has length at least 1 and the following holds.

(a) If $|\vec{x}| = 1$, then $\vec{x} = f$ for some $f \in F$ with $\Omega(f) = 0$.

(b) If $|\vec{x}| > 1$, then there are f and $\vec{y}$ such that $\vec{x} = f^\frown\vec{y}$, and $\vec{y}$ is the concatenation of exactly $\Omega(f)$ many uniquely defined well–formed strings.

The proof is as follows. Let t and u be terms represented by $\vec{x}$. Let $|\vec{x}| = 1$. Then $t = u = f$, for some $f \in F$ with $\Omega(f) = 0$. A proper prefix is the empty string, which is clearly not well formed. Now for the induction step. Let $\vec{x}$ have length at least 2. Then there is an $f \in F$ and a sequence $\vec{y}_i$, $i < \Omega(f)$, of well–formed strings such that

$$\vec{x} = f^\frown\vec{y}_0{}^\frown\cdots{}^\frown\vec{y}_{\Omega(f)-1} \tag{1.42}$$

Therefore for each $i < \Omega(f)$ there is a term u_i represented by $\vec{y}_i$. By ②, the u_i are uniquely determined by the $\vec{y}_i$. Furthermore, the symbol f is uniquely determined, too. Now let $\vec{z}_i$, $i < \Omega(f)$, be well–formed strings with

$$\vec{x} = f^\frown\vec{z}_0{}^\frown\cdots{}^\frown\vec{z}_{\Omega(f)-1} \tag{1.43}$$

Then $\vec{y}_0 = \vec{z}_0$. For no proper prefix of $\vec{z}_0$ is a well–formed term, and no proper prefix of $\vec{y}_0$ is a term. But they are prefixes of each other, so they cannot be proper prefixes of each other, that is to say, they are equal. If $\Omega(f) = 1$, we are done. Otherwise we carry on in the same way, establishing by the same argument that $\vec{y}_1 = \vec{z}_1$, $\vec{y}_2 = \vec{z}_2$, and so on. The fragmentation of the string in $\Omega(f)$ many well–formed strings is therefore unique. By inductive hypothesis, the individual strings uniquely represent the terms u_i. So, $\vec{x}$ uniquely represents the term $f(\vec{u})$. This shows ②.

Finally, we shall establish ①. Look again at the decomposition (1.42). If $\vec{u}$ is a well–formed prefix, then $\vec{u} \neq \varepsilon$. Hence $\vec{u} = f^\frown\vec{v}$ for some $\vec{v}$ which can be decomposed into $\Omega(f)$ many well–formed strings $\vec{w}_i$. As before we shall argue that $\vec{w}_i = \vec{x}_i$ for every $i < \Omega(f)$. Hence $\vec{u} = \vec{x}$, which shows that no proper prefix of $\vec{x}$ is well–formed.

Notes on this section. Throughout this book the policy is to regard any linguistic object as a string. Strings are considered the fundamental structures. This in itself is no philosophical commitment, just a matter of convenience.

Moreover, when we refer to sentences qua material objects (signifiers) we take them to be strings over the Latin alphabet. This again is only a matter of convenience. Formal language theory very often treats words rather than letters as units. If one does so, their composite nature has to be ignored. Yet, while most arguments can still be performed (since a transducer can be used to switch between these representations), some subtleties can get lost in this abstraction. We should also point out that since alphabets must be finite, there can be no infinite set of variables as a primitive set of letters, as is often assumed in logic.

Exercise 9. Prove Theorem 1.38.

Exercise 10. (The 'Typewriter Model'.) Fix an alphabet A. For each $a \in A$ assume a unary symbol $\mathtt{s}_a$. Finally, let $\mathtt{0}$ be a zeroary symbol. This defines the signature Ψ. Define a map $t : \mathrm{Tm}_\Psi \to A^*$ as follows. $\tau(\mathtt{0}) := \varepsilon$, and $\tau(\mathtt{s}_a(s)) := \tau(s)^\frown a$. Show that τ is bijective. Further, show that there is no term u over Ψ such that $\tau(u(x,y)) = \tau(x)^\frown\tau(y)$, and not even a term $v_{\vec{x}}(y)$ such that $\tau(v_{\vec{x}}(y)) = \vec{x}^\frown\tau(y)$, for any given $\vec{x} \in A^+$. On the other hand there *does* exist a $w_{\vec{y}}$ such that $\tau(w_{\vec{y}}(x)) = \tau(x)^\frown\vec{y}$ for any given $\vec{y} \in A^*$.

Exercise 11. Put $Z^*(\vec{x}) := \sum_{i<p} \mu(x_i) n^{p-i-1}$. Now put $\vec{x} <_{N^*} \vec{y}$ if and only if $Z^*(\vec{x}) < Z^*(\vec{y})$. Show that $<_{N^*}$ is transitive and irreflexive, but not total.

Exercise 12. Show that the postfix relation is a partial ordering, likewise the prefix and the subword relation. Show that the subword relation is the transitive closure of the union of the postfix relation with the prefix relation.

Exercise 13. Let F, X and $\{\mathtt{(},\mathtt{)}\}$ be three pairwise disjoint sets, Ω a signature over F. We define the following function from Ω–terms into strings over $F \cup X \cup \{\mathtt{(},\mathtt{)}\}$:

$$\begin{aligned} x^+ &:= x \\ f(t_0,\dots,t_{\Omega(f)-1})^+ &:= f^\frown\mathtt{(}^\frown t_0^{+\frown}\cdots{}^\frown t_{\Omega(f)-1}^{+}{}^\frown\mathtt{)} \end{aligned} \tag{1.44}$$

(To be clear: we represent terms by the string that we have used in Section 1.1 already.) Prove the unique readability of this notation. Notice that this does not already follow from the fact that we have chosen this notation to begin with. (We might just have been mistaken ...)

Exercise 14. Give an exact upper bound on the number of prefixes (postfixes) of a given string of length n, n a natural number. Also give a bound for the number of subwords. What can you say about the exactness of these bounds

in individual cases?

Exercise 15. Let $L, M \subseteq A^*$. Define

(1.45a) $L//M := \{\vec{y} : (\forall \vec{x} \in M)(\vec{y}^\frown \vec{x} \in L)\}$

(1.45b) $M\backslash\backslash L := \{\vec{y} : (\forall \vec{x} \in M)(\vec{x}^\frown \vec{y} \in L)\}$

Show the following for all $L, M, N \subseteq A^*$:

(1.46) $M \subseteq L\backslash\backslash N \quad \Leftrightarrow \quad L \cdot M \subseteq N \quad \Leftrightarrow \quad L \subseteq N//M$

Exercise 16. Show that not all equivalences are valid if in place of $\backslash\backslash$ and $//$ we choose $\backslash$ and $/$. Which implications remain valid, though?

3. Fundamentals of Linguistics

In this section we shall say some words about our conception of language and introduce some linguistic terminology. Since we cannot define all the linguistic terms we are using, this section is more or less meant to get those readers acquainted with the basic linguistic terminology who wish to read the book without going through an introduction into linguistics proper. (However, it is recommended to have such a book at hand.)

A central tool in linguistics is that of postulating abstract units and hierarchization. Language is thought to be more than a mere relation between sounds and meanings. In between the two realms we find a rather rich architecture that hardly exists in formal languages. This architecture is most clearly articulated in [**?**] and also [**?**]. Even though linguists might disagree with many details, this basic architecture is assumed even in most current linguistic theories. We shall outline what we think is minimal consensus. Language is organized in four levels or layers, which are also called **strata**, see Figure 2: the **phonological stratum**, the **morphological stratum**, the **syntactic stratum** and the **semantical stratum**. Each stratum possesses elementary units and rules of combination. The phonological stratum and the morphological stratum are adjacent, the morphological stratum and the syntactic stratum are adjacent, and the syntactic stratum and the semantic stratum are adjacent. Adjacent strata are interconnected by so–called **rules of realization**. On the phonological stratum we find the mere representation of the

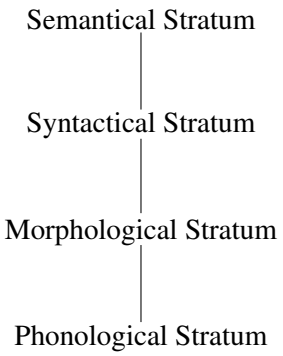


Figure 2. The Strata of Language

utterance in its phonetic and phonological form. The elementary units are the **phones**. An utterance is composed from phones (more or less) by concatenation. The terms 'phone', 'syllable', 'accent' and 'tone' refer to this stratum. In the morphological stratum we find the elementary signs of the language (see Section 3.1), which are called **morphs**. These are defined to be the smallest units that carry meaning, although the definition of 'smallest' may be difficult to give. They are different from words. The word `sees` is a word, but it is the combination of two morphs, the root `see` and the ending of the third person singular present, `s`. The units of the syntactical stratum are called **lexes**, and they more or less are the same as words. The units of the semantical stratum are the **semes**.

On each stratum we distinguish concrete from abstract units. The concrete forms represent *substance*, while the abstract ones represent the *form* only. While the relationship between these two levels is far from easy, we will simplify the matter as follows. The abstract units are seen as sets of concrete ones. The abstraction is done in such a way that the concrete member of each class that appears in a construction is defined by its context, and that substitution of another member results simply in a non well–formed unit (or else in a virtually identical one). This definition is deliberately vague; it is actually hard to make precise. The interested reader is referred to the excellent [**?**] for a thorough discussion of the structural method. We shall also return to this question in Section 6.3. The abstract counterpart of a phone is a **phoneme**. A phoneme is simply a set of phones. The sounds of a single language are a

Table 1. German Plural Morphs

singular	plural	
`Wagen`	`Wagen`	'car'
`Auto`	`Autos`	'car'
`Bus`	`Busse`	'bus'
`Licht`	`Lichter`	'light'
`Vater`	`Väter`	'father'
`Nacht`	`Nächte`	'night'

subset of the entire space of human sounds, partitioned into phonemes. This is to say that two distinct phonemes of a languages are disjoint. We shall deal with the relationship between phones and phonemes in Section 6.3. We use the following notation. We enclose phonemes in slashes while square brackets are used to name phones. So, if [p] denotes a phone then /p/ is a phoneme containing [p]. (Clearly, there are infinitely many sounds that may be called [p], but we pick just one of them.) An index is used to make clear which language the phoneme belongs to. For phonemes are strictly language bound. It makes little sense to compare phonemes across languages. Languages cut up the sound continuum in a different way. For example, let [p] and [p^h] be two distinct phones, where [p] is a phone corresponding to the letter `p` in `spit`, [p^h] a phone corresponding to the letter `p` in `put`. Hindi distinguishes these two phones as instantiations of different phonemes: $/\mathrm{p}/_H \cap /\mathrm{p}^h/_H = \varnothing$. English does not. So, $/\mathrm{p}/_E = /\mathrm{p}^h/_E$. Moreover, the context determines whether what is written `p` is pronounced either as [p] or as [p^h]. Actually, in English there is no context in which both will occur. Finally, French does not even have the sound [p^h]. We give another example. The combination of the letters `ch` is pronounced in two noticeably distinct ways in German. After [ɪ], it sounds like [ç], for example in `Licht` [lɪçt], but after [a] it sounds like [x] as in `Nacht` [naxt]; the choice between these two variants is conditioned solely by the preceding vowel. It is therefore assumed that German does not possess two phonemes but only one, written `ch`, which is pronounced in these two ways depending on the context.

In the same way one assumes that German has only one plural **morpheme** even though there is a fair number of individual plural morphs. Table 1 shows some possibilities of forming the plural in German. The plural can be ex-

pressed either by no change, or by adding an s–suffix, an e–suffix (the reduplication of s in Busse is a phonological effect and needs no accounting for in the morphology), an er–suffix, or by umlaut or a combination of umlaut together with an e–suffix. (**Umlaut** is another name for the following change of vowels: a becomes ä, o becomes ö, and u becomes ü. All other vowels remain the same. Umlaut is triggered by certain inflectional or derivational suffixes.) All these are clearly different morphs. But they belong to the same morpheme. We therefore call them **allomorphs** of the plural morpheme. The differentiation into strata allows to abstract away from irregularities. Moving up one stratum, the different members of an abstraction class are not distinguished. The different plural morphs for example, are defined as sequences of phonemes, not of phones. To decide which phone is to be inserted is the job of the phonological stratum. Likewise, the word Lichter is 'known' to the syntactical stratum only as a plural nominative noun. That it consists of the root morph Licht together with the morph er rather than any other plural morph is not visible in the syntactic stratum. The difference between concrete and abstract carries over in each stratum in the distinction between a **surface** and a **deep** sub–stratum. The morphotaxis has at deep level only the root Licht and the plural morpheme. At the surface, the latter gets realized as er. The step from deep to surface can be quite complex. For example, the plural Nächte of Nacht is formed by changing the root vowel and adding the suffix e. (Which of the vowels of the root are subject to umlauted must be determined by the phonological stratum. For example, the plural of Altar 'altar' is Altäre and not Ältare or Ältäre!) As we have already said, on the so–called deep morphological (sub–)stratum we find only the combination of two morphemes, the morpheme Nacht and the plural morpheme. On the syntactical stratum (deep or surface) nothing of that decomposition is visible. We have one lex(eme), Nächte. On the phonological stratum we find a sequence of 5 (!) phonemes, which in writing correspond to n, ä, ch, t and e. This is the deep phonological representation. On the surface, we find the allophone [ç] for the phoneme (written as) ch.

In Section 3.1 we shall propose an approach to language by means of signs. This approach distinguishes only 3 dimensions: a sign has a *realization*, it has a *combinatorics* and it has a *meaning*. While the meaning is uniquely identifiable to belong to the semantic stratum, for the other two this is not clear. The combinatorics may be seen as belonging to the syntactical stratum. The realization of a sign, finally, could be spelled out either as a sequence of phonemes, as a sequence of morphemes or as a sequence of lexemes. Each

of these choices is legitimate and yields interesting insights. However, notice that choosing sequences of morphemes or lexemes is somewhat incomplete since it further requires an additional algorithm that realizes these sequences in writing or speaking.

Language is not only spoken, it is also written. However, one must distinguish between letters and sounds. The difference between them is foremost a physical one. They use a different *channel*. A **channel** is a physical medium in which the message is manifested. Language manifests itself first and foremost acoustically, even though a lot of communication is done in writing. We principally learn a language by hearing and speaking it. Mastery of writing is achieved only after we are fully fluent just speaking the language, even though our views of language are to a large extent shaped by our writing culture (see [**?**] on that). (Sign languages form an exception that will not be dealt with here.) Each channel allows — by its mere physical properties — a different means of combination. A piece of paper is a two dimensional thing, and we are not forced to write down symbols linearly, as we are with acoustical signals. Think for example of the fact that Chinese characters are composite entities which contain parts in them. These are combined typically by juxtaposition, but characters are aligned vertically. Moreover, the graphical composition internally to a sign is of no relevance for the actual sound that goes with it. To take another example, Hindi is written in a syllabic script, which is called **Devanagari**. Each simple consonantal letter denotes a consonant plus `a`. Vowel letters may be added to these in case the vowel is different from `a`. (There are special characters for word initial vowels.) Finally, to denote consonantal clusters, the consonantal characters are melted into each other in a particular way. There is only a finite number of consonantal clusters and the way the consonants are melted is fixed. The individual consonants are usually recognizable from the graphical complex. In typesetting there is a similar phenomenon known as **ligature**. The graphemes `f` and `i` melt into one when the first is before the second: 'fi'. (Typewriters have no ligature, for obvious reasons. So you get `fi`.) Also, in mathematics the possibilities of the graphical channel are widely used. We use indices, superscripts, subscripts, underlining, arrows and so on. Many diagrams are therefore not so easy to linearize. (For example, $\widehat{x}$ is spelled out as `x hat`, $\overline{x}$ as `x bar`.) Sign languages also make use of the three–dimensional space, which proves to require different perceptual skills than spoken language.

While the acoustic manifestation of language is in some sense essential for human language, its written manifestation is typically secondary, not only

for the individual human being, as said above, but also from a cultural historic point of view. The sounds of the language and the pronunciation of words is something that comes into existence naturally, and they can hardly be fixed or determined arbitrarily. Attempts to stop language from changing are simply doomed to failure. Writing systems, on the other hand, are cultural products, and subject to sometimes severe regimentation. The effect is that writing systems show much greater variety across languages than sound systems. The number of primitive letters varies between some two dozen and a few thousand. This is so since some languages have letters for sounds (more or less) like Finnish (English is a difficult case), others have letters for syllables (Hindi, written in Devanagari) and yet others have letters for words (Chinese). It may be objected that in Chinese a character always stands for a syllable, but words may consist of several syllables, hence of several characters. Nevertheless, the difference with Devanagari is clear. The latter shows you how the word sounds like, the former does not, unless you know character by character how it is pronounced. If you were to introduce a new syllable into Chinese you would have to create a new character, but not so in Devanagari. But all this has to be taken with care. Although French uses the Latin alphabet it becomes quite similar to Chinese. You may still know how to pronounce a word that you see written down, but from hearing it you are left in the dark as to how to spell it. For example, the following words are pronounced completely alike: `au`, `haut`, `eau`, `eaux`; similarly `vers`, `vert`, `verre`, `verres`.

In what is to follow, language will be written language. This is the current practice in such books as this one; but it requires comment. We are using the so–called Latin alphabet. It is used in almost all European countries, while each country typically uses a different set of symbols. The difference is slight, but needs accounting for (for example, when you wish to produce keyboards or design fonts). Finnish, Hungarian and German, for example, use ä, ö and ü. The letter ß is used in the German alphabet (but not in Switzerland). In French, one uses ç, also accents, and so on. The resource of single characters, which we call **letters**, is for the European languages somewhere between 60 and 100. Besides each letter, both in upper and lower case, we also have the punctuation marks and some extra symbols, not to forget the ubiquitous blank. Notice, however, that not all languages have a blank (Chinese is a case in point, and also the Romans did not use any blanks). On the other hand, one blank is not distinct from two. We can either decide to disallow two blanks in a row, or postulate that they are equal to one. (So, the structure we look at

is $\mathfrak{Z}(A)/\{\square = \square^\frown\square\}$.) A final problem area to be considered is our requirement that sign composition is additive. This means that every change that occurs is underlyingly viewed as adding something that was not there. This can yield awkward results. While the fact that German umlaut is graphically speaking just the addition of two dots (`a` becomes `ä`, `o` becomes `ö`, `u` becomes `ü`), the change of a lower case letter to an upper case letter cannot be so analysed. This requires another level of representation, one at which the process is completely additive. This is harmless, if we only change the material aspect (substance) rather than the form.

The counterpart of a letter in the spoken languages is the phoneme. Every language utterance can be analyzed into a sequence of phonemes (plus some residue about which we will speak briefly below). There is generally no biunique correspondence between phonemes and letters. The connection between the visible and the audible shape of language is everything but predictable or unambiguous in either direction. English is a perfect example. There is hardly any letter that can unequivocally be related to a phoneme. For example, the letter `g` represents in many cases the phoneme [g] unless it is followed by `h`, in which case the two typically together represent a sound that can be zero (as in `sought` [sɔ:t]), or `f` (as in `laughter` ([la:ftə]). To add to the confusion, the letters represent different sets of phones in different languages. (Note that it makes no sense to speak of the same *phoneme* in two different languages, as phonemes are abstractions that are formed within a single language.) The letter `u` has many different manifestations in English, German and French that are hardly compatible. This has prompted the invention of an international standard, the so–called **International Phonetic Alphabet** (**IPA**, see [**?**]). Ideally, every sound of a given language can be uniquely transcribed into IPA such that anyone who is not acquainted with the language can reproduce the utterances correctly. The transcription of a word into this alphabet therefore changes whenever its sound manifestation changes, irrespective of the spelling norm. Unfortunately, the transcription must ultimately remain inconsequential, because even in the IPA letters stand for sets of phones, but in every language the width of a phoneme (= the set of phones it contains) is different. For example, if (English) $/\mathrm{p}/_E$ contains both (Hindi) $/\mathrm{p}/_H$ and $/\mathrm{p}^{\mathrm{h}}/_H$, we either have to represent `p` in English by (at least) two letters or else give up the exact correspondence.

The carriers of meaning are however not the sounds or letters (there is simply not enough of them); it is certain sequences thereof. Sequences of letters that are not separated by a blank or a punctuation mark other than '`-`'

are called **words**. Words are units which can be analyzed further, for example into letters, but for the most part we shall treat them as units. This is the reason why the alphabet A in the technical sense will often *not* be the alphabet in the sense of 'stock of letters' but in the sense of 'stock of words'. However, since most languages have infinitely many words (due to compounding), and since the alphabet A must be finite, some care must be exercised in choosing the alphabet. Typically, it will exclude the compound words, but it will have to include all idioms.

We have analyzed words into sequences of letters or sounds, and sentences into sequences of words. This implies that sentences and words can always be so analyzed. This is what we shall assume throughout this book. The individual occurrences of sounds (letters) are called **segments**. For example, the (occurrences of the) letters `n`, `o`, and `t` are the segments of `not`. The fact that words can be segmented is called **segmentability property**. At closer look it turns out that segmentability is an idealization. For example, a question differs from an assertion in its **intonation contour**, which is the rise and fall of the pitch during the utterance. The contour shows distribution over the whole sentence but follows specific rules. It is of course different in different languages. (Falling pitch at the end of a sentence, for example, may accompany questions in English, but not in German.) Because of its nature, intonation contour is called a **suprasegmental feature**. There are more, for example emphasis. Segmentability differs also with the channel. In writing, a question is marked by a segmental feature (the question mark), but emphasis is not. Emphasis is typically marked by underlining or italics. For example, if we want to emphasize the word 'board', we write <u>`board`</u> or *board*. As can be seen, every letter is underlined or set in italics, but underlining or italics is usually not something that is meant to emphasize those letters that are marked by it; rather, it marks emphasis of the entire word that is composed from them. We could have used a segmental symbol, just like quotes, but the fact of the matter is that we do not. Disregarding this, language typically is segmentable.

However, even if this is true, the idea that the morphemes of the language are sequences of letters is largely mistaken. To give an extreme example, the plural is formed in Bahasa Indonesia by reduplicating the noun. For example, the word `anak` means 'child', the word `anak-anak` therefore means 'children', the word `orang` means 'man', and `orang-orang` means 'men'. Clearly, there is no sequence of letters or phonemes that can be literally said to constitute a plural morph. Rather, it is the function $f\colon A^* \to A^*\colon \vec{x} \mapsto \vec{x}\text{-}\vec{x}$,

sending each string to its duplicate (with an interspersed hyphen). Actually, in writing the abbreviation `anak2` and `orang2` is commonplace. Here, `2` is a segmentable marker of plurality. However, notice that the words in the singular or the plural are each fully segmentable. Only the marker of plurality cannot be identified with any of the segments. This is to some degree also the case in German, where the rules are however much more complex, as we have seen above. The fact that morphs are (at closer look) not simply strings will be of central concern in this book.

Finally, we have to remark that letters and phonemes are not unstructured either. Phonemes consist of various so–called **distinctive features**. These are features that distinguish the phonemes from each other. For example, [p] is distinct from [b] in that it is voiceless, while [b] is voiced. Other voiceless consonants are [k], [t], while [g] and [d] are once again voiced. Such features can be relevant for the description of a language. There is a rule of German (and other languages, for example Russian) that forbids voiced consonants to occur at the end of a syllable. For example, the word `Jagd` 'hunting' is pronounced [ˈja:kt], not [ˈja:gd]. This is so since [g] and [d] may not occur at the end of the syllable, since they are voiced. Now, first of all, why do we not write `Jakt` then? This is so since inflection and derivation show that when these consonants occur non–finally in the syllable they are voiced: we have `Jagden` [ˈya:kden] 'huntings', with [d] now in fact being voiced, and also `jagen` [ˈya:gən] 'to hunt'. Second: why do we not propose that voiceless consonants become voiced when syllable initial? Because there is plenty of evidence that this does not happen. Both voiced and voiceless sounds may appear at the beginning of the syllable, and those ones that are analyzed as underlyingly voiceless remain so in whatever position. Third: why bother writing the underlying consonant rather than the one we hear? Well, first of all, since we know how to pronounce the word anyway, it does not matter whether we write [d] or [t]. On the other hand, if we know how to write the word, we also know a little bit about its morphological behaviour. What this comes down to is that to learn how to write a language is to learn how the language works. Now, once this is granted, we shall explain why we find [k] in place of [g] and [t] in place of [d]. This is because of the internal organisation of the phoneme. The phoneme is a set of distinctive features, one of which (in German) is [±voiced]. The rule is that when the voiced consonant may not occur, it is only the feature [+voiced] that is replaced by [−voiced]. Everything else remains the same. A similar situation is the relationship between upper and lower case letters. The rule says that a sentence may not begin with

a lower case letter. So, when the sentence begins, the first letter is changed to its upper case counterpart if necessary. Hence, letters too contain distinctive features. Once again, in a dictionary a word always appears as if it would normally appear elsewhere. Notice by the way that although each letter is by itself an upper or a lower case letter, written language attributes the distinction upper versus lower case to the word not to the initial letter. Disregarding some modern spellings in advertisements (like in Germany `InterRegio`, `eBusiness` and so on) this is a reasonable strategy. However, it is nevertheless not illegitimate to call it a suprasegmental feature.

In the previous section we have talked extensively about representations of terms by means of strings. In linguistics this is an important issue, which is typically discussed in conjunction with *word order*. Let us give an example. Disregarding word classes, each word of the language has one (or several) arities. The finite verb see has arity 2. The proper names Paul and Marcus on the other hand have arity 0. Any symbol of arity > 0 is called a **functor** with respect to its argument. In syntax one also speaks of **head** and **complement**. These are relative notions. In the term $\mathsf{see}(\mathsf{Marcus}, \mathsf{Paul})$, the functor or head is see, and its arguments are Paul and Marcus. To distinguish these arguments from each other, we use the terms *subject* and *object*. Marcus is the **subject** and Paul is the **object** of the sentence. The notions 'subject' and 'object' denote so–called **grammatical relations**. The correlation between argument places and grammatical relations is to a large extent arbitrary, and is of central concern in syntactical theory. Notice also that not all arguments are complements. Here, syntactical theories diverge as to which of the arguments may be called 'complement'. In generative grammar, for example, it is assumed that only the direct object is a complement.

Now, how is a particular term represented? The representation of see is `sees`, that of Marcus is `Marcus` and that of Paul is `Paul`. The whole term (1.47) is represented by the string (1.48).

(1.47) $\mathsf{see}(\mathsf{Marcus}, \mathsf{Paul})$

(1.48) `Marcus sees Paul.`

So, the verb appears after the subject, which in turn precedes the object. At the end, a period is placed. However, to spell out the relationship between a language and a formal representation is not as easy as it appears at first sight. For first of all, the term should be something that does not depend on the particular language we choose and which gives us the full meaning of the

term (so it is like a language of thought or an interlingua, if you wish). So the above term shall mean that Marcus sees Paul. We could translate the English sentence (1.48) by choosing a different representation language, but the choice between languages of representation should actually be immaterial as long as they serve the purpose. This is a very rudimentary picture but it works well for our purposes. We shall return to the idea of producing sentences from terms in Chapter 3. Now look first at the representatives of the basic symbols in some other languages.

(1.49)

	see	Marcus	Paul
German	`sieht`	`Marcus`	`Paul`
Latin	`vidit`	`Marcus`	`Paulus`
Hungarian	`látja`	`Marcus`	`Pál`

Here is how (1.47) is phrased in these languages.

(1.50) `Marcus sieht Paul.`

(1.51) `Marcus Paulum vidit.`

(1.52) `Marcus látja Pált.`

English is called an **SVO–language**, since in transitive constructions the subject precedes the verb, and the verb in turn the object. This is exactly the infix notation. (However, notice that languages do not make use of brackets.) One uses the mnemonic symbols 'S', 'V' and 'O' to define the following basic 6 types of languages: SOV, SVO, VSO, OSV, OVS, VOS. These names tell us how the subject, verb and object follow each other in a basic transitive sentence. We call a language of type VSO or VOS **verb initial**, a language of type SOV or OSV **verb final** and a language of type SVO or OVS **verb medial**. By this definition, German is SVO, Hungarian too, hence both are verb medial and Latin is SOV, hence verb final. These types are not equally distributed. Depending on the method of counting, 40 – 50 % of the world's languages are SOV languages, up to 40 % SVO languages and another 10 % are VSO languages. This means that in the vast majority of languages the order of the two arguments is: subject before object. This is why one does not generally emphasize the relative order of the subject with respect to the object. There is a bias against placing the verb initially (VSO), and a slight bias to put it finally (SOV) rather than medially (SVO).

One speaks of a **head final** (**head initial**) language if a head is consistently put at the end behind all of its arguments (at the beginning, before

all the arguments). One denotes the type of order by XH (HX), X being the complement, H the head. There is no notion of a *head medial* language for the reason that most heads only have one complement. It is often understood that the direct object is the only complement of the verb. Hence, the word orders SVO and VOS are head initial, OVS and SOV head final. (The orders VSO and OSV are problematic since the verb is not adjacent to its object.) A verb is a head, however a very important one, since it basically builds the clause. Nevertheless, different heads may place their arguments differently, so a language that is verb initial need not be head initial, a language that is verb final need not be head final. Indeed, there are few languages that are consistently head initial (medial, final). Japanese is rather consistently head final. Even a relative clause precedes the noun it modifies. Hungarian is a mixed case: adjectives precede nouns, there are no prepositions, only postpositions, but the verb tends to precede its object.

For the interested reader we give some more information on the languages shown above. First, Latin was initially an SOV language, however word order was not really fixed (see [**?**] and [**?**]). In fact, any of the six permutations of the sentence (1.51) is grammatical. Hungarian is more complex, again the word order shown in (1.52) is the least marked, but the rule is that discourse functions determine word order. (Presumably this is true for Latin as well.) German is another special case. Against all appearances there is all reason to believe that it is actually an SOV language. You can see this by noting first that only the carrier of inflection appears in second place, for example only the auxiliary if present. Second, in a subordinate clause all parts of the verb including the carrier of inflection are at the end.

(1.53) `Marcus sieht Paul.`
Marcus sees Paul.

(1.54) `Marcus will Paul sehen.`
Marcus wants to see Paul.

(1.55) `Marcus will Paul sehen können.`
Marcus wants to be able to see Paul.

(1.56) ..., `weil Marcus Paul sieht.`
..., because Marcus sees Paul.

(1.57) ..., `weil Marcus Paul sehen will.`
..., because Marcus wants to see Paul.

(1.58) ..., `weil Marcus Paul sehen können will.`
..., because Marcus wants to be able to see Paul.

So, the main sentence is not always a good indicator of the word order. Some languages allow for alternative word orders, like Latin and Hungarian. This is not to say that all variants have the same meaning or significance; it is only that they are equal as representatives of (1.47). We therefore speak of Latin as having **free word order**. However, this only means that the head and the argument can assume any order with respect to each other, not that simply all permutations of the words mean the same.

Now, notice that subject and object are coded by means of so–called **cases**. In Latin, the object carries accusative case, so we find `Paulum` instead of `Paulus`. Likewise, in Hungarian we have `Pált` in place of `Pál`, the nominative. So, the way a representing string is arrived at is rather complex. We shall return again to case marking in Chapter 5.

Natural languages also display so–called **polyvalency**. We say that a word is polyvalent if it can have several arities (even with the same meaning). The verb `to roll` can be unary (= **intransitive**) as well as binary (= **transitive** if the second argument is accusative, **intransitive** otherwise). This is not allowed in our definition of signature. However, it can easily be modified to account for polyvalent symbols.

Notes on this section. The rule that spells out the letters `ch` in German is more complex than the above explications show. For example, it is [x] in `fauchen` but [ç] in `Frauchen`. This may have two reasons: (a) There is a morpheme boundary between `u` and `ch` in the second word but not in the first. This morpheme boundary induces the difference. (b) The morpheme `chen` is special in that `ch` will always be realized as [ç]. The difference between (a) and (b) is that while (a) defines a realization rule that uses only the phonological representation, (b) uses morphological information to define the realization. Mel'čuk defines the realization rules as follows. In each stratum, there are rules that define how deep representations get mapped to surface representations. Across strata, going down, the surface representations of the higher stratum get mapped into abstract representations of the lower stratum. (For example, a sequence of morphemes is first realized as a sequence of morphs and then spelled out as a sequence of phonemes, until, finally, it gets mapped onto a sequence of phones.) Of course, one may also reverse the process. However, adjacency between (sub-)strata remains as defined.

Exercise 17. Show that in Polish Notation, unique readability is lost when there exist polyvalent function symbols.

Exercise 18. Show that if you have brackets, unique readability is guaranteed even if you have polyvalency.

Exercise 19. We have argued that German is a verb final language. But is it strictly head final? Examine the data given in this section as well as the data given below.

(1.59) `Josef pflückt eine schöne Rose für Maria.`
Josef is.picking a beautiful rose for Mary

(1.60) `Heinrich ist dicker als Josef.`
Heinrich is fatter than Josef

Exercise 20. Even if languages do not have brackets, there are elements that indicate clearly the left or right periphery of a constituent. Such elements are the determiners `the` and `a(n)`. Can you name more? Are there elements in English indicating the right periphery of a constituent? How about demonstratives like `this` or `that`?

Exercise 21. By the definitions, Unix is head initial. For example, the command `lpr` precedes its arguments. Now study the way in which optional arguments are encoded. (If you are sitting behind a computer on which Unix (or Linux) is running, type `man lpr` and you get a synopsis of the command and its syntax.) Does the syntax guarantee unique readability? (For the more linguistic minded reader: which type of marking strategy does Unix employ? Which natural language you know of corresponds best to it?)

4. Trees

Strings can also be defined as pairs $\langle \mathfrak{L}, \ell \rangle$ where $\mathfrak{L} = \langle L, < \rangle$ is a finite linearly ordered set and $\ell \colon L \to A$ a function, called the **labelling function**. Since L is finite we have $\langle L, < \rangle \cong \langle n, \in \rangle$ for $n := |L|$. (Recall that n is a set that is linearly ordered by $\in$.) Replacing $\langle L, < \rangle$ by the isomorphic $\langle n, \in \rangle$, and eliminating the redundant $\in$, a string is often defined as a pair $\langle n, \ell \rangle$, where n is a natural number. In what is to follow, we will very often have to deal with extensions

of relational structures (over a given signature Ξ) by a labelling function. They have the general form $\langle M, \mathfrak{I}, \ell \rangle$, where M is a set, $\mathfrak{I}$ an interpretation and ℓ a function from M to A. These structures shall be called **structures over** A or A**–structures**.

A very important notion in the analysis of language is that of a *tree*. A tree is a special case of a directed graph. A **directed graph** is a structure $\langle G, < \rangle$, where $< \subseteq G^2$ is a binary relation. As is common usage, we shall write $x \leq y$ if $x < y$ or $x = y$. Also, x and y are called **comparable** if $x \leq y$ or $y \leq x$. A (**directed**) **chain of length** k is a sequence $\langle x_i : i < k+1 \rangle$ such that $x_i < x_{i+1}$ for all $i < k$. An **undirected chain of length** k is a sequence $\langle x_i : i < k+1 \rangle$ where $x_i < x_{i+1}$ or $x_{i+1} < x_i$ for all $i < k$. A directed graph is called **connected** if for every two elements x and y there is an undirected chain from x to y. A directed chain of length k is called a **cycle of length** k if $x_k = x_0$. A binary relation is called **cycle free** if it only has cycles of length 0. A **root** is an element r such that for every x $x <^* r$, where $<^*$ is the reflexive, transitive closure of $<$.

Definition 1.41 *A **directed acyclic graph** (a **DAG**) is a pair $\mathfrak{G} = \langle G, < \rangle$ such that $< \subseteq G^2$ is an acyclic relation on G. If $<$ is transitive, $\mathfrak{G}$ is called a **directed transitive acyclic graph (DTAG)**.*

Definition 1.42 *$\mathfrak{G} = \langle G, < \rangle$ is called a **forest** if $<$ is transitive and irreflexive and if $x < y$ and $x < z$ then y and z are comparable. A forest with a root is called a **tree**.*

In a connected rooted DTAG the root is comparable with every other element since the relation is transitive. Furthermore, in presence of transitivity $<$ is cycle free iff it is irreflexive. For if $<$ is not irreflexive it has a cycle of length 1. Conversely, if there is a cycle $\langle x_i : i < k+1 \rangle$ of length $k > 0$, we immediately have $x_0 < x_k = x_0$, by transitivity.

If $x < y$ and there is no z such that $x < z < y$, x is called a **daughter of** y, and y the **mother of** x, and we write $x \prec y$.

Lemma 1.43 *Let $\langle T, < \rangle$ be a finite tree. If $x < y$ then there exists a $\widehat{x}$ such that $x \leq \widehat{x} \prec y$ and a $\widehat{y}$ such that $x \prec \widehat{y} \leq y$. $\widehat{x}$ and $\widehat{y}$ are uniquely determined by x and y.* □

The proof is straightforward. In infinite trees this need not hold. We define $x \circ y$ by $x \leq y$ or $y \leq x$ and say that x and y **overlap**. The following is also easy.

Lemma 1.44 (Predecessor Lemma) *Let $\mathfrak{T}$ be a finite tree and x and y nodes which do not overlap. Then there exist uniquely determined u, v and w, such that $x \leq u \prec w$, $y \leq v \prec w$ and $v \neq u$.* □

A node **branches** n **times downwards** if it has exactly n daughters; and it branches n **times upwards** if it has exactly n mothers. We say that a node **branches upwards** (**downwards**) if it branches upwards or downwards at least 2 times. A finite forest is characterized by the fact that it is transitive, irreflexive and no node branches upwards. Therefore, in connection with trees and forests we shall speak of 'branching' when we mean 'downward branching'. x is called a **leaf** if there is no $y < x$, that is, if x branches 0 times. The set of leaves of $\mathfrak{G}$ is denoted by $b(\mathfrak{G})$.

Further, we define the following notation.

$$\downarrow x := \{y : y \leq x\} \qquad \uparrow x := \{y : y \geq x\} \tag{1.61}$$

By definition of a forest, $\uparrow x$ is linearly ordered by $<$. Also, $\downarrow x$ together with the restriction of $<$ to $\downarrow x$ is a tree.

A set $P \subseteq G$ is called a **path** if it is linearly ordered by $<$ and convex, that is to say, if $x, y \in P$ then $z \in P$ for every z such that $x < z < y$. The **length of** P is defined to be $|P| - 1$. A **branch** is a maximal path with respect to set inclusion. The **height of** x in a DTAG, in symbols $h_{\mathfrak{G}}(x)$ or simply $h(x)$, is the maximal length of a branch in $\downarrow x$. It is defined inductively as follows.

$$h(x) := \begin{cases} 0 & \text{if } x \text{ is a leaf,} \\ 1 + \max\{h(y) : y \prec x\} & \text{otherwise.} \end{cases} \tag{1.62}$$

Dually we define the **depth** in a DTAG.

$$d(x) := \begin{cases} 0 & \text{if } x \text{ is a root,} \\ 1 + \max\{d(y) : y \succ x\} & \text{otherwise.} \end{cases} \tag{1.63}$$

For the entire DTAG $\mathfrak{G}$ we set

$$h(\mathfrak{G}) := \{h(x) : x \in T\} \tag{1.64}$$

and call this the **height of** $\mathfrak{G}$. (This is an ordinal, as is easily verified.)

Definition 1.45 *Let $\mathfrak{G} = \langle G, <_G \rangle$ and $\mathfrak{H} = \langle H, <_H \rangle$ be directed graphs and $G \subseteq H$. Then $\mathfrak{G}$ is called a* ***subgraph of*** *$\mathfrak{H}$ if $<_G = <_H \cap G^2$.*

If $\mathfrak{G}$ and $\mathfrak{H}$ are DTAG, forests or trees, then $\mathfrak{G}$ is a **sub–DTAG**, **subforest** and **subtree of** $\mathfrak{H}$, respectively. A subtree of $\mathfrak{H}$ with underlying set $\downarrow x$ is called a **constituent of** $\mathfrak{H}$.

Definition 1.46 *Let A be an alphabet. A **DAG over** A (or an A–**DAG**) is a pair $\langle \mathfrak{G}, \ell \rangle$ such that $\mathfrak{G} = \langle G, < \rangle$ is a DAG and $\ell \colon G \to A$ an arbitrary function.*

Alternatively, we speak of **DAGs with labels in** A, or simply of **labelled DAGs** if it is clear which alphabet is meant. Similarly with trees and DTAGs. The notions of substructures are extended analogously.

The tree structure in linguistic representations encodes the hierarchical relations between elements and not their spatial or temporal relationship. The latter have to be added explicitly. This is done by extending the signature by another binary relation symbol, $\sqsubset$. We say that x is **before** y and that y is **after** x if $x \sqsubset y$ is the case. We say that x **dominates** y if $x > y$. The relation $\sqsubset$ articulates the temporal relationship between the segments. This is first of all defined on the leaves, and it is a linear ordering. (This reflects the insistance on segmentability. It will have to be abandoned once we do not assume segmentability.) Each node x in the tree has the physical span of its segments. This allows to define an ordering between the hierarchically higher elements as well. We simply stipulate that $x \sqsubset y$ iff all leaves below x are before all leaves below y. This is not unproblematic if nodes can branch upwards, but this situation we shall rarely encounter in this book. The following is an intrinsic definition of these structures.

Definition 1.47 *An **ordered tree** is a triple $\langle T, <, \sqsubset \rangle$ such that the following holds.*

(ot1) *$\langle T, < \rangle$ is a tree.*

(ot2) *$\sqsubset$ is a linear, strict ordering on the leaves of $\langle T, < \rangle$.*

(ot3) *If $x \sqsubset z$ and $y < x$ then also $y \sqsubset z$.*
If $x \sqsubset z$ and $y < z$ then also $x \sqsubset y$.

(ot4) *If x is not a leaf and for all $y < x$ $y \sqsubset z$ then also $x \sqsubset z$.*
If z is not a leaf and for all $y < z$ $x \sqsubset y$ then also $x \sqsubset z$.

The condition (ot2) requires that the ordering is coherent with the ordering on the leaves. It ensures that $x \sqsubset y$ only if all leaves below x are before all

leaves below y. (ot3) is a completeness condition ensuring that if the latter holds, then indeed $x \sqsubset y$.

We agree on the following notation. Let $x \in G$. Put $[x] := \downarrow x \cap b(\mathfrak{G})$. We call this the **extension of** x. $[x]$ is linearly ordered by $\sqsubset$. If a labelling function ℓ is given in addition, we write $k(x) := \langle [x], \sqsubset, \ell \restriction [x] \rangle$ and call this the **associated string of** x. It may happen that two nodes have the same associated string. The string associated with the entire tree is

$$k(\mathfrak{G}) := \langle b(\mathfrak{G}), \sqsubset, \ell \restriction b(\mathfrak{G}) \rangle \tag{1.65}$$

A constituent is called **continuous** if the associated string is convex with respect to $\sqsubset$. A set M is **convex** (**with respect to** $\sqsubset$) if for all $x, y, z \in M$: if $x \sqsubset z \sqsubset y$ then $z \in M$ as well.

For sets M, N of leaves put $M \sqsubset N$ iff for all $x \in M$ and all $y \in N$ we have $x \sqsubset y$. From (ot4) and (ot3) we derive the following:

$$x \sqsubset y \quad \Leftrightarrow \quad [x] \sqsubset [y] \tag{1.66}$$

This property shows that the orderings on the leaves alone determines the relation $\sqsubset$ uniquely.

Theorem 1.48 *Let $\langle T, < \rangle$ be a tree and $\sqsubset$ a linear ordering on its leaves. Then there exists exactly one relation $\sqsubset' \supseteq \sqsubset$ such that $\langle T, <, \sqsubset' \rangle$ is an ordered tree.*

We emphasize that the ordering $\sqsubset'$ cannot be linear if the tree has more than one element. It may even happen that $\sqsubset' = \sqsubset$. One can show that overlapping nodes can never be comparable with respect to $\sqsubset$. For let $x \circ y$, say $x \leq y$. Let $u \leq x$ be a leaf. Assume $x \sqsubset y$; then by (ot3) $u \sqsubset y$ as well as $u \sqsubset u$. This contradicts the condition that $\sqsubset$ is irreflexive. Likewise $y \sqsubset x$ cannot hold. So, nodes can only be comparable if they do not overlap. We now ask: is it possible that they are comparable exactly when they do not overlap? In this case we call $\sqsubset$ **exhaustive**. Theorem 1.49 gives a criterion on the existence of exhaustive orderings. Notice that if M and N are convex sets, then so is $M \cap N$. Moreover, if $M \cap N = \varnothing$ then either $M \sqsubset N$ or $N \sqsubset M$. Also, M is convex iff for all u: $u \sqsubset M$ or $M \sqsubset u$.

Theorem 1.49 *Let $\langle T, < \rangle$ be a tree and $\sqsubset$ a linear ordering on the leaves. There exists an exhaustive extension of $\sqsubset$ iff all constituents are continuous.*

Proof. By Theorem 1.48 there exists a unique extension, $\sqsubset'$. Assume that all constituents are continuous. Let x and y are nonoverlapping nodes. Then $[x] \cap [y] = \varnothing$. Hence $[x] \sqsubset [y]$ or $[y] \sqsubset [x]$. since both sets are convex. So, by (1.66) we have $x \sqsubset' y$ or $y \sqsubset' x$. The ordering is therefore exhaustive. Conversely, assume that $\sqsubset'$ is exhaustive. Pick x. We show that $[x]$ is convex. Let u be a leaf and $u \notin [x]$. Then u does not overlap with x. By hypothesis, $u \sqsubset' x$ or $x \sqsubset' u$, whence $[u] \sqsubset [x]$ or $[x] \sqsubset [u]$, by (1.66). This means nothing but that either $u \sqsubset y$ for all $y \in [x]$ or $y \sqsubset u$ for all $y \in [x]$. So, $[x]$ is convex. □

Lemma 1.50 (Constituent Lemma) *Assume $\langle T, <, \sqsubset, \ell \rangle$ is an exhaustively ordered A–tree. Furthermore, let $p < q$. Then there is a context $C = \langle \vec{u}, \vec{v} \rangle$ such that*

$$k(q) = C(k(p)) = \vec{u}^\frown k(p)^\frown \vec{v} \tag{1.67}$$

The converse does not hold. Furthermore, it may happen that $C = \langle \varepsilon, \varepsilon \rangle$ — in which case $k(q) = k(p)$ — without $q < p$.

Proposition 1.51 *Let $\langle T, <, \sqsubset \rangle$ be an ordered tree and $x \in T$. x is 1–branching iff $[x] = [y]$ for some $y < x$.*

Proof. Let x be a 1–branching node with daughter y. Then we have $[x] = [y]$ but $x \neq y$. So, the condition is necessary. Let us show that is sufficient. Let x be minimally 2–branching. Let $u < x$. There is a daughter $z \prec x$ such that $u \leq z$, and there is $z' \prec x$ different from z. Then $[u] \subseteq [z] \subseteq [x]$ as well as $[z'] \subseteq [x]$. All sets are nonempty and $[z'] \cap [z] = \varnothing$. Hence $[z] \subsetneq [x]$ and so also $[u] \subsetneq [x]$. □

We say that a tree is **properly branching** if it has no 1–branching nodes.

There is a slightly different method of defining trees. Let T be a set and $\prec$ a cycle free relation on T such that for every x there is at most one y such that $x \prec y$. And let there be exactly one x which has no $\prec$–successor (the root). Then put $< := \prec^+$. $\langle T, < \rangle$ is a tree. And $x \prec y$ iff x is the daughter of y. Let $D(x)$ be the set of daughters of x. Now let P be a relation such that (a) $y\, P\, z$ only if y and z are sisters, (b) P^+, the transitive closure of P, is a relation that linearly orders $D(x)$ for every x, (c) for every y there is at most one z such that $y\, P\, z$ and at most one z' such that $z'\, P\, y$. Then put $x \sqsubset y$ iff there is z such that (a) $x < \widehat{x} \prec z$ for some $\widehat{x}$, (b) $y < \widehat{y} \prec y$ for some $\widehat{y}$, (c) $\widehat{x}\, P^+\, \widehat{y}$. $\prec$ and P are the immediate neighbourhood relations in the tree.

Proposition 1.52 *Let $\langle T, <, \sqsubset \rangle$ be an exhaustively ordered tree. Then $x \sqsubset y$ iff there are $x' \geq x$ and $y' \geq y$ which are sisters and $x' \sqsubset y'$.*

Finally we mention a further useful concept, that of a constituent structure.

Definition 1.53 *Let M be a set. A* ***constituent structure over*** *M is a system $\mathfrak{C}$ of subsets of M with the following properties.*

(cs1) $\{x\} \in \mathfrak{C}$ *for every* $x \in M$,

(cs2) $\varnothing \notin \mathfrak{C}$, $M \in \mathfrak{C}$,

(cs3) *if* $S, T \in \mathfrak{C}$ *and* $S \nsubseteq T$ *as well as* $T \nsubseteq S$ *then* $S \cap T = \varnothing$.

Proposition 1.54 *Let M be a nonempty set. There is a biunique correspondence between finite constituent structures over M and finite properly branching trees whose set of leaves is $\{\{x\} : x \in M\}$.*

Proof. Let $\langle M, \mathfrak{C}\rangle$ be a constituent structure. Then $\langle \mathfrak{C}, \subsetneq\rangle$ is a tree. To see this, one has to check that $\subsetneq$ is irreflexive and transitive and that it has a root. This is easy. Further, assume that $S \subsetneq T, U$. Then $U \cap T \supseteq S \neq \varnothing$, because of condition (cs2). Moreover, because of (cs3) we must have $U \subseteq T$ or $T \subseteq U$. This means nothing else than that T and U are comparable. The set of leaves is exactly the set $\{\{x\} : x \in M\}$. Conversely, let $\mathfrak{T} = \langle T, <\rangle$ be a properly branching tree. Put $M := b(\mathfrak{T})$ and $\mathfrak{C} := \{[x] : x \in T\}$. We claim that $\langle M, \mathfrak{C}\rangle$ is a constituent structure. For (cs1), notice that for every $u \in b(\mathfrak{G})$, $[u] = \{u\} \in \mathfrak{C}$. Further, for every x $[x] \neq \varnothing$, since the tree is finite. There is a root r of $\mathfrak{T}$, and we have $[r] = M$. This shows (cs2). Now we show (cs3). Assume that $[x] \nsubseteq [y]$ and $[y] \nsubseteq [x]$. Then x and y are incomparable (and different). Let u be a leaf and $u \in [x]$, then we have $u \leq x$. $u \leq y$ cannot hold since $\uparrow u$ is linear, and then x and y would be comparable. Likewise we see that from $u \leq y$ we get $u \nleq x$. Hence $[x] \cap [y] = \varnothing$. The constructions are easily seen to be inverses of each other (up to isomorphism). □

In general we can assign to every tree a constituent structure, but only if the tree is properly branching it can be properly reconstructed from this structure. The notion of a constituent structure can be extended straightforwardly to the notion of an ordered constituent structure, and we can introduce labellings.

We shall now discuss the representation of terms by means of trees. There are two different methods, both widely used. Before we begin, we shall introduce the notion of a tree domain.

Definition 1.55 *Let $T \subseteq \omega^*$ be a set of finite sequences of natural numbers. T is called a* ***tree domain*** *if the following holds.*

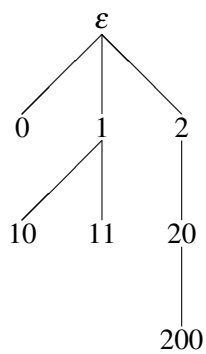


Figure 3. A Tree Domain

(td1) *If* $\vec{x}^\frown i \in T$ *then* $\vec{x} \in T$.

(td2) *If* $\vec{x}^\frown i \in T$ *and* $j < i$ *then also* $\vec{x}^\frown j \in T$.

We assign to a tree domain T an ordered tree in the following way. The set of nodes is T, (1) $\vec{x} < \vec{y}$ iff $\vec{y}$ is a proper prefix of $\vec{x}$ and (2) $\vec{x} \sqsubset \vec{y}$ iff there are numbers i, j and sequences $\vec{u}$, $\vec{v}$, $\vec{w}$ such that (a) $i < j$ and (b) $\vec{x} = \vec{u}^\frown i^\frown \vec{v}$, $\vec{y} = \vec{u}^\frown j^\frown \vec{w}$. (This is exactly the lexicographical ordering.) Together with these relations, T is an exhaustively ordered finite tree, as is easily seen. Figure 3 shows the tree domain $T = \{\varepsilon, 0, 1, 2, 10, 11, 20, 200\}$. If T is a tree domain and $\vec{x} \in T$ then put

(1.68) $\quad T/\vec{x} := \{\vec{y} : \vec{x}^\frown \vec{y} \in T\}$

This is the constituent below $\vec{x}$. (To be exact, it is not identical to this constituent, it is merely isomorphic to it. The (unique) isomorphism from $T/\vec{x}$ onto the constituent $\downarrow\vec{x}$ is the map $\vec{y} \mapsto \vec{x}^\frown \vec{y}$.)

Conversely, let $\langle T, <, \sqsubset \rangle$ be an exhaustively ordered tree. We define a tree domain T^β by induction on the depth of the nodes. If $d(x) = 0$, let $x^\beta := \varepsilon$. In this case x is the root of the tree. If x^β is defined, and y a daughter of x, then put $y^\beta := x^{\beta\frown} i$, if y is the ith daughter of x counting from the left (starting, as usual, with 0). (Hence we have $|x^\beta| = d(x)$.) We can see quite easily that the so defined set is a tree domain. For we have $\vec{u} \in T^\beta$ as soon as $\vec{u}^\frown j \in T^\beta$ for some j. Hence (td1) holds. Further, if $\vec{u}^\frown i \in T^\beta$, say $\vec{u}^\frown i = y^\beta$ then y is the ith

daughter of a node x. Take $j < i$. Then let z be the jth daughter of x (counting from the left). It exists, and we have $z^\beta = \vec{u}^\frown j$. Moreover, it can easily be shown that the relations defined on the tree domain are exactly the ones that are defined on the tree. In other words the map $x \mapsto x^\beta$ is an isomorphism.

Theorem 1.56 *Let $\mathfrak{T} = \langle T, <, \sqsubset \rangle$ be a finite, exhaustively ordered tree. The function $x \mapsto x^\beta$ is an isomorphism from $\mathfrak{T}$ onto the associated tree domain $\langle \mathfrak{T}^\beta, <, \sqsubset \rangle$. Furthermore, $\mathfrak{T} \cong \mathfrak{U}$ iff $\mathfrak{T}^\beta = \mathfrak{U}^\beta$.* □

Terms can be translated into labelled tree domains. Each term t is assigned a tree domain t^b and a labelling function t^λ. The labelled tree domain associated with t is $t^m := \langle t^b, t^\lambda \rangle$. We start with the variables. $x^b := \{\varepsilon\}$, and $x^\lambda : \varepsilon \mapsto x$. Assume that the labelled tree domains t_i^m, $i < n-1$, are defined, and put $n := \Omega(f)$. Let $s := f(t_0, \ldots, t_{n-1})$; then

$$s^b := \{\varepsilon\} \cup \bigcup_{i<n} \{i^\frown \vec{x} : \vec{x} \in t_i^b\} \tag{1.69}$$

Then s^λ is defined as follows.

$$s^\lambda(\varepsilon) := f \qquad\qquad s^\lambda(j^\frown \vec{x}) := t_j^\lambda(\vec{x}) \tag{1.70}$$

This means that s^m consists of a root named f which has n daughters, to which the labelled tree domains of $t_0, \ldots, t_{n-1}$ are isomorphic. We call the representation which sends t to t^m the **dependency coding**. This coding is more efficient that the following, which we call **structural coding**. We choose a new symbol, T, and define by induction to each term t a tree domain t^c and a labelling function t^μ. Put $x^c := \{\varepsilon, 0\}$, $x^\mu(\varepsilon) := \mathtt{T}$, $x^\mu(0) := x$. Further let for $s = f(t_0, \ldots, t_{n-1})$

$$\begin{aligned} s^c &:= \{\varepsilon, 0\} \cup \bigcup_{0<i<n+1} \{i^\frown \vec{x} : \vec{x} \in t_i^c\} \\ s^\mu(\varepsilon) &:= \mathtt{T} \\ s^\mu(0) &:= f \\ s^\mu((j+1)^\frown \vec{x}) &:= t_j^\mu(\vec{x}) \end{aligned} \tag{1.71}$$

(Compare the structural coding with the associated string in the notation without brackets.) In Figure 4 both codings are shown for the term `(3+(5*7))` for comparison. The advantage of the structural coding is that the string associated to the labelled tree domain is also the string associated to the term (with brackets dropped, as the tree encodes the structure anyway).

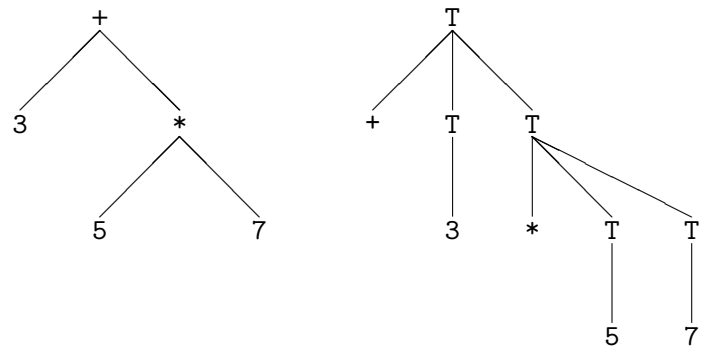


Figure 4. Dependency Coding and Structural Coding

Notes on this section. A variant of the dependency coding of syntactic structures has been proposed by Lucien Tesnière in [**?**]. He called tree representations **stemmata** (sg. *stemma*). This notation (and the theory surrounding it) became known as **dependency syntax**. See [**?**] for a survey. Unfortunately, the stemmata do not coincide with the dependency trees defined here, and this creates very subtle problems, see [**?**]. Noam Chomsky on the other hand proposed the more elaborate structural coding, which is by now widespread in linguistic theory.

Exercise 22. Define 'exhaustive ordering' on constituent structures. Show that a linear ordering on the leaves is extensible to an exhaustive ordering in a tree iff it is in the related constituent structure.

Exercise 23. Let $\mathfrak{T} = \langle T, < \rangle$ be a tree and $\sqsubset$ a binary relation such that $x \sqsubset y$ only if x, y are daughters of the same node (that is, they are sisters). Further, the daughter nodes of a given node shall be ordered linearly by $\sqsubset$. No other relations shall hold. Show that this ordering can be extended to an exhaustive ordering on $\mathfrak{T}$.

Exercise 24. Show that the number of binary branching exhaustively ordered trees over a given string is exactly

$$C_n = \frac{1}{n+1}\binom{2n}{n} \tag{1.72}$$

These numbers are called **Catalan numbers**.

Exercise 25. Show that $C_n < \frac{1}{n+1}4^n$. (One can prove that $\binom{2n}{n}$ approximates the series $\frac{4^n}{\sqrt{\pi n}}$ in the limit. The latter even majorizes the former. For the exercise there is an elementary proof.)

Exercise 26. Let L be finite with n elements and $<$ a linear ordering on L. Construct an isomorphism from $\langle L, < \rangle$ onto $\langle n, \in \rangle$.

5. Rewriting Systems

Languages are by Definition 1.36 arbitrary sets of strings over a (finite) alphabet. However, languages that interest us here are those sets which can be described by finite means, particularly by finite processes. These can be processes which generate strings directly or by means of some intermediate structure (for example, labelled trees). The most popular approach is by means of rewrite systems on strings.

Definition 1.57 *Let A be a set. A **semi Thue system** over A is a finite set $T = \{\langle \vec{x}_i, \vec{y}_i \rangle : i < m\}$ of pairs of A–strings. If T is given, write $\vec{u} \Rightarrow_T^1 \vec{v}$ if there are $\vec{s}, \vec{t}$ and some $i < m$ such that $\vec{u} = \vec{s}^\frown \vec{x}_i {}^\frown \vec{t}$ and $\vec{v} = \vec{s}^\frown \vec{y}_i {}^\frown \vec{t}$. We write $\vec{u} \Rightarrow_T^0 \vec{v}$ if $\vec{u} = \vec{v}$, and $\vec{u} \Rightarrow_T^{n+1} \vec{v}$ if there is a $\vec{z}$ such that $\vec{u} \Rightarrow_T^1 \vec{z} \Rightarrow_T^n \vec{v}$. Finally, we write $\vec{u} \Rightarrow_T^* \vec{v}$ if $\vec{u} \Rightarrow_T^n \vec{v}$ for some $n \in \omega$, and we say that $\vec{v}$ is **derivable in** T **from** $\vec{u}$.*

We can define $\Rightarrow_T^1$ also as follows. $\vec{u} \Rightarrow_T^1 \vec{v}$ iff there exists a context C and $\langle \vec{x}, \vec{y} \rangle \in T$ such that $\vec{u} = C(\vec{x})$ and $\vec{v} = C(\vec{y})$. A semi Thue system T is called a **Thue system** if from $\langle \vec{x}, \vec{y} \rangle \in T$ follows $\langle \vec{y}, \vec{x} \rangle \in T$. In this case $\vec{v}$ is derivable from $\vec{u}$ iff $\vec{u}$ is derivable from $\vec{v}$. A **derivation of** $\vec{y}$ **from** $\vec{x}$ **in** T is a finite sequence $\langle \vec{v}_i : i < n+1 \rangle$ such that $\vec{v}_0 = \vec{x}$, $\vec{v}_n = \vec{y}$ and for all $i < n$ we have $\vec{v}_i \Rightarrow_T^1 \vec{v}_{i+1}$. The **length** of this derivation is n. (A more careful definition will be given on Page 57.) Sometimes it will be convenient to admit $\vec{v}_{i+1} = \vec{v}_i$ even if there is no corresponding rule.

A *grammar* differs from a semi Thue system as follows. First, we introduce a distinction between the alphabet proper and an auxiliary alphabet, and secondly, the language is defined by means of a special symbol, the so called *start symbol*.

Definition 1.58 *A **grammar** is a quadruple $G = \langle S, N, A, R \rangle$ such that N, A are nonempty disjoint sets, $S \in N$ and R a semi Thue system over $N \cup A$ such that $\langle \vec{\gamma}, \vec{\eta} \rangle \in R$ only if $\vec{\gamma} \notin A^*$. We call S the **start symbol**, N the **nonterminal alphabet**, A the **terminal alphabet** and R the **set of rules**.*

Elements of the set N are also called **categories**. Notice that often the word 'type' is used instead of 'category', but this usage is dangerous for us in view of the fact that 'type' is reserved here for types in the λ–calculus. As a rule, we choose $S = \mathtt{S}$. This is not necessary. The reader is warned that $\mathtt{S}$ need not always be the start symbol. But if nothing else is said it is. As is common practice, nonterminals are denoted by upper case Roman letters, terminals by lower case Roman letters. A lower case Greek letter signifies a letter that is either terminal or nonterminal. The use of vector arrows follows the practice established for strings. We write $G \vdash \vec{\gamma}$ or $\vdash_G \vec{\gamma}$ in case that $S \Rightarrow_R^* \vec{\gamma}$ and say that G **generates** $\vec{\gamma}$. Furthermore, we write $\vec{\gamma} \vdash_G \vec{\eta}$ if $\vec{\gamma} \Rightarrow_R^* \vec{\eta}$. The language generated by G is defined by

$$L(G) := \{\vec{x} \in A^* : G \vdash \vec{x}\} \tag{1.73}$$

Notice that G generates strings which may contain terminal as well as nonterminal symbols. However, those that contain also nonterminals do not belong to the language that G generates. A grammar is therefore a semi Thue system which additionally defines how a derivation begins and how it ends.

Given a grammar G we call the **analysis problem** (or **parsing problem**) for G the problem (1) to say for a given string whether it is derivable in G and (2) to name a derivation in case that a string is derivable. The problem (1) alone is called the **recognition problem for** G.

A rule $\langle \vec{\alpha}, \vec{\beta} \rangle$ is often also called a **production** and is alternatively written $\vec{\alpha} \to \vec{\beta}$. We call $\vec{\alpha}$ the **left hand side** and $\vec{\beta}$ the **right hand side** of the production. The **productivity** $p(\rho)$ of a rule $\rho = \vec{\alpha} \to \vec{\beta}$ is the difference $|\vec{\beta}| - |\vec{\alpha}|$. ρ is called **expanding** if $p(\rho) \geq 0$, **strictly expanding** if $p(\rho) > 0$ and **contracting** if $p(\rho) < 0$. A rule is **terminal** if it has the form $\vec{\alpha} \to \vec{x}$ (notice that by our convention, $\vec{x} \in A^*$).

This notion of grammar is very general. There are only countably many grammars over a given alphabet — and hence only countably many languages generated by them —; nevertheless, the variety of these languages is bewildering. We shall see that every recursively enumerable language can be generated by some grammar. So, some more restricted notion of grammar is called for. Noam Chomsky has proposed the following hierarchy of grammar types. (Here, X_ε is short for $X \cup \{\varepsilon\}$.)

- ☞ Any grammar is of **Type 0**.
- ☞ A grammar is said to be of **Type 1** or **context sensitive** if all rules are

of the form $\vec{\delta}_1 X \vec{\eta}_2 \to \vec{\eta}_1 \vec{\alpha} \vec{\eta}_2$ and either (i) always $\vec{\alpha} \neq \varepsilon$ or (ii) $\mathtt{S} \to \varepsilon$ is a rule and $\mathtt{S}$ never occurs on the right hand side of a production.

☞ A grammar is said to be of **Type 2** or **context free** if it is context sensitive and all productions are of the form $X \to \vec{\alpha}$.

☞ A grammar is said to be of **Type 3** or **regular** if it is context free and all productions are of the form $X \to \vec{\alpha}$ where $\vec{\alpha} \in A_\varepsilon \cdot N_\varepsilon$.

A context sensitive rule $\vec{\eta}_1 X \vec{\eta}_2 \to \vec{\eta}_1 \vec{\alpha} \vec{\eta}_2$ is also written

$$X \to \vec{\alpha} / \vec{\eta}_1 \_\_ \vec{\eta}_2 \tag{1.74}$$

One says that X can be rewritten into $\vec{\alpha}$ **in the context** $\vec{\eta}_1 \_\_ \vec{\eta}_2$. A *language* is said to be of **Type** i if it can be generated by a grammar of Type i. It is not relevant if there also exists a grammar of Type j, $j \neq i$, that generates this language in order for it to be of Type i. We give examples of grammars of Type 3, 2 and 0.

EXAMPLE 1. There are regular grammars which generate number expressions. Here a number expression is either a number, with or without sign, or a pair of numbers separated by a dot, again with or without sign. The grammar is as follows. The set of terminal symbols is $\{\mathtt{0},\mathtt{1},\mathtt{2},\mathtt{3},\mathtt{4},\mathtt{5},\mathtt{6},\mathtt{7},\mathtt{8},\mathtt{9},\mathtt{+},\mathtt{-},\mathtt{.}\}$, the set of nonterminals is $\{\mathtt{V},\mathtt{Z},\mathtt{F},\mathtt{K},\mathtt{M}\}$. The start symbol is $\mathtt{V}$ and the productions are

$$\begin{aligned}
&\mathtt{V} \to \mathtt{+Z} \mid \mathtt{-Z} \mid \mathtt{Z} \\
&\mathtt{Z} \to \mathtt{0Z} \mid \mathtt{1Z} \mid \mathtt{2Z} \mid \cdots \mid \mathtt{9Z} \mid \mathtt{F} \\
&\mathtt{F} \to \mathtt{0} \mid \mathtt{1} \mid \mathtt{2} \mid \cdots \mid \mathtt{9} \mid \mathtt{K} \\
&\mathtt{K} \to \mathtt{.M} \\
&\mathtt{M} \to \mathtt{0M} \mid \mathtt{1M} \mid \mathtt{2M} \mid \cdots \mid \mathtt{9M} \mid \mathtt{0} \mid \mathtt{1} \mid \mathtt{2} \mid \cdots \mid \mathtt{9}
\end{aligned} \tag{1.75}$$

Here, we have used the following convention. The symbol '|' on the right hand side of a production indicates that the part on the left of this sign and the one to the right are alternatives. So, using the symbol '|' saves us from writing two rules expanding the same symbol. For example, $\mathtt{V}$ can be expanded either by $\mathtt{+Z}$, $\mathtt{-Z}$ or by $\mathtt{Z}$. The syntax of the language ALGOL has been written down in this notation, which became to be known as the **Backus–Naur Form**. The arrow was written '::='. (The Backus–Naur form actually allowed for context–free rules.)

EXAMPLE 2. The set of strings representing terms over a finite signature with finite set X of variables can be generated by a context free grammar. Let $F = \{\mathtt{F}_i : i < m\}$ and $\Omega(i) := \Omega(\mathtt{F}_i)$.

$$\mathtt{T} \quad \rightarrow \quad \mathtt{F}_i\mathtt{T}^{\Omega(i)} \qquad (i < m) \tag{1.76}$$

Since the set of rules is finite, so must be F. The start symbol is $\mathtt{T}$. This grammar generates the associated strings in Polish Notation. Notice that this grammar reflects exactly the structural coding of the terms. More on that later. If we want to have dependency coding, we have to choose instead the following grammar.

$$\begin{aligned} \mathtt{S} &\rightarrow \mathtt{F}_{j_0}\mathtt{F}_{j_1}\ldots\mathtt{F}_{j_{\Omega(i)-1}} \\ \mathtt{F}_i &\rightarrow \mathtt{F}_{j_0}\mathtt{F}_{j_1}\ldots\mathtt{F}_{j_{\Omega(i)-1}} \end{aligned} \tag{1.77}$$

This is a scheme of productions. Notice that for technical reasons the root symbol must be $\mathtt{S}$. We could dispense with the first kind of rules if we are allowed to have several start symbols. We shall return to this issue below.

EXAMPLE 3. Our example for a Type 0 grammar is the following, taken from [**?**].

$$\begin{array}{llcllcl} \text{(a)} & \mathtt{X}_0 & \rightarrow & \mathtt{a} & \mathtt{X}_0 & \rightarrow & \mathtt{aXX}_2\mathtt{Z} \\ \text{(b)} & \mathtt{X}_2\mathtt{Z} & \rightarrow & \mathtt{aa} & & & \\ \text{(c)} & \mathtt{Xa} & \rightarrow & \mathtt{aa} & \mathtt{Ya} & \rightarrow & \mathtt{aa} \\ \text{(d)} & \mathtt{X}_2\mathtt{Z} & \rightarrow & \mathtt{Y}_1\mathtt{YXZ} & & & \\ \text{(e)} & \mathtt{XY}_1 & \rightarrow & \mathtt{Y}_1\mathtt{YX} & \mathtt{YY}_1 & \rightarrow & \mathtt{Y}_1\mathtt{Y} \\ \text{(f)} & \mathtt{aY}_1 & \rightarrow & \mathtt{aXXYX}_2 & & & \\ \text{(g)} & \mathtt{X}_2\mathtt{Y} & \rightarrow & \mathtt{XY}_2 & \mathtt{Y}_2\mathtt{Y} & \rightarrow & \mathtt{YY}_2 \\ & & & & \mathtt{Y}_2\mathtt{X} & \rightarrow & \mathtt{YX}_2 \end{array} \tag{1.78}$$

$\mathtt{X}_0$ is the start symbol. This grammar generates the language $\{\mathtt{a}^{n^2} : n > 0\}$. This can be seen as follows. To start, with (a) one can either generate the string $\mathtt{a}$ or the string $\mathtt{aXX}_2\mathtt{Z}$. Let $\vec{\gamma}_i = \mathtt{aX}\vec{\delta}_i\mathtt{X}_2\mathtt{Z}$, $\vec{\delta}_i \in \{\mathtt{X},\mathtt{Y}\}^*$. We consider derivations which go from $\vec{\gamma}_i$ to a terminal string. At the beginning, only (b) or (d) can be applied. Let it be (b). Then we can only continue with (c) and then we create a string of length $4 + |\vec{\delta}_i|$. Since we have only one letter, the string is uniquely determined. Now assume that (d) has been chosen. Then we get the string $\mathtt{aX}\vec{\delta}_i\mathtt{Y}_1\mathtt{YXZ}$. The only possibility to continue is using (e). This moves the index 1 stepwise to the left and puts $\mathtt{Y}$ before every occurrence of

an X. Finally, it hits a and we use (f) to get $\mathtt{aXXYX}_2\vec{\delta}_i'\mathtt{YYXZ}$. Now there is no other choice but to move the index 2 to the right with the help of (g). This gives a string $\vec{\gamma}_{i+1} = \mathtt{aX}\vec{\delta}_{i+1}\mathtt{X}_2\mathtt{Z}$ with $\vec{\delta}_{i+1} = \mathtt{XYX}\vec{\delta}_i'\mathtt{YY}$. We have

$$(1.79) \qquad |\vec{\delta}_{i+1}| = |\vec{\delta}_i| + \ell_x(\vec{\delta}_i) + 5$$

where $\ell_x(\delta_i)$ counts the number of X in $\vec{\delta}_i$. Since $\ell_x(\vec{\delta}_{i+1}) = \ell_x(\vec{\delta}_i) + 2$, $\vec{\delta}_0 = \varepsilon$, we conclude that $\ell_x(\vec{\delta}_i) = 2i$ and so $|\vec{\delta}_i| = (i+1)^2 - 4$, $i > 0$. Hence, $|\vec{\gamma}_i| = (i+1)^2$, as promised.

In the definition of a context sensitive grammar the following must be remembered. By intention, context sensitive grammars only consist of non-contracting rules. However, since we must begin with a start symbol, there would be no way to derive the empty string if no rule is contracting. Hence, we do admit the rule $\mathtt{S} \to \varepsilon$. But in order not to let other contracting uses of this rule creep in we require that S is not on the right hand side of any rule whatsoever. Hence, $\mathtt{S} \to \varepsilon$ can only be applied once, at the beginning of the derivation. The derivation immediately terminates. This condition is also in force for context free and regular grammars although without it no more languages can be generated (see the exercises). For assume that in a grammar G with rules of the form $X \to \vec{\alpha}$ there are rules where S occurs on the right hand side of a production, and nevertheless replace S by Z in all rules which are not not of the form $\mathtt{S} \to \varepsilon$. Add also all rules $\mathtt{S} \to \vec{\alpha}'$, where $\mathtt{S} \to \vec{\alpha}$ is a rule of G and $\vec{\alpha}'$ results from $\vec{\alpha}$ by replacing S by Z. This is a context free grammar which generates the same language, and even the same structures. (The only difference is with the nodes labelled S or Z.)

The class of regular grammars is denoted by RG, the class of all context free grammars by CFG, the class of context sensitive grammars by CSG and the class of Type 0 grammars by GG. The languages generated by these grammars is analogously denoted by RL, CFL, CSL and GL. The grammar classes form a proper hierarchy.

$$(1.80) \qquad \mathrm{RG} \subsetneq \mathrm{CFG} \subsetneq \mathrm{CSG} \subsetneq \mathrm{GG}$$

This is not hard to see. It follows immediately that the languages generated by these grammar types also form a hierarchy, but not that the inclusions are proper. However, the hierarchy is once again strict.

$$(1.81) \qquad \mathrm{RL} \subsetneq \mathrm{CFL} \subsetneq \mathrm{CSL} \subsetneq \mathrm{GL}$$

We shall prove each of the proper inclusions. In Section 1.7 (Theorem 1.96) we shall show that there are languages of Type 0 which are not of Type 1. Furthermore, from the Pumping Lemma (Theorem 1.81) for CFLs it follows that $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n : n \in \omega\}$ is not context free. However, it is context sensitive (which is left as an exercise in that section). Also, by Theorem 1.65 below, the language $\{\mathtt{a}^{n^2} : n \in \omega\}$ has a grammar of Type 1. However, this language is not semilinear, whence it is not of Type 2 (see Section 2.6). Finally, it will be shown that $\{\mathtt{a}^n\mathtt{b}^n : n \in \omega\}$ is context free but not regular. (See Exercise 51.)

Let $\rho = \vec{\gamma} \to \vec{\eta}$. We call a triple $A = \langle \vec{\alpha}, C, \vec{\zeta} \rangle$ an **instance of** ρ if C is an occurrence of $\vec{\gamma}$ in $\vec{\alpha}$ and also an occurrence of $\vec{\eta}$ in $\vec{\zeta}$. This means that there exist $\vec{\kappa}_1$ and $\vec{\kappa}_2$ such that $C = \langle \vec{\kappa}_1, \vec{\kappa}_2 \rangle$ and $\vec{\alpha} = \vec{\kappa}_1{}^\frown\vec{\gamma}{}^\frown\vec{\kappa}_2$ as well as $\vec{\zeta} = \vec{\kappa}_1{}^\frown\vec{\eta}{}^\frown\vec{\kappa}_2$. We call C the **domain of** A. A **derivation of length** n is a sequence $\langle A_i : i < n \rangle$ of *instances* of rules from G such that $A_i = \langle \vec{\alpha}_i, C_i, \vec{\zeta}_i \rangle$ for $i < n$ and for every $j < n-1$ $\vec{\alpha}_{j+1} = \vec{\zeta}_j$. $\vec{\alpha}_0$ is called the **start** of the derivation, $\vec{\zeta}_{n-1}$ the **end**. We denote by $\mathrm{der}(G, \vec{\alpha})$ the set of derivations G from the string $\vec{\alpha}$ and $\mathrm{der}(G) := \mathrm{der}(G, S)$.

This definition has been carefully chosen. Let $\langle A_i : i < n \rangle$ be a derivation in G, where $A_i = \langle \vec{\alpha}_i, C_i, \vec{\alpha}_{i+1} \rangle$ $(i < n)$. Then we call $\langle \vec{\alpha}_i : i < n+1 \rangle$ the (**associated**) **string sequence**. Notice that the string sequence has one more element than the derivation. In what is to follow we shall often also call the string sequence a derivation. However, this is not quite legitimate, since the string sequence does not determine the derivation uniquely. Here is an example. Let G consist of the rules $\mathtt{S} \to \mathtt{AB}$, $\mathtt{A} \to \mathtt{AA}$ and $\mathtt{B} \to \mathtt{AB}$. Take the string sequence $\langle \mathtt{S}, \mathtt{AB}, \mathtt{AAB} \rangle$. There are two derivations for this sequence.

(1.82a) $\langle \langle \mathtt{S}, \langle \varepsilon, \varepsilon \rangle, \mathtt{AB} \rangle, \langle \mathtt{AB}, \langle \varepsilon, \mathtt{B} \rangle, \mathtt{AAB} \rangle \rangle$

(1.82b) $\langle \langle \mathtt{S}, \langle \varepsilon, \varepsilon \rangle, \mathtt{AB} \rangle, \langle \mathtt{AB}, \langle \mathtt{A}, \varepsilon \rangle, \mathtt{AAB} \rangle \rangle$

After application of a rule ρ, the left hand side $\vec{\gamma}$ is replaced by the right hand side, but the context parts $\vec{\kappa}_1$ and $\vec{\kappa}_2$ remain as before. It is intuitively clear that if we apply a rule to parts of the context, then this application could be permuted with the first. This is clarified in the following definition and theorem.

Definition 1.59 *Let $\langle \vec{\alpha}, \langle \vec{\kappa}_1, \vec{\kappa}_2 \rangle, \vec{\beta} \rangle$ be an instance of the rule $\rho = \vec{\eta} \to \vec{\vartheta}$, and let $\langle \vec{\beta}, \langle \vec{\mu}_1, \vec{\mu}_2 \rangle, \vec{\gamma} \rangle$ be an instance of $\sigma = \vec{\zeta} \to \vec{\xi}$. We call the domains of these applications* ***disjoint*** *if either (a) $\vec{\kappa}_1{}^\frown\vec{\vartheta}$ is a prefix of $\vec{\mu}_1$ or (b) $\vec{\vartheta}{}^\frown\vec{\kappa}_2$ is a suffix of $\vec{\mu}_2$.*

Lemma 1.60 (Commuting Instances) *Let $\langle \vec{\alpha}, C, \vec{\beta} \rangle$ be an instance of $\rho = \vec{\eta} \to \vec{\vartheta}$, and $\langle \vec{\beta}, D, \vec{\gamma} \rangle$ an instance of $\sigma = \vec{\zeta} \to \vec{\xi}$. Suppose that the instances are disjoint. Then there exists an instance $\langle \vec{\alpha}, D', \vec{\delta} \rangle$ of σ as well as an instance $\langle \vec{\delta}, C', \vec{\gamma} \rangle$ of ρ, and both have disjoint domains.*

The proof is easy and left as an exercise. Analogously, suppose that to the *same* string the rule ρ can be applied with context C and the rule σ can be applied with context D. Then if C precedes D, after applying one of them the domains remain disjoint, and the other can still be applied (with the context modified accordingly).

We give first an example where the instances are not disjoint. Let the following rules be given.

(1.83)
$$\begin{array}{lcl@{\qquad\qquad}lcl} \mathtt{AX} & \to & \mathtt{XA} & \mathtt{XA} & \to & \mathtt{Xa} \\ \mathtt{XB} & \to & \mathtt{Xb} & \mathtt{Xa} & \to & \mathtt{a} \end{array}$$

There are two possibilities to apply the rules to AXB. The first has domain $\langle \varepsilon, \mathtt{B} \rangle$, the second the domain $\langle \mathtt{A}, \varepsilon \rangle$. The domains overlap and indeed the first rule when applied destroys the domain of the second. Namely, if we apply the rule $\mathtt{AX} \to \mathtt{XA}$ we cannot reach a terminal string.

(1.84) $\mathtt{AXB} \Rightarrow \mathtt{XAB} \Rightarrow \mathtt{XaB}$

If on the other hand we first apply the rule $\mathtt{XB} \to \mathtt{Xb}$ we do get one.

(1.85) $\mathtt{AXB} \Rightarrow \mathtt{AXb} \Rightarrow \mathtt{XAb} \Rightarrow \mathtt{Xab} \Rightarrow \mathtt{ab}$

So much for noncommuting instances. Now take the string AXXB. Again, the two rules are in competition. However, this time none destroys the applicability of the other.

(1.86) $\mathtt{AXXB} \Rightarrow \mathtt{AXXb} \Rightarrow \mathtt{XAXb}$

(1.87) $\mathtt{AXXB} \Rightarrow \mathtt{XAXB} \Rightarrow \mathtt{XAXb}$

As before we can derive the string ab. Notice that in a CFG every pair of rules that are in competition for the same string can be used in succession with either order on condition that they do not compete for the same occurrence of a nonterminal.

Definition 1.61 *A grammar is in **standard form** if all rules are of the form $\vec{X} \to \vec{Y}$, $X \to \vec{x}$.*

In other words, in a grammar in standard form the right hand side either consists of a string of nonterminals or a string of terminals. Typically, one restricts terminal strings to a single symbol or the empty string, but the difference between these requirements is actually marginal.

Lemma 1.62 *For every grammar G of Type i there exists a grammar H of Type i in standard form such that $L(G) = L(H)$.*

Proof. Put $N' := \{\mathtt{N}_a : a \in A\} \cup N$ and $h : a \mapsto \mathtt{N}_a, X \mapsto X : N \cup A \to N^1$. For each rule ρ let $h(\rho)$ be the result of applying $\overline{h}$ to both strings. Finally, let $R' := \{h(\rho) : \rho \in R\} \cup \{\mathtt{N}_a \to a : a \in A\}$, $H := \langle \mathtt{S}, N', A, R' \rangle$. It is easy to verify, using the Commuting Instances Lemma, that $L(H) = L(G)$. (See also below for proofs of this kind.) □

We shall now proceed to show that the conditions on Type 0 grammars are actually insignificant as regards the class of generated languages. First, we may assume a set of start symbols rather than a single one. Define the notion of a **grammar*** (**of Type** i) to be a quadruple $G = \langle \Sigma, N, A, R \rangle$ such that $\Sigma \subseteq N$ and for all $S \in \Sigma$, $\langle S, N, A, R \rangle$ is a grammar (of Type i). Write $G \vdash \vec{\gamma}$ if there is an $S \in \Sigma$ such that $S \Rightarrow_R^* \vec{\gamma}$. We shall see that grammars* are not more general than grammars with respect to languages. Let G be a grammar*. Define $G^\heartsuit$ as follows. Let $S^\heartsuit \notin A \cup N$ be a new nonterminal and add the rules $S^\heartsuit \to X$ to R for all $X \in \Sigma$. It is easy to see that $L(G^\heartsuit) = L(G)$. (Moreover, the derivations differ minimally.) Notice also that we have not changed the type of the grammar.

The second simplification concerns the requirement that the set of terminals and the set of nonterminals be disjoint. We shall show that it too can be dropped without increasing the generative power. We shall sometimes work without this condition, as it can be cumbersome to deal with.

Definition 1.63 *A **quasi–grammar** is a quadruple $\langle \mathtt{S}, N, A, R \rangle$ such that A and N are finite and nonempty sets, $\mathtt{S} \in N$, and R a semi Thue system over $N \cup A$ such that if $\langle \vec{\alpha}, \vec{\beta} \rangle \in R$ then $\vec{\alpha}$ contains a symbol from N.*

Proposition 1.64 *For every quasi–grammar there exists a grammar which generates the same language.*

Proof. Let $\langle \mathtt{S}, N, A, R \rangle$ be a quasi–grammar. Put $N_1 := N \cap A$. Then assume for every $a \in N_1$ a new symbol $\mathtt{Y}_a$. Put $Y := \{\mathtt{Y}_a : a \in N_1\}$, $N^\circ := (N - N_1) \cup Y$, $A^\circ := A$. Now $N^\circ \cap A^\circ = \varnothing$. We put $\mathtt{S}^\circ := \mathtt{S}$ if $\mathtt{S} \notin A$ and $\mathtt{S}^\circ := \mathtt{Y}_\mathtt{S}$ if $\mathtt{S} \in A$.

Finally, we define the rules. Let $\vec{\alpha}^\circ$ be the result of replacing every occurrence of an $a \in N_1$ by the corresponding $\mathtt{Y}_a$. Then let

$$R^\circ := \{\vec{\alpha}^\circ \to \vec{\beta}^\circ : \vec{\alpha} \to \vec{\beta} \in R\} \cup \{\mathtt{Y}_a \to a : a \in N_1\} \tag{1.88}$$

Put $G^\circ := \langle \mathtt{S}^\circ, N^\circ, A^\circ, R^\circ \rangle$. We claim that $L(G^\circ) = L(G)$. To that end we define a homomorphism $h\colon (A \cup N)^* \to (A^\circ \cup N^\circ)^*$ by $h(a) := a$ for $a \in A - N_1$, $h(a) := \mathtt{Y}_a$ for $a \in N_1$ and $h(X) := X$ for all $X \in N - N_1$. Then $h(\mathtt{S}) = \mathtt{S}^\circ$ as well as $h(R) \subseteq R^\circ$. From this it immediately follows that if $G \vdash \vec{\alpha}$ then $G^\circ \vdash h(\vec{\alpha})$. (Induction on the length of a derivation.) Since we can derive $\vec{\alpha}$ in G° from $h(\vec{\alpha})$, we certainly have $L(G) \subseteq L(G^\circ)$. For the converse we have to convince ourselves that an instance of a rule $\mathtt{Y}_a \to a$ can always be moved to the end of the derivation. For if $\vec{\alpha} \to \vec{\beta}$ is a rule then it is of type $\mathtt{Y}_b \to b$ and replaces a $\mathtt{Y}_b$ by b; and hence it commutes with that instance of the first rule. Or it is of a different form, namely $\vec{\alpha}^\circ \to \vec{\beta}^\circ$; since a does not occur in $\vec{\alpha}^\circ$, these two instances of rules commute. Now that this is shown, we conclude from $G^\circ \vdash \vec{\alpha}$ already $G^\circ \vdash \vec{\alpha}^\circ$. This implies $G \vdash \vec{\alpha}$. □

The last of the conditions, namely that the left hand side of a production must contain a nonterminal, is also no restriction. For let $G = \langle \mathtt{S}, N, A, R \rangle$ be a grammar which does not comply with this condition. Then for every terminal a let a^1 be a new symbol and let $A^1 := \{a^1 : a \in A\}$. Finally, for each rule $\rho = \vec{\alpha} \to \vec{\beta}$ let ρ^1 be the result of replacing every occurrence of an $a \in A$ by a^1 (on every side of the production). Now set $\mathtt{S}' := \mathtt{S}$ if $\mathtt{S} \notin A$ and $\mathtt{S}' := \mathtt{S}^1$ otherwise, $R' := \{\rho^1 : \rho \in R\} \cup \{a^1 \to a : a \in A\}$. Finally put $G' := \langle \mathtt{S}', N \cup A^1, A, R' \rangle$. It is not hard to show that $L(G') = L(G)$. These steps have simplified the notion of a grammar considerably. Its most general form is $\langle \Sigma, N, A, R \rangle$, where $\Sigma \subseteq N$ is the set of start symbols and $R \subseteq (N \cup A)^* \times (N \cup A)^*$ a finite set.

Next we shall show a general theorem for context sensitive languages. A grammar is called **noncontracting** if either no rule is contracting or only the rule $\mathtt{S} \to \varepsilon$ is contracting and in this case the symbol $\mathtt{S}$ never occurs to the right of a production. Context sensitive grammars are contracting. However, not all noncontracting grammars are context sensitive. It turns out, however, that *all* noncontracting grammars generate context sensitive languages. (This can be used also to show that the context sensitive languages are exactly those languages that are recognized by a linearly space bounded Turing machine.)

Theorem 1.65 *A language is context sensitive iff there is a noncontracting grammar that generates it.*

Proof. ($\Rightarrow$) Immediate. ($\Leftarrow$) Let G be a noncontracting grammar. We shall construct a grammar $G^{\spadesuit}$ which is context sensitive and such that $L(G^{\spadesuit}) = L(G)$. To this end, let $\rho = X_0X_1\cdots X_{m-1} \to Y_0Y_1\cdots Y_{n-1}$, $m \leq n$, be a production. (As remarked above, we can reduce attention to such rules and rules of the form $X \to a$. Since the latter are not contracting, only the former kind needs attention.) We assume m new symbols, $Z_0, Z_1, \ldots, Z_{m-1}$. Let $\rho^{\spadesuit}$ be the following set of rules.

$$
\begin{aligned}
X_0X_1\cdots X_{m-1} &\to Z_0X_1\cdots X_{m-1}\\
Z_0X_1X_2\cdots X_{m-1} &\to Z_0Z_1X_2\cdots X_{m-1}\\
&\cdots\\
Z_0Z_1\cdots Z_{m-2}X_{m-1} &\to Z_0Z_1\cdots Z_{m-1}\\
Z_0Z_1\cdots Z_{m-1} &\to Y_0Z_1\cdots Z_{m-1}\\
Y_0Z_1Z_2\cdots Z_{m-1} &\to Y_0Y_1Z_2\cdots Z_{m-1}\\
&\cdots\\
Y_0Y_1\cdots Y_{m-2}Z_{m-1} &\to Y_0Y_1\cdots Y_{n-1}
\end{aligned}
\tag{1.89}
$$

Let $G^{\spadesuit}$ be the result of replacing all non context sensitive rules ρ by $\rho^{\spadesuit}$. The new grammar is context sensitive. Now let us be given a derivation in G. Then replace every instance of a rule ρ by the given sequence of rules in $\rho^{\spadesuit}$. This gives a derivation of the same string in $G^{\spadesuit}$. Conversely, let us be given a derivation in $G^{\spadesuit}$. Now look at the following. If somewhere the rule $\rho^{\spadesuit}$ is applied, and then a rule from $\rho_1^{\spadesuit}$ then the instances commute unless $\rho_1 = \rho$ and the second instance is inside that of that rule instance of $\rho^{\spadesuit}$. Thus, by suitably reordering the derivation is a sequence of segments, where each segment is a sequence of the rule $\rho^{\spadesuit}$ for some ρ, so that it begins with $\vec{X}$ and ends with $\vec{Y}$. This can be replaced by ρ. Do this for every segment. This yields a derivation in G. $\square$

Given that there are Type 0 languages that are not Type 0 (Theorem 1.96) the following theorem shows that the languages of Type 1 are not closed under arbitrary homomorphisms.

Theorem 1.66 *Let* $\mathtt{a}, \mathtt{b} \notin A$ *be (distinct) symbols. For every language* L *over* A *of Type 0 there is a language* M *over* $A \cup \{\mathtt{a}, \mathtt{b}\}$ *of Type 1 such that for every* $\vec{x} \in L$ *there is an* i *with* $\mathtt{a}^i\mathtt{b}\vec{x} \in M$ *and every* $\vec{y} \in M$ *has the form* $\mathtt{a}^i\mathtt{b}\vec{x}$ *with* $\vec{x} \in L$.

Proof. We put $N^{\clubsuit} := N \cup \{\mathtt{A}, \mathtt{B}, \mathtt{S}^{\clubsuit}\}$. Let

$$\rho = X_0 X_1 \cdots X_{m-1} \to Y_0 Y_1 \cdots Y_{n-1} \tag{1.90}$$

be a contracting rule. Then put

$$\rho^{\clubsuit} := X_0 X_1 \cdots X_{m-1} \to \mathtt{A}^{m-n} Y_0 Y_1 \cdots Y_{n-1} \tag{1.91}$$

$\rho^{\clubsuit}$ is certainly not contracting. If ρ is not contracting then put $\rho^{\clubsuit} := \rho$. Let $R^{\clubsuit}$ consist of all rules of the form $\rho^{\clubsuit}$ for $\rho \in R$ as well as the following rules.

$$\begin{aligned} \mathtt{S}^{\clubsuit} &\to \mathtt{BS} \\ X\mathtt{A} &\to \mathtt{A}X \qquad (X \in N^{\clubsuit}) \\ \mathtt{BA} &\to \mathtt{aB} \\ \mathtt{B} &\to \mathtt{b} \end{aligned} \tag{1.92}$$

Let $M := L(G^{\clubsuit})$. Certainly, $\vec{y} \in M$ only if $\vec{y} = \mathtt{a}^i \mathtt{b} \vec{x}$ for some $\vec{x} \in A^*$. For strings contain $\mathtt{B}$ (or $\mathtt{b}$) only once. Further, $\mathtt{A}$ can be changed into $\mathtt{a}$ only if it occurs directly before $\mathtt{B}$. After that we get $\mathtt{B}$ followed by $\mathtt{a}$. Hence $\mathtt{b}$ must occur after all occurrences of $\mathtt{a}$ but before all occurrences of $\mathtt{B}$. Now consider the homomorphism $\overline{v}$ defined by $v\colon \mathtt{A}, \mathtt{a}, \mathtt{B}, \mathtt{b}, \mathtt{S}^{\clubsuit} \mapsto \varepsilon$ and $v\colon X \mapsto X$ for $X \in N$, $v\colon a \mapsto a$ for $a \in A$. If $\langle \vec{\alpha}_i : i < n \rangle$ is a derivation in $G^{\clubsuit}$ then $\langle \overline{v}(\vec{\alpha}_i) : 0 < i < n \rangle$ is a derivation in G (if we disregard repetitions). In this way one shows that $\mathtt{a}^i \mathtt{b} \vec{x} \in M$ implies $\vec{x} \in L(G)$. Next, let $\vec{x} \in L(G)$. Let $\langle \vec{\alpha}_i : i < n \rangle$ be a derivation of $\vec{x}$ in G. Then do the following. Define $\vec{\beta}_0 := S^{\clubsuit}$ and $\vec{\beta}_1 = \mathtt{BS}$. Further, let $\vec{\beta}_{i+1}$ be of the form $\mathtt{BA}^{k_i} \vec{\alpha}_i$ for some k_i which is determined inductively. It is easy to see that $\vec{\beta}_{i+1} \vdash_{G^{\clubsuit}} \vec{\beta}_{i+2}$, so that one can complete the sequence $\langle \vec{\beta}_i : i < n+1 \rangle$ to a derivation. From $\mathtt{BA}^{k_n} \vec{x}$ one can derive $\mathtt{a}^{k_n} \mathtt{b} \vec{x}$. This shows that $\mathtt{a}^{k_n} \mathtt{b} \vec{x} \in M$, as desired. □

Now let $v\colon A \to B^*$ be a map. v (as well as the generated homomorphism $\overline{v}$) is called **ε–free** if $v(a) \neq \varepsilon$ for all $a \in A$.

Theorem 1.67 *Let L_1 and L_2 be languages of Type i, $0 \leq i \leq 3$. Then the following are also languages of Type i.*

① $L_1 \cup L_2$, $L_1 \cdot L_2$, L_1^*.

② $\overline{v}[L_1]$, *where v is ε–free.*

If $i \neq 1$ then $\overline{v}[L_1]$ also is of Type i even if v is not ε–free.

Proof. Before we begin, we remark the following. If $L \subseteq A^*$ is a language and $G = \langle \mathtt{S}, N, A, R\rangle$ a grammar over A which generates L then for an arbitrary $B \supseteq A$ $\langle \mathtt{S}, N, B, R\rangle$ is a grammar over B which generates $L \subseteq B^*$. Therefore we may now assume that L_1 and L_2 are languages over the same alphabet. ① is seen as follows. We have $G_1 = \langle \mathtt{S}_1, N_1, A, R_1\rangle$ and $G_2 = \langle \mathtt{S}_2, N_2, A, R_2\rangle$ with $L(G_1) = L(G_2)$. By renaming the nonterminals of G_2 we can see to it that $N_1 \cap N_2 = \varnothing$. Now we put $N_3 := N_1 \cup N_2 \cup \{\mathtt{S}^\diamondsuit\}$ (where $\mathtt{S}^\diamondsuit \notin N_1 \cup N_2$) and $R := R_1 \cup R_2 \cup \{\mathtt{S}^\diamondsuit \to \mathtt{S}_1, \mathtt{S}^\diamondsuit \to \mathtt{S}_2\}$. This defines $G_3 := \langle \mathtt{S}^\diamondsuit, N_3, A, R_3\rangle$. This is a grammar which generates $L_1 \cup L_2$. We introduce a new start symbol $\mathtt{S}^\times$ together with the rules $\mathtt{S}^\times \to \mathtt{S}_1\mathtt{S}_2$ where $\mathtt{S}_1$ is the start symbol of G_1 and G_2 the start symbol of G_2. This yields a grammar of Type i except if $i = 3$. In this case the fact follows from the results of Section 2.1. It is however not difficult to construct a grammar which is regular and generates the language $L_1 \cdot L_2$. Now for L_1^*. Let $\mathtt{S}$ be the start symbol for a grammar G which generates L_1. Then introduce a new symbol $\mathtt{S}^+$ as well as a new start symbol $\mathtt{S}^*$ together with the rules

$$\begin{aligned} \mathtt{S}^* &\to \varepsilon \mid \mathtt{S} \mid \mathtt{S}\mathtt{S}^+ \\ \mathtt{S}^+ &\to \mathtt{S} \mid \mathtt{S}\mathtt{S}^+ \end{aligned} \tag{1.93}$$

This grammar is of Type i and generates L_1^*. (Again the case $i = 3$ is an exception that can be dealt with in a different way.) Finally, ②. Let v be ε–free. We extend it by putting $v(X) := X$ for all nonterminals X. Then replace the rules $\rho = \vec{\alpha} \to \vec{\beta}$ by $\overline{v}(\rho) := \overline{v}(\vec{\alpha}) \to \overline{v}(\vec{\beta})$. If $i = 0, 2$, this does not change the type. If $i = 1$ we must additionally require that v is ε–free. For if $\vec{\gamma}X\vec{\delta} \to \vec{\gamma}\vec{\alpha}\vec{\delta}$ is a rule and $\vec{\alpha}$ is a terminal string we may have $\overline{v}(\alpha) = \varepsilon$. This is however not the case if v is ε–free. If $i = 3$ again a different method must be used. For now — after applying the replacement — we have rules of the form $X \to \vec{x}Y$ and $X \to \vec{x}$, $\vec{x} = x_0x_1 \cdots x_{n-1}$. Replace the latter by $X \to x_0Z_0$, $Z_i \to x_iZ_{i+1}$ and $Z_{n-2} \to x_{n-1}Y$ and $Z_{n-2} \to x_{n-1}$, respectively. □

Definition 1.68 *Let A be a (possibly infinite) set. A nonempty set $\mathcal{S} \subseteq \wp(A^*)$ is called an* ***abstract family of languages (AFL) over*** *A if the following holds.*

① *For every $L \in \mathcal{S}$ there is a finite $B \subseteq A$ such that $L \subseteq B^*$.*

② *If $h\colon A^* \to A^*$ is a homomorphism and $L \in \mathcal{S}$ then also $h[L] \in \mathcal{S}$.*

③ *If $h\colon A^* \to A^*$ is a homomorphism and $L \in \mathcal{S}$, $B \subseteq A$ finite, then also $h^{-1}[L] \cap B^* \in \mathcal{S}$.*

④ *If $L \in \mathcal{S}$ and R is a regular language then $L \cap R \in \mathcal{S}$.*

⑤ *If $L_1, L_2 \in \mathcal{S}$ then also $L_1 \cup L_2 \in \mathcal{S}$ and $L_1 \cdot L_2 \in \mathcal{S}$.*

We still have to show that the languages of Type i are closed with respect to intersections with regular languages. A proof for the Types 3 and 2 is found in Section 2.1, Theorem 2.14. This proof can be extended to the other types without problems.

The regular, the context free and the Type 0 languages over a fixed alphabet form an abstract family of languages. The context sensitive languages fulfill all criteria except for the closure under homomorphisms. It is easy to show that the regular languages over A form the smallest abstract family of languages. More on this subject can be found in [**?**].

Notes on this section. It is a gross simplification to view languages as sets of strings. The idea that they can be defined by means of formal processes did not become apparent until the 1930s. The idea of formalizing rules for transforming strings was first formulated by Axel Thue [**?**]. The observation that languages (in his case formal languages) could be seen as generated from semi Thue systems, is due to Emil Post. Also, he has invented independently what is now known as the Turing machine and has shown that this machine does nothing but string transformations. The idea was picked up by Noam Chomsky and he defined the hierarchy which is now named after him (see for example [**?**], but the ideas have been circulating earlier). In view of Theorem 1.66 it is unclear, however, whether grammars of Type 0 or 1 have any relevance for natural language syntax, since there is no notion of a constituent that they define as opposed to context free grammars. There are other points to note about these types of grammars. [**?**] voices clear discontentment with the requirement of a single start symbol, which is in practice anyway not complied with.

Exercise 27. Let T be a semi Thue system over A and $A \subseteq B$. Then T is also a semi Thue system T' over B. Characterize $\Rightarrow_{T'}^{*} \subseteq B^* \times B^*$ by means of $\Rightarrow_{T}^{*} \subseteq A^* \times A^*$. *Remark.* This exercise shows that with the Thue system we also have to indicate the alphabet on which it is based.

Exercise 28. Let A be a finite alphabet. Every string $\vec{x}$ is the value of a constant term $\vec{x}^E$ composed from constants $\underline{a}$ for every $a \in A$, the symbol ε, and $^\frown$. Let T be a Thue system over A. Write $T^E := \{\vec{x}^E = \vec{y}^E : \langle \vec{x}, \vec{y} \rangle \in T\}$. Let M be consist of Equations (1.27) and (1.28). T^E is an equational theory. Show that $\vec{x} \Rightarrow_T^* \vec{y}$ iff $\vec{y} \Rightarrow_T^* \vec{x}$ iff $T^E \cup M \vdash \vec{x}^E = \vec{y}^E$.

Exercise 29. Prove the Commuting Instances Lemma.

Exercise 30. Show that every finite language is regular.

Exercise 31. Let G be a grammar with rules of the form $X \to \vec{\alpha}$. Show that $L(G)$ is context free. Likewise show that $L(G)$ is regular if all rules have the form $X \to \alpha_0{}^\frown\alpha_1$ where $\alpha_0 \in A \cup \{\varepsilon\}$ and $\alpha_1 \in N \cup \{\varepsilon\}$.

Exercise 32. Let G be a grammar in which every rule distinct from $X \to a$ is strictly expanding. Show that a derivation of a string of length n takes at most $2n$ steps.

Exercise 33. Show that the language $\{\mathtt{a}^n\mathtt{b}^n : n \in \omega\}$ is context free.

Exercise 34. Write a Type 1 grammar for the language $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n : n \in \omega\}$ and one for $\{\vec{x}{}^\frown\vec{x} : \vec{x} \in A^*\}$.

6. Grammar and Structure

Processes that replace strings by strings can often be considered as processes that successively replace parts of structures by structures. In this section we shall study processes of structure replacement. They can in principle operate on any kind of structure. But we will restrict our attention to algorithms that generate ordered trees. There are basically two kinds of algorithms: the first is like the grammars of the previous section, generating intermediate structures that are not proper structures of the language; and the second, which generates in each step a structure of the language.

Instead of graphs we shall deal with so–called *multigraphs*. A **directed multigraph** is a structure $\langle V, \langle K_i : i < n\rangle\rangle$ where is V a set, the set of **vertices**, and $K_i \subseteq V \times V$ a disjoint set, the set of **edges** of type i. In our case edges are always directed. We shall not mention this fact explicitly later on. Ordered trees are one example among many of (directed) multigraphs. For technical reasons we shall not exclude the case $V = \varnothing$, so that $\langle \varnothing, \langle \varnothing : i < n\rangle\rangle$ also is a multigraph. Next we shall introduce a colouring on the vertices. A **vertex–colouring** is a function $\mu_V : V \to F_V$ where F_V is a nonempty set, the set of **vertex colours**. Think of the labelling as being a vertex colouring on the graph. The principal structures are therefore vertex coloured multigraphs. However, from a technical point of view the different edge relations can also be viewed as colourings on the edges. Namely, if v and w are vertices, we colour the edge $\langle v, w\rangle$ by the set $\{i : \langle v, w\rangle \in K_i\}$. This set may be empty.

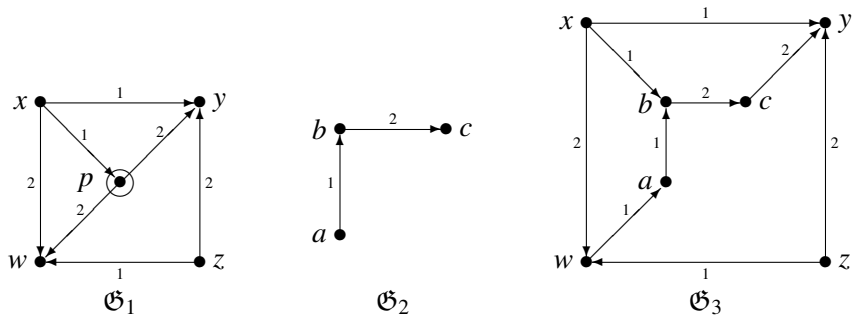


Figure 5. Graph Replacement

Definition 1.69 *An $\langle F_V, F_E\rangle$–**coloured multigraph** or simply a γ–**graph** (**over** F_V **and** F_E) is a triple $\langle V, \mu_V, \mu_E\rangle$, where V is a (possibly empty) set and $\mu_V : V \to F_V$ as well as $\mu_E : V \times V \to \wp(F_E)$ are functions.*

Now, in full analogy to the string case we shall distinguish terminal and non-terminal colours. For simplicity, we shall study only replacements of a single vertex by a graph. Replacing a vertex by another structure means embedding a structure into some other structure. We need to be told how to do so. Before we begin we shall say something about the graph replacement in general. The reader is asked to look at Figure 5. The graph $\mathfrak{G}_3$ is the result of replacing in $\mathfrak{G}_1$ the encircled dot by $\mathfrak{G}_2$. The edge colours are 1 and 2 (the vertex colours pose no problems, so they are omitted here for clarity).

Let $\mathfrak{G} = \langle E, \mu_E, \mu_K\rangle$ be a γ–graph and M_1 and M_2 be disjoint subsets of E with $M_1 \cup M_2 = E$. Put $\mathfrak{M}_i = \langle M_i, \mu_V^i, \mu_E^i\rangle$, where $\mu_V^i := \mu_V \restriction M_i$ and $\mu_E^i := \mu_E \restriction M_i \times M_i$. These graphs do not completely determine $\mathfrak{G}$ since there is no information on the edges between them. We therefore define functions $\mathrm{in}, \mathrm{out}\colon M_2 \times F_E \to \wp(M_1)$, which for every vertex of M_2 and every edge colour name the set of all vertices of M_1 which lie on an edge with a vertex that either is directed into M_1 or goes outwards from M_1.

$$\mathrm{in}(x, f) := \{y \in M_1 : f \in \mu_E(\langle y, x\rangle)\} \tag{1.94a}$$

$$\mathrm{out}(x, f) := \{y \in M_1 : f \in \mu_E(\langle x, y\rangle)\} \tag{1.94b}$$

It is clear that $\mathfrak{M}_1$, $\mathfrak{M}_2$ and the functions in and out determine $\mathfrak{G}$ completely.

In our example we have

$$\begin{aligned} \mathrm{in}(p,1) &= \{x\} & \mathrm{in}(p,2) &= \varnothing \\ \mathrm{out}(p,1) &= \varnothing & \mathrm{out}(p,2) &= \{w,y\} \end{aligned} \tag{1.95}$$

Now assume that we want to replace $\mathfrak{M}_2$ by a different graph $\mathfrak{H}$. Then not only do we have to know $\mathfrak{H}$ but also the functions $\mathrm{in}, \mathrm{out}\colon H \times F_E \to \wp(M_1)$. This, however, is not the way we wish to proceed here. We want to formulate rules of replacement that are general in that they do not presuppose exact knowledge about the embedding context. We shall only assume that the functions $\mathrm{in}(x,f)$ and $\mathrm{out}(x,f)$, $x \in H$, are systematically defined from the sets $\mathrm{in}(y,g)$, $\mathrm{out}(y,g)$, $y \in M_2$. We shall therefore only allow to specify how the sets of the first kind are formed from the sets of the second kind. This we do by means of four so–called *colour functionals*. A **colour functional from $\mathfrak{H}$ to $\mathfrak{M}_2$** is a map

$$\mathfrak{F}\colon H \times F_E \to \wp(M_2 \times F_E) \tag{1.96}$$

In our case a functional is a function from $\{a,b,c\} \times \{1,2\}$ to $\wp(\{p\} \times \{1,2\})$. We can simplify this to a function from $\{a,b,c\} \times \{1,2\}$ to $\wp(\{1,2\})$. The colour functionals are called $\mathfrak{II}$, $\mathfrak{IO}$, $\mathfrak{OI}$ and $\mathfrak{OO}$. For the example of Figure 5 we get the following colour functionals (we only give values when the functions do not yield $\varnothing$).

$$\begin{aligned} \mathfrak{II}&\colon \langle b,1\rangle \mapsto \{1\} & \mathfrak{OI}&\colon \langle a,2\rangle \mapsto \{1\} \\ \mathfrak{IO}&\colon \varnothing & \mathfrak{OO}&\colon \langle c,2\rangle \mapsto \{2\} \end{aligned} \tag{1.97}$$

The result of substituting $\mathfrak{M}_2$ by $\mathfrak{H}$ by means of the colour functionals from $\mathfrak{F}$ is denoted by $\mathfrak{G}[\mathfrak{H}/\mathfrak{M}_2 : \mathfrak{F}]$. This graph is the union of $\mathfrak{M}_1$ and $\mathfrak{H}$ together with the functions in^+ and out^+, which are defined as follows.

$$\begin{aligned} \mathrm{in}^+(x,f) &:= \bigcup \langle \mathrm{in}(x,g) : g \in \mathfrak{II}(x,f)\rangle \\ &\quad \cup \bigcup \langle \mathrm{out}(x,g) : g \in \mathfrak{OI}(x,f)\rangle \\ \mathrm{out}^+(x,f) &:= \bigcup \langle \mathrm{out}(x,g) : g \in \mathfrak{OO}(x,f)\rangle \\ &\quad \cup \bigcup \langle \mathrm{in}(x,g) : g \in \mathfrak{IO}(x,f)\rangle \end{aligned} \tag{1.98}$$

If $g \in \mathfrak{II}(x,f)$ we say that an edge with colour g into x is *transmitted as an ingoing edge of colour f to y*. If $g \in \mathfrak{OI}(x,f)$ we say that an edge with

colour g going out from x is *transmitted as an ingoing edge with colour f to y*. Analogously for $\mathfrak{IO}$ and $\mathfrak{OO}$. So, we do allow for an edge to change colour and direction when being transmitted. If edges do not change direction, we only need the functionals $\mathfrak{II}$ and $\mathfrak{OO}$, which are then denoted simply by $\mathfrak{I}$ and $\mathfrak{O}$. Now we look at the special case where M_2 consists of a single element, say x. In this case a colour functional simply is a function $\mathfrak{F}\colon H \times F_E \to \wp(F_E)$.

Definition 1.70 *A **context free graph grammar with edge replacement** — a **context free γ–grammar** for short — is a quintuple of the form*

$$\Gamma = \langle \mathfrak{S}, F_V, F_V^T, F_E, R \rangle \tag{1.99}$$

*in which F_V is a finite set of vertex colours, F_E a finite set of edge colours, $F_V^T \subseteq F_V$ a set of so–called **terminal vertex colours**, $\mathfrak{S}$ a γ–graph over F_V and F_E, the so–called **start graph**, and finally R a finite set of triples $\langle X, \mathfrak{H}, \mathbb{F} \rangle$ such that $X \in F_V - F_V^T$ is a nonterminal vertex colour, $\mathfrak{H}$ a γ–graph over F_V and F_E and $\mathbb{F}$ is a matrix of colour functionals.*

A **derivation** in a γ–grammar Γ is defined as follows. For γ–graphs $\mathfrak{G}$ and $\mathfrak{H}$ with the colours F_V and F_E, $\mathfrak{G} \Rightarrow_R^1 \mathfrak{H}$ means that there is $\langle X, \mathfrak{M}, \mathbb{F} \rangle \in R$ such that $\mathfrak{H} = \mathfrak{G}[\mathfrak{M}/\mathfrak{X} : \mathbb{F}]$, where $\mathfrak{X}$ is a subgraph consisting of a single vertex x having the colour X. Further we define $\Rightarrow_R^*$ to be the reflexive and transitive closure of $\Rightarrow_R^1$ and finally we put $\Gamma \vdash \mathfrak{G}$ if $\mathfrak{S} \Rightarrow_R^* \mathfrak{G}$. A derivation **terminates** if there is no vertex with a nonterminal colour. We write $L_\gamma(\Gamma)$ for the class of γ–graphs that can be generated from Γ. Notice that the edge colours only the vertex colours are used to steer the derivation.

We also define the productivity of a rule as the difference between the cardinality of the replacing graph and the cardinality of the graph being replaced. The latter is 1 in context free γ–grammars, which is the only type we shall study here. So, the productivity is always ≥ -1. It equals -1 if the replacing graph is the empty graph. A rule has productivity 0 if the replacing graph consists of a single vertex. In the exercises the reader will be asked to verify that we can dispense with rules of this kind.

Now we shall define two types of context free γ–grammars. Both are context free as γ–grammars but the second type can generate non–CFLs. This shows that the concept of γ–grammar is more general. We shall begin with ordinary CFGs. We can view them alternatively as grammars for string replacement or as grammars that replace trees by trees. For that we shall now

assume that there are no rules of the form $X \to \varepsilon$. (For such rules generate trees whose leaves are not necessarily marked by letters from A. This case can be treated if we allow labels to be in $A_\varepsilon = A \cup \{\varepsilon\}$, which we shall not do here.) Let $G = \langle \mathtt{S}, A, N, R \rangle$ be such a grammar. We put $F_V := A \cup (N \times 2)$. We write X^0 for $\langle X, 0 \rangle$ and X^1 for $\langle X, 1 \rangle$. $F_V^T := A \cup N \times \{0\}$. $F_E := \{<, \sqsubset\}$. Furthermore, the start graph consists of a single vertex labelled $\mathtt{S}^1$ and no edge. The rules of replacement are as follows. Let $\rho = X \to \alpha_0 \alpha_1 \cdots \alpha_{n-1}$ be a rule from G, where none of the α_i is ε. Then we define a γ–graph $\mathfrak{H}_\rho$ as follows. $H_\rho := \{y_i : i < n\} \cup \{x\}$. $\mu_V(x) = X^0$, $\mu_V(y_i) = \alpha_i$ if $\alpha_i \in A$ and $\mu_V(y_i) = \alpha_i^1$ if $\alpha_i \in N$.

$$\begin{aligned} &\mu_E^{-1}(\{<\}) := \{\langle y_i, x \rangle : i < n\}, \\ &\mu_E^{-1}(\{\sqsubset\}) := \{\langle y_i, y_j \rangle : i < j < n\}. \end{aligned} \tag{1.100}$$

This defines $\mathfrak{H}_\rho$. Now we define the colour functionals. For $u \in n$ we put

$$\begin{aligned} &\mathfrak{I}_\rho(u, \sqsubset) := \{\sqsubset\} \quad \mathfrak{O}_\rho(u, \sqsubset) := \{\sqsubset\} \\ &\mathfrak{I}_\rho(u, <) := \{<\} \quad \mathfrak{O}_\rho(u, <) := \{<\} \end{aligned} \tag{1.101}$$

Finally we put $\rho^\gamma := \langle X, \mathfrak{H}_\rho, \{\mathfrak{I}_\rho, \mathfrak{O}_\rho\} \rangle$. $R^\gamma := \{\rho^\gamma : \rho \in R\}$.

$$\gamma G := \langle \mathfrak{S}, F_E, F_E^T, F_T, R^\gamma \rangle \tag{1.102}$$

We shall show that this grammar yields exactly those trees that we associate with the grammar G. Before we do so, a few remarks are in order. The nonterminals of G are now from a technical viewpoint terminals since they are also part of the structure that we are generating. In order to have any derivation at all we define two equinumerous sets of nonterminals. Each nonterminal N is split into the nonterminal N^1 (which is nonterminal in the new grammar) and N^0 (which is now a terminal vertex colour). We call the first kind **active**, **nonactive** the second. Notice that the rules are formulated in such a way that only the leaves of the generated trees carry active nonterminals. A single derivation step is displayed in Figure 6. In it, the rule $\mathtt{X} \to \mathtt{AcA}$ has been applied to the tree to the left. The result is shown on the right hand side. It is easy to show that in each derivation only leaves carry active nonterminals. This in turn shows that the derivations of the γ–grammar are in one to one correspondence with the derivations of the CFGs. We put

$$L_B(G) := h[L_\gamma(\gamma G)] \tag{1.103}$$

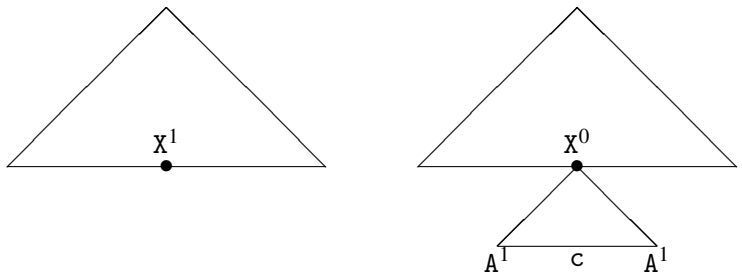


Figure 6. Replacement in a Context Free Grammar

This is the class of trees generated by γG, with X^0 and X^1 mapped to X for each $X \in N$. The rules of G can therefore be interpreted as conditions on labelled ordered trees in the following way. $\mathfrak{C}$ is called a **local subtree** of $\mathfrak{B}$ if (i) it has height 2 (so it does not possess inner nodes) and (ii) it is maximal with respect to inclusion. For a rule $\rho = X \to Y_0 Y_1 \cdots Y_{n-1}$ we define $L_\rho := \{y_i : i < n\} \cup \{x\}$, $<_\rho := \{\langle y_i, x\rangle : i < n\}$, $\sqsubset_\rho := \{\langle y_i, y_j\rangle : i < j < n\}$, and finally $\ell_\rho(x) := X$, $\ell(y_i) := Y_i$. $\mathfrak{L}_\rho := \langle L_\rho, <_\rho, \sqsubset_\rho, \ell_\rho\rangle$. Now, an isomorphism between labelled ordered trees $\mathfrak{B} = \langle B, <_\mathfrak{B}, \sqsubset_\mathfrak{B}, \ell_\mathfrak{B}\rangle$ and $\mathfrak{C} = \langle C, <_\mathfrak{C}, \sqsubset_\mathfrak{C}, \ell_\mathfrak{C}\rangle$ is a bijective map $h\colon B \to C$ such that $h[<_\mathfrak{B}] = <_\mathfrak{C}$, $h[\sqsubset_\mathfrak{B}] = \sqsubset_\mathfrak{C}$ and $\ell_\mathfrak{C}(h(x)) = \ell_\mathfrak{B}(x)$ for all $x \in B$.

Proposition 1.71 *Let $G = \langle \mathtt{S}, N, A, R\rangle$. $\mathfrak{B} \in L_B(G)$ iff every local tree of $\mathfrak{B}$ is isomorphic to an $\mathfrak{L}_\rho$ such that $\rho \in R$.*

Theorem 1.72 *Let $\boldsymbol{B}$ be a set of trees over an alphabet $A \cup N$ with terminals from A. Then $\boldsymbol{B} = L_B(G)$ for a CFG G iff there is a finite set $\{\mathfrak{L}_i : i < n\}$ of trees of height 2 and an S such that $\mathfrak{B} \in \boldsymbol{B}$ exactly if*

① *the root carries label S,*

② *a label is terminal iff the node is a leaf, and*

③ *every local tree is isomorphic to some $\mathfrak{L}_i$.*

We shall derive a few useful consequences from these considerations. It is clear that γG generates trees that do not necessarily have leaves with terminal

symbols. However, we do know that the leaves carry labels either from A or from $N^1 := N \times \{1\}$ while all other nodes carry labels from $N^0 := N \times \{0\}$. For a labelled tree we define the associated string sequence $k(\mathfrak{B})$ in the usual way. This is an element of $(A \cup N^1)^*$. Let $v\colon A \cup (N \times 2) \to A \cup N$ be defined by $v(a) := a$, $a \in A$ and $v(X^0) := v(X^1) := X$ for $X \in N$.

Lemma 1.73 *Let $\gamma G \vdash \mathfrak{B}$ and $\vec{\alpha} = k(\mathfrak{B})$. Then $\vec{\alpha} \in (A \cup N^1)^*$ and $G \vdash \overline{v}(\vec{\alpha})$.*

Proof. Induction over the length of the derivation. If the length is 0 then $\vec{\alpha} = \mathtt{S}^1$ and $\overline{v}(\mathtt{S}^1) = \mathtt{S}$. Since $G \vdash \mathtt{S}$ this case is settled. Now let $\mathfrak{B}$ be the result of an application of some rule ρ^γ on $\mathfrak{C}$ where $\rho = X \to \vec{\gamma}$. We then have $k(\mathfrak{C}) \in (A \cup N^1)^*$. The rule ρ^γ has been applied to a leaf; this leaf corresponds to an occurrence of X^1 in $k(\mathfrak{C})$. Therefore we have $k(\mathfrak{C}) = \vec{\eta}_1{}^\frown X^1{}^\frown \vec{\eta}_2$. Then $k(\mathfrak{B}) = \vec{\eta}_1{}^\frown \vec{\gamma}^\frown \vec{\eta}_2$. $k(\mathfrak{B})$ is the result of a single application of the rule ρ from $k(\mathfrak{C})$. $\square$

Definition 1.74 *Let $\mathfrak{B}$ be a labelled ordered tree. A **cut through** $\mathfrak{B}$ is a maximal set that contains no two elements comparable by $<$. If $\mathfrak{B}$ is exhaustively ordered, a cut is linearly ordered and labelled, and then we also call the string associated to this set a **cut**.*

Proposition 1.75 *Let $\gamma G \vdash \mathfrak{B}$ and let $\vec{\alpha}$ be a cut through $\mathfrak{B}$. Then $G \vdash \overline{v}(\vec{\alpha})$.*

This theorem shows that the tree provides all necessary information. If you have the tree, all essential details of the derivation can be reconstructed (up to commuting applications of rules). Now let us be given a tree $\mathfrak{B}$ and let $\vec{\alpha}$ be a cut. We say that an occurrence C of $\vec{\gamma}$ in $\vec{\alpha}$ is a **constituent of category** X **in** $\mathfrak{B}$ if this occurrence of $\vec{\gamma}$ in $\vec{\alpha}$ is that cut defined by $\vec{\alpha}$ on $\downarrow x$ where x carries the label X. This means that $\vec{\alpha} = \vec{\eta}_1{}^\frown \vec{\gamma}^\frown \vec{\eta}_2$, $C = \langle \vec{\eta}_1, \vec{\eta}_2 \rangle$, and $\downarrow x$ contains exactly those nodes that do not belong to $\vec{\eta}_1$ or $\vec{\eta}_2$. Further, let G be a CFG. A substring occurrence of $\vec{\gamma}$ is a G–constituent of category X in $\vec{\alpha}$ if there is a γG–tree for which there exists a cut $\vec{\alpha}$ such that the occurrence $\vec{\gamma}$ is a constituent of category X. If G is clear from the context, we shall omit it.

Lemma 1.76 *Let $\mathfrak{B}$ be a γG–tree and $\vec{\alpha}$ a cut through $\mathfrak{B}$. Then there exists a tree $\mathfrak{C}$ with associated string $\vec{\gamma}$ and $\overline{v}(\vec{\gamma}) = \overline{v}(\vec{\alpha})$.*

Lemma 1.77 *Let $G \vdash \vec{\alpha}_1{}^\frown \vec{\gamma}^\frown \vec{\alpha}_2$, $C = \langle \vec{\alpha}_1, \vec{\alpha}_2 \rangle$ an occurrence of $\vec{\gamma}$ as a G–constituent of category X. Then C is a G–constituent occurrence of X in $C(X) = \vec{\alpha}_1{}^\frown X^\frown \vec{\alpha}_2$.*

For a proof notice that if $\vec{\alpha}_1{}^\frown\vec{\gamma}{}^\frown\vec{\alpha}_2$ is a cut and $\vec{\gamma}$ is a constituent of category X therein then $\vec{\alpha}_1{}^\frown X^\frown\vec{\alpha}_2$ also is a cut.

Theorem 1.78 (Constituent Substitution) *Suppose that C is an occurrence of $\vec{\beta}$ as a G–constituent of category X. Furthermore, let $X \vdash_G \vec{\gamma}$. Then $G \vdash C(\vec{\gamma}) = \vec{\alpha}_1{}^\frown\vec{\gamma}{}^\frown\vec{\alpha}_2$ and C is a G–constituent occurrence of $\vec{\gamma}$ of category X.*

Proof. By assumption there is a tree in which $\vec{\beta}$ is a constituent of category X in $\vec{\alpha}_1{}^\frown\vec{\beta}{}^\frown\vec{\alpha}_2$. Then there exists a cut $\vec{\alpha}_1{}^\frown X^\frown\vec{\alpha}_2$ through this tree, and by Lemma 1.76 there exists a tree with associated string $\vec{\alpha}_1{}^\frown X^\frown\vec{\alpha}_2$. Certainly we have that X is a constituent in this tree. However, a derivation $X \vdash_G \vec{\gamma}$ can in this case be extended to a γG–derivation of $\vec{\alpha}_1{}^\frown\vec{\gamma}{}^\frown\vec{\alpha}_2$ in which $\vec{\gamma}$ is a constituent. □

Lemma 1.79 *Let G be a CFG. Then there exists a number k_G such that for each derivation tree of a string of length $\geq k_G$ there are two constituents $\downarrow y$ and $\downarrow z$ of identical category such that $y \leq z$ or $z \leq y$, and the associated strings are different.*

Proof. To begin, notice that nothing changes in our claim if we eliminate the unproductive rules. This does not change the constituent structure. Now let π be the maximum of all productivities of rules in G, and $\nu := |N|$. Then let $k_G := (1+\pi)^\nu + 1$. We claim that this is the desired number. (We can assume that $\pi > 0$. Otherwise G only generates strings of length 1, and then $k_G := 2$ satisfies our claim.) For let $\vec{x}$ be given such that $|\vec{x}| \geq k_G$. Then there exists in every derivation tree a branch of length $> \nu$. (If not, there can be no more than π^ν leaves.) On this branch we have two nonterminals with identical label. The strings associated to these nodes are different since we have no unproductive rules. □

We say, an occurrence C is a **left constituent part** (**right constituent part**) if C is an occurrence of a prefix (suffix) of a constituent. An occurrence of $\vec{x}$ contains a left constituent part $\vec{z}$ if some suffix of $\vec{x}$ is a left constituent part. We also remark that if $\vec{u}$ is a left constituent part and a proper substring of $\vec{x}$ then $\vec{x} = \vec{v}\vec{v}_1\vec{u}$ with $\vec{v}_1$ a possibly empty sequence of constituents and $\vec{v}$ a right constituent part. This will be of importance in the sequel.

Lemma 1.80 *Let G be a CFG. Then there exists a number k'_G such that for every derivation tree of a string $\vec{x}$ and every occurrence in $\vec{x}$ of a string $\vec{z}$ of length $\geq k'_G$ $\vec{z}$ contains two different left or two different right constituent parts $\vec{y}$ and $\vec{y}_1$ of constituents that have the same category. Moreover, $\vec{y}$ is a*

prefix of $\vec{y}_1$ or $\vec{y}_1$ a prefix of $\vec{y}$ in case that both are left constituent parts, and $\vec{y}$ is a suffix of $\vec{y}_1$ or $\vec{y}_1$ a suffix of $\vec{y}$ in case that both are right constituent parts.

Proof. Let $\nu := |N|$ and let π be the maximal productivity of a rule from G. We can assume that $\pi \geq 2$. Put $k'_G := (2+2\pi)^\nu$. We show by induction on the number m that a string of length $\geq (2+2\pi)^m$ has at least m left or at least m right constituent parts that are contained in each other. If $m = 1$ the claim is trivial. Assume that it holds for $m \geq 1$. We shall show that it also holds for $m+1$. Let $\vec{z}$ be of length $\geq (2+2\pi)^{m+1}$. Let $\vec{x} = \prod_{i<2\pi+2} \vec{x}_i$ for certain $\vec{x}_i$ with length at least $(2+2\pi)^m$. By induction hypothesis each $\vec{x}_i$ contains at least m constituent parts. Now we do not necessarily have $(2\pi+2)m$ constituent parts in $\vec{x}$. For if $\vec{x}_i$ contains a left part then $\vec{x}_j$ with $j > i$ may contain the corresponding right part. (There is only one. The sections in between contain subwords of that constituent occurrence.) For each left constituent part we count at most one (corresponding) right constituent part. In total we have at least $(1+\pi)m \geq m+1$ constituent parts. However, we have to verify that at least $m+1$ of these are contained inside each other. Assume this is not the case, for all i. Then $\vec{x}_i$, $i < 2\pi+2$, contains exactly m left or exactly m right constituent parts. Case 1. $\vec{x}_0$ contains m left constituent parts inside each other. If $\vec{x}_1$ also contains m left constituent parts inside each other, we are done. Now suppose that this is not the case. Then $\vec{x}_1$ contains m right constituent parts inside each other. Then we obviously get m entire constituents stacked inside each other. Again, we would be done if $\vec{x}_2$ contained m right constituent parts inside each other. If not, then $\vec{x}_2$ contains exactly m left constituent parts. And again we would be done if these would not correspond to exactly m right part that $\vec{x}_3$ contains. And so on. Hence we get a sequence of length π of constituents which each contain m constituents stacked inside each other. Now three cases arise: (a) one of the constituents is a left part of some constituent, (b) one of the constituent is a right part of some constituent. (For if neither is the case, we have a rule of arity $> \pi$, a contradiction.) In Case (a) we evidently have $m+1$ left constituent parts stacked inside each other, and in Case (b) $m+1$ right constituent parts. Case 2. $\vec{x}_0$ contains m right hand constituents stacked inside each other. Similarly. This shows our auxiliary claim. Putting $m := \nu+1$ the main claim now follows. □

Theorem 1.81 (Pumping Lemma) *Given a CFL L there exists a p_L such that for every string $\vec{z} \in L$ of length at least p_L and an occurrence of a string $\vec{r}$ of length at least p_L in $\vec{z}$, $\vec{z}$ possesses a decomposition*

(1.104) $\quad \vec{z} = \vec{u}^\frown \vec{x}^\frown \vec{v}^\frown \vec{y}^\frown \vec{w}$

such that the following holds.

① $\vec{x}^\frown\vec{y} \neq \varepsilon$.

② *Either the occurrence of* $\vec{x}$ *or the occurrence of* $\vec{y}$ *is contained in the specified occurrence of* $\vec{r}$.

③ $\{\vec{u}^\frown\vec{x}^{i\frown}\vec{v}^\frown\vec{y}^{i\frown}\vec{w} : i \in \omega\} \subseteq L$.

(The last property is called the ***pumpability*** *of the substring occurrences of* $\vec{x}$ *and* $\vec{y}$*.) Alternatively, in place of ② one may require that* $|\vec{v}| \leq p_L$*. Further we can choose* p_L *in such a way that every derivable string* $\vec{\gamma}$ *with designated occurrences of a string* $\vec{\alpha}$ *of length* $\geq p_S$ *can be decomposed in the way given.*

Proof. Let G be a grammar which generates L. Let p_L be the constant defined in Lemma 1.80. We look at a G–tree of $\vec{z}$ and the designated occurrence of $\vec{r}$. Suppose that $\vec{r}$ has length at least p_L. Then there are two left or two right constituent parts of identical category contained in $\vec{r}$. Without loss of generality we assume that $\vec{r}$ contains two left parts. Suppose that these parts are not fully contained in $\vec{r}$. Then $\vec{r} = \vec{s}\vec{x}\vec{s}_1$ where $\vec{x}\vec{s}_1$ and $\vec{s}_1$ are left constituent parts of identical category, say X. Now $|\vec{x}| > 0$. There are $\vec{s}_2$ and $\vec{y}$ such that $\vec{v} := \vec{s}_1\vec{s}_2$ and $\vec{x}\vec{s}_1\vec{s}_2\vec{y}$ are constituents of category X.

Hence there exists a decomposition

$$\vec{z} = \vec{u}^\frown\vec{x}^\frown\vec{v}^\frown\vec{y}^\frown\vec{w} \tag{1.105}$$

where $\vec{v}$ is a constituent of the same category as $\vec{x}\vec{v}\vec{y}$ satisfying ① and ②. By the Constituent Substitution Theorem we may replace the occurrence of $\vec{x}\vec{v}\vec{y}$ by $\vec{v}$ as well as $\vec{v}$ by $\vec{x}\vec{v}\vec{y}$. This yields ③, after an easy induction. Now let the smaller constituent part be contained in $\vec{r}$ but not the larger one. Then we have a decomposition $\vec{r} = \vec{s}\vec{x}\vec{v}\vec{s}_1$ such that $\vec{v}$ is a constituent part of category X and $\vec{x}\vec{v}\vec{s}_1$ a left constituent part of a constituent of category X. Then there exists a $\vec{s}_2$ such that also $\vec{x}\vec{v}\vec{s}_1\vec{s}_2$ is a constituent of category X. Now put $\vec{y} := \vec{s}_1\vec{s}_2$. Then we also have $\vec{y} \neq \varepsilon$. The third case is if both parts are proper substrings of $\vec{r}$. Also here we find the desired decomposition. If we want to have in place of ② that $\vec{v}$ is as small as possible then notice that $\vec{v}$ already is a constituent. If it has length $\geq (1+\pi)^\nu$ then there is a decomposition of $\vec{v}$ such that it contains pumpable substrings. Hence in place of ② we may require that $|\vec{v}| \leq p_G$. □

The Pumping Lemma can be stated more concisely as follows. For every large enough derivable string $\vec{x}$ there exist contexts C, D, where $C \neq \langle\varepsilon,\varepsilon\rangle$,

and a string $\vec{y}$ such $\vec{x} = D(C(\vec{y}))$, and $D(C^k(\vec{y})) \in L$ for every $k \in \omega$. The strongest form of a pumping lemma is the following. Suppose that we have two decompositions into pumping pairs $\vec{u}_1{}^\frown\vec{x}_1{}^\frown\vec{v}_1{}^\frown\vec{y}_1{}^\frown\vec{w}_1$, $\vec{u}_2{}^\frown\vec{x}_2{}^\frown\vec{v}_2{}^\frown\vec{y}_2{}^\frown\vec{w}_2$. We say that the two pairs are **independent** if either (1a) $\vec{u}_1{}^\frown\vec{x}_1{}^\frown\vec{v}_1{}^\frown\vec{y}_1$ is a prefix of $\vec{u}_2$, or (1b) $\vec{u}_2{}^\frown\vec{x}_2{}^\frown\vec{v}_2{}^\frown\vec{y}_2$ is a prefix of $\vec{u}_1$, or (1c) $\vec{u}_1{}^\frown\vec{x}_1$ is a prefix of $\vec{u}_2$ and $\vec{y}_1{}^\frown\vec{w}_1$ a suffix of $\vec{w}_2$, or (1d) $\vec{u}_2{}^\frown\vec{x}_2$ is a prefix of $\vec{u}_1$ and $\vec{y}_2{}^\frown\vec{w}_2$ a suffix of $\vec{w}_1$ and (2) each of them can be pumped any number of times independently of the other.

Theorem 1.82 (Manaster-Ramer & Moshier & Zeitman) *Let L be a CFL. Then there exists a number m_L such that if $\vec{x} \in L$ and we are given km_L occurrences of letters in $\vec{x}$ there are k independent pumping pairs, each of which contains at least one and at most m_L of the occurrences.*

This theorem implies the well–known **Ogden's Lemma** (see [**?**]), which says that given at least m_L occurrences of letters, there exists a pumping pair containing at least one and at most m_L of them.

Notice that in all these theorems we may choose $i = 0$ as well. This means that not only we can pump 'up' the string so that it becomes longer except if $i = 1$, but we may also pump it 'down' ($i = 0$) so that the string becomes shorter. However, one can pump down only once. Using the Pumping Lemma we can show that the language $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n : n \in \omega\}$ is not context free.

For suppose the contrary. Then there is an m such that for all $k \geq m$ the string $\mathtt{a}^k\mathtt{b}^k\mathtt{c}^k$ can be decomposed into

$$\mathtt{a}^k\mathtt{b}^k\mathtt{c}^k = \vec{u}^\frown\vec{v}^\frown\vec{w}^\frown\vec{x}^\frown\vec{y} \tag{1.106}$$

Furthermore there is an $\ell > k$ such that

$$\mathtt{a}^\ell\mathtt{b}^\ell\mathtt{c}^\ell = \vec{u}^\frown\vec{v}^{2\frown}\vec{w}^\frown\vec{x}^{2\frown}\vec{y} \tag{1.107}$$

The string $\vec{v}^\frown\vec{x}$ contains exactly $\ell - k$ times the letters $\mathtt{a}$, $\mathtt{b}$ and $\mathtt{c}$. It is clear that we must have $\vec{v} \subseteq \mathtt{a}^* \cup \mathtt{b}^* \cup \mathtt{c}^*$. For if $\vec{v}$ contains two distinct letters, say $\mathtt{b}$ and $\mathtt{c}$, then $\vec{v}$ contains an occurrence of $\mathtt{b}$ before an occurrence of $\mathtt{c}$ (certainly not the other way around). But then $\vec{v}^2$ contains an occurrence of $\mathtt{c}$ before an occurrence of $\mathtt{b}$, and that cannot be. Analogously it is shown that $\vec{y} \in \mathtt{a}^* \cup \mathtt{b}^* \cup \mathtt{c}^*$. But this is a contradiction. We shall meet this example of a non–CFL quite often in the sequel.

The second example of a context free graph grammar shall be the so–called *tree adjunction grammars*. We take an alphabet A and a set N of non-terminals. A **centre tree** is an ordered labelled tree over $A \cup N$ such that all

leaves have labels from A all other nodes labels from N. An **adjunction tree** is an ordered labelled tree over $A \cup N$ which is distinct from ordinary trees in that of the leaves there is exactly one with a nonterminal label; this label is the same as that of the root. Interior nodes have nonterminal labels. We require that an adjunction tree has at least one leaf with a terminal symbol. An **unregulated tree adjunction grammar**, briefly **UTAG**, over N and A, is a quadruple $\langle \mathbb{C}, N, A, \mathbb{A} \rangle$ where $\mathbb{C}$ is a finite set of centre trees over N and A, and $\mathbb{A}$ a finite set of adjunction trees over N and A. An example of a tree adjunction is given in Figure 7. The tree to the left is adjoined to a centre tree with root X and associated string `bXb`; the result is shown to the right. Tree adjunction can formally be defined as follows. Let $\mathfrak{B} = \langle B, <, \sqsubset, \ell \rangle$ be a tree and $\mathfrak{A} = \langle A, <, \sqsubset, m \rangle$ an adjunction tree. We assume that r is the root of $\mathfrak{A}$ and that s is the unique leaf such that $m(r) = m(s)$. Now let x be a node of B such that $\ell(x) = m(r)$. Then the replacement of x by $\mathfrak{B}$ is defined by naming the colour functionals. These are

$$\mathfrak{I}\mathfrak{I}_\rho(y, \sqsubset) := \begin{cases} \{\sqsubset, <\} & \text{if } s \sqsubset y, \\ \{\sqsubset\} & \text{else.} \end{cases} \qquad \mathfrak{O}\mathfrak{I}_\rho(y, \sqsubset) := \varnothing \tag{1.108}$$

$$\mathfrak{I}\mathfrak{O}_\rho(y, \sqsubset) := \begin{cases} \{<\} & \text{if } y \sqsubset s, \\ \varnothing & \text{else.} \end{cases} \qquad \mathfrak{O}\mathfrak{O}_\rho(y, \sqsubset) := \{\sqsubset\}$$

$$\mathfrak{I}\mathfrak{I}_\rho(y, <) := \begin{cases} \{<\} & \text{if } y \geq s, \\ \varnothing & \text{else.} \end{cases} \qquad \mathfrak{I}\mathfrak{O}_\rho(y, <) := \varnothing \tag{1.109}$$

$$\mathfrak{O}\mathfrak{I}\rho(y, <) := \varnothing \qquad \mathfrak{O}\mathfrak{O}_\rho(y, <) := \{<\}$$

Two things may be remarked. First, instead of a single start graph we have a finite set of them. This can be remedied by standard means. Second, all vertex colours are terminal as well as nonterminal. One may end the derivation at any given moment. We have noticed in connection with grammars for strings that this can be remedied. In fact, we have not defined context free γ–grammars but context free quasi γ–grammars*. However, we shall refrain from being overly pedantic. Suffice it to note that the adjunction grammars do not define the same kind of generative process if defined exactly as above.

Finally we shall give a graph grammar which generates all strings of the form $\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n$, $n > 0$. The idea for this grammar is due to Uwe Mönnich [**?**]. We shall exploit the fact that we may think of terms as structures. We posit a ternary symbol, `F`, which is nonterminal, and another ternary symbol, `f`, which is terminal. Further, there is a binary terminal symbol $^\frown$. The rules are

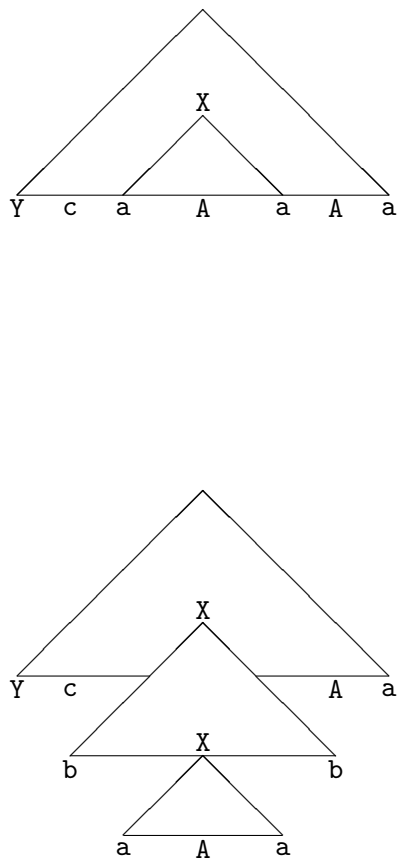


Figure 7. Tree Adjunction

as follows. (To enhance readability we shall not write terms in Polish Notation but by means of brackets.)

$$\begin{aligned} &\mathtt{F}(x,y,z) \to \mathtt{F}(\mathtt{a}^\frown x, \mathtt{b}^\frown y, \mathtt{c}^\frown z), \\ &\mathtt{F}(x,y,z) \to \mathtt{f}(x,y,z). \end{aligned} \tag{1.110}$$

These rules constitute a so–called **term replacement system**. The start term is $\mathtt{F}(\mathtt{a},\mathtt{b},\mathtt{c})$. Now suppose that $u \to v$ is a rule and that we have derived a

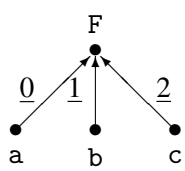
term t such that u^σ occurs in t as a subterm. Then we may substitute this occurrence by v^σ. Hence we get the following derivations.

$$
\begin{aligned}
&\mathtt{F}(\mathtt{a},\mathtt{b},\mathtt{c}) \to \mathtt{f}(\mathtt{a},\mathtt{b},\mathtt{c}),\\
&\mathtt{F}(\mathtt{a},\mathtt{b},\mathtt{c}) \to \mathtt{F}(\mathtt{a}^\frown\mathtt{a},\mathtt{b}^\frown\mathtt{b},\mathtt{c}^\frown\mathtt{c})\\
&\qquad\qquad \to \mathtt{f}(\mathtt{a}^\frown\mathtt{a},\mathtt{b}^\frown\mathtt{b},\mathtt{c}^\frown\mathtt{c})\\
&\mathtt{F}(\mathtt{a},\mathtt{b},\mathtt{c}) \to \mathtt{F}(\mathtt{a}^\frown\mathtt{a},\mathtt{b}^\frown\mathtt{b},\mathtt{c}^\frown\mathtt{c})\\
&\qquad\qquad \to \mathtt{F}(\mathtt{a}^\frown(\mathtt{a}^\frown\mathtt{a}),\mathtt{b}^\frown(\mathtt{b}^\frown\mathtt{b}),\mathtt{c}^\frown(\mathtt{c}^\frown\mathtt{c}))\\
&\qquad\qquad \to \mathtt{f}(\mathtt{a}^\frown(\mathtt{a}^\frown\mathtt{a}),\mathtt{b}^\frown(\mathtt{b}^\frown\mathtt{b}),\mathtt{c}^\frown(\mathtt{c}^\frown\mathtt{c}))
\end{aligned}
\tag{1.111}
$$

Notice that the terms denote graphs here. We make use of the dependency coding. Hence the associated strings to these terms are `abc`, `aabbcc` and `aaabbbccc`.

In order to write a graph grammar which generates the graphs for these terms we shall have to introduce colours for edges. Put $F_E := \{\underline{0},\underline{1},\underline{2},\sqsubset,<\}$, $F_V := \{\mathtt{F},\mathtt{f},\mathtt{a},\mathtt{b},\mathtt{c}\}$, and $F_V^T := \{\mathtt{f},\mathtt{a},\mathtt{b},\mathtt{c}\}$. The start graph is as follows. It has four vertices, p, q, r and s. ($<$ is empty (!), and $q \sqsubset r \sqsubset s$.) The labelling is $p \mapsto \mathtt{F}$, $q \mapsto \mathtt{a}$, $r \mapsto \mathtt{b}$ and $s \mapsto \mathtt{c}$.

(1.112)



There are two rules of replacement. The first can be written schematically as follows. The root, x, carries the label F and has three incoming edges; their colours are $\underline{0}$, $\underline{1}$ and $\underline{2}$. These come from three disjoint subgraphs, $\mathfrak{G}_0$, $\mathfrak{G}_1$ and $\mathfrak{G}_2$, which are ordered trees with respect to $<$ and $\sqsubset$ and in which there are no edges with colour $\underline{0}$, $\underline{1}$ and $\underline{2}$. In replacement, x is replaced by a graph consisting of seven vertices, p, q_i, r_i and s_i, $i < 2$, where $q_i \sqsubset r_j \sqsubset s_k$, $i,j,k < 2$, and $q \xrightarrow{\underline{0}} p$, $r \xrightarrow{\underline{1}} p$ and $s \xrightarrow{\underline{2}} p$. $<= \{\langle q_1,q_0\rangle,\langle r_1,r_0\rangle,\langle s_1,s_0\rangle\}$. The colouring is

$$
\begin{array}{llllllll}
p & \mapsto \mathtt{F} & q_0 & \mapsto\frown & r_0 & \mapsto\frown & s_0 & \mapsto\frown\\
 & & q_1 & \mapsto \mathtt{a} & r_1 & \mapsto \mathtt{b} & s_1 & \mapsto \mathtt{c}
\end{array}
\tag{1.113}
$$

(With $\{p,q_0,r_0,s_0\}$ we reproduce the begin situation.) The tree $\mathfrak{G}_0$ is attached to q_0 to the right of q_1, $\mathfrak{G}_1$ to r_0 to the right of r_1 and $\mathfrak{G}_2$ to s_0 to the right of

s_1. Additionally, we put $x < p$ for all vertices x of the $\mathfrak{G}_i$. (So, the edge $\langle x, p\rangle$ has colour $<$ for all such x.) By this we see to it that in each step the union of the relations $<$, $\underline{0}$, $\underline{1}$ and $\underline{2}$ is the intended tree ordering and that there always exists an ingoing edge with colour $\underline{0}$, $\underline{1}$ and $\underline{2}$ into the root.

The second replacement rule replaces the root by a one vertex graph with label `f` at the root. This terminates the derivation. The edges with label $\underline{0}$, $\underline{1}$ and $\underline{2}$ are transmitted under the name $<$. This completes the tree. It has the desired form.

Exercise 35. Strings can also be viewed as multigraphs with only one edge colour. Show that a CFG for strings can also be defined as a context free γ–grammar on strings. We shall show in Section 2.6 that CFLs can also be generated by UTAGs, but that the converse does not hold.

Exercise 36. Show that for every context free γ–grammar Γ there exists a context free γ–grammar Δ which has no rules of productivity -1 and which generates the same class of graphs.

Exercise 37. Show that for every context free γ–grammar there exists a context free γ–grammar with the same yield and no rules of productivity ≤ 0.

Exercise 38. Define unregulated string adjunction grammars in a similar way to UTAGs. Take note of the fact that these are quasi–grammars. Characterize the class of strings generated by these grammars in terms of ordinary grammars.

Exercise 39. Show that the language $\{\vec{w}^\frown\vec{w} : \vec{w} \in A^*\}$ is not context free but that it satisfies the Pumping Lemma. (It does not satisfy the Interchange Lemma (2.111).)

7. Turing machines

We owe to [**?**] and [**?**] the concept of a machine which is very simple and nevertheless capable of computing all functions that are believed to be computable. Without going into the details of what makes a function computable, it is nowadays agreed that there is no loss if we define 'computable' to mean *computable by a Turing machine*. The essential idea was that computations on objects can be replaced by computations on strings. The number n can for example be represented by $n+1$ successive strokes on a piece of paper. (So, the number 0 is represented by a single stroke. This is really necessary.) In

addition to the stroke we have a blank, which is used to separate different numbers. The Turing machine, however powerful, takes a lot of time to compute even the most basic functions. Hence we agree from the start that it has an arbitrary, finite stock of symbols that it can use in addition to the blank. A Turing machine is a physical device, consisting of a tape which is infinite in both directions. That is, it contains cells numbered by the set of integers (but the numbering is irrelevant for the computation). Each cell may carry a symbol from an alphabet A or a blank. The machine possesses a read and write head, which can move between the cells, one at a time. Finally, it has finitely many states, and can be programmed in the following way. We assign instructions for the machine that tell it what to do on condition that it is in state q and reads a symbol a from the tape. These instruction tell the machine whether it should write a symbol, then move the head one step or leave it at rest, and subsequently change to a state q'.

Definition 1.83 *A **(nondeterministic) Turing machine** is a quintuple*

$$T = \langle A, L, Q, q_0, f \rangle \tag{1.114}$$

*where A is a finite set, the **alphabet**, $L \notin A$ is the so–called **blank**, Q a finite set, the set of **(internal) states**, $q_0 \in Q$ the **initial state** and*

$$f \colon A_L \times Q \to \wp(A_L \times \{-1, 0, 1\} \times Q) \tag{1.115}$$

*the **transition function**. If for all $b \in A_L$ and $q \in Q$ $|f(b,q)| \leq 1$, the machine is called **deterministic**.*

Here, we have written A_L in place of $A \cup \{L\}$. Often, we use $\mathtt{L}$ or even $\Box$ as particular blanks. What this describes physically is a machine that has a two–sided infinite tape (which we can think of as a function $\tau \colon \mathbb{Z} \to A_L$), with a read/write head positioned on one of the cells. A **computation step** is as follows. Suppose the machine scans the symbol a in state q and is on cell $i \in \mathbb{Z}$. Then if $\langle b, 1, q' \rangle \in f(a,q)$, the machine may write b in place of a, advance to cell $i+1$ and change to state q'. If $\langle b, 0, q' \rangle \in f(a,q)$ the machine may write b in place of a, stay in cell i and change to state q'. Finally, if $\langle b, -1, q' \rangle \in f(a,q)$, the machine may write b in place of a, move to cell $i-1$ and switch to state q'. Evidently, in order to describe the process we need (i) the tape, (ii) the position of the head of that tape, (iii) the state the machine is currently in. We assume throughout that the tape is almost everywhere filled by a blank. (The locution 'almost all' and 'almost everywhere' is often used

in place 'all but finitely many' and 'all but finitely many places', respectively.) This means that the content of the tape plus the information on the machine may be coded by a single string, called *configuration*. Namely, if the tape is almost everywhere filled by a blank, there is a unique interval $[m,n]$ which contains all non–blank squares and the head of the machine. Suppose that the machine head is on Tape ℓ. Then let $\vec{x}_1$ be the string defined by the interval $[m,\ell-1]$ (it may be empty), and $\vec{x}_2$ the string defined by the interval $[\ell,n]$. Finally, assume that the machine is in state q. Then the string $\vec{x}_1{}^\frown q^\frown \vec{x}_2$ is the configuration corresponding to that phyical configuration. So, the state of the machine is simply written behind the symbol of the cell that is being scanned. (Obviously, A and Q are assumed to be disjoint.)

Definition 1.84 *Let $T = \langle A, \mathtt{L}, Q, q_0, f\rangle$ be a Turing machine. A T–**configuration** is a string $\vec{x}q\vec{y} \in A_L^* \times Q \times A_L^*$ such that $\vec{x}$ does not begin and $\vec{y}$ does not end with a blank.*

This configuration corresponds to a situation that the tape is almost empty (that is, almost all occurrences of symbols on it are blanks). The nonempty part is a string $\vec{x}$, with the head being placed somewhere behind the prefix $\vec{u}$. Since $\vec{x} = \vec{u}\vec{v}$ for some $\vec{v}$, we insert the state the machine is in between $\vec{u}$ and $\vec{v}$. The configuration omits most of the blanks, whence we have agreed that $\vec{u}q\vec{v}$ is the same configuration as $\Box\vec{u}q\vec{v}$ and the same $\vec{u}q\vec{v}\Box$.

We shall now describe the working of the machine using configurations. We say, $\vec{x}^\frown q^\frown \vec{y}$ is **transformed by T in one step into** $\vec{x}_1{}^\frown q_1{}^\frown \vec{y}_1$ and write $\vec{x}^\frown q^\frown \vec{y} \vdash_T \vec{x}_1{}^\frown q_1{}^\frown \vec{y}_1$ if one of the following holds.

① $\vec{x}_1 = \vec{x}$, and for some $\vec{v}$ and b and c we have $\vec{y} = b^\frown \vec{v}$ and $\vec{y}_1 = c^\frown \vec{v}$, as well as $\langle c, 0, q_1\rangle \in f(b,q)$.

② We have $\vec{x}_1 = \vec{x}^\frown c$ and $\vec{y} = b^\frown \vec{y}_1$ as well as $\langle c, 1, q_1\rangle \in f(b,q)$.

③ We have $\vec{x} = \vec{x}_1{}^\frown c$ and $\vec{y}_1 = b^\frown \vec{y}$ as well as $\langle c, -1, q_1\rangle \in f(b,q)$.

Now, for T–configurations Z and Z' we define $Z \vdash_T^n Z'$ inductively by (a) $Z \vdash_T^0 Z'$ iff $Z = Z'$ and (b) $Z \vdash_T^{n+1} Z'$ iff for some Z'' we have $Z \vdash_T^n Z'' \vdash_T Z'$.

It is easy to see that we can define a semi Thue system on configurations that mimicks the computation of T. The canonical Thue system, $C(T)$, is shown in Table 2. (x and y range over A_L and q and q' over Q.) Notice that we have to take care not to leave a blank at the left and right end of the strings. This is why the definition is more complicated than expected. The

Table 2. The Canonical Thue System

$$
\begin{aligned}
C(T) := \quad & \{\langle \vec{u}qx\vec{v}, \vec{u}yq'\vec{v}\rangle : \langle y,1,q'\rangle \in f(x,q); \vec{u} \neq \varepsilon \text{ or } y \in A; \\
& \qquad\qquad \vec{v} \neq \varepsilon \text{ or } x \in A\} \\
\cup \quad & \{\langle \vec{u}q, \vec{u}yq'\rangle : \langle y,1,q'\rangle \in f(\Box,q); \vec{u} \neq \varepsilon \text{ or } y \in A\} \\
\cup \quad & \{\langle qx\vec{v}, q'\vec{v}\rangle : \langle \Box,1,q'\rangle \in f(x,q); \vec{v} \neq \varepsilon \text{ or } x \in A\} \\
\cup \quad & \{\langle q, q'\rangle : \langle \Box,\alpha,q'\rangle \in f(\Box,q), \alpha \in \{-1,0,1\}\} \\
\cup \quad & \{\langle \vec{u}xq\vec{v}, \vec{u}q'y\vec{v}\rangle : \langle y,-1,q'\rangle \in f(x,q); \\
& \qquad\qquad \vec{u} \neq \varepsilon \text{ or } x \in A; \vec{v} \neq \varepsilon \text{ or } y \in A\} \\
\cup \quad & \{\langle q\vec{v}, q'y\vec{v}\rangle : \langle y,-1,q'\rangle \in f(\Box,q); \vec{v} \neq \varepsilon \text{ or } y \in A\} \\
\cup \quad & \{\langle \vec{u}xq, \vec{u}q'\rangle : \langle \Box,-1,q'\rangle \in f(x,q); \vec{u} \neq \varepsilon \text{ or } x \in A\} \\
\cup \quad & \{\langle \vec{u}qx\vec{v}, \vec{u}q'y\vec{v}\rangle : \langle y,0,q'\rangle \in f(x,q); \\
& \qquad\qquad \vec{v} \neq \varepsilon \text{ or } x,y \in A\}
\end{aligned}
$$

alphabet of the semi Thue system is $(Q \cup A_L)^*$. The following is easily shown by induction.

Proposition 1.85 *Let T be a Turing machine, $C(T)$ be its associated semi Thue system. Then for all T–configurations Z and Z' and for all $n > 0$: $Z \vdash_T^n Z'$ iff $Z \Rightarrow_{C(T)}^n Z'$. Moreover, if Z is a T–configuration and $Z \Rightarrow_{C(T)}^n \vec{u}$ for an arbitrary string $\vec{u} \in (Q \cup A_L)^*$, then $\vec{u}$ is a T–configuration and $Z \vdash_T^n \vec{u}$.*

Of course, the semi Thue system defines transitions on strings that are not configurations, but this is not relevant for the theorem.

Definition 1.86 *Let T be a Turing machine, Z a configuration and $\vec{x} \in A^*$. Z is called an **end configuration** if there is no configuration Z' such that $Z \vdash_T Z'$. T **accepts** $\vec{x}$ if there is an end configuration Z such that $q_0{}^\frown\vec{x} \vdash_T^* Z$. The **language accepted by** T, $L(T)$, is the set of all strings from A^* which are accepted by T.*

It takes time to get used to the concept of a Turing machine and the languages that are accepted by such machines. We suggest to the interested reader to play a little while with these machines and see if he can program them to compute a few very easy functions. A first example is the machine which computes the successor function on binary strings. Assume our alphabet is $\{0,1\}$. We want to build a machine which computes the next string for $\vec{x}$ in the numerical encoding (see Section 1.2 for its definition). This means that if

Table 3. The Successor Machine

q_0	0	$\mapsto$	$\langle 0, 1, q_0 \rangle$
	1	$\mapsto$	$\langle 1, 1, q_0 \rangle$
	$\square$	$\mapsto$	$\langle \square, -1, q_1 \rangle$
q_1	0	$\mapsto$	$\langle 1, -1, q_2 \rangle$
	1	$\mapsto$	$\langle 0, -1, q_1 \rangle$
	$\square$	$\mapsto$	$\langle 0, -1, q_3 \rangle$
q_2	0	$\mapsto$	$\langle 0, -1, q_2 \rangle$
	1	$\mapsto$	$\langle 1, -1, q_2 \rangle$
	$\square$	$\mapsto$	$\langle \square, 0, q_3 \rangle$
q_3			

the machine starts with $q_0{}^\frown\vec{x}$ it shall halt in the configuration $q_0{}^\frown\vec{y}$ where $\vec{y}$ is the word immediately following $\vec{x}$ in the numerical ordering. (If in the sequel we think of numbers rather than strings we shall simply think instead of the string $\vec{x}$ of the number n, where $\vec{x}$ occupies the nth place in the numerical ordering.)

How shall such a machine be constructed? We need four states, q_i, $i < 4$. First, the machine advances the head to the right end of the string, staying in q_0 until it reads $\square$. Finally, when it hits $\square$, it changes to state q_1 and starts moving to the left. As long as it reads 1, it changes 1 to 0 and continues in state q_1, moving to the left. When it hits 0, it replaces it by 1, moves left and changes to state q_2. When it sees a blank, that blank is filled by 0 and the machine changes to state q_3, the final state. In q_2, the machine simply keeps moving leftwards until it hits a blanks and then stops in state q_3. The machine is shown in Table 3. (If you want a machine that computes the successor in the binary encoding, you have to replace Line 6 by $\square \mapsto \langle 1, -1, q_3 \rangle$.) In recursion theory the notions of computability are defined for functions on the set of natural numbers. By means of the function Z, which is bijective, these notions can be transferred to functions on strings.

Definition 1.87 *Let A and B be alphabets and $f\colon A^* \to B^*$ a function. f is called* ***computable*** *if there is a deterministic Turing machine T such that for every $\vec{x} \in A^*$ there is a $q_t \in Q$ such that $q_0{}^\frown\vec{x} \vdash_T^* q_t{}^\frown f(\vec{x})$ and $q_t{}^\frown f(\vec{x})$ is an end configuration. Let $L \subseteq A^*$. L is called* ***recursively enumerable*** *if $L = \varnothing$ or there is a computable function $f\colon \{0,1\}^* \to A^*$ such that $f[\{0,1\}^*] = L$.*

L is ***decidable*** *if both L and $A^* - L$ are recursively enumerable.*

Lemma 1.88 *Let $f\colon A^* \to B^*$ and $g\colon B^* \to C^*$ be computable functions. Then $g \circ f\colon A^* \to C^*$ is computable as well.*

The proof is a construction of a machine U from machines T and T' computing f and g, respectively. Simply write T and T' using disjoint sets of states, and then take the union of the transition functions. However, make the transition function of T first such that it changes to the starting state of T' as soon as the computation by T is finished (that is, whenever T does not define any transitions).

Lemma 1.89 *Let $f\colon A^* \to B^*$ be computable and bijective. Then $f^{-1}\colon B^* \to A^*$ also is computable (and bijective).*

Write a machine that generates all strings of A^* in successive order (using the successor machine, see above), and computes $f(\vec{x})$ for all these strings. As soon as the target string is found, the machine writes $\vec{x}$ and deletes everything else.

Lemma 1.90 *Let A and B be finite alphabets. Then there are computable bijections $f\colon A^* \to B^*$ and $g\colon B^* \to A^*$ such that $f = g^{-1}$.*

In this section we shall show that the recursively enumerable sets are exactly the sets which are accepted by a Turing machine. Further, we shall show that these are exactly the Type 0 languages. This establishes the first correspondence result between types of languages and types of automata. Following this we shall show that the recognition problem for Type 0 languages is in general not decidable. The proofs proceed by a series of reduction steps for Turing machines. First, we shall generalize the notion of a Turing machine. A k–**tape Turing machine** is a quintuple $\langle A, L, Q, q_0, f\rangle$ where A, L, Q, and q_0 are as before but now

$$f\colon A_L^k \times Q \to \wp(A_L^k \times \{-1, 0, 1\} \times Q) \tag{1.116}$$

This means, intuitively speaking, that the Turing machine manipulates k tapes in place of a single tape. There is a read and write head on each of the tapes. In each step the machine can move only one of the heads. The next state depends on the symbols read on all the tapes plus the current internal state. The initial configuration is as follows. All tapes except the first are empty.

The heads are anywhere on these tapes (we may require them to be in position 0). On the first tape the head is immediately to the left of the input. The k–tape machine has $k-1$ additional tapes for recording intermediate results. The reader may verify that we may also allow such configurations as initial configurations in which the other tapes are filled with some finite string, with the head immediately to the left of it. This does not increase the recognition power. However, it makes the definition of a machine easier which computes a function of several variables. We may also allow that the information to the right of the head consists in a sequence of strings each separated by a blank (so that when two successive blanks follow the machine knows that the input is completely read). Again, there is a way to recode these machines using a basic multitape Turing machine, modulo computable functions. We shall give a little more detail concerning the fact that also k–tape Turing machines (in whatever of the discussed forms) cannot compute more functions than 1–tape machines. For this define the following coding of the k tapes using a single tape. We shall group $2k$ cells together to a macro cell. The (micro) cell $2kp+2m$ corresponds to the entry on cell p on Tape m. The (micro) cell number $2kp+2m+1$ only contains `1` or `0` depending on whether the head of the machine is placed on cell p on tape m. (Hence, every second micro cell is filled only with `1` or `0`.) Now given a k–tape Turing machine T we shall define a machine U that simulates T under the given coding. This machine operates as follows. For a single step of T it scans the actual string for the positions of the read and write heads and remembers the symbols on which they are placed (they can be found in the adjacent cell). Remembering this information requires only finite amount of memory, and can be done using the internal states. The machine scans the tape again for the head that will have to be changed in position. (To identify it, the machine must be able to do calculations modulo $2k$. Again finite memory is sufficient.) It adjusts its position and the content of the adjacent cell. Now it changes into the appropriate state. Notice that each step of T costs $2k \cdot |\vec{x}|$ time for U to simulate, where $\vec{x}$ is the longest string on the tapes. If there is an algorithm taking $f(n)$ steps to compute then the simulating machine needs at most $2k(f(n)+n)^2$ time to compute that same function under simulation. (Notice that in $f(n)$ steps the string(s) may acquire length at most $f(n)+n$.)

We shall use this to show that the nondeterministic Turing machines cannot compute more functions than the deterministic ones.

Proposition 1.91 *Let $L = L(T)$ for a Turing machine. Then there is a deter-*

ministic Turing machine U such that $L = L(U)$.

Proof. Let $L = L(T)$. Choose a number b such that $|f(q,x)| < b$ for all $q \in Q$, $x \in A$. We fix an ordering on $f(q,x)$ for all x and q. V is a 3–tape machine that does the following. On the first tape V writes the input $\vec{x}$. On the second tape we generate all sequences $\vec{p}$ of numbers $< b$ of length n, for increasing n. These sequences describe the action sequences of T. For each sequence $\vec{p} = a_0 a_1 \cdots a_{n-1}$ we copy $\vec{x}$ from Tape 1 onto Tape 3 and let V work as follows.

The head on Tape 2 is to the left of the sequence $\vec{a}$. In the first step V follows the a_0th alternative for machine T on the 3rd tape and advances head number 2 one step to the right. In the second step it follows the alternative a_1 in the transition set of T and executes it on Tape 3. Then the head of Tape 2 is advanced one step to the right. If $a_{n-1} < b$ and the a_{n-1}st alternative does not exist for T but there is a computation for $a_0 a_1 \cdots a_{n-2} a'$ for some $a' < a_{n-1}$, V exits the computation on Tape 3 and deletes $\vec{p}$ on Tape 2. If $a_{n-1} = b$, the a_{n-1}st alternative does not exist for T, and none exists for any $a' < b$, then V halts. In this way V executes on Tape 3 a single computation of T for the input and checks the prefixes for paths for which a computation exists. Clearly, V is deterministic. It halts iff for some n T halts on some alternative sequences of length $n-1$. □

It is easy to see that we can also write a machine that enumerates all possible outputs of T for a given input.

Lemma 1.92 *L is recursively enumerable iff $L = L(T)$ for a Turing machine T.*

Proof. The case $L = \varnothing$ has to be dealt with separately. It is easy to construct a machine that halts on no word. This shows the equivalence in this case. Now assume that $L \neq \varnothing$. Let L be recursively enumerable. Then there exists a function $f\colon \{0,1\}^* \to A^*$ such that $f[\{0,1\}^*] = L$ and a Turing machine U which computes f. Now we construct a (minimally) 3–tape Turing machine V as follows. The input $\vec{x}$ will be placed on the first tape. On the second tape V generates all strings $\vec{y} \in \{0,1\}^*$ starting with ε, in the numerical order. In order to do this we use the machine computing the successors in this ordering. If we have computed the string $\vec{y}$ on the second tape the machine computes the value $f(\vec{y})$ on the third tape. (Thus, we emulate machine T on the third tape, with input given on the second tape.) Since f is computable, V halts on Tape 3. Then it compares the string on Tape 3, $f(\vec{y})$, with $\vec{x}$. If they are equal, it halts, if not it computes the successor of $\vec{y}$ and starts the process over again.

It is easy to see that $L = L(V)$. By the previous considerations, there is a one tape Turing machine W such that $L = L(W)$. Now conversely, let $L = L(T)$ for some Turing machine T. We wish to show that L is recursively enumerable. We may assume, by the previous theorem, that T is deterministic. We leave it to the reader to construct a machine U which computes a function $f\colon \{\mathtt{0},\mathtt{1}\}^* \to A^*$ whose image is L. □

Theorem 1.93 *The following are equivalent.*

① *L is of Type 0.*

② *L is recursively enumerable.*

③ *$L = L(T)$ for a Turing machine T.*

Proof. We shall show ① ⇒ ② and ③ ⇒ ①. The theorem then follows with Lemma 1.92. Let L be of Type 0. Then there is a grammar $\langle \mathtt{S}, N, A, R\rangle$ which generates L. We have to construct a Turing machine which lists all strings that are derivable from $\mathtt{S}$. To this end it is enough to construct a nondeterministic machine that matches the grammar. This machine always starts at input $\mathtt{S}$ and in each cycle it scans the string for a left hand side of a rule and replaces that substring by the right hand side. This shows ②. Now let $L = L(T)$ for some Turing machine. Choose the following grammar G: in addition to the alphabet let $\mathtt{X}$ be the start symbol, $\mathtt{0}$ and $\mathtt{1}$ two nonterminals, and let each $q \in Q$ $\mathtt{Y}_q$ be a nonterminal. The rules are as follows.

$$
\begin{aligned}
\mathtt{X} &\to \mathtt{X0} \mid \mathtt{X1} \mid \mathtt{Y}_{q_0} && \\
\mathtt{Y}_q b &\to c\mathtt{Y}_r && \text{if } \langle c,1,r\rangle \in f(b,q) \\
\mathtt{Y}_q b &\to \mathtt{Y}_r c && \text{if } \langle c,0,r\rangle \in f(b,q) \\
\mathtt{Y}_q b &\to \mathtt{Y}_r cb && \text{if } \langle c,-1,r\rangle \in f(b,q) \\
\mathtt{Y}_q b &\to b && \text{if } f(b,q) = \varnothing
\end{aligned}
\tag{1.117}
$$

Starting with $\mathtt{X}$ this grammar generates strings of the form $\mathtt{Y}_{q_0}\vec{x}$, where $\vec{x}$ is a binary string. This codes the input for T. The additional rules code in a transparent way the computation of T on the string. If the computation stops, it is allowed to eliminate $\mathtt{Y}_q$. If the string is terminal it will be generated by G. In this way it is seen that $L(G) = L(T)$. □

Now we shall derive an important fact, namely that there exist undecidable languages of Type 0. We first of all note that Turing machines can be regarded

as semi Thue systems, as we have done earlier. Now one can design a machine U which takes two inputs, one being the code of a Turing machine T and the other a string $\vec{x}$, and U computes what T computes on $\vec{x}$. Such a machine is called a **universal Turing machine**. The coding of Turing machines can be done as follows. We only use the letters a, b and c, which are, of course, also contained in the alphabet B. Let $A = \{\mathtt{a}_i : i < n\}$. Then let $\gamma(\mathtt{a}_i)$ be the number i in dyadic coding (over $\{\mathtt{a}, \mathtt{b}\}$, where a replaces 0 and b replaces 1). The number 0 is coded by a to distinguish it from ε. Furthermore, we associate the number n with the blank, L. The states are coded likewise; we assume that $Q = \{0, 1, \ldots, n-1\}$ for some n and that $q_0 = 0$. Now we still have to write down f. f is a subset of

$$A_{\mathrm{L}} \times Q \times A_{\mathrm{L}} \times \{-1, 0, 1\} \times Q \tag{1.118}$$

Each element $\langle a, q, b, m, r\rangle$ of f can be written down as

$$\vec{x}^\frown \mathtt{c}^\frown \vec{u}^\frown \mathtt{c}^\frown \vec{\mu}^\frown \mathtt{c}^\frown \vec{y}^\frown \mathtt{c}^\frown \vec{v}^\frown \mathtt{c} \tag{1.119}$$

where $\vec{x} = \gamma(a)$, $\vec{u} = Z^{-1}(q)$, $\vec{y} = \gamma(b)$, $\vec{v} = Z^{-1}(r)$. Further, we have $\vec{\mu} = \mathtt{a}$ if $m = -1$, $\vec{\mu} = \mathtt{b}$ if $m = 0$ and $\vec{\mu} = \mathtt{ab}$ if $m = 1$. Now we simply write down f as a list, the entries being separated by cc. (This is not necessary, but is easier to handle.) We call the code of T $T^{\spadesuit}$. The set of all codes of Turing machines is decidable. (This is essential but not hard to see.) It should not be too hard to see that there is a machine U with two tapes, which for two strings $\vec{x}$ and $\vec{y}$ does the following. If $\vec{y} = T^{\spadesuit}$ for some T then U computes on $\vec{x}$ exactly as T does. If $\vec{y}$ is not the code of a machine, U moves into a special state and stops.

Suppose that there is a Turing machine V which decides for given $\vec{x}$ and $T^{\spadesuit}$ wether or not $\vec{x} \in L(T)$. Now we construct a two tape machine W as follows. The input is $\vec{x}$, and it is given on both tapes. If $\vec{x} = T^{\spadesuit}$ for some T then W computes T on $\vec{x}$. (This is done by emulating V.) If T halts on $\vec{x}$, we send W into an infinite loop. If T does not halt, W shall stop. (If $\vec{x}$ is not the code of a machine, the computation stops right away.) Now we have the following: $W^{\spadesuit} \in L(W)$ exactly if $W^{\spadesuit} \notin L(W)$. For $W^{\spadesuit} \in L(W)$ exactly when W stops if applied to $W^{\spadesuit}$. This however is the case exactly if W does *not* stop. If on the other hand $W^{\spadesuit} \notin L(W)$ then W does not stop if applied to $W^{\spadesuit}$, which we can decide with the help of machine V, and then W does halt on the input $W^{\spadesuit}$. Contradiction. Hence, V cannot exist. There is, then, no machine that can decide for any Turing machine (in code) and any input whether that machine halts on that string. It is still conceivable that this is decidable for

every T, but that we simply do not know how to extract such an algorithm for given T. Now, in order to show that this too fails, we use the universal Turing machine U, in its single tape version. Suppose that $L(U)$ is decidable. Then we can decide whether U halts on $\vec{x}^\frown \mathtt{L}^\frown T^\spadesuit$. Since U is universal, this means that we can decide for given T and given $\vec{x}$ whether T halts on $\vec{x}$. We have seen above that this is impossible.

Theorem 1.94 (Markov, Post) *There is a recursively enumerable set which is not decidable.*

So we also shown that the Type 1 languages are properly contained in the Type 0 languages. For it turns out that the Type 1 languages are all decidable.

Theorem 1.95 (Chomsky) *Every Type 1 language is decidable.*

Proof. Let G be of Type 1 and let $\vec{x}$ be given. Put $n := |\vec{x}|$ and $\alpha := |A \cup N|$. If there is a derivation of $\vec{x}$ that has length $> \alpha^n$, there is a string that occurs twice in it, since all occurring strings must have length $\leq n$. Then there exists a shorter derivation for $\vec{x}$. So, $\vec{x} \in L(G)$ iff it has a G–derivation of length $\leq \alpha^n$. This is decidable. $\square$

Corollary 1.96 CSL $\subsetneq$ GL.

Chomsky [**?**] credits Hilary Putnam with the observation that not all decidable languages are of Type 1. Actually, we can give a characterization of context sensitive languages as well. Say that a Turing machine is **linearly space bounded** if given input $\vec{x}$ it may use only $O(|\vec{x}|)$ on each of its tapes. Then the following holds.

Theorem 1.97 (Landweber, Kuroda) *A language L is context sensitive iff $L = L(T)$ for some linear space bounded Turing machine T.*

The proof can be assembled from Theorem 1.65 and the proof of Theorem 1.93.

We briefly discuss so–called *word problems*. Recall from Section 1.5 the definition of a Thue process T. Let A be an alphabet. Consider the monoid $\mathfrak{Z}(A)$. The set of pairs $\langle s,t\rangle \in A^* \times A^*$ such that $s \Rightarrow_T^* t$ is a congruence on $\mathfrak{Z}(A)$. Denote the factor algebra by $\mathfrak{Mon}(T)$. (One calls the pair $\langle A,T\rangle$ a **presentation** of $\mathfrak{Mon}(T)$.) It can be shown to be undecidable whether $\mathfrak{Mon}(T)$ is the one element monoid. From this one deduces that it is undecidable whether

or not $\mathfrak{Mon}(T)$ is a finite monoid, whether it is isomorphic to a given finite monoid, and many more.

Before we close this chapter we shall introduce a few measures for the complexity of computations. In what is to follow we shall often have to deal with questions of how fast and with how much space a Turing machine can compute a given problem. Let $f\colon \omega \to \omega$ be a function, T a Turing machine which computes a function $g\colon A^* \to B^*$. We say that T needs $O(f)$–**space** if there is a constant c such that for all but finitely many $\vec{x} \in A^*$ there is a computation of an accepting configuration $q_t{}^\frown g(\vec{x})$ from $q_0{}^\frown \vec{x}$ in which every configuration has length $\leq c \times f(|\vec{x}|)$. For a multi tape machine we simply add the lengths of all words on the tapes. We say that T needs $O(f)$–**time** if for almost all $\vec{x} \in A^*$ there is a $k \leq c \times f(|\vec{x}|)$ such that $q_0{}^\frown \vec{x} \vdash_T^k q_0{}^\frown g(\vec{x})$. We denote by $\mathbf{DSPACE}(f)$ ($\mathbf{DTIME}(f)$) the set of all functions which for some k are computable by a deterministic k–tape Turing machine in $O(f)$–space ($O(f)$–time). Analogously the notation $\mathbf{NSPACE}(f)$ and $\mathbf{NTIME}(f)$ is defined for nondeterministic machines. We always have

$$\mathbf{DTIME}(f) \subseteq \mathbf{NTIME}(f) \subseteq \mathbf{NSPACE}(f) \tag{1.120}$$

as well as

$$\mathbf{DSPACE}(f) \subseteq \mathbf{NSPACE}(f) \tag{1.121}$$

For a machine can fill at most k cells in k steps, regardless of whether it is deterministic or nondeterministic. This applies as well to multi tape machines, since they can only write on one cell and move one head at a time.

The reason for not distinguishing between the time complexity $f(n)$ and the $cf(n)$ (c a constant) is the following result.

Theorem 1.98 (Speed Up Theorem) *Let f be a computable function and let T be a Turing machine which computes $f(\vec{x})$ in at most $g(|\vec{x}|)$ steps (using at most $h(|\vec{x}|)$ cells) where $\inf_{n\to\infty} g(n)/n = \infty$. Further, let c be an arbitrary real number > 0. Then there exists a Turing machine U which computes f in at most $c \cdot g(|\vec{x}|)$ steps (using at most $c \cdot h(|\vec{x}|)$ cells).*

The proof results from the following fact. In place of the original alphabet A_L we may introduce a new alphabet $B'_{L'} := A \cup B \cup \{L'\}$, where each symbol from B corresponds to a sequence of length k of symbols from A_L. The symbol L' then corresponds to L^k. The alphabet A_L is still used for giving the

input. The new machine, upon receiving $\vec{x}$ recodes the input and calculates completely inside B'_L.

Since to each single letter corresponds a block of k letters in the original alphabet, the space requirement shrinks by the factor k. (However, we need to ignore the length of the input.) Likewise, the time is cut by a factor k, since one move of the head simulates up to k moves. However, the exact details are not so easy to sum up. They can be found in [**?**].

Typically, one works with the following complexity classes.

Definition 1.99 ***PTIME*** *is the class of functions computable in deterministic polynomial time,* ***NP*** *the class of functions computable in nondeterministic polynomial time.* ***PSPACE*** *is the class of functions computable in polynomial space,* ***EXPTIME (NEXPTIME)*** *the class of functions computable in deterministic (nondeterministic) exponential time.*

Definition 1.100 *A language $L \subseteq A^*$ is in a complexity class $\mathcal{P}$ iff $\chi_L \in \mathcal{P}$.*

Notes on this section. In the mid 1930s, several people have independently studied the notion of feasibility. Alonzo Church and Stephen Kleene have defined the notion of λ–definablity and of a general recursive function, Emil Post and Alan Turing the notion of computability by a certain machine, now called the Turing machine. All three notions can be shown to identify the same class of functions, as these people have subsequently shown. It is known as Church's Thesis that these are all the functions that humans can compute, but for the purpose of this book it is irrelevant whether it is correct. We shall define the λ–calculus later in Chapter 3, without going into the details alluded to here, however. It is to be kept in mind that the Turing machine is a physical device. Hence, its computational capacities depend on the structure of the space–time continuum. This is not any more a speculation. Quantum computing exploits the different physical behaviour of quantum physics to do parallel computation. This radically changes the time complexity of problems (see [**?**]). This asks us to be cautious not to attach too much significance to complexity results in connection with human behaviour since we do not know too well how the brain works.

Exercise 40. Construct a Turing machine which computes the lexicographic predecessor of a string, and which returns ε for input ε.

Exercise 41. Construct a Turing machine which, given a list of strings (each

string separated from the next by a single blank), moves the first string onto the end of the list.

Exercise 42. Let T be a Turing machine over A. Show how to write a Turing machine over $\{\mathtt{0},\mathtt{1}\}$ which computes the same partial function over A under a coding that assigns each letter of A a unique block of fixed length.

Exercise 43. In many definitions of a Turing machine the tape is only one sided. Its cells can be numbered by natural numbers. This requires the introduction of a special symbol # that marks the left end of the tape, or of a predicate left-end, which is true each time the head is at the left end of the tape. The transitions are different depending on whether the machine is at the left end of the tape or not. (There is an alternative, namely to stop the computation once that the left end is reached, but this is not recommended. Such a machine can compute only very uninteresting functions.) Show that for a Turing machine with a one sided tape there is a corresponding Turing machine in our sense computing the same function, and that for each Turing machine in our sense there is a one sided machine computing the same function.

Exercise 44. Prove Lemma 1.90. *Hint.* Show first that it is enough to look at the case $|A| = 1$.

Exercise 45. Show that $L \subseteq A^*$ is decidable iff $\chi_L : A^* \to \{\mathtt{0},\mathtt{1}\}$ is computable.

Chapter 2
Context Free Languages

1. Regular Languages

Type 3 or regular grammars are the most simple grammars in the Chomsky Hierarchy. There are several characterizations of regular languages: by means of finite state automata, by means of equations over strings, and by means of so–called regular expressions. Before we begin, we shall develop a simple form for regular grammars. First, all rules of the form $X \to Y$ can be eliminated. To this end, the new set of rules will be

$$\begin{aligned} R^{\heartsuit} := \quad & \{X \to aY : X \vdash_G aY\} \\ & \cup \{X \to \vec{x} : X \vdash_G \vec{x}, \vec{x} \in A_\varepsilon\} \end{aligned} \tag{2.1}$$

It is easy to show that the grammar with $R^{\heartsuit}$ in place of R generates the same strings. We shall introduce another simplification. For each $a \in A$ we introduce a new nonterminal U_a. In place of the rules $X \to a$ we now add the rules $X \to aU_a$ as well as $U_a \to \varepsilon$. Now every rule with the exception of $U_a \to \varepsilon$ is strictly expanding. This grammar is therefore not regular if $\varepsilon \in L(G)$ but it generates the same language. However, the last kind of rules can be used only once, at the end of the derivation. For the derivable strings all have the form $\vec{x}^\frown Y$ with $\vec{x} \in A^*$ and $Y \in N$. If one applies a rule $Y \to \varepsilon$ then the nonterminal disappears and the derivation is terminated. We call a regular grammar **strictly binary** if there are only rules of the form $X \to aY$ or $X \to \varepsilon$.

Definition 2.1 *Let A be an alphabet. A **(partial) finite state automaton** is a quintuple $\mathfrak{A} = \langle A, Q, i_0, F, \delta \rangle$ such that Q is a finite set, $i_0 \in Q$, $F \subseteq Q$ and $\delta\colon Q \times A \to \wp(Q)$. Q is the set of **states**, i_0 is called **the initial state**, F the set of **accepting states** and δ the **transition function**. $\mathfrak{A}$ is called **deterministic** if $\delta(q,a)$ contains exactly one element for each $q \in Q$ and $a \in A$.*

δ can be extended to sets of states and strings in the following way ($S \subseteq Q$, $a \in A$).

$$\delta(S, \varepsilon) := S \tag{2.2a}$$

$$\delta(S, a) := \bigcup \langle \delta(q,a) : q \in S \rangle \tag{2.2b}$$

$$\delta(S, \vec{x}^\frown a) := \delta(\delta(S, \vec{x}), a) \tag{2.2c}$$

With this defined, we can now define the accepted language.

$$L(\mathfrak{A}) = \{\vec{x} : \delta(\{i_0\}, \vec{x}) \cap F \neq \varnothing\} \tag{2.3}$$

$\mathfrak{A}$ is strictly partial if there is a state q and some $a \in A$ such that $\delta(q,a) = \varnothing$. An automaton can always be transformed into an equivalent automaton which is not partial. Just add another state $q_\varnothing$ and add to the transition function the following transitions.

$$\delta^+(q,a) := \begin{cases} \delta(q,a) & \text{if } \delta(q,a) \neq \varnothing \text{ and } q \neq q_\varnothing, \\ q_\varnothing & \text{if } \delta(q,a) = \varnothing \text{ or } q = q_\varnothing. \end{cases} \tag{2.4}$$

Furthermore, $q_\varnothing$ shall *not* be an accepting state. In the case of a deterministic automaton we have $\delta(q,\vec{x}) = \{q'\}$ for some q'. In this case we think of the transition function as yielding states from states plus strings, that is, we now have $\delta(q,\vec{x}) = q'$. Then the definition of the language of an automaton $\mathfrak{A}$ can be refined as follows.

$$L(\mathfrak{A}) = \{\vec{x} : \delta(i_0, \vec{x}) \in F\} \tag{2.5}$$

For every given automaton there is a deterministic automaton that accepts the same language. Put

$$\mathfrak{A}^d := \langle A, \wp(Q), \{i_0\}, F^d, \delta \rangle \tag{2.6}$$

where $F^d := \{G \subseteq Q : G \cap F \neq \varnothing\}$ and δ is the transition function of $\mathfrak{A}$ extended to sets of states.

Proposition 2.2 *$\mathfrak{A}^d$ is deterministic and $L(\mathfrak{A}^d) = L(\mathfrak{A})$. Hence every language accepted by a finite state automaton is a language accepted by a deterministic finite state automaton.*

The proof is straightforward and left as an exercise. Now we shall first show that a regular language is a language accepted by a finite state automaton. We may assume that G is (almost) strictly binary, as we have seen above. So, let $G = \langle \mathtt{S}, N, A, R \rangle$. We put $Q_G := N$, $i_0 := \mathtt{S}$, $F_G := \{X : X \to \varepsilon \in R\}$ as well as

$$\delta_G(X,a) := \{Y : X \to aY \in R\} \tag{2.7}$$

Now put $\mathfrak{A}_G := \langle A, Q_G, i_0, F_G, \delta_G \rangle$.

Lemma 2.3 *For all $X, Y \in N$ and $\vec{x}$ we have $Y \in \delta(X, \vec{x})$ iff $X \Rightarrow_R^* \vec{x}^\frown Y$.*

Proof. Induction over the length of $\vec{x}$. The case $|\vec{x}| = \varepsilon$ is evident. Let $\vec{x} = a \in A$. Then $Y \in \delta_G(X, a)$ by definition iff $X \to aY \in R$, and from this we get $X \Rightarrow_R^* aY$. Conversely, from $X \Rightarrow_R^* aY$ follows that $X \to aY \in R$. For since the derivation uses only strictly expanding rules except for the last step, the derivation of aY from X must be the application of a single rule. This finishes the case of length 1. Now let $\vec{x} = \vec{y}^\frown a$. By definition of δ_G we have

$$\delta_G(X, \vec{x}) = \delta_G(\delta_G(X, \vec{y}), a) \tag{2.8}$$

Hence there is a Z such that $Z \in \delta_G(X, \vec{y})$ and $Y \in \delta_G(Z, a)$. By induction hypothesis this is equivalent with $X \Rightarrow_R^* \vec{y}^\frown Z$ and $Z \Rightarrow_R^* aY$. From this we get $X \Rightarrow_R^* \vec{y}^\frown a^\frown Y = \vec{x}^\frown Y$. Conversely, from $X \Rightarrow_R^* \vec{x}^\frown Y$ we get $X \Rightarrow_R^* \vec{y}^\frown Z$ and $Z \Rightarrow_R^* aY$ for some Z, since G is regular. Now, by induction hypothesis, $Z \in \delta_G(X, \vec{y})$ and $Y \in \delta_G(Z, a)$, and so $Y \in \delta_G(\vec{x}, X)$. □

Proposition 2.4 $L(\mathfrak{A}_G) = L(G)$.

Proof. It is easy to see that $L(G) = \{\vec{x} : G \vdash \vec{x}^\frown Y, Y \to \varepsilon \in R\}$. By Lemma 2.3 $\vec{x}^\frown Y \in L(G)$ iff $S \Rightarrow_R^* \vec{x}^\frown Y$. The latter is equivalent with $Y \in \delta_G(\mathsf{S}, \vec{x})$. And this is nothing but $\vec{x} \in L(\mathfrak{A}_G)$. Hence $L(G) = L(\mathfrak{A}_G)$. □

Given a finite state automaton $\mathfrak{A} = \langle A, Q, i_0, F, \delta \rangle$ put $N_{\mathfrak{A}} := Q$, $S_{\mathfrak{A}} := i_0$. $R_{\mathfrak{A}}$ consists of all rules of the form $X \to aY$ where $Y \in \delta(X, a)$ as well as all rules of the form $X \to \varepsilon$ for $X \in F$. Finally, $G_{\mathfrak{A}} := \langle S_{\mathfrak{A}}, N_{\mathfrak{A}}, A, R_{\mathfrak{A}} \rangle$. $G_{\mathfrak{A}}$ is strictly binary and $\mathfrak{A}_{G_{\mathfrak{A}}} = \mathfrak{A}$. Therefore we have $L(G_{\mathfrak{A}}) = L(\mathfrak{A})$.

Theorem 2.5 *The regular languages are exactly those languages that are accepted by some deterministic finite state automaton.* □

Now we shall turn to a further characterization of regular languages. A **regular term over** A is a term which is composed from A with the help of the symbols 0 (0–ary), ε (0–ary), $\cdot$ (binary), $\cup$ (binary) and * (unary). A regular term defines a language over A as follows.

$$L(0) := \varnothing \tag{2.9a}$$

$$L(\varepsilon) := \{\varepsilon\} \tag{2.9b}$$

$$L(a) := \{a\} \tag{2.9c}$$

$$L(R \cdot S) := L(R) \cdot L(S) \tag{2.9d}$$

$$L(R \cup S) := L(R) \cup L(S) \tag{2.9e}$$

$$L(R^*) := L(R)^* \tag{2.9f}$$

(Commonly, one writes R in place of $L(R)$, a usage that we will follow in the sequel to this section.) Also, $R^+ := R^* \cdot R$ is an often used abbreviation. Languages which are defined by a regular term can also be viewed as solutions of some very simple systems of equations. We introduce variables (say X, Y and Z) which are variables for subsets of A^* and we write down equations for the terms over these variables and the symbols 0, ε, a ($a \in A$), $\cdot$, $\cup$ and *. An example is the equation $X = \mathtt{b} \cup \mathtt{a}X$, whose solution is $X = \mathtt{a}^*\mathtt{b}$.

Lemma 2.6 *Assume $R \neq 0$ and $\varepsilon \notin L(R)$. Then R^* is the unique solution of $X = \varepsilon \cup R \cdot X$.*

Proof. The proof is by induction over the length of $\vec{x}$. $\vec{x} \in X$ means by definition that $\vec{x} \in \varepsilon \cup R \cdot X$. If $\vec{x} = \varepsilon$ then $\vec{x} \in R^*$. Hence let $\vec{x} \neq \varepsilon$; then $\vec{x} \in R \cdot X$ and so it is of the form $\vec{u}_0{}^\frown\vec{x}_0$ where $\vec{u}_0 \in R$ and $\vec{x}_0 \in X$. Since $\vec{u}_0 \neq \varepsilon$, $\vec{x}_0$ has smaller length than $\vec{x}$. By induction hypothesis we therefore have $\vec{x}_0 \in R^*$. Hence $\vec{x} \in R^*$. The other direction is as easy. □

Lemma 2.7 *Let C, D be regular terms, $D \neq 0$ and $\varepsilon \notin L(D)$. The equation*

$$X = C \cup D \cdot X \tag{2.10}$$

has exactly one solution, namely $X = D^* \cdot C$. □

We shall now show that regular languages can be seen as solutions of systems of equations. A general system of string equations is a set of equations of the form $X_j = Q \cup \bigcup_{i<m} T^i$ where Q is a regular term and the T^i have the form $R \cdot X_k$ where R is a regular term. Here is an example.

$$\begin{aligned} X_0 &= \mathtt{a}^* \cup \mathtt{c} \cdot \mathtt{a} \cdot \mathtt{b} \cdot X_1 \\ X_1 &= \mathtt{c} \cup \quad \mathtt{c} \cdot \mathtt{b}^3 \cdot X_0 \end{aligned} \tag{2.11}$$

Notice that like in other systems of equations a variable need not occur to the right in every equation. Moreover, a system of equations contains any given variable only once on the left. The system is called **proper** if for all i and j we have $\varepsilon \notin L(T^i_j)$. We shall call a system of equations **simple** if it is proper and Q as well as the T^i_j consist only of terms made from elements of A using ε and $\cup$. The system displayed above is proper but not simple.

Let now $\langle \mathtt{S}, N, A, R \rangle$ be a strictly binary regular grammar. Introduce for each nonterminal X a variable Q_X. This variable Q_X shall stand for the set of all strings which can be generated from X in this grammar, that is, all strings

$\vec{x}$ for which $X \Rightarrow_R^* \vec{x}$. This latter set we denote by $[X]$. We claim that the Q_X so interpreted satisfy the following system of equations.

$$\begin{aligned} Q_Y = & \bigcup\{\varepsilon : Y \to \varepsilon \in R\} \\ & \cup \bigcup\{a \cdot Q_X : Y \to aX \in R\} \end{aligned} \tag{2.12}$$

This system of equations is simple. We show $Q_Y = [Y]$ for all $Y \in N$. The proof is by induction over the length of the string. To begin, we show that $Q_Y \subseteq [Y]$. For let $\vec{y} \in Q_Y$. Then either $\vec{y} = \varepsilon$ and $Y \to \varepsilon \in R$ or we have $\vec{y} = a^\frown\vec{x}$ with $\vec{x} \in Q_X$ and $Y \to a^\frown\vec{x} \in R$. In the first case $Y \to \varepsilon \in R$, whence $\varepsilon \in [Y]$. In the second case $|\vec{x}| < |\vec{y}|$ and so by induction hypothesis $\vec{x} \in [X]$, hence $X \Rightarrow_R^* \vec{x}$. Then we have $Y \Rightarrow_R^* a^\frown\vec{x} = \vec{y}$, from which $\vec{y} \in [Y]$. This shows the first inclusion. Now we show that $[Y] \subseteq Q_Y$. To this end let $Y \Rightarrow_R^* \vec{y}$. Then either $\vec{y} = \varepsilon$ and so $Y \to \varepsilon \in R$ or $\vec{y} = a^\frown\vec{x}$ for some $\vec{x}$. In the first case $\vec{y} \in Q_Y$, by definition. In the second case there must be an X such that $Y \to aX \in R$ and $X \Rightarrow_R^* \vec{x}$. Then $|\vec{x}| < |\vec{y}|$ and therefore by induction hypothesis $\vec{x} \in Q_X$. Finally, by definition of Q_Y, $\vec{y} \in Q_Y$, which had to be shown.

So, a regular language is the solution of a simple system of equations. Conversely, every simple system of equations can be rewritten into a regular grammar which generates the solution of this system. Finally, it remains to be shown that regular terms describe nothing but regular languages. What we shall establish is more general and derives the desired conclusion. We shall show that every proper system of equations which has as many equations as it has variables has as its solution for each variable a regular language. To this end, let such a system $X_j = \bigcup_{i<m_j} T_j^i$ be given. We begin by eliminating X_0 from the system of equations. We distinguish two cases. (1) X_0 appears in the equation $X_0 = \bigcup_{i<m_0} T_j^i$ only to the left. This equation is fixed, and called the **pivot equation for** X_0. Then we can replace X_0 in the other equations by $\bigcup_{i<m_0} T_j^i$. (2) The equation is of the form $X_0 = C \cup D \cdot X_0$, C a regular term, which does not contain X_0, D free of variables and $\varepsilon \notin L(D)$. Then $X_0 = D^* \cdot C$ by Lemma 2.7. Now X_0 does not occur and we can replace X_0 in the other equations as in (1). The system of equations that we get is not simple, even if it was simple at the beginning. We can proceed in this fashion and eliminate step by step the variables from the right hand side (and putting aside the corresponding pivot equations) until we reach the last equation. The solution for X_{n-1} does not contain any variables at all and is a regular term. The solution can be inserted into the other equations, and then we continue with X_{n-2}, then with X_{n-3}, and so on. As an example, we take the following

system of equations.

$$
\begin{array}{llrrr}
\text{(I)} & X_0 & = & \mathtt{a} \cup \mathtt{a}\cdot X_0 & \cup\, \mathtt{b}\cdot X_1 & \cup\, \mathtt{c}\cdot X_2 \\
 & X_1 & = & \mathtt{c}\cdot X_0 & & \cup\, \mathtt{a}\cdot X_2 \\
 & X_2 & = & \mathtt{b} \cup \mathtt{a}\cdot X_0 & \cup\, \mathtt{b}\cdot X_1 & \\
\\
\text{(II)} & X_0 & = & \mathtt{a}^+ & \cup\, \mathtt{a}^*\mathtt{b}\cdot X_1 & \cup\, \mathtt{a}^*\mathtt{c}\cdot X_2 \\
 & X_1 & = & \mathtt{ca}^+ & \cup\, \mathtt{ca}^*\mathtt{b}\cdot X_1 & \cup\, (\mathtt{ca}^*\mathtt{c} \cup \mathtt{a})\cdot X_2 \\
 & X_2 & = & \mathtt{b} \cup \mathtt{aa}^+ & \cup\, (\mathtt{a}^+\mathtt{b} \cup \mathtt{b})\cdot X_1 & \cup\, \mathtt{a}^*\mathtt{c}\cdot X_2 \\
\\
\text{(III)} & X_1 & = & & (\mathtt{ca}^*\mathtt{b})^*\mathtt{ca}^+ \cup (\mathtt{ca}^*\mathtt{b})^*(\mathtt{ca}^*\mathtt{c} \cup \mathtt{a})\cdot X_2 \\
 & X_2 & = & & (\mathtt{b} \cup \mathtt{aa}^+) \cup [\mathtt{a}^*\mathtt{b}(\mathtt{ca}^*\mathtt{b})^*(\mathtt{ca}^*\mathtt{c} \cup \mathtt{a}) \cup \mathtt{a}^*\mathtt{c}]\cdot X_2 \\
\\
\text{(IV)} & X_2 & = & & [\mathtt{a}^*\mathtt{b}(\mathtt{ca}^*\mathtt{b})^*(\mathtt{ca}^*\mathtt{c} \cup \mathtt{a}) \cup \mathtt{a}^*\mathtt{c}]^*(\mathtt{b} \cup \mathtt{aa}^+)
\end{array}
$$

Now that X_2 is known, X_1 can be determined by inserting the regular term for X_2, and, finally, X_0 is obtained by inserting the values for X_2 and X_1.

Theorem 2.8 (Kleene) *Let L be a language over A. Then the following are equivalent:*

① *L is regular.*

② *$L = L(\mathfrak{A})$ for a finite, deterministic automaton $\mathfrak{A}$ over A.*

③ *$L = L(R)$ for some regular term R over A.*

④ *L is the solution for X_0 of a simple system of equations over A with variables X_i, $i < m$.*

Further, there exist algorithms which (i) for a given automaton $\mathfrak{A}$ compute a regular term R such that $L(\mathfrak{A}) = L(R)$; (ii) for a given regular term R compute a simple system of equations Σ over $\vec{X}$ whose solution for a given variable X_0 is exactly $L(R)$; and (iii) which for a given simple system of equations Σ over $\{X_i : i < m\}$ compute an automaton $\mathfrak{A}$ such that $\vec{X}$ is its set of states and the solution for X_i is exactly the set of strings which send the automaton from state X_0 into X_i. □

This is the most important theorem for regular languages. We shall derive a few consequences. Notice we can turn a finite state automaton $\mathfrak{A}$ into a Turing machine T accepting the same language in linear time and no additional space. Therefore, the recognition problem for regular languages is in

DTIME(n) and in **DSPACE**(n). This also applies to the parsing problem, as is easily seen.

Corollary 2.9 *The recognition and the parsing problem are in* ***DTIME***(n) *and* ***DSPACE***(n).

Corollary 2.10 *The set of regular languages over A is closed under intersection and relative complement. Further, for given regular terms R and S one can determine terms U and V such that $L(U) = A^* - L(R)$ and $L(V) = L(R) \cap L(S)$.*

Proof. It is enough to do this construction for automata. Using Theorem 2.8 it follows that we can do it also for the corresponding regular terms. Let $\mathfrak{A} = \langle A, Q, i_0, F, \delta \rangle$. Without loss of generality we may assume that $\mathfrak{A}$ is deterministic. Then let $\mathfrak{A}^- := \langle A, Q, i_0, Q - F, \delta \rangle$. We then have $L(\mathfrak{A}^-) = A^* - L(\mathfrak{A})$. This shows that for given $\mathfrak{A}$ we can construct an automaton which accepts the complement of $L(\mathfrak{A})$. Now let $\mathfrak{A}' = \langle A, Q', i_0', F', \delta' \rangle$. Put

$$\mathfrak{A} \times \mathfrak{A}' := \langle A, Q \times Q', \langle i_0, i_0' \rangle, F \times F', \delta \times \delta' \rangle \tag{2.13}$$

where

$$(\delta \times \delta')(\langle q, q' \rangle, a) := \{\langle r, r' \rangle : r \in \delta(q, a), r' \in \delta'(q', a)\} \tag{2.14}$$

It is easy to show that $L(\mathfrak{A} \times \mathfrak{A}') = L(\mathfrak{A}) \cap L(\mathfrak{A}')$. □
The proof of the next theorem is an exercise.

Theorem 2.11 *Let L and M be regular languages. Then so are L/M and $M \backslash L$. Moreover, L^T, $L^P := L/A^*$ as well as $L^S := A^* \backslash L$ are regular.*

Furthermore, the following important consequence can be established.

Theorem 2.12 *Let $\mathfrak{A}$ and $\mathfrak{B}$ be finite state automata. Then it is decidable whether $L(\mathfrak{A}) = L(\mathfrak{B})$.*

Proof. Let $\mathfrak{A}$ and $\mathfrak{B}$ be given. By Theorem 2.8 we can compute a regular term R with $L(R) = L(\mathfrak{A})$ as well as a regular term S with $L(S) = L(\mathfrak{B})$. Then $L(\mathfrak{A}) = L(\mathfrak{B})$ iff $L(R) = L(S)$ iff $(L(R) - L(S)) \cup (L(S) - L(R)) = \varnothing$. By Corollary 2.10 we can compute a regular term U such that

$$L(U) = (L(R) - L(S)) \cup (L(S) - L(R)) \tag{2.15}$$

Hence $L(\mathfrak{A}) = L(\mathfrak{B})$ iff $L(U) = \varnothing$. This is decidable by Lemma 2.13. □

Lemma 2.13 *The problem '$L(R) = \varnothing$', where R is a regular term, is decidable.*

Proof. By induction on R. If $R = \varepsilon$ or $R = a$ then $L(R) \neq \varnothing$. If $R = 0$ then by definition $L(R) = \varnothing$. Now assume that the problems '$L(R) = \varnothing$' and '$L(S) = \varnothing$' are decidable. Notice that (a) $L(R \cup S) = \varnothing$ iff $L(R) = \varnothing$ and $L(S) = \varnothing$, (b) $L(R \cdot S) = \varnothing$ iff $L(R) = \varnothing$ or $L(S) = \varnothing$ and (c) $L(R^*) = \varnothing$ iff $L(R) = \varnothing$. All three problems are decidable. □

We conclude with the following theorem, which we have used already in Section 1.5.

Theorem 2.14 *Let L be context free and R regular. Then $L \cap R$ is context free.*

Proof. Let be $G = \langle S, N, A, R \rangle$ be a CFG with $L(G) = L$ and $\mathfrak{A} = \langle n, 0, F, \delta \rangle$ a deterministic automaton consisting of n states such that $L(\mathfrak{A}) = R$. We may assume that rules of G are of the form $X \to a$ or $X \to \vec{Y}$. We define new nonterminals, which are all of the form ${}^iX^j$, where $i, j < n$ and $X \in N$. The interpretation is as follows. X stands for the set of all strings $\vec{\alpha} \in A^*$ such that $X \vdash_G \vec{\alpha}$. ${}^iX^j$ stands for the set of all $\vec{\alpha}$ such that $X \vdash_G \vec{\alpha}$ and $\delta(i, \vec{\alpha}) = j$. We have a set of start symbols, consisting of all ${}^0S^j$ with $j \in F$. As we already know, this does not increase the generative power. A rule $X \to Y_0 Y_1 \cdots Y_{k-1}$ is now replaced by the set of all rules of the form

$$(2.16) \qquad {}^iX^j \to {}^iY_0^{i_0} {}^\frown {}^{i_0}Y_1^{i_1} {}^\frown \ldots {}^\frown {}^{i_{k-2}}Y_{k-1}^j$$

Finally, we take all rules of the form ${}^iX^j \to a$, $\delta(i, a) = j$. This defines the grammar G_r. We shall show: $\vdash_{G^r} \vec{x}$ iff $\vdash_G \vec{x}$ and $\vec{x} \in L(\mathfrak{A})$. ($\Rightarrow$) Let $\mathfrak{B}$ be a G^r–tree with associated string $\vec{x}$. The map ${}^iX^j \mapsto X$ turns $\mathfrak{B}$ into a G–tree. Hence $\vec{x} \in L(G)$. Further, it is easily shown that $\delta(0, x_0 x_1 \cdots x_j) = k_j$, where ${}^{k_{j-1}}X^{k_j}$ is the node dominating x_j. Also, if $|\vec{x}| = n$, then ${}^0S^{k_n}$ is the top node and by construction $k_n \in F$. Hence $\delta(\vec{x}, 0) \in F$ and so $\vec{x} \in L(\mathfrak{A})$. ($\Leftarrow$) Let $\vec{x} \in L(G)$ and $\vec{x} \in L(\mathfrak{A})$. We shall show that $\vec{x} \in L(G^r)$. We take a G–tree $\mathfrak{B}$ for $\vec{x}$. We shall now prove that one can replace the G–nonterminals in $\mathfrak{B}$ in such a way by G^r–nonterminals that we get a G_r–tree. The proof is by induction on the height of a node. We begin with nodes of height 1. Let $\vec{x} = \prod_{i<n} x_i$; and let X_i be the nonterminal above x_i. Further let $\delta(0, \prod_{i<j} x_i) = j_i$. Then $p_0 = 0$ and $p_n \in F$. We replace X_i by ${}^{p_i}X^{p_{i+1}}$. We say that two nodes x and y **connect** if they are adjacent and for the labels ${}^iX^j$ of x and ${}^kY^\ell$ of y we have $j = k$. Let x be a node of height $n+1$ with label X and let x be mother of the nodes with

labels $Y_0Y_1 \cdots Y_{n-1}$ in G. We assume that below x all nodes carry labels from G^r in such a way that adjacent nodes connect. Then there exists a rule in G_r such that X can be labelled with superscripts, the left hand superscript of Y_0 to its left and the right hand superscript of Y_{n-1} to its right. All adjacent nodes of height $n+1$ connect, as is easily seen. Further, the leftmost node carries the left superscript 0, the rightmost node carries a right superscript p_n, which is an accepting state. Eventually, the root has superscripts as well. It carries the label ${}^0S^{p_n}$, and so we have a G_r–tree. □

Exercise 46. Prove Theorem 2.11.

Exercise 47. Show that a language is regular iff it can be generated by a grammar with rules of the form $X \to Y$, $X \to Ya$, $X \to a$ and $X \to \varepsilon$. Such a grammar is called **left regular**, in contrast to the grammars of Type 3, which we also call **right regular**. Show also that it is allowed to add rules of the form $X \to \vec{x}$ and $X \to Y\vec{x}$.

Exercise 48. Show that there is a grammar with rules of the form $X \to a$, $X \to aY$ and $X \to Ya$ which generates a nonregular language. This means that a Type 3 grammar may contain (in general) only left regular rules or only right regular rules, but not both.

Exercise 49. Show that if L and M are regular, then so are L/M and $M\backslash L$.

Exercise 50. Let L be a language over A. Define an equivalence relation $\sim_S$ over A^* as follows. $\vec{x} \sim_S \vec{y}$ iff for all $\vec{z} \in A^*$ we have $\vec{x}^\frown\vec{z} \in L \Leftrightarrow \vec{y}^\frown\vec{z} \in L$. L is said to have **finite index** if there are only finitely many equivalence classes with respect to $\sim_S$. Show that L is regular iff it has finite index.

Exercise 51. Show that the language $\{\mathtt{a}^n\mathtt{b}^n : n \in \omega\}$ does not have finite index. Hence it is not regular.

Exercise 52. Show that the intersection of a context sensitive language with a regular language is again context sensitive.

Exercise 53. Show that L is regular iff it is accepted by a read only 1–tape Turing machine.

2. Normal Forms

In the remaining sections of this chapter we shall deal with CFGs and their languages. In view of the extensive literature about CFLs it is only possible

to present an overview. In this section we shall deal in particular with normal forms. There are many normal forms for CFGs, each having a different purpose. However, notice that the transformation of a grammar into a normal form necessarily destroys some of its properties. So, to say that a grammar can be transformed into another is meaningless unless we specify exactly what properties remain constant under this transformation. If, for example, we are only interested in the language generated then we can transform any CFG into Chomsky Normal Form. However, if we want to maintain the constituent structures, then only the so–called standard form is possible. A good exposition of this problem area can be found in [**?**].

Before we deal with reductions of grammars we shall study the relationship between derivations, trees and sets of rules. To be on the safe side, we shall assume that every symbol occurs at least once in a tree, that is, that the grammar is slender in the sense of Definition 2.17. From the considerations of Section 1.6 we conclude that for any two CFGs $G = \langle \mathtt{S}, N, A, R\rangle$ and $G' = \langle \mathtt{S}', N', A, R'\rangle$ $L_B(G) = L_B(G')$ iff $\mathrm{der}(G) = \mathrm{der}(G')$. Likewise we see that for all $X \in N \cup N'$ $\mathrm{der}(G, X) = \mathrm{der}(H, X)$ iff $R = R'$. Now let $G = \langle \mathtt{S}, N, A, R\rangle$ and a sequence $\Gamma = \langle \vec{\alpha}_i : i < n\rangle$ be given. In order to test whether Γ is a G–string sequence we have to check for each $i < n-1$ whether $\vec{\alpha}_{i+1}$ can be derived from $\vec{\alpha}_i$ with a single application of a rule. To this end we have to choose an $\vec{\alpha}_i$ and apply a rule and check whether the string obtained equals $\vec{\alpha}_{i+1}$. Checking this needs $a_G \times |\vec{\alpha}_i|$ steps, where a_G is a constant which depends only on G. Hence for the whole derivation we need $\sum_{i<n} a_G|\vec{\alpha}_i|$ steps. This can be estimated from above by $a_G \times n \times |\vec{\alpha}_{n-1}|$ and if G is strictly expanding also by $a_G \times |\vec{\alpha}_{n-1}|^2$. It can be shown that there are grammars for which this is the best possible bound. In order to check for an ordered labelled tree whether it can be generated by γG we need less time. We only need to check for each node whether the local tree at x conforms to some rule of G. This can be done in constant time. The time therefore only linearly depends on the size of the tree.

There is a tight connection between derivations and trees. To begin, a derivation has a unique tree corresponding to it. Simply translate the derivation in G into a derivation in γG. Conversely, however, there may exist many derivations for the same tree. Their number can be very large. However, we can obtain them systematically in the following way. Let $\mathfrak{B}$ be an (exhaustively ordered, labelled) tree. Call $\lhd \subseteq B^2$ a **linearisation** if $\lhd$ is an irreflexive, linear ordering and from $x > y$ follows $x \lhd y$. Given a linearisation, a derivation is found as follows. We begin with the element which is smallest

with respect to $\lhd$. This is, as is easy to see, the root. The root carries the label $\mathtt{S}$. Inductively, we shall construct cuts $\vec{\alpha}_i$ through $\mathfrak{B}$ such that the sequence $\langle \vec{\alpha}_i : i < n \rangle$ is a derivation of the associated string. (Actually, the derivation is somewhat more complex than the string sequence, but we shall not complicate matters beyond need here.) The beginning is clear: we put $\vec{\alpha}_0 := \mathtt{S}$. Now assume that $\vec{\alpha}_i$ has been established, and that it is not identical to the associated string of $\mathfrak{B}$. Then there exists a node y with nonterminal label in $\vec{\alpha}_i$. (There is a unique correspondence between nodes of the cut and segments of the strings $\vec{\alpha}_i$.) We take the smallest such node with respect to $\lhd$. Let its label be Y. Since we have a G–tree, the local tree with root y corresponds to a rule of the form $Y \to \vec{\beta}$ for some $\vec{\beta}$. In $\vec{\alpha}_i$ y defines a unique instance of that rule. Then $\vec{\alpha}_{i+1}$ is the result of replacing that occurrence of Y by $\vec{\beta}$. The new string is then the result of applying a rule of G, as desired.

It is also possible to determine for each derivation a linearisation of the tree which yields that derivation in the described manner. However, there can be several linearisations that yield the same derivation.

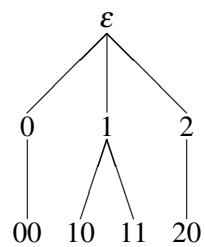
Theorem 2.15 *Let G be a CFG and $\mathfrak{B} \in L_B(G)$. Further, let $\lhd$ be a linearisation of $\mathfrak{B}$. Then $\lhd$ determines a G–derivation* $\mathrm{der}(\lhd)$ *of the string which is associated to $\mathfrak{B}$. If $\blacktriangleleft$ is another linearisation of $\mathfrak{B}$ then* $\mathrm{der}(\blacktriangleleft) = \mathrm{der}(\lhd)$ *is the case iff $\blacktriangleleft$ and $\lhd$ coincide on the interior nodes of $\mathfrak{B}$.* $\square$

Linearisations can also be considered as top down search strategies on a tree. We shall present examples. The first is a particular case of the so–called **depth–first** search and the linearisation shall be called **leftmost linearisation**. It is as follows. $x \lhd y$ iff $x > y$ or $x \sqsubset y$. For every tree there is exactly one leftmost linearisation. We shall denote the fact that there is a leftmost derivation of $\vec{\alpha}$ from X by $X \vdash_G^{\ell} \vec{\alpha}$. We can generalize the situation as follows. Let $\blacktriangleleft$ be a linear ordering uniformly defined on the leaves of local subtrees. That is to say, if $\mathfrak{B}$ and $\mathfrak{C}$ are isomorphic local trees (that is, if they correspond to the same rule ρ) then $\blacktriangleleft$ orders the leaves $\mathfrak{B}$ linearly in the same way as $\lhd$ orders the leaves of $\mathfrak{C}$ (modulo the unique (!) isomorphism). In the case of the leftmost linearisation the ordering is the one given by $\sqsubset$. Now a minute's reflection reveals that every linearisation of the local subtrees of a tree induces a linearisation of the entire tree but not conversely (there are orderings which do not proceed in this way, as we shall see shortly). $X \vdash_G^{\blacktriangleleft} \vec{\alpha}$ denotes the fact that there is a derivation of $\vec{\alpha}$ from X determined by $\blacktriangleleft$. Now call π a priorisation for $G = \langle \mathtt{S}, N, A, R \rangle$ if π defines a linearisation on the local tree $\mathfrak{H}_\rho$, for every $\rho \in R$. Since the root is always the first element in a linearisa-

tion, we only need to order the daughters of the root node, that is, the leaves. Let this ordering be $\blacktriangleleft$. We write $X \vdash_G^{\pi} \vec{\alpha}$ if $X \vdash_G^{\blacktriangleleft} \vec{\alpha}$ for the linearisation $\blacktriangleleft$ defined by π.

Proposition 2.16 *Let π be a priorisation. Then $X \vdash_G^{\pi} \vec{x}$ iff $X \vdash_G \vec{x}$.*

A different strategy is the *breadth–first search*. This search goes through the tree in increasing depth. Let S_n be the set of all nodes x with $d(x) = n$. For each n, S_n shall be ordered linearly by $\sqsubset$. The **breadth–first search** is a linearisation Δ, which is defined as follows. (a) If $d(x) = d(y)$ then $x \,\Delta\, y$ iff $x \sqsubset y$, and (b) if $d(x) < d(y)$ then $x \,\Delta\, y$. The difference between these search strategies, depth–first and breadth–first, can be made very clear with tree domains (see Section 1.4). The depth–first search traverses the tree domain in the lexicographical order, the breadth–first search in the numerical order. Let the following tree domain be given.



The depth–first linearisation is

(2.17) $\varepsilon, 0, 00, 1, 10, 11, 2, 20$

The breadth–first linearisation, however, is

(2.18) $\varepsilon, 0, 1, 2, 00, 10, 11, 20$

Notice that with these linearisations the tree domain ω^* cannot be enumerated. Namely, the depth–first linearisation begins as follows.

(2.19) $\varepsilon, 0, 00, 000, 0000, \ldots$

So we never reach 1. The breadth–first linearisation goes like this.

(2.20) $\varepsilon, 0, 1, 2, 3, \ldots$

So, we never reach 00. On the other hand, ω^* is countable, so we do have a linearisation, but it is more complicated than the given ones.

The first reduction of grammars we look at is the elimination of superfluous symbols and rules. Let $G = \langle \mathtt{S}, A, N, R \rangle$ be a CFG. Call $X \in N$ **reachable** if $G \vdash \vec{\alpha}^\frown X^\frown \vec{\beta}$ for some $\vec{\alpha}$ and $\vec{\beta}$. X is called **completable** if there is an $\vec{x}$ such that $X \Rightarrow_R^* \vec{x}$.

$$
\begin{array}{lllllll}
 & \mathtt{S} & \rightarrow & \mathtt{AB} & \mathtt{A} & \rightarrow & \mathtt{CB} \\
(2.21) & \mathtt{B} & \rightarrow & \mathtt{AB} & \mathtt{A} & \rightarrow & \mathtt{x} \\
 & \mathtt{D} & \rightarrow & \mathtt{Ay} & \mathtt{C} & \rightarrow & \mathtt{y}
\end{array}
$$

In the given grammar A, C and D are completable, and S, A, B and C are reachable. Since S, the start symbol, is not completable, no symbol is both reachable and completable. The grammar generates no terminal strings.

Let N' be the set of symbols which are both reachable and completable. If $\mathtt{S} \notin N'$ then $L(G) = \varnothing$. In this case we put $N' := \{\mathtt{S}\}$ and $R' := \varnothing$. Otherwise, let R' be the restriction of R to the symbols from $A \cup N'$. This defines $G' = \langle \mathtt{S}, N', A, R' \rangle$. It may be that throwing away rules may make some nonterminals unreachable or uncompletable. Therefore, this process must be repeated until $G' = G$, in which case every element is both reachable and completable. Call the resulting grammar G^s. It is clear that $G \vdash \vec{\alpha}$ iff $G^s \vdash \vec{\alpha}$. Additionally, it can be shown that every derivation in G is a derivation in G^s and conversely.

Definition 2.17 *A CFG is called* ***slender*** *if either $L(G) = \varnothing$ and G has no nonterminals except for the start symbol and no rules; or $L(G) \neq \varnothing$ and every nonterminal is both reachable and completable.*

Two slender grammars have identical sets of derivations iff their rule sets are identical.

Proposition 2.18 *Let G and H be slender. Then $G = H$ iff* $\mathrm{der}(G) = \mathrm{der}(H)$.

Proposition 2.19 *For every CFG G there is an effectively constructable slender CFG $G^s = \langle \mathtt{S}, N^s, A, R^s \rangle$ such that $N^s \subseteq N$, which has the same set of derivations as G. In this case it also follows that $L_B(G^s) = L_B(G)$.* □

Next we shall discuss the role of the nonterminals. Since these symbols do not occur in $L(G)$, their name is irrelevant for the purposes of $L(G)$. To make this precise we shall introduce the notion of a rule simulation. Let G and

G' be grammars with sets of nonterminals N and N'. Let $\sim \subseteq N \times N'$ be a relation. This relation can be extended to a relation $\approx \subseteq (N \cup A)^* \times (N' \cup A)^*$ by putting $\vec{\alpha} \approx \vec{\beta}$ if $\vec{\alpha}$ and $\vec{\beta}$ are of equal length and $\alpha_i \sim \beta_i$ for every i. A relation $\sim \subseteq N \times N'$ is called a **forward rule simulation** or an **R–simulation** if (0) $\mathtt{S} \sim \mathtt{S}'$, (1) if $X \to \vec{\alpha} \in R$ and $X \sim Y$ then there exists a $\vec{\beta}$ such that $\vec{\alpha} \approx \vec{\beta}$ and $Y \to \vec{\beta} \in R'$, and (2) if $Y \to \vec{\beta} \in R'$ and $X \sim Y$ then there exists an $\vec{\alpha}$ such that $\vec{\alpha} \approx \vec{\beta}$ and $X \to \vec{\alpha} \in R$. A **backward simulation** is defined thus. (0) From $\mathtt{S} \sim X$ follows $X = \mathtt{S}'$ and from $Y \sim \mathtt{S}'$ follows $Y = \mathtt{S}$, (1) if $X \to \vec{\alpha} \in R$ and $\vec{\alpha} \approx \vec{\beta}$ then $Y \to \vec{\beta} \in R'$ for some Y such that $X \sim Y$, and (2) if $Y \to \vec{\beta} \in R'$ and $\vec{\beta} \approx \vec{\alpha}$ then $X \to \vec{\alpha} \in R$ for some X such that $X \sim Y$.

We give an example of a forward simulation. Let G and G' be the following grammars.

$$
\begin{array}{lll}
\mathtt{S} & \to \mathtt{ASB} \mid \mathtt{AB} & \mathtt{S} \to \mathtt{ATB} \mid \mathtt{ASC} \mid \mathtt{AC} \\
\mathtt{A} & \to \mathtt{b} & \mathtt{T} \to \mathtt{ATC} \mid \mathtt{AC} \\
\mathtt{B} & \to \mathtt{b} & \mathtt{A} \to \mathtt{a} \\
 & & \mathtt{B} \to \mathtt{b} \\
 & & \mathtt{C} \to \mathtt{b}
\end{array}
\tag{2.22}
$$

The start symbol is $\mathtt{S}$ in both grammars. Then the following is an R–simulation.

$$\sim := \{\langle \mathtt{A},\mathtt{A}\rangle, \langle \mathtt{B},\mathtt{B}\rangle, \langle \mathtt{S},\mathtt{S}\rangle, \langle \mathtt{B},\mathtt{C}\rangle, \langle \mathtt{S},\mathtt{T}\rangle\} \tag{2.23}$$

Together with $\sim$ also the converse relation $\sim^{\smile}$ is an R–simulation. If $\sim$ is an R–simulation and $\langle \vec{\alpha}_i : i < n+1\rangle$ is a G–derivation there exists a G'–derivation $\langle \vec{\beta}_i : i < n+1\rangle$ such that $\vec{\alpha}_i \approx \vec{\beta}_i$ for every $i < n+1$. We can say more exactly that if $\langle \vec{\alpha}_i, C, \vec{\alpha}_{i+1}\rangle$ is an instance of a rule from G where $C = \langle \kappa_1, \kappa_2\rangle$ then there is a context $D = \langle \lambda_1, \lambda_2\rangle$ such that $\langle \vec{\beta}_i, D, \vec{\beta}_{i+1}\rangle$ is an instance of a rule from G'. In this way we get that for every $\mathfrak{B} = \langle B, <, \sqsubset, \ell\rangle \in L_B(G)$ there is a $\mathfrak{C} = \langle B, <, \sqsubset, \mu\rangle \in L_B(G')$ such that $\ell(x) = \mu(x)$ for every leaf and $\ell(x) \sim \mu(x)$ for every nonleaf. Analogously to a rule simulation we can define a simulation of derivation by requiring that for every G–derivation Γ there is a G'–derivation Δ which is equivalent to it.

Proposition 2.20 *Let G_1 and G_2 be slender CFGs and $\sim \subseteq N_1 \times N_2$ be an R–simulation. Then for every G_1–derivation $\langle \vec{\alpha}_i : i < n\rangle$ there exists a G_2–derivation $\langle \vec{\beta}_i : i < n\rangle$ such that $\vec{\alpha}_i \approx \vec{\beta}_i$, $i < n$.* □

We shall look at two special cases of simulations. Two grammars G and G' are called **equivalent** if there is a bijection $b\colon N \cup A \to N' \cup A$ such that $b(x) = x$

for every $x \in A$, $b(S) = S'$ and $\overline{b}$ induces a bijection between G–derivations and G'–derivations. This notion is more restrictive than the one which requires that $\overline{b}$ is a bijection between the sets of rules. For it may happen that certain rules can never be used in a derivation. For given CFGs we can easily decide whether they are equivalent. To begin, we bring them into a form in which all rules are used in a derivation, by removing all symbols that are not reachable and not completable. Such grammars are equivalent if there is a bijection b which puts the rules into correspondence. The existence of such a bijection is easy to check.

The notion of equivalence just proposed is too strict in one sense. There may be nonterminal symbols which cannot be distinguished. We say G is **reducible to** G' if there is a surjective function $b\colon N \cup A \twoheadrightarrow N' \cup A'$ such that $b(S) = S'$, $b(x) = x$ for every $x \in A$ and such that $\overline{b}$ maps every G–derivation onto a G'–derivation, while every preimage under $\overline{b}$ of a G'–derivation is a G–derivation. (We do not require however that the preimage of the start symbol from G' is unique; only that the start symbol from G has *one* preimage which is a start symbol of G'.)

Definition 2.21 *G is called **reduced** if every grammar G' such that G is reducible onto G' can itself be reduced onto G.*

Given G we can effectively construct a reduced grammar onto which it can be reduced. We remark that in our example above G' is not reducible onto G. For even though $\sim^{\smile}$ is a function (with $\mathtt{A} \mapsto \mathtt{A}, \mathtt{B} \mapsto \mathtt{B}, \mathtt{C} \mapsto \mathtt{B}, \mathtt{S} \mapsto \mathtt{S}, \mathtt{T} \mapsto \mathtt{S}$) and ASB can be derived from S in one step, ATB cannot be derived from S in one step. Given G and the function $\sim^{\smile}$ the following grammar is reduced onto G.

$$
\begin{aligned}
&\mathtt{S} \to \mathtt{ASB} \mid \mathtt{ATB} \mid \mathtt{ASC} \mid \mathtt{ATC} \mid \mathtt{AB} \mid \mathtt{AC}\\
&\mathtt{T} \to \mathtt{ASB} \mid \mathtt{ATB} \mid \mathtt{ASC} \mid \mathtt{ATC} \mid \mathtt{AB} \mid \mathtt{AC}\\
&\mathtt{A} \to \mathtt{a}\\
&\mathtt{B} \to \mathtt{b}\\
&\mathtt{C} \to \mathtt{b}
\end{aligned}
\tag{2.24}
$$

Now let G be a CFG. We add to A two more symbols, namely (and), not already contained in A. Subsequently, we replace every rule $X \to \vec{\alpha}$ by the

rule $X \to (^\frown \vec{\alpha}^\frown)$. The so–constructed grammar is denoted by G^b.

(2.25)

G			G^b		
S	$\to$	AS | SB | AB	S	$\to$	(AS) | (SB) | (AB)
A	$\to$	a	A	$\to$	(a)
B	$\to$	b	B	$\to$	(b)

The grammar G generates the language $\mathtt{a}^+\mathtt{b}^+$. The string aabb has several derivations, which correspond to different trees.

(2.26)
$$\langle \mathtt{S}, \mathtt{AS}, \mathtt{ASB}, \mathtt{AABB}, \ldots, \mathtt{aabb} \rangle$$
$$\langle \mathtt{S}, \mathtt{SB}, \mathtt{ASB}, \mathtt{AABB}, \ldots, \mathtt{aabb} \rangle$$

If we look at the analogous derivations in G^b we get the strings

(2.27) $\quad$ ((a)(((a)(b))(b))), $\quad$ (((a)((a)(b)))(b))

These are obviously distinct. Define a homomorphism $\overline{e}$ by $\overline{e}(a) := a$, if $a \in A$, $\overline{e}\colon) \mapsto \varepsilon$ and $\overline{e}\colon) \mapsto \varepsilon$. Then it is not hard to see that

(2.28) $\quad L(G) = \overline{e}[L(G^b)]$

Now look at the class of trees $L(G)$ and forget the labels of all nodes which are not leaves. Then the structure obtained shall be called a **bracketing analysis** of the associated strings. The reason is that the bracketing analyses are in one–to–one correspondence with the strings which $L(G^b)$ generates. Now we will ask ourselves whether for two given grammars G and H it is decidable whether they generate the same bracketing analyses. We ask ourselves first what the analogon of a derivation of G is in G^b. Let $\vec{\gamma}X\vec{\eta}$ be derivable in G, and let the corresponding G^b–string in this derivation be $\vec{\gamma}^b X \vec{\eta}^b$. In the next step X is replaced by α. Then we get $\vec{\gamma}\vec{\alpha}\vec{\eta}$, and in G^b the string $\vec{\gamma}^b(\vec{\alpha})\vec{\delta}^b$. If we have an R–simulation to H then it is also an R–simulation from G^b to H^b provided that it sends the opening bracket of G^b to the opening bracket of H^b and the closing bracket of G^b to the closing bracket of H^b. It follows that if there is an R–simulation from G to H then not only we have $L(G) = L(H)$ but also $L(G^b) = L(H^b)$.

Theorem 2.22 *We have $L(G^b) = L(H^b)$ if there is an R–simulation from G to H.*

The bracketing analysis is too strict for most purposes. First of all it is not customary to put a single symbol into brackets. Further, it makes no sense to distinguish between $((\vec{x}))$ and $(\vec{x})$, since both strings assert that $\vec{x}$ is a constituent. We shall instead use what we call **constituent analyses**. These are pairs $\langle \vec{x}, \mathfrak{C} \rangle$ in which $\vec{x}$ is a string and $\mathfrak{C}$ an exhaustively ordered constituent structure defined over $\vec{x}$. We shall denote by $L_c(G)$ the class of all constituent analyses generated by G. In order to switch from bracketing analyses to constituent analyses we only have to eliminate the unary rules. This can be done as follows. Simply replace every rule $\rho = Y \to \vec{\alpha}$, where $|\vec{\alpha}| > 1$, by the set $\rho^2 := \{Z \to \vec{\alpha} : Z \Rightarrow^* Y\}$. $R^> := \bigcup \langle \rho^2 : \rho \in R \rangle$. Finally, let $G^> := \langle \mathtt{S}, N, A, R^> \rangle$. Every rule is strictly productive and we have $L_c(G) = L_c(G^>)$. (Exception needs to be made for $\mathtt{S} \to \varepsilon$, as usual. Also, if necessary, we shall assume that $G^>$ is slender.)

Definition 2.23 *A CFG is in* ***standard form*** *if every rule different from* $\mathtt{S} \to \varepsilon$ *has the form* $X \to \vec{Y}$ *with* $|\vec{Y}| > 1$ *or the form* $X \to a$*. A grammar is in* ***2–standard form*** *or* ***Chomsky Normal Form*** *if every rule is of the form* $\mathtt{S} \to \varepsilon$*,* $X \to Y_0 Y_1$ *or* $X \to a$*.*

(Notice that by our conventions a CFG in standard form contains the rule $X \to \varepsilon$ for $X = \mathtt{S}$, but this happens only if $\mathtt{S}$ is not on the right hand side of a rule.) We already have proved that the following holds.

Theorem 2.24 *For every CFG G one can construct a slender CFG* G^n *in standard form which generates the same constituent structures as G.*

Theorem 2.25 *For every CFG G we can construct a slender CFG* G^c *in Chomsky Normal Form such that* $L(G^c) = L(G)$*.*

Proof. We may assume that G is in standard form. Let $\rho = X \to Y_0 Y_1 \cdots Y_{n-1}$ be a rule with $n > 2$. Let $Z_0^\rho, Z_1^\rho, \ldots, Z_{n-2}^\rho$ be new nonterminals. Replace ρ by the rules

$$\rho_0^c := X \to Y_0 Z_0^\rho,\ \rho_1^c := Z_0^\rho \to Y_1 Z_1^\rho, \ldots, \qquad \rho_{n-2}^c := Z_{n-3}^\rho \to Y_{n-2} Y_{n-1} \tag{2.29}$$

Every derivation in G of a string $\vec{\alpha}$ can be translated into a derivation in G^c by replacing every instance of ρ by a sequence $\rho_0^c, \rho_1^c, \ldots, \rho_{n-1}^c$. For the converse we introduce the following priorisation π on the rules. Let Z_i^ρ be

always before Y_i. However, in $Z^{\rho}_{n-3} \to Y_{n-2}Y_{n-1}$ we choose the leftmost priorisation. We show $G \vdash^{\ell} \vec{x}$ iff $G^c \vdash^{\pi} \vec{x}$. For if $\langle \alpha_i : i < p+1 \rangle$ is a leftmost derivation of $\vec{x}$ in G, then replace every instance of a rule ρ by the sequence ρ^c_0, ρ^c_1, and so on until ρ^c_{n-2}. This is a G^c–derivation, as is easily checked. It is also a π–derivation. Conversely, let $\langle \beta_j : j < q+1 \rangle$ be a G^c–derivation which is priorized with π. If β_{i+1} is the result of an application of the rule ρ^c_k, $k < n-2$, then $i+2 < q+1$ and β_{i+2} is the result of an application of ρ^c_{k+1} on β_{i+1}, which replaced exactly the occurrence Z_k of the previous instance. This means that every ρ^c_k in a block of instances of $\rho^c_0, \rho^c_1, \ldots, \rho^c_{n-2}$ corresponds to a single instance of ρ. There exists a G–derivation of $\vec{x}$, which can be obtained by backward replacement of the blocks. It is a leftmost derivation. □

For example, the right hand side grammar is the result of the conversion of the left hand grammar into Chomsky Normal Form.

(2.30)

S	→ ASBBT | ABB	S	→ AX | AV
		V	→ BB
		X	→ SY
		Y	→ BZ
		Z	→ BT
T	→ CTD | CD	T	→ CW | CD
		W	→ TD
A	→ a	A	→ a
B	→ b	B	→ b
C	→ c	C	→ c
D	→ d	D	→ d

Definition 2.26 *A CFG is called* ***invertible*** *if from $X \to \vec{\alpha} \in R$ and $Y \to \vec{\alpha} \in R$ it follows that $X = Y$.*

For an invertible grammar the labelling on the leaves uniquely determines the labelling on the entire tree. We propose an algorithm which creates an invertible grammar from a CFG. For simplicity a rule is of the form $X \to \vec{Y}$ or $X \to \vec{x}$. Now we choose our nonterminals from the set $\wp(N) - \{\varnothing\}$. The terminal rules are now of the form $\mathsf{X} \to \vec{x}$, where $\mathsf{X} = \{X : X \to \vec{x} \in R\}$. The nonterminal rules are of the form $\mathsf{X} \to \mathsf{Y}_0\mathsf{Y}_1 \cdots \mathsf{Y}_{n-1}$ with

(2.31) $$\mathsf{X} = \{X : X \to Y_0Y_1 \cdots Y_{n-1} \in R \text{ for some } Y_i \in \mathsf{Y}_i\}$$

Further, we choose a start symbol, Σ, and we take the rules $\Sigma \to \vec{\mathsf{X}}$ for every $\vec{X}$, for which there are $X_i \in \mathsf{X}_i$ with $S \to \vec{X} \in R$. This grammar we call G^i.

It is not difficult to show that G^i is invertible. For let $\mathsf{Y}_0\mathsf{Y}_1\cdots\mathsf{Y}_{n-1}$ be the right hand side of a production. Then there exist $Y_i \in \mathsf{Y}_i$, $i < n$, and an X such that $X \to \vec{Y}$ is a rule in G. Hence there is an X such that $\mathsf{X} \to \vec{\mathsf{Y}}$ is in G^i. X is uniquely determined. Further, G^i is in standard form (Chomsky Normal Form), if this is the case with G.

Theorem 2.27 *Let G be a CFG. Then we can construct an invertible CFG G^i which generates the same bracketing analyses as G.* □

The advantage offered by invertible grammars is that the labelling can be reconstructed from the labellings on the leaves. The reader may reflect on the fact that G is invertible exactly if G^b is.

Definition 2.28 *A CFG is called **perfect** if it is in standard form, slender, reduced and invertible.*

It is instructive to see an example of a grammar which is invertible but not reduced.

(2.32)

G			H		
S	$\to$	AS | BS | A | B	S	$\to$	CS | C
A	$\to$	a	C	$\to$	a | b
B	$\to$	b			

G is invertible but not reduced. To this end look at H and the map $\mathtt{A} \mapsto \mathtt{C}$, $\mathtt{B} \mapsto \mathtt{C}$, $\mathtt{S} \mapsto \mathtt{S}$. This is an R–simulation. H is reduced and invertible.

Theorem 2.29 *For every CFG we can construct a perfect CFG which generates the same constituent structures.*

Finally we shall turn to the so–called *Greibach Normal Form*. This form most important for algorithms recognizing languages by reading the input from left to right. Such algorithms have problems with rules of the form $X \to Y^\frown\vec{\alpha}$, in particular if $Y = X$.

Definition 2.30 *Let $G = \langle \mathtt{S}, N, A, R\rangle$ be a CFG. G is in **Greibach (Normal) Form** if every rule is of the form $\mathtt{S} \to \varepsilon$ or of the form $X \to x^\frown\vec{Y}$.*

Proposition 2.31 *Let G be in Greibach Normal Form. If $X \vdash_G \vec{\alpha}$ then $\vec{\alpha}$ has a leftmost derivation from X in G iff $\vec{\alpha} = \vec{y}^\frown\vec{Y}$ for some $\vec{y} \in A^*$ and $\vec{Y} \in N^*$ and $\vec{y} = \varepsilon$ only if $\vec{Y} = X$.*

The proof is not hard. It is also not hard to see that this property characterizes the Greibach form uniquely. For if there is a rule of the form $X \to Y^\frown\vec{\gamma}$ then there is a leftmost derivation of $Y^\frown\vec{\gamma}$ from X, but not in the desired form. Here we assume that there are no rules of the form $X \to X$.

Theorem 2.32 (Greibach) *For every CFG one can effectively construct a grammar G^g in Greibach Normal Form with $L(G^g) = L(G)$.*

Before we start with the actual proof we shall prove some auxiliary statements. We call ρ an X–**production** if $\rho = X \to \vec{\alpha}$ for some $\vec{\alpha}$. Such a production is called **left recursive** if it has the form $X \to X^\frown\vec{\beta}$. Let $\rho = X \to \vec{\alpha}$ be a rule; define $R^{-\rho}$ as follows. For every factorisation $\vec{\alpha} = \vec{\alpha}_1{}^\frown Y^\frown\vec{\alpha}_2$ of $\vec{\alpha}$ and every rule $Y \to \vec{\beta}$ add the rule $X \to \vec{\alpha}_1{}^\frown\vec{\beta}^\frown\vec{\alpha}_2$ to R and finally remove the rule ρ. Now let $G^{-\rho} := \langle \mathtt{S}, N, A, R^{-\rho}\rangle$. Then $L(G^{-\rho}) = L(G)$. We call this construction as **skipping** the rule ρ. The reader may convince himself that the tree for $G^{-\rho}$ can be obtained in a very simple way from trees for G simply by removing all nodes x which dominate a local tree corresponding to the rule ρ, that is to say, which are isomorphic to $\mathfrak{H}_\rho$. (This has been defined in Section 1.6.) This technique works only if ρ is not an $\mathtt{S}$–production. In this case we proceed as follows. Replace ρ by all rules of the form $\mathtt{S} \to \vec{\beta}$ where $\vec{\beta}$ derives from $\vec{\alpha}$ by applying a rule. Skipping a rule does not necessarily yield a new grammar. This is so if there are rules of the form $X \to Y$ (in particular rules like $X \to X$).

Lemma 2.33 *Let $G = \langle \mathtt{S}, N, A, R\rangle$ be a CFG and let $X \to X^\frown\vec{\alpha}_i$, $i < m$, be all left recursive X–productions as well as $X \to \vec{\beta}_j$, $j < n$, all non left recursive X–productions. Now let $G^1 := \langle \mathtt{S}, N \cup \{Z\}, A, R^1\rangle$, where $Z \notin N \cup A$ and R^1 consists of all Y–productions from R with $Y \neq X$ as well as the productions*

$$\begin{array}{llll} X \to \vec{\beta}_j & j < n, & Z \to \vec{\alpha}_i & i < m, \\ X \to \vec{\beta}_j{}^\frown Z & j < n, & Z \to \vec{\alpha}_i{}^\frown Z & i < m. \end{array} \tag{2.33}$$

Then $L(G^1) = L(G)$.

Proof. We shall prove this lemma rather extensively since the method is relatively tricky. We consider the following priorisation on G^1. In all rules of the form $X \to \vec{\beta}_j$ and $Z \to \vec{\alpha}_i$ we take the natural ordering (that is, the leftmost ordering) and in all rules $X \to \vec{\beta}_j Z$ as well as $Z \to \vec{\alpha}_i Z$ we also put the left to right ordering except that Z precedes all elements from $\vec{\alpha}_j$ and $\vec{\beta}_i$, respectively. This defines the linearisation $\blacktriangleleft$. Now, let $M(X)$ be the set of all $\vec{\gamma}$ such

that there is a leftmost derivation from X in G in such a way that $\vec{\gamma}$ is the first element not of the form $X^\frown\vec{\delta}$. Likewise, we define $P(X)$ to be the set of all $\vec{\gamma}$ which can be derived from X priorized by $\blacktriangleleft$ in G^1 such that $\vec{\gamma}$ is the first element which does not contain Z. We claim that $P(X) = M(X)$. It can be seen that

$$M(X) = \bigcup_{j<n} \vec{\beta}_j \cdot (\bigcup_{i<m} \vec{\alpha}_i)^* = P(X) \tag{2.34}$$

From this the desired conclusion follows thus. Let $\vec{x} \in L(G)$. Then there exists a leftmost derivation $\Gamma = \langle A_i : i < n+1 \rangle$ of $\vec{x}$. (Recall that the A_i are instances of rules.) This derivation is cut into segments Σ_i, $i < \sigma$, of length k_i, such that

$$\Sigma_i = \langle A_j : \sum_{p<i} k_p \leq j < 1 + \sum_{p<i+1} k_i \rangle \tag{2.35}$$

This partitioning is done in such a way that each Σ_i is a maximal portion of Γ of X–productions or a maximal portion of Y–productions with $Y \neq X$. The X–segments can be replaced by a $\blacktriangleleft$–derivation $\widehat{\Sigma}_i$ in G^1, by the previous considerations. The segments which do not contain X–productions are already G^1–derivations. For them we put $\widehat{\Sigma}_i := \Sigma_i$. Now let $\widehat{\Gamma}$ be result of stringing together the $\widehat{\Sigma}_i$. This is well–defined, since the first string of $\widehat{\Sigma}_i$ equals the first string of Σ_i, as the last string of $\widehat{\Sigma}_i$ equals the last string of Σ_i. $\widehat{\Gamma}$ is a G^1–derivation, priorized by $\blacktriangleleft$. Hence $\vec{x} \in L(G^1)$. The converse is analogously proved, by beginning with a derivation priorized by $\blacktriangleleft$. $\square$

Now to the proof of Theorem 2.32. We may assume at the outset that G is in Chomsky Normal Form. We choose an enumeration of N as $N = \{X_i : i < p\}$. We claim first that by taking in new nonterminals we can see to it that we get a grammar G^1 such that $L(G^1) = L(G)$ in which the X_i–productions have the form $X_i \to x^\frown\vec{Y}$ or $X_i \to X_j{}^\frown\vec{Y}$ with $j > i$. This we prove by induction on i. Let i_0 be the smallest i such that there is a rule $X_i \to X_j{}^\frown\vec{Y}$ with $j \leq i$. Let j_0 be the largest j such that $X_{i_0} \to X_j{}^\frown\vec{Y}$ is a rule. We distinguish two cases. The first is $j_0 = i_0$. By the previous lemma we can eliminate the production by introducing some new nonterminal symbol Z_{i_0}. The second case is $j_0 < i_0$. Here we apply the induction hypothesis on j_0. We can skip the rule $X_{i_0} \to X_{j_0}{}^\frown\vec{Y}$ and introduce rules of the form (a) $X_{i_0} \to X_k{}^\frown\vec{Y}'$ with $k > j_0$. In this way the second case is either eliminated or reduced to the first.

Now let $P := \{Z_i : i < p\}$ be the set of newly introduced nonterminals. It may happen that for some j Z_j does not occur in the grammar, but this does

not disturb the proof. Let finally $P_i := \{Z_j : j < i\}$. At the end of this reduction we have rules of the form

$$X_i \to X_j{}^\frown\vec{Y} \qquad (j > i) \tag{2.36a}$$

$$X_i \to x^\frown\vec{Y} \qquad (x \in A) \tag{2.36b}$$

$$Z_i \to \vec{W} \qquad (\vec{W} \in (N \cup P_i)^{+\frown}(\varepsilon \cup Z_i)) \tag{2.36c}$$

It is clear that every X_{p-1}–production already has the form $X_{p-1} \to x^\frown\vec{Y}$. If some X_{p-2}–production has the form (2.36a) then we can skip this rule and get rules of the form $X_{p-2} \to \vec{x}\vec{Y}'$. Inductively we see that all rules of the form can be eliminated in favour of rules of the form (2.36b). Now finally the rules of type (2.36c). Also these rules can be skipped, and then we get rules of the form $Z \to x^\frown\vec{Y}$ for some $x \in A$, as desired.

For example, let the following grammar be given.

$$\begin{array}{lcl@{\qquad}lcl} \mathtt{S} & \to & \mathtt{SDA} \mid \mathtt{CC} & \mathtt{A} & \to & \mathtt{a} \\ \mathtt{D} & \to & \mathtt{DC} \mid \mathtt{AB} & \mathtt{B} & \to & \mathtt{b} \\ & & & \mathtt{C} & \to & \mathtt{c} \end{array} \tag{2.37}$$

The production $\mathtt{S} \to \mathtt{SDA}$ is left recursive. We replace it according to the above lemma by

$$\mathtt{S} \to \mathtt{CCZ}, \quad \mathtt{Z} \to \mathtt{DA}, \quad \mathtt{Z} \to \mathtt{DAZ} \tag{2.38}$$

Likewise we replace the production $\mathtt{D} \to \mathtt{DC}$ by

$$\mathtt{D} \to \mathtt{ABY}, \quad \mathtt{Y} \to \mathtt{C}, \quad \mathtt{Y} \to \mathtt{CY} \tag{2.39}$$

With this we get the grammar

$$\begin{array}{lcl@{\qquad}lcl} \mathtt{S} & \to & \mathtt{CC} \mid \mathtt{CCZ} & \mathtt{A} & \to & \mathtt{a} \\ \mathtt{Z} & \to & \mathtt{DA} \mid \mathtt{DAZ} & \mathtt{B} & \to & \mathtt{b} \\ \mathtt{D} & \to & \mathtt{AB} \mid \mathtt{ABY} & \mathtt{C} & \to & \mathtt{c} \\ \mathtt{Y} & \to & \mathtt{C} \mid \mathtt{CY} & & & \end{array} \tag{2.40}$$

Next we skip the D–productions.

$$\begin{array}{lcl@{\qquad}lcl} \mathtt{S} & \to & \mathtt{CC} \mid \mathtt{CCZ} & \mathtt{A} & \to & \mathtt{a} \\ \mathtt{Z} & \to & \mathtt{ABA} \mid \mathtt{ABYA} \mid \mathtt{ABAZ} \mid \mathtt{ABYAZ} & \mathtt{B} & \to & \mathtt{b} \\ \mathtt{D} & \to & \mathtt{AB} \mid \mathtt{ABY} & \mathtt{C} & \to & \mathtt{c} \\ \mathtt{Y} & \to & \mathtt{C} \mid \mathtt{CY} & & & \end{array} \tag{2.41}$$

Next D can be eliminated (since it is not reachable) and we can replace on the right hand side of the productions the first nonterminals by terminals.

$$
\begin{array}{lcl}
\mathtt{S} & \rightarrow & \mathtt{cC} \mid \mathtt{cCZ} \\
\mathtt{Z} & \rightarrow & \mathtt{aBA} \mid \mathtt{aBYA} \mid \mathtt{aBAZ} \mid \mathtt{aBYZ} \\
\mathtt{Y} & \rightarrow & \mathtt{c} \mid \mathtt{cY}
\end{array}
\tag{2.42}
$$

Now the grammar is in Greibach Normal Form.

Exercise 54. Show that for a CFG G it is decidable (a) whether $L(G) = \varnothing$, (b) whether $L(G)$ is finite, (c) whether $L(G)$ is infinite.

Exercise 55. Let G^i be the invertible grammar constructed from G as defined above. Show that the relation $\sim$ defined by

$$\mathsf{X} \sim Y \quad \Leftrightarrow \quad Y \in \mathsf{X} \tag{2.43}$$

is a backward simulation from G^i to G.

Exercise 56. Let $\langle B, <, \sqsubset, \ell \rangle$ be an ordered labelled tree. If x is a leaf then $\uparrow x$ is a branch and can be thought of in a natural way as a string $\langle \uparrow x, >, \ell \rangle$. Since the leaf x plays a special role, we shall omit it. We say, a **branch expression of** $\mathfrak{B}$ is a string of the form $\langle \uparrow x - \{x\}, >, \ell \rangle$, x a leaf of $\mathfrak{B}$. We call it $\zeta(x)$. Show that the set of all branch expressions of trees from $L_B(G)$ is regular.

Exercise 57. Let G be in Greibach Normal Form and $\vec{x}$ a terminal string of length $n > 0$. Show that every derivation of $\vec{x}$ has exactly the length n. How long is a derivation for an arbitrary string $\vec{\alpha}$?

3. Recognition and Analysis

CFLs can be characterized by special classes of automata, just like regular languages. Since there are CFLs that are not regular, automata that recognize them cannot all be finite state automata. They must have an infinite memory. The special way such a memory is organized and manipulated differentiates the various kinds of nonregular languages. CFLs can be recognized by so–called *pushdown automata*. These automata have a memory in the form of a stack onto which they can put symbols and remove (and read them) one by one. However, the automaton only has access to the symbol added most recently. A *stack* over the alphabet D is a string over D. We shall agree that the first letter of the string is the highest entry in the stack and the last letter

corresponds to the lowest entry. To denote the end of the stack, we need a special symbol, which we denote by #. (See Exercise 43 for the necessity of an end–of–stack marker.)

A *pushdown automaton* steers its actions by means of the highest entry of the stack and the momentary memory state. Its actions consist of three successive steps. (1) The disposal or removal of a symbol on the stack. (2) The moving or not moving of the read head to the right. (3) The change into a memory state (possibly the same one). If the automaton does not move the head in (2) we call the action an ε–**move**. We write A_ε in place of $A \cup \{\varepsilon\}$.

Definition 2.34 *A **pushdown automaton over** A is a septuple*

$$\mathfrak{K} = \langle Q, i_0, A, F, D, \#, \delta \rangle \tag{2.44}$$

where Q and D are finite sets, $i_0 \in Q$, $\# \in D$ and $F \subseteq Q$, as well as

$$\delta \colon Q \times D \times A_\varepsilon \to \wp(Q \times D^*) \tag{2.45}$$

*a function such that $\delta(q,a,d)$ is always finite. We call Q the set of **states**, i_0 the **initial state**, F the set of **accepting states**, D the **stack alphabet**, # the **beginning of the stack** and δ the **transition function**.*

We call $\mathfrak{z} := \langle q, \vec{d} \rangle$, where $q \in Q$ and $\vec{d} \in D^*$, a **configuration**. We now write

$$\langle p, \vec{d} \rangle \xrightarrow{x} \langle p', \vec{d}' \rangle \tag{2.46}$$

if for some $\vec{d}_1$ $\vec{d} = Z^\frown \vec{d}_1$, $\vec{d}' = \vec{e}^\frown \vec{d}_1$ and $\langle p', \vec{e} \rangle \in \delta(p, Z, x)$. We call this a **transition**. We extend the function δ to configurations. $\langle p', \vec{d}' \rangle \in \delta(p, \vec{d}, x)$ is also used. Notice that in contrast to a pushdown automaton a finite state automaton may not change into a new state without reading a new symbol. For a pushdown automaton this is necessary in particular if the automaton wants to clear the stack. If the stack is empty then the automaton cannot work further. This means, however, that the pushdown automaton is necessarily partial. The transition function can now analogously be extended to strings. Likewise, we can define it for sets of states.

$$\mathfrak{z} \xrightarrow{\vec{x}^\frown \vec{y}} \mathfrak{z}' \quad \Leftrightarrow \quad \text{there exists } \mathfrak{z}'' \text{ with } \mathfrak{z} \xrightarrow{\vec{x}} \mathfrak{z}'' \xrightarrow{\vec{y}} \mathfrak{z}' \tag{2.47}$$

If $\mathfrak{z} \xrightarrow{\vec{x}} \mathfrak{z}'$ we say that there is a $\mathfrak{K}$–**computation for** $\vec{x}$ **from** $\mathfrak{z}$ **to** $\mathfrak{z}'$. Now

$$L(\mathfrak{K}) := \{\vec{x} : \textit{for some } q \in F, \vec{z} \in D^* \colon \langle i_0, \# \rangle \xrightarrow{\vec{x}} \langle q, \vec{z} \rangle\} \tag{2.48}$$

We call this the language which is **accepted by $\mathfrak{K}$ by state**. We call a pushdown automaton **simple** if from $\langle q, \vec{z}\rangle \in \delta(p, Z, a)$ follows $|\vec{z}^\frown a| \leq 2$. It is an exercise to prove the next theorem.

Proposition 2.35 *For every pushdown automaton $\mathfrak{K}$ there is a simple pushdown automaton $\mathfrak{L}$ such that $L(\mathfrak{L}) = L(\mathfrak{K})$.*

For this reason we shall tacitly assume that the automaton does not write arbitrary strings but a single symbol. In addition to $L(\mathfrak{K})$ there also is a language which is **accepted by $\mathfrak{K}$ by stack**.

$$L^s(\mathfrak{K}) := \{\vec{x} : \text{for some } q \in Q\colon \langle i_0, \#\rangle \stackrel{\vec{x}}{\to} \langle q, \varepsilon\rangle\} \tag{2.49}$$

The languages $L(\mathfrak{K})$ and $L^s(\mathfrak{K})$ are not necessarily identical for given $\mathfrak{K}$. However, the set of all languages of the form $L(\mathfrak{K})$ for some pushdown automaton equals the set of all languages of the form $L^s(\mathfrak{K})$ for some pushdown automaton. This follows from the next theorem.

Proposition 2.36 *For every pushdown automaton $\mathfrak{K}$ there is an $\mathfrak{L}$ with $L(\mathfrak{K}) = L^s(\mathfrak{L})$ as well as a pushdown automaton $\mathfrak{M}$ with $L^s(\mathfrak{K}) = L(\mathfrak{M})$.*

Proof. Let $\mathfrak{K} = \langle Q, i_0, A, F, D, \#, \delta\rangle$ be given. We add to Q two states, q_i and q_f. q_i shall be the new initial state and $F^{\mathfrak{L}} := \{q_f\}$. Further, we add a new symbol $\flat$ which is the beginning of the stack of $\mathfrak{L}$. We define $\delta^{\mathfrak{L}}(q_i, \flat, \varepsilon) := \{\langle i_0, \#^\frown\flat\rangle\}$. There are no more $\delta^{\mathfrak{L}}$–transitions exiting q_i. For $q \neq q_i, q_f$ and $Z \neq \flat$ $\delta^{\mathfrak{L}}(q, Z, \vec{x}) := \delta^{\mathfrak{K}}(q, Z, x)$, $x \in A$. Further, if $q \in F$ and $Z \neq \flat$, we put $\delta^{\mathfrak{L}}(q, Z, \varepsilon) := \delta^{\mathfrak{K}}(q, Z, \varepsilon) \cup \{\langle q_f, \varepsilon\rangle\}$ and otherwise $\delta^{\mathfrak{L}}(q, Z, \varepsilon) := \delta^{\mathfrak{K}}(q, Z, \varepsilon)$. Finally, let $\delta^{\mathfrak{L}}(q_f, Z, x) := \varnothing$ for $x \in A$ and $\delta^{\mathfrak{L}}(q_f, Z, \varepsilon) := \{\langle q_f, \varepsilon\rangle\}$ for $Z \in D \cup \{\flat\}$. Assume now $\vec{x} \in L(\mathfrak{K})$. Then there is a $\mathfrak{K}$–computation $\langle i_0, \#\rangle \stackrel{\vec{x}}{\to} \langle q, \vec{d}\rangle$ for some $q \in F$ and so we also have an $\mathfrak{L}$–computation $\langle q_i, \flat\rangle \stackrel{\vec{x}}{\to} \langle q_f, \vec{d}\rangle$. Since $\langle q_f, \vec{d}\rangle \stackrel{\varepsilon}{\to} \langle q_f, \varepsilon\rangle$ we have $\vec{x} \in L^s(\mathfrak{L})$. Hence $L(\mathfrak{K}) \subseteq L^s(\mathfrak{L})$. Now, conversely, let $\vec{x} \in L^s(\mathfrak{L})$. Then $\langle q_i, \flat\rangle \stackrel{\vec{x}}{\to} \langle p, \varepsilon\rangle$ for a certain p. Then $\flat$ is deleted only at last since it happens only in q_f and so $p = q_f$. Further, we have $\langle q_i, \flat\rangle \stackrel{\vec{x}}{\to} \langle q, \vec{d}^\frown\flat\rangle$ for some state $q \in F$. This means that there is an $\mathfrak{L}$–computation $\langle i_0, \#^\frown\flat\rangle \stackrel{\vec{x}}{\to} \langle q, \vec{d}^\frown\flat\rangle$. This, however, is also a $\mathfrak{K}$–computation. This shows that $L^s(\mathfrak{L}) \subseteq L(\mathfrak{K})$ and so also the first claim. Now for the construction of $\mathfrak{M}$. We add two new states, q_f and q_i, and a new symbol, $\flat$, which shall be the begin of stack of $\mathfrak{M}$, and we put $F^{\mathfrak{M}} := \{q_f\}$. Again we put

$\delta^{\mathfrak{M}}(q_i, \flat, x) := \varnothing$ for $x \in A$ and $\delta^{\mathfrak{M}}(q_i, \flat, \varepsilon) := \{\langle i_0, \#^\frown \flat\rangle\}$. Also, we put $\delta^{\mathfrak{M}}(q, Z, x) := \delta^{\mathfrak{K}}(q, Z, x)$ for $Z \neq \flat$ and $\delta^{\mathfrak{M}}(q, \flat, \varepsilon) := \{\langle q_f, \varepsilon\rangle\}$, as well as $\delta^{\mathfrak{M}}(q, \flat, x) := \varnothing$ for $x \in A$. Further, $\delta^{\mathfrak{M}}(q_f, Z, x) := \varnothing$. This defines $\delta^{\mathfrak{M}}$. Now consider an $\vec{x} \in L^s(\mathfrak{K})$. There is a $\mathfrak{K}$–computation $\langle i_0, \#\rangle \xrightarrow{\vec{x}} \langle p, \varepsilon\rangle$ for some p. Then there exists an $\mathfrak{L}$–computation

$$\langle q_i, \flat\rangle \xrightarrow{\vec{x}} \langle p, \flat\rangle \xrightarrow{\varepsilon} \langle q_f, \varepsilon\rangle \tag{2.50}$$

Hence $\vec{x} \in L(\mathfrak{M})$. Conversely, let $\vec{x} \in L(\mathfrak{M})$. Then there exists an $\mathfrak{L}$–computation $\langle q_i, \flat\rangle \xrightarrow{\vec{x}} \langle q_f, \vec{d}\rangle$ for some $\vec{d}$. One can see quite easily that $\vec{d} = \varepsilon$. Further, this computation factors as follows.

$$\langle q_i, \flat\rangle \xrightarrow{\varepsilon} \langle i_0, \#^\frown \flat\rangle \xrightarrow{\vec{x}} \langle p, \flat\rangle \xrightarrow{\varepsilon} \langle q_f, \varepsilon\rangle \tag{2.51}$$

Here $p \in Q$, whence $p \neq q_f, q_i$. But every $\mathfrak{M}$–transition from i_0 to p is also a $\mathfrak{K}$–transition. Hence there is a $\mathfrak{K}$–computation $\langle i_0, \#\rangle \xrightarrow{\vec{x}} \langle p, \varepsilon\rangle$. From this follows $\vec{x} \in L^s(\mathfrak{K})$, and so $L^s(\mathfrak{K}) = L(\mathfrak{M})$. □

Lemma 2.37 *Let L be a CFL over A. Then there exists a pushdown automaton $\mathfrak{K}$ such that $L = L^s(\mathfrak{K})$.*

Proof. We take a CFG $G = \langle \mathtt{S}, N, A, R\rangle$ in Greibach Form with $L = L(G)$. We assume that $\varepsilon \notin G$. (If $\varepsilon \in L(G)$, then we construct an automaton for $L(G) - \{\varepsilon\}$ and then modify it slightly.) The automaton possesses only one state, i_0, and uses N as its stack alphabet. The beginning of the stack is $\mathtt{S}$.

$$\delta(i_0, X, x) := \{\langle i_0, \vec{Y}\rangle : X \to x^\frown \vec{Y} \in R\} \tag{2.52}$$

This defines $\mathfrak{K} := \langle \{i_0\}, i_0, A, \{i_0\}, N, \mathtt{S}, \delta\rangle$. We show that $L = L^s(\mathfrak{K})$. To this end recall that for every $\vec{x} \in L(G)$ there is a leftmost derivation. In a grammar in Greibach Form every leftmost derivation derives strings of the form $\vec{y}^\frown \vec{Y}$. Now one shows by induction that $G \vdash \vec{y}^\frown \vec{Y}$ iff $\langle i_0, \vec{Y}\rangle \in \delta(i_0, \mathtt{S}, \vec{y})$. □

Lemma 2.38 *Let $\mathfrak{K}$ be a pushdown automaton. Then $L^s(\mathfrak{K})$ is context free.*

Proof. Let $\mathfrak{K} = \langle Q, i_0, A, F, D, \#, \delta\rangle$ be a pushdown automaton. We may assume that it is simple. Put $N := Q \times D \times (Q \cup \{\mathtt{S}\})$, where $\mathtt{S}$ is a new symbol. $\mathtt{S}$ shall also be the start symbol. We write a general element of N in the form

$[q,A,p]$. Now we define $R := R^s \cup R^0 \cup R^\delta \cup R^\varepsilon$, where

$$
\begin{array}{lll}
R^s & := \{\mathtt{S} \to [i_0,\#,q] : & q \in Q\} \\
R^0 & := \{[p,Z,q] \to x : & \langle r,\varepsilon\rangle \in \delta(p,Z,x)\} \\
R^\delta & := \{[p,Z,q] \to x[r,Y,q] : & \langle r,Y\rangle \in \delta(p,Z,x)\} \\
R^\varepsilon & := \{[p,Z,q] \to [p',X,r][r,Y,q] : & \langle p',XY\rangle \in \delta(p,Z,\varepsilon)\}
\end{array}
\tag{2.53}
$$

The grammar thus defined is called $G(\mathfrak{A})$. We claim that for every $\vec{x} \in A^*$, every $p,q \in Q$ and every $Z \in D$

$$
[p,Z,q] \vdash_G \vec{x} \quad \Leftrightarrow \quad \langle q,\varepsilon\rangle \in \delta(p,Z,\vec{x}) \tag{2.54}
$$

This suffices for the proof. For if $\vec{x} \in L(G)$ then we have $[i_0,\#,q] \vdash_G \vec{x}$ and so because of (2.54) $\langle q,\varepsilon\rangle \in \delta(i_0,\#,\vec{x})$, which means nothing but $\vec{x} \in L^s(\mathfrak{K})$. And if the latter holds then we have $[i_0,\#,q] \vdash_G \vec{x}$ and so $\mathtt{S} \vdash_G \vec{x}$, which is nothing else but $\vec{x} \in L(G)$.

Now we show (2.54). It is clear that (2.54) follows from (2.55).

$$
\begin{aligned}
&[p,Z,q] \vdash_G^\ell \vec{y}^\frown [q_0,Y_0,q_1][q_1,Y_1,q_2]\cdots[q_{m-1},Y_{m-1},q] \\
&\qquad\qquad \Leftrightarrow \qquad \langle q_0,Y_0Y_1\cdots Y_{m-1}\rangle \in \delta(p,Z,\vec{y})
\end{aligned}
\tag{2.55}
$$

(2.55) is proved by induction. □

On some reflection it is seen that for every automaton $\mathfrak{K}$ there is an automaton $\mathfrak{L}$ with only one accepting state which accepts the same language. If one takes $\mathfrak{L}$ in place of $\mathfrak{K}$ then there is no need to use the trick with a new start symbol. Said in another way, we may choose $[i_0,\#,q]$ as a start symbol where q is the accepting state of $\mathfrak{L}$.

Theorem 2.39 (Chomsky) *The CFLs are exactly the languages which are accepted by a pushdown automaton, either by state or by stack.*

From this proof we can draw some further conclusions. The first conclusion is that for every pushdown automaton $\mathfrak{K}$ we can construct a pushdown automaton $\mathfrak{L}$ for which $L^s(\mathfrak{L}) = L^s(\mathfrak{K})$ and which contains no ε–moves. Also, there exists a pushdown automaton $\mathfrak{M}$ such that $L^s(\mathfrak{M}) = L^s(\mathfrak{K})$ and which contains only one state, which is at the same time an initial and an accepting state. For such an automaton these definitions reduce considerably. Such an automaton possesses as a memory only a string. The transition function can be reduced to a function ζ from $A \times D^*$ into finite subsets of D^*. (We do not allow ε–transitions.)

The pushdown automaton runs along the string from left to right. It recognizes in linear time whether or not a string is in the language. However, the automaton is nondeterministic.

Definition 2.40 *A pushdown automaton* $\mathfrak{K} = \langle Q, i_0, A, F, D, \#, \delta \rangle$ *is* ***deterministic*** *if for every* $q \in Q$, $Z \in D$ *and* $x \in A_\varepsilon$ *we have* $|\delta(q,Z,x)| \leq 1$ *and for all* $q \in Q$ *and all* $Z \in D$ *either (a)* $\delta(q,Z,\varepsilon) = \varnothing$ *or (b)* $\delta(q,Z,a) = \varnothing$ *for all* $a \in A$. *A language* L *is called* ***deterministic*** *if* $L = L(\mathfrak{A})$ *for a deterministic automaton* $\mathfrak{A}$. *The set of deterministic languages is denoted by* Δ.

Deterministic languages are such languages which are accepted by a deterministic automaton by state. Now, is it possible to build a deterministic automaton accepting that language just like regular languages? The answer is negative. To this end we consider the **mirror language** $\{\vec{x}\vec{x}^T : \vec{x} \in A^*\}$. This language is surely context free. There are, however, no deterministic automata that accept it. To see this one has to realize that the automaton will have to put into the stack the string $\vec{x}\vec{x}^T$ at least up to $\vec{x}$ in order to compare it with the remaining word, $\vec{x}^T$. The machine, however, has to guess when the moment has come to change from putting onto stack to removing from stack. The reader may reflect that this is not possible without knowing the entire word.

Theorem 2.41 *Deterministic languages are in* ***DTIME***(n).

The proof is left as an exercise.

We have seen that also regular languages are in **DTIME**(n). However, there are deterministic languages which are not regular. Such a language is $L = \{\vec{x}\mathtt{c}\vec{x}^T : \vec{x} \in \{\mathtt{a},\mathtt{b}\}^*\}$. In contrast to the mirror language L is deterministic. For now the machine does not have to guess where the turning point is: it is right after the symbol $\mathtt{c}$.

Now there is the question whether a deterministic automaton can recognize languages using the stack. This is not the case. For let $L = L^s(\mathfrak{K})$, for some deterministic automaton $\mathfrak{K}$. Then, if $\vec{x}\vec{y} \in L$ for some $\vec{y} \neq \varepsilon$ then $\vec{x} \notin L$. We say that L is **prefix free** if it has this property. For if $\vec{x} \in L$ then there exists a $\mathfrak{K}$–computation from $\langle q_0, \#\rangle$ to $\langle q, \varepsilon\rangle$. Further, since $\mathfrak{K}$ is deterministic: if $\langle q_0, \#\rangle \stackrel{\vec{x}}{\to} \mathfrak{z}$ then $\mathfrak{z} = \langle q, \varepsilon\rangle$. However, if the stack has been emptied the automaton cannot work further. Hence $\vec{x}\vec{y} \notin L$. There are deterministic languages which are not prefix free. We present an important class of such languages, the *Dyck–languages*. Let A be an alphabet. For each $x \in A$ let $\underline{x}$ be another symbol. We write $\underline{A} := \{\underline{x} : x \in A\}$. We introduce a congruence θ on

$\mathfrak{Z}(A \cup \underline{A})$. It is generated by the equations

(2.56) $\quad a\underline{a} \; \theta \; \varepsilon$

for all $a \in A$. (The analogous equations $\underline{a}a \; \theta \; \varepsilon$ are *not* included.) A string $\vec{x} \in (A \cup \underline{A})^*$ is called **balanced** if $\vec{x} \; \theta \; \varepsilon$. $\vec{x}$ is balanced iff $\vec{x}$ can be rewritten into ε by successively replacing substrings of the form $x\underline{x}$ into ε.

Definition 2.42 *D_r denotes the set of balanced strings over an alphabet consisting of $2r$ symbols. A language is called a **Dyck–language** if it has the form D_r for some r (and some alphabet $A \cup \underline{A}$).*

The language XML (**Extensible Markup Language**, an outgrowth of HTML) embodies like no other language the features of Dyck–languages. For every string $\vec{x}$ it allows to form a pair of tags <$\vec{x}$> (opening tag) and </$\vec{x}$> (closing tag). The syntax of XML is such that the tags always come in pairs. The tags alone (not counting the text in between) form a Dyck Language. What distinguishes XML from other languages is that tags can be freely formed.

Proposition 2.43 *Dyck–languages are deterministic but not prefix free.*

The following grammars generate the Dyck–languages:

(2.57) $\quad \mathtt{S} \to \mathtt{SS} \mid x\mathtt{S}\underline{x} \mid \varepsilon$

Dyck–languages are therefore context free. It is easy to see that together with $\vec{x}, \vec{y} \in D_r$ also $\vec{x}\vec{y} \in D_r$. Hence Dyck–languages are not prefix free. That they are deterministic follows from some general results which we shall establish later. We leave it to the reader to construct a deterministic automaton which recognizes D_r. This shows that the languages which are accepted by a deterministic automaton by empty stack are a proper subclass of the languages which are accepted by an automaton by state. This justifies the following definition.

Definition 2.44 *A language L is called **strict deterministic** if there is a deterministic automaton $\mathfrak{K}$ such that $L = L^s(\mathfrak{K})$. The class of strict deterministic languages is denoted by Δ^s.*

Theorem 2.45 *L is strict deterministic if L is deterministic and prefix free.*

Proof. We have seen that strict deterministic languages are prefix free. Now let L be deterministic and prefix free. Then there exists an automaton $\mathfrak{K}$ which accepts L by state. Since L is prefix free, this holds for every $\vec{x} \in L$, and for every proper prefix $\vec{y}$ of $\vec{x}$ we have that if $\langle q_0, \#\rangle \xrightarrow{\vec{y}} \langle q, \vec{Y}\rangle$ then q is not an accepting state. Thus we shall rebuild $\mathfrak{K}$ in the following way. Let $\delta_1(q,Z,x) := \delta^{\mathfrak{K}}(q,Z,x)$ if q is not accepting. Further, let $\delta_1(q,Z,x) := \varnothing$ if $q \in F$ and $x \in A$; let $\delta_1(q,Z,\varepsilon) := \{\langle q,\varepsilon\rangle\}$, $Z \in D$. Finally, let $\mathfrak{L}$ be the automaton which results from $\mathfrak{K}$ by replacing $\delta^{\mathfrak{K}}$ with δ_1. $\mathfrak{L}$ is deterministic as is easily checked. Further, an $\mathfrak{L}$–computation can be factored into an $\mathfrak{K}$–computation followed by a deletion of the stack. We claim that $L(\mathfrak{K}) = L^s(\mathfrak{L})$. The claim then follows. So let $\vec{x} \in L(\mathfrak{K})$. Then there exists a $\mathfrak{K}$–computation using $\vec{x}$ from $\langle q_0, \#\rangle$ to $\langle q, \vec{Y}\rangle$ where $q \in F$. For no proper prefix $\vec{y}$ of $\vec{x}$ there is a computation into an accepting state since L is prefix free. So there is an $\mathfrak{L}$–computation with $\vec{x}$ from $\langle q_0, \#\rangle$ to $\langle q, \vec{Y}\rangle$. Now $\langle q, \vec{Y}\rangle \xrightarrow{\varepsilon} \langle q, \varepsilon\rangle$ and so $\vec{x} \in L^s(\mathfrak{L})$. Conversely, assume $\vec{x} \in L^s(\mathfrak{L})$. Then there is a computation $\langle q_0, \#\rangle \xrightarrow{\vec{x}} \langle q, \varepsilon\rangle$. Let $\vec{Y} \in D^*$ be the longest string such that $\langle q_0, \#\rangle \xrightarrow{\vec{x}} \langle q, \vec{Y}\rangle$. Then the $\mathfrak{L}$–step before reaching $\langle q, \vec{Y}\rangle$ is a $\mathfrak{K}$–step. So there is a $\mathfrak{K}$–computation for $\vec{x}$ from $\langle q_0, \#\rangle$ to $\langle q, \vec{Y}\rangle$, and so $\vec{x} \in L(\mathfrak{K})$. $\square$

The proof of this theorem also shows the following.

Theorem 2.46 *Let U be a deterministic CFL. Let L be the set of all $\vec{x} \in U$ for which no proper prefix is in U. Then L is strict deterministic.*

For the following definition we make the following agreement, which shall be used quite often in the sequel. We denote by ${}^{(k)}\vec{\alpha}$ the prefix of $\vec{\alpha}$ of length k in case $\vec{\alpha}$ has length at least k; otherwise ${}^{(k)}\vec{\alpha} := \vec{\alpha}$.

Definition 2.47 *Let $G = \langle \mathrm{S}, N, A, R\rangle$ be a grammar and $\Pi \subseteq \wp(N \cup A)$ a partition. We write $\alpha \equiv \beta$ if there is an $M \in \Pi$ such that $\alpha, \beta \in M$. Π is called* ***strict for G*** *if the following holds.*

① $A \in \Pi$

② *For $C, C' \in N$ and $\vec{\alpha}, \vec{\gamma}_1, \vec{\gamma}_2 \in (N \cup A)^*$: if $C \equiv C'$ and $C \to \vec{\alpha}\,\vec{\gamma}_1$ as well as $C' \to \vec{\alpha}\,\vec{\gamma}_2 \in R$ then either*

   (a) $\vec{\gamma}_1, \vec{\gamma}_2 \neq \varepsilon$ *and* ${}^{(1)}\vec{\gamma}_1 \equiv {}^{(1)}\vec{\gamma}_2$ *or*

   (b) $\vec{\gamma}_1 = \vec{\gamma}_2 = \varepsilon$ *and* $C = C'$.

Definition 2.48 *A CFG G is called* ***strict deterministic*** *if there is a strict partition for G.*

We look at the following example (taken from [**?**]):

$$
\begin{array}{lcl@{\qquad}lcl}
\mathtt{S} & \to & \mathtt{aA} \mid \mathtt{aB} & \mathtt{C} & \to & \mathtt{bC} \mid \mathtt{a} \\
\mathtt{A} & \to & \mathtt{aAa} \mid \mathtt{bC} & \mathtt{D} & \to & \mathtt{bDc} \mid \mathtt{c} \\
\mathtt{B} & \to & \mathtt{aB} \mid \mathtt{bD} & & &
\end{array}
\tag{2.58}
$$

$\Pi = \{\{\mathtt{a},\mathtt{b},\mathtt{c}\},\{\mathtt{S}\},\{\mathtt{A},\mathtt{B}\},\{\mathtt{C},\mathtt{D}\}\}$ is a strict partition. The language generated by this grammar is $\{\mathtt{a}^n\mathtt{b}^k\mathtt{a}^n, \mathtt{a}^n\mathtt{b}^k\mathtt{c}^k : k,n \geq 1\}$.

We shall now show that the languages generated by strict deterministic grammars are exactly the strict deterministic languages. This justifies the terminology in retrospect. To begin, we shall draw a few conclusions from the definitions. If $G = \langle \mathtt{S}, N, A, R\rangle$ is strict deterministic and $R' \subseteq R$ then $G' = \langle \mathtt{S}, N, A, R'\rangle$ is strict deterministic as well. Therefore, for a strict deterministic grammar we can construct a weakly equivalent strict deterministic slender grammar. We denote by $\vec{\alpha} \Rightarrow_L^n \vec{\gamma}$ the fact that there is a leftmost derivation of length n of $\vec{\gamma}$ from $\vec{\alpha}$.

Lemma 2.49 *Let G be a CFG with a strict partition Π. Then the following is true. For $C, C' \in N$ and $\vec{\alpha}, \vec{\gamma}_1, \vec{\gamma}_2 \in (N \cup A)^*$: if $C \equiv C'$ and $C \Rightarrow_L^n \vec{\alpha}\,\vec{\gamma}_1$ as well as $C' \Rightarrow_L^n \vec{\alpha}\,\vec{\gamma}_2$ then either*

① $\vec{\gamma}_1, \vec{\gamma}_2 \neq \varepsilon$ *and* ${}^{(1)}\vec{\gamma}_1 \equiv {}^{(1)}\vec{\gamma}_2$ *or*

② $\vec{\gamma}_1 = \vec{\gamma}_2 = \varepsilon$ *and* $C = C'$.

The proof is an easy induction over the length of the derivation.

Lemma 2.50 *Let G be slender and strict deterministic. Then if $C \Rightarrow_L^+ D\,\vec{\alpha}$ we have $C \not\equiv D$.*

Proof. Assume $C \Rightarrow_L^n D\,\vec{\alpha}$. Then because of Lemma 2.49 we have for all $k \geq 1$: $C \Rightarrow_L^{kn} D\,\vec{\gamma}$ for some $\vec{\gamma}$. From this it follows that there is no terminating leftmost derivation from C. This contradicts the fact that G is slender. □

It follows that a strict deterministic grammar is not left recursive, that is, $A \Rightarrow_L^+ A\,\vec{\alpha}$ cannot hold. We can construct a Greibach Normal Form for G in the following way. Let $\rho = C \to \alpha\,\vec{\gamma}$ be a rule. If $\alpha \notin A$ then we skip ρ by replacing it with the set of all rules $C \to \vec{\eta}\,\vec{\gamma}$ such that $\alpha \to \vec{\eta} \in R$. Then Lemma 2.49 assures us that Π is a strict partition also for the new grammar. This operation we repeat as often as necessary. Since G is not left recursive, this process terminates.

Theorem 2.51 *For every strict deterministic grammar G there is a strict deterministic grammar H in Greibach Normal Form such that* $L(G) = L(H)$.

Now for the promised correspondence between strict deterministic languages and strict deterministic grammars.

Lemma 2.52 *Let L be strict deterministic. Then there exists a deterministic automaton with a single accepting state which accepts L by stack.*

Proof. Let $\mathfrak{A}$ be given. We add a new state q into which the automaton changes as soon as the stack is empty. □

Lemma 2.53 *Let* $\mathfrak{A}$ *be a deterministic automaton with a single accepting state. Then* $G(\mathfrak{A})$ *is strict deterministic.*

Proof. Let $\mathfrak{A} = \langle Q, i_0, A, F, D, \#, \delta \rangle$. By the preceding lemma we may assume that $F = \{q_f\}$. Now let $G(\mathfrak{A})$ defined as in (2.53). Put

$$\alpha \equiv \beta \quad :\Leftrightarrow \quad \begin{cases} & \alpha, \beta \in A \\ \text{or} & \alpha = [q, Z, q'], \beta = [q, Z, q''] \\ & \quad \text{for some } q, q', q'' \in Q, Z \in D. \end{cases} \tag{2.59}$$

We show that $\equiv$ is a strict partition. To this end, let $[q, Z, q'] \to \vec{\alpha}\vec{\gamma}_1$ and $[q, Z, q''] \to \vec{\alpha}\vec{\gamma}_2$ be two rules. Assume first of all $\vec{\gamma}_1, \vec{\gamma}_2 \neq \varepsilon$. Case 1. $\vec{\alpha} = \varepsilon$. Consider $\zeta_i := {}^{(1)}\vec{\gamma}_i$. If $\zeta_1 \in A$ then also $\zeta_2 \in A$, since $\mathfrak{A}$ is deterministic. If on the other hand $\zeta_1 \notin A$ then we have $\zeta_1 = [q, Y_0, q_1]$ and $\zeta_2 = [q, Y_0, q'_1]$, and so $\zeta_1 \equiv \zeta_2$. Case 2. $\vec{\alpha} \neq \varepsilon$. Let then $\eta := {}^{(1)}\vec{\alpha}$. If $\eta \in A$, then we now have $\zeta_1 = [q_i, Y_i, q_{i+1}]$ and $\zeta_2 = [q_i, Y_i, q'_{i+1}]$ for some $q_i, q_{i+1}, q'_{i+1} \in Q$. This completes this case.

Assume now $\vec{\gamma}_1 = \varepsilon$. Then $\vec{\alpha}\vec{\gamma}_1$ is a prefix of $\vec{\alpha}\vec{\gamma}_2$. Case 1. $\vec{\alpha} = \varepsilon$. Then $\vec{\alpha}\vec{\gamma}_2 = \varepsilon$, hence $\vec{\gamma}_2 = \varepsilon$. Case 2. $\vec{\alpha} \neq \varepsilon$. Then it is easy to see that $\vec{\gamma}_2 = \varepsilon$. Hence in both cases we have $\vec{\gamma}_2 = \varepsilon$, and so $q' = q''$. This shows the claim. □

Theorem 2.54 *Let L be a strict deterministic language. Then there exists a strict deterministic grammar G such that* $L(G) = L$.

The strategy to put a string onto the stack and then subsequently remove it from there has prompted the following definition. A **stack move** is a move where the machine writes a symbol onto the stack or removes a symbol from the stack. (So the stack either increases in length or it decreases.) The automaton is said to make a **turn** if in the last stack move it increased the stack and now it decreases it or, conversely, in the last stack move it diminishes the stack and now increases it.

Definition 2.55 *A language L is called an n–**turn language** if there is a pushdown automaton which recognizes every string from L with at most n turns. L is **ultralinear** if it is an n–turn language for some $n \in \omega$.*

Notice that a CFL is n–turn exactly if there is an automaton which accepts L and in which for every string $\vec{x}$ *every* computation needs at most n turns. For given any automaton $\mathfrak{K}$ which recognizes L, we build another automaton $\mathfrak{L}$ which has the same computations as $\mathfrak{K}$ except that they are terminated before the $n+1$st turn. This is achieved by adding a memory that counts the number of turns.

We shall not go into the details of ultralinear languages. One case is worth noting, that of 1–turn languages. A CFG is called **linear** if in every rule $X \to \vec{\alpha}$ the string $\vec{\alpha}$ contains at most one occurrence of a nonterminal symbol. A language is **linear** if it is generated by a linear grammar.

Theorem 2.56 *A CFL is linear iff it is 1–turn.*

Proof. Let G be a linear grammar. Without loss of generality a rule is of the form $X \to aY$ or $X \to Ya$. Further, there are rules of the form $X \to \varepsilon$. We construct the following automaton. $D := \{\#\} \cup N$, where # is the beginning of the stack, $Q := \{+,-,q\}$, $i_0 := +$, $F := \{q\}$. Further, for $x \in A$ we put $\delta(+,X,x) := \{\langle +,Y\rangle\}$ if $X \to xY \in R$ and $\delta(+,X,\varepsilon) := \{\langle +,Y\rangle\}$ if $X \to Yx \in R$; let $\delta(-,Y,x) := \{\langle -,\varepsilon\rangle\}$ if $X \to Yx \in R$. And finally $\delta(\langle +,X,x\rangle) := \{\langle -,\varepsilon\rangle\}$ if $X \to x \in R$. Finally, $\delta(-,\#,\varepsilon) := \{\langle q,\varepsilon\rangle\}$. This defines the automaton $\mathfrak{K}(G)$. It is not hard to show that $\mathfrak{K}(G)$ only admits computations without stack moves. For if the automaton is in state $+$ the stack may not decrease unless the automaton changes into the state $-$. If it is in $-$, the stack may not increase and it may only be changed into a state $-$, or, finally, into q. We leave it to the reader to check that $L(\mathfrak{K}(G)) = L(G)$. Therefore $L(G)$ is a 1–turn language. Conversely, let $\mathfrak{K}$ be an automaton which allows computations with at most one turn. It is then clear that if the stack is emptied the automaton cannot put anything on it. The automaton may only fill the stack and later empty it. Let us consider the automaton $G(\mathfrak{K})$ as defined above. Then all rules are of the form $X \to x\vec{Y}$ with $x \in A_\varepsilon$. Let $\vec{Y} = Y_0Y_1 \cdots Y_{n-1}$. We claim that every Y_i–production for $i > 0$ is of the form $Y_i \to a$ or $Y_i \to X$. If not, there is a computation in which the automaton makes two turns, as we have indicated above. (This argument makes tacit use of the fact that the automaton possesses a computation where it performs a transition to $Y_i = [p,X,q]$ that is to say, that it goes from p to q where X is the topmost stack symbol.

If this is not the case, however, then the transitions can be eliminated without harm from the automaton.) Now it is easy to eliminate the rules of the form $Y_i \to X$ by skipping them. Subsequent skipping of the rules $Y_i \to a$ yields a linear grammar. □

The automata theoretic analyses suggest that the recognition problem for CFLs must be quite hard. However, this is not the case. It turns out that the recognition and parsing problem are solvable in $O(n^3)$ steps. To see this, let a grammar G be given. We assume without loss of generality that G is in Chomsky Normal Form. Let $\vec{x}$ be a string of length n. As a first step we try to list all substrings which are constituents, together with their category. If $\vec{x}$ is a constituent of category S then $\vec{x} \in L(G)$; if it is not, then $\vec{x} \notin L(G)$. In order to enumerate the substrings we use an $(n+1) \times (n+1)$–matrix whose entries are subsets of N. Such a matrix is called a **chart**. Every substring is defined by a pair $\langle i, j\rangle$ of numbers, where $0 \leq i < j \leq n+1$. In the cell $\langle i, j\rangle$ we enter all $X \in N$ for which the substring $x_i x_{i+1} \cdots x_{j-1}$ is a constituent of category X. In the beginning the matrix is empty. Put $d := i - j$. Now we start by filling the matrix starting at $d = 1$ and counting up to $d = n$. For each d, we go from $i = 0$ until $i = n - d$. So, we begin with $d = 1$ and compute for $i = 0$, $i = 1$, $i = 2$ and so on. Then we set $d := 2$ and compute for $i = 0$, $i = 1$ etc. We consider the pair $\langle d, i\rangle$. The substring $x_i \cdots x_{i+d}$ is a constituent of category X iff it decomposes into substrings $\vec{y} = x_i \cdots x_{i+e}$ and $\vec{z} = x_{i+e+1} \cdots x_{i+d}$ such that there is a rule $X \to YZ$ where $\vec{y}$ is a constituent of category Y and $\vec{z}$ is a constituent of category Z. This means that the set of all $X \in N$ which we enter at $\langle i, i+d\rangle$ is computed from all decompositions into substrings. There are $d - 1 \leq n$ such decomposition. For every decomposition the computational effort is limited and depends only on a constant c_G whose value is determined by the grammar. For every pair we need $c_G \cdot (n+1)$ steps. Now there exist $\binom{n}{2}$ proper subwords. Hence the effort is bounded by $c_G \cdot n^3$.

In Figure 8 we have shown the computation of a chart based on the word `abaabb`. Since the grammar is invertible any substring has at most one category. In general, this need not be the case. (Because of Theorem 2.27 we can always assume the grammar to be invertible.)

$$
\begin{array}{lcl}
\mathtt{S} & \to & \mathtt{SS} \mid \mathtt{AB} \mid \mathtt{BA} \\
\mathtt{A} & \to & \mathtt{AS} \mid \mathtt{SA} \mid \mathtt{a} \\
\mathtt{B} & \to & \mathtt{BS} \mid \mathtt{SB} \mid \mathtt{b}
\end{array}
\tag{2.60}
$$

The construction of the chart is as follows. Let $C_{\vec{x}}(i, j)$ be the set of all nonterminals X such that $X \vdash_G x_i x_{i+1} \cdots x_{j-1}$. Further, for two nonterminals X

			S		
		A		B	
	∅		S		S
	A		A		A		B
S		S		∅		S		∅
A		B		A		A		B		B
a		b		a		a		b		b

Figure 8. A Chart for `abaabb`

and $Y\ X \odot Y := \{Z : Z \to XY \in R\}$ and for sets $\mathbb{U}, \mathbb{V} \subseteq N$ let

$$\mathbb{U} \odot \mathbb{V} := \bigcup \langle X \odot Y : X \in \mathbb{U}, Y \in \mathbb{V} \rangle \tag{2.61}$$

Now we can compute $C_{\vec{x}}(i,j)$ inductively. The induction parameter is $j-i$. If $j-i=1$ then $C_{\vec{x}}(i,j) = \{X : X \to x \in R\}$. If $j-i>1$ then the following equation holds.

$$C_{\vec{x}}(i,j) = \bigcup_{i<k<j} C_{\vec{x}}(i,k) \odot C_{\vec{x}}(k,j) \tag{2.62}$$

We always have $j-k, k-i < j-i$. Now let $\vec{x} \in L(G)$. How can we find a derivation for $\vec{x}$? To that end we use the fully computed chart. We begin with $\vec{x}$ and decompose it in an arbitrary way; since $\vec{x}$ has the category `S`, there must be a rule $\mathtt{S} \to XY$ and a decomposition into $\vec{x}$ of category X and $\vec{y}$ of category Y. Or $\vec{x} = a \in A$ and $\mathtt{S} \to a$ is a rule. If the composition has been found, then we continue with the substrings $\vec{x}$ and $\vec{y}$ in the same way. Every decomposition needs some time, which only depends on G. A substring of length i has $i \leq n$ decompositions. In our analysis we have at most $2n$ substrings. This follows from the fact that in a properly branching tree with n leaves there are at most $2n$ nodes. In total we need time at most $d_G \cdot n^2$ for a certain constant d_G which only depends on G.

From this it follows that in general even if the grammar is not in Chomsky Normal Form the recognition and analysis only needs $O(n^3)$ steps where

at the same time we only need $O(n^2)$ cells. For let G be given. Now transform G into 2–standard form into the grammar G^2. Since $L(G^2) = L(G)$, the recognition problem for G is solvable in the same amount of time as G^2. One needs $O(n^2)$ steps to construct a chart for $\vec{x}$. One also needs an additional $O(n^2)$ steps in order to create a G–tree for $\vec{x}$ and $O(n)$ steps to turn this into a derivation.

However, this is not already a proof that the problem is solvable in $O(n^3)$ steps and $O(n^2)$ space, for we need to find a Turing machine which solves the problem in the same time and space. This is possible; this has been shown independently by Cocke, Kasami and Younger.

Theorem 2.57 (Cocke, Kasami, Younger) *CFLs have the following multitape complexity.*

① $CFL \subseteq \boldsymbol{DTIME}(n^3)$.

② $CFL \subseteq \boldsymbol{DSPACE}(n^2)$.

Proof. We construct a deterministic 3 tape Turing machine which only needs $O(n^2)$ space and $O(n^3)$ time. The essential trick consists in filling the tape. Also, in addition to the alphabet A we need an auxiliary alphabet consisting of $\mathtt{B}$ and $\mathtt{Q}$ as well as for every $U \subseteq N$ a symbol $[U]$ and a symbol $[U]^{\surd}$. On Tape 1 we have the input string, $\vec{x}$. Put $C(i,j) := C_{\vec{x}}(i,j)$. Let $\vec{x}$ have length n. On Tape 1 we construct a sequence of the following form.

$$\mathtt{Q}\mathtt{B}^n\mathtt{Q}\mathtt{B}^{n-1}\mathtt{Q}\cdots\mathtt{Q}\mathtt{B}\mathtt{B}\mathtt{Q}\mathtt{B}\mathtt{Q} \tag{2.63}$$

This is the skeleton of the chart. We call a sequence of $\mathtt{B}$s in between two $\mathtt{Q}$s a **block**. The first block is being filled as follows. The string $\vec{x}$ is deleted step by step and the sequence $\mathtt{B}^n$ is being replaced by the sequence of the $C(i,i+1)$. This procedure requires $O(n^2)$ steps. For every d from 1 to $n-1$ we shall fill the $d+1$st block. So, let d be given. On Tape 2 we write the sequence

$$\begin{aligned}&\mathtt{Q}[C(0,1)][C(0,2)]\cdots[C(0,d)]^\frown\\&{}^\frown\mathtt{Q}[C(1,2)][C(1,3)]\cdots[C(1,d+1)]^\frown\cdots\\&{}^\frown\mathtt{Q}[C(n-d,n-d+1)][C(n-d,n-d+2)]\cdots[C(n-d,n)]\mathtt{Q}\end{aligned} \tag{2.64}$$

On Tape 3 we write the sequence

$$\begin{aligned}&\mathtt{Q}[C(0,d)][C(1,d)]\cdots[C(d-1,d)]^\frown\\&{}^\frown\mathtt{Q}[C(1,d+1)][C(2,d+1)]\cdots[C(d,d+1)]^\frown\cdots\\&{}^\frown\mathtt{Q}[C(n-d,n)][C(n-d+1,n)]\cdots[C(n-1,n)]\mathtt{Q}\end{aligned} \tag{2.65}$$

From this sequence we can compute the $d+1$st block quite fast. The automaton has to traverse the first block on Tape 2 and the second block on Tape 3 cogradiently and memorize the result of $C(0,j) \odot C(j,d+1)$. When it reaches the end it has computed $C(0,d+1)$ and can enter it on Tape 1. Now it moves on to the next block on the second and the third tape and computes $C(1,d+2)$. And so on. It is clear that the computation is linear in the length of the Tape 2 (and the Tape 3) and therefore needs $O(n^2)$ time. At the end of this procedure Tape 2 and 3 are emptied. Also this needs quadratic time. At the end we need to consider that the filling of Tapes 2 and 3 needs $O(n^2)$ time. Then for every d the time consumption is at most $O(n^2)$ and in total $O(n^3)$. For this we first write `Q` and position the head of Tape 1 on the element $[C(0,1)]$. We write $[C(0,1)]$ onto Tape 2 and $[C(0,1)]^{\surd}$ onto Tape 1. (So, we 'tick off' the symbol. This helps us to remember what we did.) Now we advance to $[C(1,2)]$ copy the result onto Tape 2 and replace it by $[C(1,2)]^{\surd}$. And so on. This only needs linear time; for the symbols $[C(i,i+1)]$ we recognize because they are placed before the `Q`. If we are ready we write `Q` onto Tape 2 and move on Tape 1 on to the beginning and then to the first symbol to the right of a 'ticked off' symbol. This is $[C(1,2)]$. We copy this symbol onto Tape 2 and tick it off. Now we move on to the next symbol to the right of the symbol which has been ticked off, copy it and tick it off. In this way Tape 2 is filled in quadratic time. At last the symbols that have been ticked off are being ticked 'on', which needs $O(n^2)$ time. Analogously the Tape 3 is filled. □

Exercise 58. Prove Proposition 2.35.

Exercise 59. Prove Theorem 2.41. *Hint.* Show that the number of ε–moves of an automaton $\mathfrak{A}$ in scanning of the string $\vec{x}$ is bounded by $k_{\mathfrak{A}} \cdot |\vec{x}|$, where $k_{\mathfrak{A}}$ is a number that depends only on $\mathfrak{A}$. Now code the behaviour of an arbitrary pushdown automaton using a 2–tape Turing machine and show that to every move of the pushdown automaton corresponds a bounded number of steps of the Turing machine.

Exercise 60. Show that a CFL is 0–turn iff it is regular.

Exercise 61. Give an algorithm to code a chart onto the tape of a Turing machine.

Exercise 62. Sketch the behaviour of a deterministic Turing machine which recognizes a given CFL using $O(n^2)$ space.

Exercise 63. Show that $\{\vec{w}\,\vec{w}^T : \vec{w} \in A^*\}$ is context free but not deterministic.

Exercise 64. Construct a deterministic automaton which recognizes a given Dyck–language.

Exercise 65. Prove Theorem 2.46.

4. Ambiguity, Transparency and Parsing Strategies

In this section we will deal with the relationship between strings and trees. As we have explained in Section 1.6, there is a bijective correspondence between derivations in G and derivations in the corresponding graph grammar γG. Moreover, every derivation $\Delta = \langle A_i : i < p\rangle$ of G defines an exhaustively ordered tree $\mathfrak{B}$ with labels in $N \cup A$ whose associated string is exactly $\vec{\alpha}_p$, where $A_{p-1} = \langle \vec{\alpha}_{p-1}, C_{p-1}, \vec{\alpha}_p\rangle$. If $\vec{\alpha}_p$ is not a terminal string, the labels of the leaves are also not all terminal. We call such a tree a **partial** G–**tree**.

Definition 2.58 *Let G be a CFG. $\vec{\alpha}$ is called a G–**constituent of category** A if $A \vdash_G \vec{\alpha}$. Let $\mathfrak{B}$ be a G–tree with associated string $\vec{x}$ and $\vec{y}$ a substring of $\vec{x}$. Assume further that $\vec{y}$ is a G–constituent of category A and $\vec{x} = D(\vec{y})$. The occurrence D of $\vec{y}$ in $\vec{x}$ is called an **accidental G–constituent of category** A **in** $\mathfrak{B}$ if it is not a G–constituent of category A in $\mathfrak{B}$.*

We shall illustrate this terminology with an example. Let G be the following grammar.

$$\begin{array}{lcl} \mathtt{S} & \to & \mathtt{SS} \mid \mathtt{AB} \mid \mathtt{BA} \\ \mathtt{A} & \to & \mathtt{AS} \mid \mathtt{SA} \mid \mathtt{a} \\ \mathtt{B} & \to & \mathtt{BS} \mid \mathtt{SB} \mid \mathtt{b} \end{array} \tag{2.66}$$

The string $\vec{x} = \mathtt{abaabb}$ has several derivations, which generate among other the following bracketing analyses.

$$(\mathtt{a}(\mathtt{b}(\mathtt{a}((\mathtt{ab})\mathtt{b})))), \quad ((\mathtt{ab})(((\mathtt{a}(\mathtt{ab}))\mathtt{b}))) \tag{2.67}$$

We now list all G–constituents which occur in $\vec{x}$:

$$\begin{array}{l} \mathtt{A} : \mathtt{a}, \mathtt{aab}, \mathtt{aba}, \mathtt{baa}, \mathtt{abaab} \\ \mathtt{B} : \mathtt{b}, \mathtt{abb} \\ \mathtt{S} : \mathtt{ab}, \mathtt{aabb}, \mathtt{abaabb} \end{array} \tag{2.68}$$

Some constituents occur several times, for example ab in $\langle \varepsilon, \mathtt{aabb} \rangle$ and also in $\langle \mathtt{aba}, \mathtt{b} \rangle$. Now we look at the first bracketing, (a(b(a((ab)b)))). The constituents are a (contexts: $\langle \varepsilon, \mathtt{baabb} \rangle$, $\langle \mathtt{ab}, \mathtt{abb} \rangle$, $\langle \mathtt{aba}, \mathtt{bb} \rangle$), b, ab (for example in the context: $\langle \mathtt{aba}, \mathtt{b} \rangle$), abb in the context $\langle \mathtt{aba}, \varepsilon \rangle$, aabb, baabb and abaabb. These are the constituents of the tree. The occurrence $\langle \varepsilon, \mathtt{aabb} \rangle$ of ab in ababb is therefore an accidental occurrence of a G–constituent of category S in that tree. For although ab is a G–constituent, this occurrence in the tree is not a constituent occurrence of it. Notice that it may happen that $\vec{y}$ is a constituent of the tree $\mathfrak{B}$ but that as a G–constituent of category C it occurs accidentally since its category in $\mathfrak{B}$ is $D \neq C$.

Definition 2.59 *A grammar G is called **transparent** if no G–constituent occurs accidentally in a G–string. A grammar which is not transparent will be called **opaque**. A language for which no transparent grammar exists will be called **inherently opaque**.*

An example shall illustrate this. For any given signature Ω, Polish Notation can be generated by a transparent grammar.

$$\mathtt{S} \to \mathtt{F}_{\Omega(f)}\mathtt{S}^{\Omega(f)} \qquad \mathtt{F}_{\Omega(f)} \to f \tag{2.69}$$

This defines the grammar Π_Ω for PN_Ω. Moreover, given a string $\vec{x}$ generated by this grammar, the subterm occurrences of $\vec{x}$ under a given analysis are in one to one correspondence with the subcontituents of category S. An occurrence of an n–ary function symbol is a constituent of type $\mathtt{F}_n$. We shall show that this grammar is not only unambiguous, it is transparent.

Let $\vec{x} = x_0 x_1 \cdots x_{n-1}$ be a string. Then let $\gamma(\vec{x}) := \sum_{i<n} \gamma(x_i)$, where for every $f \in F$, $\gamma(f) := \Omega(f) - 1$. (So, if $\Omega(f) = 0$, $\gamma(f) = -1$.) The proof of the following is left as an exercise.

Lemma 2.60 *$\vec{x} \in \mathrm{PN}_\Omega$ iff (a) $\gamma(\vec{x}) = -1$ and (b) for every proper prefix $\vec{y}$ of $\vec{x}$ we have $\gamma(\vec{y}) \geq 0$.*

It follows from this theorem that no proper prefix of a term is a term. (However, a suffix of a term may again be a term.) The constituents are therefore all the substrings that have the properties (a) and (b). We show that the grammar is transparent. Now suppose that $\vec{x}$ contains an accidental occurrence of a term $\vec{y}$. Then this occurrence overlaps properly with a constituent $\vec{z}$. Without loss of generality $\vec{y} = \vec{u}^\frown\vec{v}$ and $\vec{z} = \vec{v}^\frown\vec{w}$ (with $\vec{u}, \vec{w} \neq \varepsilon$). It follows that $\gamma(\vec{v}) = \gamma(\vec{y}) - \gamma(\vec{u}) < 0$ since $\gamma(\vec{u}) \geq 0$. Hence there exists a proper prefix $\vec{u}_1$

of $\vec{u}$ such that $\vec{u}_1 = -1$. (In order to show this one must first conclude that the set $P(\vec{x}) := \{\gamma(\vec{p}) : \vec{p}$ is a prefix of $\vec{x}\}$ is a convex set for every term $\vec{x}$. See Exercise 68.)

Theorem 2.61 *The grammar Π_Ω is transparent.* □

Now look at the languages $\mathtt{a}^+\mathtt{b}$ and $\mathtt{a}^+$. Both are regular. There is a transparent regular grammar for $\mathtt{a}^+\mathtt{b}$. It has the rules $\mathtt{S} \to \mathtt{aB}$, $\mathtt{B} \to \mathtt{AB} \mid \mathtt{b}$. $\mathtt{a}^+$ is on the other hand inherently opaque. For any CFG must generate at least two constituents of the form $\mathtt{a}^p$ and $\mathtt{a}^q$, $q > p$. Now there exist two occurrences of $\mathtt{a}^p$ in $\mathtt{a}^q$ which properly overlap. One of them must be accidental.

Proposition 2.62 $\mathtt{a}^+$ *is inherently opaque.* □

It can easily be seen that if L is transparent and $\varepsilon \in L$, then $L = \{\varepsilon\}$. Also, a language over an alphabet consisting of a single letter can only be transparent if it contains no more than a single string. Many properties of CFGs are undecidable. Transparency is different in this respect.

Theorem 2.63 (Fine) *Let G be a CFG. It is decidable whether or not G is transparent.*

Proof. Let k be the constant from the Pumping Lemma (1.81). This constant can effectively be determined. By Lemma 2.64 there is an accidental occurrence of a constituent iff there is an accidental occurrence of a right hand side of a production. These are of the length $p+1$ where p is the maximum productivity of a rule from G. Further, because of Lemma 2.66 we only need to check those constituents for accidental occurrences whose length does not exceed $p^2 + p$. This can be done in finite amount of time. □

Lemma 2.64 *G is opaque iff there is a production $\rho = A \to \vec{\alpha}$ such that $\vec{\alpha}$ has an accidental occurrence in a partial G–tree.*

Proof. Let $\vec{\varphi}$ be a string of minimal length which occurs accidentally. And let C be an accidental occurrence of $\vec{\varphi}$. Further, let $\vec{\varphi} = \vec{\gamma}_1\vec{\alpha}\vec{\gamma}_2$, and let $A \to \vec{\alpha}$ be a rule. Then two cases may occur. (A) The occurrence of $\vec{\alpha}$ is accidental. Then we have a contradiction to the minimality of $\vec{\varphi}$. (B) The occurrence of $\vec{\alpha}$ is not accidental. Then $\vec{\eta} := \vec{\gamma}_1 A \vec{\gamma}_2$ also occurs accidentally in $C(\vec{\eta})$! (We can undo the replacement $A \to \vec{\alpha}$ in the string $C(\vec{\varphi})$ since $\vec{\alpha}$ is a constituent.) Also this contradicts the minimality of $\vec{\varphi}$. So, $\vec{\varphi}$ is the right hand side of a production. □

Lemma 2.65 *Let G be a CFG without rules of productivity* -1 *and let* $\vec{\alpha}$, $\vec{\gamma}$ *be strings. Further, assume that* $\vec{\gamma}$ *is a G–constituent of category A in which* $\vec{\alpha}$ *occurs accidentally and in which* $\vec{\gamma}$ *is minimal in the following sense: there is no* $\vec{\eta}$ *of category A with (1)* $|\vec{\eta}| < |\vec{\gamma}|$ *and (2)* $\vec{\eta} \vdash_G \vec{\gamma}$ *and (3)* $\vec{\alpha}$ *occurs accidentally in* $\vec{\eta}$*. Then every constituent of length* > 1 *overlaps with the accidental occurrence of* $\vec{\alpha}$*.*

Proof. Let $\vec{\gamma} = \vec{\sigma}_1 \vec{\eta} \vec{\sigma}_2$, $|\vec{\eta}| > 1$, and assume that the occurrence of $\vec{\eta}$ is a constituent of category A which does not overlap with $\vec{\alpha}$. Then $\vec{\alpha}$ occurs accidentally in $\vec{\delta} := \vec{\sigma}_1 A \vec{\sigma}_2$. Further, $|\vec{\delta}| < |\vec{\gamma}|$, contradicting the minimality of $\vec{\gamma}$. □

Lemma 2.66 *Let G be a CFG where the productivity of rules is at least 0 and at most p, and let* $\vec{\alpha}$ *be a string of length n which occurs accidentally. Then there exists a constituent* $\vec{\gamma}$ *of length* $\leq np$ *in which* $\vec{\alpha}$ *occurs accidentally.*

Proof. Let $A \vdash_G \vec{\gamma}$ be minimal in the sense of the previous lemma. Then we have that every constituent of $\vec{\gamma}$ of length > 1 overlaps properly with $\vec{\alpha}$. Hence $\vec{\gamma}$ has been obtained by at most n applications of rules of productivity > 0. Hence $|\vec{\gamma}| \leq np$. □

The property of transparency is stronger than that of unique readability, also known as unambiguity, which is defined as follows.

Definition 2.67 *A CFG G is called* ***unambiguous*** *if for every string* $\vec{x}$ *there is at most one G–tree whose associated string is* $\vec{x}$*. If G is not unambiguous, it is called* ***ambiguous****. A CFL L is called* ***inherently ambiguous*** *if every CFG generating it is ambiguous.*

Proposition 2.68 *Every transparent grammar is unambiguous.*

There exist inherently ambiguous languages. Here is an example.

Theorem 2.69 (Parikh) *The language L is inherently ambiguous.*

(2.70) $$L := \{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^m : n,m \in \omega\} \cup \{\mathtt{a}^m\mathtt{b}^n\mathtt{c}^n : n,m \in \omega\}$$

Proof. L is context free and so there exists a CFG G such that $L(G) = L$. We shall show that G must be ambiguous. There is a number k which satisfies the Pumping Lemma (1.81). Let $n := k!(:= \prod_{i=1}^{k} i)$. Then there exists a decomposition of $\mathtt{a}^{2n}\mathtt{b}^{2n}\mathtt{c}^{3n}$ into

(2.71) $$\vec{u}_1{}^\frown\vec{x}_1{}^\frown\vec{v}_1{}^\frown\vec{y}_1{}^\frown\vec{z}_1$$

in such a way that $|\vec{u}_1| \leq k$. Furthermore, we may also assume that $|\vec{v}_1| \leq k$. It is easy to see that $\vec{x}_1\,\vec{y}_1$ may not contain occurrences of a, b and c at the same time. Since it contains a, it may not contain c. So we have $\vec{x}_1 = \mathtt{a}^p$ and $\vec{y}_1 = \mathtt{b}^p$ for some p. We consider a maximal constituent of (2.71) of the form $\mathtt{a}^q\,\mathtt{b}^{q'}$. Such a constituent must exist. ($\vec{x}_1{}^\frown\vec{v}_1{}^\frown\vec{y}_1$ is of that form.) In it there is a constituent of the form $\mathtt{a}^{q-i}\,\mathtt{b}^{q'-i}$ for some $i < k$. This follows from the Pumping Lemma. Hence we can pump up $\mathtt{a}^i$ and $\mathtt{b}^i$ at the same time and get strings of the form

(2.72) $\quad \mathtt{a}^{2p+ki}\,\mathtt{b}^{2p+ki}\,\mathtt{c}^{3q}$

while there exists a constituent of the form $\mathtt{a}^{2p+ki-r}\,\mathtt{b}^{2p+ki-s}$ for certain $r, s \leq k$. In particular, for $k := p/i$ we get

(2.73) $\quad \mathtt{a}^{3p}\,\mathtt{b}^{3p}\,\mathtt{c}^{3q}$

Now we form a decomposition of $\mathtt{a}^{3n}\mathtt{b}^{2n}\mathtt{c}^{2n}$ into

(2.74) $\quad \vec{u}_2{}^\frown\vec{x}_2{}^\frown\vec{v}_2{}^\frown\vec{y}_2{}^\frown\vec{z}_2$

in such a way that $|\vec{z}_2|, |\vec{v}_2| \leq k$. Analogously we get a constituent of the form $\mathtt{b}^{2p-s'}{}^\frown\mathtt{c}^{2p-r'}$ for certain $r', s' \leq k$. These occurrences overlap. For the left hand constituent contains $3p - s$ many occurrences of b and the right hand constituent contains $3p - s'$ many occurrences of b. Since $3p - s + 3p - s' = 6p - (s + s') > 3p$, these constituents must overlap. However, they are not equal. But this is impossible. So G is ambiguous. Since G was arbitrary, L is inherently ambigous. □

Now we discuss a property which is in some sense the opposite of the property of unambiguity. It says that if a right hand side occurs in a constituent, then under some different analysis this occurrence is actually a constituent occurrence.

Definition 2.70 *A CFG has the* ***NTS–property*** *if from* $C \vdash_G \vec{\alpha}_1{}^\frown\vec{\beta}{}^\frown\vec{\alpha}_2$ *and* $B \to \vec{\beta} \in R$ *follows:* $C \vdash_G \vec{\alpha}_1{}^\frown B^\frown\vec{\alpha}_2$. *A language is called an* ***NTS–language*** *if it has an NTS–grammar.*

The following grammar is not an NTS–grammar.

(2.75) $\quad \mathtt{X} \to \mathtt{aX}, \quad \mathtt{X} \to \mathtt{a}$

For we have $\mathtt{X} \vdash \mathtt{aa}$ but it does not hold that $\mathtt{X} \vdash \mathtt{Xa}$. In general, regular grammars are not NTS. However, we have

Theorem 2.71 *All regular languages are NTS–languages.*

Proof. Assume that L is regular. Then there exists a finite state automaton $\mathfrak{A} = \langle A, Q, q_0, F, \delta \rangle$ such that $L = L(\mathfrak{A})$. Put $N := \{S^\star\} \cup \{L(p,q) : p, q \in Q\}$. Further, put $G := \langle \mathtt{S}^\star, N, A, R \rangle$, where R consists of

$$
\begin{array}{llll}
\mathtt{S}^\star & \to & \mathtt{L}(q_0, q) & (q \in F) \\
\mathtt{L}(p,q) & \to & \mathtt{L}(p,r)\mathtt{L}(r,q) & \\
\mathtt{L}(p,q) & \to & a & (q \in \delta(p,a))
\end{array}
\tag{2.76}
$$

Then we have $\mathtt{L}(p,q) \vdash_G \vec{x}$ iff $q \in \delta(p, \vec{x})$, as is checked by induction. From this follows that $\mathtt{S}^\star \vdash_G \vec{x}$ iff $\vec{x} \in L(\mathfrak{A})$. Hence we have $L(G) = L$. It remains to show that G has the NTS–property. To this end let $\mathtt{L}(p,q) \vdash_G \vec{\alpha}_1{}^\frown\vec{\beta}^\frown\vec{\alpha}_2$ and $\mathtt{L}(r,s) \vdash_G \vec{\beta}$. We have to show that $\mathtt{L}(p,q) \vdash_G \vec{\alpha}_1{}^\frown\mathtt{L}(r,s)^\frown\vec{\alpha}_2$. In order to do this we extend the automaton $\mathfrak{A}$ to an automaton which reads strings from $N \cup A$. Here $q \in \delta(p, C)$ iff for every string $\vec{y}$ with $C \vdash_G \vec{y}$ we have $q \in \delta(p, \vec{y})$. Then it is clear that $q \in \delta(p, \mathtt{L}(p,q))$. Then it still holds that $\mathtt{L}(p,q) \vdash_G \vec{\alpha}$ iff $q \in \delta(p, \vec{\alpha})$. Hence we have $r \in \delta(p, \vec{\alpha}_1)$ and $q \in \delta(s, \vec{\alpha}_2)$. From this follows that $\mathtt{L}(p,q) \vdash_G \mathtt{L}(p,r)\mathtt{L}(r,s)\mathtt{L}(s,q)$ and finally $\mathtt{L}(p,q) \vdash_G \vec{\alpha}_1\mathtt{L}(r,s)\vec{\alpha}_2$. □

If a grammar has the NTS–property, strings can be recognized very fast. We sketch a pushdown automaton that recognizes $L(G)$. Scanning the string from left to right it puts the symbols onto the stack. Using its states the automaton memorizes the content of the stack up to κ symbols deep, where κ is the length of a longest right hand side of a production. If the upper part of the stack matches a right hand side of a production $A \to \vec{\alpha}$ in the appropriate order, then $\vec{\alpha}$ is deleted from the stack and A is put on top of it. At this moment the automaton rescans the upper part of the stack up to κ symbols deep. This is done using a series of empty moves. The automaton pops κ symbols and then puts them back onto the stack. Then it continues the procedure above. It is important that the replacement of a right hand side by a left hand side is done whenever first possible.

Theorem 2.72 *Let G be an NTS–grammar. Then G is deterministic. Furthermore, the recognition and parsing problem are in* ***DTIME***(n).

We shall deepen this result. To this end we abstract somewhat from the pushdown automata and introduce a calculus which manipulates pairs $\vec{\alpha} \vdash \vec{x}$ of strings separated by a turnstile. Here, we think of $\vec{\alpha}$ as the stack of the automaton and $\vec{x}$ as the string to the right of the reading head. It is not really

necessary to have terminal strings on the right hand side; however, the generalization to arbitrary strings is easy to do. There are several operations. The first is called **shift**. It simulates the reading of the first symbol.

$$\text{(2.77)} \qquad \text{shift:} \qquad \frac{\vec{\eta} \ \vdash x\vec{y}}{\vec{\eta}x \vdash \vec{y}}$$

Another operation is **reduce**.

$$\text{(2.78)} \qquad \text{reduce } \rho\text{:} \qquad \frac{\vec{\eta}\,\vec{\alpha} \vdash \vec{x}}{\vec{\eta}X \vdash \vec{x}}$$

Here $\rho = X \to \vec{\alpha}$ must be a G–rule. This calculus shall be called the **shift–reduce–calculus for** G. The following theorem is easily proved by induction on the length of a derivation.

Theorem 2.73 *Let G be a CFG. $\vec{\alpha} \vdash_G \vec{x}$ iff there is a derivation of $\vec{\alpha} \vdash \varepsilon$ from $\varepsilon \vdash \vec{x}$ in the shift–reduce–calculus for G.*

This strategy can be applied to every language. We take the following grammar.

$$\text{(2.79)} \qquad \begin{array}{lcl} \mathtt{S} & \to & \mathtt{ASB} \mid \mathtt{c} \\ \mathtt{A} & \to & \mathtt{a} \\ \mathtt{B} & \to & \mathtt{b} \end{array}$$

Then we have $S \vdash_G \mathtt{aacbb}$. Indeed, we get a derivation shown in Table 4. Of course the calculus does not provide unique solutions. On many occasions we have to guess whether to shift or whether to reduce, and if the latter, then by what rule. Notice namely that if some right hand side of a production is a suffix of a right hand side of another production we have an option. We call a k–**strategy** a function f which tells us for every pair $\vec{\alpha} \vdash \vec{x}$ whether or not we shall shift or reduce (and by which rule). Further, f shall only depend (1) on the reduction rules which can be at all applied to $\vec{\alpha}$ and (2) on the first k symbols of $\vec{x}$. We assume that in case of competition only one rule is chosen. So, a k–strategy is a map $R \times \bigcup_{i<k} A^i$ to $\{s, r\}$. If $\vec{\alpha} \vdash \vec{x}$ is given then we determine the next rule application as follows. Let $\vec{\beta}$ be a suffix of $\vec{\alpha}$ which is reducible. If $f(\vec{\beta}, {}^{(k)}\vec{x}) = s$, then we shift; if $f(\vec{\beta}, {}^{(k)}\vec{x}) = r$ then we apply reduction to $\vec{\beta}$. This is in fact not really unambigous. For a right hand side

Table 4. A Derivation by Shifting and Reducing

ε	$\vdash$ aacbb
a	$\vdash$ acbb
A	$\vdash$ acbb
Aa	$\vdash$ cbb
AA	$\vdash$ cbb
AAc	$\vdash$ bb
AAS	$\vdash$ bb
AASb	$\vdash$ b
AASB	$\vdash$ b
AS	$\vdash$ b
ASb	$\vdash \varepsilon$
ASB	$\vdash \varepsilon$
S	$\vdash \varepsilon$

of a production may be the suffix of a right hand side of another production. Therefore, we look at another property.

(2.80) If $\rho_1 = X_1 \to \vec{\beta}_1 \in R$ and $\rho_2 = X_2 \to \vec{\beta}_2 \in R$, $\rho_1 \neq \rho_2$, and if $|\vec{y}| \leq k$ then $f(\vec{\beta}_1, \vec{y})$ or $f(\vec{\beta}_2, \vec{y})$ is undefined.

Definition 2.74 *A CFG G is called an* ***LR***(k)*–**grammar** if not* $\mathtt{S} \Rightarrow^+ \mathtt{S}$ *and if for some* $k \in \omega$ *there is a k–strategy for the shift–and–reduce calculus for G. A language is called an LR*(k)*–**language** if it is generated by some LR*(k)*–grammar.*

Theorem 2.75 *A CFG is an LR*(k)*–grammar if the following holds: Suppose that* $\vec{\eta}_1\vec{\alpha}_1\vec{x}_1$ *and* $\vec{\eta}_2\vec{\alpha}_2\vec{x}_2$ *have a rightmost derivation and that with* $p := |\vec{\eta}_1\vec{\alpha}_1| + k$ *we have*

(2.81) $^{(p)}\vec{\eta}_1\vec{\alpha}_1\vec{x}_1 = {}^{(p)}\vec{\eta}_2\vec{\alpha}_2\vec{x}_2$

Then $\vec{\eta}_1 = \vec{\eta}_2$, $\vec{\alpha}_1 = \vec{\alpha}_2$ *and* $^{(k)}\vec{x}_1 = {}^{(k)}\vec{x}_2$.

This theorem is not hard to show. It says that the strategy may be based indeed only on the k–prefix of the string which is to be read. This is essentially the

property (2.80). One needs to convince oneself that a derivation in the shift–reduce–calculus corresponds to a rightmost derivation, provided reduction is scheduled as early as possible.

Theorem 2.76 *$LR(k)$–languages are deterministic.*

We leave the proof of this fact to the reader. The task is to show how to extract a deterministic automaton from a strategy. The following is easy.

Lemma 2.77 *Every $LR(k)$–language is an $LR(k+1)$–language.*

So we have the following hierarchy.

$$LR(0) \subseteq LR(1) \subseteq LR(2) \subseteq LR(3)\ldots \tag{2.82}$$

This hierarchy is stationary already from $k = 1$.

Lemma 2.78 *Let $k > 0$. If L is an $LR(k+1)$–language then L also is an $LR(k)$–language.*

Proof. For a proof we construct an $LR(k)$–grammar $G^{>}$ from an $LR(k+1)$–grammar G. For simplicity we assume that G is in Chomsky Normal Form. The general case is easily shown in the same way. The idea behind the construction is as follows. A constituent of $G^{>}$ corresponds to a constituent of G which has been shifted one letter to the right. To implement this idea we introduce new symbols, $[a,X,b]$, where $a,b \in A$, $X \in N$, and $[a,X,\varepsilon]$, $a \in A$. The start symbol of $G^{>}$ is the start symbol of G. The rules are as follows, where a,b,c range over A.

$$\begin{array}{lcll}
\mathtt{S} & \to & \varepsilon & \text{if } \mathtt{S} \to \varepsilon \in R, \\
\mathtt{S} & \to & a\,[a,\mathtt{S},\varepsilon] & a \in A, \\
[a,X,b] & \to & [a,Y,c]\,[c,Z,b] & \text{if } X \to YZ \in R, \\
[a,X,\varepsilon] & \to & [a,Y,c]\,[c,Z,\varepsilon] & \text{if } X \to YZ \in R, \\
[a,X,b] & \to & b & \text{if } X \to a \in R, \\
[a,X,\varepsilon] & \to & \varepsilon & \text{if } X \to a \in R.
\end{array} \tag{2.83}$$

By induction on the length of a derivation the following is shown.

$$[a,X,b] \vdash_{G^{>}} \vec{\alpha}\,b \quad \Leftrightarrow \quad X \vdash_G a\,\vec{\alpha} \tag{2.84a}$$

$$[a,X,\varepsilon] \vdash_{G^{>}} \vec{\alpha} \quad \Leftrightarrow \quad X \vdash_G a\,\vec{\alpha} \tag{2.84b}$$

From this we can deduce that $G^{>}$ is an $LR(k)$–grammar. To this end let $\vec{\eta}_1\vec{\alpha}_1\vec{x}_1$ and $\vec{\eta}_2\vec{\alpha}_2\vec{x}_2$ be rightmost derivable in $G^{>}$, and let $p := |\vec{\eta}_1\vec{\alpha}_1| + k$ as well as

$$(2.85) \qquad {}^{(p)}\vec{\eta}_1\vec{\alpha}_1\vec{x}_1 = {}^{(p)}\vec{\eta}_2\vec{\alpha}_2\vec{x}_2$$

Then $a\vec{\eta}_1\vec{\alpha}_1\vec{x}_1 = \vec{\eta}_1'\vec{\alpha}_1'b\vec{x}_1$ for some $a, b \in A$ and some $\vec{\eta}_1'$, $\vec{\alpha}_1'$ with $a\vec{\eta}_1 = \vec{\eta}_1'c$ for $c \in A$ and $c\vec{\alpha}_1 = \vec{\alpha}_1'b$. Furthermore, we have $a\vec{\eta}_2\vec{\alpha}_2\vec{x}_2 = \vec{\eta}_2'\vec{\alpha}_2'b\vec{x}_2$, $a\vec{\eta}_2 = \vec{\eta}_2'c$ and $c\vec{\alpha}_2 = \vec{\alpha}_2'$ for certain $\vec{\eta}_2'$ and $\vec{\alpha}_2'$. Hence we have

$$(2.86) \qquad {}^{(p+1)}\vec{\eta}_1'\vec{\alpha}_1'b\vec{x}_1 = {}^{(p+1)}\vec{\eta}_2'\vec{\alpha}_2'b\vec{x}_2$$

and $p+1 = |\vec{\eta}_1'\vec{\alpha}_1'| + k + 1$. Furthermore, the left hand and the right hand string have a rightmost derivation in G. From this it follows, since G is an $LR(k+1)$–grammar, that $\vec{\eta}_1' = \vec{\eta}_2'$ and $\vec{\alpha}_1' = \vec{\alpha}_2'$, as well as ${}^{(k+1)}b\vec{x}_1 = {}^{(k+1)}b\vec{x}_2$. From this we get $\vec{\eta}_1 = \vec{\eta}_2$, $\vec{\alpha}_1 = \vec{\alpha}_2$ and ${}^{(k)}\vec{x}_1 = {}^{(k)}\vec{x}_2$, as required. □

Now we shall prove the following important theorem.

Theorem 2.79 *Every deterministic language is an $LR(1)$–language.*

The proof is relatively long. Before we begin we shall prove a few auxiliary theorems which establish that strictly deterministic languages are exactly the languages that are generated by strict deterministic grammars, and that they are unambiguous and in $LR(0)$. This will give us the key to the general theorem.

We still owe the reader a proof that strict deterministic grammars only generate strict deterministic languages. This is essentially the consequence of a property that we shall call **left transparency**. We say $\vec{\alpha}$ **occurs in** $\vec{\eta}_1\,\vec{\alpha}\,\vec{\eta}_2$ **with left context** $\vec{\eta}_1$.

Definition 2.80 *Let G be a CFG. G is called **left transparent** if a constituent may never occur in a string accidentally with the same left context. This means that if $\vec{x}$ is a constituent of category C in $\vec{y}_1\vec{x}\vec{y}_2$ and if $\vec{z} := \vec{y}_1\vec{x}\vec{y}_3$ is a G–string then $\vec{x}$ also is a constituent of category C in $\vec{z}$.*

For the following theorem we need a few definitions. Let $\mathfrak{B}$ be a tree and $n \in \omega$ a natural number. Then ${}^{(n)}\mathfrak{B}$ denotes the tree which consists of all nodes above the first n leaves from the left. Let P the set of leaves of $\mathfrak{B}$, say $P = \{p_i : i < q\}$, and let $p_i \sqsubset p_j$ iff $i < j$. Then put $N_n := \{p_i : i < n\}$, and $O_n := \uparrow N_n$. ${}^{(n)}\mathfrak{B} := \langle O_n, r, <, \sqsubset\rangle$, where $<$ and $\sqsubset$ are the relations relativized to O_n. If ℓ is a labelling function and $\mathfrak{T} = \langle \mathfrak{B}, \ell\rangle$ a labelled tree then let ${}^{(n)}\mathfrak{T} :=$

$\langle {}^{(n)}\mathfrak{B}, \ell \upharpoonright O_n \rangle$. Again, we denote $\ell \upharpoonright O_n$ simply by ℓ. We remark that the set $R_n := {}^{(n)}\mathfrak{B} - {}^{(n-1)}\mathfrak{B}$ is linearly ordered by $<$. We look at the largest element z from R_n. Two cases arise. (a) z has no right sister. (b) z has a right sister. In Case (a) the constituent of the mother of z is closed at the transition from ${}^{(n-1)}\mathfrak{B}$ to ${}^{(n)}\mathfrak{B}$. Say that y is at the **right edge** of $\mathfrak{T}$ if there is no z such that $y \sqsubset z$. Then $\uparrow R_n$ consists exactly of the elements which are at the right edge of ${}^{(n)}\mathfrak{B}$ and R_n consists of all those elements which are at the right edge of ${}^{(n)}\mathfrak{B}$ but not contained in ${}^{(n-1)}\mathfrak{B}$. Now the following holds.

Proposition 2.81 *Let G be a strict deterministic grammar. Then G is left transparent. Furthermore: let $\mathfrak{T}_1 = \langle \mathfrak{B}_1, \ell_1 \rangle$ and $\mathfrak{T}_2 = \langle \mathfrak{B}_2, \ell_2 \rangle$ be partial G–trees such that the following holds.*

① *If C_i is the label of the root of $\mathfrak{T}_i$ then $C_1 \equiv C_2$.*

② ${}^{(n)}k(\mathfrak{T}_1) = {}^{(n)}k(\mathfrak{T}_2)$.

Then there is an isomorphism $f\colon {}^{(n+1)}\mathfrak{B}_1 \twoheadrightarrow {}^{(n+1)}\mathfrak{B}_2$ such that $\ell_2(f(x)) = \ell_1(x)$ in case x is not at the right edge of ${}^{(n+1)}\mathfrak{B}_1$ and $\ell_2(f(x)) \equiv \ell_1(x)$ otherwise.

Proof. We show the theorem by induction on n. We assume that it holds for all $k < n$. If $n = 0$, it holds anyway. Now we show the claim for n. There exists by assumption an isomorphism $f_n\colon {}^{(n)}\mathfrak{B}_1 \to {}^{(n)}\mathfrak{B}_2$ satisfying the conditions given above. Again, put $R_{n+1} := {}^{(n+1)}\mathfrak{B}_1 - {}^{(n)}\mathfrak{B}_1$. At first we shall show that $\ell_2(f_n(x)) = \ell_1(x)$ for all $x \notin \uparrow R_{n+1}$. From this it immediately follows that $\ell_2(f_n(x)) \equiv \ell_1(x)$ for all $x \in \uparrow R_{n+1} - R_{n+1}$ since G is strict deterministic. This claim we show by induction on the height of x. If $h(x) = 0$, then x is a leaf and the claim holds because of the assumption that $\mathfrak{T}_1$ and $\mathfrak{T}_2$ have the same associated string. If $h(x) > 0$ then every daughter of x is in $\uparrow R_{n+1}$. By induction hypothesis therefore $\ell_2(f_n(y)) = \ell_1(y)$ for every $y \prec x$. Since G is strict deterministic, the label of x is uniquely fixed by this for $\ell_2(f_n(x)) \equiv \ell_1(x)$, by induction hypothesis. So we now have $\ell_2(f_n(x)) = \ell_1(x)$. This shows the first claim. Now we extend f_n to an isomorphism f_{n+1} from ${}^{(n+1)}\mathfrak{B}_1$ onto ${}^{(n+1)}\mathfrak{B}_2$ and show at the same time that $\ell_2(f_n(x)) \equiv \ell_1(x)$ for every $x \in \uparrow R_{n+1}$. This holds already by inductive hypothesis for all $x \notin R_{n+1}$. So, we only have to show this for $x \in R_{n+1}$. This we do as follows. Let u_0 be the largest node in R_{n+1}. Certainly, u_0 is not the root. So let v be the mother of u_0. f_n is defined on v and we have $\ell_2(f_n(v)) \equiv \ell_1(v)$. By assumption, $\ell_2(f_n(x)) = \ell_1(x)$ for all $x \sqsubset u$. So, we first of all have that there is a daughter x_0 of $f_n(v)$ which is not

in the image of f_n. We choose x_0' minimal with this property. Then we put $f_{n+1}(u_0) := x_0$. Now we have $\ell_2(f_{n+1}(u_0)) \equiv \ell_1(u_0)$. We continue with u_0 in place of v. In this way we obtain a map f_{n+1} from ${}^{(n)}\mathfrak{B}_1 \cup R_{n+1} = {}^{(n+1)}\mathfrak{B}_1$ to ${}^{(n+1)}\mathfrak{B}_2$ with $\ell_2(f_{n+1}(x)) \equiv \ell_1(x)$, if $x \in R_{n+1}$ and $\ell_2(f_{n+1}(x)) = \ell_1(x)$ otherwise. That f_{n+1} is surjective is seen as follows. Suppose that u_k is the leaf of $\mathfrak{B}_1$ in R_{n+1}. Then $x_k = f_{n+1}(u_k)$ is not a leaf in $\mathfrak{B}_2$, and then there exists a x_{k+1} in ${}^{(n+1)}\mathfrak{B}_2 - {}^{(n)}\mathfrak{B}_2$. We have $\ell_2(f_{n+1}(x_k)) \equiv \ell_1(u_k)$. Let x_p be the leaf in L. By Lemma 2.50 $\ell_2(x_p) \not\equiv \ell_2(x_k)$ and therefore also $\ell_2(x_p) \not\equiv \ell_1(u_k)$. However, by assumption x_p is the $n+1$st leaf of $\mathfrak{B}_2$ and likewise u_k is the $n+1$st leaf of $\mathfrak{B}_1$, from which we get $\ell_1(u_k) = \ell_2(x_p)$ in contradiction to what has just been shown. □

Theorem 2.82 *Let G be a strict deterministic grammar. Then $L(G)$ is unambiguous. Further, G is an $LR(0)$–grammar and $L(G)$ is strict deterministic.*

Proof. The strategy of shifting and reducing can be applied as follows: every time we have identified a right hand side of a rule $X \to \vec{\mu}$ then this is a constituent of category X and we can reduce. This shows that we have a 0–strategy. Hence the grammar is an $LR(0)$–grammar. $L(G)$ is certainly unambiguous. Furthermore, $L(G)$ is deterministic, by Theorem 2.76. Finally, we have to show that $L(G)$ is prefix free for then by Theorem 2.45 it follows that $L(G)$ is strict deterministic. Now let $\vec{x}\vec{y} \in L(G)$. If also $\vec{x} \in L(G)$, then by Proposition 2.81 we must have $\vec{y} = \varepsilon$. □

At first sight it appears that Lemma 2.78 also holds for $k = 0$. The construction can be extended to this case without trouble. Indeed, in this case we get something of an $LR(0)$–grammar; however, it is to be noted that a strategy for $G^{>}$ does not only depend on the next symbol. Additionally, it depends on the fact whether or not the string that is yet to be read is empty. The strategy is therefore not entirely independent of the right context even though the dependency is greatly reduced. That $LR(0)$–languages are indeed more special than $LR(1)$–languages is the content of the next theorem.

Theorem 2.83 (Geller & Harrison) *Let L be a deterministic CFL. Then the following are equivalent.*

① *L is an $LR(0)$–language.*

② *If $\vec{x} \in L$, $\vec{x}\vec{v} \in L$ and $\vec{y} \in L$ then also $\vec{y}\vec{v} \in L$.*

③ *There are strict deterministic languages U and V such that $L = U \cdot V^*$.*

Proof. Assume ①. Then there is an $LR(0)$–grammar G for L. Hence, if $X \to \vec{\alpha}$ is a rule and if $\vec{\eta}\,\vec{\alpha}\,\vec{y}$ is G–derivable then also $\vec{\eta}\,X\,\vec{y}$ is G–derivable. Using induction, this can also be shown of all pairs X, $\vec{\alpha}$ for which $X \vdash_G \vec{\alpha}$. Now let $\vec{x} \in L$ and $\vec{x}\vec{v} \in L$. Then $\mathtt{S} \vdash_G \vec{x}$, and so by the previous $\vdash_G \mathtt{S}\,\vec{v}$. Therefore, since $\mathtt{S} \vdash_G \vec{y}$ we have $\vdash_G \vec{y}\vec{v}$. Hence ② obtains. Assume now ②. Let U be the set of all $\vec{x} \in L$ such that $\vec{y} \notin L$ for every proper prefix $\vec{y}$ of $\vec{x}$. Let V be the set of all $\vec{v}$ such that $\vec{x}\vec{v} \in L$ for some $\vec{x} \in U$ but $\vec{x}\vec{w} \notin L$ for every $\vec{x} \in U$ and every proper prefix $\vec{w}$ of $\vec{v}$. Now, V is the set of all $\vec{y} \in V^* - \{\varepsilon\}$ for which no proper prefix is in $V^* - \{\varepsilon\}$. We show that $U \cdot V^* = L$. To this end let us prove first that $L \subseteq U \cdot V^*$. Let $\vec{u} \in L$. We distinguish two cases. (a) No proper prefix of $\vec{u}$ is in L. Then $\vec{u} \in U$, by definition of U. (b) There is a proper prefix $\vec{x}$ of $\vec{u}$ which is in L. We choose $\vec{x}$ minimally. Then $\vec{x} \in U$. Let $\vec{u} = \vec{x}\vec{v}$. Now two subcases arise. (A) For no proper prefix $\vec{w}_0$ of $\vec{v}$ we have $\vec{x}\vec{w}_0 \in L$. Then $\vec{v} \in V$, and we are done. (B) There is a proper prefix $\vec{w}_0$ of $\vec{v}$ with $\vec{x}\vec{w}_0 \in L$. Let $\vec{v} = \vec{w}_0\,\vec{v}_1$. Then, by ②, we have $\vec{x}\vec{v}_1 \in L$. (In ②, put $\vec{x}\vec{w}_0$ in place of $\vec{x}$ and in place of $\vec{y}$ put $\vec{x}$ and for $\vec{w}$ put $\vec{v}_1$.) $\vec{x}\vec{v}_1$ has smaller length than $\vec{x}\vec{v}$. Continue with $\vec{x}\vec{v}_1$ in the same way. At the end we get a partition of $\vec{v} = \vec{w}_0\,\vec{w}_1 \cdots \vec{w}_{n-1}$ such that $\vec{w}_i \in V$ for every $i < n$. Hence $L \subseteq U \cdot V^*$. We now show $U \cdot V^* \subseteq L$. Let $\vec{u} = \vec{x}^\frown \prod_{i<n} \vec{w}_i$. If $n = 0$, then $\vec{u} = \vec{x}$ and by definition of U we have $\vec{u} \in L$. Now let $n > 0$. With ② we can show that $\vec{x}^\frown \prod_{i<n-1} \vec{w}_i \in L$. This shows that $\vec{u} \in L$. Finally, we have to show that U and V are deterministic. This follows for U from Theorem 2.46. Now let $\vec{x}, \vec{y} \in U$. Then by ② $P := \{\vec{v} : \vec{x}\vec{v} \in L\} = \{\vec{v} : \vec{y}\vec{v} \in L\}$. The reader may convince himself that P is deterministic. Now let V be the set of all $\vec{v}$ for which there is no prefix in $P - \{\varepsilon\}$. Then $P = V^*$ and because of Theorem 2.46 V is strict deterministic. This shows ③. Finally, assume ③. We have to show that L is an $LR(0)$–language. To this end, let $G_1 = \langle \mathtt{S}, N_1, A, R_1 \rangle$ be a strict deterministic grammar which generates U and $G_2 = \langle \mathtt{S}_2, N_2, A, R_2 \rangle$ a strict deterministic grammar which generates V. Then let $G_3 := \langle \mathtt{S}_3, N_1 \cup N_2 \cup \{\mathtt{S}_3, \mathtt{S}_4\}, A, R_3 \rangle$ be defined as follows.

$$R_3 := R_1 \cup R_2 \cup \{\mathtt{S}_3 \to \mathtt{S}_1, \mathtt{S}_3 \to \mathtt{S}_1\,\mathtt{S}_4, \mathtt{S}_4 \to \mathtt{S}_2, \mathtt{S}_4 \to \mathtt{S}_2\,\mathtt{S}_4\} \tag{2.87}$$

It is not hard to show that G_3 is an $LR(0)$–grammar and that $L(G_3) = L$. □

The decomposition in ③ is unique, if we exclude the possibility that $V = \varnothing$ and if we require that $U = \{\varepsilon\}$ shall be the case only if $V = \{\varepsilon\}$. In this way we take care of the cases $L = \varnothing$ and $L = U$. The case $U = V$ may arise. Then $L = U^+$. The semi Dyck languages are of this kind.

Now we proceed to the proof of Theorem 2.79. Let L be deterministic. Then put $M := L \cdot \{\$\}$, where \$ is a new symbol. M is certainly deterministic; and it is prefix free and so strict deterministic. It follows that M is an $LR(0)$–language. Therefore there exists a strict deterministic grammar G which generates M. From the next theorem we now conclude that L is an $LR(1)$–language.

Lemma 2.84 *Let G be an $LR(0)$–grammar of the form $G = \langle \mathtt{S}, N \cup \{\$\}, A, R\rangle$ with $R \subseteq N \times ((N \cup A)^* \cup (N \cup A)^* \cdot \$)$ and $L(G) \subseteq A^*\$$, and assume that there is no derivation $\mathtt{S} \Rightarrow_R \mathtt{S}\$$ in G. Then let $H := \langle \mathtt{S}, N, A, R'\rangle$, where*

$$R' := \{A \to \vec{\alpha} : A \to \vec{\alpha} \in R, \vec{\alpha} \in (N \cup A)^*\} \cup \{A \to \vec{\alpha} : A \to \vec{\alpha}\,\$ \in R\} \tag{2.88}$$

Then H is an $LR(1)$–grammar and $L(H) \cdot \$ = L(G)$.

For a proof consider the following. We do not have $\mathtt{S} \Rightarrow_L^+ \mathtt{S}$ in H. Further: if $\mathtt{S} \Rightarrow_L^+ \vec{\alpha}$ in H then there exists a D such that $\mathtt{S} \Rightarrow_L^+ \vec{\alpha} D$ in G, and if $\mathtt{S} \Rightarrow_L^+ \vec{\beta}$ in G then we have $\vec{\beta} = \vec{\alpha} D$ and $\mathtt{S} \Rightarrow_L^+ \vec{\alpha}$ in H. From this we can immediately conclude that H is an $LR(1)$–grammar.

Finally, let us return to the calculus of shifting and reducing. We generalize this strategy as follows. For every symbol α of our grammar we add a symbol $\underline{\alpha}$. This symbol is a formal inverse of α; it signals that at its place we look for an α but haven't identified it yet. This means that we admit the following transitions.

$$\frac{\vec{\eta}\,\underline{\alpha}\alpha \vdash \vec{x}}{\vec{\eta} \quad \vdash \vec{x}} \tag{2.89}$$

We call this rule **cancellation**. We write for strings $\vec{\alpha}$ also $\underline{\vec{\alpha}}$. This denotes the formal inverse of the entire string. If $\vec{\alpha} = \prod_{i<n} \alpha_i$ then $\underline{\alpha} = \prod_{i<n} \underline{\alpha_{n-i}}$. Notice that the order is reversed. For example $\underline{\mathtt{AB}} = \underline{\mathtt{B}}^\frown\underline{\mathtt{A}}$. These new strings allow to perform reductions on the left hand side even when only part of the right hand side of a production has been identified. The most general rule is this one.

$$\frac{\vec{\eta}\,\underline{X}\vec{\alpha} \vdash \vec{x}}{\vec{\eta}\,\underline{\vec{\tau}} \quad \vdash \vec{x}} \tag{2.90}$$

This rule is called the **LC–rule**. Here $X \to \vec{\alpha}\vec{\tau}$ must be a G–rule. This means intuitively speaking that $vec\alpha$ is an X if followed by $\vec{\tau}$. Since $\vec{\tau}$ is not yet there

we have to write $\underline{\vec{\tau}}$. The **LC–calculus** consists of the rules shift, reduce and LC. Now the following holds.

Theorem 2.85 *Let G be a grammar. $\vec{\alpha} \vdash_G \vec{x}$ holds iff there is a derivation of $\varepsilon \vdash \varepsilon$ from $\underline{\vec{\alpha}} \vdash \vec{x}$ in the **LC**–calculus.*

A special case is $\vec{\alpha} = \varepsilon$. Here no part of the production has been identified, and one simply guesses a rule. If in place of the usual rules only this rule is taken, we get a strategy known as **top–down strategy**. In it, one may shift, reduce and guess a rule. A grammar is called an $LL(k)$–grammar if it has a deterministic recognition algorithm using the top–down–strategy in which the next step depends on the first k symbols of $\vec{x}$. The case $k = 0$ is of little use (see the exercises).

This method is however too flexible to be really useful. However, the following is an interesting strategy. The right hand side of a production is divided into two parts, which are separated by a dot.

$$\begin{array}{lcl} \mathtt{S} & \to & \mathtt{A.SB} \mid \mathtt{c.} \\ \mathtt{A} & \to & \mathtt{a.} \\ \mathtt{B} & \to & \mathtt{b.} \end{array} \tag{2.91}$$

This dot fixes the part of the rule that must have been read when the corresponding LC–rule is triggered. A strategy of this form is called **generalized left corner strategy**. If the dot is at the right edge we get the bottom–up strategy, if it is at the left edge we get the top–down strategy.

Exercise 66. Let R be a set of context free rules, $\mathtt{S}$ a symbol, N and A finite sets, and $G := \langle \mathtt{S}, N, A, R \rangle$. Show that if $\Rightarrow_R^* \varepsilon$ and G is transparent then G is a CFG. *Remark.* Transparency can obviously be generalized to any grammar that uses context free rules.

Exercise 67. Show Theorem 2.76.

Exercise 68. Prove Lemma 2.60. Show in addition: *If $\vec{x}$ is a term then the set $P(\vec{x}) := \{\gamma(\vec{y}) : \vec{y}$ is a prefix of $\vec{x}\}$ is convex.*

Exercise 69. Show the following: *If L is deterministic then also $L/\{\vec{x}\}$ as well as $\{\vec{x}\}\backslash L$ are deterministic.* (See Section 1.2 for notation.)

Exercise 70. Show that a grammar is an $LL(0)$–grammar if it generates exactly one tree.

Exercise 71. Give an example of an NTS–language which is not an $LR(0)$–language.

Table 5. The Generalized LC–Strategy

$\underline{\mathtt{S}}$	$\vdash$ aacbb
$\underline{\mathtt{S}}$a	$\vdash$ acbb
$\underline{\mathtt{S}}$A	$\vdash$ acbb
$\underline{\mathtt{B}}\underline{\mathtt{S}}$	$\vdash$ acbb
$\underline{\mathtt{B}}\underline{\mathtt{S}}$a	$\vdash$ cbb
$\underline{\mathtt{B}}\underline{\mathtt{S}}$A	$\vdash$ cbb
$\underline{\mathtt{B}}\underline{\mathtt{B}}\underline{\mathtt{S}}$	$\vdash$ cbb
$\underline{\mathtt{B}}\underline{\mathtt{B}}\underline{\mathtt{S}}$c	$\vdash$ bb
$\underline{\mathtt{B}}\underline{\mathtt{B}}\underline{\mathtt{S}}$S	$\vdash$ bb
$\underline{\mathtt{B}}\underline{\mathtt{B}}$	$\vdash$ bb
$\underline{\mathtt{B}}\underline{\mathtt{B}}$b	$\vdash$ b
$\underline{\mathtt{B}}\underline{\mathtt{B}}$B	$\vdash$ b
$\underline{\mathtt{B}}$	$\vdash$ b
$\underline{\mathtt{B}}$b	$\vdash \varepsilon$
$\underline{\mathtt{B}}$B	$\vdash \varepsilon$
ε	$\vdash \varepsilon$

5. Semilinear Languages

In this section we shall study semilinear languages. The notion of semilinearity is important in itself as it is widely believed that natural languages are semilinear. Whether or not this is case, is still open (see Section 2.7). The issue of semilinearity is important, because many grammar formalisms proposed in the past only generate semilinear languages (or else are generally so powerful that they generate every recursively enumerable set). Even though semilinearity in natural languages is the rule rather than the exception, the counterexamples show that the grammar formalisms do not account for natural language in a satisfactory way.

In this chapter we shall prove a theorem by Ginsburg and Spanier which says that the semilinear subsets of ω^n are exactly the sets definable in Presburger Arithmetic. This theorem has numerous consequences, in linguistics as well as in mathematics. The proof given here differs substantially from the original one.

Definition 2.86 *A* ***commutative monoid*** *or* ***commutative semigroup with unit***

is a structure $\langle H, 0, +\rangle$ *in which the following holds for every* $x, y, z \in H$.

$$\begin{aligned} x+0 &= x \\ x+(y+z) &= (x+y)+z \\ x+y &= y+x \end{aligned} \tag{2.92}$$

Notice that because of associativity we may dispense with brackets. Alternatively, any term can be arbitrarily bracketed without affecting its value. We define the notation $\mu \cdot x$ as follows: $0 \cdot x := 0$ and $(\mu+1) \cdot x := \mu \cdot x + x$. (Later on we shall drop $\cdot$.) Then $\mu \cdot x_0 + \nu \cdot x_0 = (\mu+\nu) \cdot x_0$, and $\mu \cdot (\nu \cdot x_0) = (\mu\nu) \cdot x_0$, simply by definition. Furthermore, $\mu \cdot (x+y) = (\mu \cdot x) + (\mu \cdot y)$, by induction on μ. This can be generalized.

Lemma 2.87 *In a commutative semigroup, the following holds.*

$$\mu \cdot (\sum_{i<m} \nu_i \cdot x_i) = \sum_{i<m} (\mu\nu_i) \cdot x_i \tag{2.93}$$

$$\sum_{i<m} \mu_i \cdot x_i + \sum_{i<m} \nu_i \cdot x_i = \sum_{i<m} (\mu_i + \nu_i) \cdot x_i \tag{2.94}$$

Proof. Induction on m. The case $m = 1$ has been dealt with. Now:

$$\begin{aligned} \mu \cdot (\sum_{i<m+1} \nu_i \cdot x_i) &= \mu \cdot (\sum_{i<m} \nu_i \cdot x_i + \nu_m \cdot x_m) \\ &= \mu \cdot (\sum_{i<m} \nu_i \cdot x_i) + \mu \cdot (\nu_m \cdot x_m) \\ &= \sum_{i<m} (\mu\nu_i) \cdot x_i + (\mu\nu_m) \cdot x_m \\ &= \sum_{i<m+1} (\mu\nu_i) \cdot x_i \end{aligned} \tag{2.95}$$

Also

$$\begin{aligned} &\sum_{i<m+1} \mu_i \cdot x_i + \sum_{i<m+1} \nu_i \cdot x_i \\ =&(\sum_{i<m} \mu_i \cdot x_i + \mu_m \cdot x_m) + (\sum_{i<m} \nu_i \cdot x_i + \nu_m \cdot x_m) \\ =&(\sum_{i<m} \mu_i \cdot x_i + \sum_{i<m} \nu_i \cdot x_i) + (\mu_m + \nu_m) \cdot x_m \\ =&\sum_{i<m} (\mu_i + \nu_i) x_i + (\mu_m + \nu_m) \cdot x_m \\ =&\sum_{i<m+1} (\mu_i + \nu_i) x_i \end{aligned} \tag{2.96}$$

This finishes the proof. □

We shall denote by $M(A)$ set underlying the commutative monoid freely generated by A. By construction, $\mathfrak{M}(A) := \langle M(A), 0, +\rangle$ is a commutative semigroup with unit. What is more, $\mathfrak{M}(A)$ is freely generated by A as a commutative semigroup. We now look at the set ω^n of all n–long sequences of natural numbers, endowed with the operation $+$ defined by

$$\langle x_i : i < n\rangle + \langle y_i : i < n\rangle := \langle x_i + y_i : i < n\rangle \tag{2.97}$$

This also forms a commutative semigroup with unit. Here the unit is the sequence $\vec{0}$ consisting of n 0s. We denote this semigroup by Ω^n. For the following theorem we also need the so–called **Kronecker symbol**.

$$\delta^i_j := \begin{cases} 1 & \text{if i = j,} \\ 0 & \text{otherwise.} \end{cases} \tag{2.98}$$

Theorem 2.88 *Let $A = \{a_i : i < n\}$. Let h be the map which assigns to each element a_i the sequence $\vec{e}_i = \langle \delta^i_j : j < n\rangle$. Then the homomorphism which extends h is an isomorphism from $\mathfrak{M}(A)$ onto Ω^n.*

Proof. Let θ be the smallest congruence relation on $\mathrm{Tm}_\Omega(A)$ (with $\Omega\colon 0 \mapsto 0, + \mapsto 2$) which satisfies (2.92). It follows from Lemma 2.87 by induction on the level of the term t that for $t \in \mathrm{Tm}_\Omega(A)$ there is a $u\ \theta\ t$ of the form

$$u = \sum_{i<n} k_i \cdot a_i \tag{2.99}$$

If (2.99) obtains, put $q(t) := \langle k_i : i < n\rangle$. Now, it is immediately seen that $\theta = \ker q$, whence $\mathrm{Tm}_\Omega(A)/\theta \cong \Omega^n$. On the other hand, $\mathrm{Tm}_\Omega(A)/\theta \cong \mathfrak{M}(A)$, since it is easily shown that the first is also freely generated by A. Namely, suppose that $v : a_i \mapsto n_i$ is a map from A into $\mathfrak{N}$. Let $\overline{v} : \mathrm{Tm}_\Omega(A) \to N$ be the extension of v. Then, since $\mathfrak{N}$ is a monoid, $\theta \subseteq \ker \overline{v}$, so that we can define a map $q : \mathfrak{Tm}_\Omega(A) \to \mathfrak{N}$ such that $\overline{v} = q \circ h_\theta$. □

This theorem tells us that free commutative semigroups can be thought of as vectors of numbers. A general element of $M(A)$ can be written down as $\sum_{i<n} k_i \cdot a_i$ where $k_i \in \omega$.

Now we shall define the map $\mu : A^* \to M(A)$ by

$$\begin{aligned} \mu(\varepsilon) &= 0 \\ \mu(a_i) &= a_i \\ \mu(\vec{x}^\frown \vec{y}) &= \mu(\vec{x}) + \mu(\vec{y}) \end{aligned} \tag{2.100}$$

This map is a homomorphism of monoids and also surjective. It is not injective, except in the case where A consists of one element only. The map μ is called the **Parikh map**. We have

$$\mu\left(\prod_{i<k}\vec{x}_i\right) = \sum_{i<k}\mu(\vec{x}_i) \tag{2.101}$$

Definition 2.89 *Two languages $L, M \subseteq A^*$ are called **letter equivalent** if we have $\mu[L] = \mu[M]$.*

Definition 2.90 *Elements of $M(A)$ will also be denoted using vector arrows. Moreover, if $\vec{x} \in \omega^n$ we write $\vec{x}(i)$ for the ith component of $\vec{x}$. A set $U \subseteq M(A)$ is called **linear** if for some $\alpha \in \omega$ and some $\vec{u}, \vec{v}_i \in M(A)$*

$$U = \{\vec{u} + \sum_{i<\alpha} k_i \cdot \vec{v}_i : k_0, \ldots, k_{\alpha-1} \in \omega\} \tag{2.102}$$

*The $\vec{v}_i$ are called **cyclic vectors of** U. The smallest α for which U has such a representation is called the **dimension of** U. U is said to be **semilinear** if U is the finite union of linear sets. A language $L \subseteq A^*$ is called **semilinear** if $\mu[L]$ is semilinear.*

We can denote semilinear sets rather compactly as follows. If U and V are subsets of $M(A)$ then write $U + V := \{\vec{x} + \vec{y} : \vec{x} \in U, \vec{y} \in V\}$. Further, let $\vec{x} + U := \{\vec{x} + \vec{y} : \vec{y} \in U\}$. So, vectors are treated as singleton sets. Also, we write $nU := \{n\vec{x} : n \in \omega\}$. Finally, we denote by ωU the union of all nU, $n \in \omega$. With these abbreviations we write the set U from Definition 2.90 as follows.

$$U = \vec{u} + \omega\vec{v}_0 + \omega\vec{v}_1 + \cdots + \omega\vec{v}_{\alpha-1} \tag{2.103}$$

This in turn we abbreviate by

$$U = \vec{u} + \sum_{i<\alpha} \omega\vec{v}_i \tag{2.104}$$

Finally, for $V = \{\vec{v}_i : i < \alpha\}$

$$\Sigma(U;V) := U + \sum_{i<\alpha} \omega\vec{v}_i \tag{2.105}$$

Lemma 2.91 *The following holds.*

① $\Sigma(U;V) \cup \Sigma(U';V) = \Sigma(U \cup U';V)$.

② $\Sigma(U;V) + \Sigma(U';V') = \Sigma(U + U';V \cup V')$.

③ $\omega\Sigma(U;V) = \Sigma(\{\vec{0}\};U \cup V)$.

Theorem 2.92 (Parikh) *A language is semilinear iff it is letter equivalent to a regular language.*

Proof. ($\Rightarrow$) It is enough to show this for linear languages. Suppose that $\pi[L] = \Sigma(\{\vec{u}\};V)$, $V = \{\vec{v}_i : i < n\}$. Pick a string $\vec{x}$ and $\vec{y}_i$, $i < n$, such that $\pi(\vec{x}) = \vec{u}$ and $\pi(\vec{y}_i) = \vec{v}_i$ for all $i < n$. Put

$$M := \vec{x}^\frown(\bigcup_{i<n} \vec{y}_i)^* \tag{2.106}$$

Clearly, M is regular and letter equivalent to L. ($\Leftarrow$) By induction on the length of the regular term R we shall show that $\mu[L(R)]$ is semilinear. This is clear for $R = a_i$ or $R = \varepsilon$. It is also clear for $R = S_1 \cup S_2$. Now let $R = S_1 \cdot S_2$. Using the equations $(S \cup T) \cdot U = S \cdot U \cup T \cdot U$ and $U \cdot (S \cup T) = U \cdot S \cup U \cdot T$, we can assume that S_1 and S_2 are linear. Then by definition $\mu[L(S_1)] = \Sigma(\{\vec{u}\};C_1)$ and $\mu[L(S_2)] = \Sigma(\{\vec{v}\};C_2)$, for certain $\vec{u}$, $\vec{v}$, and sets C_1 and C_2. Then, using Lemma 2.91, we get

$$\mu[L(R)] = \Sigma(\{\vec{u}\};C_1) + \Sigma(\{\vec{v}\};C_2) = \Sigma(\{\vec{u} + \vec{v}\};C_1 \cup C_2) \tag{2.107}$$

Now, finally, $R = S^*$. If $S = T \cup U$, then $R = (T^* \cdot U^*)^*$, so that we may again assume that S is linear, say, $S = \Sigma(\{\vec{u}\};C)$ for some $\vec{u}$ and C. By Lemma 2.91

$$\mu[L(R)] = \omega\Sigma(\{\vec{u}\};C) = \Sigma(\{\vec{0}\};\{\vec{u}\} \cup C) \tag{2.108}$$

Hence R too is linear. This ends the proof. □

We draw a useful conclusion from the definitions.

Theorem 2.93 *Let A be a (possibly infinite) set. The set of semilinear languages over A form an AFL with the exception that the intersection of a semilinear language with a regular language need not be semilinear.*

Proof. Closure under union, star and concatenation are immediate. We have to show that semilinear languages are closed under homomorphisms and inverse homomorphisms. The latter is again trivial. Now let $v\colon A \to A^*$ be a

homomorphism. v induces a map $\kappa_v\colon \mathfrak{M}(A) \to \mathfrak{M}(A)$. The image under κ_v of a semilinear set is semilinear. For given a string $\vec{x} \in A^*$ we have $\mu(\overline{v}(\vec{x})) = \kappa_v(\mu(\vec{x}))$, as is easily checked by induction on the length of $\vec{x}$. Let M be linear, say $M = \vec{u} + \sum_{i<k} \omega \cdot \vec{v}_i$. Then

$$\kappa_v[M] = \kappa_v(\vec{u}) + \sum_{i<k} \omega \kappa_v(\vec{v}_i) \tag{2.109}$$

From this the claim follows. Hence we have $\mu[\overline{v}[L]] = \kappa_v[\mu[L]]$. The right hand side is semilinear as we have seen. Finally, take the language $L := \{\mathtt{a}^{2^i}\mathtt{b}^{2^i} : i \in \omega\} \cup \{\mathtt{b}^j\mathtt{a}^j : j \in \omega\}$. L is semilinear. $L \cap \mathtt{a}^*\mathtt{b}^* = \{\mathtt{a}^{2^i}\mathtt{b}^{2^i} : i \in \omega\}$ is not semilinear, however. □

Likewise, a subset of $\mathbb{Z}^n$ ($\mathbb{Q}^n$) is called **linear** if it has the form

$$\vec{v}_0 + \mathbb{Z}\vec{v}_1 + \mathbb{Z}\vec{v}_2 + \cdots + \mathbb{Z}\vec{v}_m \tag{2.110}$$

for subsets of $\mathbb{Z}^n$ as well as

$$\vec{v}_0 + \mathbb{Q}\vec{v}_1 + \mathbb{Q}\vec{v}_2 + \cdots + \mathbb{Q}\vec{v}_m \tag{2.111}$$

for subsets of $\mathbb{Q}^n$. The linear subsets of $\mathbb{Q}^n$ are nothing but the affine subspaces. A subset of ω^n ($\mathbb{Z}^n$, $\mathbb{Q}^n$) is called **semilinear** if it is the finite union of linear sets.

Presburger Arithmetic is defined as follows. The basic symbols are $\mathtt{0}$, $\mathtt{1}$, $\mathtt{+}$, $\mathtt{<}$ and $\equiv_m$, $m \in \omega - \{0,1\}$. Then Presburger Arithmetic is the set of first order sentences which are valid in $\underline{\mathbb{Z}} := \langle \mathbb{Z}, 0, 1, +, <, \langle \equiv_m \colon 1 < m \in \omega \rangle \rangle$, where $a \equiv_m b$ iff $a - b$ is divisible by m (for FOL see Sections 3.8 and 4.4).

Negation can be eliminated. Notice namely that $\neg(\mathtt{x}_0{=}\mathtt{x}_1)$ is equivalent to $(\mathtt{x}_0{<}\mathtt{x}_1)\vee(\mathtt{x}_1{<}\mathtt{x}_0)$, $\neg(\mathtt{x}_0{<}\mathtt{x}_1)$ to $(\mathtt{x}_0{=}\mathtt{x}_1)\vee(\mathtt{x}_1{<}\mathtt{x}_0)$ and $\neg(\mathtt{x}_0{\equiv_m}\mathtt{x}_1)$ is equivalent to $\bigvee_{0<n<m}\mathtt{x}_0{\equiv_m}(\mathtt{x}_1{+}\underline{n})$. Here, $\underline{n}$ is defined by $\underline{0} := \mathtt{0}$, $\underline{n+1} := (\underline{n}{+}\mathtt{1})$. We shall use $\mathtt{x}_0{\leq}\mathtt{x}_1$ for $(\mathtt{x}_0{<}\mathtt{x}_1)\vee(\mathtt{x}_0{=}\mathtt{x}_1)$. Moreover, multiplication by a given natural number also is definable: put $0t := \overline{0}$, and $(n+1)t := (nt{+}t)$. Every term in the variables $\mathtt{x}_i$, $i < n$, is equivalent to a term $\mathtt{x}_0{+}\sum_{i<n} a_i\mathtt{x}_i$, where $b, a_i \in \omega$, $i < n$. A subset S of $\mathbb{Z}^n$ is **definable** if there is a formula $\varphi(\mathtt{x}_0, \mathtt{x}_1, \ldots, \mathtt{x}_{n-1})$ such that

$$S = \{\langle k_i : i < n \rangle \in \mathbb{Z}^n : \underline{\mathbb{Z}} \vDash \varphi[k_0, k_1, \ldots, k_{n-1}]\} \tag{2.112}$$

The definable subsets of $\mathbb{Z}^n$ are closed under union, intersection and complement and permutation of the coordinates. Moreover, if $S \subseteq \mathbb{Z}^{n+1}$ is definable,

so is its projection

$$\pi_n[S] := \{\langle k_i : i < n\rangle : \text{ there is } k_n \in \mathbb{Z} \text{ such that } \langle k_i : i < n+1\rangle \in S\} \tag{2.113}$$

The same holds for definable subsets of ω^n, which are simply those definable subsets of $\mathbb{Z}^n$ that are included in ω^n. Clearly, if $S \subseteq \mathbb{Z}^n$ is definable, so is $S \cap \omega^n$.

Lemma 2.94 *Suppose that $a + \sum_{i<n} p_i x_i = b + \sum_{i<n} q_i x_i$ is a linear equation with rational numbers a, b, p_i and q_i ($i < n$). Then there is an equation*

$$g + \sum_{i<n} u_i x_i = h + \sum_{i<n} v_i x_i \tag{2.114}$$

with the same solutions, but with positive integer coefficients such that $g \cdot h = 0$ and for every $i < n$: $v_i u_i = 0$.

Proof. First, multiply with the least common denominator to transform the equation into an equation with integer coefficients. Next, add $-p_i x_i$ to both sides if $p_i < 0$, unless $q_i < p_i < 0$, in which case we add $-q_i x_i$. Now all coefficients are positive. Next, for every $i < n$, substract $q_i x_i$ from both sides if $p_i > q_i$ and $p_i x_i$ otherwise. These transformations preserve the set of solutions. □

Call an equation **reduced** if it has the form

$$g + \sum_{i<m} k_i x_i = \sum_{m \le i < n} k_i x_i \tag{2.115}$$

with positive integer coefficients g and k_i, $i < n$. Likewise for an inequation. Evidently, modulo renaming of variables we can transform every rational equation into reduced form.

Lemma 2.95 *The set of solutions of a reduced equation is semilinear.*

Proof. Let μ be the least common multiple of the k_i. Consider a vector of the form $\vec{c}_{i,j} = (\mu/k_i)\vec{e}_i + (\mu/k_j)\vec{e}_j$, where $i < m$ and $m \le j < n$. Then if $\vec{v}$ is a solution, so is $\vec{v} + \vec{c}_{i,j}$ and conversely. Put $C := \{\vec{c}_{i,j} : i < m \le j < n\}$ and

$$P := \left\{\vec{u} : g + \sum_{i<m} k_i \vec{u}(i) = \sum_{m \le i < n} k_i \vec{u}(i), \vec{u}(i) < \mu/k_i\right\} \tag{2.116}$$

Both P and C are finite. Moreover, the set of solutions is exactly $\Sigma(P;C)$. □

Lemma 2.96 *The set of solutions of a reduced inequation is semilinear.*

Proof. Assume that the inequation has the form

$$g + \sum_{i<m} k_i x_i \leq \sum_{m \leq i < n} k_i x_i \tag{2.117}$$

Define C and P as before. Let $E := \{\vec{e}_i : m \leq i < n\}$. Then the set of solutions is $\Sigma(P; C \cup E)$. If the inequation has the form

$$g + \sum_{i<m} k_i x_i \geq \sum_{m \leq i < n} k_i x_i \tag{2.118}$$

The set of solutions is $\Sigma(P; C \cup F)$, where $F := \{\vec{e}_i : i < m\}$. □

Lemma 2.97 *Let $M \subseteq \mathbb{Q}^n$ be an affine subspace. Then $M \cap \mathbb{Z}^n$ is a semilinear subset of $\mathbb{Z}^n$.*

Proof. Let $\vec{v}_i$, $i < m+1$, be vectors such that

$$M = \vec{v}_0 + \mathbb{Q}\vec{v}_1 + \mathbb{Q}\vec{v}_2 + \cdots + \mathbb{Q}\vec{v}_{m-1} \tag{2.119}$$

We can assume that the $\vec{v}_i$ are linearly independent. Clearly, since $\mathbb{Q}\vec{w} = \mathbb{Q}(\lambda\vec{w})$ for any nonzero rational number λ, we can assume that $\vec{v}_i \in \mathbb{Z}^n$, $i < m$. Now, put

$$V := \{\vec{v}_0 + \sum_{0<i<m} \lambda_i \vec{v}_i : 0 \leq \lambda_i < 1\} \tag{2.120}$$

$V \cap \mathbb{Z}^n$ is finite. Moreover, if $\vec{v}_0 + \sum_{0<i<m} \kappa_i \vec{v}_i \in \mathbb{Z}^n$ then $\vec{v}_0 + \sum_{0<i<m} \kappa_i' \vec{v}_i \in \mathbb{Z}^n$ if $\kappa_i - \kappa_i' \in \mathbb{Z}$. Hence,

$$M = \bigcup_{\vec{w} \in V} (\vec{w} + \mathbb{Z}\vec{v}_1 + \cdots + \mathbb{Z}\vec{v}_m) \tag{2.121}$$

This is a semilinear set. □

Lemma 2.98 *Let $M \subseteq \mathbb{Z}^n$ be a semilinear subset of $\mathbb{Z}^n$. Then $M \cap \omega^n$ is semilinear.*

Proof. It suffices to show this for linear subsets. Let $\vec{v}_i$, $i < m+1$, be vectors such that

$$M = \vec{v}_0 + \mathbb{Z}\vec{v}_1 + \mathbb{Z}\vec{v}_2 + \cdots + \mathbb{Z}\vec{v}_{m-1} \tag{2.122}$$

Put $\vec{w}_i := -\vec{v}_i$, $0 < i < m$. Then

$$M = \vec{v}_0 + \omega\vec{v}_1 + \omega\vec{v}_2 + \cdots + \omega\vec{v}_{m-1} + \omega\vec{w}_1 + \cdots + \omega\vec{w}_{m-1} \tag{2.123}$$

Thus, we may without loss of generality assume that

$$M = \vec{v}_0 + \omega\vec{v}_1 + \omega\vec{v}_2 + \cdots + \omega\vec{v}_{m-1} \tag{2.124}$$

Notice, however, that these vectors are not necessarily in ω^n. For i starting at 1 until n we do the following.

Let $x^i_j := \vec{v}_j(i)$. Assume that for $0 < j < p$ we have $x^i_j \geq 0$, and that for $p \leq j < m$ we have $x^i_j > 0$. (A renaming of the variables can achieve this.) We introduce new cyclic vectors $\vec{c}_{j,k}$ for $0 < j < p$ and $p \leq k < m$. Let μ the least common multiple of the $|x^i_s|$, for all $0 < s < m$ where $x^i_s \neq 0$.

$$\vec{c}_{i,j} := (\mu/x^i_j)\vec{v}_j + (\mu/x^i_k)\vec{v}_k \tag{2.125}$$

Notice that the s–coordinates of these vectors are positive for $s < i$, since this is a positive sum of positive numbers. The ith coordinate of these vectors is 0. Suppose that the ith coordinate of

$$\vec{w} = \vec{v}_0 + \sum_{0<j<m} \lambda_j \vec{v}_j \tag{2.126}$$

is ≥ 0, where $\lambda_j \in \omega$ for all $0 < j < m$. Suppose further that for some $k \geq p$ we have $\lambda_k \geq v^i_0 + m(\mu/|x^i_k|)$. Then there must be a $j < p$ such that $\lambda_j \geq (\mu/x^i_j)$. Then put $\lambda'_r := \lambda_r$ for $r \neq j,k$, $\lambda'_j := \lambda_j - (\mu/x^i_j)$ and $\lambda'_k := \lambda_k + (\mu/x^i_k)$. Then

$$\vec{w} = \vec{c}_{j,k} + \sum_{0<j<m} \lambda'_j \vec{v}_j \tag{2.127}$$

Moreover, $\lambda'_j \leq \lambda_j$ for all $j < p$, and $\lambda'_k < \lambda_k$. Thus, by adding these cyclic vectors we can see to it that the coefficients of the $\vec{v}_k$ for $p \leq k < m$ are bounded. Now define P to be the set of all $\vec{w}$ which have a decomposition

$$\vec{w} = \vec{v}_0 + \sum_{0<j<m} \lambda_j \vec{v}_j \in \omega^n \tag{2.128}$$

where $\lambda_j < v^j_0 + m|\mu/x^i_j|$ for all $0 < j < m$. Then

$$M \cap \omega^n = \bigcup_{\vec{u}\in P} \left(\vec{u} + \sum_{0<j<p} \lambda_j \vec{v}_j + \sum_{0<j<p\leq k<m} \kappa_{j,k}\vec{c}_{j,k} \right) \tag{2.129}$$

with all λ_j, $\kappa_{j,k} \geq 0$. Now we have achieved that all jth coordinates of vectors are positive. $\square$

The following is now immediate.

Lemma 2.99 *Let $M \subseteq \mathbb{Q}^n$ be an affine subspace. Then $M \cap \omega^n$ is a semilinear subset of ω^n.*

Lemma 2.100 *The intersection of semilinear sets is again semilinear.*

Proof. It is enough to show the claim for linear sets. So, let S_0 and S_1 be linear. Then there are $C_0 = \{\vec{u}_i : i < m\}$ and $C_1 = \{\vec{v}_i : i < n\}$ and $\vec{u}$ and $\vec{v}$ such that $S_0 = \Sigma(\{\vec{u}\}; C_0)$ and $S_1 := \Sigma(\{\vec{v}\}; C_1)$. Notice that $\vec{w} \in S_0 \cap S_1$ iff there are natural numbers κ_i $(i < m)$ and λ_j $(j < n)$ such that

$$\vec{w} = \vec{u} + \sum_{i<m} \kappa_i \vec{u}_i = \vec{v} + \sum_{i<n} \lambda_i \vec{v}_i \tag{2.130}$$

So, we have to show that the set of these $\vec{w}$ is semilinear.

The equations are now taken as linear equations with κ_i, $i < m$ and λ_i, $i < n$, as variables. Thus we have equations for $m+n$ variables. We solve these equations first in $\mathbb{Q}^{m+n}$. The solutions form an affine subspace $V \subseteq \mathbb{Q}^{m+n} \cong \mathbb{Q}^m \oplus \mathbb{Q}^n$. By Lemma 2.99, $V \cap \omega^{m+n}$ is semilinear, and so is its projection onto ω^m (or to ω^n for that matter). Let it be $\bigcup_{i<p} L_i$, where for each $i < p$, $L_i \subseteq \omega^m$ is linear. Thus there is a representation of L_i as

$$L_i = \vec{\theta} + \omega\vec{\eta}_0 + \cdots + \omega\vec{\eta}_{\gamma-1} \tag{2.131}$$

Now put

$$W_i := \{\vec{u} + \sum_{i<m} \vec{\kappa}(i)\vec{u}_i : \vec{\kappa} \in L_i\} \tag{2.132}$$

From the construction we get that

$$S_0 \cap S_1 = \bigcup_{i<p} W_i \tag{2.133}$$

Define vectors $\vec{q}_i := \sum_{j<m} \vec{\eta}_i(j)\vec{u}_i$, $i < \gamma$ and $\vec{r} := \vec{c} + \sum_{j<m} \vec{\theta}(j)\vec{u}_i$. Then

$$W_i = \vec{r} + \omega\vec{q}_0 + \cdots + \omega\vec{q}_{\gamma-1} \tag{2.134}$$

So, the W_i are linear. This shows the claim. $\square$

Lemma 2.101 *If $S \subseteq \omega^n$ is semilinear, so is its projection $\pi_n[S]$.*

We need one more prerequisite. Say that a first–order theory T has **quantifier elimination** if for every formula $\varphi(\vec{x})$ there exists a quantifier free formula $\chi(\vec{x})$ such that $T \vdash \varphi(\vec{x}) \leftrightarrow \chi(\vec{x})$. We follow the proof of [**?**].

Theorem 2.102 (Presburger) *Presburger Arithmetic has quantifier elimination.*

Proof. It is enough to show that for every formula $(\exists \mathtt{x}_0)\,\varphi(\vec{y},\mathtt{x}_0)$ with $\varphi(\vec{y},x)$ quantifier free there exists a quantifier free formula $\chi(\vec{y})$ such that

$$\mathbb{Z} \vDash (\forall \vec{y})\,(\exists \mathtt{x}_0)\,(\varphi(\vec{y},\mathtt{x}_0) \leftrightarrow \chi(\vec{y})) \tag{2.135}$$

We may further eliminate negation (see the remarks above) and disjunctions inside $\varphi(\vec{y},x)$ (since $(\exists \mathtt{x}_0)(\alpha \vee \beta)$ is equivalent with $((\exists \mathtt{x}_0)\alpha) \vee ((\exists \mathtt{x}_0)\beta)$. Finally, we may assume that all conjuncts contain $\mathtt{x}_0$. For if α does not contain $\mathtt{x}_0$ free, $(\exists \mathtt{x}_0)(\alpha \wedge \beta)$ is equivalent to $(\alpha \wedge (\exists \mathtt{x}_0)\beta)$. So, φ can be assumed to be a conjunction of atomic formulae of the following form:

$$(\exists \mathtt{x}_0)\,(\textstyle\bigwedge_{i<p} n_i\mathtt{x}_0 = t_i \wedge \bigwedge_{i<q} n_i'\mathtt{x}_0 < t_i' \wedge \bigwedge_{i<r} n_i''\mathtt{x}_0 > t_i'' \\ \wedge \bigwedge_{i<s} n_i'''\mathtt{x}_0 \equiv_{m_i} t_i''') \tag{2.136}$$

Since $s \equiv_m t$ is equivalent with $ns \equiv_m nt$, so after suitable multiplication we may see to it that all the n_i, n_i', n_i'' and n_i''' are the same number ν.

$$(\exists \mathtt{x}_0)\,(\textstyle\bigwedge_{i<p} \nu\mathtt{x}_0 = \tau_i \wedge \bigwedge_{i<q} \nu\mathtt{x}_0 < \tau_i' \wedge \bigwedge_{i<r} \nu\mathtt{x}_0 > \tau_i'' \\ \wedge \bigwedge_{i<s} \nu\mathtt{x}_0 \equiv_{m_i} \tau_i''') \tag{2.137}$$

We may rewrite the formula in the following way (replacing $\nu\mathtt{x}_0$ by $\mathtt{x}_0$ and adding instead the condition that $\mathtt{x}_0$ is divisible by ν).

$$(\exists \mathtt{x}_0)\,(\mathtt{x}_0 \equiv_\nu 0 \wedge \textstyle\bigwedge_{i<p} \mathtt{x}_0 = \tau_i \wedge \bigwedge_{i<q} \mathtt{x}_0 < \tau_i' \wedge \bigwedge_{i<r} \mathtt{x}_0 > \tau_i'' \\ \wedge \bigwedge_{i<s} \mathtt{x}_0 \equiv_{m_i} \tau_i''') \tag{2.138}$$

Assume that $p > 0$. Then the first set of conjunctions is equivalent with the conjunction of $\bigwedge_{i<j<p} \tau_i = \tau_j$ (which does not contain $\mathtt{x}_0$) and $\mathtt{x}_0 = \tau_0$. We may therefore eliminate all occurrences of $\mathtt{x}_0$ by τ_0 in the formula.

Thus, from now on we may assume that $p = 0$. Furthermore, notice that $(\mathtt{x}_0 < \sigma \wedge \mathtt{x}_0 < \tau)$ is equivalent to $(\mathtt{x}_0 < \sigma \wedge \sigma \leq \tau) \vee (\mathtt{x}_0 < \tau \wedge \tau < \sigma)$. This means that we can assume $q \leq 1$, and likewise that $r \leq 1$. Next we show that we can actually have $s \leq 1$. To see this, notice the following.

Let u, v, w, x be integers, $w, x > 1$, and let p be the least common multiple of w and x. Then $\gcd(p/w, p/x) = 1$, and so there exist integers m, n such that $1 = m \cdot p/w + n \cdot p/x$. It follows that the following are equivalent.

① $y \equiv u \pmod{w}$ and $y \equiv v \pmod{x}$

② $u \equiv v \pmod{\gcd(w,x)}$ and $y \equiv m(p/w)u + n(p/x)v \pmod{p}$.

The Euclidean algorithm yields numbers m and n as required (see [**?**]). Now suppose that the first obtains. Then $y - u = ew$ and $y - v = fx$ for some numbers e and f. Then $u - v = fx - ew$, which is divisible by $\gcd(x, w)$. So, $u \equiv v \pmod{\gcd(w,x)}$. Furthermore,

$$\begin{aligned} y - m(p/w)u - n(p/x)v &= m(p/w)y + n(p/x)y \\ &\quad - m(p/w)u - n(p/x)v \\ &= m(p/w)(y-u) \\ &\quad + n(p/x)(y-v) \\ &= m(p/w)em + n(p/x)fn \\ &\equiv 0 \pmod{p} \end{aligned} \tag{2.139}$$

So, the second holds. Conversely, if the second holds, then for some k we have $u - v = k\gcd(w,x)$. Then

$$\begin{aligned} y - u &= y - m(p/w)u - n(p/x)u \\ &= y - m(p/w)u - n(p/x)v - n(p/x)k \cdot \gcd(m,n) \\ &\equiv 0 \pmod{w} \end{aligned} \tag{2.140}$$

Analogously $y \equiv v \pmod{x}$ is shown.

Using this equivalence we can reduce the congruence statements to a conjunction of congruences where only one involves $\mathtt{x}_0$.

This leaves us with 8 possibilities. If $r = 0$ or $s = 0$ the formula is actually trivially true. So, $(\exists \mathtt{x}_0)(\mathtt{x}_0 < \tau)$, $(\exists \mathtt{x}_0)(\upsilon < \mathtt{x}_0)$, $(\exists \mathtt{x}_0)(\mathtt{x}_0 =_m \xi)$, as well as $(\exists \mathtt{x}_0)(\mathtt{x}_0 < \tau \wedge \mathtt{x}_0 \equiv_m \xi)$ and $(\exists \mathtt{x}_0)(\upsilon < \mathtt{x}_0 \wedge \mathtt{x}_0 \equiv_m \xi)$ can all be dropped or replaced by $\top$. Finally, $(\exists \mathtt{x}_0)(\mathtt{x}_0 < \tau \wedge \upsilon < \mathtt{x}_0)$ is equivalent with $\upsilon + 1 < \tau$ and $(\exists \mathtt{x}_0)(\mathtt{x}_0 < \tau \wedge \upsilon < \mathtt{x}_0 \wedge \mathtt{x}_0 =_m \xi)$ is equivalent with $\bigvee_{i<m}(\tau + 1 + i < \upsilon \wedge \tau + 1 + i \equiv_m \xi)$. This shows the claim. □

Theorem 2.103 (Ginsburg & Spanier) *A subset of ω^n is semilinear iff it is definable in Presburger Arithmetic.*

Proof. ($\Rightarrow$) Every semilinear set is definable in Presburger Arithmetic. To see this it is enough to show that linear sets are definable. For if M is a union of N_i, $i < p$, and each N_i is linear and hence definable by a formula $\varphi_i(\vec{x})$, then M is definable by $\bigvee_{i<p} \varphi_i(\vec{x})$. Now let $M = \vec{v} + \omega\vec{v}_0 + \cdots + \omega\vec{v}_{m-1}$ be linear. Then put

$$\varphi(\vec{x}) := (\exists \mathtt{x}_n)(\exists \mathtt{x}_{n+1})\cdots(\exists \mathtt{x}_{n+m-1})(\bigwedge_{i<m} 0 \leq \mathtt{x}_{n+i} \wedge \bigwedge_{i<n} (\vec{v}(i) + \sum_{j<m} \mathtt{x}_{n+i}\vec{v}(i)_j = \mathtt{x}_i)) \tag{2.141}$$

$\varphi(\vec{x})$ defines M. ($\Leftarrow$) Let $\varphi(\vec{x})$ be a formula defining S. By Theorem 2.102, there exists a quantifier free formula $\chi(\vec{x})$ defining S. Moreover, as we have remarked above, χ can be assumed to be negation free. Thus, χ is a disjunction of conjunctions of atomic formulae. By Lemma 2.100, the set of semilinear subsets of ω^n is closed under intersection of members, and it is also closed under union. Thus, all we need to show is that atomic formulae define semilinear sets. Now, observe that $\mathtt{x}_0 \equiv_m \mathtt{x}_1$ is equivalent to $(\exists \mathtt{x}_2)(\mathtt{x}_0 = \mathtt{x}_1 + m\mathtt{x}_2)$, which is semilinear, as it is the projection of $\mathtt{x}_0 = \mathtt{x}_1 + m\mathtt{x}_2$ onto the first two components. □

Exercise 72. Let $|A| = 1$. Show that $\mathfrak{Z}(A)$ is isomorphic to $\mathfrak{M}(A)$. Derive from this that there are only countably many semilinear languages over A.

Exercise 73. Let $L \subseteq A^*$. Call L **almost periodical** if there are numbers p (the modulus of periodicity) and n_0 such that for all $\vec{x} \in L$ with length $\geq n_0$ there is a string $\vec{y} \in L$ such that $|\vec{y}| = |\vec{x}| + p$. Show that a semilinear language is almost periodical.

Exercise 74. Let $A = \{\mathtt{a}, \mathtt{b}\}$. Further, let $U := \mathtt{a}^* \cup \mathtt{b}^*$. Now let $N \subseteq M(A)$ be a set such that $N - U$ is infinite. Show that there are $2^{\aleph_0}$ many languages L with $\mu[L] = N$. (The cardinality of A^* is $\aleph_0$, hence there can be no more than $2^{\aleph_0}$ such languages. The exercise consists in showing that there are no less of them either.)

Exercise 75. Show that semilinear languages have the following pumping property: *For every semilinear set $V \subseteq \omega^n$ there exists a number n such that if $\vec{v} \in V$ has length $\geq n$, there exist $\vec{w}$ and $\vec{x}$ such that $\vec{v} = \vec{w} + \vec{x}$ and $\vec{w} + \omega\vec{x} \subseteq V$.*

Exercise 76. Let $\Omega \subseteq \omega$. Let $V_\Omega \subseteq \omega^2$ be defined by

$$V_\Omega := \{\langle m, n\rangle : m \neq n \text{ or } m \in \Omega\} \tag{2.142}$$

Show that V_Ω satisfies the pumping property of the previous exercise. Show

further that V_Ω is semilinear iff Ω is.

Exercise 77. Show that for every sentence φ of Presburger Arithmetic it is decidable whether or not it is true in $\mathbb{Z}$. *Hint.* Use quantifiers elimination and the fact that the elimination is constructive.

6. Parikh's Theorem

Now we shall turn to the already announced embedding of context free tree sets into tree sets generated by UTAGs. (The reader may wonder why we speak of sets and not of classes. In fact, we shall tacitly assume that trees are really tree domains, so that classes of finite trees are automatically sets.) Let $G = \langle \mathtt{S}, N, A, R\rangle$ be a CFG. We want to define a tree adjunction grammar $\mathsf{Ad}_G = \langle \mathbb{C}_G, N, A, \mathbb{A}_G\rangle$ such that $L_B(G) = L_B(\mathsf{Ad}_G)$. We define $\mathbb{C}_G$ to be the set of all (ordered labelled) tree (domains) $\mathfrak{B}$ which can be generated by $L_B(G)$ and which are centre trees and in which on no path not containing the root some nonterminal symbol occurs twice. Since there are only finitely many symbols and the branching is finite, this set is actually finite. Now we define $\mathbb{A}_G$. Let $\mathbb{A}_G$ contain all adjunction trees $\mathfrak{B}_X$, $X \in N$, (modulo identification of Y^0, Y^1 with Y for all $Y \in N$) such that (1) $\mathfrak{B}_X$ can be derived from X in γG, (2) no symbol occurs twice along a path that does contain the root. Also $\mathbb{A}_G$ is finite. It is not hard to show that $L_B(\mathsf{Ad}_G) \subseteq L_B(G)$. The reverse inclusion we shall show by induction on the number of nodes in the tree (domain). Let $\mathfrak{B}$ be in $L_B(G)$. Either there is a path not containing the root along which some symbol occurs twice, or there is not. In the second case the tree is in $\mathbb{C}_G$. Hence $\mathfrak{B} \in L_B(\mathsf{Ad}_G)$ and we are done. In the first case we choose an $x \in B$ of minimal height such that there is a $y < x$ with identical label; let the label be X. Consider the subtree $\mathfrak{U}$ induced by the set $(\downarrow x - \downarrow y) \cup \{y\}$. We claim that $\mathfrak{U} \in \mathbb{A}_G$. For this we have to show the following. (a) $\mathfrak{U}$ is an adjunction tree, (b) $\mathfrak{U}$ can be deduced from X, (c) no symbol symbol occurs twice along a path which does not contain x. Ad (a). A leaf of $\mathfrak{U}$ is either a leaf of $\mathfrak{B}$ or $= y$. In the first case the label is a terminal symbol in the second case it is identical to that of the root. Ad (b). If $\mathfrak{B}$ is a tree of γG then $\mathfrak{U}$ can be derived from X. Ad (c). Let π be a path which does not contain x and let $u, v \in \pi$ nodes with identical label and $u < v$. Then $v < x$, and this contradicts the minimality of x. Hence all three conditions are met. So we can disembed $\mathfrak{U}$. This means that there is a tree $\mathfrak{B}'$ such that $\mathfrak{B}$ is derived from $\mathfrak{B}'$ by adjoining $\mathfrak{U}$. We have $\mathfrak{B}' \in L_B(G)$ and by induction hypothesis $\mathfrak{B}' \in L_B(\mathsf{Ad}_G)$. Hence $\mathfrak{B} \in L_B(\mathsf{Ad}_G)$, which had

to be shown.

Theorem 2.104 (Joshi & Levy & Takahashi) *Every set of labelled ordered tree domains generated by a CFG is also one generated by a UTAG.* □

Now we shall prove Parikh's Theorem for UTAGs. Let α be a letter and $\mathfrak{B}$ a tree. Then $\sigma_\alpha(\mathfrak{B})$ is the number of nodes whose label is α. If $\mathfrak{B}$ is an adjunction tree then the label of the root is *not counted*. Now let $\langle \mathbb{C}, N, A, \mathbb{A} \rangle$ be a UTAG and $\mathbb{C} = \{\mathfrak{C}_i : i < \alpha\}$, $\mathbb{A} = \{\mathfrak{A}_j : j < \beta\}$.

Lemma 2.105 *Let $\mathfrak{B}'$ result from $\mathfrak{B}$ by adjoining the tree $\mathfrak{A}$. Then $\sigma_\alpha(\mathfrak{B}') = \sigma_\alpha(\mathfrak{B}) + \sigma_\alpha(\mathfrak{A})$.*

The proof of this lemma is easy. From this it follows that we only need to know for an arbitrarily derived tree how many times which tree has been adjoined and what the starting tree was. So let $\mathfrak{B}$ be a tree which resulted from $\mathfrak{C}_i$ by adjoining $\mathfrak{A}_j$ p_j times, $j < \beta$. Then

$$\sigma_\alpha(\mathfrak{B}) = \sigma_\alpha(\mathfrak{C}_i) + \sum_{i<\beta} p_j \cdot \sigma_\alpha(\mathfrak{A}_j) \tag{2.143}$$

Let now $\mu(\mathfrak{B}) := \sum_{a \in A} \sigma_a(\mathfrak{B}) \cdot a$. Then

$$\mu(\mathfrak{B}) = \mu(\mathfrak{C}_i) + \sum_{i<\beta} p_j \cdot \mu(\mathfrak{B}_j) \tag{2.144}$$

We define the following sets

$$\Sigma_i := \mu(\mathfrak{C}_i) + \sum_{j<\beta} \omega\mu(\mathfrak{A}_j) \tag{2.145}$$

Then $\mu[L_B(\langle \mathbb{C}, \mathbb{A} \rangle)] \subseteq \bigcup_{i<n} \Sigma_i$. However, equality need not always hold. We have to notice the following problem. A tree $\mathfrak{A}_j$ can be adjoined to a tree $\mathfrak{B}$ only if its root label actually occurs in the tree $\mathfrak{B}$. Hence not all values of $\bigcup \Sigma_i$ are among the values under μ of a derived tree. However, if a tree can be adjoined *once* it can be adjoined any number of times and to all trees that result from this tree by adjunction. Hence we modify our starting set of trees somewhat. We consider the set D of all pairs $\langle k, W \rangle$ such that $k < \alpha$, $W \subseteq \beta$ and there is a derivation of a tree that starts with $\mathfrak{C}_k$ and uses exactly the trees from W. For $\langle k, W \rangle \in D$

$$L(k, W) = \mu(\mathfrak{C}_i) + \sum_{j \in W} \omega \cdot \mu(\mathfrak{A}_j) \tag{2.146}$$

Then $L := \bigcup \langle L(k, W) : \langle k, W \rangle \in D \rangle$ is semilinear. At the same time it is the set of all $\mu(\mathfrak{B})$ where $\mathfrak{B}$ is derivable from $\langle \mathbb{C}, N, A, \mathbb{A} \rangle$.

Theorem 2.106 *Let L be the language of an unregulated tree adjunction grammar then L is semilinear.* □

Corollary 2.107 (Parikh) *Let L be context free. Then L is semilinear.* □

This theorem is remarkable is many respects. We shall meet it again several times. Semilinear sets are closed under complement (Theorem 2.103) and hence also under intersection. We shall show, however, that this does not hold for semilinear languages.

Proposition 2.108 *There are CFLs L_1 and L_2 such that $L_1 \cap L_2$ is not semilinear.*

Proof. Let $M_1 := \{\mathtt{a}^n\mathtt{b}^n : n \in \omega\}$ and $M_2 := \{\mathtt{b}^n\mathtt{a}^{2n} : n \in \omega\}$. Put

$$L_1 := \mathtt{b}M_1^*\mathtt{a}^* \qquad\qquad L_2 := M_2^+ \tag{2.147}$$

Because of Theorem 1.67 L_1 and L_2 are context free. Now look at $L_1 \cap L_2$. It is easy to see that the intersection consists of the following strings.

$$\mathtt{ba}^2, \quad \mathtt{ba}^2\mathtt{b}^2\mathtt{a}^4, \quad \mathtt{ba}^2\mathtt{b}^2\mathtt{a}^4\mathtt{b}^4\mathtt{a}^8, \quad \mathtt{ba}^2\mathtt{b}^2\mathtt{a}^4\mathtt{b}^4\mathtt{a}^8\mathtt{b}^8\mathtt{a}^{16}, \ldots \tag{2.148}$$

The Parikh image is $\{(2^{n+2}-2)\mathtt{a} + (2^{n+1}-1)\mathtt{b} : n \in \omega\}$. This set is not semilinear, since the result of deleting the symbol b (that is, the result of applying the projection onto $\mathtt{a}^*$) is not almost periodical. □

We know that for every semilinear set $N \subseteq M(A)$ there is a regular grammar G such that $\mu[L(G)] = N$. However G can be relatively complex. Now the question arises whether the complete preimage $\mu^{-1}[N]$ under μ is at least regular or context free. This is not the case. However, we do have the following.

Theorem 2.109 *The full preimage of a semilinear set over a single letter alphabet is regular.*

This is the best possible result. The theorem becomes false as soon as we have two letters.

Theorem 2.110 *The full preimage of $\omega(\mathtt{a}+\mathtt{b})$ is not regular; it is however context free. The full preimage of $\omega(\mathtt{a}+\mathtt{b}+\mathtt{c})$ is not context free.*

Proof. We show the second claim first. Let

$$W := \mu^{-1}[\omega(\mathtt{a}+\mathtt{b}+\mathtt{c})] \tag{2.149}$$

Assume that W is context free. Then the intersection with the regular language $\mathtt{a}^*\mathtt{b}^*\mathtt{c}^*$ is again context free. This is precisely the set $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n : n \in \omega\}$. Contradiction. Now for the first claim. Denote by $b(\vec{x})$ the number of occurrences of $\mathtt{a}$ in $\vec{x}$ minus the number of occurrences of $\mathtt{b}$ in $\vec{x}$. Then $V := \{\vec{x} : b(\vec{x}) = 0\}$ is the full preimage of $\omega(\mathtt{a}+\mathtt{b})$. V is not regular; otherwise the intersection with $\mathtt{a}^*\mathtt{b}^*$ is also regular. However, this is $\{\mathtt{a}^n\mathtt{b}^n : n \in \omega\}$. Contradiction. However, V is context free. To show this we shall construct a CFG G over $A = \{\mathtt{a},\mathtt{b}\}$ which generates V. We have three nonterminals, $\mathtt{S}$, $\mathtt{A}$, and $\mathtt{B}$. The rules are

$$\begin{array}{lcl} \mathtt{S} & \to & \mathtt{SS} \mid \mathtt{AB} \mid \mathtt{BA} \\ \mathtt{A} & \to & \mathtt{AS} \mid \mathtt{SA} \mid \mathtt{a} \\ \mathtt{B} & \to & \mathtt{BS} \mid \mathtt{SB} \mid \mathtt{b} \end{array} \tag{2.150}$$

The start symbol is $\mathtt{S}$. We claim: $\mathtt{S} \vdash_G \vec{x}$ iff $b(\vec{x}) = 0$, $\mathtt{A} \vdash_G \vec{x}$ iff $b(\vec{x}) = 1$ and $\mathtt{B} \vdash_G \vec{x}$ iff $b(\vec{x}) = -1$. The directions from left to right are easy to verify. It therefore follows that $V \subseteq L(G)$. The other directions we show by induction on the length of $\vec{x}$. It suffices to show the following claim.

> If $b(\vec{x}) \in \{1,0,-1\}$ there are $\vec{y}$ and $\vec{z}$ such that $|\vec{y}|, |\vec{z}| < |\vec{x}|$ and such that $\vec{x} = \vec{y}\vec{z}$ as well as $|b(\vec{y})|, |b(\vec{z})| \leq 1$.

Hence let $\vec{x} = \prod_{i<n} x_i$ be given. Define $k(\vec{x}, j) := b({}^{(j)}\vec{x})$, and $K := \{k(\vec{x}, j) : j < n+1\}$. As is easily seen, $K = [m, m']$ with $m \leq 0$. Further, $k(\vec{x}, n) = b(\vec{x})$. (a) Let $b(\vec{x}) = 0$. Then put $\vec{y} := x_0$ and $\vec{z} := \prod_{0<i<n} x_i$. This satisfies the conditions. (b) Let $b(\vec{x}) = 1$. Case 1: $x_0 = \mathtt{a}$. Then put again $\vec{y} := x_0$ and $\vec{z} := \prod_{0<i<n} x_i$. Case 2: $x_0 = \mathtt{b}$. Then $k(\vec{x}, 1) = -1$ and there is a j such that $k(\vec{x}, j) = 0$. Put $\vec{y} := \prod_{i<j} x_i$, $\vec{z} := \prod_{j \leq i<n} x_i$. Since $0 < j < n$, we have $|\vec{y}|, |\vec{z}| < |\vec{x}|$. Furthermore, $b(\vec{y}) = 0$ and $b(\vec{z}) = 1$. (c) $b(\vec{x}) = -1$. Similar to (b). □

Exercise 78. Let $|A| = 1$ and Ad be a UTAG. Show that the language generated by Ad over A^* is regular.

Exercise 79. Prove Theorem 2.109. *Hint.* Restrict your attention first to the case that $A = \{\mathtt{a}\}$.

Exercise 80. Let $N \subseteq M(A)$ be semilinear. Show that the full preimage is of Type 1 (that is, context sensitive). *Hint.* It is enough to show this for linear sets.

Exercise 81. In this exercise we sketch an alternative proof of Parikh's Theorem. Let $A = \{\mathtt{a}_i : i < n\}$ be an alphabet. In analogy to the regular terms we define semilinear terms. (a) $\mathtt{a}_i$, $i < n$, is a semilinear term, with interpretation $\{\vec{e}_i\}$. (b) If A and B are semilinear terms, so is $A \oplus B$ with interpretation $\{\vec{u} + \vec{v} : \vec{u} \in A, \vec{v} \in B\}$, $A \cup B$, with interpretation $\{\vec{u} : \vec{u} \in A \text{ or } \vec{u} \in B\}$ and ωA with interpretation $\{k\vec{u} : k \in \omega, \vec{u} \in A\}$. The first step is to translate a CFG into a set of equations of the form $X_i = C_i(X_0, X_1, \ldots, X_{q-1})$, q the number of nonterminals, C_i semilinear terms. This is done as follows. Without loss of generality we can assume that in a rule $X \to \vec{\alpha}$, $\vec{\alpha}$ contains a given variable at most once. Now, for each nonterminal X let $X \to \vec{\alpha}_i$, $i < p$, be all the rules of G. Corresponding to these rules there is an obvious equation of the form

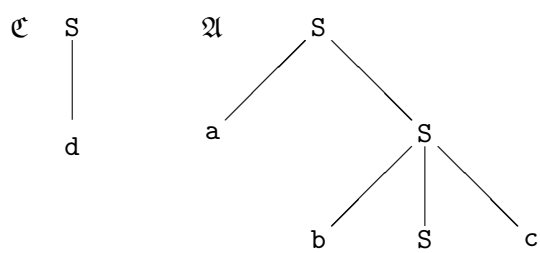
$$X = A \cup (B \oplus X) \text{ or } X = A \tag{2.151}$$

where A and B are semilinear terms that do not contain X. The second step is to prove the following lemma:

> *Let $X = A \cup (B \oplus X) \cup (C \oplus \omega X)$, with A, B and C semilinear terms not containing X. Then the least solution of that equation is $A \cup \omega B \cup \omega C$. If $B \oplus X$ is missing from the equation, the solution is $A \cup \omega C$, and if $C \oplus \omega X$ is missing the solution is $A \cup \omega B$.*

Using this lemma it can be shown that the system of equations induced by G can be solved by constant semilinear terms for each variable.

Exercise 82. Show that the UTAG $\langle \{\mathfrak{C}\}, \{\mathtt{S}\}, \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}, \{\mathfrak{A}\} \rangle$ generates exactly the strings of the form $\vec{x}\mathtt{dc}^n$, where $\vec{x}$ is a string of n $\mathtt{a}$'s and n $\mathtt{b}$'s such that every prefix of $\vec{x}$ has at least as many $\mathtt{a}$'s as $\mathtt{b}$'s.



Show also that this language is not context free. (This example is due to [**?**].)

7. Are Natural Languages Context Free?

We shall finish our discussion of CFLs by looking at some naturally arising languages. We shall give examples of languages and constructions which are definitely not context free. The complexity of natural languages has been high on the agenda ever since the introduction of this hierarchy. Chomsky's intention was in part to discredit structuralism, which he identified with the view that natural languages always are context free. By contrast, he claimed that natural languages are not context free and gave many examples. It is still widely believed that Chomsky had won his case. (For an illuminating discussion read [**?**].)

It has emerged over the years that the arguments given by Noam Chomsky and Paul Postal against the context freeness of natural languages were faulty. Gerald Gazdar, Geoffrey Pullum and others have repeatedly found holes in the argumentation. This has finally led to the bold claim that natural languages are all context free (see [**?**]). The first to deliver a correct proof of the contrary was Riny Huybregts, only shortly later followed by Stuart Shieber. (See [**?**] and [**?**].) Counterevidence from Bambara was given by Culy [**?**]. Of course, it was hardly doubted that from structural point of view natural languages are not context free (see the analyses of Dutch and German within GB, for example, or [**?**]), but it was not shown decisively that they are not even weakly context free.

How is a proof the non context freeness of a language L possible? A typical method is this. Take a suitable regular language R and intersect it with L. If L is context free, so is $L \cap R$. Now choose a homomorphism h and map the language $L \cap R$ onto a known non–CFL. We give an example from the paper by Stuart Shieber. Look at (2.152) – (2.154). If one looks at the nested infinitives in Swiss German (first rows) we find that they are structured differently from English (last rows) and High German (middle rows). (Instead of a gloss, we offer the following parallels: `das` ≙ `dass` ≙ `that`, `hälfe` ≙ `helfen` ≙ `help`, `aastriche` ≙ `anstreichen` ≙ `paint`, `huus` ≙ `Haus` ≙ `house`, `mer` ≙ `wir` ≙ `we`, `lönd` ≙ `lassen` ≙ `let`, `chind` ≙ `Kinder` ≙ `children`.)

(2.152) `Jan säit, das Hans es huus aastricht.`
`Jan sagt, dass Hans das Haus anstreicht.`
`Jan says that Hans is painting the house.`

(2.153) `Jan säit, das mer em Hans es huus hälfed`

aastriche.

Jan sagt, dass wir Hans das Haus anstreichen helfen.

Jan says that we help Hans paint the house.

(2.154) Jan säit, das mer d'chind em Hans es huus lönd hälfe aastriche.

Jan sagt, dass wir die Kinder Hans das Haus anstreichen helfen lassen.

Jan says that we let the children help Hans paint the house.

(2.155) *Jan säit, das mer de Hans es huus hälfed aastriche.

(2.156) *Jan säit, das mer em chind em Hans es huus lönd hälfe aastriche.

By asking who does what to whom (we let, the children help, Hans paints) we see that the constituents are quite different in the three languages. Subject and corresponding verb are together in English (see (2.157a)), in High German they are on opposite sides of the embedded infinitive (see (2.157b), this is called the **nesting order**). Swiss German, however, is still different. The verbs follow each other in the reverse order as in German (so, they occur in the order of the subjects, see (2.157c)). This is called the **crossing order**.

(2.157a) $S_1\ V_1\ S_2\ V_2\ S_3\ V_3\ \ldots$

(2.157b) $S_1\ S_2\ S_3\ \ldots\ V_3\ V_2\ V_1$

(2.157c) $S_1\ S_2\ S_3\ \ldots\ V_1\ V_2\ V_1 \ldots$

Now we assume — this is an empirical assumption, to be sure — that this is the general pattern. It shall be emphasized that the processing of such sentences becomes difficult with four or five infinitives. Nevertheless, the resulting sentences are considered grammatical.

Now we proceed as follows. The verbs require accusative or dative on their complements. The following examples show that there is a difference between dative and accusative. In (2.155) `de Hans` is accusative and the complement of `aastriche`, which selects dative. The resulting sentence is ungrammatical. In (2.156), `em chind` is dative, while `lönd` selects accusative.

Again the sentence is ungrammatical. We now define the following regular language (recall the definition of $\diamond$ from Section 1.2).

$$
\begin{aligned}
(2.158)\qquad R := & \texttt{Jan} \diamond \texttt{säit}^\frown\texttt{,} \diamond \texttt{das} \diamond \texttt{mer} \\
& \diamond ((\texttt{em}^\frown\square \cup \texttt{d'} \cup \texttt{de}^\frown\square)^\frown(\texttt{chind}^\frown\square \cup \texttt{Hans}^\frown\square))^* \\
& {}^\frown\texttt{es} \diamond \texttt{huus} \\
& \diamond (\texttt{laa}^\frown\square \cup \texttt{lönd}^\frown\square \cup \texttt{hälfe}^\frown\square)^*{}^\frown\texttt{aastriche}^\frown\texttt{.}
\end{aligned}
$$

This is defined over the standard alphabet. It is not hard to see (invoking the Transducer Theorem, 6.40) that the corresponding language over the alphabet of lexemes is also regular. We define the following mapping from the lexemes (denoted by their strings). v sends `d'`, `de`, `laa` and `lönd` to `a`, `em` and `hälfe` to `d`, everything else inculding the blank is mapped to ε. The claim is that

$$(2.159)\qquad h[S \cap R] = \{\vec{x}\vec{x} : \vec{x} \in \texttt{a} \cdot (\texttt{a} \cup \texttt{d})^*\}$$

To this end we remark that a verb is sent to `d` if it has a dative object and to `a` if it has an accusative object. An accusative object is of the form `de` N or `d'` N (N a noun) and is mapped to `a` by $\overline{v}$. A dative object has the form `em` N, N a noun, and is mapped onto `d`. Since the nouns are in the same order as the associated infinitives we get the desired result.

In mathematics we find a phenomenon similar to Swiss German. Consider the integral of a function. If $f(x)$ is a function, the integral of $f(x)$ in the interval $[a,b]$ is denoted by

$$(2.160)\qquad \int_a^b f(x)dx$$

This is not in all cases well formed. For example, $\int_0^1 x^{-1}dx$ is ill formed, since there Riemann approximation leads to a sequence which is not bounded, hence has no limit. Similarly, $\lim_{n\to\infty}(-1)^n$ does not exist. Notice that the value range of x is written at the integral sign without saying with what variable the range is associated. For example, let us look at

$$(2.161)\qquad \int_a^b \int_c^d f(x,y)dxdy$$

The rectangle over which we integrate the function is $a \leq x \leq b$ and $c \leq y \leq d$. Hence, the first integral sign corresponds to the operator dx, which occurs first

in the list. Likewise for three integrals:

$$\int_{a_0}^{b_0}\int_{a_1}^{b_1}\int_{a_2}^{b_2} f(x_0,x_1,x_2)dx_0dx_1dx_2 \tag{2.162}$$

where the value range is $a_i \leq x_i \leq b_i$ for all $i < 3$. Consider the following functions:

$$f(x_0,\ldots,x_n) := \prod_{i<n} x_i^{\alpha_i} \tag{2.163}$$

with $\alpha_i \in \{-1,1\}$, $i < n$. Further, we allow for the interval $[a_i,b_i]$ either $[0,1]$ or $[1,2]$. Then an integral expression

$$\int_{a_0}^{b_0}\int_{a_1}^{b_1}\cdots\int_{a_{n-1}}^{b_{n-1}} f(x_0,x_1,\ldots,x_{n-1})dx_0dx_1\cdots dx_{n-1} \tag{2.164}$$

is well formed iff $a_i > 0$ for all $i < n$ such that $\alpha_i = -1$. The dependencies are crossing, and the order of elements is exactly as in Swiss German (considering the boundaries and the variables). The complication is the mediating function, which determines which of the boundary elements must be strictly positive.

In [**?**], it is argued that even English is not context free. The argument applies a theorem from [**?**]. If L is a language, let L_n denote the set of strings that are in L and have length n. The following theorem makes use of the fact that a string of length n possesses $n(n+1)/2$ proper substrings and that $n(n+1)/2 < n^2$ for all $n > 1$. Denote by $\ulcorner c \urcorner$ the smallest integer $\geq c$.

Theorem 2.111 (Interchange Lemma) *Let L be a CFL. Then there exists a real number c_L such that for every natural number $n > 0$ and every set $Q \subseteq L_n$ there is a $k \geq \ulcorner |Q|/(c_L n^2) \urcorner$, and strings $\vec{x}_i$, $\vec{y}_i$, $\vec{z}_i$, $i < k$, such that*

① *for all $i < i < k$: $|\vec{x}_i| = |\vec{x}_j|$, $|\vec{y}_i| = |\vec{y}_j|$, and $|\vec{z}_i| = |\vec{z}_j|$.*

② *for all $i < k$: $|\vec{y}_i|, |\vec{x}_i\vec{z}_i| > 0$,*

③ *for all $i < k$: $\vec{x}_i\vec{y}_i\vec{z}_i \in Q$, and*

④ *for all $i, j < k$: $\vec{x}_i\vec{y}_j\vec{z}_i \in L_n$.*

Proof. Let G be a CFG that generates L. Let $c_L := |N|$. We show that c_L satisfies the above conditions. Take any set $Q \subseteq L_n$. Then there is $E \subseteq Q$ of cardinality $\geq 2|Q|/(n+1)n$ and numbers $k \geq 0$ and $\ell > 0$ such that every member

of E possesses a decomposition $\vec{x}\vec{y}\vec{z}$ where $\vec{x}$ has length k, $\vec{y}$ has length ℓ, and $\langle\vec{x},\vec{z}\rangle$ is a constituent occurrence of $\vec{y}$ in the string. It is then clear that there is a subset $F \subseteq E$ of cardinality $\geq 2|Q|/((n+1)n|N|) > |Q|/(c_L n^2)$ such that all $\langle\vec{x},\vec{z}\rangle$ are constituent occurrences of identical nonterminal category. The above conditions are now satisfied for F. Moreover, $|F| \geq \ulcorner |Q|/(c_L n^2) \urcorner$, which had to be shown. □

Note that if the sequence of numbers L_n/n^2 is bounded, then L satisfies the conditions of the Interchange Lemma. For assume that there is a c such that for all n we have $L_n/n^2 \leq c$. Then $c_L := \sup\{|L_n|/n^2 : n \in \mathbb{N}\} \leq c$. Then for every n and every subset Q of L_n, $\ulcorner |Q|/(c_L n^2) \urcorner \leq \ulcorner |L_n|/(c_L n^2) \urcorner \leq 1$. However, with $k = 1$ the conditions above become empty.

Theorem 2.112 *Let $L \subseteq A^*$ be a language such that $(|L_n|/n^2)_{n\in\mathbb{N}}$ is a bounded sequence. Then L satisfies the conditions of the Interchange Lemma. This is always the case if $|A| = 1$.*

Kac, Manaster–Ramer and Rounds use constructions with `respectively` shown below, in which there is an equal number of nouns and verb phrases to be matched. In these constructions, the *n*th noun must agree in number with the *n*th verb phrase.

(2.165) `This land can be expected to sell itself/`
`*themselves.`
`These woods can be expected to sell *itself/`
`themselves.`

(2.166) `This land and these woods can be expected to rent`
`itself and sell themselves respectively.`

(2.167) `*This land and these woods can be expected to rent`
`themselves and sell itself respectively.`

(2.168) `This land and these woods and this land can be`
`expected to sell themselves and rent themselves`
`respectively.`

The problematic aspect of these constructions is illustrated by (2.168). There need not be an exact match of NPs and VPs, and when there is no match,

agreement becomes obscured (though it follows clear rules). Now let

(2.169) $$\begin{aligned} A := &(\texttt{this} \diamond \texttt{land} \cup \texttt{these} \diamond \texttt{woods}) \diamond \texttt{and} \\ &\diamond (\texttt{this} \diamond \texttt{land}^\frown \square \cup \texttt{these} \diamond \texttt{woods}^\frown \square)^+ \\ &^\frown\texttt{can} \diamond \texttt{be} \diamond \texttt{expected} \diamond \texttt{to} \\ &\diamond (\texttt{rent} \cup \texttt{sell}) \diamond (\texttt{itself}^\frown \square \cup \texttt{themselves}^\frown \square)^+ \\ &^\frown\texttt{and} \diamond (\texttt{rent} \cup \texttt{sell}) \diamond (\texttt{itself}^\frown \square \cup \texttt{themselves}^\frown \square)^+ \\ &^\frown\texttt{respectively}^\frown\texttt{.} \end{aligned}$$

and let D be the set of strings of A that contain as many nouns as they contain pronouns. B is that subset of D where the ith noun is `land` iff the ith pronoun is `itself`. The empirical fact about English is that the intersection of English with D is exactly B. Based on this we show that English is not context free. For suppose it were. Then we have a constant c_L satisfying the Interchange Lemma. (We ignore the blanks and the period from now on.) Let n be given. Choose $Q := B_n$, the set of strings of length n in B. Notice that $|B_n| \geq 2^{(n-8)/2}$ for all n. Therefore, for some n, $|B_n| > 2n^2 c_L$ so that $\ulcorner |B_n|/c_L n^2 \urcorner \geq 2$. This means that there are $\vec{x}_1$, $\vec{x}_2$, $\vec{z}_1$, $\vec{z}_2$ and $\vec{y}_1$ and $\vec{y}_2$ such that B_n contains $\vec{x}_1\vec{y}_1\vec{z}_1$ as well as $\vec{x}_2\vec{y}_2\vec{z}_2$, but $\vec{x}_1\vec{y}_2\vec{z}_1$ and $\vec{x}_2\vec{y}_1\vec{z}_2$ are also grammatical (and therefore even in B_n). It is easy to see that this cannot be.

The next example in our series is modelled after the proof of the non context freeness of ALGOL. It deals with a quite well known language, namely predicate logic. Predicate logic is defined as a language over a set of relation and function symbols of varying arity and a set of variables $\{\mathtt{x}_i : i \in \omega\}$. In order to be able to conceive of predicate logic as a language in our sense, we code the variables as consisting of sequences $\mathtt{x}\vec{\alpha}$, where $\vec{\alpha} \in \{\mathtt{0}, \mathtt{1}\}^*$. We have $\mathtt{x}\vec{\alpha} = \mathtt{x}\vec{\beta}$ iff $\vec{\alpha} = \vec{\beta}$. (Leading zeros are not suppressed. The numbers are usually put as subscripts, but we shall not do that here.) We restrict ourselves to the language of pure equality. The alphabet is $\{\forall, \exists, (,), =, \mathtt{x}, \mathtt{0}, \mathtt{1}, \wedge, \neg, \rightarrow\}$. The grammar rules are as follows.

(2.170) $$\begin{aligned} \mathtt{F} &\rightarrow \mathtt{Q(F)} \mid \neg\mathtt{(F)} \mid \mathtt{(F)}\wedge\mathtt{(F)} \mid \mathtt{(F)}\rightarrow\mathtt{(F)} \mid \mathtt{P} \\ \mathtt{P} &\rightarrow \mathtt{V{=}V} \\ \mathtt{Q} &\rightarrow (\forall\mathtt{V})\mathtt{F} \mid (\exists\mathtt{V})\mathtt{F} \\ \mathtt{V} &\rightarrow \mathtt{x} \mid \mathtt{xZ} \\ \mathtt{Z} &\rightarrow \mathtt{0Z} \mid \mathtt{1Z} \mid \mathtt{0} \mid \mathtt{1} \end{aligned}$$

Here F stands for the set of formulae P for the set of prime formulae Q for the set of quantifier prefixes, V the set of variables and E for the set of strings over 0 and 1. Let $\vec{x}$ be a formula and C an occurrence of a variable $\mathtt{x}\vec{\alpha}$. We now say that this occurrence of a variable is **bound** in $\vec{x}$ if it is an occurrence D of a formula $(Q\mathtt{x}\vec{\alpha})\vec{y}$ in $\vec{x}$ with $Q \in \{\forall, \exists\}$ which contains C. A formula is called a **sentence** if every occurrence of a variable is bound.

Theorem 2.113 *The set of sentences of predicate logic of pure equality is not context free.*

Proof. Let L be the set of sentences of pure equality of predicate logic. Assume this set is context free. Then by the Pumping Lemma there is a k such that every string of length $\geq k$ has a decomposition $\vec{u}\vec{x}\vec{v}\vec{y}\vec{w}$ such that $\vec{u}\vec{x}^i\vec{v}\vec{y}^i\vec{z} \in L$ for all i and $|\vec{x}\vec{v}\vec{y}| \leq k$. Define the following formulae.

(2.171) $\quad (\forall \mathtt{x}\vec{\alpha})(\mathtt{x}\vec{\alpha}=\mathtt{x}\vec{\alpha})$

All these formulae are sentences. If $\vec{\alpha}$ is sufficiently long (for example, longer than k) then there is a decomposition as given. Since $\vec{x}\vec{v}\vec{y}$ must have length $\leq k$ $\vec{x}$ and $\vec{y}$ cannot both be disjoint to all occurrences of $\vec{\alpha}$. On the other hand, it follows from this that $\vec{x}$ and $\vec{y}$ consist only of 0 and 1, and so necessarily they are disjoint to some occurrence of $\vec{\alpha}$. If one pumps up $\vec{x}$ and $\vec{y}$, necessarily one occurrence of a variable will end up being unbound. □

We can strengthen this result considerably.

Theorem 2.114 *The set of sentence of predicate logic of pure equality is not semilinear.*

Proof. Let P be the set of sentences of predicate logic of pure equality. Assume that P is semilinear. Then let P_1 be the set of sentences which contain only one occurrence of a quantifier, and let this quantifier be $\exists$. $\mu[P_1]$ is the intersection of $\mu[P]$ with the set of all vectors whose $\exists$–component is 1 and whose $\forall$–component is 0. This is then also semilinear. Now we consider the image of $\mu[P_1]$ under deletion of all symbols which are different from x, 0 and 1. The result is denoted by Q_1. Q_1 is semilinear. By construction of P_1 there is an $\vec{\alpha} \in \{0,1\}^*$ such that every occurrence of a variable is of the form $\mathtt{x}\vec{\alpha}$. If this variable occurs k times and if $\vec{\alpha}$ contains p occurrences of 0 and q occurrences of 1 we get as a result the vector $k\mathtt{x} + kp0 + kq1$. It is easy to see that k must be odd. For a variable occurs once in the quantifier and

elsewhere once to the left and once to the right of the equation sign. Now we have among others the following sentences.

$$
\begin{aligned}
&(\exists \mathtt{x}\vec{\alpha})(\mathtt{x}\vec{\alpha}=\mathtt{x}\vec{\alpha})\\
(2.172)\qquad &(\exists \mathtt{x}\vec{\alpha})((\mathtt{x}\vec{\alpha}=\mathtt{x}\vec{\alpha})\wedge(\mathtt{x}\vec{\alpha}=\mathtt{x}\vec{\alpha}))\\
&(\exists \mathtt{x}\vec{\alpha})((\mathtt{x}\vec{\alpha}=\mathtt{x}\vec{\alpha})\wedge((\mathtt{x}\vec{\alpha}=\mathtt{x}\vec{\alpha})\wedge(\mathtt{x}\vec{\alpha}=\mathtt{x}\vec{\alpha})))
\end{aligned}
$$

Since we may choose any sequence $\vec{\alpha}$ we have

$$(2.173)\qquad Q_1 = \{(2k+3)(\mathtt{x}+p\mathtt{0}+q\mathtt{1}) : k,p,q \in \omega\}$$

Q_1 is an infinite union of planes of the form $(2k+3)(\mathtt{x}+\omega\mathtt{0}+\omega\mathtt{1})$. We show: no finite union of linear planes equals Q_1. From this we automatically get a contradiction. So, assume that Q_1 is the union of U_i, $i < n$, U_i linear. Then there exists a U_i which contains infinitely many vectors of the form $(2k+3)\mathtt{x}$. From this one easily deduces that U_i contains a cyclic vector of the form $m\mathtt{x}$, $m > 0$. (This is left as an exercise.) However, it is clear that if $v \in Q_1$ then we have $m\mathtt{x}+v \notin Q_1$, and then we have a contradiction. □

Now we shall present an easy example of a 'natural' language which is not semilinear. It has been proposed in somewhat different form by Arnold Zwicky. Consider the number names of English. The stock of primitive names for numbers is finite. It contains the names for digits (`zero` up to `nine`) the names for the multiples of ten (`ten` until `ninety`), the numbers from `eleven` and `twelve` until `nineteen` as well as some names for the powers of ten: `hundred`, `thousand`, `million`, `billion`, and a few more. (Actually, using Latin numerals we can go to very high powers, but few people master these numerals, so they will hardly know more than these.) Assume without loss of generality that `million` is the largest of them. Then there is an additional recipe for naming higher powers, namely by stacking the word `million`. The number 10^{6k} is represented by the k–fold iteration of the word `million`. For example, the sequence

(2.174) `one million million million million`

names the number 10^{24}. (It is also called `octillion`, from Latin `octo` 'eight', because there are eight blocks of three zeros.) For arbitrary numbers the schema is as follows. A number in digital expansion is divided from right to left into blocks of six. So, it is divided as follows:

$$(2.175)\qquad \alpha_0 + \alpha_1 \times 10^6 + \alpha_2 \times 10^{12} \cdots$$

where $\alpha_i < 10^6$ for all i. The associated number name is then as follows.

$$\cdots \diamond \vec{\eta}_2 \diamond \texttt{million} \diamond \texttt{million} \diamond \vec{\eta}_1 \diamond \texttt{million} \diamond \vec{\eta}_0 \tag{2.176}$$

where $\vec{\eta}_i$ is the number name of α_i. If $\alpha_i = 0$ the ith block is omitted. Let Z be the set of number names. We define a function φ as follows. $\varphi(\texttt{million}) = \texttt{b}$; $\varphi(\Box) := \varepsilon$, all other primitive names are mapped onto $\texttt{a}$. The Parikh image of $\varphi[Z]$ is denoted by W. Now we have

$$W = \left\{ k_0\texttt{a} + k_1\texttt{b} : k_1 \geq \binom{\llcorner k_0/9 \lrcorner}{2} \right\} \tag{2.177}$$

Here, $\llcorner k \lrcorner$ is the largest integer $\leq k$. We have left the proof of this fact to the reader. We shall show that W is not semilinear. This shows that Z is also not semilinear. Suppose that W is semilinear, say $W = \bigcup_{i<n} N_i$ where all the N_i are linear. Let

$$N_i = u_i + \sum_{j<p_i} \omega v^i_j \tag{2.178}$$

for certain u_i and $v^i_j = \lambda^i_j\texttt{a} + \mu^i_j\texttt{b}$. Suppose further that for some i and j we have $\lambda^i_j \neq 0$. Consider the set

$$P := u_i + \omega v^i_j = \{u_i + k\lambda^i_j\texttt{a} + k\mu^i_j\texttt{b} : k \in \omega\} \tag{2.179}$$

Certainly we have $P \subseteq N_i \subseteq W$. Furthermore, we surely have $\mu^i_j \neq 0$. Now put $\zeta := \lambda^i_j/\mu^i_j$. Then

$$P = \{u_i + k\mu^i_j(\texttt{a} + \zeta\texttt{b}) : k \in \omega\} \tag{2.180}$$

Lemma 2.115 *For every $\varepsilon > 0$ almost all elements of P have the form $p\texttt{a} + q\texttt{b}$ where $q/p \leq \zeta + \varepsilon$.*

Proof. Let $u_i = x\texttt{a} + y\texttt{b}$. Then a general element of the set P is of the form $(x + k\lambda^i_j)\texttt{a} + (y + k\mu^i_j)\texttt{b}$. We have to show that for almost all k the inequality

$$\frac{x + k\lambda^i_j}{y + k\mu^i_j} \leq \varepsilon + \zeta \tag{2.181}$$

is satisfied. Indeed, if $k > \frac{x}{\mu^i_j\varepsilon}$, then

$$\frac{x + k\lambda^i_j}{y + k\mu^j_i} \leq \frac{x + k\lambda^i_j}{k\mu^i_j} = \zeta + \frac{x}{k\mu^i_j} < \zeta + \frac{x}{\mu^i_j x/\mu^i_j\varepsilon} = \zeta + \varepsilon \tag{2.182}$$

This holds for almost all k. $\Box$

Lemma 2.116 *Almost all points of P are outside of W.*

Proof. Let n_0 be chosen in such a way that $\binom{\lfloor n_0/9 \rfloor}{2} > n_0(\zeta+1)$. Then for all $n \geq n_0$ we also have $\binom{\lfloor n/9 \rfloor}{2} > n(\zeta+1)$. Let $p\mathtt{a}+q\mathtt{b} \in W$ with $p \geq n_0$. Then we have $\frac{q}{p} > \zeta+\varepsilon$, and therefore $p\mathtt{a}+q\mathtt{b} \notin P$. Put $H := \{p\mathtt{a}+q\mathtt{b} : p \geq n_0\}$. Then $P \cap H = \varnothing$. However $W \cap -H$ is certainly finite. Hence $W \cap P$ is finite, as required. □

Now have the desired contradiction. For on the one hand no vector is a multiple of $\mathtt{a}$; on the other hand there can be no vector $m\mathtt{a}+n\mathtt{b}$ with $n \neq 0$. Hence W is not semilinear.

Notes on this section. The question concerning the complexity of variable binding is discussed in [**?**]. It is shown there that the language of sentences of predicate logic is not context free (a result that was 'folklore') but that it is at least an indexed language. (Indexed languages neeed not be semilinear.) On the other hand, it has been conjectured that if we take V to the set of formulae in which every quantifier binds at least one free occurrence of a variable, the language V is not even an indexed language. See also Section 5.6. Philip Miller [**?**] argues that Swedish and Norwegian are not context free, and if right branching analyses are assumed, they are not even indexed languages.

Exercise 83. Formalize the language of functions and integral expressions. Prove that the language of proper integral expressions is not context free.

Exercise 84. Show the following: *Let U be a linear set which contains infinitely many vectors of the form $k\mathtt{a}$. Then there exists a cyclic vector of the form $m\mathtt{a}$, $m > 0$. Hint.* Notice that the alphabet may consist of more than one letter.

Exercise 85. Show that W has the claimed form.

Exercise 86. Show that the set V is not semilinear.

$$V := \left\{ k_0\mathtt{a}+k_1\mathtt{b} : k_1 \leq \binom{k_0}{2} \right\} \tag{2.183}$$

Hint. Evidently, no linear set $\subseteq V$ may contain a vector $k\mathtt{b}$. Therefore the following is well–defined.

$$\gamma := \max\left\{ \frac{\mu_j^i}{\lambda_j^i} : i < n, j < p_i \right\} \tag{2.184}$$

Show now that for every $\varepsilon > 0$ almost all elements of W are of the form $x\mathtt{a}+y\mathtt{b}$ where $y \leq (\gamma+\varepsilon)x$. If we put for example $\varepsilon = 1$ we now get a con-

tradiction.

Exercise 87. Prove the unique readability of predicate logic. *Hint.* Since we have strictly speaking not defined terms, restrict yourself to proving that the grammar given above is unambiguous. You might try to show that it is also transparent.

Exercise 88. Let $\Omega \subseteq \omega$. Put $L_\Omega := \{\mathtt{a}^m\mathtt{b}^n : m \neq n \text{ or } m \in \Omega\}$. Then $\pi[L_\Omega] = V_\Omega$, as defined in Exercise 76. Show that L_Ω satisfies the properties of Theorem 1.82 and of Theorem 2.111. It follows that there are $2^{\aleph_0}$ many languages over a and b that satisfy these criteria for context freeness and are not even semilinear.

Chapter 3
Categorial Grammar and Formal Semantics

1. Languages as Systems of Signs

Languages are certainly not sets of strings. They are systems for communication. This means in particular that the strings have meaning, a meaning which all speakers of the language more or less understand. And since natural languages have potentially infinitely many strings, there must be a way to find out what meaning a given string has on the basis of finite information. An important principle in connection with this is the so–called *Principle of Compositionality*. It says in simple words that the meaning of a string only depends on its derivation. For a CFG this means: if $\rho = \beta \to \alpha_0\alpha_1 \cdots \alpha_{n-1}$ is a rule and $\vec{u}_i$ a string of category α_i then $\vec{v} := \vec{u}_0\vec{u}_1 \cdots \vec{u}_{n-1}$ is a string of category β and the meaning of $\vec{v}$ depends only on the meaning of the $\vec{u}_i$ and ρ. In this form the principle of compositionality is still rather vague, and we shall refine and precisify it in the course of this section. However, for now we shall remain with this definition. It appears that we have admitted only context free rules. This is a restriction, as we know. We shall see later how we can get rid of it.

To begin, we shall assume that meanings come from some set M, which shall not be specified further. As before, exponents are members of A^*, where A is a finite alphabet. (Alternatives to this assumption will be discussed later.)

Definition 3.1 *An* ***interpreted (string) language over the alphabet*** A *and* ***with meanings in*** M *is a relation* $\mathcal{I} \subseteq A^* \times M$. *The* ***string language associated with*** $\mathcal{I}$ *is*

$$L(\mathcal{I}) := \{\vec{x} : \textit{ there is } m \in M \textit{ such that } \langle \vec{x}, m\rangle \in \mathcal{I}\} \tag{3.1}$$

The meanings expressed by $\mathcal{I}$ *are*

$$M(\mathcal{I}) := \{m : \textit{ there is } \vec{x} \in A^* \textit{ such that } \langle \vec{x}, m\rangle \in \mathcal{I}\} \tag{3.2}$$

Alternatively, we may regard a language as a function from A^* to $\wp(M)$. Then $L(f) := \{\vec{x} : f(\vec{x}) \neq \varnothing\}$ is the string language associated with f and $M(f) := \bigcup_{\vec{x} \in A^*} f(\vec{x})$ the set of expressed meanings of f. These definitions are

not equivalent when it comes to compositionality. In the original definition, any particular meaning of a composite expression is derived from some particular meanings of its parts, in the second the totality of meanings is derived from the totality of the meanings of the parts.

We give an example. We consider the number terms as known from everyday life as for example `((3+5)*2)`. We shall write a grammar with which we can compute the value of a term as soon as its analysis is known. This means that we regard an interpreted language as a set of pairs $\langle t, x\rangle$ where t is an arithmetical term and x its value. Of course, the analysis does not directly reveal the value but we must in addition to the rules of the grammar specify in which way the value of the term is computed inductively over the analysis. Since the nodes correspond to the subterms this is straightforward. Let T be the following grammar.

$$
\begin{array}{lcl}
\texttt{T} & \rightarrow & \texttt{(T+T)} \mid \texttt{(T-T)} \mid \texttt{(T*T)} \mid \texttt{(T/T)} \\
\texttt{T} & \rightarrow & \texttt{Z} \mid \texttt{(-Z)} \\
\texttt{Z} & \rightarrow & \texttt{0} \mid \texttt{1} \mid \texttt{2} \mid \cdots \mid \texttt{9}
\end{array}
\tag{3.3}
$$

(This grammar only generates terms which have ciphers in place of decimal strings. But see Section 3.4.) Let now an arbitrary term be given. To this term corresponds a unique number (if for a moment we disregard division by 0). This number can indeed be determined by induction over the term. To this end we define a partial interpretation map I, which if defined assigns a number to a given term.

$$
\begin{array}{lcl}
I(\texttt{(}\vec{x}\texttt{+}\vec{y}\texttt{)}) & := & I(\vec{x}) + I(\vec{y}) \\
I(\texttt{(}\vec{x}\texttt{-}\vec{y}\texttt{)}) & := & I(\vec{x}) - I(\vec{y}) \\
I(\texttt{(}\vec{x}\texttt{*}\vec{y}\texttt{)}) & := & I(\vec{x}) \times I(\vec{y}) \\
I(\texttt{(}\vec{x}\texttt{/}\vec{y}\texttt{)}) & := & I(\vec{x}) \div I(\vec{y}) \\
I(\texttt{(-}\vec{x}\texttt{)}) & := & -I(\vec{x}) \\
I(\texttt{0}) & := & 0 \\
I(\texttt{1}) & := & 1 \\
 & \cdots & \\
I(\texttt{9}) & := & 9
\end{array}
\tag{3.4}
$$

If a function f is undefined on x we write $f(x) = \star$. We may also regard $\star$ as a value. The rules for $\star$ are then as follows. If at least one argument is $\star$, so is the value. Additionally, $a/0 = \star$ for all a. If $\vec{x}$ is a term, then $I(\vec{x})$ is uniquely defined. For either $\vec{x}$ is a cipher from `0` to `9` or it is a negative cipher,

or $\vec{x} = (\vec{y}_1 \odot \vec{y}_2)$ for some uniquely determined $\vec{y}_1, \vec{y}_2$ and $\odot \in \{\texttt{+}, \texttt{-}, \texttt{*}, \texttt{/}\}$. In this way one can calculate $I(\vec{x})$ if one knows $I(\vec{y}_1)$ and $I(\vec{y}_2)$. The value of a term can be found by naming a derivation and then computing the value of each of its subterms. Notice that the grammar is transparent so that only one syntactical analysis can exist for each string.

The method just described has a disadvantage: the interpretation of a term is in general not unique, for example if a string is ambiguous. (For example, if we erase all brackets then the term `3+5*2` has two values, 13 or 16.) As explained above, we could take the meaning of a string to be a set of numbers. If the language is unambiguous this set has at most one member. Further, we have $I(\vec{x}) \neq \varnothing$ only if $\vec{x}$ is a constituent. However, in general we wish to avoid taking this step. Different meanings should arise only from different analyses. There is a way to implement this idea no matter what the grammar is. Let U be the grammar which results from T by deleting the brackets of T.

$$
\begin{array}{lcl}
\texttt{T} & \to & \texttt{T+T} \mid \texttt{T-T} \mid \texttt{T*T} \mid \texttt{T/T} \\
\texttt{T} & \to & \texttt{Z} \mid \texttt{-Z} \\
\texttt{Z} & \to & \texttt{0} \mid \texttt{1} \mid \texttt{2} \mid \cdots \mid \texttt{9}
\end{array}
\tag{3.5}
$$

The strings of U can be viewed as images of a canonical transparent grammar. This could be (3.3). However, for some reason that will become clear we shall choose a different grammar. Intuitively, we think of the string as the image of a term which codes the derivation tree. This tree differs from the structure tree in that the intermediate symbols are not nonterminals but symbols for rules. The derivation tree is coded by term in Polish Notation. For each rule ρ we add a new symbol $\texttt{R}_\rho$. In place of the rule $\rho = A \to \vec{\alpha}$ we now take the rule $A \to \texttt{R}_\rho \vec{\alpha}$. This grammar, call it V, is transparent (see Exercise 89). $\vec{x} \in L(V)$ is called a **derivation term**. We define two maps ζ and ι. ζ yields a string for each derivation term, and ι yields an interpretation. Both maps shall be homomorphisms from the term algebra, though the concrete definition is defined over strings. ζ can be uniformly defined by deleting the symbols $\texttt{R}_\rho$. However, notice that the rules below yield values only if the strings are derivation terms.

$$
\begin{aligned}
\zeta(\texttt{R}_\rho \vec{\alpha}_0 \cdots \vec{\alpha}_{n-1}) &:= \zeta(\alpha_0)^\frown \zeta(\alpha_1)^\frown \cdots {}^\frown \zeta(\alpha_{n-1}) \\
\zeta(\alpha) &:= \alpha
\end{aligned}
\tag{3.6}
$$

In the last line, α is different from all $\texttt{R}_\rho$. We have assumed here that the grammar has no rules of the form $A \to \varepsilon$ even though a simple adaptation can

help here as well. Now on to the definition of ι. In the case at hand this is without problems.

$$
\begin{array}{lcl}
\iota(\mathtt{R}_{+}\vec{\alpha}_0\mathtt{+}\vec{\alpha}_1) & := & \iota(\vec{\alpha}_0)+\iota(\vec{\alpha}_1) \\
\iota(\mathtt{R}_{-2}\vec{\alpha}_0\mathtt{-}\vec{\alpha}_1) & := & \iota(\vec{\alpha}_0)-\iota(\vec{\alpha}_1) \\
\iota(\mathtt{R}_{*}\vec{\alpha}_0\mathtt{*}\vec{\alpha}_1) & := & \iota(\vec{\alpha}_0)\times\iota(\vec{\alpha}_1) \\
\iota(\mathtt{R}_{/}\vec{\alpha}_0\mathtt{/}\vec{\alpha}_1) & := & \iota(\vec{\alpha}_0)\div\iota(\vec{\alpha}_1) \\
\iota(\mathtt{R}_{-1}\mathtt{-}\vec{\alpha}) & := & -\iota(\vec{\alpha})
\end{array}
\tag{3.7}
$$

Here we have put the derivation term into Polish Notation, since it is uniquely readable. However, this only holds under the condition that every symbol is unique. Notice, namely, that some symbols can have different meanings — as in our example the minus symbol. To this end we have added an additional annotation of the symbols. Using a superscript we have distinguished between the unary minus and the binary one. Since the actual language does not do so (we write '$\mathtt{-}$' without distinction), we have written $\mathtt{R}_{-1}$ if the rule for the unary symbol has been used, and $\mathtt{R}_{-2}$ if the one for the binary symbol has been used.

The mapping ι is a homomorphism of the algebra of derivation terms into the algebra of real numbers with $\star$, which is equivalent to a partial homomorphism from the algebra of terms to the algebra of real numbers. For example the symbol $\mathtt{R}_{+}$ is interpreted by the function $+\colon \mathbb{R}_\star\times\mathbb{R}_\star\to\mathbb{R}_\star$, where $\mathbb{R}_\star := \mathbb{R}\cup\{\star\}$ and $\star$ satisfies the laws specified above. In principle this algebra can be replaced by any other which allows to interpret unary and binary function symbols. We emphasize that it is not necessary that the interpreting functions are basic functions of the algebras. It is enough if they are polynomial functions (see [**?**] on this point). For example, we can introduce a unary function symbol $\mathtt{d}$ whose interpretation is duplication. Now $2x = x+x$, and hence the duplication is a polynomial function of the algebra $\langle\mathbb{R},+,\cdot,0,1\rangle$, but not basic. However, the formal setup is easier if we interpret each function symbol by a basic function. (It can always be added, if need be.)

This exposition motivates a terminology which sees meanings and strings as images of abstract signs under a homomorphism. We shall now develop this idea in full generality. The basis is formed by an algebra of signs. Recall from Section 1.1 the notion of a strong (partial) subalgebra. A strong subalgebra is determined by the set B. The functions on B are the restrictions of the respective functions on A. Notice that it is not allowed to partialize functions additionally. For example, $\langle A,\Xi\rangle$ with $\Xi(f)=\varnothing$ is not a strong subalgebra of $\mathfrak{A}$ unless $\Pi(f)=\varnothing$.

A **sign** is a triple $\sigma = \langle e, c, m \rangle$ where e is the exponent of σ, usually some kind of string over an alphabet A, c the category of σ and m its meaning. Abstractly, however, we shall set this up differently. We shall first define an algebra of signs as such, and introduce exponent, category and meaning as values of the signs under some homomorphisms. This will practically amount to the same, however. So, we start by fixing a signature $\langle F, \Omega \rangle$. In this connection the function symbols from F are called **modes**. Over this signature we shall define an algebra of signs, of exponents, of categories and meanings. An algebra of signs over $\langle F, \Omega \rangle$ is simply a 0–generated partial algebra $\mathfrak{A}$ over this signature together with certain homomorphisms, which will be defined later.

Definition 3.2 *A (partial) Ω–algebra $\mathfrak{A} = \langle A, \Pi \rangle$ is called n–**generated** if there is an n–element subset $X \subseteq A$ such that the smallest strong subalgebra containing X is $\mathfrak{A}$.*

Definition 3.3 *The quadruple $\langle \mathfrak{A}, \varepsilon, \gamma, \mu \rangle$ is called a **sign grammar over the signature** Ω if $\mathfrak{A}$ is a 0–generated partial Ω–algebra and $\varepsilon : \mathfrak{A} \to \mathfrak{E}$, $\gamma : \mathfrak{A} \to \mathfrak{C}$ and $\mu : \mathfrak{A} \to \mathfrak{M}$ homomorphisms to certain partial Ω–algebras such that the homomorphism $\langle \varepsilon, \gamma, \mu \rangle$ is injective and strong. $\mathfrak{A}$ is called the **algebra of signs**, $\mathfrak{E}$ the **algebra of exponents**, $\mathfrak{C}$ the **algebra of categories** and $\mathfrak{M}$ the **algebra of meanings**.*

This means in particular:

- ☞ Every sign σ is uniquely characterized by three things:
  - ∗ its so–called **exponent** $\varepsilon(\sigma)$,
  - ∗ its (**syntactical**) **category** $\gamma(\sigma)$ (which is also often called its **type**),
  - ∗ its meaning $\mu(\sigma)$.
- ☞ To every function symbol $f \in F$ corresponds an $\Omega(f)$–ary function $f^{\mathfrak{E}}$ in $\mathfrak{E}$, an $\Omega(f)$–ary function $f^{\mathfrak{C}}$ in $\mathfrak{C}$ and an $\Omega(f)$–ary function $f^{\mathfrak{M}}$ in $\mathfrak{M}$.
- ☞ Signs can be combined with the help of the function $f^{\mathfrak{A}}$ any time their respective exponents can be combined with the help of $f^{\mathfrak{E}}$, their respective categories can be combined with $f^{\mathfrak{C}}$ and their respective meanings with $f^{\mathfrak{M}}$. (This corresponds to the condition of strongness.)

In the sequel we shall write f^{ε} in place of $f^{\mathfrak{E}}$, f^{γ} in place of $f^{\mathfrak{C}}$ and f^{μ} in place of $f^{\mathfrak{M}}$. This will allow us to suppress mentioning which actual algebras are chosen. If σ is a sign, then $\langle \varepsilon(\sigma), \gamma(\sigma), \mu(\sigma) \rangle$ is uniquely defined by σ, and on the other hand it uniquely defines σ as well. We shall call this triple the **realization** of σ. Additionally, we can represent σ by a term in the free Ω–algebra. We shall now deal with the correspondences between these viewpoints.

Let $\mathfrak{Tm}_{\Omega} := \langle \mathrm{PN}_{\Omega}, \{g^{\mathfrak{Tm}_{\Omega}} : g \in F\} \rangle$, where PN_{Ω} is the set of constant Ω–terms written in Polish Notation and

$$g^{\mathfrak{Tm}_{\Omega}}(\vec{x}_0, \ldots, \vec{x}_{\Omega(g)-1}) := g^{\frown} \prod_{i<\Omega(g)} \vec{x}_i \tag{3.8}$$

$\mathfrak{Tm}_{\Omega}$ is a freely 0–generated Ω–algebra. The elements of PN_{Ω} are called **structure terms**. We use $\mathfrak{s}$, $\mathfrak{t}$, $\mathfrak{u}$ and so on as metavariables for structure terms. We give an example. Suppose that $\mathtt{N}$ is a 0–ary mode and $\mathtt{S}$ a unary mode. Then we have $\mathtt{N}^{\mathfrak{Tm}_{\Omega}} = \mathtt{N}$ and $\mathtt{S}^{\mathfrak{Tm}_{\Omega}} : \vec{x} \mapsto \mathtt{S}^{\frown}\vec{x}$. This yields the following strings as representatives of structure terms.

$$\mathtt{N}, \mathtt{SN}, \mathtt{SSN}, \mathtt{SSSN}, \ldots \tag{3.9}$$

We denote by $h\colon M \xrightarrow{p} N$ the fact that h is a partial function from M to N. We now define partial maps $\dot{\varepsilon}\colon \mathrm{PN}_{\Omega} \xrightarrow{p} E$, $\dot{\gamma}\colon \mathrm{PN}_{\Omega} \xrightarrow{p} C$ and $\dot{\mu}\colon \mathrm{PN}_{\Omega} \xrightarrow{p} M$ in the following way.

$$\dot{\varepsilon}(g^{\mathfrak{Tm}_{\Omega}}(\mathfrak{s}_0, \ldots, \mathfrak{s}_{\Omega(g)-1})) := g^{\varepsilon}(\dot{\varepsilon}(\mathfrak{s}_0), \ldots, \dot{\varepsilon}(\mathfrak{s}_{\Omega(g)-1})) \tag{3.10}$$

Here, the left hand side is defined iff the right hand side is and then the two are equal. If we have a 0–ary mode g, then it is a structure term $\dot{\varepsilon}(g) = g^{\varepsilon} \in E$. Likewise we define the other maps.

$$\dot{\gamma}(g^{\mathfrak{Tm}_{\Omega}}(\mathfrak{s}_0, \ldots, \mathfrak{s}_{\Omega(g)-1})) := g^{\gamma}(\dot{\gamma}(\mathfrak{s}_0), \ldots, \dot{\gamma}(\mathfrak{s}_{\Omega(g)-1})) \tag{3.11}$$

$$\dot{\mu}(g^{\mathfrak{Tm}_{\Omega}}(\mathfrak{s}_0, \ldots, \mathfrak{s}_{\Omega(g)-1})) := g^{\mu}(\dot{\mu}(\mathfrak{s}_0), \ldots, \dot{\mu}(\mathfrak{s}_{\Omega(g)-1})) \tag{3.12}$$

As remarked above, for every sign there is a structure term. The converse need not hold.

Definition 3.4 *We say, a structure term $\mathfrak{s}$ is **orthographically definite** if $\dot{\varepsilon}(\mathfrak{s})$ is defined. $\mathfrak{s}$ is **syntactically definite** if $\dot{\gamma}(\mathfrak{s})$ is defined and **semantically definite** if $\dot{\mu}(\mathfrak{s})$ is defined. Finally, $\mathfrak{s}$ is **definite** if $\mathfrak{s}$ is orthographically, syntactically as well as semantically definite.*

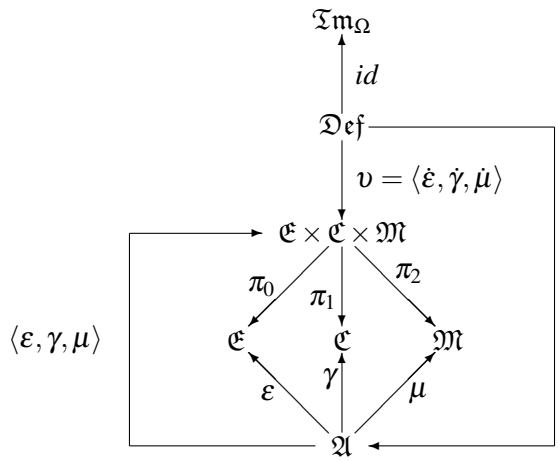


Figure 9. Synopsis

Definition 3.5 *The partial map* $\upsilon := \langle \dot{\varepsilon}, \dot{\gamma}, \dot{\mu} \rangle$ *is called the* ***unfolding map****.*

The reader is referred to Figure 9 for a synopsis of the various algebras and maps between them. In the sequel we shall often identify the structure term $\mathfrak{s}$ with its image under the unfolding map. This will result in rather strange types of definitions, where on the left we find a string (which *is* the structure term, by convention) and on the right a triple. This abuse of the language shall hopefully present no difficulty. $\mathfrak{A}$ is isomorphic to the partial algebra of all $\langle \dot{\varepsilon}(\mathfrak{s}), \dot{\gamma}(\mathfrak{s}), \dot{\mu}(\mathfrak{s}) \rangle$, where $\mathfrak{s}$ is a definite structure term. This we can also look at differently. Let D be the set of definite structure terms. This set becomes a partial Ω–algebra together with the partial functions $g^{\mathfrak{Tm}_\Omega} \restriction D$. We denote this algebra by $\mathfrak{Def}$. $\mathfrak{Def}$ is usually not a strong subalgebra of $\mathfrak{Tm}_\Omega$. For let $j\colon \mathfrak{s} \mapsto \mathfrak{s}$ be the identity map. Then we have $j(g^{\mathfrak{Def}}(\mathfrak{s}_0, \ldots, \mathfrak{s}_{\Omega(g)-1})) = g^{\mathfrak{Tm}_\Omega}(j(\mathfrak{s}_0), \ldots, j(\mathfrak{s}_{\Omega(g)-1}))$. The right hand side is always defined, the left hand side need not be.

The homomorphism $\upsilon \restriction D$ (which we also denote by υ) is however strong. Now look at the relation $\Theta := \{\langle \mathfrak{s}_0, \mathfrak{s}_1 \rangle : \upsilon(\mathfrak{s}_0) = \upsilon(\mathfrak{s}_1)\}$. Θ is a congruence on $\mathfrak{Def}$; for it clearly is an equivalence relation and if $\mathfrak{s}_i \,\Theta\, \mathfrak{u}_i$ for all $i < \Omega(f)$

then $f(\vec{\mathfrak{s}})$ is defined iff $f(\vec{\mathfrak{u}})$ is. And in this case we have $f(\vec{\mathfrak{s}})\ \Theta\ f(\vec{\mathfrak{u}})$. We can now put:

$$f^{\mathfrak{A}}(\langle[\mathfrak{s}_i]\Theta : i < \Omega(f)\rangle) := [f(\langle\mathfrak{s}_i : i < \Omega(f)\rangle)]\Theta \tag{3.13}$$

This is well–defined and we get an algebra, the algebra $\mathfrak{Def}/\Theta$. The following is easy to see.

Proposition 3.6 $\mathfrak{A} \cong \mathfrak{Def}/\Theta$.

So, $\mathfrak{Def}/\Theta$ is isomorphic to the algebra of signs. For every sign there is a structure term, but there might also be several. As an instructive example we look at the sign system of triples of the form $\langle$ $, T, 285\rangle$, where is the arrangement of hands of an ordinary clock (here showing 4:45), T a fixed letter, and 285 the number of minutes past midnight/noon that is symbolized by this arrangement. So, the above triple is a sign of the language, while $\langle$ $, T, 177\rangle$ is not, since the hands show 3:10, which equals 190 minutes, not 177. We propose two modes: $\mathtt{N}$ (the zero, 0–ary) and $\mathtt{S}$ (the successor function, unary). So, the unfolding of $\mathtt{N}$ is $\langle$ $, T, 0\rangle$, and the unfolding of $\mathtt{S}$ is the advancement by one minute. Then $\upsilon(\mathtt{S})$ is a total function, and we have

$$\upsilon(\mathtt{N}) = \upsilon(\mathtt{S}^{720}\mathtt{N}) \tag{3.14}$$

From this one easily gets that for every structure term $\mathfrak{s}$, $\upsilon(\mathfrak{s}) = \upsilon(\mathtt{S}^{720}\mathfrak{s})$. Hence every sign has infinitely many structure terms, and so is inherently structurally ambiguous. If instead we take as meanings the natural numbers (say, the minutes that elapsed since some fixed reference point) and $\mathtt{N}^{\mu} := 0$ as well as $\mathtt{S}^{\mu} := \lambda n.n + 1$ then every structure term represents a different sign! However, still there are only 720 exponents. Only that every exponent has infinitely many meanings.

We shall illustrate the concepts of a sign grammar by proceeding with our initial example. Our alphabet is now

$$R := \{\mathtt{0}, \mathtt{1}, \ldots, \mathtt{9}, \mathtt{+}, \mathtt{-}, \mathtt{*}, \mathtt{/}, \mathtt{(}, \mathtt{)}\} \tag{3.15}$$

The algebra $\mathfrak{E}$ consists of R^* together with some functions that we still have to determine. We shall now begin to determine the modes. They are $\mathtt{R}_{\mathtt{+}}$, $\mathtt{R}_{\mathtt{-}^2}$, $\mathtt{R}_{\mathtt{*}}$, $\mathtt{R}_{\mathtt{/}}$, which are binary, $\mathtt{R}_{\mathtt{-}^1}$, $\mathtt{V}$, which are unary, and — finally — ten 0–ary modes, namely $\mathtt{Z}_{\mathtt{0}}, \mathtt{Z}_{\mathtt{1}}, \ldots, \mathtt{Z}_{\mathtt{9}}$.

We begin with the 0–ary modes. These are, by definition, signs. For their identification we only need to know the three components. For example, to

the mode $\mathtt{Z}_0$ corresponds the triple $\langle \mathtt{0}, \mathtt{Z}, 0 \rangle$. This means: the exponent of the sign $\mathtt{Z}_0$ (what we get to see) is the digit $\mathtt{0}$; its category is $\mathtt{Z}$, and its meaning the number 0. Likewise with the other 0–ary modes. Now on to the unary modes. These are operations taking signs to make new signs. We begin with $\mathtt{R}_{-1}$. On the level of strings we get the polynomial $\mathtt{R}^{\varepsilon}_{-1}$, which is defined as follows.

$$\mathtt{R}^{\varepsilon}_{-1}(\vec{x}) := (\mathtt{-}\vec{x}) \tag{3.16}$$

On the level of categories we get the function

$$\mathtt{R}^{\gamma}_{-1}(c) := \begin{cases} \mathtt{T} & \text{if } c = \mathtt{Z}, \\ \star & \text{otherwise.} \end{cases} \tag{3.17}$$

Here $\star$ is again the symbol for the fact that the function is not defined. Finally we have to define $\mathtt{R}^{\mu}_{-1}$. We put

$$\mathtt{R}^{\mu}_{-1}(x) := -x \tag{3.18}$$

Notice that even if the function $x \mapsto -x$ is iterable, the mode $\mathtt{R}_{-1}$ is not. This is made impossible by the categorial assignment. This is an artefact of the example. We could have set things up differently. The mode $\mathtt{V}$ finally is defined by the following functions. $\mathtt{V}^{\varepsilon}(\vec{x}) := \vec{x}$, $\mathtt{V}^{\mu}(x) := x$ and $\mathtt{V}^{\gamma}(c) := \mathtt{R}_{-1}(c)$. Finally we turn to the binary modes. Let us look at $\mathtt{R}_{/}$. $\mathtt{R}^{\mu}_{*}$ is the partial (!) binary function $\div$ on $\mathbb{R}$. Further, we put

$$\mathtt{R}^{\varepsilon}_{/}(\vec{x}, \vec{y}) := (\vec{x}/\vec{y}) \tag{3.19}$$

as well as

$$\mathtt{R}^{\gamma}_{/}(c, d) := \begin{cases} \mathtt{T} & \text{if } c = d = \mathtt{T}, \\ \star & \text{otherwise.} \end{cases} \tag{3.20}$$

The string $\mathtt{R}_{*}\mathtt{R}_{+}\mathtt{Z}_3\mathtt{Z}_5\mathtt{Z}_7$ defines — as is easily computed — a sign whose exponent is $\mathtt{((3+5)*7)}$. By contrast, $\mathtt{R}_{/}\mathtt{Z}_2\mathtt{Z}_0$ does *not* represent a sign. It is syntactically definite but not semantically, since we may not divide by 0.

Definition 3.7 *A* ***linear system of signs*** *over the* ***alphabet*** *A, the set of* ***categories*** *C and the set of* ***meanings*** *M is a set $\Sigma \subseteq A^* \times C \times M$. Further, let $\mathtt{S}$ be a category. Then the interpreted language of Σ with respect to this category $\mathtt{S}$ is defined by*

$$\mathtt{S}(\Sigma) := \{\langle \vec{x}, m \rangle : \langle \vec{x}, \mathtt{S}, m \rangle \in \Sigma\} \tag{3.21}$$

We added the qualifying phrase 'linear' to distinguish this from sign systems which do not generally take strings as exponents. (For example, pictograms are nonlinear.)

A system of signs is simply a set of signs. The question is whether one can define an algebra over it. This is always possible. Just take a 0–ary mode for every sign. Since this is certainly not as intended, we shall restrict the possibilities as follows.

Definition 3.8 *Let $\Sigma \subseteq E \times C \times M$ be a system of signs. We say that Σ is* ***compositional*** *if there is a finite signature Ω and partial Ω–algebras $\mathfrak{E} = \langle E, \{f^{\mathfrak{E}} : f \in F\}\rangle$, $\mathfrak{C} = \langle C, \{f^{\mathfrak{C}} : f \in F\}\rangle$, $\mathfrak{M} = \langle M, \{f^{\mathfrak{M}} : f \in F\}\rangle$ such that all functions are computable and Σ is the carrier set of the 0–generated partial (strong) subalgebra of signs from $\mathfrak{E} \times \mathfrak{C} \times \mathfrak{M}$. Σ is* ***weakly compositional*** *if there is a compositional system Σ' such that $\Sigma = \Sigma' \cap E \times C \times M$.*

Notice that $\Sigma' \subseteq E' \times C' \times M'$ for certain sets E', C' and M'. We remark that a partial function $f\colon M^n \xrightarrow{p} M$ in the sense of the definition above is a computable total function $f^\star\colon M^n_\star \to M_\star$ such that $f^\star \restriction M^n = f$. So, the computation always halts, and we are told at its end whether or not the function is defined and if so what the value is.

Two conditions have been made: the signature has to be finite and the functions on the algebras computable. We shall show that however strong they appear, they do not really restrict the class of sign systems in comparison to weak compositionality.

We start by drawing some immediate conclusions from the definitions. If σ is a sign we say that $\langle \varepsilon(\sigma), \gamma(\sigma), \mu(\sigma)\rangle$ (no dots!) is its **realization**. We have introduced the unfolding map υ above.

Proposition 3.9 *Let $\langle \mathfrak{A}, \varepsilon, \gamma, \mu\rangle$ be a compositional sign grammar. Then the unfolding map is computable.*

Simply note that the unfolding of a structure term can be computed inductively. This has the following immediate consequence.

Corollary 3.10 *Let Σ be compositional. Then Σ is recursively enumerable.*

This is remarkable inasmuch as the set of all signs over $E \times C \times M$ need not even be enumerable. For typically M contains uncountably many elements (which can of course not all be named by a sign)!

Theorem 3.11 *A system of signs is weakly compositional iff it is recursively enumerable.*

Proof. Let $\Sigma \subseteq E \times C \times M$ be given. If Σ is weakly compositional, it also is recursively enumerable. Now, let us assume that Σ is recursively enumerable, say $\Sigma = \{\langle e_i, c_i, m_i\rangle : 0 < i \in \omega\}$. (Notice that we start counting with 1.) Now let $\mathtt{V}$ be a symbol and $\Delta := \{\langle \mathtt{V}^n, \mathtt{V}^n, \mathtt{V}^n\rangle : n \in \omega\}$ a system of signs. By properly choosing $\mathtt{V}$ we can see to it that $\Delta \cap \Sigma = \varnothing$ and that no $\mathtt{V}^n$ occurs in E, C or M. Let $F := \{\mathtt{Z}_0, \mathtt{Z}_1, \mathtt{Z}_2\}$, $\Omega(\mathtt{Z}_0) := 0$, $\Omega(\mathtt{Z}_1) := 1$ and $\Omega(\mathtt{Z}_2) := 1$.

$$
\begin{aligned}
\mathtt{Z}_0 &:= \langle \mathtt{V}, \mathtt{V}, \mathtt{V}\rangle, \\
\mathtt{Z}_1(\sigma) &:= \begin{cases} \langle \mathtt{V}^{i+1}, \mathtt{V}^{i+1}, \mathtt{V}^{i+1}\rangle & \text{if } \sigma = \langle \mathtt{V}^i, \mathtt{V}^i, \mathtt{V}^i\rangle, \\ \star & \text{otherwise,} \end{cases} \\
\mathtt{Z}_2(\sigma) &:= \begin{cases} \langle e_i, c_i, m_i\rangle & \text{if } \sigma = \langle \mathtt{V}^i, \mathtt{V}^i, \mathtt{V}^i\rangle, \\ \star & \text{otherwise.} \end{cases}
\end{aligned} \tag{3.22}
$$

This is well–defined. Further, the functions are all computable. For example, the map $\mathtt{V}^i \mapsto e_i$ is computable since it is the concatenation of the computable functions $\mathtt{V}^i \mapsto i$, $i \mapsto \langle e_i, c_i, m_i\rangle$ with $\langle e_i, c_i, m_i\rangle \mapsto e_i$. We claim: the system of signs generated is exactly $\Delta \cup \Sigma$. For this we notice first that a structure term is definite iff it has the following form. (a) $t = \mathtt{Z}_1^i \mathtt{Z}_0$, or (b) $t = \mathtt{Z}_2 \mathtt{Z}_1^i \mathtt{Z}_0$. In Case (a) we get the sign $\langle \mathtt{V}^{i+1}, \mathtt{V}^{i+1}, \mathtt{V}^{i+1}\rangle$, in Case (b) the sign $\langle e_{i+1}, c_{i+1}, m_{i+1}\rangle$. Hence we generate exactly $\Delta \cup \Sigma$. So, Σ is weakly compositional. □

Notice that the algebra of exponents uses additional symbols which are only used to create new objects which are like natural numbers. The just presented algebra is certainly not very satisfying. (It is also not compositional.) Hence one has sought to provide a more systematic theory of categories and their meanings. A first step in this direction are the categorial grammars. To motivate them we shall give a construction for CFGs that differs markedly from the one in Theorem 3.11. The starting point is once again an interpreted language $\mathcal{I} = \{\langle \vec{x}, f(\vec{x})\rangle : \vec{x} \in L\}$, where L is context free and f computable. Then let $G = \langle \mathtt{S}, N, A, R\rangle$ be a CFG with $L(G) = L$. Put $A' := A$, $C' := N \cup \{\mathtt{S}^\heartsuit\}$ and $M' := M \cup A^*$. For simplicity we presuppose that G is already in Chomsky Normal Form. For every rule ρ of the form $\rho = A \to \vec{x}$ we take a 0–ary mode $\mathtt{R}_\rho$, which is defined as follows:

$$\mathtt{R}_\rho := \langle \vec{x}, A, \vec{x}\rangle \tag{3.23}$$

For every rule ρ of the form $\rho = A \to B\,C$ we take a binary mode $\mathtt{R}_\rho$ defined

by

$$\mathtt{R}_\rho(\langle\vec{x},B,\vec{x}\rangle,\langle\vec{y},C,\vec{y}\rangle) := \langle\vec{x}\vec{y},A,\vec{x}\vec{y}\rangle \tag{3.24}$$

Finally we choose a unary mode S:

$$\mathtt{S}(\langle\vec{x},\mathtt{S},\vec{x}\rangle) := \langle\vec{x},\mathtt{S}^\heartsuit,f(\vec{x})\rangle \tag{3.25}$$

Then $\mathcal{I}$ is indeed the set of signs with category $\mathtt{S}^\heartsuit$. As one can see, this algebra of signs is more perspicuous. The strings are just concatenated. The meanings, however, are not the ones we expect to see. And the category assignment is unstructured. This grammar is not compositional, since it still uses nonstandard meanings. Hence once again some pathological examples, which will show that there exist nonrecursive compositional systems of signs.

Suppose that Δ is a decidable system of signs. This means that there are countable sets E, C and M such that either (i) $\Delta = E\times C\times M$, or (ii) $\Delta = \varnothing$, or (iii) there are two computable functions,

$$d_\bullet : \omega \twoheadrightarrow \Delta, \qquad d_\circ : \omega \twoheadrightarrow (E\times C\times M - \Delta) \tag{3.26}$$

In particular, E, C and M are finite or countable. Also, we can find a bijection $\delta_\bullet : \kappa \to \Delta$, where $\kappa = |\Delta|$. (Simply generate a list $d_\bullet(i)$ for $i = 0,1,\ldots$ and skip repeated items.) Its inverse is also computable. Now we look at the projections $\pi_0 : \langle e,c,m\rangle \mapsto e$, $\pi_1 : \langle e,c,m\rangle \mapsto c$ and $\pi_2 : \langle e,c,m\rangle \mapsto m$.

Definition 3.12 *Let Δ be a system of signs. Δ is called* ***enumerative*** *if the projections π_0, π_1, and π_2 are either bijective and computable or constant.*

Here is an enumerative subsystem of English. Take E to be the set of number names of English (see Section 2.7), $C = \{\nu\}$, where ν is the category of numbers, and $M = \omega$. Now let $\mathcal{E}$ be the set of signs $\langle\vec{x},\nu,n\rangle$, where $\vec{x}$ names the number n in English. It is straightforward to check that $\mathcal{E}$ is enumerative.

Let Δ be enumerative. We introduce two modes, N (zeroary) and S (unary) and say that

$$\begin{aligned} \mathtt{N} &:= \delta_\bullet(0) \\ \mathtt{S}(\sigma) &:= \delta_\bullet(\delta_\bullet^{-1}(\sigma)+1) \end{aligned} \tag{3.27}$$

This generates Δ, as is easily verified. This, however, is not compositional, unless we can show that the S can be defined componentwise. Therefore put

$$\mathtt{S}^\varepsilon(e) := \begin{cases} e & \text{if } \pi_0 \text{ is constant,} \\ \pi_0(\mathtt{S}(\pi_0^{-1}(e))) & \text{otherwise.} \end{cases} \tag{3.28}$$

This is computable if it is decidable whether or not e is in the image of π_0. So, the set $\pi_0[\Delta]$ must be decidable. Similarly $\mathtt{S}^\gamma$ and $\mathtt{S}^\mu$ are defined, and are computable if $\pi_1[\Delta]$ and $\pi_2[\Delta]$, respectively, are decidable.

Definition 3.13 *Δ is called **modularly decidable** if Δ, $\pi_0[\Delta]$, $\pi_1[\Delta]$ and $\pi_2[\Delta]$ are decidable.*

Theorem 3.14 *Suppose that Δ is modularly decidable and enumerative. Then Δ is compositional.* □

Theorem 3.15 (Extension) *Let $\Sigma \subseteq E \times C \times M$ be a recursively enumerable set of signs. Let $\Delta \subseteq \Sigma$ be modularly decidable and enumerative. Assume that E is finite iff π_0 is constant on Δ; similarly for C and M. Then Σ is compositional.*

Proof. We first assume that E, C and M are all infinite. By Theorem 3.14, Δ is compositional. Further, Σ is recursively enumerable. So there is a computable function $\xi : \omega \twoheadrightarrow \Sigma$. Moreover, $\delta_\bullet^{-1}$ is also computable, and so $\xi \circ \delta_\bullet^{-1} : \Delta \twoheadrightarrow \Sigma$ is computable. Add a unary mode $\mathtt{F}$ to the signature and let

$$\begin{aligned} \mathtt{F}^\varepsilon(e) :=& \pi_0((\xi \circ \delta_\bullet^{-1})(\pi_0^{-1}(e))) \\ \mathtt{F}^\varepsilon(c) :=& \pi_1((\xi \circ \delta_\bullet^{-1})(\pi_1^{-1}(c))) \\ \mathtt{F}^\varepsilon(m) :=& \pi_2((\xi \circ \delta_\bullet^{-1})(\pi_2^{-1}(m))) \end{aligned} \tag{3.29}$$

(On all other inputs the functions are not defined.) This is well–defined and surjective. $\langle \mathtt{F}^\varepsilon, \mathtt{F}^\gamma, \mathtt{F}^\mu \rangle$ is partial, computable, and defined only on Δ. Its full image is Σ. Now assume that one of the projections, say π_0, is constant. Then E is finite, by assumption on Σ, say $E = \{e_i : i < n\}$ for some n. Then put $\Sigma_i := \Sigma \cap (\{e_i\} \times C \times M)$. Σ_i is also recursively enumerable. We do the proof as before, with an enumeration $\xi_i : \omega \twoheadrightarrow \Sigma_i$ in place of ξ. Assume n new unary modes, $\mathtt{G}_i$, and put

$$\begin{aligned} \mathtt{G}_i^\varepsilon(e) :=& e_i \\ \mathtt{G}_i^\varepsilon(c) :=& \pi_1((\xi_i \circ \delta_\bullet^{-1})(\pi_1^{-1}(c))) \\ \mathtt{G}_i^\varepsilon(m) :=& \pi_2((\xi_i \circ \delta_\bullet^{-1})(\pi_2^{-1}(m))) \end{aligned} \tag{3.30}$$

All $\langle \mathtt{G}_i^\varepsilon, \mathtt{G}_i^\gamma, \mathtt{G}_i^\mu \rangle$ are computable, partial, and defined exactly on Δ, which they map onto Σ_i. □

In this construction all occurring signs are in Σ. Still, we do want to say that the grammar just constructed is compositional. Namely, if we apply $\mathtt{F}^{\varepsilon}$ to the string $\vec{x}$ we may get a string that may have nothing to do with $\vec{x}$ at all. Evidently, we need to further restrict our operations, for example, by not allowing arbitrary string manipulations. We shall deal with this problem in Section 5.7.

Compositionality in the weak sense defines semantics as an autonomous component of language. When a rule is applied, the semantics may not 'spy' into the phonological form or the syntax to see what it is supposed to do. Rather, it acts autonomously, without that knowledge. Its only input is the semantics of the argument signs and the mode that is being applied. In a similar way syntax is autonomous from phonology and semantics. That this is desirable has been repeatedly argued for by Noam Chomsky. It means that syntactic rules apply regardless of the semantics or the phonological form. It is worthwile to explain that our notion of compositionality not only makes semantics autonomous from syntax and phonology, but also syntax autonomous from phonology and semantics and phonology autonomous from syntax and semantics.

Notes on this section. The notion of sign defined here is the one that is most commonly found in linguistics. In essence it goes back to de Saussure [**?**], published posthumously in 1916, who takes a linguistic sign to consist of a signifier and denotatum (see also Section 5.8). De Saussure therewith diverged from Peirce, for whom a sign was a triadic relation between the signifier, the interpreting subject and the denotatum. (See also [**?**] for a discussion.) On the other hand, following the mainstream we have added to de Saussure signs the category, which is nothing but a statement of the combinatorics of that sign. This structure of a sign is most clearly employed, for example, in Montague Grammar and in the Meaning–to–Text framework of Igor Mel'čuk (see for example [**?**]). Other theories, for example early HPSG and Unification Categorial Grammar also use the tripartite distinction between what they call phonology, syntax and semantics, but signs are not triples but much more complex in structure.

The distinction between compositionality and weak compositionality turns on the question whether the generating functions should work inside the language or whether they may introduce new objects. We strongly opt for the former not only because it gives us a stronger notion. The definition in its informal rendering makes reference to the parts of an expression and their meanings — and in actual practice the parts from which we compose an ex-

pression do have meanings, and it is these meanings we employ in forming the meaning of a complex expression.

Exercise 89. Let $G = \langle \mathtt{S}, N, A, R\rangle$ be a CFG. Put $N' := N \cup \{\mathtt{R}_\rho : \rho \in R\}$, and $R" := \{X \to \mathtt{R}_\rho \vec{\alpha} : \rho = X \to \vec{\alpha} \in R\}$, $G' := \langle \mathtt{S}, N', A, R'\rangle$. Show that G' is transparent.

Exercise 90. Show that English satisfies the conditions of Theorem 3.15. Hence English is compositional!

Exercise 91. Construct an undecidable set Δ such that its projections $\pi_0[\Delta]$, $\pi_1[\Delta]$ and $\pi_2[\Delta]$ are decidable. Construct a Δ which is decidable but not its projection $\pi_0[\Delta]$.

Exercise 92. Show that the functions postulated in the proof of Theorem 3.15, z_γ and m_γ, do exist if Σ is recursively enumerable.

Exercise 93. Say that $\Sigma \subseteq E \times C \times M$ is **extra weakly compositional** if there exists a finite signature Ω and Ω–algebras $\mathfrak{E}'$, $\mathfrak{C}'$ and $\mathfrak{M}'$ over sets $E' \supseteq E$, $C' \supseteq C$ and $M' \supseteq M$, respectively, such that Σ is the carrier set of the 0–generated partial subalgebra of $\mathfrak{E}' \times \mathfrak{C}' \times \mathfrak{M}'$ which belong to the set $E \times C \times M$. (So, the definition is like that of weak compositionality, only that the functions are not necessarily computable.) Show that Σ is extra weakly compositional iff it is countable. (See also [**?**].)

2. Propositional Logic

Before we can enter a discussion of categorial grammar and type systems, we shall have to introduce some techniques from propositional logic. We seize the opportunity to present boolean logic using our notions of the previous section. The alphabet is defined to be $A_P := \{\mathtt{p}, \mathtt{0}, \mathtt{1}, \mathtt{(}, \mathtt{)}, \bot, \neg\}$. Further, let $T := \{P\}$, and $M := \{0, 1\}$. Next, we define the following modes. The zeroary modes are

$$\mathtt{X}_{\vec{\alpha}} := \langle \mathtt{p}\vec{\alpha}, P, 0\rangle, \quad \mathtt{Y}_{\vec{\alpha}} := \langle \mathtt{p}\vec{\alpha}, P, 1\rangle, \quad \mathtt{M}_\bot := \langle \bot, P, 0\rangle \tag{3.31}$$

Here, $\vec{\alpha}$ ranges over (possibly empty) sequences of $\mathtt{0}$ and $\mathtt{1}$. (So, the signature is infinite.) Further, let $\supset$ be the following function:

(3.32)

$\supset$	0	1
0	1	1
1	0	1

The binary mode $\mathtt{M}_{\rightarrow}$ of implication formation is spelled out as follows.

(3.33) $\mathtt{M}_{\rightarrow}(\langle \vec{x}, P, \eta \rangle, \langle \vec{y}, P, \theta \rangle) := \langle (\vec{x} \rightarrow \vec{y}), P, \eta \supset \theta \rangle$

The system of signs generated by these modes is called **boolean logic** and is denoted by Σ_{B}. To see that this is indeed so, let us explain in more conventional terms what these definitions amount to. First, the string language L we have defined is a subset of A_P^*, which is generated as follows.

① If $\vec{\alpha} \in \{\mathtt{0}, \mathtt{1}\}^*$, then $\mathtt{p}\vec{\alpha} \in L$. These sequences are called **propositional variables**.

② $\bot \in L$.

③ If $\vec{x}, \vec{y} \in L$ then $(\vec{x} \rightarrow \vec{y}) \in L$.

$\vec{x}$ is also called a **well–formed formula** (**wff**) or simply a **formula** iff it belongs to L. There are three kinds of wffs.

Definition 3.16 *Let $\vec{x}$ be a well–formed formula. $\vec{x}$ is a **tautology** if $\langle \vec{x}, P, 0 \rangle \notin \Sigma_{\mathsf{B}}$. $\vec{x}$ is a **contradiction** if $\langle \vec{x}, P, 1 \rangle \notin \Sigma_{\mathsf{B}}$. If $\vec{x}$ is neither a tautology nor a contradiction, it is called **contingent**.*

The set of tautologies is denoted by $\mathrm{Taut}_{\mathsf{B}}(\rightarrow, \bot)$, or simply by $\mathrm{Taut}_{\mathsf{B}}$ if the language is clear from the context. It is easy to see that $\vec{x}$ is a tautology iff $(\vec{x} \rightarrow \bot)$ is a contradiction. Likewise, $\vec{x}$ is a contradiction iff $(\vec{x} \rightarrow \bot)$ is a tautology. We now agree on the following convention. Lower case Greek letters are proxy for well–formed formulae, upper case Greek letters are proxy for sets of formulae. Further, we write $\Delta; \varphi$ instead of $\Delta \cup \{\varphi\}$ and $\varphi; \chi$ in place of $\{\varphi, \chi\}$.

Our first task will be to present a calculus with which we can generate all the tautologies of Σ_{B}. For this aim we use a so–called *Hilbert style calculus*. Define the following sets of formulae.

(3.34)
(a0) $(\varphi \rightarrow (\psi \rightarrow \varphi))$
(a1) $((\varphi \rightarrow (\psi \rightarrow \chi)) \rightarrow ((\varphi \rightarrow \psi) \rightarrow (\varphi \rightarrow \chi)))$
(a2) $(\bot \rightarrow \varphi)$
(a3) $(((\varphi \rightarrow \bot) \rightarrow \bot) \rightarrow \varphi)$

The logic axiomatized by (a0) – (a3) is known as **classical** or **boolean logic**, the logic axiomatized by (a0) – (a2) as **intuitionistic logic**. To be more precise, (a0) – (a3) each are sets of formulae. For example:

(3.35) $(\mathrm{a0}) = \{(\varphi \rightarrow (\psi \rightarrow \varphi)) : \varphi, \psi \in L\}$

We call (a0) an **axiom schema** and its elements **instances of** (a0). Likewise with (a1) – (a3).

Definition 3.17 *A finite sequence* $\Pi = \langle \delta_i : i < n \rangle$ *of formulae is a* B*–**proof of*** φ *if (a)* $\delta_{n-1} = \varphi$ *and (b) for all* $i < n$ *either (b1)* δ_i *is an instance of (a0) – (a3) or (b2) there are* $j, k < i$ *such that* $\delta_k = (\delta_j \to \delta_i)$*. The number n is called the **length of*** Π*. We write* $\vdash^{\mathsf{B}} \varphi$ *if there is a* B*–proof of* φ*.*

The formulae (a0) – (a3) are called the **axioms** of this calculus. Moreover, this calculus uses a single inference rule, which is known as Modus Ponens. It is the inference from $(\varphi \to \chi)$ and φ to χ. The easiest part is to show that the calculus generates only tautologies.

Lemma 3.18 *If* $\vdash^{\mathsf{B}} \varphi$ *then* φ *is a tautology.*

The proof is by induction on the length of the proof. The completeness part is somewhat harder and requires a little detour. We shall extend the notion of proof somewhat to cover proofs from assumptions.

Definition 3.19 *A* B*–**proof of*** φ ***from*** Δ *is a finite sequence* $\Pi = \langle \delta_i : i < n \rangle$ *of formulae such that (a)* $\delta_{n-1} = \varphi$ *and (b) for all* $i < n$ *either (b1)* δ_i *is an instance of (a0) – (a3) or (b2) there are* $j, k < i$ *such that* $\delta_k = (\delta_j \to \delta_i)$ *or (b3)* $\delta_i \in \Delta$*. We write* $\Delta \vdash^{\mathsf{B}} \varphi$ *if there is a* B*–proof of* φ *from* Δ*.*

To understand this notion of a hypothetical proof, we shall introduce the notion of an **assignment**. It is common to define an assignment to be a function from variables to the set $\{0, 1\}$. Here, we shall give an effectively equivalent definition.

Definition 3.20 *An **assignment** is a maximal subset A of*

$$\{\mathtt{X}_{\vec{\alpha}} : \vec{\alpha} \in (\mathtt{0} \cup \mathtt{1})^*\} \cup \{\mathtt{Y}_{\vec{\alpha}} : \vec{\alpha} \in (\mathtt{0} \cup \mathtt{1})^*\} \tag{3.36}$$

such that for no $\vec{\alpha}$ *both* $\mathtt{X}_{\vec{\alpha}}, \mathtt{Y}_{\vec{\alpha}} \in A$*.*

(So, an assignment is a set of zeroary modes.) Each assignment defines a closure under the modes $\mathtt{M}_{\bot}$ and $\mathtt{M}_{\to}$, which we denote by $\Sigma_{\mathsf{B}}(A)$.

Lemma 3.21 *Let A be an assignment and* φ *a well–formed formula. Then either* $\langle \varphi, P, 0 \rangle \in \Sigma_{\mathsf{B}}(A)$ *or* $\langle \varphi, P, 1 \rangle \in \Sigma_{\mathsf{B}}(A)$*, but not both.*

The proof is by induction on the length of $\vec{x}$. We say that an assignment A makes a formula φ **true** if $\langle \varphi, P, 1 \rangle \in \Sigma_{\mathsf{B}}(A)$.

Definition 3.22 *Let Δ be a set of formulae and φ a formula. We say that φ* ***follows from*** *(or* ***is a consequence of****) Δ if for all assignments A: if A makes all formulae of Δ true then it makes φ true as well. In that case we write $\Delta \vDash \varphi$.*

Our aim is to show that the Hilbert calculus characterizes this notion of consequence:

Theorem 3.23 $\Delta \vdash^{\mathsf{B}} \varphi$ *iff* $\Delta \vDash \varphi$.

Again, the proof has to be deferred until the matter is sufficiently simplified. Let us first show the following fact, known as the **Deduction Theorem** (**DT**).

Lemma 3.24 (Deduction Theorem) $\Delta; \varphi \vdash^{\mathsf{B}} \chi$ *iff* $\Delta \vdash^{\mathsf{B}} (\varphi \to \chi)$.

Proof. The direction from right to left is immediate and left to the reader. Now, for the other direction suppose that $\Delta; \varphi \vdash^{\mathsf{B}} \chi$. Then there exists a proof $\Pi = \langle \delta_i : i < n \rangle$ of χ from $\Delta; \varphi$. We shall inductively construct a proof $\Pi' = \langle \delta'_j : j < m \rangle$ of $(\varphi \to \chi)$ from Δ. The construction is as follows. We define Π_i inductively.

$$\Pi_0 := \varepsilon, \qquad \Pi_{i+1} := \Pi_i^\frown \Sigma_i, \tag{3.37}$$

where Σ_i, $i < n$, is defined as given below. Furthermore, we will verify inductively that Π_{i+1} is a proof of its last formula, which is $(\varphi \to \delta_i)$. Then $\Pi' := \Pi_n$ will be the desired proof, since $\delta_{n-1} = \chi$. Choose $i < n$. Then either (1) $\delta_i \in \Delta$ or (2) δ is an instance of (a0) – (a3) or (3) $\delta_i = \varphi$ or (4) there are $j, k < i$ such that $\delta_k = (\delta_j \to \delta_i)$. In the first two cases we put $\Sigma_i := \langle \delta_i, (\delta_i \to (\varphi \to \delta_i)), (\varphi \to \delta_i) \rangle$. In Case (3) we put

$$\begin{aligned} \Sigma_i := \quad & \langle ((\varphi \to ((\varphi \to \varphi) \to \varphi)) \to ((\varphi \to (\varphi \to \varphi)) \to (\varphi \to \varphi))), \\ & (\varphi \to ((\varphi \to \varphi) \to \varphi)), \\ & ((\varphi \to (\varphi \to \varphi)) \to (\varphi \to \varphi)), \\ & (\varphi \to (\varphi \to \varphi)), \\ & (\varphi \to \varphi) \rangle \end{aligned} \tag{3.38}$$

Σ_i is a proof of $(\varphi \to \varphi)$, as is readily checked. Finally, Case (4). There are $j, k < i$ such that $\delta_k = (\delta_j \to \delta_i)$. Then, by induction hypothesis, $(\varphi \to \delta_j)$ and $(\varphi \to \delta_k) = (\varphi \to (\delta_j \to \delta_i))$ already occur in the proof. Then put

$$\begin{aligned} \Sigma_i := \quad & \langle ((\varphi \to (\delta_j \to \delta_i)) \to ((\varphi \to \delta_j) \to (\varphi \to \delta_i))), \\ & ((\varphi \to \delta_j) \to (\varphi \to \delta_i)), \\ & (\varphi \to \delta_i) \rangle \end{aligned} \tag{3.39}$$

It is verified that Π_{i+1} is a proof of $(\varphi\to\delta_i)$. □

A special variant is the following.

Lemma 3.25 (Little Deduction Theorem) *For all* Δ *and* φ*:* $\Delta \vdash^{\mathsf{B}} \varphi$ *if and only if* $\Delta; (\varphi\to\bot) \vdash^{\mathsf{B}} \bot$.

Proof. Assume that $\Delta \vdash^{\mathsf{B}} \varphi$. Then there is a proof Π of φ from Δ. It follows that $\Pi^\frown\langle(\varphi\to\bot),\bot\rangle$ is a proof of $\bot$ from $\Delta; (\varphi\to\bot)$. Conversely, assume that $\Delta; (\varphi\to\bot) \vdash^{\mathsf{B}} \bot$. Applying DT we get $\Delta \vdash^{\mathsf{B}} ((\varphi\to\bot)\to\bot)$. Using (a3) we get $\Delta \vdash^{\mathsf{B}} \varphi$. □

Proposition 3.26 *The following holds.*

① $\varphi \vdash^{\mathsf{B}} \varphi$.

② *If* $\Delta \subseteq \Delta'$ *and* $\Delta \vdash^{\mathsf{B}} \varphi$ *then also* $\Delta' \vdash^{\mathsf{B}} \varphi$.

③ *If* $\Delta \vdash^{\mathsf{B}} \varphi$ *and* $\Gamma; \varphi \vdash^{\mathsf{B}} \chi$ *then* $\Gamma; \Delta \vdash^{\mathsf{B}} \chi$.

This is easily verified. Now we are ready for the proof of Theorem 3.23. An easy induction on the length of a proof establishes that if $\Delta \vdash^{\mathsf{B}} \varphi$ then also $\Delta \vDash \varphi$. (This is called the *correctness* of the calculus.) So, the converse implication, which is the *completeness* part needs proof. Assume that $\Delta \nvdash^{\mathsf{B}} \varphi$. We shall show that also $\Delta \nvDash \varphi$. Call a set Σ **consistent** (**in** $\vdash^{\mathsf{B}}$) if $\Sigma \nvdash^{\mathsf{B}} \bot$.

Lemma 3.27 ① *Let* $\Delta; (\varphi\to\chi)$ *be consistent. Then either* $\Delta; (\varphi\to\bot)$ *is consistent or* $\Delta; \chi$ *is consistent.*

② *Let* $\Delta; ((\varphi\to\chi)\to\bot)$ *be consistent. Then also* $\Delta; \varphi; (\chi\to\bot)$ *is consistent.*

Proof. ①. Assume that both $\Delta; (\varphi\to\bot)$ and $\Delta; \chi$ are inconsistent. Then we have $\Delta; (\varphi\to\bot) \vdash^{\mathsf{B}} \bot$ and $\Delta; \chi \vdash^{\mathsf{B}} \bot$. So $\Delta \vdash^{\mathsf{B}} ((\varphi\to\bot)\to\bot)$ by DT and, using (a3), $\Delta \vdash^{\mathsf{B}} \varphi$. Hence $\Delta; (\varphi\to\chi) \vdash^{\mathsf{B}} \varphi$ and so $\Delta; (\varphi\to\chi) \vdash^{\mathsf{B}} \chi$. Because $\Delta; \chi \vdash^{\mathsf{B}} \bot$, we also have $\Delta; (\varphi\to\chi) \vdash^{\mathsf{B}} \bot$, showing that $\Delta; (\varphi\to\chi)$ is inconsistent. ②. Assume $\Delta; \varphi; (\chi\to\bot)$ is inconsistent. Then $\Delta; \varphi; (\chi\to\bot) \vdash^{\mathsf{B}} \bot$. So, $\Delta; \varphi \vdash^{\mathsf{B}} ((\chi\to\bot)\to\bot)$, by applying DT. So, $\Delta; \varphi \vdash^{\mathsf{B}} \chi$, using (a3). Applying DT we get $\Delta \vdash^{\mathsf{B}} (\varphi\to\chi)$. Using (a3) and DT once again it is finally seen that $\Delta; ((\varphi\to\chi)\to\bot)$ is inconsistent. □

Finally, let us return to our proof of the completeness theorem. We assume that $\Delta \nvdash^{\mathsf{B}} \varphi$. We have to find an assignment A that makes Δ true but not φ. We may also apply the Little DT and assume that $\Delta; (\varphi\to\bot)$ is consistent and find an assignment that makes this set true. The way to find such an assignment is by applying the so–called downward closure of the set.

Definition 3.28 *A set Δ is* ***downward closed*** *iff (1) for all $(\varphi\to\chi) \in \Delta$ either $(\varphi\to\bot) \in \Delta$ or $\chi \in \Delta$ and (2) for all formulae $((\varphi\to\chi)\to\bot) \in \Delta$ also $\varphi \in \Delta$ and $(\chi\to\bot) \in \Delta$.*

Now, by Lemma 3.27 every consistent set has a consistent closure Δ^*. (It is an exercise for the diligent reader to show this. In fact, for infinite sets a little work is needed here, but we really need this only for finite sets.) Define the following assignment.

$$A := \{\langle \mathtt{p}\vec{\alpha}, P, 1\rangle : (\mathtt{p}\vec{\alpha}\to\bot) \text{ does not occur in } \Delta^*\} \cup \{\langle \mathtt{p}\vec{\alpha}, P, 0\rangle : (\mathtt{p}\vec{\alpha}\to\bot) \text{ does occur in } \Delta^*\} \tag{3.40}$$

It is shown by induction on the formulae of Δ^* that the so–defined assignment makes every formula of Δ^* true. Using the correspondence between syntactic derivability and semantic consequence we immediately derive the following.

Theorem 3.29 (Compactness Theorem) *Let φ be a formula and Δ a set of formulae such that $\Delta \vDash \varphi$. Then there exists a finite set $\Delta' \subseteq \Delta$ such that $\Delta' \vDash \varphi$.*

Proof. Suppose that $\Delta \vDash \varphi$. Then $\Delta \vdash^{\mathsf{B}} \varphi$. Hence there exists a proof of φ from Δ. Let Δ' be the set of those formulae in Δ that occur in that proof. Δ' is finite. Clearly, this proof is a proof of φ from Δ', showing $\Delta' \vdash^{\mathsf{B}} \varphi$. Hence $\Delta' \vDash \varphi$. □

Usually, one has more connectives than just $\bot$ and $\to$. Now, two effectively equivalent strategies suggest themselves, and they are used whenever convenient. The first is to introduce a new connective as an abbreviation. So, we might define (for well–formed formulae)

$$\neg\varphi := \varphi\to\bot \tag{3.41}$$

$$\varphi\vee\chi := (\varphi\to\bot)\to\chi \tag{3.42}$$

$$\varphi\wedge\chi := (\varphi\to(\chi\to\bot))\to\bot \tag{3.43}$$

After the introduction of these abbreviations, everything is the same as before, because we have not changed the language, only our way of referring to its strings. However, we may also change the language by expanding the alphabet. In the cases at hand we will add the following unary and binary modes (depending on which symbol is to be added):

$$\mathtt{M}_{\neg}(\langle \vec{x}, P, \eta\rangle) := \langle (\neg\vec{x}), P, -\eta\rangle \tag{3.44}$$

$$\mathtt{M}_{\vee}(\langle \vec{x}, P, \eta\rangle, \langle \vec{y}, P, \theta\rangle) := \langle (\vec{x}\vee\vec{y}), P, \eta\cup\theta\rangle \tag{3.45}$$

$$\mathtt{M}_{\wedge}(\langle \vec{x}, P, \eta\rangle, \langle \vec{y}, P, \theta\rangle) := \langle (\vec{x}\wedge\vec{y}), P, \eta\cap\theta\rangle \tag{3.46}$$

(3.47)

$\cup$	0	1
0	0	1
1	1	1

$\cap$	0	1
0	0	0
1	0	1

	$-$
0	1
1	0

For $\wedge$, $\rightarrow$ and $\neg$ we need the postulates shown in (3.48), (3.49) and (3.50), respectively:

(3.48) $$(\varphi\rightarrow(\psi\rightarrow(\varphi\wedge\psi))),(\varphi\rightarrow(\psi\rightarrow(\psi\wedge\varphi))),$$
$$((\varphi\wedge\psi)\rightarrow\varphi),((\varphi\wedge\psi)\rightarrow\psi)$$

(3.49) $$(\varphi\rightarrow(\varphi\vee\psi)),(\psi\rightarrow(\varphi\vee\psi)),$$
$$(((\varphi\vee\psi)\rightarrow\chi)\rightarrow(\varphi\rightarrow\chi)),(((\varphi\vee\psi)\rightarrow\chi)\rightarrow(\psi\rightarrow\chi))$$

(3.50) $$((\varphi\rightarrow\psi)\rightarrow((\neg\psi)\rightarrow(\neg\varphi))),(\varphi\rightarrow(\neg(\neg\varphi)))$$

Notice that in defining the axioms we have made use of $\rightarrow$ alone. The formula (3.51) is derivable.

(3.51) $$((\neg(\neg\varphi))\rightarrow\varphi)$$

If we eliminate the connective $\perp$ and define $\Delta\vdash\varphi$ as before (eliminating the axioms (a2) and (a3), however) we get once again intuitionistic logic, unless we add (3.51). The semantics of intuitionistic logic is too complicated to be explained here, so we just use the Hilbert calculus to introduce it. We claim that with only (a0) and (a1) it is not possible to prove all formulae of $\mathrm{Taut}_{\mathrm{B}}$ that use only $\rightarrow$. A case in point is the formula

(3.52) $$(((\varphi\rightarrow\chi)\rightarrow\varphi)\rightarrow\varphi)$$

which is known as *Peirce's Formula*. Together with Peirce's Formula, (a0) and (a1) axiomatize the full set of tautologies of boolean logic in $\rightarrow$. The calculus based on (a0) and (a1) is called H and we write $\Delta\vdash^{\mathsf{H}}\chi$ to say that there is a proof in the Hilbert calculus of χ from Δ using (a0) and (a1).

Rather than axiomatizing the set of tautologies we can also axiomatize the deducibility relation itself. This idea goes back to Gerhard Gentzen, who used it among other to show the consistency of arithmetic (which is of no concern here). For simplicity, we stay with the language with only the arrow. We shall axiomatize the derivability of intuitionistic logic. The statements that we are deriving now have the form '$\Delta\vdash\varphi$' and are called **sequents**. Δ is called the **antecedent** and φ the **succedent** of that sequent. The axioms are

(3.53) (ax) $\varphi\vdash\varphi$

Then there are the following rules of introduction of connectives:

$$(3.54)\qquad (\mathbf{I}\rightarrow)\ \frac{\Delta;\varphi\vdash\chi}{\Delta\vdash(\varphi\rightarrow\chi)}\qquad (\rightarrow\mathbf{I})\ \frac{\Delta\vdash\varphi\qquad\Delta;\psi\vdash\chi}{\Delta;(\varphi\rightarrow\psi)\vdash\chi}$$

Notice that these rules introduce occurrences of the arrow. The rule (**I**→) introduces an occurrence on the right hand side of $\vdash$, while (→**I**) puts an occurrence on the left hand side. (The names of the rules are chosen accordingly.) Further, there are the following so–called rules of inference:

$$(3.55)\qquad (\text{cut})\ \frac{\Delta\vdash\varphi\qquad\Theta;\varphi\vdash\chi}{\Delta;\Theta\vdash\chi}\qquad (\text{mon})\ \frac{\Delta\vdash\varphi}{\Delta;\Theta\vdash\varphi}$$

The sequents above the line are called the **premisses**, the sequent below the lines the **conclusion** of the rule. Further, the formulae that are introduced by the rules (→**I**) and (**I**→) are called **main formulae**, and the formula φ in (cut) the **cut–formula**. Let us call this the **Gentzen calculus**. It is denoted by $\mathcal{H}$.

Definition 3.30 *Let $\Delta\vdash\varphi$ be a sequent. A **(sequent) proof of length** n of $\Delta\vdash\varphi$ in $\mathcal{H}$ is a sequence $\Pi=\langle\Sigma_i\vdash\chi_i : i<n+1\rangle$ such that (a) $\Sigma_n=\Delta$, $\chi_n=\varphi$, (b) for all $i<n+1$ either (ba) $\Sigma_i\vdash\chi_i$ is an axiom or (bb) $\Sigma_i\vdash\chi_i$ follows from some earlier sequents by application of a rule of $\mathcal{H}$.*

It remains to say what it means that a sequent follows from some other sequents by application of a rule. This, however, is straightforward. For example, $\Delta\vdash(\varphi\rightarrow\chi)$ follows from the earlier sequents by application of the rule (**I**→) if among the earlier sequents we find the sequent $\Delta;\varphi\vdash\chi$. We shall define also a different notion of proof, which is based on trees rather than sequences. In doing so, we shall also formulate a somewhat more abstract notion of a calculus.

Definition 3.31 *A **finitary rule** is a pair $\rho=\langle M,\mathfrak{S}\rangle$, where M is a finite set of sequents and $\mathfrak{S}$ a single sequent. (These rules are written down using lower case Greek letters as schematic variables for formulae and upper case Greek letters as schematic variables for sets of formulae.) A **sequent calculus** $\mathcal{S}$ is a set of finitary rules. An $\mathcal{S}$–**proof tree** is a triple $\mathbb{T}=\langle T,\succ,\ell\rangle$ such that $\langle T,\prec\rangle$ is a tree and for all x: if $\{y_i : i<n\}$ are the daughters of T, $\langle\{\ell(y_i) : i<n\},\ell(x)\rangle$ is an instance of a rule of $\mathcal{S}$. If r is the root of $\mathbb{T}$, we say that $\mathbb{T}$ **proves** $\ell(r)$ **in** $\mathcal{S}$. We write*

$$(3.56)\qquad \overset{\mathcal{S}}{\rightsquigarrow}\Delta\vdash\varphi$$

to say that the sequent $\Delta\vdash\varphi$ has a proof in $\mathcal{S}$.

We start with the only rule for $\bot$, which actually is an axiom.

(3.57) $\quad (\bot\mathbf{I}) \quad \bot \vdash \varphi$

For negation we have these rules.

(3.58) $$(\neg\mathbf{I})\ \frac{\Delta \vdash \varphi}{\Delta; (\neg\varphi) \vdash \bot} \qquad (\mathbf{I}\neg)\ \frac{\Delta; \varphi \vdash \bot}{\Delta \vdash (\neg\varphi)}$$

The following are the rules for conjunction.

(3.59) $$(\wedge\mathbf{I})\ \frac{\Delta; \varphi; \psi \vdash \chi}{\Delta; (\varphi\wedge\psi) \vdash \chi} \qquad (\mathbf{I}\wedge)\ \frac{\Delta \vdash \varphi \qquad \Delta \vdash \psi}{\Delta \vdash (\varphi\wedge\psi)}$$

Finally, these are the rules for $\vee$.

(3.60) $$(\vee\mathbf{I})\ \frac{\Delta; \varphi \vdash \chi \qquad \Delta; \psi \vdash \chi}{\Delta; (\varphi\vee\psi) \vdash \chi}$$
$$(\mathbf{I}_1\vee)\ \frac{\Delta \vdash \varphi}{\Delta \vdash (\varphi\vee\psi)} \qquad (\mathbf{I}_2\vee)\ \frac{\Delta \vdash \psi}{\Delta \vdash (\varphi\vee\psi)}$$

Let us return to the calculus $\mathcal{H}$. We shall first of all show that we can weaken the rule system without changing the set of derivable sequents. Notice that the following is a proof tree.

(3.61) $$\frac{\dfrac{\varphi \vdash \varphi \qquad \psi \vdash \psi}{(\varphi\to\psi); \varphi \vdash \psi}}{(\varphi\to\psi) \vdash (\varphi\to\psi)}$$

This shows us that in place of the rule (ax) we may actually use a restricted rule, where we have only $\mathtt{p}_i \vdash \mathtt{p}_i$. Call such an instance of (ax) **primitive**. This fact may be used for the following theorem.

Lemma 3.32 $\overset{\mathcal{H}}{\rightsquigarrow} \Delta \vdash (\varphi\to\chi)$ *iff* $\overset{\mathcal{H}}{\rightsquigarrow} \Delta; \varphi \vdash \chi$.

Proof. From right to left follows using the rule (**I**$\to$). Let us prove the other direction. We know that there exists a proof tree for $\Delta \vdash (\varphi\to\chi)$ from primitive axioms. Now we trace backwards the occurrence of $(\varphi\to\chi)$ in the tree from the root upwards. Obviously, since the formula has not been introduced by (ax), it must have been introduced by the rule (**I**$\to$). Let x be the node where the formula is introduced. Then we remove x from the tree, thereby also removing that instance of (**I**$\to$). Going down from x, we have to repair our proof

as follows. Suppose that at $y < x$ we have an instance of (mon). Then instead of the proof part to the left we use the one to the right.

$$\text{(3.62)} \qquad \frac{\Sigma \vdash (\varphi \to \chi)}{\Sigma; \Theta \vdash (\varphi \to \chi)} \qquad \frac{\Sigma; \varphi \vdash \chi}{\Sigma; \Theta; \varphi \vdash \chi}$$

Suppose that we have an instance of (cut). Then our specified occurrence of $(\varphi \to \chi)$ is the one that is on the right of the target sequent. So, in place of the proof part on the left we use the one on the right.

$$\text{(3.63)} \qquad \frac{\Delta \vdash \psi \quad \Theta; \psi \vdash (\varphi \to \chi)}{\Delta; \Theta \vdash (\varphi \to \chi)} \qquad \frac{\Delta \vdash \psi \quad \Theta; \varphi; \psi \vdash \chi}{\Delta; \Theta; \varphi \vdash \chi}$$

Now suppose that we have an instance of (**→I**). Then this instance must be as shown to the left. We replace it by the one on the right.

$$\text{(3.64)} \qquad \frac{\Delta \vdash \tau \quad \Delta; \psi \vdash (\varphi \to \chi)}{\Delta; (\tau \to \psi) \vdash (\varphi \to \chi)} \qquad \frac{\Delta \vdash \tau \quad \Delta; \varphi; \psi \vdash \chi}{\Delta; (\tau \to \psi); \varphi \vdash \chi}$$

The rule (**→I**) does not occur below x, as is easily seen. This concludes the replacement. It is verified that after performing these replacements, we obtain a proof tree for $\Delta; \varphi \vdash \chi$. □

Theorem 3.33 $\Delta \vdash^{\mathsf{H}} \varphi$ *iff* $\overset{\mathcal{H}}{\rightsquigarrow} \Delta \vdash \varphi$.

Proof. Suppose that $\Delta \vdash^{\mathsf{H}} \varphi$. By induction on the length of the proof we shall show that $\overset{\mathcal{H}}{\rightsquigarrow} \Delta \vdash \varphi$. Using DT we may restrict ourselves to $\Delta = \varnothing$. First, we shall show that (a0) and (a1) can be derived. (a0) is derived as follows.

$$\text{(3.65)} \qquad \begin{array}{c} \varphi \vdash \varphi \\ \hline \varphi; \psi \vdash \varphi \\ \hline \varphi \vdash (\psi \to \varphi) \\ \hline \vdash (\varphi \to (\psi \to \varphi)) \end{array}$$

For (a1) we need a little more work.

$$\text{(3.66)} \qquad \begin{array}{c} \varphi \vdash \varphi \quad \begin{array}{c} \varphi \vdash \varphi \quad \begin{array}{c} \psi \vdash \psi \quad \chi \vdash \chi \\ \hline \psi; (\psi \to \chi) \vdash \chi \end{array} \\ \hline \varphi; (\varphi \to \psi); (\psi \to \chi) \vdash \chi \end{array} \\ \hline (\varphi \to (\psi \to \chi)); (\varphi \to \psi); \varphi \vdash \chi \end{array}$$

If we apply (**I**→) three times we get (a1). Next we have to show that if we have $\overset{\mathcal{H}}{\rightsquigarrow} \varnothing \vdash \varphi$ and $\overset{\mathcal{H}}{\rightsquigarrow} \varnothing \vdash (\chi \to \varphi)$ then $\overset{\mathcal{H}}{\rightsquigarrow} \varnothing \vdash \chi$. By DT, we also have $\overset{\mathcal{H}}{\rightsquigarrow} \varphi \vdash \chi$ and then a single application of (cut) yields the desired conclusion. This proves that $\overset{\mathcal{H}}{\rightsquigarrow} \varnothing \vdash \varphi$. Now, conversely, we have to show that $\overset{\mathcal{H}}{\rightsquigarrow} \Delta \vdash \varphi$ implies that $\Delta \vdash^{\mathsf{H}} \varphi$. This is shown by induction on the height of the nodes in the proof tree. If it is 1, we have an axiom: however, $\varphi \vdash^{\mathsf{H}} \varphi$ clearly holds. Now suppose the claim is true for all nodes of depth $< i$ and let x be of depth i. Then x is the result of applying one of the four rules. (→**I**). By induction hypothesis, $\Delta \vdash^{\mathsf{H}} \varphi$ and $\Delta; \psi \vdash^{\mathsf{H}} \chi$. We need to show that $\Delta; (\varphi \to \psi) \vdash^{\mathsf{H}} \chi$. Simply let Π_1 be a proof of φ from Δ, Π_2 a proof of χ from $\Delta; \psi$. Then Π_3 is a proof of χ from $\Delta; (\varphi \to \psi)$.

$$\Pi_3 := \Pi_1^\frown \langle (\varphi \to \psi), \psi \rangle^\frown \Pi_2 \tag{3.67}$$

(**I**→). This is straightforward from DT. (cut). Suppose that Π_1 is a proof of φ from Δ and Π_2 a proof of χ from $\Theta; \varphi$. Then $\Pi_1^\frown \Pi_2$ is a proof of χ from $\Delta; \varphi$, as is easily seen. (mon). This follows from Proposition 3.26. □

Call a rule ρ **admissible** for a calculus $\mathcal{S}$ if any sequent $\Delta \vdash \varphi$ that is derivable in $\mathcal{S} + \rho$ is also derivable in $\mathcal{S}$. Conversely, if ρ is admissible in $\mathcal{S}$, we say that ρ is **eliminable from** $\mathcal{S} + \rho$. We shall show that (cut) is eliminable from $\mathcal{H}$, so that it can be omitted without losing derivable sequents. As cut–elimination will play a big role in the sequel, the reader is asked to watch the procedure carefully.

Theorem 3.34 (Cut Elimination) (cut) *is eliminable from* $\mathcal{H}$.

Proof. Recall that (cut) is the following rule.

$$\text{(cut)} \quad \frac{\Delta \vdash \varphi \quad \Theta; \varphi \vdash \chi}{\Delta; \Theta \vdash \chi} \tag{3.68}$$

Two measures are introduced. The **degree** of (3.68) is

$$d := |\Delta| + |\Theta| + |\varphi| + |\chi| \tag{3.69}$$

The **weight** of (3.68) is 2^d. The **cut–weight** of a proof tree $\mathbb{T}$ is the sum over all weights of occurrences of cuts (= instances of (cut)) in it. Obviously, the cut–weight of a proof tree is zero iff there are no cuts in it. We shall now present a procedure that operates on proof trees in such a way that it

reduces the cut–weight of every given tree if it is nonzero. This procedure is as follows. Let $\mathbb{T}$ be given, and let x be a node carrying the conclusion of an instance of (cut). We shall assume that above x no instances of (cut) exist. (Obviously, x exists if there are cuts in $\mathbb{T}$.) x has two mothers, y_1 and y_2. Case (1). Suppose that y_1 is a leaf. Then we have $\ell(y_1) = \varphi \vdash \varphi$, $\ell(y_2) = \Theta; \varphi \vdash \chi$ and $\ell(x) = \Theta; \varphi \vdash \chi$. In this case, we may simply skip the application of cut by dropping the nodes x and y_1. This reduces the degree of the cut by $2 \cdot |\varphi|$, since this application of (cut) has been eliminated without trace. Case (2). Suppose that y_2 is a leaf. Then $\ell(y_2) = \chi \vdash \chi$, $\ell(y_1) = \Delta \vdash \varphi$, whence $\varphi = \chi$ and $\ell(x) = \Delta \vdash \varphi = \ell(y_1)$. Eliminate x and y_2. This reduces the cut–weight by the weight of that cut. Case (3). Suppose that y_1 has been obtained by application of (mon). Then the proof is as shown on the left.

(3.70)
$$\dfrac{\dfrac{\Delta \vdash \varphi}{\Delta; \Delta' \vdash \varphi} \quad \Theta; \varphi \vdash \chi}{\Delta; \Delta'; \Theta \vdash \chi} \qquad\qquad \dfrac{\dfrac{\Delta \vdash \varphi \qquad \Theta; \varphi \vdash \chi}{\Delta; \Theta \vdash \chi}}{\Delta; \Delta'; \Theta \vdash \chi}$$

We may assume that $\Delta' > 0$. We replace the local tree by the one on the right. The cut weight is reduced by

(3.71)
$$2^{|\Delta|+|\Delta'|+|\Theta|+|\varphi|+|\chi|} - 2^{|\Delta|+|\Theta|+|\varphi|+|\chi|} > 0$$

Case (4). $\ell(y_2)$ has been obtained by application of (mon). This is similar to the previous case. Case (5). $\ell(y_1)$ has been obtained by (**→I**). Then the main formula is not the cut formula.

(3.72)
$$\dfrac{\dfrac{\Delta \vdash \rho \quad \Delta; \tau \vdash \varphi}{\Delta; (\rho \to \tau) \vdash \varphi} \qquad \Theta; \varphi \vdash \chi}{\Delta; \Theta; (\rho \to \tau) \vdash \chi}$$

And the cut can be rearranged as follows.

(3.73)
$$\dfrac{\dfrac{\Delta \vdash \rho}{\Delta; \Theta \vdash \rho} \qquad \dfrac{\Delta; \tau \vdash \varphi \quad \Theta; \varphi \vdash \chi}{\Delta; \Theta; \tau \vdash \chi}}{\Delta; \Theta; (\rho \to \tau) \vdash \chi}$$

Here, the degree of the cut is reduced by $|(\rho \to \tau)| - |\tau| > 0$. Thus the cut–weight is reduced as well. Case (6). $\ell(y_2)$ has been obtained by (**→I**). Assume $\varphi \neq (\rho \to \tau)$.

(3.74)
$$\dfrac{\Delta \vdash \varphi \qquad \dfrac{\Theta; \varphi \vdash \rho \quad \Theta; \varphi; \tau \vdash \chi}{\Theta; \varphi; (\rho \to \tau) \vdash \chi}}{\Delta; \Theta; (\rho \to \tau) \vdash \chi}$$

In this case we can replace the one cut by two as follows.

$$\frac{\Delta \vdash \varphi \qquad \Theta;\varphi \vdash \rho}{\Delta;\Theta \vdash \rho} \qquad \frac{\Delta \vdash \varphi \qquad \Theta;\varphi;\tau \vdash \chi}{\Delta;\Theta;\tau \vdash \chi} \tag{3.75}$$

If we now apply (**→I**), we get the same sequent. The cut–weight has been diminished by

$$2^{|\Delta|+|\Theta|+|\rho|+|\tau|+3} - 2^{|\Delta|+|\Theta|+|\rho|} - 2^{|\Delta|+|\Theta|+|\tau|} > 0 \tag{3.76}$$

(See also below for the same argument.) Suppose however $\varphi = (\rho{\to}\tau) \notin \Theta$. Then either φ is not the main formula of $\ell(y_1)$, in Case (1), (3), (5), or it actually is the main formula, and then we are in Case (7), to which we now turn. Case (7). $\ell(y_1)$ has been introduced by (**I→**). If the cut formula is not the main formula, we are in cases (2), (4), (6) or (8), which we dealt with separately. Suppose however the main formula is the cut formula. Here, we cannot simply permute the cut unless $\ell(y_2)$ is the result of applying (**→I**). In this case we proceed as follows. $\varphi = (\rho{\to}\tau)$ for some ρ and τ. The local proof is as follows.

$$\frac{\dfrac{\Delta;\rho \vdash \tau}{\Delta \vdash (\rho{\to}\tau)} \qquad \dfrac{\Theta \vdash \rho \quad \Theta;\tau \vdash \chi}{\Theta;(\rho{\to}\tau) \vdash \chi}}{\Delta;\Theta \vdash \chi} \tag{3.77}$$

This is rearranged in the following way.

$$\frac{\dfrac{\Delta;\rho \vdash \tau \qquad \Theta \vdash \rho}{\Delta;\Theta \vdash \tau} \qquad \dfrac{\Theta;\tau \vdash \chi}{\Delta;\Theta;\tau \vdash \chi}}{\Delta;\Theta \vdash \chi} \tag{3.78}$$

This operation eliminates the cut in favour of two cuts. The overall degree of these cuts may be increased, but the weight has been decreased. Let $d := |\Delta;\Theta|$, $p := |(\rho{\to}\tau)|$. Then the first cut has weight $2^{d+p+|\chi|}$. The two other cuts have weight

$$2^{d+|\rho|+|\tau|} + 2^{d+|\tau|+|\chi|} \leq 2^{d+|\rho|+|\tau|+|\chi|} < 2^{d+p+|\chi|} \tag{3.79}$$

since $p > |\rho| + |\tau| > 0$. (Notice that $2^{a+c} + 2^{a+d} = 2^a \cdot (2^c + 2^d) \leq 2^a \cdot 2^{c+d} = 2^{a+c+d}$ if $c, d > 0$.) Case (8). $\ell(y_2)$ has been obtained by (**I→**). Then $\chi =$

$(\rho{\to}\tau)$ for some ρ and τ. We replace the left hand proof part by the right hand part, and the degree is reduced by $|(\rho{\to}\tau)| - |\tau| > 0$.

$$\text{(3.80)} \qquad \dfrac{\Delta \vdash \varphi \qquad \dfrac{\Theta;\varphi;\rho \vdash \tau}{\Theta;\varphi \vdash (\rho{\to}\tau)}}{\Delta;\Theta \vdash (\rho{\to}\tau)} \qquad \dfrac{\dfrac{\Delta \vdash \varphi \qquad \Theta;\rho;\varphi \vdash \tau}{\Delta;\Theta;\rho \vdash \tau}}{\Delta;\Theta \vdash (\rho{\to}\tau)}$$

So, in each case we managed to decrease the cut–weight. This concludes the proof. □

Before we conclude this section we shall mention another deductive calculus, called **Natural Deduction**. It uses proof trees, but is based on the Deduction Theorem. First of all notice that we can write Hilbert style proofs also in tree format. Then the leaves of the proof tree are axioms, or assumptions, and the only rule we are allowed to use is **Modus Ponens**.

$$\text{(3.81)} \qquad \text{(MP)} \quad \dfrac{(\varphi{\to}\psi) \qquad \varphi}{\psi}$$

This, however, is a mere reformulation of the previous calculus. The idea behind natural deduction is that we view Modus Ponens as a rule to *eliminate* the arrow, while we add another rule that allows to introduce it. It is as follows.

$$\text{(3.82)} \qquad (\mathbf{I}{\to}) \quad \dfrac{\psi}{(\varphi{\to}\psi)}$$

However, when this rule is used, the formula φ may be eliminated from the assumptions. Let us see how this goes. Let x be a node. Let us call the set $A(x) := \{\langle y, \ell(y)\rangle : y > x, y \text{ leaf}\}$ the set of **assumptions of** x. If (**I**→) is used to introduce $(\varphi{\to}\psi)$, any number of assumptions of x that have the form $\langle y, \varphi\rangle$ may be retracted. In order to know what assumption has been effectively retracted, we check mark the retracted assumptions by a superscript (e. g. $\varphi^{\surd}$). Here are the standard rules for the other connectives. The fact that the assumption φ is or may be removed is annotated as follows:

$$\text{(3.83)} \qquad (\mathbf{I}{\to}) \quad \dfrac{\begin{array}{c}[\varphi]\\ \vdots \\ \psi\end{array}}{(\varphi{\to}\psi)} \qquad (\mathbf{E}{\to}) \quad \dfrac{(\varphi{\to}\psi) \quad \varphi}{\psi}$$

Here, $[\varphi]$ means that any number of assumptions of the form φ above the node carrying φ may be check marked when using the rule. (So, it does *not* mean that it requires these formulae to be assumptions.) The rule (**E**$\neg$) is nothing but (MP). First, conjunction.

(3.84) $$(\mathbf{I}\wedge)\ \frac{\varphi \quad \psi}{(\varphi\wedge\psi)} \qquad (\mathbf{E}_1\wedge)\ \frac{(\varphi\wedge\psi)}{\varphi} \qquad (\mathbf{E}_2\wedge)\ \frac{(\varphi\wedge\psi)}{\psi}$$

The next is $\bot$:

(3.85) $$(\mathbf{E}\bot)\ \frac{\bot}{\varphi}$$

For negation we need some administration of the check mark.

(3.86) $$(\mathbf{I}\neg)\ \frac{\begin{array}{c}[\varphi]\\ \vdots\\ \bot\end{array}}{(\neg\varphi)} \qquad (\mathbf{E}\neg)\ \frac{\varphi \quad (\neg\varphi)}{\bot}$$

So, using the rule (**I**$\neg$) any number of assumptions of the form φ may be check marked. Disjunction is even more complex.

$$(\mathbf{I}_1\vee)\ \frac{\varphi}{(\varphi\vee\psi)} \qquad (\mathbf{I}_2\vee)\ \frac{\psi}{(\varphi\vee\psi)}$$

(3.87) $$(\mathbf{E}\vee)\ \frac{(\varphi\vee\psi) \quad \begin{array}{c}[\varphi]\\ \vdots\\ \chi\end{array} \quad \begin{array}{c}[\psi]\\ \vdots\\ \chi\end{array}}{\chi}$$

In the last rule, we have three assumptions. As we have indicated, whenever it is used, we may check mark any number of assumptions of the form φ in the second subtree and any number of assumptions of the form ψ in the third.

We shall give a characterization of natural deduction trees. A **finitary rule** is a pair $\rho = \langle\{\chi_i[A_i] : i < n\}, \varphi\rangle$, where for $i < n$, χ_i is a formula, A_i a finite set of formulae and φ a single formula. A **natural deduction calculus** $\mathfrak{N}$ is a set of finitary rules. A **proof tree for** $\mathfrak{N}$ is a quadruple $\mathbb{T} = \langle T, \succ, \ell, \mathcal{C}\rangle$ such that $\langle T, \prec\rangle$ is a tree, $\mathcal{C} \subseteq T$ a set of leaves and $\mathbb{T}$ is derived in the following way. (Think of $\mathcal{C}$ as the set of leaves carrying discharged assumptions.)

- ☞ $\mathbb{T} = \langle \{x\}, \varnothing, \ell, \varnothing \rangle$, where $\ell\colon x \mapsto \varphi$.
- ☞ There is a rule $\langle \{\chi_i[A_i] : i < n\}, \gamma \rangle$, and $\mathbb{T}$ is formed from trees $\mathbb{S}_i$, $i < n$, with roots s_i, by adding a new root node r, such that $\ell_{\mathbb{S}_i}(y_i) = \chi_i$, $i < n$, $\ell_{\mathbb{T}}(x) = \gamma$. Further, $\mathcal{C}_{\mathbb{T}} = \bigcup_{i<n} \mathcal{C}_{\mathbb{S}_i} \cup \bigcup_{i<n} N_i$, where N_i is a set of leaves of $\mathbb{S}_i$ such that for all $i < n$ and all $x \in N_i$: $\ell_{\mathbb{S}_i}(x) \in A_i$.

(Notice that the second case includes $n = 0$, in which case $\mathbb{T} = \langle \{x\}, \varnothing, \ell, \{x\} \rangle$ where $\ell(x)$ is simply an axiom.) We say that $\mathbb{T}$ **proves** $\ell(r)$ **in** $\mathfrak{N}$ from $\{\ell(x) : x \text{ leaf}, x \notin \mathcal{C}\}$. Here now is a proof tree ending in (a0).

$$\text{(3.88)} \qquad \dfrac{\dfrac{\varphi^{\surd}}{(\psi \to \varphi)}}{(\varphi \to (\psi \to \varphi))}$$

Further, here is a proof tree ending in (a1).

$$\text{(3.89)} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(\varphi \to (\psi \to \chi))^{\surd} \quad \varphi^{\surd}}{(\psi \to \chi)} \quad \dfrac{(\varphi \to \psi)^{\surd} \quad \varphi^{\surd}}{\psi}}{\chi}}{(\varphi \to \chi)}}{((\varphi \to \psi) \to (\varphi \to \chi))}}{((\varphi \to (\psi \to \chi)) \to ((\varphi \to \psi) \to (\varphi \to \chi)))}$$

A formula depends on all its assumptions that have not been retracted in the following sense.

Lemma 3.35 *Let $\mathbb{T}$ be a natural deduction tree with root x. Let Δ be the set of all formulae ψ such that $\langle y, \psi \rangle$ is an unretracted assumption of x and let $\varphi := \ell(x)$. Then $\Delta \vdash^{\mathsf{H}} \varphi$.*

Proof. By induction on the derivation of the proof tree. □

The converse also holds. If $\Delta \vdash^{\mathsf{H}} \varphi$ then there is a natural deduction proof for φ with Δ the set of unretracted assumptions (this is Exercise 99).

Notes on this section. Proofs are graphs whose labels are sequents. The procedure that eliminates cuts can be described using a graph grammar. Unfortunately, the replacements also manipulate the labels (that is, the sequents), so either one uses infinitely many rules or one uses schematic rules.

Exercise 94. Show (a) $(\varphi \to (\psi \to \chi)) \vdash^{\mathsf{B}} (\psi \to (\varphi \to \chi))$ and (b) $(\varphi \wedge \psi) \vdash^{\mathsf{H}'} (\psi \wedge \varphi)$, where H' is H with the axioms for $\wedge$ added.

Exercise 95. Show that a set Σ is inconsistent iff for every φ: $\Sigma \vdash^{\mathsf{B}} \varphi$.

Exercise 96. Show that a Hilbert style calculus satisfies DT for $\rightarrow$ iff the formulae (a0) and (a1) are derivable in it. (So, if we add, for example, the connectives $\neg$, $\wedge$ and $\vee$ together with the corresponding axioms, DT remains valid.)

Exercise 97. Define $\varphi \approx \psi$ by $\varphi \vdash^{\mathsf{H}} \psi$ and $\psi \vdash^{\mathsf{H}} \varphi$. Show that if $\varphi \approx \psi$ then (a) for all Δ and χ: $\Delta; \varphi \vdash^{\mathsf{H}} \chi$ iff $\Delta; \psi \vdash^{\mathsf{H}} \chi$, and (b) for all Δ: $\Delta \vdash^{\mathsf{H}} \varphi$ iff $\Delta \vdash^{\mathsf{H}} \psi$.

Exercise 98. Let us call Int the Hilbert calculus for $\rightarrow$, $\bot$, $\neg$, $\vee$ and $\wedge$. Further, call the Gentzen calculus for these connectives $\mathcal{I}$. Show that $\Delta \vdash^{\mathsf{Int}} \varphi$ iff $\overset{\mathcal{I}}{\rightsquigarrow} \Delta \vdash \varphi$.

Exercise 99. Show the following claim: *If* $\Delta \vdash^{\mathsf{H}} \varphi$ *then there is a natural deduction proof for* φ *with* Δ *the set of unretracted assumptions*.

Exercise 100. Show that the rule of *Modus Tollens* is admissible in the natural deduction calculus defined above (with added negation).

(3.90) Modus Tollens: $$\frac{(\varphi \rightarrow \psi) \qquad (\neg \psi)}{(\neg \varphi)}$$

3. Basics of λ–Calculus and Combinatory Logic

There is a fundamental difference between a term and a function. The *term* $x^2 + 2xy$ is something that has a concrete value if x and y have a concrete value. For example, if x has value 5 and y has value 2 then $x^2 + 2xy = 25 + 20 = 45$. However, the function $f\colon \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}\colon \langle x, y\rangle \mapsto x^2 + 2xy$ does not need any values for x and y. It only needs a pair of *numbers* to yield a value. That we have used variables to define f is of no concern here. We would have obtained the same function had we written $f\colon \langle x, u\rangle \mapsto x^2 + 2xu$. However, the term $x^2 + 2xu$ is different from the term $x^2 + 2xy$. For if u has value 3, x has value 5 and y value 2, then $x^2 + 2xu = 25 + 30 = 55$, while $x^2 + 2xy = 45$. To accommodate this difference, the λ–calculus has been developed. The λ–calculus allows to define functions from terms. In the case above we may write f as

(3.91) $$f := \lambda xy.x^2 + 2xy$$

This expression defines a function f and by saying what it does to its arguments. The prefix 'λxy' means that we are dealing with a function from pairs $\langle m,n\rangle$ and that the function assigns this pair the value m^2+2mn. This is the same as what we have expressed with $\langle x,y\rangle \mapsto x^2+2xy$. Now we can also define the following functions.

$$\lambda x.\lambda y.x^2+2xy, \qquad \lambda y.\lambda x.x^2+2xy \tag{3.92}$$

The first is a function which assigns to every number m the function $\lambda y.m^2+2my$; the latter yields the value m^2+2mn for every n. The second is a function which gives for every m the function $\lambda x.x^2+2xm$; this in turn yields n^2+2nm for every n. Since in general $m^2+2mn \neq n^2+2nm$, these two functions are different.

In λ–calculus one usually does not make use of the simultaneous abstraction of several variables, so one only allows prefixes of the form 'λx', not those of the form 'λxy'. This we shall also do here. We shall give a general definition of λ–terms. Anyhow, by introducing pairing and projection (see Section 3.6) simultaneous abstraction can be defined. The alphabet consists of a set F of function symbols (for which a signature Ω needs to be given as well), λ, the variables $V := \{\mathrm{x}_i : i \in \omega\}$ the brackets (,) and the period '.'.

Definition 3.36 *The set of* ***λ–terms over the signature*** Ω*, the set of* ***$\lambda\Omega$–terms*** *for short, is the smallest set* $\mathrm{Tm}_{\lambda\Omega}(V)$ *for which the following holds:*

① *Every* Ω*–term is in* $\mathrm{Tm}_{\lambda\Omega}(V)$.

② *If* $M,N \in \mathrm{Tm}_{\lambda\Omega}(V)$ *then also* $(MN) \in \mathrm{Tm}_{\lambda\Omega}(V)$.

③ *If* $M \in \mathrm{Tm}_{\lambda\Omega}$ *and* x *is a variable then* $(\lambda x.M) \in \mathrm{Tm}_{\lambda\Omega}(V)$.

If the signature is empty or clear from the context we shall simply speak of ***λ–terms***.

Since in ② we do not write an operator symbol, Polish Notation is now ambiguous. Therefore we follow standard usage and use the brackets (and). We agree now that x, y and z and so on are metavariables for variables (that is, for elements of V). Furthermore, upper case Roman letters like M, N are metavariables for λ–terms. One usually takes F to be $\varnothing$, to concentrate on the essentials of functional abstraction. If $F=\varnothing$, we speak of **pure λ–terms**. It is customary to omit the brackets if the term is bracketed to the left. Hence $MNOP$ is short for $(((MN)O)P)$ and $\lambda x.MN$ short for $((\lambda x.(MN))$ (and

distinct from $((\lambda x.M)N))$. However, this abbreviation has to be used with care since the brackets are symbols of the language. Hence $\mathtt{x}_0\mathtt{x}_0\mathtt{x}_0$ is not a string of the language but only a shorthand for $((\mathtt{x}_0\mathtt{x}_0)\mathtt{x}_0)$, a difference that we shall ignore after a while. Likewise, outer brackets are often omitted and brackets are not stacked when several λ–prefixes appear. Notice that $(\mathtt{x}_0\mathtt{x}_0)$ is a term. It denotes the application of $\mathtt{x}_0$ to itself. We have defined occurrences of a string $\vec{x}$ in a string $\vec{y}$ as contexts $\langle\vec{u},\vec{v}\rangle$ where $\vec{u}\vec{x}\vec{v}=\vec{y}$. Ω–terms are thought to be written down in Polish Notation.

Definition 3.37 *Let x be a variable. We define the set of* ***occurrences of*** *x in a λΩ–term inductively as follows.*

① *If M is an Ω–term then the set of occurrences of x in the λΩ–term M is the set of occurrences of the variable x in the Ω–term M.*

② *The set of occurrences of x in* (MN) *is the union of the set of pairs* $\langle(\vec{u},\vec{v}N)\rangle$*, where* $\langle\vec{u},\vec{v}\rangle$ *is an occurrence of x in M and the set of pairs* $\langle(M\vec{u},\vec{v})\rangle$*, where* $\langle\vec{u},\vec{v}\rangle$ *is an occurrence of x in N.*

③ *The set of occurrences of x in* $(\lambda x.M)$ *is the set of all* $\langle(\lambda x.\vec{u},\vec{v})\rangle$*, where* $\langle\vec{u},\vec{v}\rangle$ *is an occurrence of x in M.*

So notice that — technically speaking — the occurrence of the string x in the λ–prefix of $(\lambda x.M)$ is not an occurrence of the variable x. Hence $\mathtt{x}_0$ does not occur in $(\lambda x_0.\mathtt{x}_1)$ as a λΩ–term although it does occur in it as a string!

Definition 3.38 *Let M be a λ–term, x a variable and C an occurrence of x in M. C is a* ***free occurrence of*** *x* ***in*** *M if C is not inside a term of the form* $(\lambda x.N)$ *for some N; if C is not free, it is called* ***bound****. A λ–term is called* ***closed*** *if no variable occurs free in it. The set of all variables having a free occurrence in M is denoted by* $\mathrm{fr}(M)$.

A few examples shall illustrate this. In $M=(\lambda\mathtt{x}_0.(\mathtt{x}_0\mathtt{x}_1))$ the variable $\mathtt{x}_0$ occurs only bound, since it only occurs inside a subterm of the form $(\lambda\mathtt{x}_0.N)$ (for example $N:=(\mathtt{x}_0\mathtt{x}_1)$). However, $\mathtt{x}_1$ occurs free. A variable may occur free as well as bound in a term. An example is the variable $\mathtt{x}_0$ in $(\mathtt{x}_0(\lambda\mathtt{x}_0.\mathtt{x}_0))$.

Bound and free variable occurrences behave differently under replacement. If M is a λ–term and x a variable then denote by $[N/x]M$ the result

of replacing x by N. In this replacement we do not simply replace all occurrences of x by N; the definition of replacement requires some care.

(3.93a) $$[N/x]y := \begin{cases} N & \text{if } x = y, \\ y & \text{otherwise.} \end{cases}$$

(3.93b) $$[N/x]f(\vec{s}) := f([N/x]s_0, \ldots, [N/x]s_{\Omega(f)-1})$$

(3.93c) $$[N/x](MM') := ([N/x]M)([N/x]M'))$$

(3.93d) $$[N/x](\lambda x.M) := (\lambda x.M)$$

(3.93e) $$[N/x](\lambda y.M) := (\lambda y.[N/x]M)$$
$$\text{if } y \neq x \text{ and: } y \notin \mathrm{fr}(N) \text{ or } x \notin \mathrm{fr}(M)$$

(3.93f) $$[N/x](\lambda y.M) := (\lambda z.[N/x][z/y]M)$$
$$\text{if } y \neq x, y \in \mathrm{fr}(N) \text{ and } x \in \mathrm{fr}(M)$$

In (3.93f) we have to choose z in such a way that it does not occur freely in N or M. In order for substitution to be uniquely defined we assume that $z = \mathtt{x}_i$, where i is the least number such that z satisfies the conditions. The precaution in (3.93f) of an additional substitution is necessary. For let $y = \mathtt{x}_1$ and $M = \mathtt{x}_0$. Then without this substitution we would get

(3.94) $$[\mathtt{x}_1/\mathtt{x}_0](\lambda \mathtt{x}_0.\mathtt{x}_1) = (\lambda \mathtt{x}_1.[\mathtt{x}_1/\mathtt{x}_0]\mathtt{x}_0) = (\lambda \mathtt{x}_1.\mathtt{x}_1)$$

This is clearly incorrect. For $(\lambda \mathtt{x}_1.\mathtt{x}_0)$ is the function which for given a returns the value of $\mathtt{x}_0$. However, $(\lambda \mathtt{x}_1.\mathtt{x}_1)$ is the identity function and so it is different from that function. Now the substitution of a variable by another variable shall not change the course of values of a function.

(3.95a) $M = M$

(3.95b) $M = N \Rightarrow N = M$

(3.95c) $M = N, N = L \Rightarrow M = L$

(3.95d) $M = N \Rightarrow (ML) = (NL)$

(3.95e) $M = N \Rightarrow (LM) = (LN)$

(3.95f) $(\lambda x.M) = (\lambda y.[y/x]M) \quad y \notin \mathrm{fr}(M)$ (α–conversion)

(3.95g) $(\lambda x.M)N = [N/x]M$ (β–conversion)

(3.95h) $(\lambda x.M) = M \quad x \notin \mathrm{fr}(M)$ (η–conversion)

(3.95i) $M = N \Rightarrow (\lambda x.M) = (\lambda x.N)$ (ξ–rule)

We shall present the theory of λ–terms which we shall use in the sequel. It consists in a set of equations $M = N$, where M and N are terms. These are subject to the laws above. The theory axiomatized by (3.95a) – (3.95g) and (3.95i) is called λ, the theory axiomatized by (3.95a) –(3.95i) $\lambda\eta$. Notice that (3.95a) – (3.95e) simply say that $=$ is a congruence. A different rule is the following so–called **extensionality rule**.

$$Mx = Nx \Rightarrow M = N \qquad \text{(ext)} \tag{3.96}$$

It can be shown that $\lambda + (\text{ext}) = \lambda\eta$. The model theory of λ–calculus is somewhat tricky. Basically, all that is assumed is that we have a domain D together with a binary operation $\bullet$ that interprets function application. Abstraction is defined implicitly. Call a function $\beta : V \to D$ a **valuation**. Now define $[M]^\beta$ inductively as follows.

$$[\mathtt{x}_i]^\beta := \beta(\mathtt{x}_i) \tag{3.97a}$$

$$[(MN)]^\beta := [M]^\beta([N]^\beta) \tag{3.97b}$$

$$[(\lambda x.M)]^\beta \bullet a := [M]^{\beta[x:=a]} \tag{3.97c}$$

(Here, $a \in D$.) (3.97c) does not fix the interpretation of $(\lambda x.M)$ uniquely on the basis of the interpretation of M. If it does, however, the structure is called *extensional*. We shall return to that issue below. First we shall develop some more syntactic techniques for dealing with λ–terms.

Definition 3.39 *Let M and N be λ–terms. We say, N* ***is obtained from*** *M* ***by replacement of bound variables*** *or by* ***α–conversion*** *and write $M \rightsquigarrow_\alpha N$ if there is a subterm $(\lambda y.L)$ of M and a variable z which does not occur in L such that N is the result of replacing an occurrence of $(\lambda y.L)$ by $(\lambda z.[z/y]L)$. The relation $\rhd_\alpha$ is the transitive closure of $\rightsquigarrow_\alpha$. N is* ***congruent to*** *M, in symbols $M \equiv_\alpha N$, if both $M \rhd_\alpha N$ and $N \rhd_\alpha M$.*

Similarly the definition of β–conversion.

Definition 3.40 *Let M be a λ–term. We write $M \rightsquigarrow_\beta N$ and say that M* ***contracts to*** *N if N is the result of a single replacement of an occurrence of $((\lambda x.L)P)$ in M by $([P/x]L)$. Further, we write $M \rhd_\beta N$ if N results from M by a series of contractions and $M \equiv_\beta N$ if $M \rhd_\beta N$ and $N \rhd_\beta M$.*

A term of the form $((\lambda x.M)N)$ is called a **redex** and $[N/x]M$ its **contractum**. The step from the redex to the contractum represents the evaluation of

a function to its argument. A λ–term is **evaluated** or **in normal form** if it contains no redex.

Similarly for the notation $\rightsquigarrow_{\alpha\beta}$, $\triangleright_{\alpha\beta}$ and $\equiv_{\alpha\beta}$. Call M and N $\alpha\beta$–**equivalent** ($\alpha\beta\eta$–**equivalent**) if $\langle M,N\rangle$ is contained in the least equivalence relation containing $\triangleright_{\alpha\beta}$ ($\triangleright_{\alpha\beta\eta}$).

Proposition 3.41 $\lambda \vdash M = N$ *iff* M *and* N *are* $\alpha\beta$*–equivalent.* $\lambda\eta \vdash M = N$ *iff* M *and* N *are* $\alpha\beta\eta$*–equivalent.*

If $M \triangleright_{\alpha\beta} N$ and N is in normal form then N is called a **normal form of** M. Without proof we state the following theorem.

Theorem 3.42 (Church, Rosser) *Let* L,M,N *be* λ*–terms such that* $L \triangleright_{\alpha\beta} M$ *and* $L \triangleright_{\alpha\beta} N$*. Then there exists a* P *such that* $M \triangleright_{\alpha\beta} P$ *and* $N \triangleright_{\alpha\beta} P$*.*

The proof can be found in all books on the λ–calculus. This theorem also holds for $\triangleright_{\alpha\beta\eta}$.

Corollary 3.43 *Let* N *and* N' *be normal forms of* M*. Then* $N \equiv_{\alpha} N'$*.*

The proof is simple. For by the previous theorem there exists a P such that $N \triangleright_{\alpha\beta} P$ and $N' \triangleright_{\alpha\beta} P$. But since N as well as N' do not contain any redex and α–conversion does not introduce any redexes then P results from N and N' by α–conversion. Hence P is α–congruent with N and N' and hence N and N' are α–congruent.

Not every λ–term has a normal form. For example

(3.98)
$$\begin{aligned} &((\lambda x_0.(x_0x_0))(\lambda x_0.(x_0x_0))) \\ \triangleright_{\beta}\ &((\lambda x_0.(x_0x_0))(\lambda x_0.(x_0x_0))) \end{aligned}$$

Or

(3.99)
$$\begin{aligned} &((\lambda x_0.((x_0x_0)x_1))(\lambda x_0.((x_0x_0)x_1))) \\ \triangleright_{\beta}\ &(((\lambda x_0.((x_0x_0)x_1))(\lambda x_0.((x_0x_0)x_1)))x_1) \\ \triangleright_{\beta}\ &((((\lambda x_0.((x_0x_0)x_1))(\lambda x_0.((x_0x_0)x_1)))x_1)x_1) \end{aligned}$$

The typed λ–calculus differs from the calculus which has just been presented by an important restriction, namely that every term must have a type.

Definition 3.44 *Let* B *be a set. The set of* ***types over*** B*,* $\mathrm{Typ}_{\rightarrow}(B)$*, is the smallest set* M *for which the following holds.*

① $B \subseteq M$.

② *If* $\alpha \in M$ *and* $\beta \in M$ *then* $\alpha \to \beta \in M$.

In other words: types are simply terms in the signature $\{\to\}$ with $\Omega(\to) = 2$ over a set of basic types. Each term is associated with a type and the structure of terms is restricted by the type assignment. Further, all Ω–terms are admitted. Their type is already fixed. The following rules are valid.

① If (MN) is a term of type γ then there is a type α such that M has the type $\alpha \to \gamma$ and N the type γ.

② If M has the type γ and x_α is a variable of type α then $(\lambda x_\alpha . M)$ is of type $\alpha \to \gamma$.

Notice that for every type α there are countably many variables of type α. More exactly, the set of variables of type α is $V^\alpha := \{\mathtt{x}^i_\alpha : i \in \omega\}$. We shall often use the metavariables x_α, y_α and so on. If $\alpha \neq \beta$ then also $x_\alpha \neq x_\beta$ (they represent different variables). With these conditions the formation of λ–terms is severely restricted. For example $(\lambda \mathtt{x}_0 . (\mathtt{x}_0 \mathtt{x}_0))$ is not a typed term no matter which type $\mathtt{x}_0$ has. One can show that a typed term always has a normal form. This is in fact an easy matter. Notice by the way that if the term $(\mathtt{x}_0 + \mathtt{x}_1)$ has type α and $\mathtt{x}_0$ and $\mathtt{x}_1$ also have the type α, the function $(\lambda \mathtt{x}_0 . (\lambda \mathtt{x}_1 . (\mathtt{x}_0 + \mathtt{x}_1)))$ has the type $\alpha \to (\alpha \to \alpha)$. The type of an Ω–term is the type of its value, in this case α. The types are nothing but a special version of *sorts*. Simply take $\mathrm{Typ}_\to(B)$ to be the set of sorts. However, while application (written $\bullet$) is a single symbol in the typed λ–calculus, we must now assume in place of it a family of symbols $\bullet^\beta_\alpha$ of signature $\langle \alpha \to \beta, \alpha, \beta \rangle$ for every type α, β. Namely, $M \bullet^\beta_\alpha N$ is defined iff M has type $\alpha \to \beta$ and N type α, and the result is of sort (= type) β. While the notation within many sorted algebras can get clumsy, the techniques (ultimately derived from the theory of unsorted algebras) are very useful, so the connection is very important for us. Notice that algebraically speaking it is not λ but $\lambda \mathtt{x}_\alpha$ that is a member of the signature, and once again, in the many sorted framework, $\lambda \mathtt{x}_\alpha$ turns into a family of operations $\lambda^\beta \mathtt{x}_\alpha$ of sort $\langle \beta, \alpha \to \beta \rangle$. That is to say, $\lambda^\beta \mathtt{x}_\alpha$ is a function symbol that only forms a term with an argument of sort (= type) β and yields a term of type $\alpha \to \beta$.

We shall now present a model of the λ–calculus. We begin by studying the purely applicative structures and then turn to abstraction after the introduction of combinators. In the untyped case application is a function that is

everywhere defined. The model structures are therefore so–called *applicative structures*.

Definition 3.45 *An **applicative structure** is a pair $\mathfrak{A} = \langle A, \bullet \rangle$ where $\bullet$ is a binary operation on A. If $\bullet$ is only a partial operation, $\langle A, \bullet \rangle$ is called a **partial applicative structure**. $\mathfrak{A}$ is called **extensional** if for all $a, b \in A$:*

(3.100) $\quad a = b$ *iff for all* $c \in A : a \bullet c = b \bullet c$

Definition 3.46 *A **typed applicative structure** over a given set of basic types B is a structure $\langle \{A_\alpha : \alpha \in \mathrm{Typ}_{\rightarrow}(B)\}, \bullet \rangle$ such that (a) A_α is a set for every α, (b) $A_\alpha \cap A_\beta = \varnothing$ if $\alpha \neq \beta$ and (c) $a \bullet b$ is defined iff there are types $\alpha \rightarrow \beta$ and α such that $a \in A_{\alpha \rightarrow \beta}$ and $b \in A_\alpha$, and then $a \bullet b \in A_\beta$.*

A typed applicative structure defines a partial applicative structure. Namely, put $A := \bigcup_\alpha A_\alpha$; then $\bullet$ is nothing but a partial binary operation on A. The typing is then left implicit. (Recovering the types of elements is not a trivial affair, see the exercises.) Not every partial applicative structure can be typed, though.

One important type of models are those where A consists of sets and $\bullet$ is the usual functional application as defined in sets. More precisely, we want that A_α is a set of sets for every α. So if the type is associated with the set S then a variable may assume as value any member of S. So, it follows that if β is associated with the set T and M has the type β then the interpretation of $(\lambda \mathtt{x}_\alpha.M)$ is a function from S to T. We set the realization of $\alpha \rightarrow \beta$ to be the set of all functions from S to T. This is an arbitrary choice, a different choice (for example a suitable subset) would do as well.

Let M and N be sets. Then a function from M to N is a subset F of the cartesian product $M \times N$ which satisfies certain conditions (see Section 1.1). Namely, for every $x \in M$ there must be a $y \in N$ such that $\langle x, y \rangle \in F$ and if $\langle x, y \rangle \in F$ and $\langle x, y' \rangle \in F$ then $y = y'$. (For partial functions the first condition is omitted. Everything else remains. For simplicity we shall deal only with totally defined functions.) Normally one thinks of a function as something that gives values for certain arguments. This is not so in this case. F is not a function in this sense, it is just the **graph** of a function. In set theory one does not distinguish between a function and its graph. We shall return to this later. How do we have to picture F as a set? Recall that we have defined

(3.101) $\quad M \times N = \{\langle x, y \rangle : x \in M, y \in N\}$

This is a set. Notice that $M \times (N \times O) \neq (M \times N) \times O$. However, the mapping

$$\rtimes : \langle x, \langle y, z \rangle \rangle \mapsto \langle \langle x, y \rangle, z \rangle : M \times (N \times O) \to (M \times N) \times O \tag{3.102}$$

is a bijection. Its inverse is the mapping

$$\ltimes : \langle \langle x, y \rangle, z \rangle \mapsto \langle x, \langle y, z \rangle \rangle : (M \times N) \times O \to M \times (N \times O) \tag{3.103}$$

Finally we put

$$M \to N := \{F \subseteq M \times N : F \text{ a function } \} \tag{3.104}$$

Elsewhere we have used the notation N^M for that set. Now functions are also sets and their arguments are sets, too. Hence we need a map which applies a function to an argument. Since it must be defined for all cases of functions and arguments, it must by necessity be a partial function. If x is a function and y an arbitrary object, we define $\mathsf{app}(x, y)$ as follows.

$$\mathsf{app}(x, y) := \begin{cases} z & \text{if } \langle y, z \rangle \in x, \\ \star & \text{if no } z \text{ exists such that } \langle y, z \rangle \in x. \end{cases} \tag{3.105}$$

app is a partial function. Its graph in the universe of sets is a proper class, however. It is the class of pairs $\langle \langle F, x \rangle, y \rangle$, where F is a function and $\langle x, y \rangle \in F$.

Note that if $F \in M \to (N \to O)$ then

$$F \subseteq M \times (N \to O) \subseteq M \times (N \times O) \tag{3.106}$$

Then $\rtimes[F] \subseteq (M \times N) \times O$, and one calculates that $\rtimes[F] \subseteq (M \times N) \to O$. In this way a unary function with values in $N \to O$ becomes a unary function from $M \times N$ to O (or a binary function from M, N to O). Conversely, one can see that if $F \in (M \times N) \to O$ then $\ltimes[F] \in M \to (N \to O)$.

Theorem 3.47 *Let V_ω be the set of finite sets. Then $\langle V_\omega, \mathsf{app} \rangle$ is a partial applicative structure.*

In place of V_ω one can take any V_κ where κ is an ordinal. However, only if κ is a limit ordinal (that is, an ordinal without predecessor), the structure will be combinatorially complete. A more general result is described in the following theorem for the typed calculus. Its proof is straightforward.

Theorem 3.48 *Let B be the set of basic types and M_b, $b \in B$, arbitrary sets. Let M_α be inductively defined by $M_{\alpha\to\beta} := (M_\beta)^{M_\alpha}$. Then*

(3.107) $\langle\{M_\alpha : \alpha \in \mathrm{Typ}_\to(B)\}, \mathtt{app}\rangle$

is a typed applicative structure. Moreover, it is extensional.

For a proof of this theorem one simply has to check the conditions.

In categorial grammar, with which we shall deal in this chapter, we shall use λ–terms to name meanings for symbols and strings. It is important however that the λ–term is only a formal entity (namely a certain string), and it is not the meaning in the proper sense of the word. To give an example, $(\lambda \mathtt{x}_0 . (\lambda \mathtt{x}_1 . \mathtt{x}_0 + \mathtt{x}_1))$ is a string which names a function. In the set universe, this function is a subset of $\mathbb{N} \to (\mathbb{N} \to \mathbb{N})$. For this reason one has to distinguish between equality $=$ and the symbol(s) $\equiv$/$=$. $M = N$ means that we are dealing with the same strings (hence literally the same λ–terms) while $M \equiv N$ means that M and N name the same function. In this sense $(\lambda \mathtt{x}_0 . (\lambda \mathtt{x}_1 . \mathtt{x}_0 + \mathtt{x}_1))(\mathtt{x}_0)(\mathtt{x}_2) \neq \mathtt{x}_0 + \mathtt{x}_2$, but they also denote the same value. Nevertheless, in what is to follow we shall not always distinguish between a λ–term and its interpretation, in order not to make the notation too opaque.

The λ–calculus has a very big disadvantage, namely that it requires some caution in dealing with variables. However, there is a way to avoid having to use variables. This is achieved through the use of combinators. Given a set V of variables and the zeroary constants $\mathtt{S}$, $\mathtt{K}$, $\mathtt{I}$, combinators are terms over the signature that has only one more binary symbol, $\bullet$. This symbol is generally omitted, and terms are formed using infix notation with brackets. Call this signature Γ.

Definition 3.49 *An element of* $\mathrm{Tm}_\Gamma(V)$ *is called a* ***combinatorial term****. A* ***combinator*** *is an element of* $\mathrm{Tm}_\Gamma(\varnothing)$.

Further, the redex relation $\triangleright$ is defined as follows.

(3.108a) $\mathtt{I}X \triangleright X$

(3.108b) $\mathtt{K}XY \triangleright X$

(3.108c) $\mathtt{S}XYZ \triangleright XZ(YZ)$

(3.108d) $X \triangleright X$

(3.108e) if $X \triangleright Y$ and $Y \triangleright Z$ then $X \triangleright Z$

(3.108f) if $X \triangleright Y$ then $(XZ) \triangleright (YZ)$

(3.108g) if $X \triangleright Y$ then $(ZX) \triangleright (ZY)$

Combinatory logic (CL) is (3.108a) – (3.108e). It is an equational theory if we read $\triangleright$ simply as equality. (The only difference is that $\triangleright$ is not symmetric. So, to be exact, the rule 'if $X = Y$ then $Y = X$' needs to be added.) We note that there is a combinator C containing only $\mathtt{K}$ and $\mathtt{S}$ such that $\mathsf{C} \triangleright \mathtt{I}$ (see Exercise 104). This explains why $\mathtt{I}$ is sometimes omitted.

We shall now show that combinators can be defined by λ–terms and vice versa. First, define

(3.109a) $\mathsf{I} := (\lambda \mathtt{x}_0 . \mathtt{x}_0)$

(3.109b) $\mathsf{K} := (\lambda \mathtt{x}_0 . (\lambda \mathtt{x}_1 . \mathtt{x}_0))$

(3.109c) $\mathsf{S} := (\lambda \mathtt{x}_0 . (\lambda \mathtt{x}_1 . (\lambda \mathtt{x}_2 . \mathtt{x}_0 \mathtt{x}_2 (\mathtt{x}_1 \mathtt{x}_2))))$

Define a translation $^\lambda$ by $X^\lambda := X$ for $X \in V$, $\mathtt{S}^\lambda := \mathsf{S}$, $\mathtt{K}^\lambda := \mathsf{K}$, $\mathtt{I}^\lambda := \mathsf{I}$. Then the following is proved by induction on the length of the proof.

Theorem 3.50 *Let C and D be combinators. If $C \triangleright D$ then $C^\lambda \triangleright_\beta D^\lambda$. Also, if $\mathsf{CL} \vdash C = D$ then $\lambda \vdash C^\lambda = D^\lambda$.*

The converse translation is more difficult. We shall define first a function $[x]$ on combinatory terms. (Notice that there are no bound variables, so $\mathrm{var}(M) = \mathrm{fr}(M)$ for any combinatorial term M.)

(3.110a) $[x]x := \mathtt{I}.$

(3.110b) $[x]M := \mathtt{K}M,$ if $x \notin \mathrm{var}(M)$.

(3.110c) $[x]Mx := M,$ if $x \notin \mathrm{var}(M)$.

(3.110d) $[x]MN := \mathtt{S}([x]M)([x]N),$ otherwise.

(So, (3.110d) is applied only if (3.110b) and (3.110c) cannot be applied.) For example $[\mathtt{x}_1]\mathtt{x}_1\mathtt{x}_0 = \mathtt{S}([\mathtt{x}_1]\mathtt{x}_1)([\mathtt{x}_1]\mathtt{x}_0) = \mathtt{SI}(\mathtt{K}\mathtt{x}_0)$. Indeed, if one applies this to $\mathtt{x}_1$, then one gets

(3.111) $\mathtt{SI}(\mathtt{K}\mathtt{x}_0)\mathtt{x}_1 \triangleright \mathtt{I}\mathtt{x}_1(\mathtt{K}\mathtt{x}_0\mathtt{x}_1) \triangleright \mathtt{x}_1(\mathtt{K}\mathtt{x}_0\mathtt{x}_1) \triangleright \mathtt{x}_1\mathtt{x}_0$

Further, one has

(3.112) $\mathsf{U} := [\mathtt{x}_1]([\mathtt{x}_0]\mathtt{x}_1\mathtt{x}_0) = [\mathtt{x}_1]\mathtt{SI}(\mathtt{K}\mathtt{x}_0) = \mathtt{S}(\mathtt{K}(\mathtt{SI}))\mathtt{K}$

The reader may verify that $\mathsf{U}\mathtt{x}_0\mathtt{x}_1 \triangleright \mathtt{x}_1\mathtt{x}_0$. Now define $^\kappa$ by $x^\kappa := x$, $x \in V$, $(MN)^\kappa := (M^\kappa N^\kappa)$ and $(\lambda x.N)^\kappa := [x]N^\kappa$.

Theorem 3.51 *Let C be a closed λ–term. Then $\lambda \vdash C = C^{\kappa}$.*

Now we have defined translations from λ–terms to combinators and back. It can be shown, however, that the theory λ is stronger than CL under translation. Curry found a list $\mathbf{A}_{\beta}$ of five equations such that λ is as strong as $\mathsf{CL} + \mathbf{A}_{\beta}$ in the sense of Theorem 3.52 below. Also, he gave a list $\mathbf{A}_{\beta\eta}$ such that $\mathsf{CL} + \mathbf{A}_{\beta\eta}$ is equivalent to $\lambda\eta = \lambda + (\text{ext})$. $\mathbf{A}_{\beta\eta}$ also is equivalent to the first–order postulate (ext): $(\forall xy)((\forall z)(x \bullet z = y \bullet z) \to x = y)$.

Theorem 3.52 (Curry) *Let M and N be λ–terms.*

① *If $\lambda \vdash M = N$ then $\mathsf{CL} + \mathbf{A}_{\beta} \vdash M^{\kappa} = N^{\kappa}$.*

② *If $\lambda\eta \vdash M = N$ then $\mathsf{CL} + \mathbf{A}_{\beta\eta} \vdash M^{\kappa} = N^{\kappa}$.*

There is also a typed version of combinatorial logic. There are two basic approaches. The first is to define typed combinators. The basic combinators now split into infinitely many typed versions as follows.

(3.113)

Combinator	Type
$\mathtt{I}_{\alpha}$	$\alpha \to \alpha$
$\mathtt{K}_{\alpha,\beta}$	$\alpha \to (\beta \to \alpha)$
$\mathtt{S}_{\alpha,\beta,\gamma}$	$(\alpha \to (\beta \to \gamma)) \to ((\alpha \to \beta) \to (\alpha \to \gamma))$

Together with $\bullet$ they form the typed signature Γ^{τ}. For each type there are countably infinitely many variables of that type in V. **Typed combinatorial terms** are elements of $\mathrm{Tm}_{\Gamma^{\tau}}(V)$, and **typed combinators** are elements of $\mathrm{Tm}_{\Gamma^{\tau}}$. Further, if M is a combinator of type $\alpha \to \beta$ and N a combinator of type α then (MN) is a combinator of type β. In this way, every typed combinatorial term has a unique type.

The second approach is to keep the symbols $\mathtt{I}$, $\mathtt{K}$ and $\mathtt{S}$ and to let them stand for any of the above typed combinators. In terms of functions, $\mathtt{I}$ takes an argument N of any type α and returns N (of type α). Likewise, $\mathtt{K}$ is defined on any M, N of type α and β, respectively, and $\mathtt{K}MN = M$ of type α. Also, $\mathtt{K}M$ is defined and of type $\beta \to \alpha$. Basically, the language is the same as in the untyped case. A combinatorial term is **stratified** if for each variable and each occurrence of $\mathtt{I}$, $\mathtt{K}$, $\mathtt{S}$ there exists a type such that if that (occurrence of the) symbol is assigned that type, the resulting string is a typed combinatorial term. (So, while each occurrence of $\mathtt{I}$, $\mathtt{K}$ and $\mathtt{S}$, respectively, may be given a different type, each occurrence of the same variable must have the same type.) For example, $\mathsf{B} := \mathtt{S}(\mathtt{KS})\mathtt{K}$ is stratified, while $\mathtt{SII}$ is not.

We show the second claim first. Suppose that there are types α, β, γ, δ, ε such that $((\mathtt{S}_{\alpha,\beta,\gamma}\mathtt{I}_\delta)\mathtt{I}_\varepsilon)$ is a typed combinator.

$$\text{(3.114)}\qquad \frac{\begin{array}{c}((\mathtt{S}_{\alpha,\beta,\gamma}\end{array} \quad \begin{array}{c}\mathtt{I}_\delta)\\ \delta\to\delta\end{array} \quad \begin{array}{c}\mathtt{I}_\varepsilon)\\ \varepsilon\to\varepsilon\end{array}}{\begin{array}{c}(\alpha\to(\beta\to\gamma))\\ \to((\alpha\to\beta)\to(\alpha\to\gamma))\end{array}}$$

Then, since $\mathtt{S}_{\alpha,\beta,\gamma}$ is applied to $\mathtt{I}_\delta$ we must have $\delta\to\delta=\alpha\to(\beta\to\gamma)$, whence $\alpha=(\beta\to\gamma)$. So, $(\mathtt{S}_{\alpha,\beta,\gamma}\mathtt{I}_\delta)$ has the type

$$\text{(3.115)}\qquad ((\beta\to\gamma)\to\beta)\to((\beta\to\gamma)\to\gamma)$$

This combinator is applied to $\mathtt{I}_\varepsilon$, and so we have $(\beta\to\gamma)\to\beta=\varepsilon\to\varepsilon$, whence $\beta\to\gamma=\varepsilon=\beta$, which is impossible. So, $\mathtt{SII}$ is not stratified. On the other hand, $\mathtt{B}$ is stratified. Assume types such that $\mathtt{S}_{\zeta,\eta,\theta}(\mathtt{K}_{\alpha,\beta}\mathtt{S}_{\gamma,\delta,\varepsilon})\mathtt{K}_{\iota,\kappa}$ is a typed combinator. First, $\mathtt{K}_{\alpha,\beta}$ is applied to $\mathtt{S}_{\gamma,\delta,\varepsilon}$. This means that

$$\text{(3.116)}\qquad \alpha=(\gamma\to(\delta\to\varepsilon))\to((\gamma\to\delta)\to(\gamma\to\varepsilon))$$

The result has type

$$\text{(3.117)}\qquad \beta\to((\gamma\to(\delta\to\varepsilon))\to((\gamma\to\delta)\to(\gamma\to\varepsilon)))$$

This is the argument of $\mathtt{S}_{\zeta,\eta,\theta}$. Hence we must have

$$\text{(3.118)}\qquad \begin{aligned}&\zeta\to(\eta\to\theta)\\ &\qquad=\beta\to((\gamma\to(\delta\to\varepsilon))\to((\gamma\to\delta)\to(\gamma\to\varepsilon)))\end{aligned}$$

So, $\zeta=\beta$, $\eta=\gamma\to(\delta\to\varepsilon)$, $\theta=(\gamma\to\delta)\to(\gamma\to\varepsilon)$. The resulting type is $(\zeta\to\eta)\to(\zeta\to\theta)$. This is applied to $\mathtt{K}_{\iota,\kappa}$ of type $\iota\to(\kappa\to\iota)$. For this to be well–defined we must have $\iota\to(\kappa\to\iota)=\zeta\to\eta$, or $\iota=\zeta=\beta$ and $\kappa\to\iota=\eta=\gamma\to(\delta\to\varepsilon)$. Finally, this results in $\kappa=\gamma$, $\iota=\beta=\delta\to\varepsilon$. So, α, γ, δ and ε may be freely chosen, and the other types are immediately defined.

It is the second approach that will be the most useful for us later on. We call combinators **implicitly typed** if they are thought of as typed in this way. (In fact, they simply are untyped terms.) The same can be done with λ–terms, giving rise to the notion of a stratified λ–term. In the sequel we shall not distinguish between combinators and their representing λ–terms.

Finally, let us return to the models of the λ–calculus. Recall that we have defined abstraction only implicitly, using Definition (3.97c) repeated below:

(3.119) $\quad [(\lambda x.M)]^{\beta} \bullet a := [M]^{\beta[x:=a]}$

In general, this object need not exist, in which case we do not have a model for the λ–calculus.

Definition 3.53 *An applicative structure $\mathfrak{A}$ is called* ***combinatorially complete*** *if for every term t in the language with free variables from $\{\mathtt{x}_i : i < n\}$ there exists a y such that for all $b_i \in A$, $i < n$:*

(3.120) $\quad (\cdots((y \bullet b_0) \bullet b_1) \bullet \cdots \bullet b_{n-1}) = t(b_0, \ldots, b_{n-1})$

This means that for every term t there exists an element which represents this term:

(3.121) $\quad (\lambda \mathtt{x}_0 . (\lambda \mathtt{x}_1 . \cdots . (\lambda \mathtt{x}_{n-1} . t(\mathtt{x}_0, \ldots, \mathtt{x}_{n-1})) \cdots))$

Thus, this defines the notion of an applicative structure in which every element can be abstracted. It is these structures that can serve as models of the λ–calculus. Still, no explicit way of generating the functions is provided. One way is to use countably many abstraction operations, one for every number $i < \omega$ (see Section 4.5). Another way is to translate λ–terms into combinatory logic using $[-]$ for abstraction. In view of the results obtained above we get the following result.

Theorem 3.54 (Schönfinkel) *$\mathfrak{A}$ is combinatorially complete iff there are elements k and s such that*

(3.122) $\quad ((k \bullet a) \bullet b) = a \qquad (((s \bullet a) \bullet b) \bullet c) = (a \bullet c) \bullet (b \bullet c)$

Definition 3.55 *A structure $\mathfrak{A} = \langle A, \bullet, \mathsf{k}, \mathsf{s} \rangle$ is called a* ***combinatory algebra*** *if $\mathfrak{A} \vDash \mathsf{k} \bullet x \bullet y = x, \mathsf{s} \bullet x \bullet y \bullet z = x \bullet z \bullet (y \bullet z)$. It is a* ***$\lambda$–algebra*** *(or* ***extensional****) if it satisfies $\mathbf{A}_{\beta}$ ($\mathbf{A}_{\beta\eta}$) in addition.*

So, the class of combinatory algebras is an equationally definable class. (This is why we have not required $|A| > 1$, as is often done.) Again, the partial case is interesting. Hence, we can use the theorems of Section 1.1 to create structures. Two models are of particular significance. One is based on the algebra of combinatorial terms over V modulo derivable identity, the other

is the algebra of combinators modulo derivable identity. Indirectly, this also shows how to create models for the λ–calculus. We shall explain a different method below in Section 4.5.

Call a structure $\langle A, \bullet, \mathsf{k}, \mathsf{s}\rangle$ a **partial combinatory algebra** if (i) $\mathsf{s} \bullet x \bullet y$ is always defined and (ii) the defining equations hold in the intermediate sense, that is, if one side is defined so is the other and they are equal (cf. Section 1.1). Consider once again the universe V_ω. Define

$$\mathfrak{k} := \{\langle x, \langle y, x\rangle\rangle : x, y \in V_\omega\} \tag{3.123}$$

$$\mathfrak{s} := \{\langle x, \langle y, \langle z, \mathsf{app}(\mathsf{app}(x,z), \mathsf{app}(y,z))\rangle\rangle : x, y, z \in V_\omega\} \tag{3.124}$$

$\langle V_\omega, \mathsf{app}, \mathfrak{k}, \mathfrak{s}\rangle$ is not a partial combinatory algebra because $\mathsf{app}(\mathsf{app}(\mathfrak{k}, x), y)$ is not always defined. So, the equation $(k \bullet x) \bullet y = x$ does not hold in the intermediate sense (since the right hand is obviously always defined). The defining equations hold only in the weak sense: if both sides are defined, then they are equal. Thus, V_ω is a useful model only in the typed case.

In the typed case we need a variety of combinators. More exactly: for all types α, β and γ we need elements $\mathsf{k}_\delta \in A_\delta$, $\delta = \alpha \to (\beta \to \alpha)$ and $\mathsf{s}_\eta \in A_\eta$, $\eta = (\alpha \to (\beta \to \gamma)) \to ((\alpha \to \beta) \to (\alpha \to \gamma))$ such that for all $a \in A_\alpha$ and $b \in A_\beta$ we have

$$(\mathsf{k}_\delta \bullet a) \bullet b = a \tag{3.125}$$

and for every $a \in A_{\alpha \to (\beta \to \gamma)}$, $b \in A_{\alpha \to \beta}$ and $c \in A_\alpha$ we have

$$((\mathsf{s}_\eta \bullet a) \bullet b) \bullet c = (a \bullet c) \bullet (b \bullet c) \tag{3.126}$$

We now turn to an interesting connection between intuitionistic logic and type theory, known as the *Curry–Howard–Isomorphism*. Write $M : \varphi$ if M is a λ–term of type φ. Notice that while each term has exactly one type, there are infinitely many terms having the same type. The following is a Gentzen–calculus for statements of the form $M : \varphi$. Here, Γ, Δ, Θ denote arbitrary sets of such statements, x, y individual variables (of appropriate type), and M, N terms. The rules are shown in Table 6. First of all notice that if we strip off the labelling by λ–terms we get a natural deduction calculus for intuitionistic logic (in the only connective $\to$). Hence if a sequent $\{M_i : \varphi_i : i < n\} \vdash N : \chi$ is derivable then $\overset{\mathcal{H}}{\rightsquigarrow} \{\varphi_i : i < n\} \vdash \chi$, whence $\{\varphi_i : i < n\} \vdash^{\mathsf{H}} \chi$. Conversely, given a natural deduction proof of $\{\varphi_i : i < n\} \vdash \chi$, we can decorate the proof with λ–terms by assigning the variables at the leaves of the tree for the axioms

Table 6. Rules of the Labelled Calculus

$$\text{(axiom)}\quad x:\varphi \vdash x:\varphi \qquad \text{(M)}\quad \frac{\Gamma \vdash M:\varphi}{\Gamma, x:\chi \vdash M:\varphi}$$

$$\text{(cut)}\quad \frac{\Gamma \vdash M:\varphi \quad \Delta, x:\varphi, \Theta \vdash N:\chi}{\Delta, \Gamma, \Theta \vdash [M/x]N:B}$$

$$\textbf{(E}^{\rightarrow}\textbf{)}\quad \frac{\Gamma \vdash M:(\varphi \rightarrow \chi) \quad \Delta \vdash N:\varphi}{\Gamma, \Delta \vdash (MN):\chi}$$

$$\textbf{(I}^{\rightarrow}\textbf{)}\quad \frac{\Gamma, x:\varphi \vdash M:\chi}{\Gamma \vdash (\lambda x.M):(\varphi \rightarrow \chi)}$$

and then descending it until we hit the root. Then we get a proof of the sequent $\{M_i : \varphi_i : i < n\} \vdash N : \chi$ in the above calculus.

Now we interpret the intuitionistic formulae in this proof calculus as types. For a set Γ of λ–terms over the set B of basic types we put

$$|\Gamma| := \{\varphi \in \mathrm{Typ}_{\rightarrow}(B) : \text{ there is } M \in \Gamma \text{ of type } \varphi\} \tag{3.127}$$

Definition 3.56 *For a set Γ of types and a single type φ over a set B of basic types we put $\Gamma \vdash^{\boldsymbol{\lambda}} \varphi$ if there is a term M of type φ such that every type of a variable occurring free in M is in Γ.*

Returning to our calculus above we notice that if

$$\{M_i : \varphi_i : i < n\} \vdash N : \chi \tag{3.128}$$

is derivable, we also have $\{\varphi_i : i < n\} \vdash^{\boldsymbol{\lambda}} \chi$. This is established by induction on the proof. Moreover, the converse also holds (by induction on the derivation). Hence we have the following result.

Theorem 3.57 (Curry) $\Gamma \vdash^{\boldsymbol{\lambda}} \varphi$ *iff* $\Gamma \vdash^{\mathsf{H}} \varphi$.

The correspondence between intuitionistic formulae and types has also been used to obtain a rather nice characterization of shortest proofs. Basically, it turns out that a proof of $\Gamma \vdash N : \varphi$ can be shortened if N contains a redex. Suppose, namely, that N contains the redex $((\lambda x.M)U)$. Then, as is easily seen, the proof contains a proof of $\Delta \vdash (\lambda x.M)U) : \chi$. This proof part can be shortened. To simplify the argument here we assume that no use of (cut) and (M) has been made. Observe that we can assume that this very sequent

has been introduced by the rule (**I**→) and its left premiss by the rule (**E**→) and $\Delta = \Delta' \cup \Delta''$.

$$\text{(3.129)} \qquad \frac{\dfrac{\Delta', x : \psi \vdash M : \chi}{\Delta' \vdash (\lambda x.M) : (\psi \to \chi)} \qquad \Delta'' \vdash U : \psi}{\Delta', \Delta'' \vdash ((\lambda x.M)U) : \chi}$$

Then a single application of (cut) gives this:

$$\text{(3.130)} \qquad \frac{\Delta'' \vdash U : \psi \qquad \Delta', x : \psi \vdash M : \chi}{\Delta', \Delta'' \vdash [M/x]U : \chi}$$

While the types and the antecedent have remained constant, the conclusion now has a term associated to it that is derived from contracting the redex. The same can be shown if we take intervening applications of (cut) and (M), but the proof is more involved. Essentially, we need to perform more complex proof transformations. There is another simplification that can be made, namely when the derived term is explicitly α–converted. Then we have a sequent of the form $\Gamma \vdash (\lambda x.Mx) : (\varphi \to \chi)$. Then, again putting aside intervening occurrences of (cut) and (M), the proof is as follows.

$$\text{(3.131)} \qquad \frac{\dfrac{\Gamma \vdash (\lambda x.Mx) : \varphi \to \chi \qquad y : \varphi \vdash y : \varphi}{\Gamma, y : \varphi \vdash (My) : \chi}}{\Gamma \vdash (\lambda y.My) : (\varphi \to \chi)}$$

This proof part can be eliminated completely, leaving only the proof of the left hand premiss. An immediate corollary of this fact is that if the sequent $\{x_i : \varphi_i : i < n\} \vdash N : \chi$ is provable for some N, then there is an N' obtained from N by a series of α–/β– and η–normalization steps such that the sequent $\{x_i : \varphi_i : i < n\} \vdash N' : \chi$ is also derivable. The proof of the latter formula is shorter than the first on condition that N contains a subterm that can be β– or η–reduced.

Notes on this section. λ–abstraction already appeared in [**?**] (written in 1891). Frege wrote $\dot{\varepsilon}.f(\varepsilon)$. The first to study abstraction systematically was Alonzo Church (see [**?**]). Combinatory logic on the other hand has appeared first in the work of Moses Schönfinkel [**?**] and Haskell Curry [**?**]. The typing is reminiscent of Husserl's semantic categories. More on that in Chapter 4. Suffice it to say that two elements are of the same semantic category iff they can meaningfully occur in the same terms. There are exercises below on applicative structures that demonstrate that Husserl's conception characterizes

exactly the types up to renaming of the basic types.

Exercise 101. Show that in ZFC, $M \times (N \times O) \neq (M \times N) \times O$.

Exercise 102. Find combinators G and C such that $\mathsf{G}XYZ \triangleright X(ZYZ)$ and $\mathsf{C}XYZ \triangleright XZY$.

Exercise 103. Determine all types of G and C of the previous exercise.

Exercise 104. We have seen in Section 3.2 that $(\varphi \to \varphi)$ can be derived from (a0) and (a1). Use this proof to give a definition of $\mathtt{I}$ in terms of $\mathtt{K}$ and $\mathtt{S}$.

Exercise 105. Show that any combinatorially complete applicative structure with more than one element is infinite.

Exercise 106. Show that $\bullet$, $\mathfrak{k}$ and $\mathfrak{s}$ defined on V_ω are proper classes in V_ω. *Hint.* It suffices to show that they are infinite. However, there is a proof that works for any universe V_κ, so here is a more general method. Say that $C \subseteq V_\kappa$ is *rich* if for every $x \in V_\kappa$, $x \in^+ C$. Show that no set is rich. Next show that $\bullet$, $\mathfrak{k}$ and $\mathfrak{s}$ are rich.

Exercise 107. Let $\langle \{A_\alpha : \alpha \in \mathrm{Typ}_\to(B)\}, \bullet \rangle$ be a typed applicative structure. Now define the partial algebra $\langle A, \bullet \rangle$ where $A := \bigcup_\alpha A_\alpha$. Show that if the applicative structure is combinatorially complete, the type assignment is unique up to permutation of the elements of B. Show also that if the applicative structure is not combinatorially complete, uniqueness fails. *Hint.* First, establish the elements of basic type, and then the elements of type $b \to c$, where $b, c \in C$ are basic. Now, an element of type $b \to c$ can be applied to all and only the elements of type c. This allows to define which elements have the same basic type.

Exercise 108. Let $V := \{\mathtt{p}\vec{\alpha} : \vec{\alpha} \in \{\mathtt{0}, \mathtt{1}\}^*\}$. Denote the set of all types of combinators that can be formed over the set V by C. Show that C is exactly the set of intuitionistically valid formulae, that is, the set of formulae derivable in $\vdash^{\mathsf{H}}$.

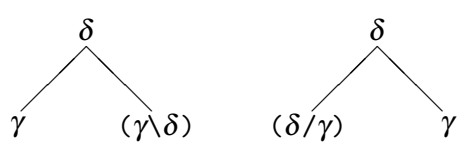
4. The Syntactic Calculus of Categories

Categorial grammars — in contrast to phrase structure grammars — specify no special set of rules, but instead associate with each lexical element a finite set of context schemata. These context schemata can either be defined over strings or over structure trees. The second approach is older and leads to the

so called Ajdukiewicz–Bar Hillel–Calculus (AB), the first to the Lambek–Calculus (L). We present first the calculus AB.

We assume that all trees are strictly binary branching with exception of the preterminal nodes. Hence, every node whose daughter is not a leaf has exactly two daughters. The phrase structure rule $X \to YZ$ licenses the expansion of the symbol X to the sequence YZ. In categorial grammar, the category Y represents the set of trees whose root has label Y, and the rule says that trees with root label Y and Z, respectively, may be composed to a tree with root X. The approach is therefore from bottom to top rather than top to bottom. The fact that a tree of the named kind may be composed is coded by the so called **category assignment**. To this end we first have to define *categories*. Categories are simply terms over a signature. If the set of proper function symbols is M and the set of 0–ary function symbols is C we write $\mathrm{Cat}_M(C)$ rather than $\mathrm{Tm}_M(C)$ for the set of terms over this signature. The members are called **categories** while members of C are called **basic categories**. In the AB–Calculus we have $M = \{\backslash, /\}$. (L also has $\bullet$.) Categories are written in infix notation. So, we write `(a/b)` in place of `/ab`. Categories will be denoted by lower case Greek letters, basic categories by lower case Latin letters. If $C = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ then `((a/b)\c)`, `(c/a)` are categories. Notice that we take the actual strings to be the categories. This convention will soon be relaxed. Then we also use left associative bracketing as with λ–terms. So, `a/b/c/b/a` will be short for `((((a/b)/c)/b)/a)`. (Notice the change in font signals that the way the functor is written down has been changed.) The interpretation of categories in terms of trees is as follows. A *tree* is understood to be an exhaustively ordered strictly binary branching tree with labels in $\mathrm{Cat}_{\backslash,/}(C)$, which results from a constituent analysis. This means that nonterminal nodes branch exactly when they are not preterminal. Otherwise they have a single daughter, whose label is an element of the alphabet. The labelling function ℓ must be correct in the sense of the following definition.

(3.132)



Call a tree **2–standard** if a node is at most binary branching, and if it is nonbranching iff it is preterminal.

Definition 3.58 *Let A be an alphabet and $\zeta : A_\varepsilon \to \wp(\mathrm{Cat}_{\backslash,/}(C))$ be a func-*

tion for which $\zeta(a)$ is always finite. Then ζ is called a ***category assignment****. Let $\mathfrak{T} = \langle T, <, \sqsubset, t\rangle$ be a 2–standard tree with labels in $\mathrm{Cat}_{\backslash,/}(C)$. $\mathfrak{T}$ is* ***correctly ζ–labelled*** *if (1) for every nonbranching x with daughter y $\ell(x) \in \zeta(\ell(y))$, and (2) for every branching x which immediately dominates y_0, y_1 and $y_0 \sqsubset y_1$ we have: $\ell(y_0) = (\ell(x)/\ell(y_1))$ or $\ell(y_1) = (\ell(y_0)\backslash\ell(x))$.*

Definition 3.59 *The quadruple $K = \langle S, C, A, \zeta\rangle$ is an* ***AB–grammar*** *if A and C are finite sets, the* ***alphabet*** *and the set of* ***basic categories****, respectively, $S \in C$, and $\zeta\colon A \to \wp(\mathrm{Cat}_{\backslash,/}(C))$ a category assignment. The set of labelled trees that is accepted by K is denoted by $L_B(K)$. It is the set of 2–standard correctly ζ–labelled trees with labelling $\ell\colon T \to \mathrm{Cat}_{\backslash,/}(C)$ such that the root carries the label S.*

We emphasize that for technical reasons also the empty string must be assigned a category. Otherwise no language which contains the empty string is a language accepted by a categorial grammar. We shall ignore this case in the sequel, but in the exercises will shed more light on it.

AB–grammars only allow to define the mapping ζ. For given ζ, the set of trees that are correctly ζ–labelled are then determined and can be enumerated. To this end we need to simply enumerate all possible constituents. Then for each preterminal x we choose an appropriate label $\gamma \in \zeta(\ell(y))$, where $y \prec x$. The labelling function therefore is fixed on all other nodes. In other words, the AB–grammars (which will turn out to be variants of CFGs) are invertible. The algorithm for finding analysis trees is not very effective. However, despite this we can show that already a CFG generates all trees, which allows us to import the results on CFGs.

Theorem 3.60 *Let $K = \langle \mathtt{S}, C, A, \zeta\rangle$ be an AB–grammar. Then there exists a CFG G such that $L_B(K) = L_B(G)$.*

Proof. Let N be the set of all subterms of terms in $\zeta(a)$, $a \in A$. N is clearly finite. It can be seen without problem that every correctly labelled tree only carries labels from N. The start symbol is that of K. The rules have the form

$$\gamma \to (\gamma/\delta) \quad \delta \tag{3.133}$$

$$\gamma \to \delta \quad (\delta\backslash\gamma) \tag{3.134}$$

$$\gamma \to a \qquad\qquad (\gamma \in \zeta(a)) \tag{3.135}$$

where γ, δ run through all symbols of N and a through all symbols from A. This defines $G := \langle \mathtt{S}, N, A, R\rangle$. If $\mathfrak{T} \in L_B(G)$ then the labelling is correct, as is

easily seen. Conversely, if $\mathfrak{T} \in L_B(K)$ then every local tree is an instance of a rule from G, the root carries the symbol $\mathtt{S}$, and all leaves carry a terminal symbol. Hence $\mathfrak{T} \in L_B(G)$. □

Conversely every CFG can be converted into an AB–grammar; however, these two grammars need not be strongly equivalent. Given L, there exists a grammar G in Greibach Normal Form such that $L(G) = L$. We distinguish two cases. Case 1. $\varepsilon \in L$. We assume that $\mathtt{S}$ is never on the right hand side of a production. (This can be installed keeping to Greibach Normal Form; see the exercises.) Then we choose a category assignment as in Case 2 and add $\zeta(\varepsilon) := \{\mathtt{S}\}$. Case 2. $\varepsilon \notin L$. Now define

$$\zeta_G(a) := \{X/Y_{n-1}/\cdots/Y_1/Y_0 : X \to a^\frown \prod_{i<n} Y_i \in R\} \tag{3.136}$$

Put $K := \langle \mathtt{S}, N_G, A, \zeta_G \rangle$. We claim that $L(K) = L(G)$. To this end we shall transform G by replacing the rules $\rho = X \to a^\frown \prod_{i<n} Y_i$ by the rules

$$Z_0^\rho \to aY_0, \quad Z_1^\rho \to Z_0Y_1, \quad \ldots, \quad Z_{n-1}^\rho \to Y_{n-2}Y_{n-1} \tag{3.137}$$

This defines the grammar H. We have $L(H) = L(G)$. Hence it suffices to show that $L(K) = L(H)$. In place of K we can also take a CFG F; the nonterminals are N_F. We show now that that F and H generate the same trees modulo the R–simulation $\sim \subseteq N_H \times N_F$, which is defined as follows. (a) For $X \in N_G$ we have $X \sim Y$ iff $X = Y$. (b) $Z_i^\rho \sim W$ iff $W = X/Y_{n-1}/\cdots/Y_{i+1}$ and $\rho = X \to Y_0{}^\frown Y_1{}^\frown \cdots {}^\frown Y_{n-1}$ for certain Y_j, $i < j < n$. To this end it suffices to show that the rules of F correspond via $\sim$ to the rules of H. This is directly calculated.

Theorem 3.61 (Bar–Hillel & Gaifman & Shamir) *Let L be a language. L is context free iff $L = L_B(K)$ for some AB–grammar.* □

Notice that we have used only $/$. It is easy to see that $\backslash$ alone would also have sufficed.

Now we look at Categorial Grammar from the standpoint of the sign grammars. We introduce a binary operation '$\cdot$' on the set of categories which satisfies the following equations.

$$(\gamma/\delta) \cdot \delta = \gamma, \qquad \delta \cdot (\delta\backslash\gamma) = \gamma \tag{3.138}$$

Hence $\delta \cdot \eta$ is defined only when $\eta = (\delta\backslash\gamma)$ or $\delta = (\gamma/\eta)$ for some γ. Now let us look at the construction of a sign algebra for CFGs of Section 3.1.

Because of the results of this section we can assume that the set T' is a subset of $\mathrm{Cat}_{\backslash,/}(C)$ which is closed under $\cdot$. Then for our proper modes we may proceed as follows. If a is of category γ then there exists a context free rule $\rho = \gamma \to a$ and we introduce a 0–ary mode $\mathtt{R}_\rho := \langle a, \gamma, a\rangle$. The other rules can be condensed into a single mode

$$\mathtt{A}(\langle \vec{x}, \gamma, \vec{x}\rangle, \langle \vec{y}, \beta, \vec{y}\rangle) := \langle \vec{x}\vec{y}, \gamma \cdot \beta, \vec{x}\vec{y}\rangle \tag{3.139}$$

(Notice that $\mathtt{A}$ is actually a structure term, so should actually write $\upsilon(\mathtt{A})$ is place of it. We will not do so, however, to avoid clumsy notation.)

However, this still does not generate the intended meanings. We still have to introduce $\mathtt{S}^\heartsuit$ as in Section 3.1. We do not want to do this, however. Instead we shall deal with the question whether one can generate the meanings in a more systematic fashion. In general this is not possible, for we have only assumed that f is computable. However, in practice it appears that the syntactic categories are in close connection to the meanings. This is the philosophy behind Montague Semantics.

Let an arbitrary set C of basic categories be given. Further, let a set B of basic types be given. From B we can form types in the sense of the typed λ–calculus and from C categories in the sense of categorial grammar. We shall require that these two are connected by a homomorphism from the algebra of categories to the algebra of types. Both are realized over strings. So, for each basic category $c \in C$ we choose a type γ_c. Then we put

$$\begin{aligned} \sigma(c) &:= \gamma_c \\ \sigma((\gamma/\delta)) &:= (\sigma(\delta) \to \sigma(\gamma)) \\ \sigma((\delta\backslash\gamma)) &:= (\sigma(\delta) \to \sigma(\gamma)) \end{aligned} \tag{3.140}$$

Let now $\mathfrak{A} = \langle \{A_\alpha \colon \alpha \in \mathrm{Typ}_{\to}(B)\}, \bullet\rangle$ be a typed applicative structure. σ defines a **realization of** B in $\mathfrak{A}$ by assigning to each category γ the set $A_{\sigma(\gamma)}$, which we also denote by $\ulcorner\gamma\urcorner$. We demonstrate this with our arithmetical terms. The applicative structure shall be based on sets, using app as the interpretation of function application. This means that $A_{(\alpha\to\beta)} = A_\alpha \to A_\beta$. Consequently, $\ulcorner(\gamma/\delta)\urcorner = \ulcorner(\delta\backslash\gamma)\urcorner = \ulcorner\delta\urcorner \to \ulcorner\gamma\urcorner$. There is the basic category $\mathtt{Z}$, and it is realized by the set of numbers from 0 to 9. Further, there is the category $\mathtt{T}$ which gets realized by the rational numbers $\mathbb{Q}$ — for example.

$$\begin{aligned} \ulcorner\mathtt{Z}\urcorner &:= \{0, 1, \ldots, 9\} \\ \ulcorner\mathtt{T}\urcorner &:= \mathbb{Q} \end{aligned} \tag{3.141}$$

$+\colon \mathbb{Q}\times\mathbb{Q}\to\mathbb{Q}$ is a binary function. We can redefine it as shown in Section 3.3 to an element of $\mathbb{Q}\to(\mathbb{Q}\to\mathbb{Q})$, which we also denote by $+$. The syntactic category which we assign to $+$ has to match this. We choose `((T\T)/T)`. Now we have

$$\ulcorner\texttt{((T\backslash T)/T)}\urcorner = \mathbb{Q}\to(\mathbb{Q}\to\mathbb{Q}) \tag{3.142}$$

as desired. Now we have to see to it that the meaning of the string `5+7` is indeed 12. To this end we require that if `+` is combined with `7` to the constituent `+7` the meaning of `+` (which is a function) is applied to the number 7. So, the meaning of `+7` is the function $x\mapsto x+7$ on $\mathbb{Q}$. If we finally group `+7` and `5` together to a constituent then we get a constituent of category `T` whose meaning is 12.

If things are arranged in this way we can uniformly define two modes for AB, $\mathtt{A}_{>}$ and $\mathtt{A}_{<}$.

$$\mathtt{A}_{>}(\langle\vec{x},\alpha,M\rangle,\langle\vec{y},\beta,N\rangle) := \langle\vec{x}\vec{y},\alpha\cdot\beta,MN\rangle \tag{3.143a}$$

$$\mathtt{A}_{<}(\langle\vec{x},\alpha,M\rangle,\langle\vec{y},\beta,N\rangle) := \langle\vec{x}\vec{y},\alpha\cdot\beta,NM\rangle \tag{3.143b}$$

We further assume that if $a\in A$ has category α then there are only finitely many $M\in\ulcorner\alpha\urcorner$ which are meanings of a of category α. For each such meaning M we assume a 0–ary mode $\langle a,\alpha,M\rangle$. Therewith AB is completely standardized. In the respective algebras $\mathfrak{Z}$, $\mathfrak{T}$ and $\mathfrak{M}$ there is only one binary operation. In $\mathfrak{Z}$ it is the concatenation of two strings, in $\mathfrak{T}$ it is cancellation, and in $\mathfrak{M}$ function application. The variability is not to be found in the proper modes, only in the 0–ary modes, that is, the lexicon. Therefore one speaks of Categorial Grammar as a 'lexical' theory; all information about the language is in the lexicon.

Definition 3.62 *A sign grammar $\langle\mathfrak{A},\varepsilon,\gamma,\mu\rangle$ is called an **AB–sign grammar** if the signature consists of the two modes $\mathtt{A}_{>}$ and $\mathtt{A}_{<}$ and finitely many 0–ary modes $\mathtt{M}_i$, $i<n$ such that*

① $\mathtt{M}_i^{\upsilon} = \langle\vec{x}_i,\gamma_i,N_i\rangle$, $i<n$,

② $\mathfrak{Z} = \langle A^*,{}^\frown,\langle\vec{x}_i : i<n\rangle\rangle$,

③ $\mathfrak{T} = \langle \mathrm{Cat}_{\backslash,/}(C),\cdot,\langle\gamma_i : i<n\rangle\rangle$ *for some set C,*

④ $\mathfrak{M} = \langle\{M_\alpha : \alpha\in\mathrm{Typ}_{\to}(B)\},\bullet,\langle N_i : i<n\rangle\rangle$ *is an expansion of a typed applicative structure by constants,*

⑤ *and* $N_i \in M_{\sigma(\gamma_i)}$, $i < n$.

Notice that the algebra of meanings is partial and has as its unique operation function application. (This is not defined if the categories do not match.) As we shall see, the concept of a categorial grammar is somewhat restrictive with respect to the language generated (it has to be context free) and with respect to the categorial symbols, but it is not restrictive with respect to meanings.

We shall give an example. We look at our alphabet of ten digits. Every nonempty string over this alphabet denotes a unique number, which we name by this very sequence. For example, the sequence `721` denotes the number 721, which in binary is `101101001` or `LOLLOLOOL`. We want to write an AB–grammar which couples a string of digits with its number. This is not as easy as it appears at first sight. In order not to let the example appear trivial we shall write a grammar for binary numbers, with `L` in place of 1 and `O` in place of 0. To start, we need a category `Z` as in the example above. This category is realized by the set of natural numbers. Every digit has the category `Z`. So, we have the following 0–ary modes.

(3.144) $\mathtt{Z_0} := \langle \mathtt{O}, Z, 0\rangle \qquad \mathtt{Z_1} := \langle \mathtt{L}, Z, 1\rangle$

Now we additionally agree that digits have the category `Z\Z`. With this the number `LOL` is analyzed in this way.

(3.145)
```
 L    O      L
 Z  (Z\Z)  (Z\Z)
 ---------
     Z
 ------------------
        Z
```

This means that digits are interpreted as functions from ω to ω. As one easily finds out these are the functions $\lambda x_0.2x_0 + k$, $k \in \{0,1\}$. Here k must be the value of the digit. So, we additionally need the following zeroary modes.

(3.146) $\mathtt{M_0} := \langle \mathtt{0}, \mathtt{(Z\backslash Z)}, \lambda x_0.2x_0\rangle$

(3.147) $\mathtt{M_1} := \langle \mathtt{1}, \mathtt{(Z\backslash Z)}, \lambda x_0.2x_0 + 1\rangle$

(Notice that we write $\lambda x_0.2x_0$ and not `(λx0.(2*x0))`, since the latter is a string, while the former is actually a function in a particular algebra.) However, the grammar does not have the ideal form. For every digit has two different meanings which do not need to have anything to do with each other.

For example, we could have introduced the following mode in place of — or even in addition to — $\mathtt{M}_1$.

$$\mathtt{M}_2 := \langle \mathtt{0}, (\mathtt{Z}\backslash\mathtt{Z}), \lambda x_0.2x_0+1\rangle \tag{3.148}$$

We can avoid this by introducing a second category symbol, T, which stands for a sequence of digits, while Z only stands for digits. In place of $\mathtt{M}_0$ we now define the empty modes $\mathtt{N}_0$, and $\mathtt{N}_1$:

$$\mathtt{N}_0 := \langle \varepsilon, (\mathtt{T}/\mathtt{Z}), \lambda x_0.x_0\rangle \tag{3.149}$$

$$\mathtt{N}_1 := \langle \varepsilon, ((\mathtt{T}/\mathtt{T})/\mathtt{Z}), \lambda x_1.\lambda x_0.2x_1+x_0\rangle \tag{3.150}$$

For example, we get LOL as the exponent of the term

$$\mathtt{A}_{>}\mathtt{A}_{>}\mathtt{N}_1\mathtt{A}_{>}\mathtt{A}_{>}\mathtt{N}_1\mathtt{A}_{>}\mathtt{N}_0\mathtt{Z}_1\mathtt{Z}_0\mathtt{Z}_1 \tag{3.151}$$

The meaning of this term is calculated as follows.

$$\begin{aligned}
&(\mathtt{A}_{>}\mathtt{A}_{>}\mathtt{N}_1\mathtt{A}_{>}\mathtt{A}_{>}\mathtt{N}_1\mathtt{A}_{>}\mathtt{N}_0\mathtt{Z}_1\mathtt{Z}_0\mathtt{Z}_1)^{\mu}\\
=&\mathtt{N}_1^{\mu}(\mathtt{N}_1^{\mu}(\mathtt{N}_0^{\mu}(\mathtt{Z}_1^{\mu}))(\mathtt{Z}_0^{\mu}))(\mathtt{Z}_1^{\mu})\\
=&\mathtt{N}_1^{\mu}(\mathtt{N}_1^{\mu}((\mathtt{N}_0^{\mu}(1))(0))(1)\\
=&\mathtt{N}_1^{\mu}(\mathtt{N}_1^{\mu}((\lambda x_0.x_0)(1))(0))(1)\\
=&\mathtt{N}_1^{\mu}(\mathtt{N}_1^{\mu}(1)(0))(1)\\
=&\mathtt{N}_1^{\mu}((\lambda x_1.\lambda x_0.(2x_1+x_0))(1)(0))(1)\\
=&\mathtt{N}_1^{\mu}(2)(1)\\
=&(\lambda x_1.\lambda x_0.(2x_1+x_0))(2)(1)\\
=&5
\end{aligned} \tag{3.152}$$

This solution is far more elegant than the first. Despite of this, it too is not satisfactory. We had to postulate additional modes which one cannot see on the string. Also, we needed to distinguish strings from digits. For comparison we show a solution that involves restricting the concatenation function. Put

$$\vec{x}\star\vec{y} := \begin{cases} \vec{x}^\frown\vec{y} & \text{if } \vec{y}\in A,\\ \text{undefined} & \text{otherwise.}\end{cases} \tag{3.153}$$

Now take a binary symbol P and set

$$\mathtt{P}(\langle\vec{x},\mathtt{Z},m\rangle,\langle\vec{y},\mathtt{Z},n\rangle) = \langle\vec{x}\star\vec{y},\mathtt{Z},2m+n\rangle \tag{3.154}$$

One could also define two unary modes for appending a digit. But this would mean making the empty string an exponent for 0, or else it requires another set of two digits to get started. A further problem is the restricted functionality in the realm of strings. With the example of the grammar T of the previous section we shall exemplify this. We have agreed that every term is enclosed by brackets, which merely are devices to help the eye. These brackets are now symbols of the alphabet, but void of real meaning. To place the brackets correctly, some effort must be made. We propose the following grammar.

$$
\begin{aligned}
&\mathtt{O}_1 := \langle \mathtt{+}, \mathtt{((T\backslash U)/T)}, \lambda x_1.\lambda x_0.x_0 + x_1 \rangle \\
&\mathtt{O}_2 := \langle \mathtt{-}, \mathtt{((T\backslash U)/T)}, \lambda x_1.\lambda x_0.x_0 - x_1 \rangle \\
&\mathtt{O}_3 := \langle \mathtt{/}, \mathtt{((T\backslash U)/T)}, \lambda x_1.\lambda x_0.x_0 / x_1 \rangle \\
&\mathtt{O}_4 := \langle \mathtt{*}, \mathtt{((T\backslash U)/T)}, \lambda x_1.\lambda x_0.x_0 x_1 \rangle \\
&\mathtt{O}_5 := \langle \mathtt{-}, \mathtt{(U/T)}, \lambda x_0. - x_0 \rangle \\
&\mathtt{O}_6 := \langle \mathtt{(}, \mathtt{(L/U)}, \lambda x_0.x_0 \rangle \\
&\mathtt{O}_7 := \langle \mathtt{)}, \mathtt{(L\backslash T)}, \lambda x_0.x_0 \rangle \\
&\mathtt{Z}_0 := \langle \mathtt{L}, \mathtt{T}, 0 \rangle \\
&\mathtt{Z}_1 := \langle \mathtt{O}, \mathtt{T}, 1 \rangle
\end{aligned}
\tag{3.155}
$$

The conception is that an operation symbol generates an unbracketed term which needs a left and a right bracket to become a 'real' term. A semantics that fits with this analysis will assign the identity to all these. We simply take $\mathbb{Q}$ for all basic categories. The brackets are interpreted by the identity function. If we add a bracket, nothing happens to the value of the term. This is a viable solution. However, it amplifies the set of basic categories without any increase in semantic types as well.

The application of a function to an argument is by far not the only possible rule of composition. In particular Peter Geach has proposed in [**?**] to admit further rules of combination. This idea has been realized on the one hand in the Lambek–Calculus, which we will study later, and also in **combinatory categorial grammars**. The idea to the latter is as follows. Each mode in Categorial Grammar is interpreted by a semantical typed combinator. For example, $\mathtt{A}_<$ acts on the semantics like the combinator U (defined in Section 3.3) and $\mathtt{A}_>$ is interpreted by the combinator $\mathtt{I}$. This choice of combinators is — seen from the standpoint of combinatory logic — only one of many possible choices. Let us look at other possibilities. We could add to the ones we have

also the functions corresponding to the following closed λ–term.

(3.156) $\mathsf{B} := (\lambda \mathtt{x}_0 . (\lambda \mathtt{x}_1 . (\lambda \mathtt{x}_2 . (\mathtt{x}_0 (\mathtt{x}_1 \mathtt{x}_2)))))$

$\mathsf{B}MN$ is nothing but function composition of the functions M and N. For evidently, if $\mathtt{x}_2$ has type γ then $\mathtt{x}_1$ must have the type $\beta \to \gamma$ for some β and $\mathtt{x}_0$ the type $\alpha \to \beta$ for some α. Then $\mathsf{B}\mathtt{x}_0\mathtt{x}_1 \triangleright (\lambda \mathtt{x}_2 . (\mathtt{x}_0 (\mathtt{x}_1 \mathtt{x}_2)))$ is of type $\alpha \to \gamma$. Notice that for each α, β and γ we have a typed λ–term $\mathsf{B}_{\alpha,\beta,\gamma}$.

(3.157) $\mathsf{B}_{\alpha,\beta,\gamma} := (\lambda \mathtt{x}^0_{\alpha\to\beta} . (\lambda \mathtt{x}^1_{\beta\to\gamma} . (\lambda \mathtt{x}^2_{\alpha} . (\mathtt{x}^0_{\alpha\to\beta} (\mathtt{x}^1_{\beta\to\gamma} \mathtt{x}^2_{\gamma})))))$

However, as we have explained earlier, we shall not use the explicitly typed terms, but rather resort to the implicitly typed terms (or combinators). We define two new category products $\oplus$ and $\ominus$ by

(3.158a) $(\gamma/\beta) \oplus (\beta/\alpha) := (\gamma/\alpha)$

(3.158b) $(\beta/\alpha) \ominus (\beta\backslash\gamma) := (\gamma/\alpha)$

(3.158c) $(\gamma/\beta) \oplus (\alpha\backslash\beta) := (\alpha\backslash\gamma)$

(3.158d) $(\alpha\backslash\beta) \ominus (\beta\backslash\gamma) := (\alpha\backslash\gamma)$

Further, we define two new modes, $\mathtt{B}_>$ and $\mathtt{B}_<$, as follows:

(3.159) $\mathtt{B}_>(\langle \vec{x}, \alpha, M\rangle, \langle \vec{y}, \beta, N\rangle) := \langle \vec{x}^\frown \vec{y}, \alpha \oplus \beta, \mathsf{B}MN\rangle$

(3.160) $\mathtt{B}_<(\langle \vec{x}, \alpha, M\rangle, \langle \vec{y}, \beta, N\rangle) := \langle \vec{x}^\frown \vec{y}, \alpha \ominus \beta, \mathsf{B}NM\rangle$

Here, it is not required that the type of M matches α in any way, or the type of N the category β. In place of $\mathsf{B}NM$ we could have used $\mathsf{V}MN$, where

(3.161) $\mathsf{V} := (\lambda \mathtt{x}_0 . (\lambda \mathtt{x}_1 . (\lambda \mathtt{x}_2 . (\mathtt{x}_1 (\mathtt{x}_0 \mathtt{x}_2)))))$

We denote by CCG(B) the extension of AB by the implicitly typed combinator B. This grammar not only has the modes $\mathtt{A}_>$ and $\mathtt{A}_<$ but also the modes $\mathtt{B}_>$ and $\mathtt{B}_<$. The resulting tree sets are however of a new kind. For now, if x is branching with daughters y_0 and y_1, x can have the category α/γ if y_0 has the category α/β and y_1 the category β/γ. In the definition of the products $\oplus$ and $\ominus$ there is a certain arbitrariness. What we must expect from the semantic typing regime is that the type of $\sigma(\alpha \oplus \beta)$ and $\sigma(\beta \ominus \alpha)$ equals $\eta \to \theta$ if $\sigma(\alpha) = \zeta \to \theta$ and $\sigma(\beta) = \eta \to \zeta$ for some η, ζ and θ. Everywhere else the syntactic product should be undefined. However, in fact the syntactic product has been symmetrified, and the directions specified. This goes as follows.

Table 7. The Products $\gtrdot$ and $\lessdot$

(α/δ)	$\gtrdot$	$((\delta/\beta)/\gamma)$	$:=$	$((\alpha/\beta)/\gamma)$
$((\delta/\beta)/\gamma)$	$\lessdot$	$(\delta\backslash\alpha)$	$:=$	$((\alpha/\beta)/\gamma)$
(α/δ)	$\gtrdot$	$((\beta\backslash\delta)/\gamma)$	$:=$	$((\beta\backslash\alpha)/\gamma)$
$((\beta\backslash\delta)/\gamma)$	$\lessdot$	$(\delta\backslash\alpha)$	$:=$	$((\beta\backslash\alpha)/\gamma)$
(α/δ)	$\gtrdot$	$(\gamma\backslash(\delta/\beta))$	$:=$	$(\gamma\backslash(\alpha/\beta))$
$(\gamma\backslash(\delta/\beta))$	$\lessdot$	$(\delta\backslash\alpha)$	$:=$	$(\gamma\backslash(\alpha/\beta))$
(α/δ)	$\gtrdot$	$(\gamma\backslash(\beta\backslash\delta))$	$:=$	$(\gamma\backslash(\beta\backslash\alpha))$
$(\gamma\backslash(\beta\backslash\delta))$	$\lessdot$	$(\delta\backslash\alpha)$	$:=$	$(\gamma\backslash(\beta\backslash\alpha))$

By applying a rule a category (here ζ) is cancelled. In the category η/θ the directionality (here: right) is viewed as a property of the argument, hence of θ. If θ is not cancelled, we must find θ being selected to the right again. If, however, it is cancelled from η/θ, then the latter must be to the left of its argument, which contains some occurrence of θ (as a result, not as an argument). This yields the rules as given. We leave it to the reader to show that the tree sets that can be generated from an initial category assignment ζ are again all context free. Hence, not much seems to have been gained. We shall next study another extension, CCG(P). Here

(3.162) $\quad \mathsf{P} := (\lambda \mathtt{x}_0.(\lambda \mathtt{x}_1.(\lambda \mathtt{x}_2.(\lambda \mathtt{x}_3.(\mathtt{x}_0(\mathtt{x}_1\mathtt{x}_2)\mathtt{x}_3)))))$

In order for this to be properly typed we may freely choose the type of $\mathtt{x}_2$ and $\mathtt{x}_3$, say β and γ. Then $\mathtt{x}_1$ is of type $\gamma \to (\beta \to \delta)$ for some δ and $\mathtt{x}_0$ of type $\delta \to \alpha$ for some α. $\mathtt{x}_1$ stands for an at least binary function, $\mathtt{x}_0$ for a function that needs at least one argument. If the combinator is defined, the mode is fixed if we additionally fix the syntactic combinatorics. To this end we define the products $\gtrdot$, $\lessdot$ as in Table 7. Now we define the following new modes:

(3.163) $\quad \mathtt{P}_{>}(\langle \vec{x}, \alpha, M\rangle, \langle \vec{y}, \beta, N\rangle) := \langle \vec{x}^\frown\vec{y}, \alpha \gtrdot \beta, \mathsf{P}MN\rangle$

(3.164) $\quad \mathtt{P}_{<}(\langle \vec{x}, \alpha, M\rangle, \langle \vec{y}, \beta, N\rangle) := \langle \vec{x}^\frown\vec{y}, \alpha \lessdot \beta, \mathsf{P}NM\rangle$

We shall study this type of grammar somewhat closer. We take the following

```
    A          B          A          C     a  a  b  a
((c/a)/c)  ((c/b)/c)  ((c/a)/c)  (c/a)   a  a  b  a
    ⋮          ⋮        ((c/a)/c)     c       ⋮  ⋮  ⋮
    ⋮          ⋮          (c/a)           a  ⋮  ⋮
    ⋮        ((c/b)/c)                c       ⋮  ⋮
    ⋮                    (c/b)                b  ⋮
((c/a)/c)                          c           ⋮
                (c/a)                          a
                          c
```

```
    A          B          A          C     a  a  b  a
((c/a)/c)  ((c/b)/c)  ((c/a)/c)  (c/a)   a  a  b  a
    ⋮        (((c/b)/a)/c)       (c/a)   a  ⋮  ⋮  ⋮
          ((((c/a)/b)/a)/c)            c       ⋮  ⋮  ⋮
                (((c/a)/b)/a)                a  ⋮  ⋮
                          ((c/a)/b)             b  ⋮
                                    (c/a)           a
                                          c
```

Figure 10. Two Analyses of `ABACaaba`

modes.

$$
\begin{aligned}
&\mathtt{M}_0 := \langle \mathtt{A}, \mathtt{((c/a)/c)}, \lambda x_0.\lambda x_1.x_0 + x_1 \rangle \\
&\mathtt{M}_1 := \langle \mathtt{B}, \mathtt{((c/b)/c)}, \lambda x_0.\lambda x_1.x_0 x_1 \rangle \\
&\mathtt{M}_2 := \langle \mathtt{a}, \mathtt{a}, 1 \rangle \\
&\mathtt{M}_3 := \langle \mathtt{b}, \mathtt{b}, 2 \rangle \\
&\mathtt{M}_4 := \langle \mathtt{C}, \mathtt{(c/a)}, \lambda x_0.x_0 \rangle
\end{aligned}
\tag{3.165}
$$

Take the string `ABACaaba`. It has two analyses, shown in Figure 10. In both analyses the meaning is 5. In the first analysis only the mode $\mathtt{A}_>$ has been used. The second analysis uses the mode $\mathtt{P}_>$. Notice that in the course of the derivation the categories get larger and larger (and therefore also the types).

Theorem 3.63 *There exist* CCG(P)*–grammars which generate non context*

free tree sets.

We shall show that the grammar just defined is of this kind. To this end we shall make a few more considerations.

Lemma 3.64 *Let* $\alpha = \eta_1/\eta_2/\eta_3$, $\beta = \eta_3/\eta_4/\eta_5$ *and* $\gamma = \eta_5/\eta_6/\eta_7$. *Then*

$$\alpha \gtrdot (\beta \gtrdot \gamma) = (\alpha \gtrdot \beta) \gtrdot \gamma \tag{3.166}$$

Proof. Proof by direct computation. For example, $\alpha \gtrdot \beta = \eta_1/\eta_2/\eta_3/\eta_4/\eta_5$. □

In particular, $\gtrdot$ is associative if defined (in contrast to '$\cdot$'). Now, let us look at a string of the form $\vec{x}\mathtt{Ca}\vec{y}$, where $\vec{x} \in (\mathtt{A} \cup \mathtt{B})^*$, $\vec{y} \in (\mathtt{a} \cup \mathtt{b})^*$ and $h(\vec{x}) = \vec{y}^T$, where $h\colon \mathtt{A} \mapsto \mathtt{a}, \mathtt{B} \mapsto \mathtt{b}$. An example is the string AABACabaaa. Then with the exception of $\vec{x}\mathtt{C}$ all prefixes are constituents. For prefixes of $\vec{x}$ are constituents, as one can easily see. It follows easily that the tree sets are not context free. For if $\vec{x} \neq \vec{y}$ then $\vec{x}\mathtt{Ca}h(\vec{y}^T)$ is not derivable. However, $\vec{x}\mathtt{Ca}h(\vec{x}^T)$ is derivable. If the tree set was context free, there cannot be infinitely many such $\vec{x}$, a contradiction.

So, we have already surpassed the border of context freeness. However, we can push this up still further. Let $\mathfrak{N}$ be the following grammar.

$$\begin{aligned}
\mathtt{N}_0 &:= \langle \mathtt{A}, (\mathtt{c}\backslash(\mathtt{c}/\mathtt{a})), \lambda x_0.\lambda x_1.x_0 + x_1 \rangle \\
\mathtt{N}_1 &:= \langle \mathtt{B}, (\mathtt{c}\backslash(\mathtt{c}/\mathtt{b})), \lambda x_0.\lambda x_1.x_0 \cdot x_1 \rangle \\
\mathtt{N}_2 &:= \langle \mathtt{a}, \mathtt{a}, 1 \rangle \\
\mathtt{N}_3 &:= \langle \mathtt{b}, \mathtt{b}, 2 \rangle \\
\mathtt{N}_4 &:= \langle \mathtt{C}, \mathtt{c}, \lambda x_0.x_0 \rangle
\end{aligned} \tag{3.167}$$

Theorem 3.65 $\mathfrak{N}$ *generates a non context free language.*

Proof. Let L be the language generated by $\mathfrak{N}$. Put $M := \mathtt{C}(\mathtt{A} \cup \mathtt{B})^*(\mathtt{a} \cup \mathtt{b})^*$. If L is context free, so is $L \cap M$ (by Theorem 2.14). Define h by $h(\mathtt{A}) := h(\mathtt{a}) := \mathtt{a}$, $h(\mathtt{B}) := h(\mathtt{b}) := \mathtt{b}$ as well as $h(\mathtt{C}) := \varepsilon$. We show:

$$\vec{x} \in L \cap M \text{ iff (a) } \vec{x} \in L \text{ and (b) } h(\vec{x}) = \vec{y}\vec{y} \text{ for some } \vec{y} \in (\mathtt{a} \cup \mathtt{b})^* \tag{3.168}$$

Hence $h[L \cap M] = \{\vec{y}\vec{y} : \vec{y} \in (\mathtt{a} \cup \mathtt{b})^*\}$. The latter is not context free. From this follows by Theorem 1.67 that $L \cap M$ is not context free, hence L is not context free either. Now for the proof of (3.168). If $\Delta = \langle \delta_i : i < n \rangle$ then let $\mathtt{c}/\Delta$ denote the category $\mathtt{c}/\delta_0/\delta_1/\cdots/\delta_{n-1}$. Then we have:

$$\mathtt{c}\backslash(\mathtt{c}/\Delta_1) \gtrdot \mathtt{c}\backslash(\mathtt{c}/\Delta_2) = \mathtt{c}\backslash(\mathtt{c}/\Delta_2;\Delta_1) \tag{3.169}$$

Now let $\mathtt{C}\vec{x}\vec{y}$ be such that $\vec{x} \in (\mathtt{A} \cup \mathtt{B})^*$ and $\vec{y} \in (\mathtt{a} \cup \mathtt{b})^*$. It is not hard to see that then $\mathtt{C}\vec{x}$ is a constituent. (Basically, one can either multiply or apply. The complex categories cannot be applied to the right, they can only be applied to the left, so this can happen only with `C`. If one applies `(c\(c/a))` to `c` one gets `(c/a)`, which cannot be multiplied by $\gtrdot$ with any other constituent formed. It cannot be applied either (assuming that the string is not `CAa`, in which case `CA` does become a constituent under this analysis), because nothing on the right of it has category `a`. Now let $\vec{x} := x_0 x_1 \cdots x_{n-1}$. Further, let $d_i := \mathtt{a}$ if $x_i = \mathtt{A}$ and $d_i := \mathtt{b}$ if $x_i = \mathtt{B}$, $i < n$. Then the category of $\vec{x}$ equals $\mathtt{c}\backslash(\mathtt{c}/\Delta)$ with $\Delta = \langle d_{n-i-1} : i < n \rangle$. Hence $\mathtt{C}\vec{x}$ is a constituent of category $\mathtt{c}/\Delta$. This means, however, that y_0 has the category d_0 (because d_0 is the last in the list hence the first to be discharged), y_1 the category d_1 and so on. But if y_i has the category d_i then $h(x_i) = y_i$, as is easily checked. This yields that $h(\vec{x}) = \vec{y}$. If on the other hand this is the case, the string is derivable. □

Hence we now have a grammar which generates a non context free language. CCGs are therefore stronger than AB–grammars.

There is a still different way to introduce CCGs. There we do not enlarge the set of combinatorial rules but instead introduce empty modes.

$$
\begin{aligned}
\mathtt{B}_0 &:= \langle \varepsilon, \gamma/\alpha/(\gamma/\beta)/(\beta/\alpha), \mathsf{B} \rangle \\
\mathtt{B}_1 &:= \langle \varepsilon, (\alpha\backslash\gamma)/(\gamma/\beta)/(\alpha\backslash\beta), \mathsf{B} \rangle \\
\mathtt{B}_2 &:= \langle \varepsilon, \gamma/\alpha/(\beta\backslash\gamma)/(\beta/\alpha), \mathsf{V} \rangle \\
\mathtt{B}_3 &:= \langle \varepsilon, (\alpha\backslash\gamma)/(\beta\backslash\gamma)/(\alpha\backslash\gamma), \mathsf{V} \rangle
\end{aligned}
\tag{3.170}
$$

Here we do not have four but infinitely many modes, one for each choice of α, β and γ. Only in this way it is possible to generate non context free languages. Lexical elements that have a parametric (= implicitly typed) set of categories (together with parametric meanings) are called **polymorphic**. Particularly interesting cases of polymorphic elements are the logical connectors, `and` and `not`. Syntactically, they have the category $(\alpha\backslash\alpha)/\alpha$ and α/α, respectively, where α can assume any (non parametric) category. This means that two constituents of identical category can be conjoined by `and` to another constituent of the same category, and every constituent can be turned by `not` to a constituent of identical category.

Notes on this section. Although we have said that the meanings shall be functions in an applicative structure, we sometimes put strings in their place. These strings only denote these functions. This is not an entirely harmless affair. For example, the string $(\lambda \mathtt{x}_0 . \mathtt{x}_0 + 1)$ and the string $(\lambda \mathtt{x}_1 . \mathtt{x}_1 + 1)$ denote

the same function. In fact, for reduced terms terms uniqueness holds only up to renaming of bound variables. It is standard practice in λ–calculus to consider λ–terms 'up to renaming of bound variables' (see [**?**] for a discussion). A possible remedy might be to use combinators. But here the same problem arises. Different strings may denote the same function. This is why normalisation becomes important. On the other hand, strings as meanings have the advantage to be finite, and thus may function as objects that can be stored (like codes of a Turing machine, see the discussion of Section 4.1).

Exercise 109. Let $\zeta\colon A_\varepsilon \to \wp(\mathrm{Cat}_{\backslash,/}(C))$ be a category assignment. Show that the correctly labelled trees form a context free tree set.

Exercise 110. Show that for every CFG there exists a weakly equivalent grammar in Greibach Normal Form, where the start symbol $\mathtt{S}$ does not occur on the right hand side of a production.

Exercise 111. Let $\zeta\colon A_\varepsilon \to \wp(\mathrm{Cat}_{\backslash,/}(C))$ be a category assignment. Further, let $\mathtt{S}$ be the distinguished category. ζ' is called **normal** if $\zeta(\varepsilon) = \mathtt{S}$ and no $\zeta(a)$ contains an α of the form $\gamma/\beta_0/\cdots/\beta_{n-1}$ with $\beta_i = \mathtt{S}$ for some $i < n$. Show that for any ζ there is a normal ζ' such that ζ' and ζ have the same language.

Exercise 112. Let $L \subseteq A^*$ be context free and $f\colon A^* \to M$ a computable function. Write an AB–sign grammar whose interpreted language is $\{\langle \vec{x}, f(\vec{x})\rangle : \vec{x} \in L\}$.

Exercise 113. Let $\langle \mathfrak{A}, \varepsilon, \gamma, \mu\rangle$ be an AB–sign grammar. Show for all signs $\langle \vec{x}, \alpha, M\rangle$ generated by that grammar: M has the type $\sigma(\alpha)$. *Hint.* Induction on the length of the structure term.

Exercise 114. Show that the CCG(B) grammars only generate context free string languages, even context free tree sets. *Hint.* Show the following: if A is an arbitrary finite set of categories, then with B one can generate at most $|A|^n$ many categories.

Exercise 115. Suppose we defined a product $\circ$ on categories as follows. $\alpha \circ \beta$ is defined whenever (a) $\alpha \oplus \beta$ is defined (and has the same value), or (b) $\alpha \ominus \beta$ is defined (and has the same value). Show that this does not allow to unambiguously define the semantics. (Additional question: why does this problem not arise with $\cdot$?) *Hint.* Take $\alpha = \beta = (\gamma/\gamma)$.

5. The AB–Calculus

We shall now present a calculus to derive all the valid derivability statements for AB–grammars. Notice that the only variable element is the elementary category assignment. We choose an alphabet A and an elementary category assignment ζ. We write $[\alpha]_\zeta$ for the set of all unlabelled binary constituent structures over A that have root category α under some correct ζ–labelling. As ζ is completely arbitrary, we shall deal here only with the constituent structures obtained by taking away the terminal nodes. This eliminates ζ and A, and leaves a class of purely categorial structures, denoted by $[\alpha]$. Since AB–grammars are invertible, for any given constituent structure there exists at most one labelling function (with the exception of the terminal labels). Now we introduce a binary symbol $\circ$, which takes as arguments correctly ζ–labelled constituent structures. Let $\langle X, \mathfrak{X}, \ell\rangle$ and $\langle Y, \mathfrak{Y}, m\rangle$ such constituent structures and $X \cap Y = \varnothing$. Then let

(3.171) $$\langle X, \mathfrak{X}, \ell\rangle \circ \langle Y, \mathfrak{Y}, m\rangle := \langle X \cup Y, \mathfrak{X} \cup \mathfrak{Y} \cup \{X \cup Y\}, n\rangle$$

(3.172) $$n(z) := \begin{cases} \ell(z) & \text{if } z \in \mathfrak{X}, \\ m(z) & \text{if } z \in \mathfrak{Y}, \\ \ell(X) \cdot m(Y) & \text{if } z = X \cup Y. \end{cases}$$

(In principle, $\circ$ is well defined also if the constituent structures are not binary branching.) In case where $X \cap Y \neq \varnothing$ one has to proceed to the disjoint sum. We shall not spell out the details. With the help of $\circ$ we shall form terms over A, that is, we form the algebra freely generated by A by means of $\circ$. To every term we inductively associate a constituent structure in the following way.

(3.173a) $$\alpha^k := \langle \{0\}, \{\{0\}\}, \langle \{0\}, \alpha\rangle\rangle$$

(3.173b) $$(s \circ t)^k := s^k \circ t^k$$

Notice that $\circ$ has been used with two meanings. Finally, we take a look at $[\alpha]$. It denotes classes of binary branching constituent structures over A. The following holds.

(3.174a) $$[\alpha/\beta] \circ [\beta] \subseteq [\alpha]$$

(3.174b) $$[\beta] \circ [\beta \backslash \alpha] \subseteq [\alpha]$$

We abstract now from A and ζ. In place of interpreting $\circ$ as a constructor for constituent structures over A we now interpret it as a constructor to form

constituent structures over $\mathrm{Cat}_{\backslash,/}(C)$ for some given C. We call a term from categories with the help of $\circ$ a **category complex**. Categories are denoted by lower case Greek letters, category complexes by upper case Greek letters. Inductively, we extend the interpretation $[-]$ to structures as follows.

(3.175) $\quad [\Gamma \circ \Delta] := [\Gamma] \circ [\Delta]$

Next we introduce yet another symbol, $\vdash$. This is a relation between structures and categories. If Γ is a structure and α a category then $\Gamma \vdash \alpha$ denotes the fact that for every interpretation in some alphabet A with category assignment ζ $[\Gamma] \subseteq [\alpha]$. We call the object $\Gamma \vdash \alpha$ a **sequent**. The interpretation that we get in this way we call the **cancellation interpretation**. Here, categories are inserted as concrete labels which are assigned to nodes and which are subject to the cancellation interpretation.

We shall now introduce two different calculi, one of which will be shown to be adequate for the cancellation interpretation. In formulating the rules we use the following convention. $\Gamma[\alpha]$ above the line means in this connection that Γ is a category complex in which we have fixed a single occurrence of α. When we write, for example, $\Gamma[\Delta]$ below the line, then this denotes the result of replacing that occurrence of α by Δ.

(3.176)
$$\begin{array}{llll}
\text{(ax)} & \alpha \vdash \alpha & \text{(cut)} & \dfrac{\Gamma \vdash \alpha \qquad \Delta[\alpha] \vdash \beta}{\Delta[\Gamma] \vdash \beta} \\[2ex]
(\mathbf{I}\text{–}/) & \dfrac{\Gamma \circ \alpha \vdash \beta}{\Gamma \vdash \beta/\alpha} & (/\text{–}\mathbf{I}) & \dfrac{\Gamma \vdash \alpha \qquad \Delta[\beta] \vdash \gamma}{\Delta[\beta/\alpha \circ \Gamma] \vdash \gamma} \\[2ex]
(\mathbf{I}\text{–}\backslash) & \dfrac{\alpha \circ \Gamma \vdash \beta}{\Gamma \vdash \alpha\backslash\beta} & (\backslash\text{–}\mathbf{I}) & \dfrac{\Gamma \vdash \alpha \qquad \Delta[\beta] \vdash \gamma}{\Delta[\Gamma \circ \alpha\backslash\beta] \vdash \gamma}
\end{array}$$

We denote the above calculus by $\mathsf{AB} + (\text{cut})$, and by AB the calculus without (cut). Further, the calculus consisting of (ax) and the rules ($\backslash$–**I**) and ($/$–**I**) is called AB^{-}.

Definition 3.66 *Let M be a set of category constructors. A **categorial sequent grammar** is a quintuple*

(3.177) $\quad G = \langle \mathtt{S}, C, \zeta, A, \mathcal{S} \rangle$

*where C is a finite set, the set of **basic categories**, $\mathtt{S} \in C$ the so called **distinguished category**, A a finite set, the alphabet, $\zeta : A \to \wp(\mathrm{Cat}_M(C))$ a category assignment, and $\mathcal{S}$ a sequent calculus. We write $\vdash_G \vec{x}$ if for some category complex Γ whose associated string via ζ is $\vec{x}$ we have $\overset{\mathcal{S}}{\rightsquigarrow} \Gamma \vdash \mathtt{S}$.*

We stress here that the sequent calculi are calculi to derive sequents. A sequent corresponds to a grammatical rule, or, more precisely, the sequent $\Gamma \vdash \alpha$ expresses the fact that a category complex of type Γ is a constituent that has the category α by the rules of the grammar. The rules of the sequent calculus can then be seen as *metarules*, which allow to pass from one valid statement concerning the grammar to another.

Proposition 3.67 (Correctness) *If* $\Gamma \vdash \alpha$ *is derivable in* AB^- *then* $[\Gamma] \subseteq [\alpha]$.

AB is strictly stronger than AB^-. Notice namely that the following sequent is derivable in AB:

(3.178) $\alpha \vdash (\beta/\alpha)\backslash\beta$

In natural deduction style calculi this corresponds to the following unary rule:

(3.179) $$\frac{\alpha}{(\beta/\alpha)\backslash\beta}$$

This rule is known as **type raising**, since it allows to proceed from the category α to the "raised" category $(\beta/\alpha)\backslash\beta$. Perhaps one should better call it *category raising*, but the other name is standard. To see that it is not derivable in AB^- we simply note that it is not correct for the cancellation interpretation. We shall return to the question of interpretation of the calculus AB in the next section.

An important property of these calculi is their decidability. Given Γ and α we can decide in finite time whether or not $\Gamma \vdash \alpha$ is derivable.

Theorem 3.68 (Cut Elimination) *There exists an algorithm to construct a proof of a sequent* $\Gamma \vdash \alpha$ *in* AB *from a proof of* $\Gamma \vdash \alpha$ *in* $\mathsf{AB} + (\text{cut})$. *Hence (cut) is admissible for* AB.

Proof. We presented a rather careful proof of Theorem 3.34, so that here we just give a sketch to be filled in accordingly. We leave it to the reader to verify that each of the operations reduces the cut–weight. We turn immediately to the case where the cut is on a main formula of a premiss. The first case is that the formula is introduced by (**I**–/).

(3.180) $$\frac{\dfrac{\Gamma \circ \alpha \vdash \beta}{\Gamma \vdash \beta/\alpha} \qquad \Delta[\beta/\alpha] \vdash \gamma}{\Delta[\Gamma] \vdash \gamma}$$

Now look at the rule instance that is immediately above $\Delta[\beta/\alpha] \vdash \gamma$. There are several cases. Case (0). The premiss is an axiom. Then $\gamma = \beta/\alpha$, and the cut is superfluous. Case (1). β/α is a main formula of the right hand premiss. Then $\Delta[\beta/\alpha] = \Theta[\beta/\alpha \circ \Xi]$ for some Θ and Ξ, and the instance of the rule was as follows.

$$\text{(3.181)} \qquad \frac{\Xi \vdash \alpha \qquad \Theta[\beta] \vdash \gamma}{\Theta[\beta/\alpha \circ \Xi] \vdash \gamma}$$

Now we can restructure (3.181) as follows.

$$\text{(3.182)} \qquad \frac{\dfrac{\Gamma \circ \alpha \vdash \beta \qquad \Theta[\beta] \vdash \gamma}{\Theta[\Gamma \circ \alpha] \vdash \gamma} \qquad \Xi \vdash \alpha}{\Theta[\Gamma \circ \Xi] \vdash \gamma}$$

Now we assume that the formula is not a main formula of the right hand premiss. Case (2). $\gamma = \zeta/\varepsilon$ and the premiss is obtained by application of (**I**–/).

$$\text{(3.183)} \qquad \frac{\Delta[\beta/\alpha] \circ \varepsilon \quad \vdash \zeta}{\Delta[\beta/\alpha] \quad \vdash \zeta/\varepsilon}$$

We replace (3.183) by

$$\text{(3.184)} \qquad \frac{\dfrac{\dfrac{\Gamma \circ \alpha \vdash \beta}{\Gamma \vdash \beta/\alpha} \qquad \Delta[\beta/\alpha] \circ \varepsilon \vdash \zeta}{\Delta[\Gamma] \circ \varepsilon \vdash \zeta}}{\Delta[\Gamma] \vdash \varepsilon/\zeta}$$

Case (3). $\gamma = \varepsilon\backslash\zeta$ and has been obtained by applying the rule (**I**–\). Then proceed as in Case (2). Case (4). The application of the rule introduces a formula which occurs in Δ. This case is left to the reader.

Now if the left hand premiss has been obtained by (\–**I**), then one proceeds quite analogously. So, we assume that the left hand premiss is created by an application of (\–**I**).

$$\text{(3.185)} \qquad \frac{\dfrac{\Gamma \vdash \alpha \quad \Delta[\beta] \vdash \gamma}{\Delta[\beta/\alpha \circ \Gamma] \vdash \gamma} \quad \Theta[\gamma] \vdash \delta}{\Theta[\Delta[\beta/\alpha \circ \Gamma]] \vdash \delta}$$

We can restructure (3.185) as follows.

$$\text{(3.186)}\qquad \dfrac{\Gamma \vdash \alpha \quad \dfrac{\Delta[\beta] \vdash \gamma \quad \Theta[\gamma] \vdash \delta}{\Theta[\Delta[\beta]] \vdash \delta}}{\Theta[\Delta[\beta/\alpha \circ \Gamma]] \vdash \delta}$$

Also here one calculates that the degree of the new cut is less than the degree of the old cut. The case where the left hand premiss is created by (\–**I**) is analogous. All cases have now been looked at. □

Corollary 3.69 AB + (cut) *is decidable.* □

AB gives a method to test category complexes for their syntactic category. We expect that the meanings of the terms are likewise systematically connected with a term and that we can determine the meaning of a certain string once we have found a derivation for it. We now look at the rules of AB to see how they can be used as rules for deducing sign–sequents. Before we start we shall distinguish two interpretations of the calculus. The first is the intrinsic interpretation: every sequent we derive should be correct, with all basic parts of it belonging to the original lexicon. The second is the global interpretation: the sequents we derive should be correct if the lexicon was suitably expanded. This only makes a difference with respect to signs with empty exponent. If a lexicon has no such signs the intrinsic interpretation bans their use altogether, but the global interpretation leaves room for their addition. Adding them, however, will make more sequents derivable that are based on the original lexicon only.

We also write $\vec{x} : \alpha : M$ for the sign $\langle \vec{x}, \alpha, M \rangle$. If $\vec{x}$, α or M is irrelevant in the context it is omitted. For the meanings we use λ–terms, which are however only proxy for the 'real' meanings (see the discussion at the end of the preceding section). Therefore we now write $(\lambda x.xy)$ in place of $(\lambda \mathtt{x}_0 . (\mathtt{x}_0 \mathtt{x}_1))$. A **sign complex** is a term made of signs with the help of $\circ$. Sequents are pairs $\Gamma \vdash \tau$ where Γ is a sign complex and τ a sign. σ maps categories to types, as in Section 3.4. If $\vec{x} : \alpha : M$ is derivable, we want that M is of type $\sigma(\alpha)$. Hence, the rules of AB should preserve this property. We define first a relation $\Vdash$ between sign complexes. It proceeds by means of the following rules.

$$\text{(3.187)}\qquad \Gamma[\vec{x} : \alpha/\beta : M \circ \vec{y} : \beta : N] \succ \Gamma[\vec{x}^\frown \vec{y} : \alpha : (MN)]$$

$$\text{(3.188)}\qquad \Gamma[\vec{x} : \beta : M \circ \vec{y} : \beta \backslash \alpha : N] \succ \Gamma[\vec{x}^\frown \vec{y} : \alpha : (NM)]$$

Since M and N are actually functions and not λ–terms, one may exchange any two λ–terms that denote the same function. However, if one considers them being actual λ–terms, then the following rule has to be added:

(3.189) $\Gamma[\vec{x} : \alpha : M] \succ \Gamma[\vec{x} : \alpha : N]$ if $M \equiv N$

For Γ a sign complex and σ a sign put $\Gamma \Vdash \sigma$ iff $\Gamma \succ^* \sigma$. We want to design a calculus that generates all and only the sequents $\Gamma \vdash \sigma$ such that $\Gamma \Vdash \sigma$.

To begin, we shall omit the strings and deal with the meanings. Later we shall turn to the strings, which pose independent problems. The axioms are as follows.

(3.190) (ax) $\alpha : M \vdash \alpha : M$

where M is a term of type $\sigma(\alpha)$. (cut) looks like this.

(3.191) (cut) $$\frac{\Gamma \vdash \alpha : N \qquad \Delta[\alpha : x_\eta] \vdash \beta : M}{\Delta[\Gamma] \vdash \beta : [N/x_\eta]M}$$

So, if $\Delta[\alpha : x_\eta]$ is a sign complex containing an occurrence of $\alpha : x_\eta$, then the occurrence of this sign complex is replaced and with it the variable x_η in M. So, semantically speaking cut is substitution. Notice that since we cannot tell which occurrence in M is to be replaced, we have to replace all of them. We will see that there are reasons to require that every variable has exactly one occurrence, so that this problem will never arise. (We could make this a condition on (cut). But see below for the fate of (cut).) The other rules are more complex.

(3.192) (/–**I**) $$\frac{\Gamma \vdash \alpha : M \qquad \Delta[\beta : x_\zeta] \vdash \gamma : N}{\Delta[\beta/\alpha : x_{\eta\to\zeta} \circ \Gamma] \vdash \gamma : [(x_{\eta\to\zeta}M)/x_\zeta]N}$$

This corresponds to the replacement of a primitive constituent by a complex constituent or the replacement of a value $M(x)$ by the pair $\langle M, x\rangle$. Here, the variable $x_{\eta\to\zeta}$ is introduced, which stands for a function from objects of type η to objects of type ζ. The variable x_ζ has, however, disappeared. This is a serious deficit of the calculus (which has other advantages, however). We shall below develop a different calculus. Analogously for the rule (\–**I**).

(3.193) (**E**–/) $$\frac{\Gamma \vdash \alpha : M \quad \Delta \vdash \beta/\alpha : N}{\Delta \circ \Gamma \vdash \beta : (NM)}$$

(**E**–\) $$\frac{\Gamma \vdash \alpha : M \quad \Delta \vdash \alpha\backslash\beta : N}{\Gamma \circ \Delta \vdash \beta : (NM)}$$

Lemma 3.70 *The rule* (**E**–/) *is derivable from (cut) and* (/–**I**). *Likewise, the rule* (**E**–/) *is derivable from (cut) and* (/–**I**).

Proof. The following is an instance of (/–**I**).

$$\frac{\beta : x_\zeta \vdash \beta : x_\zeta \qquad \alpha : x_\eta \vdash \alpha : x_\eta}{\alpha/\beta : x_{\zeta\to\eta} \circ \beta : x_\zeta \vdash \alpha : x_\eta} \tag{3.194}$$

Now two cuts, with $\alpha : M \vdash \alpha : M$ and with $\beta : N \vdash \beta : N$, give (**E**–/). □

Thus, the rules (3.187) and (3.188) are accounted for.

The rules (**I**–/) and (**I**–\) can be interpreted as follows. Assume that Γ is a constituent of category α/β.

$$\text{(3.195)} \qquad (\mathbf{I}\text{–}/) \qquad \frac{\Gamma \circ \alpha : x_\eta \vdash \beta : M}{\Gamma \vdash \beta/\alpha : (\lambda x_\eta . M)}$$

Here, the meaning of Γ is of the form M and the meaning of α is N. Notice that AB forces us in this way to view the meaning of a word of category α/β to be a function from η–objects to ζ–objects. For it is formally required that Γ has to have the meaning of a function. We call the rules (**I**–/) and (**I**–\) also **abstraction rules**. These rules have to be restricted, however. Define for a variable x and a term M, $\mathrm{focc}(x,M)$ to be the number of free occurrences of x in M. In the applications of the introduction rules, we add a side condition:

$$\text{(3.196)} \qquad \text{In } (\mathbf{I}\text{–}/) \text{ and } (\mathbf{I}\text{–}\backslash) : \mathrm{focc}(x_\eta, M) \leq 1$$

(In fact, one can show that from this condition already follows $\mathrm{focc}(x_\eta, M) = 1$, by induction on the proof.) To see the need for this restriction, look at the following derivation.

$$\cfrac{\cfrac{(\beta/\alpha/\alpha : M \circ \alpha : x_\eta) \circ \alpha : x_\eta \vdash \beta : (Mx_\eta)x_\eta}{\beta/\alpha/\alpha : M \circ \alpha : x_\eta \vdash \beta/\alpha : \lambda x_\eta.(Mx_\eta)x_\eta}}{\beta/\alpha/\alpha : M \vdash \beta/\alpha/\alpha : \lambda x_\eta.\lambda x_\eta.((Mx_\eta)x_\eta)}$$

The first is obtained using two applications of the derivable (**E**–/).

This rule must be further restricted, however, as the next example shows. In the rule (\–**I**) put $\Delta := \gamma := \beta$.

$$\frac{\alpha : x_\eta \vdash \alpha : x_\eta \qquad \beta : x_\zeta \vdash \beta : x_\zeta}{\alpha/\beta : x_{\zeta\to\eta} \circ \beta : x_\zeta \vdash \alpha : (x_{\zeta\to\eta} x_\zeta)} \tag{3.197}$$

Using (**I**–/), we get

$$\text{(3.198)} \qquad \frac{\alpha/\beta : x_{\zeta\to\eta} \circ \beta : x_\zeta \vdash \alpha : (x_{\zeta\to\eta} x_\zeta)}{\alpha/\beta : x_{\zeta\to\eta} \vdash \alpha/\beta : (\lambda x_\zeta.(x_{\zeta\to\eta} x_\zeta))}$$

Now $\lambda x_\zeta.x_{\zeta\to\eta} x_\zeta$ is the same function as $x_{\zeta\to\eta}$. On the other hand, by applying (**I**–\) we get

$$\text{(3.199)} \qquad \frac{\alpha/\beta : x_{\zeta\to\eta} \circ \beta : x_\zeta \vdash \alpha : (x_{\zeta\to\eta} x_\zeta)}{\beta : x_\zeta \vdash (\alpha/\beta)\backslash\alpha : (\lambda x_{\zeta\to\eta}.(x_{\zeta\to\eta} x_\zeta))}$$

This is the type raising rule which we have discussed above. A variable x_ζ can also be regarded as a function, which for given function f taking arguments of type ζ returns the value $f(x_\zeta)$. However, x_ζ is not the same function as $(\lambda x_{\zeta\to\eta}.(x_{\zeta\to\eta} x_\zeta))$. (The latter has the type $(\beta \to \alpha) \to \alpha$.) Therefore the application of the rule is incorrect in this case. Moreover, in the typed λ–calculus the equation $x_\zeta = (\lambda x_{\zeta\to\eta}.(x_{\zeta\to\eta} x_\zeta))$ is invalid.

To remedy the situation we must require that the variable which we have abstracted over appears on the left hand side of $\vdash$ in the premiss as an argument variable and not as a variable of a function that is being applied to something. So, the final form of the right slash–introduction rule is as follows.

$$\text{(3.200)} \qquad (\mathbf{I}\text{–}/) \quad \frac{\Gamma \circ \alpha : x_\eta \vdash \beta : M}{\Gamma \vdash \beta/\alpha : (\lambda x_\eta.M)} \quad \begin{array}{l} x_\eta \text{ an argument variable,} \\ \text{and } \mathrm{focc}(x_\eta, M) \leq 1 \end{array}$$

How can one detect whether x_η is an argument variable? To this end we require that the sequent $\Gamma \vdash \beta/\alpha$ be derivable in categorial AB^-. This seems paradoxical. For with this restriction the calculus seems to be as weak as AB^-. Why should one make use of the rule (**I**–/) if the sequent is anyway derivable? To understand this one should take note of the difference between the categorial calculus and the interpreted calculus. We allow the use of the interpreted rule (**I**–/) if $\Gamma \vdash \beta/\alpha$ is derivable in the categorial calculus; or, to be more prosaic, if Γ has the category β/α and hence the type $\alpha \to \beta$. That this indeed strengthens the calculus can be seen as follows. In the interpreted AB^- the following sequent is not derivable (though it is derivable in AB). The proof of this claim is left as an exercise.

$$\text{(3.201)} \qquad \alpha/\beta : x_{\zeta\to\eta} \vdash \alpha/\beta : \lambda x_\zeta.x_{\zeta\to\eta} x_\zeta$$

We assign to a sign complex a sign as follows.

$$
\begin{aligned}
&\S(\vec{x} : \alpha : M) := \langle \vec{x}, \alpha, M \rangle \\
(3.202) \qquad &\S(\vec{x} : \alpha/\beta : M \circ \vec{y} : \beta : N) := \mathtt{A}_{>}(\langle \vec{x}, \alpha/\beta, M \rangle, \langle \vec{y}, \beta, N \rangle) \\
&\S(\vec{x} : \beta : M \circ \vec{y} : \beta\backslash\alpha : N) := \mathtt{A}_{<}(\langle \vec{x}, \beta, M \rangle, \langle \vec{y}, \beta\backslash\alpha, N \rangle)
\end{aligned}
$$

It is easy to see that if $\Gamma \vdash \alpha : M$ is derivable in the interpreted AB then $\S(\Gamma) = \langle \vec{x}, \alpha, M' \rangle$ for some $M' \equiv M$. (Of course, M and M' are just notational variants denoting the same object. Thus they are identical qua objects they represent.)

The calculus that has just been defined has drawbacks. We will see below that (cut) cannot be formulated for strings. Thus, we have to do without it. But then we cannot derive the rules (**E**–/) and (**E**–\). The calculus N obviates the need for that.

Definition 3.71 *The calculus* N *has the rules (ax),* (**I**–/)*,* (**E**–/) (**I**–/) *and* (**E**–\).

In (the interpreted) N, (cut) is admissible. (The proof of that is left as an exercise.)

Now let us turn to strings. Now we omit the interpretation, since it has been dealt with. Our objects are now written as $\vec{x} : \alpha$ where $\vec{x}$ is a string and α a category. The reader is reminded of the fact that $\vec{y}/\vec{x}$ denotes that string which results from $\vec{y}$ by removing the postfix $\vec{x}$. This is clearly defined only if $\vec{y} = \vec{u}^\frown\vec{x}$ for some $\vec{u}$, and then we have $\vec{y}/\vec{x} = \vec{u}$. Analogously for $\vec{x}\backslash\vec{y}$.

$$
(3.203) \qquad
\begin{array}{lll}
\text{(ax)} & \vec{x} : \alpha \vdash \vec{x} : \alpha & \\
(\mathbf{I}\text{–}/) & \dfrac{\Gamma \circ \vec{x} : \alpha \vdash \vec{y} : \beta}{\Gamma \vdash \vec{y}/\vec{x} : \beta/\alpha} & (\mathbf{E}\text{–}/) \quad \dfrac{\Gamma \vdash \vec{x} : \alpha \quad \Delta \vdash \vec{y} : \beta/\alpha}{\Delta \circ \Gamma \vdash \vec{y}^\frown\vec{x} : \beta} \\
(\mathbf{I}\text{–}\backslash) & \dfrac{\vec{x} : \alpha \circ \Gamma \vdash \vec{y} : \beta}{\Gamma \vdash \vec{x}\backslash\vec{y} : \alpha\backslash\beta} & (\mathbf{E}\text{–}\backslash) \quad \dfrac{\Gamma \vdash \vec{x} : \alpha \quad \Delta \vdash \vec{y} : \alpha\backslash\beta}{\Gamma \circ \Delta \vdash \vec{x}^\frown\vec{y} : \beta}
\end{array}
$$

The cut rule is however no more a rule of the calculus. There is no formulation of it at all. Suppose we try to formulate a cut rule. Then it would go as follows.

$$
(3.204) \qquad \frac{\Gamma \vdash \vec{x} : \alpha \qquad \Delta[\vec{y} : \alpha] \vdash \vec{z} : \beta}{\Delta[\Gamma] \vdash [\vec{y}/\vec{x}]\vec{z} : \beta}
$$

Here, $[\vec{y}/\vec{x}]\vec{z}$ denotes the result of replacing $\vec{y}$ for $\vec{x}$ in $\vec{z}$. So, on the strings (cut) becomes constituent replacement. Notice that only one occurrence may

be replaced, so if $\vec{x}$ occurs several times, the result of the operation $[\vec{y}/\vec{x}]\vec{z}$ is not uniquely defined. Moreover, $\vec{x}$ may occur accidentally in $\vec{z}$! Thus, it is not clear which of the occurrences is the right one to be replaced. So the rule of (cut) cannot even be properly formulated. On the other hand, semantically it is admissible, so for the semantics we can do without it anyway. However, the same problem of substitution arises with the rules (/–**I**) and (\–**I**). Thus, they had to be eliminated as well.

This completes the definition of the sign calculus N. Call E the calculus consisting of just (/–**E**) and (\–**E**). Based on Lemma 3.70 the completeness of E for $\Vdash$ is easily established.

Theorem 3.72 $\overset{\mathsf{E}}{\leadsto} \Gamma \vdash \sigma$ *iff* $\Gamma \Vdash \sigma$.

N is certainly correct for the global interpretation, but it is correct for the intrinsic interpretation? The answer is actually yes! The fact is that the introduction rules are toothless tigers: they can only eliminate a variable that has never had a chance to play a role. For assume that we have an N–proof. If (**I**–/) is used, let the highest application be as follows.

$$\frac{\Gamma \circ \varepsilon : \alpha : x_\eta \vdash \vec{y} : \beta : M}{\Gamma \vdash \vec{y} : \beta/\alpha : (\lambda x_\eta . M)} \tag{3.205}$$

Then the sequent above the line has been introduced by (**E**–/):

$$\frac{\Gamma \vdash \vec{y} : \beta/\alpha : N \qquad \varepsilon : \alpha : x_\eta \vdash \varepsilon : \alpha : x_\eta}{\Gamma \circ \varepsilon : \alpha : x_\eta \vdash \vec{y} : \beta : Nx_\eta} \tag{3.206}$$

Here, $Nx_\eta = M$. Since $(\lambda x_\eta . M) = (\lambda x_\eta . Nx_\eta) = N$, this part of the proof can be eliminated.

Theorem 3.73 *The rules* (**I**–/) *and* (**I**–\) *are admissible in* E.

In the next section, however, we shall study the effect of adding associativity. In presence of associativity the introduction rules actually do considerable work. In that case, to regain correctness, we can either ban the introduction rules, or we can restrict the axioms. Given an AB–sign grammar $\mathfrak{A}$ we can restrict the set of axioms to

$$(\mathrm{ax}_{\mathfrak{A}}) \quad \upsilon(f) \vdash \upsilon(f) \qquad \text{where } f \in F \text{ and } \Omega(f) = 0 \tag{3.207}$$

For an AB–grammar does not possess any modes of the form $\langle \varepsilon, \alpha, x_\alpha \rangle$ where x_α is a variable.

Exercise 116. Prove the correctness theorem, Proposition 3.67.

Exercise 117. Define a function p from category complexes to categories as follows.

$$\begin{aligned} p(\alpha) &:= \alpha \\ p(\Gamma \circ \Delta) &:= p(\Gamma) \cdot p(\Delta) \end{aligned} \tag{3.208}$$

Show that $\Gamma \vdash \alpha$ is derivable in AB^- iff $p(\Gamma) = \alpha$. Show that this also holds for $\mathsf{AB}^- + (\text{cut})$. Conclude from this that (cut) is admissible in (categorial!) AB^-. (This could in principle be extracted from the proof for AB, but this proof here is quite simple.)

Exercise 118. Show that every CFL can be generated by an AB–grammar using only two basic categories.

Exercise 119. Show the following claim: in the interpreted AB^-–calculus no sequents are derivable which contain bound variables.

Exercise 120. Show that (cut) is admissible for N.

6. The Lambek–Calculus

The Lambek–Calculus, L, is in many respects an extension of AB. It has been introduced in [**?**]. In contrast to AB, in L categories are not interpreted as sets of labelled trees but as sets of strings. This means that the calculus has different laws. Furthermore, L possesses a new category constructor, *pair formation*; it is written $\bullet$ and has a counterpart on the level of categories, also denoted by that symbol. The constructors of the classical Lambek–calculus for categories therefore are $\backslash$, $/$ and $\bullet$. Given an alphabet A and an elementary category assignment ζ we denote by $\{\alpha\}_\zeta$ the set of all strings over A which are of category α with respect to ζ. Then the following holds.

$$\begin{aligned} \{\alpha \bullet \beta\}_\zeta &:= \{\alpha\}_\zeta \cdot \{\beta\}_\zeta \\ \{\Gamma \circ \Delta\}_\zeta &:= \{\Gamma\}_\zeta \cdot \{\Delta\}_\zeta \end{aligned} \tag{3.209}$$

Since we have the constructor $\bullet$ at our disposal, we can in principle dispense with the symbol $\circ$. However, we shall not do so. We shall formulate the calculus as before using $\circ$, which makes it directly comparable to the ones we have defined above. Hence as before we distinguish *terms* from *structures*.

We write $\Gamma \vdash \beta$ if $\{\Gamma\}_\zeta \subseteq \{\alpha\}_\zeta$. We shall axiomatize the sequents of $\vdash$. In order to do so we add the following rules to the calculus AB (without (cut)).

$$\textbf{(ass1)}\ \frac{\Gamma[\Delta_1 \circ (\Delta_2 \circ \Delta_3)] \vdash \alpha}{\Gamma[(\Delta_1 \circ \Delta_2) \circ \Delta_3] \vdash \alpha} \qquad \textbf{(ass2)}\ \frac{\Gamma[(\Delta_1 \circ \Delta_2) \circ \Delta_3] \vdash \alpha}{\Gamma[\Delta_1 \circ (\Delta_2 \circ \Delta_3)] \vdash \alpha}$$

$$\textbf{(}\bullet\textbf{–I)}\ \frac{\Gamma[\alpha \circ \beta] \vdash \gamma}{\Gamma[\alpha \bullet \beta] \vdash \gamma} \qquad \textbf{(I–}\bullet\textbf{)}\ \frac{\Gamma \vdash \alpha \qquad \Delta \vdash \beta}{\Gamma \circ \Delta \vdash \alpha \bullet \beta}$$

This calculus is called the **Lambek–Calculus**, or simply L. Further, we put $\mathsf{NL} := \mathsf{AB} + (\textbf{I–}\bullet) + (\bullet\textbf{–I})$ and $\mathsf{NL}^- := \mathsf{AB}^- + (\bullet\textbf{–I}) + (\textbf{I–}\bullet)$. NL is also called the **Nonassociative Lambek–Calculus**.

Theorem 3.74 (Lambek) *(cut) is admissible for* L.

Corollary 3.75 (Lambek) L *with or without (cut) is decidable.*

For a proof we only have to look at applications of (cut) following an application of the new rules. Assume that the left hand premiss has been obtained by an application of (ass1).

$$\frac{\dfrac{\Gamma[\Theta_1 \circ (\Theta_2 \circ \Theta_2)] \vdash \alpha}{\Gamma[(\Theta_1 \circ \Theta_2) \circ \Theta_2] \vdash \alpha} \qquad \Delta[\alpha] \vdash \beta}{\Delta[\Gamma[(\Theta_1 \circ \Theta_2) \circ \Theta_3]] \vdash \beta} \tag{3.210}$$

This proof part we reformulate into the following one.

$$\frac{\dfrac{\Gamma[\Theta_1 \circ (\Theta_2 \circ \Theta_3)] \vdash \alpha \qquad \Delta[\alpha] \vdash \beta}{\Delta[\Gamma[\Theta_1 \circ (\Theta_2 \circ \Theta_3)]] \vdash \beta}}{\Delta[\Gamma[(\Theta_1 \circ \Theta_2) \circ \Theta_3]] \vdash \beta} \tag{3.211}$$

Analogously if the left hand premiss has been obtained by using (ass2). We leave it to the reader to treat the case where the right hand premiss has been obtained by using (ass1) or (ass2). We have to remark here that by reformulation we do not diminish the degree of the cut. So the original proof is not easily transported into the new setting. However, the *depth* of the application has been diminished. Here, depth means (intuitively) the length of a longest path through the proof tree from the top up to the rule occurrence. If we assume that $\Gamma[\Theta_1 \circ (\Theta_2 \circ \Theta_3)] \vdash \alpha$ has depth i and $\Delta[\alpha] \vdash \beta$ depth j then in the first tree the application of (cut) has depth $\max\{i, j\} + 1$, in the second however it has depth $\max\{i, j\}$.

Let us look at the cases of introduction of $\bullet$. The case of ($\bullet$–**I**) on the left hand premiss is easy.

$$(3.212)\qquad \dfrac{\dfrac{\Gamma[\theta_1 \circ \theta_2] \vdash \alpha}{\Gamma[\theta_1 \bullet \theta_2] \vdash \alpha} \qquad \Delta[\alpha] \vdash \gamma}{\Delta[\Gamma[\theta_1 \bullet \theta_2]] \vdash \gamma} \quad \rightsquigarrow \quad \dfrac{\dfrac{\Gamma[\theta_1 \circ \theta_2] \vdash \alpha \quad \Delta[\alpha] \vdash \gamma}{\Delta[\Gamma[\theta_1 \circ \theta_2]] \vdash \gamma}}{\Delta[\Gamma[\theta_1 \bullet \theta_2]] \vdash \gamma}$$

Now for the case of (**I**–$\bullet$) on the right hand premiss.

$$(3.213)\qquad \dfrac{\Gamma \vdash \alpha \qquad \dfrac{\Theta_1 \vdash \theta_1 \qquad \Theta_2 \vdash \theta_2}{\Delta[\alpha] \vdash \gamma}}{\Delta[\Gamma] \vdash \gamma}$$

In this case $\gamma = \theta_1 \bullet \theta_2$. Furthermore, $\Delta = \Theta_1 \circ \Theta_2$ and the marked occurrence of α either is in Θ_1 or in Θ_2. Without loss of generality we assume that it is in Θ_1. Then we can replace the proof by

$$(3.214)\qquad \dfrac{\dfrac{\Gamma \vdash \alpha \qquad \Theta_1[\alpha] \vdash \theta_1}{\Theta_1[\Gamma] \vdash \theta_1} \qquad \Theta_2 \vdash \theta_2}{\Theta_1[\Gamma] \circ \Theta_2 \vdash \theta_1 \bullet \theta_2}$$

We have $\Theta_1[\Gamma] \circ \Theta_2 = \Delta[\Gamma]$ by hypothesis on the occurrence of α. Now we look at the case where the left hand premiss of cut has been introduced by (**I**–$\bullet$). We may assume that the right hand premiss has been obtained through application of ($\bullet$–**I**). The case where α is a side formula is once again easy. So let α be main formula. We get the following local tree.

$$(3.215)\qquad \dfrac{\dfrac{\Theta_1 \vdash \theta_1 \qquad \Theta_2 \vdash \theta_2}{\Theta_1 \circ \Theta_2 \vdash \theta_1 \bullet \theta_2} \qquad \dfrac{\Delta[\theta_1 \circ \theta_2] \vdash \gamma}{\Delta[\theta_1 \bullet \theta_2] \vdash \gamma}}{\Delta[\Theta_1 \circ \Theta_2] \vdash \gamma} \quad \rightsquigarrow \quad \dfrac{\Theta_2 \vdash \theta_2 \qquad \dfrac{\Theta_1 \vdash \theta_1 \qquad \Delta[\theta_1 \circ \theta_2] \vdash \gamma}{\Delta[\Theta_1 \circ \Theta_2] \vdash \gamma}}{\Delta[\Theta_1 \circ \Theta_2] \vdash \gamma}$$

In all cases the cut–weight (or the sum of the depth of the cuts) has been reduced.

We shall also present a different formulation of L using natural deduction over ordered DAGs. Here are the rules:

$$
\begin{array}{llll}
(\mathbf{I}\text{–}\bullet) & \dfrac{\alpha \quad \beta}{(\alpha \bullet \beta)} & (\mathbf{E}\text{–}\bullet) & \dfrac{(\alpha \bullet \beta)}{\alpha \quad \beta} \\[2ex]
(\mathbf{I}\text{–}/) & \dfrac{\dfrac{\begin{array}{c}[\alpha]\\ \vdots\end{array}}{\beta}}{(\alpha/\beta)} & (\mathbf{E}\text{–}/) & \dfrac{\beta \quad (\beta\backslash\alpha)}{\alpha} \\[2ex]
(\mathbf{I}\text{–}\backslash) & \dfrac{\dfrac{\begin{array}{c}[\alpha]\\ \vdots\end{array}}{\beta}}{(\alpha/\beta)} & (\mathbf{E}\text{–}\backslash) & \dfrac{(\alpha/\beta) \quad \beta}{\alpha}
\end{array}
\tag{3.216}
$$

These rules are very much like the natural deduction rules for intuitionistic logic. However, two differences must be noted. First, suppose we disregard for the moment the rules for $\bullet$. (This would incidentally give exactly the natural deduction calculus corresponding to AB.) The rules must be understood to operate on ordered trees. Otherwise, the difference between then rules for / and the rules for \ would be obliterated. Second, the elimination rule for $\bullet$ creates two linearly ordered daughters for a node, thus we not only create ordered trees, we in fact create ordered DAGs. We shall not spell out exactly how the rules are interpreted in terms of ordered DAGs, but we shall point out a few noteworthy things. First, this style of presentation is very much linguistically oriented. We may in fact proceed in the same way as for AB and define algorithms that decorate strings with certain categorial labels and proceed downward using the rules shown above. Yet, it must be clear that the so created structures cannot be captured by constituency rules (let alone rules of a CFG) for the simple reason that they are not trees. The following derivation is illustrative of this.

$$
\begin{array}{ccc}
\multicolumn{3}{c}{(\alpha \bullet (\alpha\backslash\gamma)/\beta) \bullet \beta} \\
\hline
\multicolumn{2}{c}{\alpha \bullet (\alpha\backslash\gamma)/\beta} & \beta \\
\hline
\alpha & (\alpha\backslash\gamma)/\beta & \beta \\
\hline
\vdots & \alpha\backslash\gamma & \\
\hline
\multicolumn{3}{c}{\gamma}
\end{array}
\tag{3.217}
$$

Notice that if a rule has two premisses, these must be adjacent and follow each other in the order specified in the rule. No more is required. This allows among other to derive associativity, that is, $(\alpha \bullet \beta) \bullet \gamma \dashv\vdash \alpha \bullet (\beta \bullet \gamma)$. However, notice the role of the so–called assumptions and their discharge. Once an assumption is discharged, it is effectively removed, so that the items to its left and its right are now adjacent. This plays a crucial role in the derivation of the rule of function composition.

$$\dfrac{\dfrac{\alpha/\beta \qquad \dfrac{\beta/\gamma \qquad \gamma^{\surd}}{\beta}}{\alpha}}{\alpha/\gamma} \tag{3.218}$$

As soon as the assumption γ is removed, the top sequence reads $\alpha/\beta, \beta/\gamma$.

The relationship with L is as follows. Let Γ be a sequence of categories. We interpret this as a labelled DAG, which is linearly ordered. Now we successively apply the rules above. It is verified that each rule application preserves the property that the leaves of the DAG are linearly ordered. Define a category corresponding to a sequence as follows.

$$\begin{aligned} \alpha^{\bullet} &:= \alpha \\ (\alpha, \Delta)^{\bullet} &:= \alpha \bullet \Delta^{\bullet} \end{aligned} \tag{3.219}$$

First of all we say that for two sequences Δ and Δ', Δ' is **derivable from** Δ in the natural deduction style calculus if there is a DAG constructed according to the rules above, whose topmost sequence is Δ and whose lowermost sequence is Δ'. (Notice that assumptions get discharged, so that we cannot simply say that Δ is the sequence we started off with.) The following is then shown by induction.

Theorem 3.76 *Let Δ and Θ be two sequences of categories. Θ is derivable from Δ iff $\Delta \vdash \Theta^{\bullet}$ is derivable in L.*

This shows that the natural deduction style calculus is effectively equivalent to L.

L allows for a result akin to the Curry–Howard–Isomorphism. This is an extension of the latter result in two respects. First, we have the additional type constructor $\bullet$, which we have to match by some category constructor, and second, there are different structural rules. First, the new type constructor is actually the pair–formation.

Table 8. L with λ–Term Annotation

$$\text{(ax)} \quad x : \varphi \vdash x : \varphi$$

$$\text{(cut)} \quad \frac{\Gamma \vdash M : \varphi \quad \Delta, x : \varphi, \Theta \vdash N : \beta}{\Delta, \Gamma, \Theta \vdash [M/x]N : \beta}$$

$$(\mathbf{E}\text{–}/) \quad \frac{\Gamma \vdash M : \alpha/\beta \quad \Delta \vdash N : \beta}{\Gamma, \Delta \vdash MN : \alpha} \qquad (\mathbf{I}\text{–}/) \quad \frac{\Gamma, x : \beta \vdash M : \alpha}{\Gamma \vdash \lambda x.M : \alpha/\beta}$$

$$(\mathbf{E}\text{–}\backslash) \quad \frac{\Gamma \vdash M : \beta\backslash\alpha \quad \Delta \vdash N : \beta}{\Delta, \Gamma \vdash MN : \alpha} \qquad (\mathbf{I}\text{–}\backslash) \quad \frac{x : \beta, \Gamma \vdash M : \alpha}{\Gamma \vdash \lambda x.M : \beta\backslash\alpha}$$

$$(\mathbf{E}\text{–}\bullet) \quad \frac{\Gamma \vdash M : \alpha \bullet \beta \quad \Delta, x : \alpha, y : \beta, \Theta \vdash U : \psi}{\Delta, \Gamma, \Theta \vdash [p_1(M)/x][p_2(M)/y]U : \psi}$$

$$(\mathbf{I}\text{–}\bullet) \quad \frac{\Gamma \vdash M : \alpha \quad \Delta \vdash N : \beta}{\Gamma, \Delta \vdash \langle M, N \rangle : \alpha \bullet \beta}$$

Definition 3.77 *Every λ–term is a $\lambda^\bullet$–**term**. Given two $\lambda^\bullet$–terms M and N, $\langle M,N \rangle$, $p_1(M)$ and $p_2(M)$ also are $\lambda^\bullet$–**terms**. Further, the following equations hold.*

$$(3.220) \qquad p_1(\langle M,N \rangle) = M, \qquad p_2(\langle M,N \rangle) = N.$$

*$p_1(U)$ and $p_2(U)$ are not defined if U is not of the form $\langle M,N \rangle$ for some M and N. The functions p_1 and p_2 are called the **projections**.*

Notice that antecedents of sequents no longer consist of sets of sequences. Hence, Γ, Δ, Θ now denote sequences rather than sets. In Table 8 we display the new calculus. We have also put a general constraint on the proofs that variables may not be used twice. To implement this constraint, we define the notion of a linear term:

Definition 3.78 *A term M is **strictly linear** if for every variable x and every subterm N,* $\mathrm{focc}(x,N) \leq 1$. *A term is **linear** if it results from a strictly linear term M by iterated replacement of a subterm M' by $[p_1(N)/x][p_2(N)/y]M'$, where N is a linear term.*

The calculus above yields only linear terms if we start with variables and require in the rules (**I**–$\bullet$), (**E**–/), (**E**–\) that the sets of free variables be disjoint, and that in (**I**–/) and (**I**–\) the variable occurs exactly once free in M.

In this way we can ensure that for every sequent derivable in L there actually exists a labelling such that the labelled sequent is derivable in the labelled calculus. This new calculus establishes a close correspondence between linear $\lambda^{\bullet}$–terms and the so–called multiplicative fragment of linear logic, which naturally arises from the above calculus by stripping off the terms and leaving only the formulae. A variant of proof normalization can be shown, and all this yields that L has quite well–behaved properties.

In presence of the rules (ass1) and (ass2) $\bullet$ behaves exactly like concatenation, that is, it is a fully associative operation. Therefore we shall change the notation in what is to follow. In place of structures consisting of categories we shall consider finite sequences of categories, that is, strings over $\mathrm{Cat}_{\backslash,\bullet,/}(C)$. We denote concatenation by comma, as is commonly done.

Now we return to the theory of meaning. In the previous section we have seen how to extend AB by a component for meanings which computes the meaning in tandem with the category. We shall do the same here. To this end we shall have to first clarify what we mean by a realization of $\alpha \bullet \beta$. We shall agree on the following.

(3.221) $\ulcorner \alpha \bullet \beta \urcorner := \ulcorner \alpha \urcorner \times \ulcorner \beta \urcorner$

The rules are tailored to fit this interpretation. They are as follows.

(3.222) $$\frac{\Gamma[\Delta_1 \circ (\Delta_2 \circ \Delta_3)] \vdash \alpha : M}{\Gamma[(\Delta_1 \circ \Delta_2) \circ \Delta_3] \vdash \alpha : M}$$

This means that the restructuring of the term is without influence on its meaning. Likewise we have

(3.223) $$\frac{\Gamma[(\Delta_1 \circ \Delta_2) \circ \Delta_3] \vdash \alpha : M}{\Gamma[\Delta_1 \circ (\Delta_2 \circ \Delta_3)] \vdash \alpha : M}$$

So, for $\bullet$ we assume the following rule.

(3.224) $$\frac{\Gamma[\alpha : x_\alpha \circ \beta : x_\beta] \vdash \gamma : M}{\Gamma[\alpha \bullet \beta] \vdash \gamma : [p_1(z_{\alpha\bullet\beta})/x_\alpha, p_2(z_{\alpha\bullet\beta})/x_\beta]M}$$

(3.224) says that in place of a function of two arguments α and β we can form a function of a single argument of type $\alpha \bullet \beta$. The two arguments we can recover by application of the projection functions. The fourth rule finally tells us how the type/category $\alpha \bullet \beta$ is interpreted.

(3.225) $$\frac{\Gamma \vdash \alpha : M \qquad \Delta \vdash \beta : N}{\Gamma \circ \Delta \vdash \alpha \bullet \beta : \langle M, N \rangle}$$

Table 9. The Sign Based Calculus L

(ax)	$\vec{x} : \alpha : x_\zeta \vdash \vec{x} : \alpha : x_\zeta,$	$\zeta = \sigma(\alpha)$
(**I**–/)	$\dfrac{\Gamma \circ \vec{x} : \alpha : x_\zeta \vdash \vec{y} : \beta : N}{\Gamma \vdash \vec{y}/\vec{x} : \beta/\alpha : \lambda x_\zeta.N}$	x_ζ an argument variable, $\mathrm{focc}(x_\zeta, N) \leq 1$
(**I**–\)	$\dfrac{\vec{x} : \alpha : x_\zeta \circ \Gamma \vdash \vec{y} : \beta : N}{\Gamma \vdash \vec{x}\backslash\vec{y} : \alpha\backslash\beta : \lambda x_\zeta.N}$	x_ζ an argument variable, $\mathrm{focc}(x_\zeta, N) \leq 1$
(**E**–/)	$\dfrac{\Gamma \vdash \vec{x} : \alpha : M \qquad \Delta \vdash \vec{y} : \beta/\alpha : N}{\Delta \circ \Gamma \vdash \vec{y}^\frown\vec{x} : \beta : NM}$	
(**E**–\)	$\dfrac{\Gamma \vdash \vec{x} : \alpha : M \qquad \Delta \vdash \vec{y} : \alpha\backslash\beta : N}{\Gamma \circ \Delta \vdash \vec{x}^\frown\vec{y} : \beta : NM}$	
(•–**I**)	$\dfrac{\vec{x} : \alpha : x_\zeta \circ \vec{y} : \beta : y_\eta \vdash \vec{z} : \gamma : M}{\vec{x}^\frown\vec{y} : \alpha \bullet \beta : z_{\zeta\bullet\eta} \vdash \vec{z} : \gamma : [p_1(z_{\zeta\bullet\eta})/x_\zeta, p_2(z_{\zeta\bullet\eta})/y_\eta]M}$	
(**I**–•)	$\dfrac{\Gamma \vdash \vec{x} : \alpha : M \qquad \Delta \vdash \vec{y} : \beta : N}{\Gamma \circ \Delta \vdash \vec{x}^\frown\vec{y} : \alpha \bullet \beta : \langle M, N\rangle}$	

Here we have the same problem as before with AB. The meaning assignments that are being computed are not in full accord with the interpretation. The term $(\mathtt{x}_0(\mathtt{x}_1\mathtt{x}_2))$ does not denote the same function as $((\mathtt{x}_0\mathtt{x}_1)\mathtt{x}_2)$. (Usually, one of them is not even well defined.) So this raises the question whether it is at all legitimate to proceed in this way. We shall avoid the question by introducing a totally different calculus, sign based L (see Table 9), which builds on the calculus N of the previous section. The rules (**ass1**) and (**ass2**) are dropped. Furthermore, (•–**I**) is restricted to $\Gamma = \varnothing$. These restrictions are taken over from N for the abstraction rules. Sign based L has the global side condition that no variable is used in two different leaves. This condition can be replaced (up to α–conversion) by the condition that all occurring terms are linear. In turn, this can be implemented by the adding suitable side conditions on the rules.

Sign based L is not as elegant as plain categorial L. However, it is semantically correct. If one desperately wants to have associativity, one has to introduce combinators at the right hand side. So, a use of the associativity rule is accompanied in the semantics by a use of C with $\mathsf{C}MNP = M(NP)$. We shall not spell out the details here.

Exercise 121. Assume that in place of sequents of the form $\alpha \vdash \alpha$ for arbi-

trary α only sequents $c \vdash c$, $c \in C$, are axioms. Show that with the rules of L $\alpha \vdash \alpha$ can be derived for every α.

Exercise 122. Let $G = \langle C, S, A, \zeta, \mathsf{NL}^- \rangle$ be a categorial sequent grammar. Show that the language $\{\vec{x} : \vdash_G \vec{x}\}$ is context free.

Exercise 123. Show that the sequent $\alpha/\beta \circ \beta/\gamma \vdash \alpha/\gamma$ is derivable in L but not in AB. What semantics does the structure $\alpha/\beta \circ \beta/\gamma$ have?

Exercise 124. A **loop** is a structure $\langle L, \cdot, \backslash, / \rangle$ where $\Omega(\cdot) = \Omega(\backslash) = \Omega(/) = 2$ and the following equations hold for all $x, y \in L$.

$$x \cdot (x \backslash y) = y, \quad (y/x) \cdot x = y \tag{3.226}$$

The categories do not form a loop with respect to $\backslash$, $/$ and $\cdot$ (!), for the reason that $\cdot$ is only partially defined. Here is a possible remedy. Define $\approx \subseteq \mathrm{Cat}_{\backslash, \bullet, /}(C)^2$ to be the least congruence such that

$$(\alpha \bullet \beta)/\beta \approx \alpha, \qquad \beta \backslash (\beta \bullet \alpha) \approx \alpha \tag{3.227}$$

Show that the free algebra of categories over C factored by $\approx$ is a loop. What is $\alpha \cdot \beta$ in the factored algebra?

Exercise 125. Show that the following rules are admissible in L.

$$(\bullet\text{–}\mathbf{E})\ \frac{\Gamma[\theta_1 \bullet \theta_2] \vdash \alpha}{\Gamma[\theta_1 \circ \theta_2] \vdash \alpha} \quad (\mathbf{E}\text{–}/)\ \frac{\Gamma \vdash \alpha/\beta}{\Gamma \circ \beta \vdash \alpha} \quad (\mathbf{E}\text{–}\backslash)\ \frac{\Gamma \vdash \beta \backslash \alpha}{\beta \circ \Gamma \vdash \alpha} \tag{3.228}$$

7. Pentus' Theorem

It was conjectured by Noam Chomsky that the languages generated by L are context free, which means that L is in effect not stronger than AB. This was first shown by Mati Pentus (see [**?**]). His proof makes use of the fact that L has interpolation. We start with a simple observation. Let $\mathfrak{G} := \langle G, \cdot, ^{-1}, 1 \rangle$ be a group and $\gamma \colon C \to G$. We extend γ to all types and structures as follows.

$$\begin{aligned} \gamma(\alpha \bullet \beta) &:= \gamma(\alpha) \cdot \gamma(\beta) \\ \gamma(\alpha/\beta) &:= \gamma(\alpha) \cdot \gamma(\beta)^{-1} \\ \gamma(\beta \backslash \alpha) &:= \gamma(\beta)^{-1} \cdot \gamma(\alpha) \\ \gamma(\Gamma \circ \Delta) &:= \gamma(\Gamma) \cdot \gamma(\Delta) \end{aligned} \tag{3.229}$$

We call γ a **group valued interpretation**.

Theorem 3.79 (Roorda) *If $\Gamma \vdash \alpha$ is derivable in L then for all group valued interpretations γ $\gamma(\Gamma) = \gamma(\alpha)$.*

The proof is performed by induction over the length of the derivation and is left as an exercise. Let C be given and $c \in C$. For a category α over C we define

$$\sigma_c(c') := \begin{cases} 1 & \text{if } c = c', \\ 0 & \text{otherwise.} \end{cases}$$

$$\begin{aligned} \sigma_c(\alpha \bullet \beta) &:= \sigma_c(\alpha) + \sigma_c(\beta) \\ \sigma_c(\alpha / \beta) &:= \sigma_c(\alpha) + \sigma_c(\beta) \\ \sigma_c(\beta \backslash \alpha) &:= \sigma_c(\alpha) + \sigma_c(\beta) \end{aligned} \tag{3.230}$$

Likewise we define

$$|\alpha| := \sum_{c \in C} \sigma_c(\alpha) \tag{3.231}$$

$$\pi(\alpha) := \{c \in C : \sigma_c(\alpha) > 0\} \tag{3.232}$$

These definitions are extended in the canonical way to structures. Let Δ be a nonempty structure (that is, $\Delta \neq \varepsilon$) and $\Gamma[-]$ a structure containing a marked occurrence of a substructure. An **interpolant** for a sequent $\Gamma[\Delta] \vdash \alpha$ in a calculus $\mathcal{S}$ with respect to Δ is a category θ such that

① $\sigma_c(\theta) \leq \min\{\sigma_c(\Gamma) + \sigma_c(\alpha), \sigma_c(\Delta)\}$, for all $c \in C$,

② $\Delta \vdash \theta$ is derivable in $\mathcal{S}$,

③ $\Gamma[\theta] \vdash \alpha$ is derivable in $\mathcal{S}$.

In particular $\pi(\theta) \subseteq \pi(\Gamma \circ \alpha) \cap \pi(\Delta)$ if θ satisfies these conditions. We say that $\mathcal{S}$ has **interpolation** if for every derivable $\Gamma[\Delta] \vdash \alpha$ there exists an interpolant with respect to Δ.

We are interested in the calculi AB and L. In the case of L we have to remark that in presence of full associativity the interpolation property can be formulated as follows. We deal with sequents of the form $\Gamma \vdash \alpha$ where Γ is a sequence of categories. If $\Gamma = \Theta_1, \Delta, \Theta_2$ with $\Delta \neq \varepsilon$ then there is an interpolant with respect to Δ. For let Δ° be a structure in $\circ$ which corresponds to Δ (after omitting all occurrences of $\circ$). Then there exists a sequent $\Gamma^\circ \vdash \alpha$ which is derivable and in which Δ° occurs as a substructure.

Interpolation is shown by induction on the derivation. In the case of an axiom there is nothing to show. For there we have a sequent $\alpha \vdash \alpha$ and the marked structure Δ has to be α. In this case α is an interpolant. Now let us assume that the rule (**I**–/) has been applied to yield the final sequent. Further, assume that the interpolation property has been shown for the premisses. Then we have the following constellation.

$$\frac{\Gamma[\Delta] \circ \alpha \vdash \beta}{\Gamma[\Delta] \vdash \beta/\alpha} \tag{3.233}$$

We have to find an interpolant with respect to Δ. By induction hypothesis there is a formula θ such that $\Gamma[\theta] \circ \alpha \vdash \beta$ and $\Delta \vdash \theta$ are both derivable and $\sigma_c(\theta) \leq \min\{\sigma_c(\Gamma \circ \alpha \circ \beta), \sigma_c(\Delta)\}$ for all $c \in C$. Then also $\Gamma[\theta] \vdash \beta/\alpha$ and $\Delta \vdash \theta$ are derivable and we have $\sigma_c(\theta) \leq \min\{\sigma_c(\Gamma \circ \beta/\alpha), \sigma_c(\Delta)\}$. Hence θ also is an interpolant with respect to Δ in $\Gamma[\Delta] \vdash \beta/\alpha$. The case of (**I**–\) is fully analogous.

Now we look at the case that the last rule is (/–**I**).

$$\frac{\Gamma \vdash \beta \qquad \Delta[\alpha] \vdash \gamma}{\Delta[\alpha/\beta \circ \Gamma] \vdash \gamma} \tag{3.234}$$

Choose a substructure Z from $\Delta[\alpha/\beta \circ \Gamma]$. Several cases have to be distinguished. (1) Z is a substructure of Γ, that is, $\Gamma = \Gamma'[Z]$. Then there exists an interpolant θ for $\Gamma'[Z] \vdash \beta$ with respect to Z. Then θ also is an interpolant for $\Delta[\alpha/\beta \circ \Gamma'[Z]] \vdash \gamma$ with respect to Z. (2) Z is disjoint with $\alpha/\beta \circ \Gamma$. Then we have $\Delta[\alpha] = \Delta'[Z, \alpha]$ (with two marked occurrences of structures) and there is an interpolant θ with respect to Z for $\Delta'[Z, \alpha] \vdash \gamma$. Also in this case one calculates that θ is the desired interpolant. (3) $Z = \alpha/\beta$. By induction hypothesis there is an interpolant θ_ℓ for $\Gamma \vdash \beta$ with respect to Γ, as well as an interpolant θ_r for $\Delta[\alpha] \vdash \gamma$ with respect to α. Then $\theta := \theta_r/\theta_\ell$ is the interpolant. For we have

$$\begin{aligned} \sigma_c(\theta) &= \sigma_c(\theta_r) + \sigma_c(\theta_\ell) \\ &\leq \min\{\sigma_c(\Delta \circ \gamma), \sigma_c(\alpha)\} + \min\{\sigma_c(\beta), \sigma_c(\Gamma)\} \\ &\leq \min\{\sigma_c(\Delta \circ \Gamma \circ \gamma), \sigma_c(\alpha/\beta)\} \end{aligned} \tag{3.235}$$

Furthermore,

$$\frac{\Gamma \vdash \theta_\ell \qquad \Delta[\theta_r] \vdash \gamma}{\Delta[\theta_r/\theta_\ell \circ \Gamma] \vdash \gamma} \tag{3.236}$$

$$\cfrac{\cfrac{\theta_\ell \vdash \beta \qquad \alpha \vdash \theta_r}{\alpha/\beta \circ \theta_\ell \vdash \theta_r}}{\alpha/\beta \vdash \theta_r/\theta_\ell}$$

(4) $Z = \Theta[\alpha/\beta \circ \Gamma]$. Then $\Delta[\alpha/\beta \circ \Gamma] = \Delta'[\Theta[\alpha/\beta \circ \Gamma]]$ for some Δ'. Then by hypothesis there is an interpolant for $\Delta'[\Theta[\alpha]] \vdash \gamma$ with respect to $\Theta[\alpha]$. We show that θ is the desired interpolant.

$$\frac{\Gamma \vdash \beta \qquad \Theta[\alpha] \vdash \theta}{\Theta[\alpha/\beta \circ \Gamma] \vdash \theta} \qquad\qquad \Delta'[\theta] \vdash \gamma \tag{3.237}$$

In addition

$$\begin{aligned} \sigma_c(\theta) &\leq \min\{\sigma_c(\Delta' \circ \gamma), \sigma_c(\Theta[\alpha])\} \\ &\leq \min\{\sigma_c(\Delta' \circ \gamma), \sigma_c(\Theta[\alpha/\beta \circ \Gamma])\} \end{aligned} \tag{3.238}$$

This ends the proof for the case (**/–I**). The case (**\–I**) again is fully analogous.

Theorem 3.80 AB *has interpolation.* □

Now we move on to L. Clearly, we only have to discuss the new rules. Let us first consider the case where we add $\bullet$ together with its introduction rules. Assume that the last rule is (**•–I**).

$$\frac{\Gamma[\alpha \circ \beta] \vdash \gamma}{\Gamma[\alpha \bullet \beta] \vdash \gamma} \tag{3.239}$$

Choose a substructure Z of $\Gamma[\alpha \circ \beta]$. (1) Z does not contain the marked occurrence of $\alpha \bullet \beta$. Then $\Gamma[\alpha \bullet \beta] = \Gamma'[Z, \alpha \bullet \beta]$, and by induction hypothesis we get an interpolant θ for $\Gamma'[Z, \alpha \circ \beta] \vdash \gamma$ with respect to Z. It is easily checked that θ also is an interpolant for $\Gamma'[Z, \alpha \bullet \beta] \vdash \gamma$ with respect to Z. (2) Let $Z = \Theta[\alpha \bullet \beta]$. Then $\Gamma[\alpha \bullet \beta] = \Gamma'[\Theta[\alpha \bullet \beta]]$. By induction hypothesis there is an interpolant θ for $\Gamma'[\Theta[\alpha \circ \beta]] \vdash \gamma$ with respect to $\Theta[\alpha \circ \beta]$, and it also is an interpolant for $\Gamma'[\Theta[\alpha \bullet \beta]] \vdash \gamma$ with respect to $\Theta[\alpha \bullet \beta]$. In both cases we have found an interpolant.

Now we turn to the case (**I–•**).

$$\frac{\Gamma \vdash \alpha \qquad \Delta \vdash \beta}{\Gamma \circ \Delta \vdash \alpha \bullet \beta} \tag{3.240}$$

There are now three cases for Z. (1) $\Gamma = \Gamma'[Z]$. By induction hypothesis there is an interpolant θ_ℓ for $\Gamma'[Z] \vdash \alpha$ with respect to Z. This is the desired interpolant. (2) $\Delta = \Delta'[Z]$. Analogous to (1). (3) $Z = \Gamma \circ \Delta$. By hypothesis there is

an interpolant θ_ℓ for $\Gamma \vdash \alpha$ with respect to Γ and an interpolant θ_r for $\Delta \vdash \beta$ with respect to Δ. Put $\theta := \theta_\ell \bullet \theta_r$. This is the desired interpolant. For

$$(3.241) \qquad \frac{\Gamma \vdash \theta_\ell \qquad \Delta \vdash \theta_r}{\Gamma \circ \Delta \vdash \theta_\ell \bullet \theta_r} \qquad\qquad \frac{\dfrac{\theta_\ell \vdash \alpha \qquad \theta_r \vdash \beta}{\theta_\ell \circ \theta_r \vdash \alpha \bullet \beta}}{\theta_\ell \bullet \theta_r \vdash \alpha \bullet \beta}$$

In addition it is calculated that $\sigma_c(\theta) \leq \min\{\sigma_c(\alpha \bullet \beta), \sigma_c(\Gamma \circ \Delta)\}$.

This concludes a proof of interpolation for NL. Finally we must study L. The rules (**ass1**), (**ass2**) pose a technical problem since we cannot proceed by induction on the derivation. For the applications of these rules change the structure. Hence we change to another system of sequents and turn — as discussed above — to sequents of the form $\Gamma \vdash \alpha$ where Γ is a sequence of categories. In this case the rules (**ass1**) and (**ass2**) must be eliminated. However, in the proof we must make more distinctions in cases. The rules (**I**–/) and (**I**–\) are still unproblematic. So we look at a more complicated case, namely an application of the rule (/–**I**).

$$(3.242) \qquad \frac{\Gamma \vdash \beta \qquad \Delta[\alpha] \vdash \gamma}{\Delta[\alpha/\beta, \Gamma] \vdash \gamma}$$

We can segment the structure $\Delta[\alpha/\beta, \Gamma]$ into $\Delta', \alpha/\beta, \Gamma, \Delta''$. Let a subsequence Z be distinguished in $\Delta', \alpha/\beta, \Gamma, \Delta''$. The case where Z is fully contained in Δ' is relatively easy; likewise the case where Z is fully contained in Δ''. The following cases remain. (1) $Z = \Delta_1, \alpha/\beta, \Gamma_1$, where $\Delta' = \Delta_0, \Delta_1$ for some Δ_0, and $\Gamma = \Gamma_1, \Gamma_2$ for some Γ_2. Even if Z is not empty Δ_1 as well as Γ_1 may be empty. Assume $\Gamma_1 \neq \varepsilon$. In this case θ_ℓ an interpolant for $\Gamma \vdash \beta$ with respect to Γ_2 and θ_r an interpolant of $\Delta[\alpha] \vdash \gamma$ with respect to Δ_1, α. (Here it becomes clear why we need not assume $\Delta_1 \neq \varepsilon$.) The following sequents are therefore derivable.

$$(3.243) \qquad \begin{array}{cc} \Gamma_2 \vdash \theta_\ell & \Gamma_1, \theta_\ell \vdash \beta \\ \Delta_1, \alpha \vdash \theta_r & \Delta_0, \theta_r, \Delta'' \vdash \gamma \end{array}$$

Now put $\theta := \theta_r/\theta_\ell$. Then we have on the one hand

$$(3.244) \qquad \frac{\dfrac{\Delta_1, \alpha \vdash \theta_r \qquad \Gamma_1, \theta_\ell \vdash \beta}{\Delta_1, \alpha/\beta, \Gamma_1, \theta_\ell \vdash \theta_r}}{\Delta_1, \alpha/\beta, \Gamma_1 \vdash \theta_r/\theta_\ell}$$

and on the other

$$\frac{\Gamma_2 \vdash \theta_\ell \qquad \Delta_0, \theta_r, \Delta'' \vdash \gamma}{\Delta_0, \theta_r/\theta_\ell, \Gamma_2, \Delta'' \vdash \gamma} \tag{3.245}$$

The conditions on the numbers of occurrences of symbols are easy to check. (2) As Case (1), but Γ_1 is empty. Let then θ_ℓ be an interpolant for $\Gamma \vdash \beta$ with respect to Γ and θ_r an interpolant for $\Delta_0, \Delta_1, \alpha, \Delta'' \vdash \gamma$ with respect to Δ_1, α. Then put $\theta := \theta_r/\theta_\ell$. θ is an interpolant for the end sequent with respect to Z.

$$\frac{\dfrac{\theta_\ell \vdash \beta \qquad \Delta_1, \alpha \vdash \theta_r}{\Delta_1, \alpha/\beta, \theta_\ell \vdash \theta_r}}{\Delta_1, \alpha/\beta \vdash \theta_r/\theta_\ell} \qquad \frac{\Gamma \vdash \theta_\ell \qquad \Delta_0, \theta_r, \Delta'' \vdash \gamma}{\Delta_0, \theta_r/\theta_\ell, \Gamma, \Delta'' \vdash \gamma} \tag{3.246}$$

(3) Z does not contain the marked occurrence of α/β. In this case $Z = \Gamma_2, \Delta_1$ for some final part Γ_2 of Γ and an initial part Δ_1 of Δ''. Γ_2 as well as Δ_1 may be assumed to be nonempty, since otherwise we have a case that has already been discussed. The situation is therefore as follows with $Z = \Gamma_2, \Delta_1$.

$$\frac{\Gamma_1, \Gamma_2 \vdash \beta \qquad \Delta', \alpha, \Delta_1, \Delta_2 \vdash \gamma}{\Delta', \alpha/\beta, \Gamma_1, \Gamma_2, \Delta_1, \Delta_2 \vdash \gamma} \tag{3.247}$$

Let θ_ℓ be an interpolant for $\Gamma_1, \Gamma_2 \vdash \beta$ with respect to Γ_2 and θ_r an interpolant for $\Delta', \alpha, \Delta_1, \Delta_2 \vdash \gamma$ with respect to Δ_1. Then the following are derivable

$$\begin{array}{ll} \Gamma_2 \vdash \theta_\ell & \Gamma_1, \theta_\ell \vdash \beta \\ \Delta_1 \vdash \theta_r & \Delta', \alpha, \theta_r, \Delta_2 \vdash \gamma \end{array} \tag{3.248}$$

Now we choose $\theta := \theta_\ell \bullet \theta_r$. Then we have both

$$\frac{\Gamma_2 \vdash \theta_\ell \qquad \Delta_1 \vdash \theta_r}{\Gamma_2, \Delta_1 \vdash \theta_\ell \bullet \theta_r} \tag{3.249}$$

and

$$\frac{\dfrac{\Gamma_1, \theta_\ell \vdash \beta \qquad \Delta', \alpha, \theta_r, \Delta_2 \vdash \gamma}{\Delta', \alpha/\beta, \Gamma_1, \theta_\ell, \theta_r, \Delta_2 \vdash \gamma}}{\Delta', \alpha/\beta, \Gamma_1, \theta_\ell \bullet \theta_r, \Delta_2 \vdash \gamma} \tag{3.250}$$

In this case as well the conditions on numbers of occurrences are easily checked. This exhausts all cases. Notice that we have used $\bullet$ to construct the interpolant. In the case of the rules (**I**–$\bullet$) and ($\bullet$–**I**) there are no surprises with respect to AB.

Theorem 3.81 (Roorda) L *has interpolation.* □

Now we shall move on to show that L is context free. To this end we introduce a series of weak calculi of which we shall show that together they are not weaker than L. These calculi are called L_m, $m < \omega$. The axioms of L_m are sequents $\Gamma \vdash \alpha$ such that the following holds.

① $\Gamma = \beta_1, \beta_2$ or $\Gamma = \beta_1$ for certain categories β_1 and β_2.

② $\Gamma \vdash \alpha$ is derivable in L.

③ $|\alpha|, |\beta_1|, |\beta_2| < m$.

(cut) is the only rule of inference. The main work is in the proof of the following theorem.

Theorem 3.82 (Pentus) *Let* $\Gamma = \beta_0, \beta_1, \ldots, \beta_{n-1}$. $\Gamma \vdash \alpha$ *is derivable in* L_m *iff*

① $|\beta_i| < m$ *for all* $i < m$,

② $|\alpha| < m$ *and*

③ $\Gamma \vdash \alpha$ *is derivable in* L.

We shall show first how to get from this fact that L–grammars are context free. We weaken the calculi still further. The calculus $\mathsf{L}_m^{\boxminus}$ has the axioms of L_m but (cut) may be applied only if the left hand premiss is an axiom.

Lemma 3.83 *For all sequents* $\Gamma \vdash \alpha$ *the following holds:* $\Gamma \vdash \alpha$ *is derivable in* $\mathsf{L}_m^{\boxminus}$ *iff* $\Gamma \vdash \alpha$ *is derivable in* L_m.

The proof is relatively easy and left as an exercise.

Theorem 3.84 *The languages accepted by* L*–grammars are context free.*

Proof. Let $\mathbb{L} = \langle \mathtt{S}, C, \zeta, A, \mathsf{L} \rangle$ be given. Let m be larger than the maximum of all $|\alpha|$, $\alpha \in \zeta(a)$, $a \in A$. Since A as well as $\zeta(a)$ are finite, m exists. For simplicity we shall assume that $C = \bigcup \langle \pi(\alpha) : \alpha \in \zeta(a), a \in A \rangle$. Now we put $N := \{\alpha : |\alpha| < m\}$. $G := \langle \mathtt{S}, N, A, R \rangle$, where

$$
\begin{aligned}
R := \quad & \{\alpha \to a : a \in \zeta(a)\} \\
& \cup \{\alpha \to \beta : \alpha, \beta \in N, \overset{\mathsf{L}}{\rightsquigarrow} \beta \vdash \alpha\} \\
& \cup \{\alpha \to \beta_0\beta_1 : \alpha, \beta_0, \beta_1 \in N, \overset{\mathsf{L}}{\rightsquigarrow} \beta_0, \beta_1 \vdash \alpha\}
\end{aligned} \tag{3.251}
$$

Now let $\mathbb{L} \vdash \vec{x}, \vec{x} = x_0{}^\frown x_1{}^\frown \cdots x_{n-1}$. Then for all $i < n$ there exist an $\alpha_i \in \zeta(x_i)$ such that $\Gamma \vdash \mathtt{S}$ is derivable in L, where $\Gamma := \alpha_0, \alpha_1, \ldots, \alpha_{n-1}$. By Theorem 3.82 and Lemma 3.83 $\Gamma \vdash \mathtt{S}$ is also derivable in $\mathsf{L}_m^{\boxminus}$. Induction over the length of the derivation yields that $\vdash_G \alpha_0{}^\frown \alpha_1{}^\frown \cdots {}^\frown \alpha_{n-1}$ and hence also $\vdash_G \vec{x}$. Now let conversely $\vdash_G \vec{x}$. We extend the category assignment ζ to $\zeta^+ \colon A \cup N \to \mathrm{Cat}_{\backslash,\bullet,/}(C)$ by putting $\zeta^+(\alpha) := \{\alpha\}$ while $\zeta^+ \restriction A = \zeta$. By induction over the length of the derivation of $\vec{\alpha}$ one shows that from $\vdash_G \vec{\alpha}$ we get $\mathbb{L} \vdash \vec{\alpha}$. □

Now on to the proof of Theorem 3.82.

Definition 3.85 *A category α is called **thin** if $\sigma_c(\alpha) \leq 1$ for all $c \in C$. A sequent $\Gamma \vdash \alpha$ is called **thin** if the following holds.*

① $\Gamma \vdash \alpha$ *is derivable in* L.

② *All categories occurring in Γ as well as α are thin.*

③ $\sigma_c(\Gamma, \alpha) \leq 2$ *for all $c \in C$.*

For a thin category α we always have $|\alpha| = |\pi(\alpha)|$. We remark that for a thin sequent only $\sigma_c(\Gamma, \alpha) = 0$ or $= 2$ can occur since $\sigma_c(\Gamma, \alpha)$ always is an even number in a derivable sequent (see Exercise 127). Let us look at a thin sequent $\Gamma[\Delta] \vdash \alpha$ and an interpolant θ of it with respect to Δ. Then $\sigma_c(\theta) \leq \sigma_c(\Delta) \leq 1$. For either $c \notin \pi(\Delta)$, and then $c \notin \pi(\theta)$, whence $\sigma_c(\theta) = 0$. Or $c \in \pi(\Delta)$; but then $c \in \pi(\Gamma, \alpha)$, and so by assumption $\sigma_c(\Delta) = 1$.

$$\sigma_c(\Delta, \theta) \leq \sigma_c(\Delta) + \sigma_c(\theta) \leq \sigma_c(\Gamma[\Delta], \alpha) + \sigma_c(\theta) \leq 2 + 1 \tag{3.252}$$

Now $\sigma_c(\Delta) + \sigma_c(\theta)$ is an even number hence either 0 or 2. Hence $\Delta \vdash \theta$ also is thin. Likewise it is shown that $\Gamma[\theta] \vdash \alpha$ is thin.

Lemma 3.86 *Let $\Gamma, \Theta, \Delta \vdash \alpha$ be a sequent and $c, d \in C$ two distinct elementary categories. Further, let $c \in \pi(\Gamma) \cap \pi(\Delta)$ as well as $d \in \pi(\Theta) \cap \pi(\alpha)$. Then $\Gamma, \Theta, \Delta \vdash \alpha$ is not thin.*

Proof. Let $\mathfrak{F}_G(C)$ be the free group generated by the elementary categories. The elements of this group are finite products of the form $c_0^{s_0} \cdot c_2^{s_2} \cdot \cdots \cdot c_{n-1}^{s_{n-1}}$, where $c_i \neq c_{i+1}$ for $i < n-1$ and $s_i \in \mathbb{Z} - \{0\}$. (If $n = 0$ then the empty product denotes the group unit, 1.) For if $c_0 = c_1$ the term $c_0^{s_0} \cdot c_1^{s_1}$ can be shortened to $c_0^{s_0+s_1}$. Look at the group valued interpretation γ sending every element of C to itself. If the sequent was thin we would have $\gamma(\Gamma) \cdot \gamma(\Theta) \cdot \gamma(\Delta) = \gamma(\alpha)$.

By hypothesis the left hand side is of the form $w \cdot c^{\pm 1} \cdot x \cdot d^{\pm 1} \cdot y \cdot c^{\pm 1} \cdot z$ for certain products w, x, y, z. The right hand side equals $t \cdot d^{\pm 1} \cdot u$ for certain t, u. Furthermore, we know that terms which stand for w, x, y, z as well as t and u cannot contain c or d if maximally reduced. But then equality cannot hold. □

Lemma 3.87 *Let $\alpha_0, \alpha_1, \ldots, \alpha_n \vdash \alpha_{n+1}$ be thin, $n > 0$. Then there is a k with $0 < k < n+1$ and $\pi(\alpha_k) \subseteq \pi(\alpha_{k-1}) \cup \pi(\alpha_{k+1})$.*

Proof. The proof is by induction on n. We start with $n = 1$. Here the sequent has the form $\alpha_0, \alpha_1 \vdash \alpha_2$. Let $c \in \pi(\alpha_1)$. Then $\sigma_c(\alpha_1) = 1$ since the sequent is thin. And since $\sigma_c(\alpha_0, \alpha_1, \alpha_2) = 2$, we have $\sigma_c(\alpha_0, \alpha_2) = 1$, whence $c \in \pi(\alpha_0) \cup \pi(\alpha_2)$. This finishes the case $n = 1$. Now let $n > 1$ and the claim proved for all $m < n$. **Case a.** $\pi(\alpha_0, \alpha_1, \ldots, \alpha_{n-2}) \cap \pi(\alpha_n) = \varnothing$. Then we choose $k := n$. For if $c \in \pi(\alpha_n)$ then $\sigma_c(\alpha_0, \ldots, \alpha_{n-2}) = 0$, and so we have $\sigma_c(\alpha_{n-1}) + \sigma_c(\alpha_{n+1}) = 1$. Hence we get $c \in \pi(\alpha_{n-1}) \cup \pi(\alpha_{n+1})$. **Case b.** $\pi(\alpha_0, \alpha_1, \ldots, \alpha_{n-2}) \cap \pi(\alpha_n) \neq \varnothing$. Then there exists an elementary category c with $c \in \pi(\alpha_0, \ldots, \alpha_{n-2})$ and $c \in \pi(\alpha_n)$. Put $\Gamma := \alpha_0, \alpha_1, \ldots, \alpha_{n-1}$, $\Delta := \alpha_n$. Let θ be an interpolant for $\Gamma, \Delta \vdash \alpha_{n+1}$ with respect to Γ. Then $\Gamma \vdash \theta$ and $\theta, \alpha_n \vdash \alpha_{n+1}$ are thin. By induction hypothesis there exists a k such that $\pi(\alpha_k) \subseteq \pi(\alpha_{k-1}) \cup \pi(\alpha_{k+1})$, if $k < n-1$, or $\pi(\alpha_k) \subseteq \pi(\alpha_{k-1}) \cup \pi(\theta)$ in case $k = n-1$. If $k < n-1$ then k is the desired number for the main sequent. Let now $k = n-1$. Then

$$\pi(\alpha_{n-1}) \subseteq \pi(\alpha_{n-2}) \cup \pi(\theta) \subseteq \pi(\alpha_{n-2}) \cup \pi(\alpha_n) \cup \pi(\alpha_{n+1}) \tag{3.253}$$

We show that k in this case too is the desired number for the main sequent. Let $\pi(\alpha_{n-1}) \cap \pi(\alpha_{n+1}) \neq \varnothing$, say $d \in \pi(\alpha_{n-1}) \cap \pi(\alpha_{n+1})$. Then surely $d \notin \pi(\alpha_n)$, so $d \neq c$. Therefore the sequent is not thin, by Lemma 3.86. Hence we have $\pi(\alpha_{n-1}) \cap \pi(\alpha_{n+1}) = \varnothing$, and so $\pi(\alpha_{n-1}) \subseteq \pi(\alpha_{n-2}) \cup \pi(\alpha_n)$. □

Lemma 3.88 *Let $\Gamma \vdash \gamma$ be an L–derivable thin sequent in which all categories have length $< m$. Then $\Gamma \vdash \gamma$ is already derivable in L_m.*

Proof. Let $\Gamma = \alpha_0, \alpha_1, \ldots, \alpha_{n-1}$; put $\alpha_n := \gamma$. If $n \leq 2$ then $\Gamma \vdash \gamma$ already is an axiom of L_m. So, let $n > 2$. By the previous lemma there is a k such that $\pi(\alpha_k) \subseteq \pi(\alpha_{k-1}) \cup \pi(\alpha_{k+1})$. **Case 1.** $k < n$. **Case 1a.** $|\pi(\alpha_{k-1}) \cap \pi(\alpha_k)| \geq |\pi(\alpha_{k+1}) \cap \pi(\alpha_k)|$. Put $\Xi := \alpha_0, \alpha_1, \ldots, \alpha_{k-2}$, $\Theta := \alpha_{k+1}, \ldots, \alpha_{n-1}$, and $\Delta := \alpha_{k-1}, \alpha_k$. Let θ be an interpolant for $\Xi, \Delta, \Theta \vdash \alpha_n$ with respect to Δ. Then the sequent

$$\alpha_0, \ldots, \alpha_{k-2}, \theta, \alpha_{k+1}, \ldots, \alpha_{n-1} \vdash \alpha_n \tag{3.254}$$

is thin. Furthermore

$$\begin{aligned}\pi(\theta) \subseteq&(\pi(\alpha_{k-1}) \cup \pi(\alpha_k)) \cap \pi(\Xi, \Theta, \alpha_n)\\ =&(\pi(\alpha_{k-1}) \cap \pi(\Xi, \Theta, \alpha_n)) \cup (\pi(\alpha_k) \cap \pi(\Xi, \Theta, \alpha_n)).\end{aligned} \tag{3.255}$$

Let $c \in \pi(\alpha_{k-1})$. Then $\sigma_c(\alpha_{k-1}) = 1$ and $\sigma_c(\Xi, \alpha_{k-1}, \alpha_k, \Theta, \alpha_n) = 2$, from which $\sigma_c(\Xi, \alpha_k, \Theta, \alpha_n) = 1$. Hence either $\sigma_c(\alpha_k) = 1$ or $\sigma_c(\Xi, \Theta, \alpha_n) = 1$. Since c was arbitrary we have

$$\pi(\alpha_k) \cap \pi(\Xi, \Theta, \alpha_n) = \pi(\alpha_{k-1}) - (\pi(\alpha_{k-1}) \cap \pi(\alpha_k)) \tag{3.256}$$

By choice of k, $\pi(\alpha_k) \cap \pi(\Xi, \Theta, \alpha_n) = \pi(\alpha_k) \cap \pi(\alpha_{k+1})$. Hence

$$\begin{aligned}\pi(\theta)=&(\pi(\alpha_{k-1}) \cap \pi(\Xi, \Theta, \alpha_n))(\pi(\alpha_k) \cap \pi(\Xi, \Theta, \alpha_n))\\ \subseteq&(\pi(\alpha_k) - (\pi(\alpha_{k-1}) \cap \pi(\alpha_k)) \cup (\pi(\alpha_k) \cap \pi(\alpha_{k+1})).\end{aligned} \tag{3.257}$$

So

$$\begin{aligned}|\pi(\theta)| &= |\pi(\alpha_{k-1})| + |\pi(\alpha_{k-1}) \cap \pi(\alpha_k)| + |\pi(\alpha_k) \cap \pi(\alpha_{k+1})|\\ &\leq |\pi(\alpha_{k-1})|\\ &< m\end{aligned} \tag{3.258}$$

(Note that $|\pi(\alpha_{k-1})| = |\alpha_k|$.) Therefore also $|\theta| < m$ and so $\alpha_{k-1}, \alpha_k \vdash \theta$ is an axiom of L_m. Hence, by induction hypothesis $\Xi, \theta, \Theta \vdash \alpha_n$ is derivable in L_m. A single application from both sequents yields the main sequent. It is therefore derivable in L_m. **Case 1b.** $|\pi(\alpha_{k-1}) \cap \pi(\alpha_k)| < |\pi(\alpha_k) \cap \pi(\alpha_{l+1})|$. Here one puts $\Xi := \alpha_0, \ldots, \alpha_{k-1}$, $\Delta := \alpha_k, \alpha_{k+1}$, $\Theta := \alpha_{k+1}, \ldots, \alpha_{n-1}$ and proceeds as in Case 1a. **Case 2.** $k = n-1$. So, $\pi(\alpha_{n-1}) \subseteq \pi(\alpha_{n-2}) \cup \pi(\gamma)$. Also here we distinguish to cases. **Case 2a.** $|\pi(\alpha_{n-2}) \cap \pi(\alpha_{n-1})| \geq |\pi(\alpha_{n-1}) \cap \pi(\alpha_n)|$. This case is similar to Case 1a. **Case 2b.** $|\pi(\alpha_{n-2}) \cap \pi(\alpha_{n-1})| < |\pi(\alpha_{n-1}) \cap \pi(\alpha_n)|$. Here put $\Delta := \alpha_0, \ldots, \alpha_{n-2}$, $\Theta := \alpha_{n-1}$. Let θ be an interpolant for $\Delta, \Theta \vdash \alpha_n$ with respect to Δ. Then $\Delta \vdash \theta$ as well as $\theta, \alpha_{n-1} \vdash \alpha_n$ are thin. Further we have

$$\begin{aligned}\pi(\theta) \subseteq&\pi(\Delta) \cap (\pi(\alpha_{n-1}) \cup \pi(\alpha_n))\\ =&(\pi(\Delta) \cap \pi(\alpha_{n-1})) \cup (\pi(\Delta) \cap \pi(\alpha_n))\\ =&(\pi(\alpha_{n-2}) \cap \pi(\alpha_{n-1})) \cup (\pi(\alpha_n) - (\pi(\alpha_{n-1}) \cap \pi(\alpha_n))).\end{aligned} \tag{3.259}$$

As in Case 1a we conclude that

$$\begin{aligned}|\pi(\theta)| &= |\pi(\alpha_{n-2}) \cap \pi(\alpha_{n-1})| + |\pi(\alpha_n)| - |\pi(\alpha_{n-1}) \cap \pi(\alpha_n)|\\ &< |\pi(\alpha_n)|\\ &< m\end{aligned} \tag{3.260}$$

Hence $\theta, \alpha_{n-1} \vdash \alpha_n$ is an axiom of L_m. By induction hypothesis, $\Delta \vdash \theta$ is derivable in L_m. A single application of (cut) yields the main sequent, which is therefore derivable in L_m. $\square$

Finally we proceed to the proof of Theorem 3.82. Let $|\gamma_i| < m$ for all $i < n$, and $|\alpha| < m$. Finally, let $\gamma_0, \gamma_1, \ldots, \gamma_{m-1} \vdash \alpha$ be derivable in L. We choose a derivation of this sequent. We may assume here that the axioms are only sequents of the form $c \vdash c$. For every occurrence of an axiom $c \vdash c$ we choose a new elementary category $\widehat{c}$ and replace this occurrence of $c \vdash c$ by $\widehat{c} \vdash \widehat{c}$. We extend this to the entire derivation and so we get a new derivation of a sequent $\widehat{\gamma}_0, \widehat{\gamma}_1, \ldots, \widehat{\gamma}_{n-1} \vdash \widehat{\alpha}$. We get $\sigma_c(\widehat{\alpha}) + \sum_{i<n} \sigma_c(\widehat{\gamma}_i) = 2$, if c occurs at all in the sequent. Nevertheless, the sequent need not be thin, since it may contain categories which are not thin. However, if $\sigma_c(\delta) = 2$ for some δ and some c, then c is not contained in any other category. We exploit this as follows. By successively applying interpolation we get the following sequents, which are all derivable in L.

$$
\begin{array}{rcl}
\widehat{\gamma}_0 \vdash \theta_0 & & \theta_0, \widehat{\gamma}_1, \widehat{\gamma}_2, \ldots, \widehat{\gamma}_{n-1} \vdash \widehat{\alpha} \\
\widehat{\gamma}_1 \vdash \theta_1 & & \theta_0, \theta_1, \widehat{\gamma}_2, \ldots, \widehat{\gamma}_{n-1} \vdash \widehat{\alpha} \\
\vdots & \vdots & \\
\widehat{\gamma}_{n-1} \vdash \theta_{n-1} & & \theta_0, \theta_1, \ldots, \theta_{n-1} \vdash \widehat{\alpha} \\
\theta_0, \theta_1, \ldots, \theta_{n-1} \vdash \gamma & & \gamma \vdash \widehat{\alpha}
\end{array}
\tag{3.261}
$$

It is not hard to show that $\sigma_c(\theta_i) \leq 1$ for all c and all $i < n$. So the sequent $\theta_0, \theta_1, \ldots, \theta_{n-1} \vdash \gamma$ is thin. Certainly $|\gamma| \leq |\widehat{\alpha}| = |\alpha| < m$ as well as $|\theta_i| \leq |\widehat{\alpha}_i| = |\alpha_i| < m$ for all $i < n$. By Lemma 3.88 the sequent $\theta_0, \theta_1, \ldots, \theta_{n-1} \vdash \gamma$ is derivable in L_m. The sequents $\widehat{\gamma}_i \vdash \theta_i$, $i < n$, as well as $\gamma \vdash \widehat{\alpha}_n$ are axioms of L_m. Hence $\widehat{\gamma}_0, \widehat{\gamma}_0, \ldots, \widehat{\gamma}_{n-1} \vdash \widehat{\alpha}$ is derivable in L_m. We undo the replacement in the derivation. This can in fact be done by applying a homomorphism (substitution) t which replaces $\widehat{c}$ by c. So, we get a derivation of $\gamma_0, \gamma_1, \ldots, \gamma_{n-1} \vdash \gamma_n$ in L_m. This concludes the proof of Theorem 3.82.

We remark that Pentus has also shown in [**?**] that L is complete with respect to so–called L–frames.

Definition 3.89 *An **L–frame** is a free semigroup of the form $\langle A^+, \cdot \rangle$. A **valuation** is a function $v : C \to \wp(A^+)$. v is extended to categories and sequents*

as follows:

$$
\begin{aligned}
v((\alpha \bullet \beta)) &:= v(\alpha) \cdot v(\beta) \\
v((\alpha / \beta)) &:= v(\alpha) // v(\beta) \\
v((\beta \backslash \alpha)) &:= v(\beta) \backslash\backslash v(\alpha) \\
v(\Gamma \circ \Delta) &:= v(\Gamma) \cdot v(\Delta)
\end{aligned}
\tag{3.262}
$$

$\Gamma \vdash \alpha$ *is* ***true under*** v *if* $v(\Gamma) \subseteq v(\alpha)$. *It is* ***valid in an L–frame*** *if it is true under all valuations.*

Theorem 3.90 (Pentus) $\overset{\mathsf{L}}{\leadsto} \Gamma \vdash \alpha$ *iff* $\Gamma \vdash \alpha$ *is valid in all L–frames.*

A survey of this subject area can be found in [**?**].

Exercise 126. Prove Theorem 3.79.

Exercise 127. Let $\Gamma \vdash \alpha$ be derivable in L, $c \in C$. Show that $\sigma_c(\Gamma) + \sigma_c(\alpha)$ is an even number.

Exercise 128. Prove Lemma 3.83.

Exercise 129. Show that if $\overset{\mathsf{L}}{\leadsto} \Gamma \vdash \alpha$, $\Gamma \vdash \alpha$ is valid in all L–frames.

8. Montague Semantics I

Until the beginning of the 1970s semantics of natural languages was considered a hopeless affair. Natural language was thought of as being completely illogical so that no formal theory of semantics for natural languages could ever be given. By contrast, Montague believed that natural languages can be analysed in the same way as formal languages. Even if this was too optimistic (and it is quite certain that Montague did deliberately overstate his case) there is enough evidence that natural languages are quite well–behaved. To prove his claim, Montague considered a small fragment of English, for whose semantics he produced a formal account. In this section we shall give a glimpse of the theory shaped by Montague. Before we can start, we have to talk about predicate logics and its models. For Montague has actually built his semantics somewhat differently than we have done so far. In place of defining the interpretation in a model directly, he defined a translation into λ–calculus over predicate logic, whose interpretation on the other hand is fixed by some general conventions.

A language of first–order predicate logic with identity has the following symbols:

① a set R of relation symbols, a disjoint set F of function symbols,

② a countably infinite set $V := \{\mathtt{x}_i : i \in \omega\}$ of variables,

③ the equality symbol $=$,

④ the booleans $\neg$, $\wedge$, $\vee$, $\rightarrow$,

⑤ the quantifiers $\forall$, $\exists$.

As outlined in Section 1.1, the language is defined by choosing a signature $\langle \Omega, \Xi \rangle$. Then r is a $\Xi(r)$–ary relation symbol and f a $\Omega(f)$–ary function symbol. Equality is always a binary relation symbol (so, $\Xi(=) = 2$). We define the set of terms as usual. Next we define formulae (see also Section 2.7).

① If t_i, $i < \Xi(r)$, are terms then $r(t_0, \ldots, t_{\Xi(r)-1})$ is a formula.

② If t_0 and t_1 are terms then $t_0 = t_1$ is a formula.

③ If φ and ψ are formulae, so are $(\neg\varphi)$, $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$ and $(\varphi \rightarrow \psi)$.

④ If φ is a formula and $x \in V$, then $(\forall x)\,\varphi$ and $(\exists x)\,\varphi$ are formulae.

A $\langle \Omega, \Xi \rangle$–**structure** is a triple $\langle M, \{f^{\mathfrak{M}} : f \in F\}, \{r^{\mathfrak{M}} : r \in R\} \rangle$ such that $f^{\mathfrak{M}} \colon M^{\Omega(f)} \to M$ for every $f \in F$ and $r^{\mathfrak{M}} \subseteq M^{\Xi(r)}$ for every $r \in R$. Now let $\beta \colon V \to M$. Then we define $\langle \mathfrak{M}, \beta \rangle \vDash \varphi$ for a formula by induction. To begin, we associate with every t its value $[t]^{\beta}$ under β.

$$
\begin{aligned}
[x]^{\beta} &:= \beta(x) \\
[f(t_0, \ldots, t_{\Omega(f)-1})]^{\beta} &:= f^{\mathfrak{M}}([t_0]^{\beta}, \ldots, [t_{\Omega(f)-1}]^{\beta})
\end{aligned}
\tag{3.263}
$$

Now we move on to formulae. (In this definition, $\gamma \sim_x \beta$, for $x \in V$, if $\beta(y) \neq$

$\gamma(y)$ only if $y = x$.)

$$
\begin{aligned}
\langle \mathfrak{M}, \beta \rangle &\vDash (s_0 \texttt{=} s_1) :\Leftrightarrow [s_0]^\beta = [s_1]^\beta \\
\langle \mathfrak{M}, \beta \rangle &\vDash r(\vec{s}) :\Leftrightarrow \langle [s_i] : i < \Xi(r) \rangle \in r^{\mathfrak{M}} \\
\langle \mathfrak{M}, \beta \rangle &\vDash (\neg\varphi) :\Leftrightarrow \langle \mathfrak{M}, \beta \rangle \nvDash \varphi \\
\langle \mathfrak{M}, \beta \rangle &\vDash (\varphi \wedge \psi) :\Leftrightarrow \langle \mathfrak{M}, \beta \rangle \vDash \varphi \text{ and } \langle \mathfrak{M}, \beta \rangle \vDash \psi \\
\langle \mathfrak{M}, \beta \rangle &\vDash (\varphi \vee \psi) :\Leftrightarrow \langle \mathfrak{M}, \beta \rangle \vDash \varphi \text{ or } \langle \mathfrak{M}, \beta \rangle \vDash \psi \\
\langle \mathfrak{M}, \beta \rangle &\vDash (\varphi \rightarrow \psi) :\Leftrightarrow \langle \mathfrak{M}, \beta \rangle \nvDash \varphi \text{ or } \langle \mathfrak{M}, \beta \rangle \vDash \psi \\
\langle \mathfrak{M}, \beta \rangle &\vDash (\exists x)\, \varphi :\Leftrightarrow \text{ there is } \beta' \sim_x \beta : \langle \mathfrak{M}, \beta' \rangle \vDash \varphi \\
\langle \mathfrak{M}, \beta \rangle &\vDash (\forall x)\, \varphi :\Leftrightarrow \text{ for all } \beta' \sim_x \beta : \langle \mathfrak{M}, \beta' \rangle \vDash \varphi
\end{aligned}
\tag{3.264}
$$

In this way formulae are interpreted in models.

Definition 3.91 *Let Δ be a set of formulae, and φ a formula. Then $\Delta \vDash \varphi$ if for all models $\langle \mathfrak{M}, \beta \rangle$: if $\langle \mathfrak{M}, \beta \rangle \vDash \delta$ for every $\delta \in \Delta$, then also $\langle \mathfrak{M}, \beta \rangle \vDash \varphi$.*

For example, the arithmetical terms in `+`, `0` and `*` with the relation `<` can be interpreted in the structure $\mathbb{N}$ where $\texttt{+}^{\mathbb{N}} = +$ and $\texttt{*}^{\mathbb{N}} = \cdot$ are the usual operations, $\texttt{0}^{\mathbb{N}} = 0$ and $\texttt{<}^{\mathbb{N}} = <$. Then for the valuation β with $\beta(\texttt{x}_2) = 7$ we have:

$$\langle \mathbb{N}, \beta \rangle \vDash (\forall \texttt{x}_0)(\forall \texttt{x}_1)((\texttt{x}_0 \texttt{*} \texttt{x}_1) \texttt{=} \texttt{x}_2 \rightarrow (\texttt{x}_0 \texttt{=} 1 \vee \texttt{x}_1 \texttt{=} 1)) \tag{3.265}$$

This formula says that $\beta(\texttt{x}_2)$ is a prime number. For a number w is a prime number iff for all numbers u and v: if $u \cdot v = w$ then $u = 1$ or $v = 1$. We compare this with (3.265). (3.265) holds if for all β' different only on $\texttt{x}_0$ from β

$$\langle \mathbb{N}, \beta' \rangle \vDash (\forall \texttt{x}_1)((\texttt{x}_0 \texttt{*} \texttt{x}_1) \texttt{=} \texttt{x}_2 \rightarrow (\texttt{x}_0 \texttt{=} 1 \vee \texttt{x}_1 \texttt{=} 1)) \tag{3.266}$$

This in turn is the case if for all β'' different only on $\texttt{x}_1$ from β'

$$\langle \mathbb{N}, \beta'' \rangle \vDash ((\texttt{x}_0 \texttt{*} \texttt{x}_1) \texttt{=} \texttt{x}_2 \rightarrow (\texttt{x}_0 \texttt{=} 1 \vee \texttt{x}_1 \texttt{=} 1)) \tag{3.267}$$

This means: if $u := \beta''(\texttt{x}_0)$, $v := \beta''(\texttt{x}_1)$ and $w := \beta''(\texttt{x}_2)$ and if we have $w = u \cdot v$, then $u = 1$ or $v = 1$. This holds for all u and v. Since on the other hand $w = \beta(\texttt{x}_2)$ we have (3.265) iff $\beta(\texttt{x}_2)$, that is to say 7, is a prime number.

The reader may convince himself that for every β

$$\begin{aligned}\langle \mathbb{N}, \beta \rangle \vDash (\forall \texttt{x}_0)(\exists \texttt{x}_1)(\forall \texttt{x}_2)(\forall \texttt{x}_3) \\ (\texttt{x}_0 \texttt{<} \texttt{x}_1 \wedge ((\texttt{x}_2 \texttt{*} \texttt{x}_3) \texttt{=} \texttt{x}_1 \rightarrow (\texttt{x}_2 \texttt{=} 1 \vee \texttt{x}_3 \texttt{=} 1)))\end{aligned} \tag{3.268}$$

This says that for every number there exists a prime number larger than it.

For later use we introduce a type e. This is the type of terms. e is realized by M. Before we can start designing a semantics for natural language we shall have to eliminate the relations from predicate logic. To this end we shall introduce a new basic type, t, which is the type of truth values. It is realized by the set $\{0,1\}$. An n–place relation r is now replaced by the characteristic function $r^{\spadesuit}$ from n–tuples of objects to truth values, which is defined as follows.

$$r^{\spadesuit}(x_0, x_1, \ldots, x_{\Xi(r)-1}) = 1 :\Leftrightarrow r(x_0, \ldots, x_{\Xi(r)-1}) \tag{3.269}$$

This allows us to use λ–calculus for handling the argument places of r. For example, from the binary relation r we can define the following functions r_1 and r_2.

$$r_1 := \lambda x_e.\lambda y_e.r^{\spadesuit}(x_e, y_e) \tag{3.270}$$

$$r_2 := \lambda x_e.\lambda y_e.r^{\spadesuit}(y_e, x_e) \tag{3.271}$$

So, we can define functions that either take the first argument of $r^{\spadesuit}$ first, or one which takes the first argument of $r^{\spadesuit}$ second.

Further, we shall also interpret $\neg$, $\wedge$, $\vee$ and $\rightarrow$ by the standard set–theoretic functions $-$, $\cap$, $\cup$ and $\supset$, respectively:

(3.272)

	$-$
0	1
1	0

$\cap$	0	1
0	0	0
1	0	1

$\cup$	0	1
0	0	1
1	1	1

$\supset$	0	1
0	1	1
1	0	1

Syntactically speaking $\neg$ has category t/t and $\wedge$, $\vee$ and $\rightarrow$ have category $(t\backslash t)/t$. Finally, also the quantifiers must be turned into functions. To this end we introduce the function symbols Π and Σ of type $((e \rightarrow t) \rightarrow t)$. Moreover, $\Pi(X)$ is true iff for all x $X(x)$ is true, and $\Sigma(X)$ is true iff for some x $X(x)$ is true. $(\forall x)\varphi$ is now replaced by $\Pi(\lambda x.\varphi)$, and $(\exists x)\varphi$ by $\Sigma(\lambda x.\varphi)$. So, ignoring types for the moment, we have the equations

$$\forall = \lambda \mathtt{x}_0.\lambda \mathtt{x}_1.\Pi(\lambda \mathtt{x}_0.\mathtt{x}_1) \tag{3.273}$$

$$\exists = \lambda \mathtt{x}_0.\lambda \mathtt{x}_1.\Sigma(\lambda \mathtt{x}_0.\mathtt{x}_1) \tag{3.274}$$

We shall however continue to write $\forall x.\varphi$ and $\exists x.\varphi$. This definition can in fact be used to define quantification for all functions. This is the core idea behind

the language of simple type theory (STT) according to Church [**?**]. Church assumes that the set of basic categories contains at least t. The symbol $\neg$ has the type $t \to t$, while the symbols $\wedge$, $\vee$ and $\rightarrow$ have type $t \to (t \to t)$. (Church actually works only with negation and conjunction as basic symbols, but this is just a matter of convenience.) To get the power of predicate logic we assume for each type α a symbol Π^α of type $(\alpha \to t) \to t$ and a symbol ι^α of type $\alpha \to (\alpha \to t)$. Put $\mathcal{S} := \mathrm{Typ}_{\to}(B)$.

Definition 3.92 *A **Henkin frame** is a structure*

(3.275) $\mathfrak{H} = \langle \{D_\alpha : \alpha \in \mathcal{S}\}, \bullet, -, \cap, \{\pi^\alpha : \alpha \in \mathcal{S}\}, \{\iota^\alpha : \alpha \in \mathcal{S}\}\rangle$

such that the following holds.

① $\langle \{D_\alpha : \alpha \in \mathcal{S}\}, \bullet\rangle$ *is a functionally complete typed applicative structure.*

② $D_t = \{0,1\}$, $-: D_t \to D_t$ *and* $\cap: D_t \to (D_t \to D_t)$ *are complement and intersection, respectively.*

③ *For every* $a \in D_{\alpha\to t}$ $\pi^\alpha \bullet a = 1$ *iff for every* $b \in D_\alpha$: $b \bullet a = 1$.

④ *For every* $a \in D_{\alpha\to t}$, *if there is a* $b \in D_\alpha$ *such that* $a \bullet b = 1$ *then also* $a \bullet (\iota^\alpha \bullet a) = 1$.

A valuation into a Henkin frame is a function β such that for every variable x of type α $\beta(x) \in D_\alpha$. For every N of type t, $\langle \mathfrak{H}, \beta\rangle \vDash N$ iff $[N]^\beta = 1$. Further, for a set Γ of expressions of type t and every N of type t, $\Gamma \vDash N$ if for every Henkin frame and every valuation β: if $\langle \mathfrak{H}, \beta\rangle \vDash M$ for all $M \in \Gamma$ then $\langle \mathfrak{H}, \beta\rangle \vDash N$.

π^α is the interpretation of Π^α and ι^α the interpretation of ι^α. So, π^α is the device discussed above that allows to define the universal quantifier for functions of type $\alpha \to t$. ι^α on the other hand is a kind of choice or 'witness' function. If a is a function from objects of type α into truth values then $\iota^\alpha \bullet a$ is an object of type α, and, moreover, if a is at all true on some b of type α, then it is true on $\iota^\alpha \bullet a$. In Section 4.4 we shall deliver an axiomatization of STT and show that the axiomatization is complete with respect to these models. The reason for explaining about STT is that every semantics or calculus that will be introduced in the sequel can easily be interpreted into STT.

We now turn to Montague Semantics. To begin we choose a very small base of words.

(3.276) $\{\texttt{Paul}, \texttt{Peter}, \texttt{sleeps}, \texttt{sees}\}$

The type of (the meaning of) `Paul` and `Peter` is e, the type of `sleeps` is $e \to t$, the type of `sees` $e \to (e \to t)$. This means: names are interpreted by individuals, intransitive verbs by unary relations, and transitive verbs by binary relations. The (finite) verb `sleeps` is interpreted by the relation sleeps' and `sees` by the relation see'. Because of our convention a transitive verb denotes a function (!) of type $e \to (e \to t)$. So the semantics of these verbs is

(3.277) $\quad \texttt{sleeps} \mapsto \lambda x_e.\mathsf{sleep}'(x_e)$

(3.278) $\quad \texttt{sees} \mapsto \lambda x_e.\lambda y_e.\mathsf{see}'(y_e, x_e)$

We already note here that the variables are unnecessary. After we have seen how the predicate logical formulae can be massaged into typed λ–expressions, we might as well forget this history and write sleep' in place of the function $\lambda x_e.\mathsf{sleep}'(x_e)$ and see' in place of $\lambda x_e.\lambda y_e.\mathsf{see}'(y_e, x_e)$. This has the additional advantage that we need not mention the variables at all (which is a moot point, as we have seen above). We continue in this section to use the somewhat more longwinded notation, however. We agree further that the value of `Paul` shall be the constant paul' and the value of `Peter` the constant peter'. Here are finally our 0–ary modes.

(3.279)
$$\begin{array}{lll} \langle \texttt{Paul}, & e, & \mathsf{paul}' \rangle \\ \langle \texttt{Peter}, & e, & \mathsf{peter}' \rangle \\ \langle \texttt{sleeps}, & e\backslash t, & \lambda x_e.\mathsf{sleep}'(x_e) \rangle \\ \langle \texttt{sees}, & (e\backslash t)/e, & \lambda x_e.\lambda y_e.\mathsf{see}'(y_e, x_e) \rangle \end{array}$$

The sentences `Peter sleeps` or `Peter sees Peter` are grammatical, and their meaning is $\mathsf{sleep}'(\mathsf{paul}')$ and $\mathsf{see}'(\mathsf{peter}', \mathsf{peter}')$.

The syntactic categories possess an equivalent in syntactic terminology. e for example is the category of proper names. The category $e\backslash t$ is the category of intransitive verbs and the category $(e\backslash t)/e$ is the category of transitive verbs.

This minilanguage can be extended. For example, we can introduce the word `not` by means of the following constant mode.

(3.280) $\quad \langle \texttt{not}, (e\backslash t)\backslash(e\backslash t), \lambda e_{e\to t}.\lambda x_e.\neg x_{e\to t}(x_e) \rangle$

The reader is asked to verify that now `sleeps not` is an intransitive verb, whose meaning is the complement of the meaning of `sleeps`. So, `Paul sleeps not` is true iff `Paul sleeps` is false. This is perhaps not such a

good example, since the negation in English is formed using the auxiliary `do`. To give a better example, we may introduce `and` by the following mode.

(3.281) $\langle \texttt{and}, ((e\backslash t)\backslash(e\backslash t))/(e\backslash t),$
$\lambda x_{e\to t}.\lambda y_{e\to t}.\lambda z_e.x_{e\to t}(z_e) \wedge y_{e\to t}(z_e)\rangle$

In this way we have a small language which can generate infinitely many grammatical sentences and which assigns them correct meanings. Of course, English is by far more complex than this.

The real advance that Montague made was to show that one can treat quantification. Let us take a look at how this can be done. (Actually, what we are going to outline right now is not Montague's own solution, since it is not in line with Categorial Grammar. We will deal with Montague's approach to quantification in Chapter 4.) Nouns like `cat` and `mouse` are not proper names but semantically speaking unary predicates. For `cat` does not denote a single individual but a class of individuals. Hence, following our conventions, the semantic type of `cat` and `mouse` is $e \to t$. Syntactically speaking this corresponds to either t/e or $e\backslash t$. Here, no decision is possible, for neither `Cat Paul` nor `Paul cat` is a grammatical sentence. Montague did not solve this problem; he introduced a new category constructor $//$, which allows to distinguish a category $t//e$ from t/e (the intransitive verb) even though they are not distinct in type. Our approach is simpler. We introduce a category n and stipulate that $\sigma(n) := e \to t$. This is an example where the basic categories are different from the (basic) semantic types. Now we say that the subject quantifier `every` has the sentactic category $(t/(e\backslash t))/n$. This means the following. It forms a constituent together with a noun, and that constituent has the category $t/(e\backslash t)$. This therefore is a constituent that needs an intransitive verb to form a sentence. So we have the following constant mode.

(3.282) $\langle \texttt{every}, (t/(e\backslash t))/n, \lambda x_{e\to t}.\lambda y_{e\to t}.\forall x_e.(x_{e\to t}(x_e) \to y_{e\to t}(x_e))\rangle$

Let us give an example.

(3.283) `every cat sees Peter`

The syntactic analysis is as follows.

(3.284)
$$\dfrac{\dfrac{\begin{array}{c}\texttt{every}\\ (t/(e\backslash t))/n\end{array} \quad \begin{array}{c}\texttt{cat}\\ n\end{array}}{t/(e\backslash t)} \qquad \dfrac{\begin{array}{c}\texttt{sees}\\ (e\backslash t)/e\end{array} \quad \begin{array}{c}\texttt{Peter}\\ e\end{array}}{e\backslash t}}{t}$$

This induces the following constituent structure.

(3.285) ((`every cat`) (`sees Peter`))

Now we shall have to insert the meanings in place of the words and calculate. This means converting into normal form. For by convention, a constituent has the meaning that arises from applying the meaning of one immediate part to the meaning of the other. That this is now well–defined is checked by the syntactic analysis. We calculate in several steps. `sees Peter` is a constituent and its meaning is

(3.286) $(\lambda x_e.\lambda y_e.\mathsf{see}'(y_e,x_e))(\mathsf{peter}') = \lambda y_e.\mathsf{see}'(y_e,\mathsf{peter}')$

Further, `every cat` is a constituent with the following meaning

(3.287)
$$\begin{aligned}&(\lambda x_{e\to t}.\lambda y_{e\to t}.(\forall x_e.x_{e\to t}(x_e) \to y_{e\to t}(x_e)))(\lambda x_e.\mathsf{cat}'(x_e))\\ =&\lambda y_{e\to t}.\forall x_e.((\lambda x_e.\mathsf{cat}'(x_e))(x_e) \to y_{e\to t}(x_e))\\ =&\lambda y_{e\to t}.\forall x_e.(\mathsf{cat}'(x_e) \to y_{e\to t}(x_e))\end{aligned}$$

Now we combine these two:

(3.288)
$$\begin{aligned}&(\lambda y_{e\to t}.\forall x_e.\mathsf{cat}'(x_e) \to y_{e\to t}(x_e))(\lambda y_e.\mathsf{see}'(y_e,\mathsf{peter}'))\\ =&\forall x_e.(\mathsf{cat}'(x_e) \to (\lambda y_e.\mathsf{see}'(y_e,\mathsf{peter}'))(x_e))\\ =&\forall x_e.(\mathsf{cat}'(x_e) \to \mathsf{see}'(x_e,\mathsf{peter}'))\end{aligned}$$

This is the desired result. Similarly to `every` we define `some`:

(3.289) $\langle \texttt{some}, (t/(e\backslash t))/n, \lambda x_{e\to t}.\lambda y_{e\to t}.\exists x_e.(x_{e\to t}(x_e) \wedge y_{e\to t}(x_e))\rangle$

If we also want to have quantifiers for direct objects we have to introduce new modes.

(3.290)
$$\begin{aligned}&\langle \texttt{every}, ((e\backslash t)/((e\backslash t)/e))/n,\\ &\qquad \lambda x_{e\to t}.\lambda y_{e\to(e\to t)}.\lambda y_e.\forall x_e.(x_{e\to t}(x_e) \to y_{e\to(e\to t)}(x_e)(y_e))\rangle\end{aligned}$$

(3.291)
$$\begin{aligned}&\langle \texttt{some}, ((e\backslash t)/((e\backslash t)/e))/n,\\ &\qquad \lambda x_{e\to t}.\lambda y_{e\to(e\to t)}.\lambda y_e.\exists x_e.(x_{e\to t}(x_e) \to y_{e\to(e\to t)}(x_e)(y_e))\rangle\end{aligned}$$

For `every cat` as a direct object is analyzed as a constituent which turns a transitive verb into an intransitive verb. Hence it must have the category

$(e\backslash t)/((e\backslash t)/e)$. From this follows immediately the category assignment for `every`.

Let us look at this using an example.

(3.292) `some cat sees every mouse`

The constituent structure is as follows.

(3.293) ((`some cat`) (`sees` (`every mouse`))))

The meaning of `every mouse` is, as is easily checked, the following:

(3.294) $\lambda y_{e\to(e\to t)}.\lambda y_e.\forall x_e(\mathsf{mouse}'(x_e) \to y_{e\to(e\to t)}(x_e)(y_e))$

From this we get for `sees every mouse`

(3.295)
$$\begin{aligned}&\lambda y_e.\forall x_e(\mathsf{mouse}'(x_e) \to (\lambda x_e.\lambda y_e.\mathsf{see}'(y_e,x_e))(x_e)(y_e))\\ =&\lambda y_e.\forall x_e(\mathsf{mouse}'(x_e) \to \mathsf{see}'(y_e,x_e))\end{aligned}$$

`some cat` is analogous to `every cat`:

(3.296) $\lambda y_{e\to t}.\exists x_e.(\mathsf{cat}'(x_e) \wedge y_{e\to t}(x_e))$

We combine (3.296) and (3.295).

(3.297)
$$\begin{aligned}&(\lambda y_{e\to t}.\exists x_e.(\mathsf{cat}'(x_e) \wedge y_{e\to t}(x_e)))\\ &\qquad(\lambda y_e.\forall x_e.(\mathsf{mouse}'(x_e) \to \mathsf{see}'(y_e,x_e)))\\ =&\exists x_e.(\mathsf{cat}'(x_e) \wedge (\lambda y_e.\forall x_e.(\mathsf{mouse}'(x_e) \to \mathsf{see}'(y_e,x_e)))(x_e))\\ =&\exists x_e.(\mathsf{cat}'(x_e) \wedge \forall z_e.(\mathsf{mouse}'(z_e) \to \mathsf{see}'(x_e,z_e)))\end{aligned}$$

One can see that the calculations require some caution. Sometimes variables may clash and this calls for the substitution of a variable. This is the case for example when we insert a term and by doing so create a bound occurrences of a variable. The λ–calculus is employed to do this work for us. (On the other hand, if we used plain functions here, this would again be needless.)

Montague used the cancellation interpretation for his calculus, hence the sequent formulation uses the calculus E. We have seen that this calculus can also be rendered into a sign grammar, which has two modes, forward application ($\mathtt{A}_>$) and backward application ($\mathtt{A}_<$). In syntactic theory, however, the most popular version of grammar is the Lambek–Calculus. However, the latter does not lend itself easily to a compositional interpretation. The fault lies

basically in the method of hypothetical assumptions. Let us see why this is so. An adjective like `big` has category n/n, and its type is $(e \to t) \to (e \to t)$. (This is not quite true, but good enough for illustration.) This means that it can modify nouns such as `car`, but not relational nouns such as `friend` or `neighbour`. Let us assume that the latter have category n/g (where g stands for the category of a genitive argument). Now, in Natural Deduction style Lambek–Calculus we can derive a constituent `big neighbour` by first feeding it a hypothetical argument and then abstracting over it.

(3.298)
$$\dfrac{\dfrac{\begin{array}{c}\texttt{big}\\ n/n : \mathsf{big}'\\ \vdots\end{array} \qquad \dfrac{\begin{array}{c}\texttt{neighbour}\\ n/g : \mathsf{neighbour}'\end{array} \qquad g : y}{n : \mathsf{neighbour}'(y)}}{n : \mathsf{big}'(\mathsf{neighbour}'(y))}}{n/g : \lambda y.\mathsf{big}'(\mathsf{neighbour}'(y))}$$

This allows us, for example, to coordinate `big neighbour` and `friend` and then compose with `of mine`. Notice that this proof is not available in E. There also is a sign based analogue of this. Introduce binary modes $\mathtt{L}_{>}$ and $\mathtt{L}_{<}$:

(3.299)
$$\begin{aligned}\mathtt{L}_{>}(\langle \vec{x}, \alpha, M\rangle, \langle \vec{y}, \gamma, x_\gamma\rangle) &:= \langle \vec{x}/\vec{y}, \alpha/\gamma, (\lambda x_\gamma.Mx_\gamma)\rangle\\ \mathtt{L}_{<}(\langle \vec{x}, \alpha, M\rangle, \langle \vec{y}, \gamma, x_\gamma\rangle) &:= \langle \vec{y}\backslash\vec{x}, \gamma\backslash\alpha, (\lambda x_\gamma.Mx_\gamma)\rangle\end{aligned}$$

A condition on the application of these modes is that the variable x_γ actually occurs free in the term. Now introduce a new 0–ary mode with exponent ⓒ, which shall be a symbol not in the alphabet.

(3.300) $\mathtt{V}_{\alpha:i} := \langle ⓒ, \alpha, x_{\alpha,i}\rangle$

Consider the structure term

(3.301) $\mathtt{L}_{>}\mathtt{A}_{>}\mathtt{Bg}\mathtt{A}_{>}\mathtt{Nb}\mathtt{V}_{\mathtt{g}:0}\mathtt{V}_{\mathtt{g}:0}$

Here, $\mathtt{Bg} := \langle \mathtt{big}, n/n, \mathsf{big}'\rangle$ and $\mathtt{Nb} := \mathtt{neighbour}, n/g, \mathsf{neighbour}'\rangle$. On condition that it is definite, it has the following unfolding.

(3.302) $\langle \mathtt{big\ neighbour}, n/g, \lambda x_{g,0}.\mathsf{big}'\mathsf{neighbour}'(x_{g,0})\rangle$

These modes play the role of hypothetical arguments in Natural Deduction style derivations. However, the combined effect of these modes is not exactly the same as in the Lambek–Calculus. The reason is that abstraction

can only be over a variable that is introduced right or left peripherally to the constituent. However, if we introduce two arguments in succession, we can abstract over them in any order we please, as the reader may check (see the exercises). The reason is that ⓒ bears no indication of the name of the variable that it introduces. This can be remedied by introducing instead the following 0–ary modes.

(3.303) $\mathtt{T}_{\alpha:i} := \langle ⓒ_{\alpha,i}, \alpha, x_{\alpha,i} \rangle$

Notice that these empty elements can be seen as the categorial analogon of traces in Transformational Grammar (see Section 6.5). Now the exponent reveals the exact identity of the variable and the Lambek–Calculus is exactly mirrorred by these modes. The price we pay is that there are structure terms whose exponents are not pronounceable: they contain elements that are strictly speaking not overtly visible. The strings are therefore not surface strings.

Notes on this section. Already in [**?**] the idea is defended that one must sometimes pass through 'nonexistent' strings, and TG has made much use of this. An alternative idea that suggests itself is to use combinators. This route has been taken by Steedman in [**?**; **?**]. For example, the addition of the modes $\mathtt{B}_{>}$ and $\mathtt{B}_{<}$ assures us that we can derive the these constituents as well. Steedman and Jacobson emphasize in their work also that variables can be dispensed with in favour of combinators. See [**?**; **?**] (and references therein) for a defense of variable free semantics. For a survey of approaches see [**?**].

Exercise 130. Write an AB–grammar for predicate logic over a given signature and a given structure. *Hint.* You need two types of basic categories: e and t, which now stand for *terms* and *truth–values*.

Exercise 131. The solutions we have presented here fall short of taking certain aspects of orthography into account. In particular, words are not separated by a blank, sentences do not end in a period and the first word of a sentence is written using lower case letters only. Can you think of a remedy for this situation?

Exercise 132. Show that with the help of $\mathtt{L}_{<}$ and $\mathtt{L}_{>}$ and the 0–ary modes $\mathtt{V}_{\alpha:i}$ it is possible to derive the sign

(3.304) $\langle \mathtt{give}, (e\backslash t)/e/e, \lambda x.\lambda y.\lambda z.\mathsf{give}'(z)(x)(y) \rangle$

from the sign

(3.305) $\langle \texttt{give}, (e\backslash t)/e/e, \lambda x.\lambda y.\lambda z.\mathsf{give}'(z)(y)(x)\rangle$

Exercise 133. We have noted earlier that `and`, `or` and `not` are polymorphic. The polymorphicity can be accommodated directly by defining polyadic operations in the λ–calculus. Here is how. Call a type α t–**final** if it has the following form: (a) $\alpha = t$, or (b) $\alpha = \beta \to \gamma$, where γ is t–final. Define $\wedge_\alpha$, $\vee_\alpha$ and $\neg_\alpha$ by induction. Similarly, for every type α define functions Σ_α and Π_α of type $\alpha \to t$ that interpret the existential and universal quantifier.

Exercise 134. A (**unary**) **generalized quantifier** is a function from properties to truth values (so, it is an object of type $(e \to t) \to t$). Examples are `some` and `every`, but there are many more:

(3.306) `more than three`

(3.307) `an even number of`

(3.308) `the director's`

First, give the semantics of each of the generalized quantifiers and define a sign for them. Now try to define the semantics of `more than`. (It takes a number and forms a generalized quantifier.)

Exercise 135. In CCG(B), many (but not all) substrings are constituents. We should therefore be able to coordinate them with `and`. As was noted for example by Eisenberg in [**?**], such a coordination is constrained (the brackets enclose the critical constituents).

(3.309) *[`John said that I`] `and` [`Mary said that she`]
`is the best swimmer.`

(3.310) [`John said that I`] `and` [`Mary said that she`]
`was the best swimmer.`

The constraint is as follows. $\vec{x} \diamond \texttt{and} \diamond \vec{y} \diamond \vec{z}$ is well–formed only if both $\vec{x} \diamond \vec{z}$ and $\vec{y} \diamond \vec{z}$ are. The suggestion is therefore that first the sentence $\vec{x} \diamond \vec{z} \diamond \texttt{and} \diamond \vec{y} \diamond \vec{z}$ is formed and then the first occurrence of $\vec{z}\diamond$ is 'deleted'. Can you suggest a different solution? *Note.* The construction is known as **forward deletion**. The more common **backward deletion** gives $\vec{x} \diamond \vec{z} \diamond \texttt{and} \diamond \vec{y}$, and is far less constrained.

Chapter 4
Semantics

1. The Nature of Semantical Representations

This chapter lays the foundation of semantics. In contrast to much of the current semantical theory we shall not use a model–theoretic approach but rather an algebraic one. As it turns out, the algebraic approach helps to circumvent many of the difficulties that beset a model–theoretic analysis, since it does not try to spell out the meanings in every detail, only in as much detail as is needed for the purpose at hand.

In this section we shall be concerned with the question of feasibility of interpretation. Much of semantical theory simply defines mappings from strings to meanings without assessing the question whether such mappings can actually be computed. While on a theoretical level this gives satisfying answers, one still has to address the question how it is possible that a human being can actually understand a sentence. The question is quite the same for computers. Mathematicians 'solve' the equation $x^2 = 2$ by writing $x = \pm\sqrt{2}$. However, this is just a piece of notation. If we want to know whether or not $3^{\sqrt{2}} < 6$, this requires calculation. This is the rule rather than the exception (think of trigonometric functions or the solutions of differential equations). However, hope is not lost. There are algorithms by which the number $\sqrt{2}$ can be approximated to any degree of precision needed, using only elementary operations. Much of mathematical theory has been inspired by the need to calculate difficult functions (for example logarithms) by means of elementary ones. Evidently, even though we do not have to bother any more with them thanks to computers, the computer still has to do the job for us. Computer hardware actually implements sophisticated algorithms for computing nonelementary functions. Furthermore, computers do not compute with arbitrary degree of precision. Numbers are stored in fixed size units (this is not necessary, but the size is limited anyhow by the size of the memory of the computer). Thus, they are only *close* to the actual input, not necessarily equal. Calculations on the numbers propagate these errors and in bad cases it can happen that small errors in the input yield astronomic errors in the output (problems that have this property independently of any algorithm that computes the solution are called **ill–conditioned**). Now, what reason do we have to say that a

particular machine with a particular algorithm computes, say, $\sqrt{2}$? One answer could be: that the program will yield *exactly* $\sqrt{2}$ given exact input and enough time. Yet, for approximative methods — the ones we generally have to use — the computation is never complete. However, then it computes a series of numbers a_n, $n \in \omega$, which converges to $\sqrt{2}$. That is to say, if $\varepsilon > 0$ is any real number (the error) we have to name an n_ε such that for all $n \geq n_\varepsilon$: $|a_n - \sqrt{2}| \leq \varepsilon$, given exact computation. That an algorithm computes such a series is typically shown using pure calculus over the real numbers. This computation is actually independent of the way in which the computation proceeds as long as it can be shown to compute the approximating series. For example, to compute $\sqrt{2}$ using Newton's method, all you have to do is to write a program that calculates

$$a_{n+1} := a_n - (a_n^2 - 2)/2a_n \tag{4.1}$$

For the actual computation on a machine it matters very much how this series is calculated. This is so because each operation induces an error, and the more we compute the more we depart from the correct value. Knowing the error propagation of the basic operations it is possible to compute exactly, given any algorithm, with what precision it computes. To sum up, in addition to calculus, computation on real machines needs two things:

- ☞ a theory of approximation, and
- ☞ a theory of error propagation.

Likewise, semantics is in need of two things: a theory of approximation, showing us what is possible to compute and what not, and how we can compute meanings, and second a theory of error propagation, showing us how we can determine the meanings in approximation given only limited resources for computation. We shall concern ourselves with the first of these. Moreover, we shall look only at a very limited aspect, namely: what meanings can in principle be computed and which ones cannot.

We have earlier characterized the computable functions as those that can be computed by a Turing machine. To see that this is by no means an innocent assumption, we shall look at propositional logic. Standardly, the semantics of classical propositional logic is given as follows. (This differs only slightly from the setup of Section 3.2.) The alphabet is $\{\mathtt{(},\mathtt{)},\mathtt{p},\mathtt{0},\mathtt{1},\neg,\wedge\}$ and the set of variables $V := \mathtt{p}(\mathtt{0} \cup \mathtt{1})^*$. A function $\beta : V \to 2$ is called a **valuation**. We

extend β to a mapping $\overline{\beta}$ from the entire language to 2.

$$
\begin{aligned}
\overline{\beta}(p) &:= \beta(p) && (p \in V)\\
\overline{\beta}(\texttt{(}\neg\varphi\texttt{)}) &:= -\overline{\beta}(\varphi) && (4.2)\\
\overline{\beta}(\texttt{(}\varphi\wedge\chi\texttt{)}) &:= \overline{\beta}(\varphi)\cap\overline{\beta}(\chi)
\end{aligned}
$$

To obtain from this a compositional interpretation for the language we turn matters around and define the meaning of a proposition to be a function from valuations to 2. Let 2^V be the set of functions from V to 2. Then for every proposition φ, $[\varphi]$ denotes the function from 2^V to 2 that satisfies

$$[\varphi](\beta) = \overline{\beta}(\varphi) \tag{4.3}$$

(The reader is made aware of the fact that what we have performed here is akin to type raising, turning the argument into a function over the function that applies to it.) Also $[\varphi]$ can be defined inductively.

$$
\begin{aligned}
[p] &:= \{\beta : \beta(p) = 1\} && (p \in V)\\
[\texttt{(}\neg\varphi\texttt{)}] &:= 2^V - [\varphi] && (4.4)\\
[\texttt{(}\varphi\wedge\chi\texttt{)}] &:= [\varphi]\cap[\chi]
\end{aligned}
$$

Now notice that V is infinite. However, we have excluded that the set of basic modes is infinite, and so we need to readjust the syntax. Rather than working with only one type of expression, we introduce a new type, that of a **register**. Registers are elements of $G := (\texttt{0}\cup\texttt{1})^*$. Then $V = \texttt{p}\cdot G$. Valuations are now functions from G to 2. The rest is as above. Here is now a sign grammar for propositional logic. The modes are $\texttt{E}$ (0–ary), $\texttt{P}_\texttt{0}$, $\texttt{P}_\texttt{1}$, $\texttt{V}$, $\texttt{J}_\neg$ (all unary), and $\texttt{J}_\wedge$ (binary). The exponents are strings over the alphabets, categories are either R or P, and meanings are either registers (for expressions of category R) or sets of functions from registers to 2 (for expressions of category P).

$$\texttt{E} := \langle \varepsilon, R, \varepsilon\rangle \tag{4.5a}$$

$$\texttt{P}_\texttt{0}(\langle \vec{x}, R, \vec{y}\rangle) := \langle \vec{x}^\frown\texttt{0}, R, \vec{y}^\frown\texttt{0}\rangle \tag{4.5b}$$

$$\texttt{P}_\texttt{1}(\langle \vec{x}, R, \vec{y}\rangle) := \langle \vec{x}^\frown\texttt{1}, R, \vec{y}^\frown\texttt{1}\rangle \tag{4.5c}$$

$$\texttt{V}(\langle \vec{x}, R, \vec{x}\rangle) := \langle \texttt{p}\vec{x}, P, [\texttt{p}\vec{x}]\rangle \tag{4.5d}$$

$$\texttt{J}_\neg(\langle \vec{x}, P, M\rangle) := \langle \texttt{(}\neg\vec{x}\texttt{)}, P, 2^V - M\rangle \tag{4.5e}$$

$$\texttt{J}_\wedge(\langle \vec{x}, P, M\rangle, \langle \vec{y}, P, N\rangle) := \langle \texttt{(}\vec{x}\wedge\vec{y}\texttt{)}, P, M\cap N\rangle \tag{4.5f}$$

It is easily checked that this is well–defined. This defines a sign grammar that meets all requirements for being compositional except for one: the functions on meanings are not computable. Notice that (a) valuations are infinite objects, and (b) there are uncountably many of them. However, this is not sufficient as an argument because we have not actually said how we encode sets of valuations as strings and how we compute with them. Notice also that the notion of computability is defined only on strings. Therefore, meanings too must be coded as strings. We may improve the situation a little bit by assuming that valuations are functions from finite subsets of G to 2. Then at least valuations can be represented as strings (for example, by listing pairs consisting of a register and its value). However, still the set of all valuations that make a given proposition true is infinite. On the other hand, there is an algorithm that can check for any given partial function whether it assigns 1 to a given register (it simply scans the string for the pair whose first member is the given register). Notice that if the function is not defined on the register, we must still give an output. Let it be #. We may then simply take the code of the Turing machine computing that function as the meaning the variable (see Section 1.7 for a definition). Then, inductively, we can define for every proposition φ a machine T_φ that computes the value of φ under any given partial valuation that gives a value for the occurring variables, and assigns # otherwise. Then we assign as the meaning of φ the code $T_\varphi^\spadesuit$ of that Turing machine. However, this approach suffers from a number of deficiencies.

First, the idea of using partial valuations does not always help. To see this let us now turn to predicate logic (see Section 3.8). As in the case of propositional logic we shall have to introduce binary strings for registers, to form variables. The meaning of a formula φ is by definition a function from pairs $\langle \mathfrak{M}, \beta \rangle$ to $\{0, 1\}$, where $\mathfrak{M}$ is a structure and β a function from variables to the domain of $\mathfrak{M}$. Again we have the problem to name finitary or at least computable procedures. We shall give two ways of doing so that yield quite different results. The first attempt is to exclude infinite models. Then $\mathfrak{M}$, and in particular the domain M of $\mathfrak{M}$, are finite. A valuation is a partial function from V to M with a finite domain. The meaning of a term under such a valuation is a member of M or $= \star$. (For if x_α is in t, and if β is not defined on x_α then t^β is undefined.) The meaning of a formula is either a truth value or $\star$. The truth values can be inductively defined as in Section 3.8. M has to be finite, since we usually cannot compute the value of $\forall x_\alpha . \varphi(x_\alpha)$ without knowing all values of x_α.

This definition has a severe drawback: it does not give the correct results.

For the logic of finite structures is stronger than the logic of all structures. For example, the following set of formulae is not satisfiable in finite structures while it has an infinite model. (Here `0` is a 0–ary function symbol, and `s` a unary function symbol.)

Proposition 4.1 *The theory T is consistent but has no finite model.*

$$T := \{(\forall x_0)(\neg s x_0 = 0), (\forall x_0)(\forall x_1)(s x_0 = s x_1 \to x_0 = x_1)\} \tag{4.6}$$

Proof. Let $\mathfrak{M}$ be a finite model for T. Then for some n and some $k > 0$: $s^{n+k}0 = s^n 0$. From this it follows with the second formula that $s^k 0 = 0$. Since $k > 0$, the first formula is false in $\mathfrak{M}$. There is, however, an infinite model for these formulae, namely the set of numbers together with 0 and the successor function. □

We remark here that the logic of finite structures is not recursively enumerable if we have two unary relation symbols. (This is a theorem from [**?**].) However, the logic of all structures is clearly recursively enumerable, showing that the sets are *very* different. This throws us into a dilemma: we can obviously not compute the meanings of formulae in a structure directly, since quantification requires search throughout the entire structure. (This problem has once worried some logicians, see [**?**]. Nowadays it is felt that these are not problems of logic proper.) So, once again we have to actually try out another *semantics*.

The first route is to let a formula denote the set of all formulae that are equivalent to it. Alternatively, we may take the set of all formulae that follow from it. (These are almost the same in boolean logic. For example, $\varphi \leftrightarrow \chi$ can be defined using $\to$ and $\wedge$; and $\varphi \to \chi$ can be defined by $\varphi \leftrightarrow (\varphi \wedge \chi)$. So these approaches are not very different. However the second one is technically speaking more elegant.) This set is again infinite. Hence, we do something different. We shall take a formula to denote any formula that follows from it. (Notice that this makes formulae have infinitely many meanings.) Before we start we seize the opportunity to introduce a more abstract theory. A **propositional language** is a language of formulas generated by a set V of variables and a signature. The identity of V is the same as for boolean logic above. As usual, propositions are considered here as certain strings. The language is denoted by the letter L. A **substitution** is given by a map $\sigma\colon V \to L$. σ defines a map from L to L by replacement of occurrences of variables by their σ–image. We denote by φ^σ the result of applying σ to φ.

Definition 4.2 *A **consequence relation over** L is a relation $\vdash \subseteq \wp(L) \times L$ such that the following holds. (We write $\Delta \vdash \varphi$ for the more complicated $\langle \Delta, \varphi \rangle \in \vdash$.)*

① $\varphi \vdash \varphi$.

② *If $\Delta \vdash \varphi$ and $\Delta \subseteq \Delta'$ then $\Delta' \vdash \varphi$.*

③ *If $\Delta \vdash \chi$ and $\Sigma; \chi \vdash \varphi$, then $\Delta; \Sigma \vdash \varphi$.*

*$\vdash$ is called **structural** if from $\Delta \vdash \varphi$ follows $\Delta^\sigma \vdash \varphi^\sigma$ for every substitution. $\vdash$ is **finitary** if $\Delta \vdash \varphi$ implies that there is a finite subset Δ' of Δ such that $\Delta' \vdash \varphi$.*

In the sequel consequence relations are always assumed to be structural. A rule is an element of $\wp(L) \times L$, that is, a pair $\rho = \langle \Delta, \varphi \rangle$. ρ is **finitary** if Δ is finite; it is n–**ary** if $|\Delta| = n$. Given a set R of rules, we call $\vdash^R$ the least structural consequence relation containing R. This relation can be explicitly defined. Say that χ is a **1–step** R–**consequence** of Σ if there is a substitution σ and some rule $\langle \Delta, \varphi \rangle \in R$ such that $\Delta^\sigma \subseteq \Sigma$ and $\chi = \varphi^\sigma$. Then, an n–**step consequence** of Σ is inductively defined.

Proposition 4.3 *$\Delta \vdash^R \varphi$ iff there is a natural number n such that φ is an n–step R–consequence of Δ.*

The reader may also try to generalize the notion of a proof from a Hilbert calculus and show that they define the same relation on condition that the rules are all finitary. We shall also give an abstract semantics and show its completeness. The notion of an Ω–algebra has been defined.

Definition 4.4 *Let L be a propositional logic over the signature Ω. A **matrix** for L and $\vdash$ is a pair $\mathfrak{M} = \langle \mathfrak{A}, D \rangle$, where $\mathfrak{A}$ is an Ω–algebra (the algebra of **truth values**) and D a subset of A, called the set of **designated truth values**. Let h be a homomorphism from $\mathfrak{Tm}_\Omega(V)$ into $\mathfrak{M}$. We write $\langle \mathfrak{M}, h \rangle \vDash \varphi$ if $h(\varphi) \in D$ and say that φ is **true under** h **in** $\mathfrak{M}$. Further, we write $\Delta \vDash_{\mathfrak{M}} \varphi$ if for all homomorphisms $h\colon \mathfrak{Tm}_\Omega(V) \to \mathfrak{A}$: if $h[\Delta] \subseteq D$ then $h(\varphi) \in D$.*

Proposition 4.5 *If $\mathfrak{M}$ is a matrix for L, $\vDash_{\mathfrak{M}}$ is a structural consequence relation.*

Notice that in boolean logic $\mathfrak{A}$ is the 2–element boolean algebra and $D = \{1\}$, but we shall encounter other cases later on. Here is a general method for obtaining matrices.

Definition 4.6 *Let L be a propositional language and $\vdash$ a consequence relation. Put $\Delta^{\vdash} := \{\varphi : \Delta \vdash \varphi\}$. Δ is **deductively closed** if $\Delta = \Delta^{\vdash}$. It is **consistent** if $\Delta^{\vdash} \neq L$. It is **maximally consistent** if it is consistent but no proper superset is.*

A matrix $\mathfrak{S}$ is **canonical** for $\vdash$ if $\mathfrak{S} = \langle \mathfrak{Tm}_{\Omega}(V), \Delta^{\vdash} \rangle$ for some set Δ. (Here, $\mathfrak{Tm}_{\Omega}(V)$ is the canonical algebra with carrier set L whose functions are just the associated string functions.) It is straightforward to verify that $\vdash \subseteq \vDash_{\mathfrak{S}}$. Now consider some set Δ and a formula such that $\Delta \nvdash \varphi$. Then put $\mathfrak{S} := \langle \mathfrak{Tm}_{\Omega}(V), \Delta^{\vdash} \rangle$ and let h be the identity. Then $h[\Delta] = \Delta \subseteq \Delta^{\vdash}$, but $h(\varphi) \notin \Delta^{\vdash}$ by definition of $\Delta^{\vdash}$. So, $\Delta \nvDash_{\mathfrak{S}} \varphi$. This shows the following.

Theorem 4.7 (Completeness of Matrix Semantics) *Let $\vdash$ be a structural consequence relation over L. Then*

$$\vdash = \bigcap \langle \vDash_{\mathfrak{S}} : \mathfrak{S} \textit{ canonical for } \vdash \rangle \tag{4.7}$$

(The reader may verify that an arbitrary intersection of consequence relations again is a consequence relation.) This theorem establishes that for any consequence relation we can find enough matrices such that they together characterize that relation. We shall notice also the following. Given $\vdash$ and $\mathfrak{M} = \langle \mathfrak{A}, D \rangle$, then $\vDash_{\mathfrak{M}} \supseteq \vdash$ iff D is closed under the consequence. (This is pretty trivial: all it says is that if $\Delta \vdash \varphi$ and h is a homomorphism, then if $h[\Delta] \subseteq D$ we must have $h(\varphi) \in D$.) Such sets are called **filters**. Now, let $\mathfrak{M} = \langle \mathfrak{A}, D \rangle$ be a matrix, and Θ a congruence on $\mathfrak{A}$. Suppose that for any x: $[x]\Theta \subseteq D$ or $[x]\Theta \cap D = \varnothing$. Then we call Θ **admissible for** $\mathfrak{M}$ and put $\mathfrak{M}/\Theta := \langle \mathfrak{A}/\Theta, D/\Theta \rangle$, where $D/\Theta := \{[x]\Theta : x \in D\}$. The following is easy to show.

Proposition 4.8 *Let $\mathfrak{M}$ be a matrix and Θ an admissible congruence on $\mathfrak{M}$. Then $\vDash_{\mathfrak{M}/\Theta} = \vDash_{\mathfrak{M}}$.*

Finally, call a matrix **reduced** if only the diagonal is an admissible congruence. Then, by Proposition 4.8 and Theorem 4.7 we immediately derive that every consequence relation is complete with respect to reduced matrices. One also calls a class of matrices $\mathcal{K}$ a (**matrix**) **semantics** and says $\mathcal{K}$ is **adequate for** a consequence relation $\vdash$ if $\vdash = \bigcap_{\mathfrak{M} \in \mathcal{K}} \vDash_{\mathfrak{M}}$.

Now, given L and $\vdash$, the system of signs for the consequence relation is this.

$$\Sigma_P := \{\langle \vec{x}, R, \vec{x} \rangle : \vec{x} \in G\} \cup \{\langle \vec{x}, P, \vec{y} \rangle : \vec{x} \vdash \vec{y}\} \tag{4.8}$$

How does this change the situation? Notice that we can axiomatize the consequences by means of rules. The following is a set of rules that fully axiomatizes the consequence. The proof of that will be left to the reader (see the exercises), since it is only peripheral to our interests.

(4.9a) $\rho_d := \langle\{\texttt{(}\neg\texttt{(}\neg\texttt{p))}\},\texttt{p}\rangle$

(4.9b) $\rho_{dn} := \langle\{\texttt{p}\},\texttt{(}\neg\texttt{(}\neg\texttt{p))}\rangle$

(4.9c) $\rho_u := \langle\{\texttt{p},\texttt{(}\neg\texttt{p)}\},\texttt{p0}\rangle$

(4.9d) $\rho_c := \langle\{\texttt{p},\texttt{p0}\},\texttt{(p}\wedge\texttt{p0)}\rangle$

(4.9e) $\rho_{p0} := \langle\{\texttt{(p}\wedge\texttt{p0)}\},\texttt{p}\rangle$

(4.9f) $\rho_{p1} := \langle\{\texttt{(p}\wedge\texttt{p0)}\},\texttt{p0}\rangle$

(4.9g) $\rho_{mp} := \langle\{\texttt{p},\texttt{(}\neg\texttt{(p}\wedge\texttt{(}\neg\texttt{p0)))}\},\texttt{p0}\rangle$

With each rule we can actually associate a mode. We only give examples, since the general scheme for defining modes is easily extractable.

(4.10) $\texttt{F}_{\texttt{dn}}(\langle\vec{x},P,\texttt{(}\neg\texttt{(}\neg\vec{y}\texttt{))}\rangle := \langle\vec{x},P,\vec{y}\rangle$

(4.11) $\texttt{F}_{\texttt{c}}(\langle\vec{x},P,\vec{y}\rangle,\langle\vec{x},P,\vec{z}\rangle) := \langle\vec{x},P,\texttt{(}\vec{y}\wedge\vec{z}\texttt{)}\rangle$

If we have $\rightarrow$ as a primitive symbol then the following mode corresponds to the rule ρ_{mp}, Modus Ponens.

(4.12) $\texttt{F}_{\texttt{mp}}(\langle\vec{x},P,\texttt{(}\vec{y}\rightarrow\vec{z}\texttt{)}\rangle,\langle\vec{x},P,\vec{y}\rangle) := \langle\vec{x},P,\vec{z}\rangle$

This is satisfactory in that it allows to derive all and only the consequences of a given proposition. A drawback is that the functions on the exponents are nonincreasing. They always return $\vec{x}$. The structure term of the sign $\langle\vec{x},P,\vec{y}\rangle$ on the other hand encodes a derivation of $\vec{y}$ from $\vec{x}$.

Now, the reader may get worried by the proliferation of different semantics. Aren't we always solving a different problem? Our answer is indirect. The problem is that we do not know exactly what meanings are. Given a natural language, what we can observe more or less directly is the exponents. Although it is not easy to write down rules that generate them, the entities are more or less concrete. A little less concrete are the syntactic categories. We have already seen in the previous chapter that the assignment of categories to strings (or other exponents, see next chapter) are also somewhat arbitrary. We shall return to this issue. Even less clearly definable, however, are the meanings. What, for example, is the meaning of (4.13)?

(4.13) `Caesar crossed the Rubicon.`

The first answer we have given was: a truth value. For this sentence is either true or false. But even though it is true, it might have been false, just in case Caesar did not cross the Rubicon. What makes us know this? The second answer (for first–order theories) is: the meaning is a set of models. Knowing what the model is and what the variables are assigned to, we know whether that sentence is true. But we simply cannot look at all models, and still it seems that we know what (4.13) means. Therefore the next answer is: its meaning is an algorithm, which, given a model, tells us whether the sentence is true. Then, finally, we do not have to know everything in order to know whether (4.13) is true. Most facts are irrelevant, for example, whether Napoleon was French. On the other hand, suppose we witness Caesar walk across the Rubicon, or suppose we know for sure that first he was north of the Rubicon and the next day to the south of it. This will make us believe that (4.13) is true. Thus, the algorithm that computes the truth value does not need all of a model; a small part of it actually suffices. We can introduce partial models and define algorithms on them, but all this is a variation on the same theme. A different approach is provided by our last answer: a sentence means whatever it implies.

We may cast this as follows. Start with the set L of propositions and a set (or class) $\mathcal{M}$ of models. A **primary** (or **model theoretic**) **semantics** is given in terms of a relation $\vDash \subseteq L \times \mathcal{M}$. Most approaches are variants of the primary semantics, since they more or less characterize meanings in terms of facts. However, from this semantics we may define a **secondary semantics**, which is the semantics of consequence. $\Delta \vDash \varphi$ iff for all $M \in \mathcal{M}$: if $M \vDash \delta$ for all $\delta \in \Delta$ then $M \vDash \varphi$. (We say in this case that Δ entails φ.) Secondary semantics is concerned only with the relationship between the objects of the language, there is no model involved. It is clear that the secondary semantics is not fully adequate. Notice namely that knowing the logical relationship between sentences does not reveal anything about the nature of the models. Second, even if we knew what the models were: we could not say whether a given sentence is true in a given model or not. It is perfectly conceivable that we know English to the extent that we know which sentences entail which other sentences, but still we are unable to say, for example, whether or not (4.13) is true even when we witnessed Caesar cross the Rubicon. An example might make this clear. Imagine that all I know is which sentences of English imply which other sentences, but that I know nothing more about their actual meaning. Suppose now that the house is on fire. If I realize this I know that I am in danger and I act accordingly. However, suppose that someone shouts

(4.14) at me. Then I can infer that he thinks (4.14) is true. This will certainly make me believe that (4.14) is true and even that (4.15) is true as well. But still I do not know that the house is on fire, nor that I am in danger.

(4.14) `The house is on fire.`

(4.15) `I am in danger.`

Therefore, knowing how sentences hang together in a deductive system has little to do with the actual world. The situation is not simply remedied by knowing some of the meanings. Suppose I additionally know that (4.14) means that the house is on fire. Then if I see that the house is on fire then I know that I am in danger, and I also know that (4.15) is the case. But I still may fail to see that (4.15) means that I am in danger. It may just mean something else that is being implied by (4.14). This is reminiscent of Searle's thesis that language is about the world: knowing what things mean is not constituted by an ability to manipulate certain symbols. We may phrase this as follows.

> *Indeterminacy of secondary semantics.* No secondary semantics can fix the truth conditions of propositions uniquely for any given language.

Searle's claims go further than that, but this much is perhaps quite uncontroversial. Despite the fact that secondary semantics is underdetermined, we shall not deal with primary semantics at all. We are not going to discuss what a word, say, `life` *really* means — we are only interested in how its meaning functions language internally. Formal semantics really cannot do more than that. In what is to follow we shall sketch an algebraic approach to semantics. This contrasts with the far more widespread model–theoretic approach. The latter may be more explicit and intuitive, but on the other hand it is quite inflexible.

We begin by examining a very influential principle in semantics, called Leibniz' Principle. We quote one of its original formulation from [**?**] (from *Specimen Calculi Coincidentium*, (1), 1690). *Eadem vel Coincidentia sunt quae sibi ubique substitui possunt salva veritate. Diversa quae non possunt.* Translated it says: *The same or coincident are those which can everywhere be substituted for each other not affecting truth. Different are those that cannot.* Clearly, substitution must be understood here in the context of sentences, and we must assume that what we substitute is constituent occurrences of the expressions. We therefore reformulate the principle as follows.

> *Leibniz' Principle.* Two expressions A and B have the same meaning iff in every sentence any occurrence of A can be substituted by B and any occurrence of B by A without changing the truth of that sentence.

To some people this principle seems to assume bivalence. If there are more than two truth values we might interpret Leibniz' original definition as saying that substitution does not change the truth value rather than just truth. (See also Lyons for a discussion.) We shall not do that, however. First we give some unproblematic examples. In second order logic (SO, see Chapter 1), the following is a theorem.

$$(4.16) \qquad (\forall x)(\forall y)(x = y \leftrightarrow (\forall P)(P(x) \leftrightarrow P(y)))$$

Hence, Leibniz' Principle holds of second order logic with respect to terms. There is general no identity relation for predicates, but if there is, it is defined according to Leibniz' Principle: two predicates are equal iff they hold of the same individuals. This requires full second order logic, for what we want to have is the following for each $n \in \omega$ (with P_n and Q_n variables for n–ary relations):

$$(4.17) \qquad (\forall P_n)(\forall Q_n)(P_n = Q_n \leftrightarrow (\forall \vec{x})(P_n(\vec{x}) \leftrightarrow Q_n(\vec{x})))$$

(Here, $\vec{x}$ abbreviates the n–tuple $x_0, \ldots, x_{n-1}$.) (4.16) is actually the basis for Montague's type raising. Recall that Montague identified an individual with the set of all of its properties. In virtue of (4.16) this identification does not conflate distinct individuals. To turn that around: by Leibniz' Principle, this identification is one–to–one. We shall see in the next section that boolean algebras of any kind can be embedded into powerset algebras. The background of this proof is the result that if there are two elements x, y in a boolean algebra $\mathfrak{B}$ and for all homomorphisms $h\colon \mathfrak{B} \to \mathbf{2}$ we have $h(x) = h(y)$, then $x = y$. (More on that in the next section. We have to use homomorphisms here since properties are functions that commute with the boolean operations, that is to say, homomorphisms.) Thus, Leibniz' Principle also holds for boolean semantics, defined in Section 4.6. Notice that the proof relies on the Axiom of Choice (in fact the somewhat weaker Prime Ideal Axiom), so it is not altogether innocent.

We use Leibniz' Principle to detect whether two items have the same meaning. One consequence of this principle is that semantics is essentially unique. If $\mu\colon A^* \overset{p}{\twoheadrightarrow} M$, $\mu'\colon A^* \overset{p}{\twoheadrightarrow} M'$ are surjective functions assigning meanings to expressions, and if both satisfy Leibniz' Principle, then there is a

bijection $\pi\colon M \to M'$ such that $\mu' = \pi \circ \mu$ and $\mu = \pi^{-1} \circ \mu'$. Thus, as far as formal semantics is concerned, any solution is as good any other.

As we have briefly mentioned in Section 3.5, we may use the same idea to define types. This method goes back to Husserl, and is a key ingredient to the theory of compositionality by Wilfrid Hodges (see his [**?**]). A type is a class of expressions that can be substituted for each other without changing meaningfulness. Hodges just uses pairs of exponents and meanings. If we want to assimilate his setup to ours, we may add a category U, and let for every mode f, $f^{\tau}(U,\ldots,U) := U$. However, the idea is to do without categories. If we further substract the meanings, we get what Hodges calls a *grammar*. We prefer to call it an **H–grammar**. (The letter **H** honours Hodges here.) Thus, an H–grammar is defined by some signature and corresponding operations on the set E of exponents, which may even be partial. An **H–semantics** is a partial map μ from the structure terms (!) to a set M of meanings. Structure terms $\mathfrak{s}$ and $\mathfrak{t}$ are **synonymous** if μ is defined on both and $\mu(\mathfrak{s}) = \mu(\mathfrak{t})$. We write $\mathfrak{s} \equiv_\mu \mathfrak{t}$ to say that $\mathfrak{s}$ and $\mathfrak{t}$ are synonymous. (Notice that $\mathfrak{s} \equiv_\mu \mathfrak{s}$ iff μ is defined on $\mathfrak{s}$.) An H–semantics ν is **equivalent** to μ if $\equiv_\mu = \equiv_\nu$. An **H–synonymy** is an equivalence relation on a subset of the set of structure terms. We call that subset the **field** of the H–synonymy. Given an H–synonymy $\equiv$, we may define M to be the set of all equivalence classes of $\equiv$, and set $\mu^{\equiv}(\mathfrak{s}) := [\mathfrak{s}]_\equiv$ iff $\mathfrak{s}$ is in that subset, and undefined otherwise. Thus, up to equivalence, H–synonymies and H–semantics are in one–to–one correspondence. We say that $\equiv'$ **extends** $\equiv$ if the field of $\equiv'$ contains the field of $\equiv$, and the two coincide on the field of $\equiv$.

Definition 4.9 *Let G be an H–grammar and μ an H–semantics for it. We write $\mathfrak{s} \sim_\mu \mathfrak{s}'$ iff for every structure term $\mathfrak{t}$ with a single free variable x, $[\mathfrak{s}/x]\mathfrak{t}$ is μ–meaningful iff $[\mathfrak{s}'/x]\mathfrak{t}$ is μ–meaningful. The equivalence classes of $\sim_\mu$ are called the μ–**categories**.*

This is the formal rendering of the ‘meaning categories’ that Husserl defines.

Definition 4.10 *ν and its associated synonymy is called μ–**Husserlian** if for all structure terms $\mathfrak{s}$ and $\mathfrak{s}'$: if $\mathfrak{s} \equiv_\nu \mathfrak{s}'$ then $\mathfrak{s} \sim_\mu \mathfrak{s}'$. μ is called **Husserlian** if it is μ–Husserlian.*

It is worthwhile to compare this definition with Leibniz’ Principle. The latter defines identity in meaning via intersubstitutability in all sentences; what must remain constant is truth. Husserl’s meaning categories are also defined

by intersubstitutability in all sentences; however, what must remain constant is the meaningfulness. We may connect these principles as follows.

Definition 4.11 *Let* Sent *be a set of structure terms and* $\Delta \subseteq \mathrm{Sent}$. *We call* $\mathfrak{s}$ ***sentential*** *if* $\mathfrak{s} \in \mathrm{Sent}$, *and* ***true*** *if* $\mathfrak{s} \in \Delta$. μ *is* ***Leibnizian*** *if for all structure terms* $\mathfrak{u}$ *and* $\mathfrak{u}'$: $\mathfrak{u} \equiv_\mu \mathfrak{u}'$ *iff for all structure terms* $\mathfrak{s}$ *such that* $[\mathfrak{u}/x]\mathfrak{s} \in \Delta$ *also* $[\mathfrak{u}'/x]\mathfrak{s} \in \Delta$ *and conversely.*

Under mild assumptions on μ it holds that Leibnizian implies Husserlian. The following is from [**?**].

Theorem 4.12 (Hodges) *Let* μ *be an H–semantics for the H–grammar G. Suppose further that every subterm of a* μ*–meaningful structure term is again* μ*–meaningful. Then the following are equivalent.*

① *For each mode* f *there is an* $\Omega(f)$*–ary function* $f^\mu : M^{\Omega(f)} \to M$ *such that* μ *is a homomorphism of partial algebras.*

② *If* $\mathfrak{s}$ *is a structure term and* $\mathfrak{u}_i$, $\mathfrak{v}_i$ $(i < n)$ *are structure terms such that* $[\mathfrak{u}_i/x_i : i < n]\mathfrak{s}$ *and* $[\mathfrak{v}_i/x_i : i < n]\mathfrak{s}$ *are both* μ*–meaningful and if for all* $i < n$ $\mathfrak{u}_i \equiv_\mu \mathfrak{v}_i$ *then*

$$[\mathfrak{u}_i/x_i : i < n]\mathfrak{s} \equiv_\mu [\mathfrak{v}_i/x_i : i < n]\mathfrak{s} \ .$$

Furthermore, if μ *is Husserlian then the second already holds if it holds for* $n = 1$.

It is illuminating to recast the approach by Hodges in algebraic terms. This allows to compare it with the setup of Section 3.1. Moreover, it will also give a proof of Theorem 4.12. We start with a signature Ω. The set $\mathrm{Tm}_\Omega(X)$ forms an algebra which we have denoted by $\mathfrak{Tm}_\Omega(X)$. Now select a subset $D \subseteq \mathrm{Tm}_\Omega(X)$ of *meaningful terms*. It turns out that the embedding $i\colon D \rightarrowtail \mathrm{Tm}_\Omega(X)\colon x \mapsto x$ is a strong homomorphism iff D is closed under subterms. We denote the induced algebra by $\mathfrak{D}$. It is a partial algebra. The map $\mu : D \to M$ induces an equivalence relation $\equiv_\mu$. There are functions $f^\mu : M^{\Omega(f)} \to M$ that make M into an algebra $\mathfrak{M}$ and μ into a homomorphism iff $\equiv_\mu$ is a weak congruence relation (see Definition 1.21 and the remark following it). This is the first claim of Theorem 4.12. For the second claim we need to investigate the structure of partial algebras.

Definition 4.13 *Let* $\mathfrak{A}$ *be a partial* Ω*–algebra. Put* $x \asymp_\mathfrak{A} y$ *(or simply* $x \asymp y$*) if for all* $f \in \mathrm{Pol}_1(\mathfrak{A})$*:* $f(x)$ *is defined iff* $f(y)$ *is defined.*

Proposition 4.14 *Let $\mathfrak{A}$ be a partial Ω–algebra. (a) $\asymp_{\mathfrak{A}}$ is a strong congruence relation on $\mathfrak{A}$. (b) A weak congruence on $\mathfrak{A}$ is strong iff it is contained in $\asymp_{\mathfrak{A}}$.*

Proof. (a) Clearly, $\asymp$ is an equivalence relation. So, let $f \in F$ and $a_i \asymp c_i$ for all $i < \Omega(f)$. We have to show that $f(\vec{a}) \asymp f(\vec{c})$, that is, for all $g \in \mathrm{Pol}_1(\mathfrak{A})$: $g(f(\vec{a}))$ is defined iff $g(f(\vec{c}))$ is. Assume that $g(f(\vec{a}))$ is defined. The function $g(f(x_0, a_1, \ldots, a_{\Omega(f)-1}))$ is a unary polynomial h_0, and $h_0(a_0)$ is defined. By definition of $\asymp$, $h_0(c_0) = g(f(c_0, a_1, \ldots, a_{\Omega(f)-1}))$ is also defined. Next,

$$h_1(x_1) := f(g(c_0, x_1, a_2, \ldots, a_{\Omega(f)-1})) \tag{4.18}$$

is a unary polynomial and defined on a_1. So, it is defined on c_1 and we have $h_1(c_1) = f(g(c_0, c_1, a_2, \ldots, a_{\Omega(f)-1}))$. In this way we show that $f(g(\vec{c}))$ is defined. (b) Let Θ be a weak congruence. Suppose that it is not strong. Then there is a polynomial f and vectors $\vec{a}, \vec{c} \in A^{\Omega(f)}$ with $a_i \,\Theta\, c_i$ ($i < \Omega(f)$) such that $f(\vec{a})$ is defined but $f(\vec{c})$ is not. Now, for all $i < \Omega(f)$,

$$\begin{aligned} &f(a_0, \ldots, a_{i-1}, a_i, c_{i+1}, \ldots, c_{\Omega(f)-1}) \\ &\qquad\qquad \Theta\, f(a_0, \ldots, a_{i-1}, c_i, c_{i+1}, \ldots, c_{\Omega(f)-1}) \end{aligned} \tag{4.19}$$

if both sides are defined. Now, $f(\vec{a})$ is not Θ–congruent to $f(\vec{c})$. Hence there is an $i < \Omega(f)$ such that the left hand side of (4.19) is defined and the right hand side is not. Put

$$h(x) := f(a_0, \ldots, a_{i-1}, x, c_{i+1}, \ldots, c_{\Omega(f)-1}) \tag{4.20}$$

Then $h(a_i)$ is defined, $h(c_i)$ is not, but $a_i \,\Theta\, c_i$. So, $\Theta \not\subseteq \asymp$. Conversely, if Θ is strong we can use (4.19) to show inductively that if $f(\vec{a})$ is defined, so are all members of the chain. Hence $f(\vec{c})$ is defined. And conversely. $\square$

Proposition 4.15 *Let $\mathfrak{A}$ be a partial algebra and Θ an equivalence relation on $\mathfrak{A}$. Θ is a strong congruence iff for all $g \in \mathrm{Pol}_1(\mathfrak{A})$ and all $a, c \in A$ such that $a \,\Theta\, c$: $g(a)$ is defined iff $g(c)$ is, and then $g(a) \,\Theta\, g(c)$.*

The proof of this claim is similar. To connect this with the theory by Hodges, notice that $\sim_\mu$ is the same as $\asymp_{\mathfrak{D}}$. $\equiv_\mu$ is Husserlian iff $\equiv_\mu \subseteq \asymp_{\mathfrak{D}}$.

Proposition 4.16 *$\equiv_\mu$ is Husserlian iff it is contained in $\asymp_{\mathfrak{D}}$ iff it is a strong congruence.*

Propositions 4.14 and 4.15 together show the second claim of Theorem 4.12.

If $\bullet$ is the only operation, we can actually use this method to define the types (see Section 3.5). In the following sections we shall develop an algebraic account of semantics, starting first with boolean algebras and then going over to intensionality, and finally carrying out the full algebraization.

Notes on this section. The idea that the logical interconnections between sentences constitute their meanings is also known as *holism*. This view and its implications for semantics is discussed by Dresner [**?**]. We shall briefly also mention the problem of reversibility (see Section 4.6). Most formalisms are designed only for assigning meanings to sentences, but it is generally hard or impossible to assign a sentence that expresses a given content. We shall briefly touch on that issue in Section 4.6.

Exercise 136. Prove Proposition 4.8.

Exercise 137. Let $\rho = \langle \Delta, \varphi \rangle$ be a rule. Devise a mode $\mathtt{M}_\rho$ that captures the effect of this rule in the way discussed above. Translate the rules given above into modes. What happens with 0–ary rules (that is, rules with $\Delta = \varnothing$)?

Exercise 138. There is a threefold characterization of a consequence: as a consequence relation, as a closure operator, and as a set of theories. Let $\vdash$ be a consequence relation. Show that $\Delta \mapsto \Delta^{\vdash}$ is a closure operator. The closed sets are the theories. If $\vdash$ is structural the set of theories of $\vdash$ are inversely closed under substitutions. That is to say, if T is a theory and σ a substitution, then $\sigma^{-1}[T]$ is a theory as well. Conversely, show that every closure operator on $\wp(\mathfrak{Tm}_\Omega(V))$ gives rise to a consequence relation and that the consequence relation is structural if the set of theories is inversely closed under substitutions.

Exercise 139. Show that the rules (4.9) are complete for boolean logic in $\wedge$ and $\neg$.

Exercise 140. Show that for any given finite signature the set of predicate logical formulae valid in all finite structures for that signature is co–recursively enumerable. (The latter means that its complement is recursively enumerable.)

Exercise 141. Let L be a first–order language which contains at least the symbol for equality (=). Show that a first–order theory T in L satisfies Leibniz' Principle if the following holds for any relation symbol r

$$T; \{ (x_i = y_i) : i < \Xi(r) \} \vdash^{\mathsf{FOL}} (r(\vec{x}) \to r(\vec{y})) \tag{4.21}$$

and the following for every function symbol f:

$$T; \{ (x_i{=}y_i) : i < \Omega(f) \} \vdash^{\mathsf{FOL}} (f(\vec{x}){=}f(\vec{y})) \tag{4.22}$$

Use this to show that the first–order set theory ZFC satisfies Leibniz' Principle. Further, show that every equational theory satisfies Leibniz' Principle.

2. Boolean Semantics

Boolean algebras are needed in all areas of semantics, as is demonstrated in [**?**]. Boolean algebras are the structures that correspond to propositional logic in the sense that the variety turns out to be generated from just one algebra: the algebra with two values 0 and 1, and the usual operations (Theorem 4.33). Moreover, the calculus of equations and the usual deductive calculus mutually interpret each other (Theorem 4.36). This allows to show that the axiomatization is complete (Theorem 4.39).

Definition 4.17 *An algebra $\langle B, 0, 1, -, \cap, \cup \rangle$, where $0, 1 \in B$, $-\colon B \to B$ and $\cap, \cup\colon B^2 \to B$, is called a* ***boolean algebra*** *if it satisfies the following equations for all $x, y, z \in B$.*

(as$\cap$)	$x \cap (y \cap z)$	$= (x \cap y) \cap z$	(as$\cup$)	$x \cup (y \cup z)$	$= (x \cup y) \cup z$
(co$\cap$)	$x \cap y$	$= y \cap x$	(co$\cup$)	$x \cup y$	$= y \cup x$
(id$\cap$)	$x \cap x$	$= x$	(id$\cup$)	$x \cup x$	$= x$
(ab$\cap$)	$x \cap (y \cup x)$	$= x$	(ab$\cup$)	$x \cup (y \cap x)$	$= x$
(di$\cap$)	$x \cap (y \cup z)$	$= (x \cap y) \cup (x \cap z)$	(di$\cup$)	$x \cup (y \cap z)$	$= (x \cup y) \cap (x \cup z)$
(li$-$)	$x \cap (-x)$	$= 0$	(ui$\cup$)	$x \cup (-x)$	$= 1$
(ne$\cap$)	$x \cap 1$	$= x$	(ne0)	$x \cup 0$	$= x$
(dm$\cap$)	$-(x \cap y)$	$= (-x) \cup (-y)$	(dm$\cup$)	$-(x \cup y)$	$= (-x) \cap (-y)$
(dn$-$)	$-(-x)$	$= x$			

The operation $\cap$ is generally referred to as the **meet** (**operation**) and $\cup$ as the **join** (**operation**). $-x$ is called the **complement** of x and 0 the **zero** and 1 the **one** or **unit**. Obviously, the boolean algebras form an equationally definable class of algebras.

The laws (as$\cap$) and (as$\cup$) are called **associativity laws**, the laws (co$\cap$) and (co$\cup$) **commutativity laws**, (id$\cap$) and (id$\cup$) the **laws of idempotence**

and (ab$\cap$) and (ab$\cup$) the **laws of absorption**. A structure $\langle L, \cap, \cup \rangle$ satisfying these laws is called a **lattice**. If only one operation is present and the corresponding laws hold we speak of a **semilattice**. (So, a semilattice is a semigroup that satisfies commutativity and idempotence.) Since $\cap$ and $\cup$ are associative and commutative, we follow the general practice and omit brackets whenever possible. So, rather than $(x \cap (y \cap z))$ we simply write $x \cap y \cap z$. Also, $(x \cap (y \cap x))$ is simplified to $x \cap y$. Furthermore, given a finite set $S \subseteq L$ the notation $\bigcup \langle x : x \in S \rangle$ or simply $\bigcup S$ is used for the iterated join of the elements of S. This is uniquely defined, since the join is independent of the order and multiplicity in which the elements appear.

Definition 4.18 *Let $\mathfrak{L}$ be a lattice. We write $x \leq y$ if $x \cup y = y$.*

Notice that $x \leq y$ iff $x \cap y = x$. This can be shown using the equations above. We leave this as an exercise to the reader. Notice also the following.

Lemma 4.19 ① *$\leq$ is a partial ordering.*

② *$x \cup y \leq z$ iff $x \leq z$ and $y \leq z$.*

③ *$z \leq x \cap y$ iff $z \leq x$ and $z \leq y$.*

Proof. ① (a) $x \cup x = x$, whence $x \leq x$. (b) Suppose that $x \leq y$ and $y \leq x$. Then we get $x \cup y = x$ and $y \cup x = y$, whence $y = x \cup y = x$. (c) Suppose that $x \leq y$ and $y \leq z$. Then $x \cup y = y$ and $y \cup z = z$ and so $x \cup z = x \cup (y \cup z) = (x \cup y) \cup z = y \cup z = z$. ② Let $x \cup y \leq z$. Then, since $x \leq x \cup y$, we have $x \leq z$ by (①c); for the same reason also $y \leq z$. Now assume $x \leq z$ and $y \leq z$. Then $x \cup z = y \cup z = z$ and so $z = z \cup z = (x \cup z) \cup (y \cup z) = (x \cup y) \cup z$, whence $x \cup y \leq z$. ③ Similarly, using $x \leq y$ iff $x \cap y = x$. □

In fact, it is customary to define a lattice by means of $\leq$. This is done as follows.

Definition 4.20 *Let $\leq$ be a partial order on L. Let $X \subseteq L$ be an arbitrary set. The **greatest lower bound (glb)** of X, also denoted $\bigcap X$, is that element u such that for all z: if $x \geq z$ for all $x \in X$ then also $u \geq z$ (if it exists). Analogously, the **least upper bound (lub)** of X, denoted by $\bigcup X$, is that element v such that for all z: if $x \leq z$ for all $x \in X$ then also $v \leq z$ (if it exists).*

Notice that there are partial orderings which have no lubs. For example, let $L = \langle \{0,1,2,3\}, \preccurlyeq \rangle$, where

$$\preccurlyeq := \{\langle 0,0\rangle, \langle 0,2\rangle, \langle 0,3\rangle, \langle 1,1\rangle, \langle 1,2\rangle, \langle 1,3\rangle, \langle 2,2\rangle, \langle 3,3\rangle\} \tag{4.23}$$

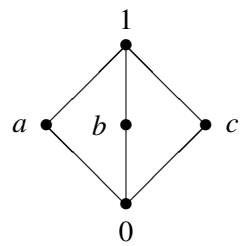


Figure 11. The Lattice M_3

Here, $\{0,1\}$ has no lub. This partial ordering does therefore not come from a lattice. For by the facts established above, the join of two elements x and y is simply the lub of $\{x,y\}$, and the meet is the glb of $\{x,y\}$. It is left to the reader to verify that these operations satisfy all laws of lattices. So, a partial order $\leq$ is the order determined by a lattice structure iff all finite sets have a least upper bound and a greatest lower bound.

The laws (di$\cap$) and (di$\cup$) are the **distributivity laws**. A lattice is called **distributive** if they hold in it. A nice example of a distributive lattice is the following. Take a natural number, say 28, and list all divisors of it: 1, 2, 4, 7, 14, 28. Write $x \leq y$ if x is a divisor of y. (So, $2 \leq 14$, $2 \leq 4$, but not $4 \leq 7$.) Then $\cap$ turns out to be the greatest common divisor and $\cup$ the least common multiple. Another example is the linear lattice defined by the numbers $< n$ with $\leq$ the usual ordering. $\cap$ is then the minimum and $\cup$ the maximum.

A **bounded lattice** is a structure $\langle L,0,1,\cap,\cup\rangle$ which is a lattice with respect to $\cap$ and $\cup$, and in which satisfies (ne$\cap$) and (ne$\cup$). From the definition of $\leq$, (ne$\cap$) means that $x \leq 1$ for all x and (ne$\cup$) that $0 \leq x$ for all x. Every finite lattice has a least and a largest element and can thus be extended to a bounded lattice. This extension is usually done without further notice.

Definition 4.21 *Let $\mathfrak{L} = \langle L,\cap,\cup\rangle$ be a lattice. An element x is **join irreducible in** $\mathfrak{L}$ if for all y and z such that $x = y \cup z$ either $x = y$ or $x = z$. x is **meet irreducible** if for all y and z such that $x = y \cap z$ either $x = y$ or $x = z$.*

It turns out that in a distributive lattice irreducible elements have a stronger property. Call x **meet prime** if for all y and z: from $x \geq y \cap z$ follows $x \geq y$ or $x \geq z$. Obviously, if x is meet prime it is also meet irreducible. The converse is generally false. Look at M_3 shown in Figure 11. Here, $c \geq a \cap b (= 0)$, but neither $c \geq a$ nor $c \geq b$ holds.

Lemma 4.22 *Let $\mathfrak{L}$ be a distributive lattice. Then x is meet (join) prime iff x is meet (join) irreducible.*

Let us now move on to the complement. (li$\cap$) and (ui$\cup$) have no special name. They basically ensure that $-x$ is the unique element y such that $x \cap y = 0$ and $x \cup y = 1$. The laws (dm$\cap$) and (dm$\cup$) are called **de Morgan laws**. Finally, (dn$-$) is the law of **double negation**.

Lemma 4.23 *The following holds in a boolean algebra.*

① $x \leq y$ *iff* $-y \leq -x$.

② $x \leq y$ *iff* $x \cap (-y) = 0$ *iff* $(-x) \cup y = 1$.

Proof. ① $x \leq y$ means $x \cup y = y$, and so $-y = -(x \cup y) = (-x) \cap (-y)$, whence $-y \leq -x$. From $-y \leq -x$ we now get $x = - - x \leq - - y = y$. ② If $x \leq y$ then $x \cap y = x$, and so $x \cap (-y) = (x \cap y) \cap (-y) = x \cap 0 = 0$. Conversely, suppose that $x \cap (-y) = 0$. Then $x \cap y = (x \cap y) \cup (x \cap (-y)) = x \cap (y \cup (-y)) = x \cap 1 = x$. So, $x \leq y$. It is easily seen that $x \cap (-y) = 0$ iff $(-x) \cup y = 1$. □

We can use the terminology of universal algebra (see Section 1.1). So, the notions of homomorphisms and subalgebras, congruences, of these structures should be clear. We now give some examples of boolean algebras. The first example is the powerset of a given set. Let X be a set. Then $\wp(X)$ is a boolean algebra with $\varnothing$ in place of 0, X in place of 1, $-A = X - A$, $\cap$ and $\cup$ the intersection and union. We write $\mathfrak{P}(X)$ for this algebra. A subalgebra of this algebra is called a **field of sets**. Also, a subset of $\wp(X)$ closed under the boolean operations is called a field of sets. The smallest examples are the algebra $\mathbf{1} := \mathfrak{P}(\varnothing)$, consisting just of one element ($\varnothing$), and $\mathbf{2} := \mathfrak{P}(\{\varnothing\})$, the algebra of subsets of $1 = \{\varnothing\}$. Now, let X be a set and $\mathfrak{B} = \langle B, 0, 1, \cap, \cup, - \rangle$ be a boolean algebra. Then for two functions $f, g \colon X \to B$ we may define $-f$, $f \cap g$ and $f \cup g$ as follows.

$$\begin{aligned} (-f)(x) &:= -f(x) \\ (f \cup g)(x) &:= f(x) \cup g(x) \\ (f \cap g)(x) &:= f(x) \cap g(x) \end{aligned} \tag{4.24}$$

Further, let $\underline{0} \colon X \to B \colon x \mapsto 0$ and $\underline{1} \colon X \to B \colon x \mapsto 1$. It is easily verified that the set of all functions from X to B form a boolean algebra: $\langle B^X, \underline{0}, \underline{1}, -, \cap, \cup \rangle$. We denote this algebra by $\mathfrak{B}^X$. The notation has been chosen on purpose: this algebra is nothing but the direct product of $\mathfrak{B}$ indexed over X. A particular

case is $\mathfrak{B} = \mathbf{2}$. Here, we may actually think of $f\colon X \to 2$ as the characteristic function χ_M of a set, namely the set $f^{-1}(1)$. It is then again verified that $\chi_{-M} = -\chi_M$, $\chi_{M\cap N} = \chi_M \cap \chi_N$, $\chi_{M\cup N} = \chi_M \cup \chi_N$. So we find the following.

Theorem 4.24 $\mathbf{2}^X$ *is isomorphic to* $\mathfrak{P}(X)$.

We provide some applications of these results. The intransitive verbs of English have the category $e\backslash t$. Their semantic type is therefore $e \to t$. This in turn means that they are interpreted as functions from objects to truth values. We assume that the truth values are just 0 and 1 and that they form a boolean algebra with respect to the operations $\cap$, $\cup$ and $-$. Then we can turn the interpretation of intransitive verbs into a boolean algebra in the way given above. Suppose that the interpretation of `and`, `or` and `not` is also canonically extended in the given way. That is: suppose that they can now also be used for intransitive verbs and have the meaning given above. Then we can account for a number of inferences, such as the inference from (4.25) to (4.26) and (4.27), and from (4.26) and (4.27) together to (4.25). Or we can infer that (4.25) implies that (4.28) is false; and so on.

(4.25) `Claver walks and talks.`

(4.26) `Claver walks.`

(4.27) `Claver talks.`

(4.28) `Claver does not walk.`

With the help of that we can now also assign a boolean structure to the transitive verb denotations. For their category is $(e\backslash t)/e$, which corresponds to the type $e \to (e \to t)$. Now that the set functions from objects to truth values carries a boolean structure, we may apply the construction again. This allows us then to deduce (4.30) from (4.29).

(4.29) `Claver sees or hears Patrick.`

(4.30) `Claver sees Patrick or Claver hears Patrick.`

Obviously, any category that finally ends in t has a space of denotations associated to it that can be endowed with the structure of a boolean algebra. (See also Exercise 133.) These are, however, not all categories. However, for the remaining ones we can use a trick used already by Montague. Montague was concerned with the fact that names such as `Peter` and `Susan` denote

objects, which means that their type is e. Yet, they fill a subject NP position, and subject NP positions can also be filled by (nominative) quantified NPs such as `some philosopher`, which are of type $(e \to t) \to t$. In order to have homogeneous type assignment, Montague lifted the denotation of `Peter` and `Susan` to $(e \to t) \to t$. In terms of syntactic categories we lift from e to $t/(e\backslash t)$. We have met this earlier in Section 3.4 as raising. Cast in terms of boolean algebras this is the following construction. From an arbitrary set X we first form the boolean algebra $\mathfrak{P}(X)$ and then the algebra $\mathbf{2}^{\mathfrak{P}(X)}$.

Proposition 4.25 *The map $x \mapsto x^\dagger$ given by $x^\dagger(f) := f(x)$ is an embedding of X into $\mathbf{2}^{\mathfrak{P}(X)}$.*

Proof. Suppose that $x \neq y$. Then $x^\dagger(\chi_{\{x\}}) = \chi_{\{x\}}(x) = 1$, while $y^\dagger(\chi_{\{x\}}) = \chi_{\{x\}}(y) = 0$. Thus $x^\dagger \neq y^\dagger$. □

To see that this does the trick, consider the following sentence.

(4.31) `Peter and Susan walk.`

We interpret `Peter` now by $\mathsf{peter}'^\dagger$, where peter' is the individual Peter. Similarly, $\mathsf{susan}'^\dagger$ interprets `Susan`. Then (4.31) means

$$\begin{aligned} &(\mathsf{peter}'^\dagger \cap \mathsf{susan}'^\dagger)(\mathsf{walk}') \\ =&(\mathsf{peter}'^\dagger(\mathsf{walk}')) \cap (\mathsf{susan}'^\dagger(\mathsf{walk}')) \\ =&\mathsf{walk}'(\mathsf{peter}') \cap \mathsf{walk}'(\mathsf{susan}') \end{aligned} \tag{4.32}$$

So, this licenses the inference from (4.31) to (4.33) and (4.34), as required. (We have tacitly adjusted the morphology here.)

(4.33) `Peter walks.`

(4.34) `Susan walks.`

It follows that we can make the denotations of any linguistic category a boolean algebra.

The next theorem we shall prove is that boolean algebras are (up to isomorphism) the same as fields of sets. Before we prove the full theorem we shall prove a special case, which is very important in many applications. An **atom** is an element $x \neq 0$ such that for all $y \leq x$: either $y = 0$ or $y = x$. $\mathrm{At}(\mathfrak{B})$ denotes the set of all atoms of $\mathfrak{B}$.

Lemma 4.26 *In a boolean algebra, an element is an atom iff it is join irreducible.*

This is easy to see. An atom is clearly join irreducible. Conversely, suppose that x is join irreducible. Suppose that $0 \leq y \leq x$. Then

$$x = (x \cap y) \cup (x \cap (-y)) = y \cup (x \cap (-y)) \tag{4.35}$$

By irreducibility, either $y = x$ or $x \cap (-y) = x$. From the latter we get $x \leq -y$, or $y \leq -x$, using Lemma 4.19. Since also $y \leq x$, $y \leq x \cap (-x) = 0$. So, $y = 0$. Therefore, x is an atom. Put

$$\widehat{x} := \{y \in \mathrm{At}(\mathfrak{B}) : y \leq x\} \tag{4.36}$$

The map $x \mapsto \widehat{x}$ is a homomorphism: $\widehat{x} = \mathrm{At}(\mathfrak{A}) - \widehat{x}$. For let u be an atom. For any x, $u = (u \cap x) \cup (u \cap (-x))$; and since u is irreducible, $u = u \cap x$ or $u = u \cap (-x)$, which gives $u \leq x$ or $u \leq -x$. But not both, since $u > 0$. Second, $\widehat{x \cap y} = \widehat{x} \cap \widehat{y}$, as is immediately verified.

Now, if $\mathfrak{B}$ is finite, $\widehat{x}$ is nonempty iff $x \neq 0$.

Lemma 4.27 *If $\mathfrak{B}$ is finite, $x = \bigcup \widehat{x}$.*

Proof. Put $x' := \bigcup \widehat{x}$. Clearly, $x' \leq x$. Now suppose $x' < x$. Then $(-x') \cap x \neq 0$. Hence there is an atom $u \leq (-x') \cap x$, whence $u \leq x$. But $u \not\leq x'$, a contradiction. □

A boolean algebra is said to be **atomic** if x is the lub $\widehat{x}$ for all x.

Theorem 4.28 *Let $\mathfrak{B}$ be a finite boolean algebra. The map $x \mapsto \widehat{x}$ is an isomorphism from $\mathfrak{B}$ onto $\mathfrak{P}(\mathrm{At}(\mathfrak{B}))$.*

Now we proceed to the general case. First, notice that this theorem is false in general. A subset N of M is called **cofinite** if its complement, $M - N$, is finite. Let Ω be the set of all subsets of ω which are either finite or cofinite. Now, as is easily checked, Ω contains $\varnothing$, ω and is closed under complement, union and intersection. The singletons $\{x\}$ are the atoms. However, not every set of atoms corresponds to an element of the algebra. A case in point is $\{\{2k\} : k \in \omega\}$. Its union in ω is the set of even numbers, which is neither finite nor cofinite. Moreover, there exist infinite boolean algebras that have no atoms (see the exercises). Hence, we must take a different route.

Definition 4.29 *Let $\mathfrak{B}$ be a boolean algebra. A **point** is a homomorphism $h\colon \mathfrak{B} \to \mathbf{2}$. The set of points of $\mathfrak{B}$ is denoted by* $\mathrm{Pt}(\mathfrak{B})$.

Notice that points are necessarily surjective. For we must have $h(0^{\mathfrak{B}}) = 0$ and $h(1^{\mathfrak{B}}) = 1$. (As a warning to the reader: we will usually not distinguish $1^{\mathfrak{B}}$ and 1.)

Definition 4.30 *A **filter** of $\mathfrak{B}$ is a subset that satisfies the following.*

① $1 \in F$.

② *If $x, y \in F$ then $x \cap y \in F$.*

③ *If $x \in F$ and $x \leq y$ then $y \in F$.*

*A filter F is called an **ultrafilter** if $F \neq B$ and there is no filter G such that $F \subsetneq G \subsetneq B$.*

A filter F is an ultrafilter iff for all x: either $x \in F$ or $-x \in F$. For suppose neither is the case. Then let F^x be the set of elements y such that there is a $u \in F$ with $y \geq u \cap x$. This is a filter, as is easily checked. It is a proper filter: it does not contain $-x$. For suppose otherwise. Then $-x \geq u \cap x$ for some $u \in F$. By Lemma 4.23 this means that $0 = u \cap x$, from which we get $u \leq -x$. So, $-x \in F$, since $u \in F$. Contradiction.

Proposition 4.31 *Let $h\colon \mathfrak{B} \to \mathfrak{A}$ be a homomorphism of boolean algebras. Then $F_h := h^{-1}(1^{\mathfrak{A}})$ is a filter of $\mathfrak{B}$. Moreover, for any filter F of $\mathfrak{B}$, Θ_F defined by $x\,\Theta_F\,y$ iff $x \leftrightarrow y \in F$ is a congruence. The factor algebra $\mathfrak{B}/\Theta_F$ is also denoted by $\mathfrak{B}/F$ and the map $x \mapsto [x]\Theta_F$ by h_F.*

It follows that if $h\colon \mathfrak{B} \twoheadrightarrow \mathfrak{A}$ then $\mathfrak{A} \cong \mathfrak{B}/F_h$. Now we specialize $\mathfrak{A}$ to $\mathbf{2}$. Then if $h\colon \mathfrak{B} \to \mathbf{2}$, we have a filter $h^{-1}(1)$. It is clear that this must be an ultrafilter. Conversely, given an ultrafilter U, $\mathfrak{B}/U \cong \mathbf{2}$. We state without proof the following theorem. A set $X \subseteq B$ has the **finite intersection property** if for every finite $S \subseteq X$ we have $\bigcap S \neq 0$.

Theorem 4.32 *For every subset of B with the finite intersection property there exists an ultrafilter containing it.*

Now put $\widehat{x} := \{h \in \mathrm{Pt}(\mathfrak{B}) : h(x) = 1\}$. It is verified that

$$\begin{aligned} \widehat{-x} &= -\widehat{x} \\ \widehat{x \cap y} &= \widehat{x} \cap \widehat{y} \\ \widehat{x \cup y} &= \widehat{x} \cup \widehat{y} \end{aligned} \tag{4.37}$$

To see the first, assume $h \in \widehat{-x}$. Then $h(-x) = 1$, from which $h(x) = 0$, and so $h \notin \widehat{x}$, that is to say $h \in -\widehat{x}$. Conversely, if $h \in -\widehat{x}$ then $h(x) \neq 1$, whence $h(-x) = 1$, showing $h \in \widehat{-x}$. Second, $h \in \widehat{x \cap y}$ implies $h(x \cap y) = 1$, so $h(x) = 1$ and $h(y) = 1$, giving $h \in \widehat{x}$ as well as $h \in \widehat{y}$. Conversely, if the latter holds then $h(x \cap y) = 1$ and so $h \in \widehat{x \cap y}$. Similarly with $\cup$.

Theorem 4.33 *The map $x \mapsto \widehat{x}$ is an injective homomorphism from $\mathfrak{B}$ into the algebra $\mathfrak{P}(\mathrm{Pt}(\mathfrak{B}))$. Consequently, every boolean algebra is isomorphic to a field of sets.*

Proof. It remains to see that the map is injective. To that end, let x and y be two different elements. We claim that there is an $h\colon \mathfrak{B} \to \mathbf{2}$ such that $h(x) \neq h(y)$. For we either have $x \not\leq y$, in which case $x \cap (-y) > 0$; or we have $y \not\leq x$, in which case $y \cap -x > 0$. Assume (without loss of generality) the first. There is an ultrafilter U containing the set $\{x \cap (-y)\}$, by Theorem 4.32. Obviously, $x \in U$ but $y \notin U$. Then h_U is the desired point. □

We point out that this means that every boolean algebra is a subalgebra of a direct product of **2**. The variety of boolean algebras is therefore generated by **2**. The original representation theorem for finite boolean algebras can be extended in the following way (this is the route that Keenan and Faltz take). A boolean algebra $\mathfrak{B}$ is called **complete** if any set has a least upper bound and a greatest lower bound.

Theorem 4.34 *Let $\mathfrak{B}$ be a complete atomic boolean algebra. Then $\mathfrak{B} \cong \mathfrak{P}(\mathrm{At}(\mathfrak{B}))$.*

It should be borne in mind that within boolean semantics (say, in the spirit of Keenan and Faltz) the meaning of a particular linguistic item is a member of a boolean algebra, but it may at the same time be a function from some boolean algebra to another. For example, the denotations of adjectives form a boolean algebra, but they may also be seen as functions from the algebra of common noun denotations (type $e \to t$) to itself. These maps are, however, in general not homomorphisms. The meaning of a particular adjective, say `tall`, can in principle be any such function. However, some adjectives behave better than others. Various properties of such functions can be considered.

Definition 4.35 *Let $\mathfrak{B}$ be a boolean algebra and $f\colon B \to B$. f is called* ***monotone*** *iff for all $x, y \in B$: if $x \leq y$ then $f(x) \leq f(y)$. f is called* ***antitone*** *if for all $x, y \in B$: if $x \leq y$ then $f(x) \geq f(y)$. f is called* ***restricting*** *iff for each $x \in B$ $f(x) \leq x$. f is called* ***intersecting*** *iff for each $x \in B$: $f(x) = x \cap f(1)$.*

Adjectives that denote intersecting functions are often also called **intersective**. An example is `white`. A white car is something that is both white and a car. Hence we find that white' is intersecting. Intersecting functions are restricting but not necessarily conversely. The adjective `tall` denotes a restricting function (and is therefore also called **restricting**). A tall student is

Table 10. The Axioms of Propositional Logic

(a0)	$p_0 \to (p_1 \to p_0)$
(a1)	$(p_0 \to (p_1 \to p_2)) \to ((p_0 \to p_1) \to (p_0 \to p_2))$
(a2)	$((p_0 \to p_1) \to p_0) \to p_0$
(a3)	$\bot \to p_0$
(a4)	$\neg p_0 \to (p_0 \to \bot)$
(a5)	$(p_0 \to \bot) \to \neg p_0$
(a6)	$\top$
(a7)	$p_0 \to (p_1 \to (p_0 \wedge p_1))$
(a8)	$(p_0 \wedge p_1) \to p_0$
(a9)	$(p_0 \wedge p_1) \to p_1$
(a10)	$p_0 \to (p_0 \vee p_1)$
(a11)	$p_1 \to (p_0 \vee p_1)$
(a12)	$((p_0 \vee p_1) \to p_2) \to ((p_0 \to p_2) \wedge (p_1 \to p_2))$

certainly a student. Yet, a tall student is not necessarily also tall. The problem is that tallness varies with the property that is in question. (We may analyze it, say, as: belongs to the 10 % of the longest students. Then it becomes clear that it has this property.) Suppose that students of sports are particularly tall. Then a tall student of sports will automatically qualify as a tall student, but a tall student may not be a tall student of sports. On the other hand, if students of sports are particularly short, then a tall student will be a tall student of sports, but the converse need not hold. There are also adjectives that have none of these properties (for example, `supposed` or `alleged`). We will return to sentential modifiers in the next section.

We conclude the section with a few remarks on the connection with theories and filters. Let Ω be the signature of boolean logic: the 0–ary symbols $\top$, $\bot$, the unary $\neg$ and the binary $\vee$ and $\wedge$. Then we can define boolean algebras by means of equations, as we have done with Definition 4.17. For reference, we call the set of equations BEq. Or we may actually define a consequence relation, for example by means of a Hilbert–calculus. Table 10 gives a complete set of axioms, which together with the rule MP axiomatize boolean logic. Call this calculus PC. We have to bring the equational calculus and the deductive calculus into correspondence. We have a calculus of equations (see Section 1.1), which tells us what equations follow from what other equations.

Write $\varphi \leftrightarrow \chi$ in place of $(\varphi \to \chi) \wedge (\chi \to \varphi)$.

Theorem 4.36 *The following are equivalent.*

① $\vdash^{\mathsf{PC}} \varphi \leftrightarrow \chi$.

② *For every boolean algebra* $\mathfrak{A}$*:* $\mathfrak{A} \vDash \varphi = \chi$.

③ $\mathsf{BEq} \vdash \varphi = \chi$.

The proof is lengthy, but routine. ② and ③ are equivalent by the fact that an algebra is a boolean algebra iff it satisfies BEq. So, ① $\Leftrightarrow$ ③ needs proof. It rests on the following

Lemma 4.37 *(a)* $\vdash^{\mathsf{PC}} \varphi$ *iff* $\vdash^{\mathsf{PC}} \top \leftrightarrow \varphi$.
(b) $\mathsf{BEq} \vdash \varphi = \chi$ *iff* $\mathsf{BEq} \vdash \top = \varphi \leftrightarrow \chi$.

Proof. (a) Suppose that $\vdash^{\mathsf{PC}} \varphi$. Since $\vdash^{\mathsf{PC}} \varphi \to (\top \to \varphi)$ we get $\vdash^{\mathsf{PC}} \top \to \varphi$. Similarly, from $\vdash^{\mathsf{PC}} \top$ we get $\vdash^{\mathsf{PC}} \varphi \to \top$. Conversely, if $\vdash^{\mathsf{PC}} \top \leftrightarrow \varphi$, then with (a8) we get $\vdash^{\mathsf{PC}} \top \to \varphi$ and with (a6) and MP, $\vdash^{\mathsf{PC}} \varphi$. (b) We can take advantage of our results on BAs here. Put $a \vartriangle b := (-a \cup b) \cap (a \cup -b)$. The claim boils down to $a \vartriangle b = 1$ iff $a = b$. Now, if $a \vartriangle b = 1$, then $-a \cup b = 1$, from which $a \leq b$, and also $a \cup -b = 1$, from which $b \leq a$. Together this gives $a = b$. Conversely, if $a = b$ then $-a \cup b = -b \cup b = 1$ and $a \cup -b = a \cup -a = 1$, showing $a \vartriangle b = 1$. $\square$

The next thing to show is that if $\mathsf{BEq} \vdash \top = \varphi \to \chi; \top = \varphi$ then also $\mathsf{BEq} \vdash \top = \chi$. Finally, for all φ of the form (a1) – (a12), $\mathsf{BEq} \vdash \top = \varphi$. This will show that $\vdash^{\mathsf{PC}} \varphi$ implies $\mathsf{BEq} \vdash \top = \varphi$. ① is an immediate consequence. For the converse direction, first we establish that for all basic equations $\varphi = \chi$ of BEq we have $\vdash^{\mathsf{PC}} \varphi \leftrightarrow \chi$. This is routine. Closure under substitution is guaranteed for theorems. So we need to show that this is preserved by the inference rules of Proposition 1.12, that is:

$$\vdash^{\mathsf{PC}} \varphi \leftrightarrow \varphi \tag{4.38a}$$

$$\varphi \leftrightarrow \chi \vdash^{\mathsf{PC}} \chi \leftrightarrow \varphi \tag{4.38b}$$

$$\varphi \leftrightarrow \chi; \chi \leftrightarrow \psi \vdash^{\mathsf{PC}} \varphi \leftrightarrow \psi \tag{4.38c}$$

$$\{\varphi_i \leftrightarrow \chi_i : i < \Omega(f)\} \vdash^{\mathsf{PC}} f(\vec{\varphi}) \leftrightarrow f(\vec{\chi}) \tag{4.38d}$$

In the last line, f is one of the basic functions. The verification is once again routine. We shall now show that the so–defined logic is indeed the logic of

the two element matrix with designated element 1. By DT (which holds in PC), $\varphi \leftrightarrow \chi$ iff $\varphi \vdash^{\mathsf{PC}} \chi$ and $\chi \vdash^{\mathsf{PC}} \varphi$.

$$\Theta^{\leftrightarrow} := \{\langle \varphi, \chi \rangle : \vdash^{\mathsf{PC}} \varphi \leftrightarrow \chi\} \tag{4.39}$$

$\Theta^{\leftrightarrow}$ is a congruence on the term algebra. What is more, it is admissible for every deductively closed set. For if Σ is deductively closed and $\varphi \in \Sigma$, then also $\chi \in \Sigma$ for every $\chi \, \Theta^{\leftrightarrow} \, \varphi$, by Modus Ponens.

Lemma 4.38 *$\mathfrak{Tm}_{\Omega}(V)/\Theta^{\leftrightarrow}$ is a boolean algebra. Moreover, if Σ is a deductively closed set in $\mathfrak{Tm}_{\Omega}(V)$ then $\Sigma/\Theta^{\leftrightarrow}$ is a filter on $\mathfrak{Tm}_{\Omega}(V)/\Theta^{\leftrightarrow}$. If Σ is maximally consistent, $\Sigma/\Theta^{\leftrightarrow}$ is an ultrafilter. Conversely, if F is a filter on $\mathfrak{Tm}_{\Omega}(V)/\Theta^{\leftrightarrow}$, then $h^{-1}_{\Theta^{\leftrightarrow}}[F]$ is a deductively closed set. If F is an ultrafilter, this set is a maximally consistent set of formulae.*

Thus, $\vdash^{\mathsf{PC}}$ is the intersection of all $\vDash_{\langle \mathfrak{A}, F \rangle}$, where $\mathfrak{A}$ is a boolean algebra and F a filter. Now, instead of deductively closed sets we can also take maximal (consistent) deductively closed sets. Their image under the canonical map is an ultrafilter. However, the equivalence $\Theta_U := \{\langle x, y \rangle : x \leftrightarrow y \in U\}$ is a congruence, and it is admissible for U. Thus, we can once again factor it out and obtain the following completeness theorem.

Theorem 4.39 $\vdash^{\mathsf{PC}} = \vDash_{\langle \mathbf{2}, \{1\} \rangle}$.

This says that we have indeed axiomatized the logic of the 2–valued algebra. What is more, equations can be seen as statements of equivalence and conversely. We can draw from this characterization a useful consequence. Call a propositional logic **inconsistent** if every formula is a theorem.

Corollary 4.40 PC *is maximally complete. That is to say, if an axiom or rule ρ is not derivable in PC, $\mathsf{PC} + \rho$ is inconsistent.*

Proof. Let $\rho = \langle \Delta, \varphi \rangle$ be a rule that is not derivable in PC. Then by Theorem 4.39 there is a valuation β which makes every formula of Δ true but φ false. Define the following substitution: $\sigma(p) := \top$ if $\beta(p) = 1$, and $\sigma(p) := \bot$ otherwise. Then for every $\chi \in \Delta$, $\sigma(\chi) \leftrightarrow \top$, while $\sigma(\varphi) \leftrightarrow \bot$. Hence, as PC derives $\sigma(\chi)$ for every $\chi \in \Delta$, it also derives $\sigma(\varphi)$, and so $\bot$. On the other hand, in PC, everything follows from $\bot$. Thus, $\mathsf{PC} + \rho$ is inconsistent. $\square$

Notes on this section. The earliest sources of propositional logic are the writing of the Stoa, notably by Chrysippos. Stoic logic was couched in terms of inference rules. The first to introduce equations and a calculus of equations

was Leibniz. The characterization of $\leq$ in terms of union (or intersection) is explicitly mentioned by him. Leibniz only left incomplete notes. Later, de Morgan, Boole and Frege have completed the axiomatization of what is now known as Boolean logic.

Exercise 142. Show that $x \leq y$ iff $x \cap y = x$.

Exercise 143. For a lattice $\mathfrak{L} = \langle L, \cap, \cup \rangle$ define $\mathfrak{L}^d := \langle L, \cup, \cap \rangle$. Show that this is lattice as well. Obviously, $\mathfrak{L}^{dd} = \mathfrak{L}$. $\mathfrak{L}^d$ is called the **dual lattice** of $\mathfrak{L}$. The dual of a lattice term t^d is defined as follows. $x^d := x$ if x is a variable, $(t \cup t')^d := t^d \cap t'^d$, $(t \cap t')^d := t^d \cup t'^d$. Evidently, $\mathfrak{L} \vDash s = t$ iff $\mathfrak{L}^d \vDash s^d = t^d$. Deduce that $s = t$ holds in every lattice iff $s^d = t^d$ holds in every lattice.

Exercise 144. (Continuing the previous exercise.) For a boolean term define additionally $0^d := 1$, $1^d := 0$, $(-t)^d := -t^d$ and $\mathfrak{B}^d := \langle B, 1, 0, \cup, \cap, - \rangle$ for $\mathfrak{B} = \langle B, 0, 1, \cap, \cup, - \rangle$. Show that $\mathfrak{B}^d \cong \mathfrak{B}$. This implies that $\mathfrak{B} \vDash s = t$ iff $\mathfrak{B} \vDash s^d = t^d$.

Exercise 145. Prove Lemma 4.22.

Exercise 146. Let $\leq$ be a partial ordering on L with finite lubs and glbs. Define $x \cup y := \mathrm{lub}\{x, y\}$, and $x \cap y := \mathrm{glb}\{x, y\}$. Show that $\langle L, \cap, \cup \rangle$ is a lattice.

Exercise 147. Let $\mathbb{Z}$ be the set of entire numbers. For $i, j \in \omega$ and $j < 2^i$ let $R_{i,j} := \{m \cdot 2^i + j : m \in \mathbb{Z}\}$. Let H be the set of subsets of $\mathbb{Z}$ generated by all *finite* unions of sets of the form $R_{i,j}$. Show that H forms a field of sets (hence a boolean algebra). Show that it has no atoms.

3. Intensionality

Leibniz' Principle has given rise to a number of problems in formal semantics. One such problem is its alleged failure with respect to intensional contexts. This is what we shall discuss now. The following context does not admit any substitution of A by a B different from A without changing the truth value of the entire sentence.

(4.40) `The expression ‘`*A*`’ is the same expression as ‘`*B*`’.`

Obviously, if such sentences were used to decide about synonymy, no expression is synonymous with any other. However, the feeling with these types of sentences is that the expressions do not enter with their proper meaning here;

one says, the expressions A and B are not **used** in (4.40) they are only **mentioned**. This need not cause problems for our sign based approach. We might for example say that the occurrences of A where A is used are occurrences with a different category than those where A is mentioned. If we do not assume this we must exclude those sentences in which the occurrences of A or B are only mentioned, not used. However, in that case we need a criterion for deciding when an expression is used and when it is mentioned. The picture is as follows. Let $S(x)$ be shorthand for a sentence S missing a constituent x. We call them **contexts**. Leibniz' Principle says that A and B have identical meaning, in symbols $A \equiv B$, iff $S(A) \leftrightarrow S(B)$ is true for all $S(x)$. Now, let Σ be the set of all contexts, and Π the set of all contexts where the missing expression is used, not mentioned. Then we end up with two kinds of identity:

(4.41) $$A \equiv_\Sigma B :\Leftrightarrow (\forall S(x) \in \Sigma)(S(A) \leftrightarrow S(B))$$

(4.42) $$A \equiv_\Pi B :\Leftrightarrow (\forall S(x) \in \Pi)(S(A) \leftrightarrow S(B))$$

Obviously, $\equiv_\Sigma \subseteq \equiv_\Pi$. Generalizing this, we get a Galois correspondence here between certain sets of contexts and equivalence relations on expressions. Contexts outside of Π are called **hyperintensional**. In our view, (4.40) does not contain occurrences of the language signs for A and B but only occurrences of strings. Strings denote themselves. So, what we have inserted are not the same signs as the signs of the language, and this means that Leibniz' Principle is without force in example (4.40) with respect to the signs. However, if put into the context `the meaning of '__'`, we get the actual meaning of A that the language gives to it. Thus, the following is once again transparent for the meanings of A and B:

(4.43) `The expression '`*A*`' has the same meaning as the expression '`*B*`'.`

A hyperintensional context is

(4.44) `John thinks that palimpsests are leaflets.`

What John thinks here is that the expression `palimpsest` denotes a special kind of leaflet, where in fact it denotes a kind of manuscript. Although this is a less direct case of mentioning an expression, it still is the case that the sign with exponent `palimpsest` is not an occurrence of the genuine English language sign, because it is used with a different meaning. The meaning of that sign is once again the exponent (string) itself.

There are other problematic instances of Leibniz' Principle, for example the so–called *intensional* contexts. Consider the following sentences.

(4.45) `The morning star is the evening star.`

(4.46) `John believes that the morning star is the morning star.`

(4.47) `John believes that the morning star is the evening star.`

(4.48) `The square root of 2 is less than 3/2.`

(4.49) `John believes that the square root of 2 is less than 3/2.`

It is known that (4.45) is true. However, it is quite conceivable that (4.46) may be true and (4.47) false. By Leibniz' Principle, we must assume that `the morning star` and `the evening star` have different meaning. However, as Frege points out, in *this* world they refer to the same thing (the planet Venus), so they are not different. Frege therefore distinguishes **reference** (*Bedeutung*) from **sense** (*Sinn*). In (4.45) the expressions enter with their reference, and this is why the sentence is true. In (4.46) and (4.47), however, they do not enter with their reference, otherwise John holds an inconsistent belief. Rather, they enter with their senses, and the senses are different. Thus, we have seen that expressions that are used (not mentioned) in a sentence may either enter with their reference or with their sense. The question is however the same as before: how do we know when an expression enters with its sense rather than its reference? The general feeling is that one need not be worried by that question. Once the sense of an expression is given, we know what its reference is. We may think of the sense as an algorithm that gives us the reference on need. (This analogy has actually been pushed by Yannis Moschovakis, who thinks that sense actually *is* an algorithm (see [**?**]). However, this requires great care in defining the notion of an algorithm, otherwise it is too fine grained to be useful. Moschovakis shows that equality of meaning is decidable, while equality of denotation is not.) Contexts that do not vary with the sense only with the reference of their subexpression are called **extensional**. Nonextensional contexts are **intensional**. Just how fine grained intensional contexts are is a difficult matter. For example, it is not inconceivable that (4.48) is true but (4.49) is false. Since $\sqrt{2} < 1.5$ we expect that it cannot be otherwise, and that one cannot even believe otherwise. This

holds, for example, under the modal analysis of belief by Hintikka [**?**]. Essentially, this is what we shall assume here, too. The problem of intensionality with respect to Leibniz' Principle disappears once we realize that it speaks of identity in meaning, not just identity in denotation. These are totally different things, as Frege rightly observed. Of course, we still have to show how meaning and denotation work together, but there is no problem with Leibniz' Principle.

Intensionality has been a very important area of research in formal semantics, partly because Montague already formulated an intensional system. The influence of Carnap is clearly visible here. It will turn out that equating intensionality with normal modal operators is not always helpful. Nevertheless, the study of intensionality has helped enormously in understanding the process of algebraization.

Let $A := \{\mathtt{(},\mathtt{)},\mathtt{p},\mathtt{0},\mathtt{1},\wedge,\neg,\Box\}$, where the boolean symbols are used as before and $\Box$ is a unary symbol, which is written before its argument. We form expressions in the usual way, using brackets. The language we obtain shall be called L_M. The abbreviations $\varphi \to \chi$ and $\varphi \leftrightarrow \chi$ as well as typical shorthands (omission of brackets) are used without warning. Notice that we have a propositional language, so that the notions of substitution, consequence relation and so on can be taken over straightforwardly from Section 4.1.

Definition 4.41 *A **modal logic** is a subset L of L_M which contains all boolean tautologies and which is closed under substitution and Modus Ponens. L is called **classical** if from $\varphi \leftrightarrow \chi \in L$ follows that $\Box\varphi \leftrightarrow \Box\chi \in L$, **monotone** if from $\varphi \to \chi \in L$ follows $\Box\varphi \to \Box\chi \in L$. L is **normal** if for all $\varphi, \chi \in L_M$ (a) $\Box(\varphi \to \chi) \to (\Box\varphi \to \Box\chi) \in L$, (b) if $\varphi \in L$ then $\Box\varphi \in L$.*

The smallest normal modal logic is denoted by K, after Saul Kripke. A **quasi–normal** modal logic is a modal logic that contains K. One also defines

$$\Diamond\varphi := \neg(\Box(\neg\varphi)) \tag{4.50}$$

and calls this the **dual operator** (see Exercise 144). $\Box$ is usually called a **necessity operator**, $\Diamond$ a **possibility operator**. If φ is an axiom and L a (normal) modal logic, then $L + \varphi$ ($L \oplus \varphi$) is the smallest (normal) logic containing $L \cup \{\varphi\}$. Analogously the notation $L + \Gamma$, $L \oplus \Gamma$ for a set Γ is defined.

Definition 4.42 *Let L be a modal logic. Then $\vdash_L$ is the following consequence relation. $\Delta \vdash_L \varphi$ iff φ can be deduced from $\Delta \cup L$ using (mp) only. $\Vdash_L$ is the consequence relation generated by the axioms of L, the rule (mp)*

and the rule (mn): $\langle\{\mathtt{p}\},(\Box\mathtt{p})\rangle$. $\vdash_L$ *is called the* ***local consequence relation****,* $\Vdash_L$ *the* ***global consequence relation*** *associated with L.*

It is left to the reader to verify that this indeed defines a consequence relation. We remark here that for $\vdash_{\mathsf{K}}$ the rule (mn) is by definition admissible. However, it is not derivable (see the exercises) while in $\Vdash_{\mathsf{K}}$ it is, by definition. Before we develop the algebraic approach further, we shall restrict our attention to normal logics. For these logics, a geometric (or 'model theoretic') semantics has been given.

Definition 4.43 *A* ***Kripke–frame*** *is a pair* $\langle F,\lhd\rangle$ *where* F *is a set, the* ***set of worlds****, and* $\lhd\subseteq F^2$*, the* ***accessibility relation****. A* ***generalized Kripke–frame*** *is a triple* $\langle F,\lhd,\mathbb{F}\rangle$ *where* $\langle F,\lhd\rangle$ *is a Kripke–frame and* $\mathbb{F}\subseteq\wp(F)$ *a field of sets closed under the operation* $\blacksquare$ *on* $\wp(F)$ *defined as follows:*

$$\blacksquare A := \{x:\ \textit{for all } y:\ \textit{if } x\lhd y \textit{ then } y\in A\} \tag{4.51}$$

Call a **valuation** into a general Kripke–frame $\mathfrak{F}=\langle F,\lhd,\mathbb{F}\rangle$ a function $\beta: V\to\mathbb{F}$.

$$\begin{aligned}
\langle\mathfrak{F},\beta,x\rangle &\vDash p \Leftrightarrow x\in\beta(p) \qquad (p\in V)\\
\langle\mathfrak{F},\beta,x\rangle &\vDash (\neg\varphi) \Leftrightarrow \langle\mathfrak{F},\beta,x\rangle\nvDash\varphi\\
\langle\mathfrak{F},\beta,x\rangle &\vDash (\varphi\wedge\chi) \Leftrightarrow \langle\mathfrak{F},\beta,x\rangle\vDash\varphi;\chi\\
\langle\mathfrak{F},\beta,x\rangle &\vDash (\Box\varphi) \Leftrightarrow \text{for all } y\text{: if } x\lhd y \text{ then } \langle\mathfrak{F},\beta,y\rangle\vDash\varphi
\end{aligned} \tag{4.52}$$

(One often writes $x\vDash\varphi$, suppressing $\mathfrak{F}$ and β.) Furthermore, the **local frame consequence** is defined as follows. $\Delta\vDash_{\mathfrak{F}}\varphi$ if for every β and x: if $\langle\mathfrak{F},\beta,x\rangle\vDash\delta$ for every $\delta\in\Delta$ then $\langle\mathfrak{F},\beta,x\rangle\vDash\varphi$. This is a consequence relation. Moreover, the axioms and rules of PC are valid. Furthermore,

$$\vDash_{\mathfrak{F}}\Box(\varphi\to\chi)\to(\Box\varphi\to\Box\chi) \tag{4.53}$$

For if $x\vDash\Box(\varphi\to\chi);\Box\varphi$ and $x\lhd y$ then $y\vDash\varphi\to\chi;\varphi$, from which $y\vDash\chi$. As y was arbitrary, $x\vDash\Box\chi$. Finally, suppose that $\mathfrak{F}\vDash\varphi$. Then $\mathfrak{F}\vDash\Box\varphi$. For choose x and β. Then for all y such that $x\lhd y$: $\langle\mathfrak{F},\beta,y\rangle\vDash\varphi$, by assumption. Hence $\langle\mathfrak{F},\beta,x\rangle\vDash\Box\varphi$. Since x and β were arbitrarily chosen, the conclusion follows. Define $\Delta\vDash^g_{\mathfrak{F}}\varphi$ if for all β: if $\overline{\beta}(\delta)=F$ for all $\delta\in\Delta$, then also $\overline{\beta}(\varphi)=F$. This is the **global frame consequence** determined by $\mathfrak{F}$. For a class of frames $\mathcal{K}$ we put

$$\vDash_{\mathcal{K}} := \bigcap\langle\vDash_{\mathfrak{F}}:\ \mathfrak{F}\in\mathcal{K}\rangle \tag{4.54}$$

Analogously, $\vDash^g_{\mathcal{K}}$ is the intersection of all $\vDash^g_{\mathfrak{F}}$, $\mathfrak{F}\in\mathcal{K}$.

Theorem 4.44 *For every class $\mathcal{K}$ of frames there is a modal logic L such that $\vDash_{\mathcal{K}} = \vdash_L$. Moreover, $\vDash^g_{\mathcal{K}} = \Vdash_L$.*

Proof. We put $L := \{\varphi : \varnothing \vDash_{\mathcal{K}} \varphi\}$. We noticed that this is a normal modal logic if $\mathcal{K}$ is one membered. It is easy to see that this therefore holds for all classes of frames. Clearly, since both $\vdash_L$ and $\vDash_{\mathcal{K}}$ have a deduction theorem, they are equal if they have the same tautologies. This we have just shown. For the global consequence relation, notice first that $L = \{\varphi : \varnothing \vDash^g_{\mathcal{K}} \varphi\}$. Moreover, $\Vdash_L$ is the smallest global consequence relation containing $\vdash_L$, and similarly $\vDash^g_{\mathcal{K}}$ the smallest global consequence relation containing $\vDash_{\mathcal{K}}$. □

We shall give some applications of modal logic to the semantics of natural language. The first is that of (meta)physical necessity. In uttering (4.55) we suggest that (4.56) obtains whatever the circumstances. Likewise, in uttering (4.57) we suggest that there are circumstances under which (4.58) is true.

(4.55) `2+3 is necessarily greater than 4.`

(4.56) `2+3 is greater than 4.`

(4.57) `Caesar might not have defeated Vercingetorix.`

(4.58) `Caesar has not defeated Vercingetorix.`

The analysis is as follows. We consider `necessarily` as an operator on sentences. Although it appears here in postverbal position, it may be rephrased by `it is necessary that`, which can be iterated any number of times. The same can be done with `might`, which can be rephrased as `it is possible that` and turns out to be the dual of the first. We disregard questions of form here and represent sentential operators simply as $\Box$ and $\Diamond$, prefixed to the sentence in question. $\Box$ is a modal operator, and it is normal. For example, if A and B are both necessary, then so is $A \wedge B$, and conversely. Second, if A is logically true, then A is necessary. Necessity has been modelled following to Carnap by frames of the form $\langle W, W \times W\rangle$. Metaphysically possible worlds should be possible no matter what is the case (that is, no matter which world we are in). It turns out that the interpretation above yields a particular logic, called $\mathsf{S5}$.

(4.59) $\mathsf{S5} := \mathsf{K} \oplus \{p \to \Diamond p, \Diamond p \to \Diamond\Diamond p, p \to \Box\Diamond p\}$

We defer a proof of the fact that this characterizes $\mathsf{S5}$.

Hintikka [**?**] has axiomatized the logic of knowledge and belief. Write $[K_J]\varphi$ to represent the proposition 'John knows that φ' and $[B_J]\varphi$ to represent the proposition 'John believes that φ'. Then, according to Hintikka, both

turn out to be normal modal operators. In particular, we have the following axioms.

(4.60)	$[B_J](\varphi \to \chi)$ $\to ([B_J]\varphi \to [B_J]\chi)$	logical 'omniscience' for belief
(4.61)	$[B_J]\varphi \to [B_J][B_J]\varphi$	positive introspection for belief
(4.62)	$[K_J](\varphi \to \chi)$ $\to ([K_J]\varphi \to [K_J]\chi)$	logical omniscience
(4.63)	$[K_J]\varphi \to \varphi$	factuality of knowledge
(4.64)	$[K_J]\varphi \to [K_J][K_J]\varphi$	positive introspection
(4.65)	$\neg[K_J]\varphi \to [K_J]\neg[K_J]\varphi$	negative introspection

Further, if φ is a theorem, so is $[B_J]\varphi$ and $[K_J]\varphi$. Now, we may either study both operators in isolation, or put them together in one language, which now has two modal operators. We trust that the reader can make the necessary amendments to the above definitions to take care of any number of operators. We can can then also formulate properties of the operators in combination. It turns out, namely, that the following holds.

(4.66) $[K_J]\varphi \to [B_J]\varphi$

The logic of $[B_J]$ is known as $\mathsf{K4} := \mathsf{K} \oplus \Diamond\Diamond p \to \Diamond p$ and it is the logic of all transitive Kripke–frames; $[K_J]$ is once again $\mathsf{S5}$. The validity of this set of axioms for the given interpretation is of course open to question.

A different interpretation of modal logic is in the area of *time*. Here there is no consensus on how the correct model structures look like. If one believes in determinism, one may for example think of time points as lying on the real line $\langle \mathbb{R}, < \rangle$. Introduce an operator $\boxminus$ by

(4.67) $\langle \mathbb{R}, <, \beta, t \rangle \vDash \boxminus\chi :\Leftrightarrow$ for all $t' > t : \langle \mathbb{R}, <, \beta, t' \rangle \vDash \chi$

One may read $\boxminus\chi$ as *it will always be the case that* χ. Likewise, $\diamondminus\chi$ may be read as *it will at least once be the case that* χ. The logic of $\boxminus$ is

(4.68) $\mathsf{Th}\,\langle \mathbb{R}, < \rangle := \{\chi :$ for all $\beta, x : \langle \mathbb{R}, <, \beta, x \rangle \vDash \chi\}$

Alternatively, we may define an operator $\boxminus$ by

(4.69) $\langle \mathbb{R}, <, \beta, t \rangle \vDash \boxminus\chi :\Leftrightarrow$ for all $t' < t : \langle \mathbb{R}, <, \beta, t' \rangle \vDash \chi$

to be read as *it has always been the case that* χ. Finally, $\diamond\!\!\!\!-\chi$ reads *it has been the case that* χ. On $\langle \mathbb{R}, < \rangle$, $\boxminus$ has the same logic as $\boxminus$. We may also study both operators in combination. What we get is a bimodal logic (which is simply a logic over a language with two operators, each defining a modal logic in its own fragment). Furthermore, $\langle \mathbb{R}, < \rangle \vDash p \to \boxminus\diamond\!\!\!\!-p; p \to \boxminus\diamond\!\!\!\!-p$. The details need not be of much concern here. Suffice it to say that the modelling of time with the help of modal logic has received great attention in philosophy and linguistics. Obviously, to be able to give a model theory of tenses is an important task. Already Montague integrated into his theory a treatment of time in combination with necessity (as discussed above).

We shall use the theory of matrices to define a semantics for these logics. We have seen earlier that one can always choose matrices of the form $\langle \mathfrak{Tm}_\Omega(V), \Delta \rangle$, Δ deductively closed. Now assume that L is classical. Then put $\varphi\ \Theta_L^{\leftrightarrow}\ \chi$ if $\varphi \leftrightarrow \chi \in L$. This is a congruence relation, and we can form the factor algebra along that congruence. (Actually, classicality is exactly the condition that $\leftrightarrow$ induces a congruence relation.) It turns out that this algebra is a boolean algebra and that $\Box$ is interpreted by a function $\blacksquare$ on that boolean algebra (and $\Diamond$ by a function $\blacklozenge$).

Definition 4.45 *A* ***boolean algebra with (unary) operators (BAO)*** *is an algebra* $\langle A, 0, 1, \cap, \cup, -, \langle \blacksquare_i : i < \kappa \rangle \rangle$ *such that* $\blacksquare_i \colon A \to A$ *for all* $i < \kappa$.

If furthermore L is a normal modal logic, $\blacksquare$ turns out to be a so–called hemimorphism.

Definition 4.46 *Let* $\mathfrak{B}$ *be a boolean algebra and* $h \colon B \to B$ *a map.* h *is called a* ***hemimorphism*** *if (i)* $\blacksquare 1 = 1$ *and (2) for all* $x, y \in B$: $\blacksquare(x \cap y) = \blacksquare(x) \cap \blacksquare(y)$. *A* ***multimodal algebra*** *is an algebra* $\mathfrak{M} = \langle M, 0, 1, \cap, \cup, -, \langle \blacksquare_i : i < \kappa \rangle \rangle$, *where* $\langle M, 0, 1, \cap, \cup, - \rangle$ *is a boolean algebra and* $\blacksquare_i$, $i < \kappa$, *a hemimorphism on it.*

We shall remain with the case $\kappa = 1$ for reasons of simplicity. A hemimorphism is thus not a homomorphism (since it does not commute with $\cup$). The modal algebras form the semantics of modal propositional logic. We also have to look at the deductively closed sets. First, if $\varphi\ \Theta_L^{\leftrightarrow}\ \chi$ then $\varphi \in \Delta$ iff $\chi \in \Delta$. So, we can factor Δ by $\Theta_L^{\leftrightarrow}$. It turns out that Δ, being closed under (mp), becomes a filter of the boolean quotient algebra. Thus, normal modal logics are semantically complete with respect to matrices $\langle \mathfrak{M}, F \rangle$, where $\mathfrak{M}$ is a modal algebra and F a filter. We can refine this still further to F being an

ultrafilter. This is so since if $\Delta \nvdash_L \varphi$ there actually is a maximally consistent set of formulae that contains Δ but not φ, and reduced by $\Theta_L^{\leftrightarrow}$ this turns into an ultrafilter. Say that $\mathfrak{M} \vDash \chi$ if $\langle \mathfrak{M}, U \rangle \vDash \chi$ for all ultrafilters U on $\mathfrak{M}$. Since x is in all ultrafilters iff $x = 1$, we have $\mathfrak{M} \vDash \chi$ exactly if for all homomorphisms h into $\mathfrak{M}$, $h(\chi) = 1$. (Equivalently, $\mathfrak{M} \vDash \chi$ iff $\langle \mathfrak{M}, \{1\} \rangle \vDash \chi$.) Notice that $\mathfrak{M} \vDash \varphi \to \chi$ if for all h: $h(\varphi) \leq h(\chi)$.

Now we shall apply the representation theory of the previous section. A boolean algebra can be represented by a field of sets, where the base set is the set of all ultrafilters (alias points) over the boolean algebra. Now take a modal algebra $\mathfrak{M}$. Underlying it we find a boolean algebra, which we can represent by a field of sets. The set of ultrafilters is denoted by $U(\mathfrak{M})$. Now, for two ultrafilters U, V put $U \lhd V$ iff for all $\blacksquare x \in U$ we have $x \in V$. Equivalently, $U \lhd V$ iff $x \in V$ implies $\blacklozenge x \in U$. We end up with a structure $\langle U(\mathfrak{M}), \lhd, \mathbb{S} \rangle$, where $\lhd$ is a binary relation over $U(\mathfrak{M})$ and $\mathbb{S} \subseteq \wp(U(\mathfrak{M}))$ a field of sets closed under the operation $A \mapsto \blacksquare A$.

A modal algebra $\mathfrak{M}$ is an $\mathsf{S5}$–algebra if it satisfies the axioms given above. Let $\mathfrak{M}$ be an $\mathsf{S5}$–algebra and U, V, W ultrafilters. Then (a) $U \lhd U$. For let $x \in U$. Then $\blacklozenge x \in U$ since $\mathfrak{M} \vDash p \to \Diamond p$. Hence, $U \lhd U$. (b) Assume $U \lhd V$ and $V \lhd W$. We show that $U \lhd W$. Pick $x \in W$. Then $\blacklozenge x \in V$ and so $\blacklozenge\blacklozenge x \in U$. Since $\mathfrak{M} \vDash \Diamond\Diamond p \to \Diamond p$, we have $\blacklozenge x \in U$. (c) Assume $U \lhd V$. We show $V \lhd U$. To this end, pick $x \in U$. Then $\blacksquare\blacklozenge x \in U$. Hence $\blacklozenge x \in V$, by definition of $\lhd$. Hence we find that $\lhd$ is an equivalence relation on $U(\mathfrak{M})$. More exactly, we have shown the following.

Proposition 4.47 *Let $\mathfrak{M}$ be a modal algebra, and $\lhd \subseteq U(\mathfrak{M})^2$ be defined as above.*

① $\mathfrak{M} \vDash p \to \Diamond p$ *iff* $\lhd$ *is reflexive.*

② $\mathfrak{M} \vDash \Diamond\Diamond p \to \Diamond p$ *iff* $\lhd$ *is transitive.*

③ $\mathfrak{M} \vDash p \to \Box\Diamond p$ *iff* $\lhd$ *is symmetric.*

The same holds for Kripke–frames. For example, $\langle F, \lhd \rangle \vDash p \to \Diamond p$ iff $\lhd$ is reflexive. Therefore, $\langle U(\mathfrak{M}), \lhd \rangle$ already satisfies all the axioms of $\mathsf{S5}$. Finally, let $\langle F, \lhd \rangle$ be a Kripke–frame, $G \subseteq F$ be a set such that $x \in G$ and $x \lhd y$ implies $y \in G$. (Such sets are called **generated**.) Then the induced frame $\langle G, \lhd \cap G^2 \rangle$ is called a **generated subframe**. A special case of a generated subset is the set $F \uparrow x$ consisting of all points that can be reached in finitely many steps

from x. Write $\mathfrak{F} \uparrow x$ for the generated subframe induced by $F \uparrow x$. Then a valuation β on $\mathfrak{F}$ induces a valuation on $\mathfrak{F} \uparrow x$, which we denote also by β.

Lemma 4.48 *$\langle \mathfrak{F}, \beta, x \rangle \vDash \varphi$ iff $\langle \mathfrak{F} \uparrow x, \beta, x \rangle \vDash \varphi$. It follows that if $\mathfrak{F} \vDash L$ and $\mathfrak{G}$ is a generated subframe of $\mathfrak{F}$ then also $\mathfrak{G} \vDash L$.*

A special consequence is the following. Let $\mathfrak{F}_0 := \langle F_0, \lhd_0 \rangle$ and $\mathfrak{F}_1 := \langle F_1, \lhd_1 \rangle$ be Kripke–frames. Assume that F_0 and F_1 are disjoint.

$$\mathfrak{F}_0 \oplus \mathfrak{F}_1 := \langle F_0 \cup F_1, \lhd_0 \cup \lhd_1 \rangle \tag{4.70}$$

Then $\mathfrak{F}_0 \oplus \mathfrak{F}_1 \vDash \varphi$ iff $\mathfrak{F}_0 \vDash \varphi$ and $\mathfrak{F}_1 \vDash \varphi$. (For if $x \in F_0$ then $\langle \mathfrak{F}_0 \oplus \mathfrak{F}_1, \beta, x \rangle \vDash \varphi$ iff $\langle \mathfrak{F}_0, \beta, x \rangle \vDash \varphi$, and analogously for $x \in F_1$.) It follows that a modal logic which is determined by some class of Kripke–frames is already determined by some class of Kripke–frames generated from a single point. This shows the following.

Theorem 4.49 `S5` *is the logic of all Kripke–frames of the form $\langle M, M \times M \rangle$.*

Now that we have looked at intensionality we shall look at the question of individuation of meanings. In algebraic logic a considerable amount of work has been done concerning the semantics of propositional languages. Notably in Blok and Pigozzi [**?**] Leibniz' Principle was made the starting point of a definition of algebraizability of logics. We shall exploit this work for our purposes here. We start with a propositional language of signature Ω. Recall the definition of logics, consequence relation and matrix from Section 4.1. We distinguish between a *theory*, *(world) knowledge* and a *meaning postulate*.

Definition 4.50 *Let $\vdash$ be a consequence relation. A $\vdash$–**theory** is a set Δ such that $\Delta^{\vdash} = \Delta$. If T is a set such that $T^{\vdash} = \Delta$, T is called an **axiomatization of** Δ.*

Theories are therefore sets of formulae, and they may contain variables. For example, $\{$`p0`, `(p1→(¬p01))`$\}$ is a theory. However, in virtue of the fact that variables are placeholders, it is not appropriate to say that knowledge is essentially a theory. Rather, for a theory to be knowledge it must be closed under substitution. Sets of this form shall be called *logics*.

Definition 4.51 *Let $\vdash$ be a structural consequence relation. A $\vdash$–**logic** is a $\vdash$–theory closed under substitution.*

Finally, we turn to meaning postulates. Here, it is appropriate not to use sets of formulae, but rather equations.

Definition 4.52 *Let L be a propositional language. A* ***meaning postulate*** *for L is an equation. Given a set M of meaning postulates, an equation $s = t$ follows from M if $s = t$ holds in all algebras satisfying M.*

Thus, the meaning postulates effectively axiomatize the variety of meaning algebras, and the consequences of a set of equations can be derived using the calculus of equations of Section 1.1. In particular, if f is an n–ary operation and $s_i = t_i$ holds in the variety of meaning algebras, so does $f(\vec{s}) = f(\vec{t})$, and likewise, if $s = t$ holds, then $\sigma(s) = \sigma(t)$ holds for any substitution σ. These are natural consequences if we assume that meaning postulates characterize identity of meaning. We shall give an instructive example.

(4.71) `Caesar crossed the Rubicon.`

(4.72) `John does not believe that Caesar crossed the Rubicon.`

(4.73) `Bachelors are unmarried men.`

(4.74) `John does not believe that bachelors are unmarried men.`

It is not part of the meanings of the words that Caesar crossed the Rubicon, so John may safely believe or disbelieve it. However, it is part of the language that bachelors are unmarried men, so not believing it means associating different meanings to the words. Thus, if (4.73) is true and moreover a meaning postulate, (4.74) cannot be true.

It is unfortunate having to distinguish postulates that take the form of a formula from those that take the form of an equation. Therefore, one has sought to reduce the equational calculus to the logical calculus and conversely. The notion of equivalential logic has been studied among other by Janusz Czelakowski and Roman Suszko. The following definition is due to Prucnal and Wroński [**?**]. (For a set Φ of formulae, we write $\Delta \vdash \Phi$ to say that $\Delta \vdash \varphi$ for all $\varphi \in \Phi$.)

Definition 4.53 *Let $\vdash$ be a consequence relation. We call the set $\Delta(p,q) =$*

$\{\delta_i(p,q) : i \in I\}$ *a* ***set of equivalential terms for*** $\vdash$ *if the following holds*

(4.75a) $\vdash \Delta(p,p)$

(4.75b) $\Delta(p,q) \vdash \Delta(q,p)$

(4.75c) $\Delta(p,q);\Delta(q,r) \vdash \Delta(p,r)$

(4.75d) $\bigcup_{i<\Omega(f)} \Delta(p_i,q_i) \vdash \Delta(f(\vec{p}),f(\vec{q}))$

(4.75e) $p;\Delta(p,q) \vdash q$

$\vdash$ *is called* ***equivalential*** *if it has a set of equivalential terms, and* ***finitely equivalential*** *if it has a finite set of equivalential terms. If* $\Delta(p,q) = \{\delta(p,q)\}$ *is a set of equivalential terms for* $\vdash$ *then* $\delta(p,q)$ *is called an* ***equivalential term*** *for* $\vdash$.

As the reader may check, $p \leftrightarrow q$ is an equivalential term for $\vdash^{\mathsf{PC}}$. If there is no equivalential term then synonymy is not definable language internally. [**?**] discusses the nature of meaning postulates. He requires among other that meaning postulates should be expressible in the language itself. To that effect we can introduce a 0–ary symbol 1 and a binary symbol Δ such that Δ is an equivalential term for $\vdash$ in the expanded language. (So, we add (4.75) for $\Delta(p,q) := \{(p\,\Delta\,q)\}$.) To secure that Δ and 1 do the job as intended, we shall stipulate that the logical and the equational calculus are intertranslatable in the following way.

(4.76) $\{s_i{=}t_i : i < n\} \vDash u{=}v \quad \Leftrightarrow \quad \{(s_i\,\Delta\,t_i) : i < n\} \vdash (u\,\Delta\,v)$

(4.77) $\{\delta_i : i < \kappa\} \vdash \varphi \quad \Leftrightarrow \quad \{\delta_i{=}1 : i < \kappa\} \vDash \varphi{=}1$

Here, $\vDash$ denotes model theoretic consequence, or, alternatively, derivability in the equational calculus (see Section 1.1). In this way, equations are translated into sets of formulae. In order for this translation to be faithful in both directions we must require the following (see [**?**]).

(4.78) $x \vdash (x\,\Delta\,1)$ (G–rule)

An equivalent condition is $x;y \vdash (x\,\Delta\,y)$. Second, we must require that

(4.79) $(x\,\Delta\,y){=}1 \vDash x{=}y$ (R–rule)

Then one can show that on any algebra $\mathfrak{A}$ and any two congruences Θ, Θ' on $\mathfrak{A}$, $\Theta = \Theta'$ iff $[1]\Theta = [1]\Theta'$, so every congruence is induced by a theory.

(Varieties satisfying this are called **congruence regular**.) Classical modal logics admit the addition of $\triangle$. 1 is simply $\top$. The postulates can more or less directly be verified. Notice however that for a modal logic L there are two choices for $\vdash$ in Definition 4.53: if we choose $\Vdash_L$ then $p \leftrightarrow q$ is an equivalential term; if, however, we choose $\vdash_L$ then $\{\Box^n(p \leftrightarrow q) : n \in \omega\}$ is a set of equivalential terms. In general no finite set can be named in the local case.

In fact, this holds for any logic which is an extension of boolean logic by any number of congruential operators. There we may conflate meaning postulates with logics. However, call a logic **Fregean** if it satisfies $(p \leftrightarrow q) \to (p \triangle q)$. A modal logic is Fregean iff it contains $p \to \Box p$. There are exactly four Fregean modal logics the least of which is $\mathsf{K} \oplus p \to \Box p$. The other three are $\mathsf{K} \oplus p \leftrightarrow \Box p$, $\mathsf{K} \oplus \Box\bot$ and $\mathsf{K} \oplus \bot$, the inconsistent logic. This follows from the following theorem.

Proposition 4.54 $\mathsf{K} \oplus p \to \Box p = \mathsf{K} \oplus (\Box p \leftrightarrow (p \vee \Box\bot))$.

Proof. Put $L := \mathsf{K} \oplus p \to \Box p$. $\Box p \leftrightarrow (p \vee \Box\bot) \vdash_{\mathsf{K}} p \to \Box p$, so have to show that $\Box p \leftrightarrow (p \vee \Box\bot) \in L$. (1) $\vdash_{\mathsf{K}} \Box p \to (\Diamond p \vee \Box\bot)$. Furthermore, $\vdash_L \Diamond p \to p$. Hence also $\vdash_L \Box p \to (p \vee \Box\bot)$. (2) $\Box\bot \to \Box p$ is a theorem of K, $p \to \Box p$ holds by assumption. This shows the claim. $\Box$

Now, in a Fregean logic, any proposition φ is equivalent either to a nonmodal proposition or to a proposition $(\chi \wedge \Diamond\top) \vee (\chi' \wedge \Box\bot)$, where χ and χ' are nonmodal. It follows from this that the least Fregean logic has only constant extensions: by the axiom $\Box\bot$, by its negation $\Diamond\top$, or both (which yields the inconsistent logic).

Now let us return to Leibniz' Principle. Fix a theory T. Together with the algebra of formulae it forms a matrix. This matrix tells us what is true and what is not. Notice that the members of the algebra are called truth–values in the language of matrices. In the present matrix, a sentence is true only if it is a member of T. Otherwise it is false. Thus, although we can have as many truth values as we like, sentences are simply true or false. Now apply Leibniz' Principle. It says: two propositions φ and φ' have identical meaning iff for every proposition χ and every variable p: $[\varphi/p]\psi \in T$ iff $[\varphi'/p]\chi \in T$. This leads to the following definition.

Definition 4.55 *Let Ω be a signature and $\mathfrak{A}$ an Ω–algebra. Then for any $F \subseteq A$ put*

$$\Delta_{\mathfrak{A}} F := \{\langle a, b\rangle : \textit{for all } t \in \mathrm{Pol}_1(\mathfrak{A}) : t(a) \in F \Leftrightarrow t(b) \in F\} \tag{4.80}$$

*This is called the **Leibniz equivalence** and the map $\Delta_{\mathfrak{A}}$ the **Leibniz operator**.*

Notice that the definition uses unary polynomials, not just terms. This means in effect that we have enough constants to name all expressible meanings, not an unreasonable assumption.

Lemma 4.56 *$\Delta_{\mathfrak{A}}F$ is an admissible congruence on $\langle \mathfrak{A}, F\rangle$.*

Proof. It is easy to see that this is an equivalence relation. By Proposition 4.15 it is a congruence relation. We show that it is compatible with F. Let $x \in F$ and $x\,\Delta_{\mathfrak{A}}F\,y$. Take $t := p$ we get $[x/p]t = x$, $[y/p]p = y$. Since $x \in F$ we have $y \in F$ as well. □

We know that the consequence relation of $\langle \mathfrak{A}, F\rangle$ is the same as the congruence relation of $\langle \mathfrak{A}/\Delta_{\mathfrak{A}}F, F/\Delta_{\mathfrak{A}}F\rangle$. So, from a semantical point of view we may simply factor out the congruence $\Delta_{\mathfrak{A}}F$. Moreover, this matrix satisfies Leibniz' Principle! So, in aiming to define meanings from the language and its logic, we must first choose a theory and then factor out the induced Leibniz equivalence. We may then take as the meaning of a proposition simply its equivalence class with respect to that relation. Yet, the equivalence depends on the theory chosen and so do therefore the meanings. If the 0–ary constant c_0 is a particular sentence (say, that Caesar crossed the Rubicon) then depending on whether this sentence is true or not we get different meanings for our language objects. However, we shall certainly not make the assumption that meaning depends on accidental truth. Therefore, we shall say the following.

Definition 4.57 *Let $\vdash$ be a structural consequence relation over a language of signature Ω. Then the* ***canonical Leibniz congruence*** *is defined to be*

$$\nabla_{\vdash} := \Delta_{\mathfrak{Fm}_{\Omega}(\varnothing)}\,\mathrm{Taut}(\vdash \cap \mathrm{Tm}_{\Omega}(\varnothing)) \tag{4.81}$$

For a proposition φ free of variables, the object $[\varphi]_{\nabla_{\vdash}}$ is called the ***canonical (Leibniz) meaning*** *of φ.*

The reader is asked to check that $\mathrm{Pol}_1(\mathfrak{Tm}_{\Omega}(\varnothing)) = \mathrm{Clo}_1(\mathfrak{Tm}_{\Omega}(\varnothing))$, so that nothing hinged on the assumption made earlier to admit all polynomials for the definition of $\Delta_{\mathfrak{A}}$. We shall briefly comment on the fact that we deal only with constant propositions. From the standpoint of language, propositional variables have no meaning except as placeholders. To ask for the meaning of `(p0∨(¬p11))` in the context of language makes little sense since language is a means of communicating concrete meanings. A variable only stands in for the possible concrete meanings. Thus we end up with a single algebra of meanings, one that even satisfies Leibniz' Principle.

In certain cases the Leibniz operator actually induces an isomorphism between the lattice of deductively closed sets and the lattice of congruences on $\mathfrak{A}$. This means that different theories will actually generate different equalities and different equalities will generate different theories. For example, in boolean logic, a theory corresponds to a deductively closed set in the free algebra of propositions. Moreover, $\langle \varphi, \chi \rangle \in \Delta_{\mathfrak{B}} T$ iff $\varphi \leftrightarrow \chi \in T$. On the left hand side we find the Leibniz congruence generated by T, on the right hand side we find T applied to a complex expression formed from φ and χ. It means in words the following (taking T to be the theory generated by $\varnothing$): two propositions, φ and χ, have the same meaning iff the proposition $\varphi \leftrightarrow \chi$ is a tautology. This does not hold for modal logic; for in modal logic a theory induces a nontrivial consequence only if it is closed under the necessitation rule. (The exactness of the correspondence is guaranteed for boolean logic by the fact that it is Fregean.)

We shall now briefly address the general case of a language as a system of signs. We assume for a start a grammar, with certain modes. The grammar supplies a designated category t of sentences. We may define notions of logic, theory and so on on the level of definite structure terms of category t, since these are unique by construction. This is how Church formulated the simple theory of types, STyp (see next section). A **theory** is now a set of definite structure terms of category t which is closed under consequence. Given a theory T we define the following relation on definite structure terms: $\mathfrak{t} \,\Delta\, \mathfrak{t}'$ iff the two are intersubstitutable in any structure term preserving definiteness, and for a structure term $\mathfrak{s}$: $[\mathfrak{t}/x]\mathfrak{s} \in T$ iff $[\mathfrak{t}'/x]\mathfrak{s} \in T$. Again, this proves to be a congruence on the partial algebra of definite structure terms, and the congruence relation can be factored. What we get is the algebra of natural meanings.

Notes on this section. There have been accounts of propositional attitudes that propose representations of attitude reports that do not contain a representation of the embedded proposition. These accounts have difficulties with Leibniz' Principle. Against this argues [**?**]. For him, the representation of the report contains a representation of the embedded proposition so that Leibniz' Principle does not need to be stipulated. Modal operators actually have that property. However, they do not provide enough analysis of the kinds of attitudes involved. On the other hand, modal operators are very flexible. In general, most attitudes will not give rise to a normal logic, though classicality must be assumed, in virtue of Leibniz' Principle. Also, there is a consensus that a proposition is an expression modulo the laws of PC. However, notice

that this means only that if two expressions are interderivable in PC, we must have the same attitude towards them. It does not say, for example, that if χ follows from φ then if I believe φ I also believe that χ. Classical logics need not be monotone (see the exercises below). For the general theory of modal logic see [**?**].

Exercise 148. Set up a Galois correspondence between contexts and equivalence classes of expressions. You may do this for any category α. Can you characterize those context sets that generate the same equivalence class?

Exercise 149. Show that $\vdash_L$ as defined above is a consequence relation. Show that (mn) is not derivable in $\vDash_{\mathsf{K}}$. *Hint.* You have to find a formula φ such that $\varphi \nvDash_{\mathsf{K}} \Box\varphi$.

Exercise 150. Show that $\vDash^g_{\mathcal{K}}$ is the global consequence relation associated with $\mathsf{Th}\mathcal{K}$.

Exercise 151. Show that the logic of knowledge axiomatized above is $\mathsf{S5}$.

Exercise 152. Let $\langle F, \lhd, \mathbb{F}\rangle$ be a generalized Kripke–frame and β a valuation into it. Put $\mathfrak{M} := \langle \mathbb{F}, \varnothing, F, \cap, \cup, \blacksquare\rangle$. Then β has an obvious homomorphic extension $\overline{\beta}\colon \mathfrak{Tm}_\Omega(V) \to \mathfrak{M}$. Show that $\langle \mathfrak{F}, \beta, x\rangle \vDash \varphi$ iff $x \in \overline{\beta}(\varphi)$.

Exercise 153. Show that there are classical modal logics which are not monotone. *Hint.* There is a counterexample based on a two–element algebra.

Exercise 154. Prove Lemma 4.48.

Exercise 155. Define $\boxtimes\varphi := \varphi \vartriangle \top$. Let L be the set of formulae in $\boxtimes$ and the boolean connectives that are derivable. Show that L is a normal modal logic containing $\mathsf{K4}$.

4. Binding and Quantification

Quantification and binding are one of the most intricate phenomena of formal semantics. Examples of quantifiers we have seen already: the English phrases `every` and `some`, and $\forall$ and $\exists$ of predicate logic. Examples of binding without quantification can be found easily in mathematics. The integral

$$h(\vec{y}) := \int_0^1 f(x, \vec{y})dx \tag{4.82}$$

is a case in point. The integration operator takes a function (which may have parameters) and returns its integral over the interval $[0,1]$. What this has in common with quantification is that the function $h(\vec{y})$ does not depend on x. Likewise, the limit $\lim_{n\to\infty} a_n$ of a convergent series $a\colon \omega \to \mathbb{R}$, is independent of n. (The fact that these operations are not everywhere defined shall not concern us here.) So, as with quantifiers, integration and limits take entities that depend on a variable x and return an entity that is independent of it. The easiest way to analyze this phenomenon is as follows. Given a function f that depends on x, $\lambda x.f$ is a function which is independent of x. Moreover, everything that lies encoded in f is also encoded in $\lambda x.f$. So, unlike quantification and integration, λ–abstraction does not give rise to a loss of information. This is ensured by the identity $(\lambda x.f)x = f$. Moreover, extensionality ensures that abstraction also does not add any information: the abstracted function is essentially nothing more than the graph of the function. λ–abstraction therefore is *the* mechanism of binding. Quantifiers, integrals, limits and so on just take the λ–abstract and return a value. This is exactly how we have introduced the quantifiers: $\exists x.\varphi$ was just an abbreviation of $\Sigma(\lambda x\,.\,\varphi)$. Likewise, the integral can be decomposed into two steps: first, abstraction of a variable and then the actual integration. Notice, how the choice of variable matters:

$$y/3 = \int_0^1 x^2 y dx \neq x/2 = \int_0^1 x^2 y dy \tag{4.83}$$

The notation dx actually does the same as λx: it shows us over which variable we integrate. We may define integration as follows. First, we define an operation $I\colon \mathbb{R}^{\mathbb{R}} \to \mathbb{R}$, which performs the integration over the interval $[0,1]$ of $f \in \mathbb{R}^{\mathbb{R}}$. Then we define

$$\int_0^1 f(x)dx := I(\lambda x.f) \tag{4.84}$$

This definition decouples the definition of the actual integration from the binding process that is involved. In general, any operator $O\langle x_i : i < n\rangle.M$ which binds the variables x_i, $i < n$, and returns a value, can be defined as

$$O\langle x_i : i < n\rangle.M := \widehat{O}(\lambda x_0.\lambda x_1.\cdots.\lambda x_{n-1}.M) \tag{4.85}$$

for a suitable $\widehat{O}$. In fact, since $O\vec{x}.M$ does not depend on $\vec{x}$, we can use (4.85) to define $\widehat{O}$. What this shows is that λ–calculus can be used as a general tool for binding. It also shows that we can to some extent get rid of explicit variables,

Table 11. The Axioms for Predicate Logic (FOL)

AXIOMS.

(a0) — (a12)+

(a13) $(\forall x)(\varphi \to \chi) \to ((\forall x)\varphi \to (\forall x)\chi)$

(a14) $(\forall x)\varphi \to [t/x]\varphi$

(a15) $\varphi \to (\forall x)\varphi \qquad (x \notin \mathrm{fr}(\varphi))$

(a16) $(\forall x)\varphi \to \neg(\exists x)\neg\varphi$

(a17) $\neg(\exists x)\neg\varphi \to (\forall x)\varphi$

(a18) $(\forall x)(x = x)$

(a19) $(\forall x)(\forall y)(x = y \to y = x)$

(a20) $(\forall x)(\forall y)(\forall z)(x = y \wedge y = z \to x = z)$

(a21) $(\forall x_0)\cdots(\forall x_{\Xi(R)-1})(\forall y)((\bigwedge_{i<\Xi(R)} x_i = y_i)$
$\to (R(x_0,\ldots,x_{\Xi(R)-1}) \leftrightarrow [y/x_i]R(x_0,\ldots,x_{\Xi(R)-1})))$

RULES.

(mp) $\dfrac{\varphi \quad \varphi \to \chi}{\chi}$ (gen) $\dfrac{\varphi}{(\forall x)\varphi}$

something that is quite useful for semantics. The elimination of variables removes a point of arbitrariness in the representation that makes meanings nonunique. In this section, we shall introduce two different algebraic calculi. The first is the algebraic approach to predicate logic using so called *cylindric algebras*, the other an equational theory of λ–calculus, which embraces the (marginally popular) variable free approach to semantics for first order logic.

We have already introduced the syntax and semantics of first–order predicate logic. Now we are going to present an axiomatization. To this end we expand the set of axioms for propositional logic by the axioms (a13) – (a21) in Table 11. The calculus (a0) – (a21) with the rules (mp) and (gen) is called FOL. A **first–order theory** is a set of formulae containing (a0) – (a21) and which is closed under (mp). We write $\Delta \vdash^{\mathsf{FOL}} \varphi$ if every theory containing Δ also contains φ. In virtue of (a16) and (a17) we get that $(\forall x)\varphi \leftrightarrow \neg(\exists x)\neg\varphi$ as well as $(\exists x)\varphi \leftrightarrow \neg(\forall x)\neg\varphi$ which means that one of the quantifiers can be defined from the other. In (a21), we assume $i < \Xi(R)$.

We shall prove its completeness using a more powerful result due to Leon Henkin that simple type theory (STyp) is complete with respect to Henkin–frames. Notice that the status of (gen) is the same as that of (mn) in modal logic. (gen) is admissible with respect to the model theoretic consequence $\vDash$

defined in Section 3.8, but it is not derivable in it. To see the first, suppose that φ is a theorem and let x be a variable. Then $(\forall x)\varphi$ is a theorem, too. However, $\varphi \vDash (\forall x)\varphi$ does not follow. Simply take a unary predicate letter P and a structure consisting of two elements, 0, 1, such that P is true of 0 but not of 1. Then with $\beta(x) := 0$ we have $\langle \mathfrak{M}, \beta \rangle \vDash P(x)$ but $\langle \mathfrak{M}, \beta \rangle \nvDash (\forall x)P(x)$. Now let $\underline{P}$ be the set of all formulae that can be obtained from (a0) – (a21) by applying (gen). Then the following holds.

Theorem 4.58 $\vdash^{\mathsf{FOL}} \varphi$ *iff* φ *is derivable from* $\underline{P}$ *using only (mp).*

Proof. Let Π be a proof of χ from $\underline{P}$ using only (mp). Transform the proof in the following way. First, prefix every occurring formula by $(\forall x)$. Further, for every k such that φ_k follows from φ_i and $\varphi_i \to \varphi_j$ for some $i, j < k$, insert in front of the formula $(\forall x)\varphi_j$ the sequence

$$(\forall x)(\varphi_i \to \varphi_j) \to ((\forall x)\varphi_i \to (\forall x)\varphi_j), (\forall x)\varphi_i \to (\forall x)\varphi_j \tag{4.86}$$

The new proof is a proof of $(\forall x)\chi$ from $\underline{P}$, since the latter is closed under (gen). Hence, the set of formulae derivable from $\underline{P}$ using (mp) is closed under (gen). Therefore it contains all tautologies of FOL. □

The next theorem asserts that this axiomatization is complete.

Definition 4.59 *Let* Δ *be a set of formulae of predicate logic over a signature,* φ *a formula over that same signature. Then* $\Delta \vdash \varphi$ *iff* φ *can be proved from* $\Delta \cup \underline{P}$ *using only (mp).*

Theorem 4.60 (Gödel) $\Delta \vdash^{\mathsf{FOL}} \varphi$ *iff* $\Delta \vDash \varphi$.

Recall from Section 3.8 the definition of the simple theory of types. There we have also defined the class of models, the so called *Henkin–frames*. Recall further that this theory has operators Π^α, which allow to define the universal quantifiers in the following way.

$$(\forall x_\alpha)N_t := (\Pi^\alpha(\lambda x_\alpha.N_t)) \tag{4.87}$$

The simple theory of types is axiomatized as follows. We define a calculus exclusively on the terms of type t (truth values). However, it will also be possible to express that two terms are equal. This is done as follows. Two terms M_α and N_α of type α are equal if for every term $O_{\alpha\to t}$ the terms $O_{\alpha\to t}M_\alpha$ and $O_{\alpha\to t}N_\alpha$ are equivalent.

$$M_\alpha \triangleq N_\alpha := (\forall z_{\alpha\to t})(z_{\alpha\to t}M_\alpha \leftrightarrow z_{\alpha\to t}N_\alpha) \tag{4.88}$$

Table 12. The Simple Theory of Types

AXIOMS.

(s0) — (s12) +

(s13) $((\forall x_\alpha)(y_t \to M_{\alpha\to t}x_\alpha)) \to (y_t \to \Pi^\alpha M_{\alpha\to t})$

(s14) $(\Pi^\alpha x_{\alpha\to t}) \to x_{\alpha\to t}y_\alpha$

(s15) $(x_t \leftrightarrow y_t) \to x_t \triangleq y_t$

(s16) $((\forall z_\alpha)(x_{\alpha\to\beta}z_\alpha \triangleq y_{\alpha\to\beta}z_\alpha)) \to (x_{\alpha\to\beta} \triangleq y_{\alpha\to\beta})$

(s17) $x_{\alpha\to t}y_\alpha \to x_{\alpha\to t}(\iota^\alpha x_{\alpha\to t})$

RULES.

(mp) $\dfrac{M_t \to N_t \quad M_t}{N_t}$ (ug) $\dfrac{M_{\alpha\to t}x_\alpha}{\Pi^\alpha M_\alpha}$ $x_\alpha \notin \mathrm{fr}(M_{\alpha\to t})$

(conv) $\dfrac{M_t \quad M_t \equiv_{\alpha\beta} N_t}{N_t}$ (sub) $\dfrac{M_{\alpha\to t}x_\alpha}{M_{\alpha\to t}N_\alpha}$ $x_\alpha \notin \mathrm{fr}(M_{\alpha\to t})$

For this definition we assume that $z_{\alpha\to t}$ is free neither in M_α nor in N_α. If one dislikes the side conditions, one can prevent the accidental capture of $z_{\alpha\to t}$ using the following more refined version:

$$M_\alpha \triangleq N_\alpha := (\lambda x_\alpha.\lambda y_\alpha.(\forall z_{\alpha\to t})(z_{\alpha\to t}x_\alpha \leftrightarrow z_{\alpha\to t}y_\alpha))M_\alpha N_\alpha \tag{4.89}$$

However, if $z_{\alpha\to t}$ is properly chosen, no problem will ever arise. Now, let (s0) – (s12) be the formulae (a0) – (a12) appropriately translated into the language of types. We call the Hilbert–style calculus consisting of (s0) – (s17) and the rules given in Table 12 STyp. All instances of theorems of PC are theorems of STyp. For predicate logic this will also be true, but the proof of that requires work. The rule (gen) is a derived rule of this calculus. To see this, assume that $M_{\alpha\to t}z_\alpha$ is a theorem. Then, by (conv), $(\lambda z_\alpha.M_{\alpha\to t}z_\alpha)z_\alpha$ also is a theorem. Using (ug) we get $\Pi^\alpha(\lambda z_\alpha.M_{\alpha\to t}z_\alpha)$, which by abbreviatory convention is $(\forall z_\alpha)M_{\alpha\to t}$. We will also show that (a14) and (a15) are theorems of STyp.

Lemma 4.61 $\mathsf{STyp} \vdash (\forall x_\alpha)y_t \to [N_\alpha/x_\alpha]y_t$.

Proof. By convention, $(\forall x_\alpha)y_t = \Pi^\alpha(\lambda x_\alpha.y_t)$. Moreover, by (s14), $\mathsf{STyp} \vdash ((\forall x_\alpha)y_t) \to (\lambda x_\alpha.y_t)x_\alpha = ((\forall x_\alpha)y_t) \to y_t$. Using (sub) we get

$$\vdash^{\mathsf{STyp}} [N_\alpha/x_\alpha]((\forall x_\alpha)y_t \to y_t) = (\forall x_\alpha)y_t \to [N_\alpha/x_\alpha]y_t \tag{4.90}$$

as required. □

Lemma 4.62 *Assume that x_α is not free in N_t. Then*

(4.91) $\mathsf{STyp} \vdash N_t \to (\forall x_\alpha) N_t$

Proof. With $N_t \to N_t \equiv_{\alpha\beta} N_t \to (\lambda x_\alpha . N_t) x_\alpha$ and the fact that $(\forall x_\alpha)(N_t \to N_t)$ is derivable (using (gen)), we get with (conv) $(\forall x_\alpha)(N_t \to ((\lambda x_\alpha . N_t) x_\alpha))$ and with (s13) and (mp) we get

(4.92) $\vdash^{\mathsf{STyp}} N_t \to \Pi^\alpha(\lambda x_\alpha . N_t) = N_t \to (\forall x_\alpha) N_t$

(The fact that x_α is not free in N_t is required when using (s13). In order for the replacement of N_t for y_t in the scope of $(\forall x_\alpha)$ to yield exactly N_t again, we need that x_α is not free in N_t.) □

Lemma 4.63 *If* $\lambda\eta \vdash M_\alpha = N_\alpha$ *then* $\mathsf{STyp} \vdash M_\alpha \triangleq N_\alpha$.

Proof. $\lambda\eta = \lambda +$ (ext), and (ext) is the axiom (s16). Hence it remains to show that $\lambda \vdash M_\alpha = N_\alpha$ implies $\vdash^{\mathsf{STyp}} M_\alpha \triangleq N_\alpha$. So, assume the first. Then we have $M_\alpha \equiv_{\alpha\beta} N_\alpha$. Hence

(4.93) $z_{\alpha\to t} M_\alpha \to z_{\alpha\to t} M_\alpha \equiv_{\alpha\beta} z_{\alpha\to t} M_\alpha \to z_{\alpha\to t} N_\alpha$

Hence, using (conv), and $\vdash^{\mathsf{STyp}} z_{\alpha\to t} M_\alpha \to z_{\alpha\to t} M_\alpha$ we get

(4.94) $\vdash^{\mathsf{STyp}} z_{\alpha\to t} M_\alpha \to z_{\alpha\to t} N_\alpha$

By symmetry, $\vdash^{\mathsf{STyp}} z_{\alpha\to t} M_\alpha \leftrightarrow z_{\alpha\to t} N_\alpha$. Using (gen) we get

(4.95) $\vdash^{\mathsf{STyp}} (\forall z_{\alpha\to t})(z_{\alpha\to t} M_\alpha \leftrightarrow z_{\alpha\to t} N_\alpha)$

By abbreviatory convention, $\vdash^{\mathsf{STyp}} M_\alpha \triangleq N_\alpha$. □

We shall now show that STyp is complete with respect to Henkin–frames where $\triangleq$ simply is interpreted as identity. To do that, we first prove that $\triangleq$ is a congruence relation.

Lemma 4.64 *The following formulae are provable in* STyp.

(4.96a) $x_\alpha \triangleq x_\alpha$

(4.96b) $x_\alpha \triangleq y_\alpha \to y_\alpha \triangleq x_\alpha$

(4.96c) $x_\alpha \triangleq y_\alpha \wedge y_\alpha \triangleq z_\alpha \to x_\alpha \triangleq z_\alpha$

(4.96d) $x_{\alpha\to\beta} \triangleq x'_{\alpha\to\beta} \wedge y_\alpha \triangleq y'_\alpha . \to . x_{\alpha\to\beta} y_\alpha \triangleq x'_{\alpha\to\beta} y'_\alpha$

Proof. (4.96a) Let $z_{\alpha\to t}$ be a variable of type $\alpha \to t$. Then $z_{\alpha\to t}x_\alpha \leftrightarrow z_{\alpha\to t}x_\alpha$ is provable in STyp (as $p \leftrightarrow p$ is in PC). Hence, $(\forall z_{\alpha\to t})(z_{\alpha\to t}x_\alpha \leftrightarrow z_{\alpha\to t}x_\alpha)$ is provable, which is $x_\alpha \triangleq x_\alpha$. (4.96b) and (4.96c) are shown using predicate logic. (4.96d) Assume $x_{\alpha\to\beta} \triangleq x'_{\alpha\to\beta}$ and $y_\alpha \triangleq y'_\alpha$. Now,

$$z_{\beta\to t}(x_{\alpha\to\beta}y_\alpha) \equiv_{\alpha\beta} (\lambda u_\alpha.z_{\beta\to t}(x_{\alpha\to\beta}u_\alpha))(y_\alpha) \tag{4.97}$$

Put $M_{\alpha\to t} := (\lambda u_\alpha.z_{\beta\to t}(x_{\alpha\to\beta}u_\alpha))$. Using the rule (conv), we get

$$\begin{aligned} y_\alpha \triangleq y'_\alpha &\vdash^{\mathsf{STyp}} M_\alpha y_\alpha \leftrightarrow M_\alpha y'_\alpha \\ &\vdash^{\mathsf{STyp}} z_{\beta\to t}(x_{\alpha\to\beta}y_\alpha) \leftrightarrow z_{\beta\to t}(x_{\alpha\to\beta}y'_\alpha) \end{aligned} \tag{4.98}$$

Likewise, one can show that

$$y_\alpha \triangleq y'_\alpha \vdash^{\mathsf{STyp}} z_{\beta\to t}(x'_{\alpha\to\beta}y_\alpha) \leftrightarrow z_{\beta\to t}(x'_{\alpha\to\beta}y'_\alpha) \tag{4.99}$$

Similarly, using $N_{(\alpha\to\beta)\to t} := (\lambda u_{\alpha\to\beta}.z_{\beta\to t}(u_{\alpha\to\beta}y_\alpha))$ one shows that

$$x_{\alpha\to\beta} \triangleq x'_{\alpha\to\beta} \vdash^{\mathsf{STyp}} z_{\beta\to t}(x_{\alpha\to\beta}y_\alpha) \leftrightarrow z_{\beta\to t}(x'_{\alpha\to\beta}y_\alpha) \tag{4.100}$$

This allows to derive the desired conclusion. □

Now we get to the construction of the frame. Let C_α be the set of closed formulae of type α. Choose a maximally consistent $\Delta \supseteq C_t$. Then, for each type α, define $\approx^\alpha_\Delta$ by $M_\alpha \approx^\alpha_\Delta N_\alpha$ iff $M_\alpha \triangleq N_\alpha \in \Delta$. By Lemma 4.64 this is an equivalence relation for each α, and, moreover, if $M_{\alpha\to\beta} \approx^{\alpha\to\beta}_\Delta M'_{\alpha\to\beta}$ and $N_\alpha \approx^\alpha_\Delta N'_\alpha$, then also $M_{\alpha\to\beta}N_\alpha \approx^\beta_\Delta M'_{\alpha\to\beta}N'_\alpha$. For $M_\alpha \in C_\alpha$ put

$$[M_\alpha] := \{N_\alpha : M_\alpha \approx^\alpha_\Delta N_\alpha\} \tag{4.101}$$

Finally, put $D_\alpha := \{[M_\alpha] : M_\alpha \in C_\alpha\}$. Next, $\bullet$ is defined as usual, $- := [\neg]$, $\cap := [\wedge]$, $\pi^\alpha := [\Pi^\alpha]$ and $\iota^\alpha := [\iota^\alpha]$. This defines a structure (where $\mathcal{S} := \mathrm{Typ}_\to(B)$).

$$\mathfrak{Hfr}_\Delta := \langle \{D_\alpha : \alpha \in \mathcal{S}\}, \bullet, -, \cap, \langle \pi^\alpha : \alpha \in \mathcal{S}\}\rangle, \langle \iota^\alpha : \alpha \in \mathcal{S}\}\rangle\rangle \tag{4.102}$$

Lemma 4.65 (Witnessing Lemma)

$$\mathsf{STyp} \vdash^{\mathsf{STyp}} M_{\alpha\to t}(\iota^\alpha(\lambda x_\alpha.\neg M_\alpha)) \to \Pi^\alpha M_\alpha$$

Proof. Write $\neg N_{\alpha\to t} := \lambda x_\alpha.\neg(N_{\alpha\to t}x_\alpha)$. Now, by (s17)

(4.103) $\quad \vdash^{\mathsf{STyp}} (\neg N_{\alpha\to t})y_\alpha \to (\neg N_{\alpha\to t})(\iota^\alpha(\neg N_{\alpha\to t}))$

Using classical logic we obtain

(4.104) $\quad \vdash^{\mathsf{STyp}} \neg((\neg N_{\alpha\to t})(\iota^\alpha(\neg N_{\alpha\to t}))) \to \neg((\neg N_{\alpha\to t})y_\alpha)$

Now, $\neg((\neg N_{\alpha\to t})y_t) \rhd_\beta \neg\neg(N_{\alpha\to t}y_\alpha)$, the latter being equivalent to $N_{\alpha\to t}y_\alpha$. Similarly, $\neg((\neg N_{\alpha\to t})(\iota^\alpha(\neg N_{\alpha\to t})))$ is equivalent with $N_{\alpha\to t}(\iota^\alpha(\neg N_{\alpha\to t}))$. Hence

(4.105) $\quad \vdash^{\mathsf{STyp}} N_{\alpha\to t}(\iota^\alpha(\neg N_{\alpha\to t})) \to N_{\alpha\to t}y_\alpha$

Using (gen), (s13) and (mp) we get

(4.106) $\quad \vdash^{\mathsf{STyp}} (N_{\alpha\to t}(\iota^\alpha(\neg N_{\alpha\to t}))) \to \Pi^\alpha N_{\alpha\to t}$

This we had to show. □

Lemma 4.66 *$\mathfrak{Hrf}_\Delta$ is a Henkin–frame.*

Proof. By Lemma 4.63, if $\lambda\eta \vdash M_\alpha \triangleq N_\alpha$, then $[M_\alpha] = [N_\alpha]$. So, the axioms of the theory $\lambda\eta$ are valid, and $\langle\{D_\alpha : \alpha \in S\}, \bullet\rangle$ is a functionally complete (typed) applicative structure. Since Δ is maximally consistent, D_t consists of two elements, which we now call 0 and 1. Furthermore, we may arrange it that $[M_t] = 1$ iff $M_t \in \Delta$. It is then easily checked that the interpretation of $\neg$ is complement, and the interpretation of $\wedge$ is intersection. Now we treat $\pi^\alpha := [\Pi^\alpha]$. We have to show that for $a \in D_{\alpha\to t}$ $\pi^\alpha \bullet a = 1$ iff for every $b \in D_\alpha$: $a \bullet b = 1$. Or, alternatively, $\Pi^\alpha M_{\alpha\to t} \in \Delta$ iff $M_{\alpha\to t}N_\alpha \in \Delta$ for every closed term N_α. Suppose that $\Pi^\alpha M_{\alpha\to t} \in \Delta$. Using Lemma 4.61 and the fact that Δ is deductively closed, $M_{\alpha\to t}N_\alpha \in \Delta$. Conversely, assume $M_{\alpha\to t}N_\alpha \in \Delta$ for every constant term N_α. Then $M_\alpha(\iota^\alpha(\lambda x_\alpha.\neg M_{\alpha\to t}x_\alpha))$ is a constant term, and it is in Δ. Moreover, by the Witnessing Lemma, $\Pi^\alpha M_{\alpha\to t} \in \Delta$. Finally, we have to show that for every $a \in D_{\alpha\to t}$: if there is a $b \in D_\alpha$ such that $a \bullet b = 1$ then $a \bullet (\iota^\alpha \bullet a) = 1$. This means that for $M_{\alpha\to t}$: if there is a constant term N_α such that $M_{\alpha\to t}N_\alpha \in \Delta$ then $M_{\alpha\to t}(\iota^\alpha M_{\alpha\to t}) \in \Delta$. This is a consequence of (s17). □

Now, it follows that $\mathfrak{Hfr}_\Delta \vDash N_t$ iff $N_t \in \Delta$. More generally, let β an assignment of constant terms to variables. Let M_α be a term. Write M_α^β for the result of replacing a free occurrence of a variable x_γ by $\beta(x_\gamma)$. Then

(4.107) $\quad \langle\mathfrak{Hfr}_\Delta, \beta\rangle \vDash M_\alpha \quad \Leftrightarrow \quad M_\alpha^\beta \in \Delta$

This is shown by induction.

Lemma 4.67 *Let Δ_0 be a consistent set of constant terms. Then there exists a maximally consistent set Δ of constant terms containing Δ_0.*

Proof. Choose a well–ordering $\{N^\delta : \delta < \mu\}$ on the set of constant terms. Define Δ_i by induction as follows. $\Delta_{\kappa+1} := \Delta_\kappa \cup \{N^\delta\}$ if the latter is consistent. Otherwise, $\Delta_{\kappa+1} := \Delta_\kappa$. If $\kappa < \mu$ is a limit ordinal, $\Delta_\kappa := \bigcup_{\lambda<\kappa} \Delta_\lambda$. We shall show that $\Delta := \Delta_\mu$ is maximally consistent. Since it contains Δ_0, this will complete the proof. (a) It is consistent. This is shown inductively. By assumption Δ_0 is consistent, and if Δ_κ is consistent, then so is $\Delta_{\kappa+1}$. Finally, let λ be a limit ordinal and suppose that Δ_λ is inconsistent. Then there is a finite subset Γ which is inconsistent. There exists an ordinal $\kappa < \lambda$ such that $\Gamma \subseteq \Delta_\kappa$. Δ_κ is consistent, contradiction. (b) There is no consistent superset. Assume that there is a term $M \notin \Delta$ such that $\Delta \cup \{M\}$ is consistent. Then for some δ, $M = N^\delta$. Then $\Delta_\delta \cup \{N^\delta\}$ is consistent, whence by definition $N^\delta \in \Delta_{\delta+1}$. Contradiction. □

Theorem 4.68 (Henkin) *(a) A term N_t is a theorem of* STyp *iff it is valid in all Henkin–frames. (b) An equation $M_\alpha \triangleq N_\alpha$ is a theorem of* STyp *iff it holds in all Henkin–frames iff it is valid in the many sorted sense.*

We sketch how to prove Theorem 4.60. Let $\langle \Omega, \Xi \rangle$ be a signature for predicate logic. Define a translation into STyp:

(4.108a) $\quad f^\heartsuit := (\lambda x_0)(\lambda x_1)\dots(\lambda x_{\Omega(f)-1}) f(x_0, \dots, x_{\Omega(f)-1})$

(4.108b) $\quad r^\heartsuit := (\lambda x_0)(\lambda x_1)\dots(\lambda x_{\Xi(r)-1}) r(x_0, \dots, x_{\Xi(r)-1})$

This is extended to all formulae. Now we look at the signature for STyp with the constants $f^\heartsuit$, $r^\heartsuit$ of type

(4.109a) $\quad f^\heartsuit : (e \to (e \to \dots (e \to e) \dots))$

(4.109b) $\quad r^\heartsuit : (e \to (e \to \dots (e \to t) \dots))$

Now, given a first–order model $\mathfrak{M}$, we can construct a Henkin–frame for $\mathfrak{M}^\heartsuit$ with $D_e = M$ and $D_{\alpha\to\beta} := D_\alpha \to D_\beta$, by interpreting $f^\heartsuit$ and $r^\heartsuit$ as given by (4.108a) and (4.108b).

Lemma 4.69 *Let β be a valuation on $\mathfrak{M}$. Extend β to $\beta^\heartsuit$. Then*

(4.110) $\quad \langle \mathfrak{M}, \beta \rangle \vDash \varphi \quad \Leftrightarrow \quad \langle \mathfrak{M}^\heartsuit, \beta \rangle \vDash \varphi^\heartsuit$

Lemma 4.70 $\vdash^{\mathsf{STyp}} \varphi^{\heartsuit}$ *iff* $\vdash^{\mathsf{FOL}} \varphi$.

Right to left is by induction. Now if $\nvdash^{\mathsf{FOL}} \varphi$ then there is a model $\langle \mathfrak{M}, \beta \rangle \vDash \neg\varphi$, from which we get a Henkin–frame $\langle \mathfrak{M}^{\heartsuit}, \beta \rangle \nvDash \neg\varphi^{\heartsuit}$. The proof of Theorem 4.60 is as follows. Clearly, if φ is derivable it is valid. Suppose that it is not derivable. Then $\varphi^{\heartsuit}$ is not derivable in STyp. There is a Henkin–frame $\mathfrak{M} \nvDash \varphi^{\heartsuit}$. This allows to define a first–order model $\mathfrak{M}_{\heartsuit} \nvDash \varphi$.

Exercise 156. The set of typed λ–terms is defined over a finite alphabet if the set B of basic types is finite. Define from this a well–ordering on the set of terms. *Remark.* This shows that the proof of Lemma 4.67 does not require the use of the Axiom of Choice for obtaining the well–ordering.

Exercise 157. Show Lemma 4.69.

Exercise 158. Complete the details of the proof of Theorem 4.60.

Exercise 159. Let L be a normal modal logic. Show that $\Delta \Vdash_L \chi$ iff $\Delta^b \vdash_L \chi$, where $\Delta^b := \{\Box^n \delta : \delta \in \Delta, n \in \omega\}$. *Hint.* This is analogous to Theorem 4.58.

5. Algebraization

Now that we have shown completeness with respect to models and frames, we shall proceed to investigate the possibility of algebraization of predicate logic and simple type theory. Apart from methodological reasons, there are also practical reasons for preferring algebraic models over frames. If φ is a sentence and $\mathfrak{M}$ a model, then either $\mathfrak{M} \vDash \varphi$ or $\mathfrak{M} \vDash \neg\varphi$. Hence, the theory of a single model is maximally consistent, that is, complete. One may argue that this is as it should be; but notice that the base logic (FOL, STyp) is not complete — neither is the knowledge ordinary people have. Since models are not enough for representing incomplete theories, something else must step in their place. These are *algebras* for some appropriate signature, for the product of algebras is an algebra again, and the logic of the product is the intersection of the logics of the factors. Hence, for every logic there is an adequate algebra. However, algebraization is not straightforward. The problem is that there is no notion of binding in algebraic logic. Substitution always is replacement of an occurrence of a variable by the named string, there is never a preparatory replacement of variables being performed. Hence, what creates in fact big problems is those axioms and rules that employ the notion of a free or bound

variable. In predicate logic this is the axiom (a15). (Unlike PC, FOL has no rule of substitution.)

It was once again Tarski who first noticed the analogy between modal operators and quantifiers. Consider a language L of first order logic without quantifiers. We may interpret the atomic formulae of this language as propositional atoms, and formulae made from them using the boolean connectives. Then we have a somewhat more articulate version of our propositional boolean language. We can now introduce a quantifier Q simply as a unary operator. For example, $(\forall \mathtt{x0})$ is a unary operator on formulae. Given a formula φ, $(\forall \mathtt{x0})\varphi$ is again a formula. (Notice that the way we write the formulae is somewhat different, but this can easily be accounted for.) In this way we get an extended language: a language of formulae extended by a single quantifier. Moreover, the laws of $(\forall \mathtt{x0})$ turn the logic exactly into a normal modal logic. The quantifier $(\exists \mathtt{x0})$ then corresponds to $\Diamond$, the dual of $\Box$. Clearly, in order to reach full expressive power of predicate logic we need to add infinitely many such operators, one for each variable. The resulting algebras are called **cylindric algebras**. The principal reference is to [**?**].

We start with the intended models of cylindric algebras. A formula may be seen as a function from models, that is, pairs $\langle \mathfrak{M}, \beta \rangle$, to 2, where $\mathfrak{M}$ is a structure and β an assignment of values to the variables. First of all, we shall remove the dependency on the structure, which allows us to focus on the assignments. There is a general first order model for any complete (= maximal consistent) theory, in which exactly those sentences are valid that belong to the theory. Moreover, this model is countable. Suppose a theory T is not complete. Then let Δ_i, $i \in I$, be its completions. For each $i \in I$, let $\mathfrak{M}_i$ be the canonical structure associated with Δ_i. If $\mathfrak{A}_i$ is the cylindrical algebra associated with $\mathfrak{M}_i$ (to be defined below), the algebra associated with T will $\prod_{i \in I} \mathfrak{A}_i$. In this way, we may reduce the study to that of a cylindric algebra of a single structure.

Take a first order structure $\langle M, \mathfrak{I} \rangle$, where M is the universe and $\mathfrak{I}$ the interpretation function. For simplicity, we assume that there are no functions. (The reader shall see in the exercises that there is no loss of expressivity in renouncing functions.) Let $V := \{x_i : i \in \omega\}$ be the set of variables. Let $\mathfrak{V}(V; M)$ be the boolean algebra of sets of functions into M. Then for every formula φ we associate the following set of assignments:

(4.111) $\quad [\varphi] := \{\beta : \langle \mathfrak{M}, \beta \rangle \vDash \varphi\}$

Now, for each number i we assume the following operation A_i.

(4.112) $\mathsf{A}_i(S) := \{\beta : \text{ for all } \gamma \sim_{x_i} \beta : \gamma \in S\}$

Then $\mathsf{E}_i(S) := -\mathsf{A}_i(-S)$. (The standard notation for E_i is c_i. The letter c here is suggestive for 'cylindrification'. We have decided to stay with a more logical notation.) Furthermore, for every pair of numbers $i, j \in \omega$ we assume the element $\mathsf{d}_{i,j}$.

(4.113) $\mathsf{d}_{i,j} := \{\beta : \beta(x_i) = \beta(x_j)\}$

It is interesting to note that with the help of these elements substitution can be defined. Namely, put

(4.114) $$\mathsf{s}^i_j(x) := \begin{cases} x & \text{if } i = j, \\ \mathsf{E}_i(\mathsf{d}_{i,j} \cap x) & \text{otherwise.} \end{cases}$$

Lemma 4.71 *Let y be a variable distinct from x. Then $[y/x]\varphi$ is equivalent with* $(\exists x).((y{=}x) \wedge \varphi)$.

Thus, equality and quantification alone can define substitution. The relevance of this observation for semantics has been nicely explained in [**?**]. For example, in applications it becomes necessary to introduce constants for the relational symbols. Suppose, namely that `taller` is a binary relation symbol. Its interpretation is a binary relation on the domain. If we want to replace the structure by its associated cylindric algebra, the relation is replaced by an element of that algebra, namely

(4.115) $[\mathsf{taller}'(x_0, x_1)] := \{\beta : \langle \beta(x_0), \beta(x_1) \rangle \in \mathfrak{I}(\texttt{taller})\}$

However, this allows us prima facie only to assess the meaning of 'x_0 is taller than x_1'. We do not know, for example, what happens to 'x_2 is taller than x_7'. For that we need the substitution functions. Now that we have the unary substitution functions, any finitary substitution becomes definable. In this particular case,

(4.116) $[\mathsf{taller}'(x_2, x_7)] = \mathsf{s}^7_1 \mathsf{s}^2_0 [\mathsf{taller}'(x_0, x_1)]$

Thus, given the definability of substitutions, to define the interpretation of R we only need to give the element $[R(x_0, x_1, \ldots, x_{\Xi(R)-1})]$.

The advantage in using this formulation of predicate logic is that it can be axiomatized using equations. It is directly verified that the equations listed in the next definition are valid in the intended structures.

Definition 4.72 *A **cylindric algebra** of dimension κ, κ a cardinal number, is a structure*

$$\mathfrak{A} = \langle A, 0, 1, -, \cap, \cup, \langle \mathsf{E}_\lambda : \lambda < \kappa \rangle, \langle \mathsf{d}_{\lambda,\mu} : \lambda, \mu < \kappa \rangle \rangle \tag{4.117}$$

such that the following holds for all $x, y \in A$ and $\lambda, \mu, \nu < \kappa$:

$$\begin{array}{ll} \text{(ca1)} & \langle A, 0, 1, -, \cap, \cup \rangle \text{ is a boolean algebra.} \\ \text{(ca2)} & \mathsf{E}_\lambda 0 = 0. \\ \text{(ca3)} & x \cup \mathsf{E}_\lambda x = \mathsf{E}_\lambda x. \\ \text{(ca4)} & \mathsf{E}_\lambda (x \cap \mathsf{E}_\lambda y) = \mathsf{E}_\lambda x \cap \mathsf{E}_\lambda y. \\ \text{(ca5)} & \mathsf{E}_\lambda \mathsf{E}_\mu x = \mathsf{E}_\mu \mathsf{E}_\lambda x. \\ \text{(ca6)} & \mathsf{d}_{\lambda,\mu} = 1. \\ \text{(ca7)} & \textit{If } \lambda \neq \mu, \nu \textit{ then } \mathsf{d}_{\mu,\nu} = \mathsf{E}_\lambda (\mathsf{d}_{\mu,\lambda} \cap \mathsf{d}_{\lambda,\nu}). \\ \text{(ca8)} & \textit{If } \lambda \neq \mu \textit{ then } \mathsf{E}_\lambda (\mathsf{d}_{\lambda,\mu} \cap x) \cap \mathsf{E}_\lambda (\mathsf{d}_{\lambda,\mu} \cap (-x)) = 0. \end{array} \tag{4.118}$$

We shall see that this definition allows to capture the effect of the axioms above, with the exception of (a15). Notice first the following. $\leftrightarrow$ is a congruence in FOL as well. For if $\varphi \leftrightarrow \chi$ is a tautology then so is $(\exists x_i)\varphi \leftrightarrow (\exists x_i)\chi$. Hence, we can encode the axioms of FOL as equations of the form $\varphi \leftrightarrow \top$ as long as no side condition concerning free or bound occurrences is present. We shall not go into the details. For example, in $\varphi = (\forall x)\chi$ x occurs trivially bound. It remains to treat the rule (gen). It corresponds to the rule (mn) of modal logic. In equational logic, it is implicit anyway. For if $x = y$ then $O(x) = O(y)$ for any unary operator O.

Definition 4.73 *Let $\mathfrak{A}$ be a cylindric algebra of dimension κ, and $a \in A$. Then*

$$\Delta a := \{ i : i < \kappa, \mathsf{E}_i a \neq a \} \tag{4.119}$$

*is called the **dimension of** a. $\mathfrak{A}$ is said to be **locally finite dimensional** if $|\Delta a| < \aleph_0$ for all $a \in A$.*

A particular example of a cylindric algebra is $\mathcal{L}_\kappa / \equiv$, $\mathcal{L}_\kappa$ the formulae of pure equality based on the variables x_i, $i < \kappa$, and $\varphi \equiv \chi$ iff $\varphi \leftrightarrow \chi$ is a theorem. (If additional function or relation symbols are needed, they can be added with little change to the theory.) This algebra is locally finite dimensional and is freely κ–generated.

The second approach we are going to elaborate is one which takes substitutions as basic functions. For predicate logic this has been proposed by

Halmos [**?**], but most people credit Quine [**?**] for this idea. For an exposition see [**?**]. Basically, Halmos takes substitution as primitive. This has certain advantages that will become apparent soon. Let us agree that the index set is κ, again called the **dimension**. Halmos defines operations $\mathsf{S}(\tau)$ for every function $\tau\colon \kappa \to \kappa$ such that there are only finitely many i such that $\tau(i) \neq i$. The theory of such functions is axiomatized independently of quantification. Now, for every finite set $I \subset \kappa$ Halmos admits an operator $\mathsf{E}(I)$, which represents quantification over each of variables x_i, where $i \in I$. If $I = \varnothing$, $\mathsf{E}(I)$ is the identity, otherwise $\mathsf{E}(I)(\mathsf{E}(K)x) = \mathsf{E}(I \cup K)x$. Thus, it is immediately clear that the ordinary quantifiers $\mathsf{E}(\{i\})$ suffice to generate all the others. However, the axiomatization is somewhat easier with the polyadic quantifiers. Another problem, noted in [**?**], is the fact that the axioms for polyadic algebras cannot be schematized using letters for elements of the index set. However, Sain and Thompson [**?**] also note that the addition of transpositions is actually enough to generate the same functions. To see this, here are some definitions.

Definition 4.74 *Let I be a set and $\pi\colon I \to I$. The **support** of π, $\mathrm{supp}(\pi)$, is the set $\{i : \pi(i) \neq i\}$. A function of finite support is called a **transformation**. π is called a **permutation of** I if it is bijective. If the support contains exactly two elements, π is called a **transposition**.*

The functions whose support has at most two elements are of special interest. Notice first the case when $\mathrm{supp}(\pi)$ has exactly one element. In that case, π is called an **elementary substitution**. Then there are $i, j \in I$ such that $\pi(i) = j$ and $\pi(k) = k$ if $k \neq i$. If i and j are in I, then denote by (i, j) the permutation that sends i to j and j to i. Denote by $[i, j]$ the elementary substitution that sends i to j.

Proposition 4.75 *Let I be a set. The set $\Phi(I)$ of functions $\pi\colon I \to I$ of finite support is closed under concatenation. Moreover, $\Phi(I)$ is generated by the elementary substitutions and the transpositions.*

The proof of this theorem is left to the reader. So, it is enough if we take only functions corresponding to $[i, j]$ and (i, j). The functions of the first kind are already known: these are the s_i^j. For the functions of the second kind, write $\mathsf{p}_{i,j}$. Sain and Thompson effectively axiomatize cylindric algebras that have these additional operations. They call them **finitary polyadic algebras**. Notice also the following useful fact, which we also leave as an exercise.

Proposition 4.76 *Let $\pi\colon I \to I$ be an arbitrary function, and $M \subseteq I$ finite. Then there is a product γ of elementary substitutions such that $\gamma \restriction M = \pi \restriction M$.*

This theorem is both stronger and weaker than the previous one. It is stronger because it does not assume π to have finite support. On the other hand, γ only approximates π on a given finite set. (The reader may take notice of the fact that there is no sequence of elementary substitutions that equals the transformation $(0\ 1)$ on ω. However, we can approximate it on any finite subset.)

Rather than developing this in detail for predicate logic we shall do it for the typed λ–calculus, as the latter is more rich and allows to encode arbitrarily complex abstraction (for example, by way of using STyp). Before we embark on the project let us outline the problems that we have to deal with. Evidently, we wish to provide an algebraic axiomatization that is equivalent to the rules (3.95a) – (3.95g) and (3.95i). First, the signature we shall choose has function application and abstraction as its primitives. However, we cannot have a single abstraction symbol corresponding to λ, rather, for each variable (and each type) we must assume a different unary function symbol λ_i, corresponding to $\lambda \mathsf{x}_i$. Now, (3.95a) – (3.95e) and (3.95i) are already built into the Birkhoff–Calculus. Hence, our only concern are the rules of conversion. These are, however, quite tricky. Notice first that the equations make use of the substitution operation $[N/x]$. This operation is in turn defined with the definitions (3.93a) – (3.93f). Already (3.93a) for $N = \mathsf{x}_i$ can only be written down if we have an operation that performs an elementary substitution. So, we have to add the unary functions $\mathsf{s}_{i,j}$, to denote this substitution. Additionally, (3.93a) needs to be broken down into an inductive definition. To make this work, we need to add correlates of the variables. That is, we add zeroary function symbols x_i for every $i \in \omega$. Symbols for the functions $\mathsf{p}_{i,j}$ permuting i and j will also be added to be able to say that the variables all range over the same set. Unfortunately, this is not all. Notice that (3.95f) is not simply an equation: it has a side condition, namely that y is not free in M. In order to turn this into an equation we must introduce *sorts*, which will help us keep track of the free variables. Every term will end up having a unique sort, which will be the set of i such that x_i is free in it. B is the set of basic types. Call a member of $\mathcal{I} := \mathrm{Typ}_{\rightarrow}(B) \times \omega$ an **index**. If $\langle \alpha, i \rangle$ is an index, α is its **type** and i its numeral. Let $\mathcal{F}$ be the set of pairs $\langle \alpha, \delta \rangle$ where α is a type and δ a finite set of indices.

We now start with the signature. Let δ and δ' be finite sets of indices, α, β types, and $\iota = \langle \gamma, i \rangle$, $\kappa = \langle \gamma', i' \rangle$ indices. We list the symbols together with

their type:

(4.120a) $\mathtt{x}_\iota : \langle\langle\gamma, \{\iota\}\rangle\rangle$

(4.120b) $\lambda_\iota^{\langle\beta,\delta\rangle} : \langle\langle\beta,\delta\rangle, \langle\gamma\to\beta, \delta - \{\iota\}\rangle\rangle$

(4.120c) $\mathtt{p}_{\iota,\kappa}^{\langle\alpha,\delta\rangle} : \langle\langle\alpha,\delta\rangle, \langle\alpha, (\iota,\kappa)[\delta]\rangle\rangle$ where $\gamma = \gamma'$

(4.120d) $\mathtt{s}_{\iota,\kappa}^{\langle\alpha,\delta\rangle} : \langle\langle\alpha,\delta\rangle, \langle\alpha, [\iota,\kappa][\delta]\rangle\rangle$

(4.120e) $\bullet^{\langle\alpha\to\beta,\delta\rangle,\langle\alpha,\delta'\rangle} : \langle\langle\alpha\to\beta,\delta\rangle, \langle\alpha,\delta'\rangle, \langle\beta,\delta\cup\delta'\rangle\rangle$

Here $(\iota,\kappa)[\delta]$ is the result of exchanging ι and κ in δ and $[\iota,\kappa][\delta]$ is the result of replacing κ by ι in δ. Notice that $\mathtt{x}_\iota$ is now a constant! We may also have additional functional symbols stemming from an underlying (sorted) algebraic signature. The reader is asked to verify that nothing is lost if we assume that additional function symbols only have arity 0, and signature $\langle\langle\alpha,\varnothing\rangle\rangle$ for a suitable α. This greatly simplifies the presentation of the axioms.

This defines the language. Notice that in addition to the constants $\mathtt{x}_\iota$ we also have variables x_i^σ for each sort σ. The former represent the variable $\mathtt{x}_i$ (where $\iota = \langle\gamma, i\rangle$) of the λ–calculus and the latter range over terms of sort σ. Now, in order to keep the notation perspicuous we shall drop the sorts whenever possible. That this is possible is assured by the following fact. If t is a term without variables, and we hide all the sorts except for those of the variables, still we can recover the sort of the term uniquely. For the types this is clear, for the second component we observe the following.

Lemma 4.77 *If a term t has sort $\langle\alpha,\delta\rangle$ then* $\mathrm{fr}(t) = \{\mathtt{x}_\iota : \iota \in \delta\}$.

The proof of this fact is an easy induction.

For the presentation of the equations we therefore omit the sorts. They have in fact only been introduced to ensure that we may talk about the set of free variables of a term. The equations (vb1) — (vb6) of Table 13 characterize the behaviour of the substitution and permutation function with respect to the indices. We assume that ι, μ, ν all have the same type. (We are dropping the superscripts indicating the sort.) The equations (vb7) – (vb11) characterize the pure binding by the unary operators λ_ι. The set of equations is invariant under permutation of the indices. Moreover, we can derive the invariance under replacement of bound variables, for example. Thus, effectively, once the interpretation of $\lambda_{\langle\alpha,0\rangle}$ is known, the interpretation of all $\lambda_{\langle\alpha,i\rangle}$, $i \in \omega$, is known as well. For using the equations we can derive that $\mathtt{p}_{\langle\alpha,i\rangle,\langle\alpha,0\rangle}$ is the

Table 13. The Variable Binding Calculus VB

$$\text{(vb1)} \quad \mathtt{s}_{\iota,\mu}\mathtt{x}_\nu = \begin{cases} \mathtt{x}_\iota & \text{if } \nu \in \{\iota,\mu\}, \\ \mathtt{x}_\nu & \text{otherwise.} \end{cases}$$

$$\text{(vb2)} \quad \mathtt{p}_{\iota,\mu}\mathtt{x}_\nu = \begin{cases} \mathtt{x}_\iota & \text{if } \nu = \mu, \\ \mathtt{x}_\mu & \text{if } \nu = \iota, \\ \mathtt{x}_\nu & \text{otherwise.} \end{cases}$$

$$\begin{array}{lll}
\text{(vb3)} & \mathtt{p}_{\iota,\iota}x & = x \\
\text{(vb4)} & \mathtt{p}_{\iota,\mu}x & = \mathtt{p}_{\mu,\iota}x \\
\text{(vb5)} & \mathtt{p}_{\iota,\mu}\mathtt{p}_{\mu,\nu}x & = \mathtt{p}_{\mu,\nu}\mathtt{p}_{\iota,\mu}x \qquad \text{if } |\{\iota,\mu,\nu\}| = 3 \\
\text{(vb6)} & \mathtt{p}_{\iota,\mu}\mathtt{s}_{\mu,\iota}x & = \mathtt{s}_{\iota,\mu}x \\
\text{(vb7)} & \mathtt{s}_{\iota,\mu}\lambda_\nu x & = \lambda_\nu \mathtt{s}_{\iota,\mu}x \qquad \text{if } |\{\iota,\mu,\nu\}| = 3 \\
\text{(vb8)} & \mathtt{s}_{\mu,\iota}\lambda_\iota x & = \lambda_\iota x \\
\text{(vb9)} & \mathtt{p}_{\iota,\mu}\lambda_\nu x & = \lambda_\nu \mathtt{p}_{\iota,\mu}x \qquad \text{if } |\{\iota,\mu,\nu\}| = 3 \\
\text{(vb10)} & \mathtt{p}_{\mu,\iota}\lambda_\mu x & = \lambda_\iota \mathtt{p}_{\mu,\iota}x \\
\text{(vb11)} & \lambda_\iota(y \bullet \mathtt{x}_\iota) & = y \\
\text{(vb12)} & (\lambda_\iota(x \bullet y)) \bullet z & = ((\lambda_\iota x) \bullet z) \bullet ((\lambda_\iota y) \bullet z)
\end{array}$$

$$\text{(vb13)} \quad (\lambda_\iota\lambda_\mu x) \bullet y = \begin{cases} \lambda_\mu((\lambda_\iota x) \bullet y) & \text{if } \iota \neq \mu, \mathtt{x}_\mu \notin \mathrm{fr}(y) \\ \lambda_\iota x & \text{if } \iota = \mu \end{cases}$$

$$\text{(vb14)} \quad (\lambda_\iota \mathtt{x}_\mu) \bullet y = \begin{cases} y & \text{if } \iota = \mu \\ \mathtt{x}_\mu & \text{if } \iota \neq \mu \end{cases}$$

inverse of $\mathtt{p}_{\langle\alpha,0\rangle,\langle\alpha,i\rangle}$, and so

$$\text{(4.121)} \qquad \lambda_{\langle\alpha,i\rangle}x = \mathtt{p}_{\langle\alpha,0\rangle,\langle\alpha,i\rangle}\lambda_{\langle\alpha,0\rangle}\mathtt{p}_{\langle\alpha,i\rangle,\langle\alpha,0\rangle}x$$

The equivalent of (3.95f) now turns out to be derivable. However, we still need to take care of (3.95g). Since we do not dispose of the full substitution $[N/x]$, we need to break down (3.95g) into an inductive definition (vb12) — (vb14). The condition $\mathtt{x}_\mu \notin \mathrm{fr}(y)$ is just a shorthand; all it says is that we take only those equations where the term y has sort $\langle\alpha,\delta\rangle$ and $\mu \notin \delta$. Notice that the disjunction in (vb13) is not complete. From these equations we deduce

that $\mathtt{x}_{\langle\alpha,k\rangle} = \mathtt{p}_{\langle\alpha,k\rangle,\langle\alpha,0\rangle}\mathtt{x}_{\langle\alpha,0\rangle}$, so we could in principle dispense with all but one variable symbol for each type.

The theory of sorted algebras now provides us with a class of models which is characteristic for that theory. We shall not spell out a proof that these models are equivalent to models of the λ–calculus in a sense made to be precise. Rather, we shall outline a procedure that turns an Ω–algebra into a model of the above equations. Start with a signature $\langle F,\Omega\rangle$, sorted or unsorted. For ease of presentation let it be sorted. Then the set B of basic types is the set of sorts. Let Eq_Ω be the equational theory of the functions from the signature alone. For complex types, put $A_{\alpha\to\beta} := A_\alpha \to A_\beta$. Now transform the original signature into a new signature $\Omega' = \langle F,\Omega'\rangle$ where $\Omega'(f) = 0$ for all $f \in F$. Namely, for $f\colon \prod_{i<n} A_{\sigma_i} \to A_\tau$ set

$$f^\star := \lambda x_{\langle\sigma_{n-1},n-1\rangle}.\cdots.\lambda x_{\langle\sigma_0,0\rangle}.f^{\mathfrak{A}}(x_{\langle\sigma_{n-1},n-1\rangle},\ldots,x_{\langle\sigma_0,0\rangle}) \tag{4.122}$$

This is an element of A_π where

$$\pi := (\sigma_0 \to (\sigma_1 \to \cdots(\sigma_{n-1} \to \tau)\cdots)) \tag{4.123}$$

We blow up the types in the way described above. This describes the transition from the signature Ω to a new signature Ω^λ. The original equations are turned into equations over Ω^λ as follows.

$$\mathtt{x}^\lambda_{\alpha,i} := \mathtt{x}_{\langle\alpha,i\rangle} \tag{4.124a}$$

$$(f(\vec{s}))^\lambda := (\cdots((f^* \bullet s_0^\lambda) \bullet s_1^\lambda) \bullet \cdots \bullet s_{n-1}^\lambda) \tag{4.124b}$$

$$(s = t)^\lambda := s^\lambda = t^\lambda \tag{4.124c}$$

Next, given an Ω–theory, T, let T^λ be the translation of T, with the postulates (vb1) – (vb14) added. It should be easy to see that if $T \vDash s = t$ then also $T^\lambda \vDash s^\lambda = t^\lambda$. For the converse we provide a general model construction that for each multisorted Ω–structure for T gives a multisorted Ω^λ–structure for T^λ in which that equation fails.

An **environment** is a function β from $\mathcal{I} = \mathcal{S} \times \omega$ ($\mathcal{S} = \mathrm{Typ}_\to(B)$) into $\bigcup\langle A_\alpha : \alpha \in \mathcal{S}\rangle$ such that for every index $\langle i,\alpha\rangle$, $\beta(\langle\alpha,i\rangle) \in A_\alpha$. We denote the set of environments by $\mathcal{E}$. Now let $C_{\langle\alpha,\delta\rangle}$ be the set of functions from $\mathcal{E}$ to A_α which depend at most on δ. That is to say, if β and β' are environments such that for all $\iota \in \delta$, $\beta(\iota) = \beta'(\iota)$, and if $f \in C_{\langle\alpha,\delta\rangle}$ then $f(\beta) = f(\beta')$.

The constant f is now interpreted by the function $[f^\star]\colon \beta \mapsto f^\star$. For the 'variables' we put $[\mathrm{x}_\iota]\colon \beta \mapsto \beta(\iota)$. A transformation $\tau\colon \mathcal{I} \to \mathcal{I}$ naturally induces a map $\widehat{\tau}\colon \mathcal{E} \to \mathcal{E}\colon \beta \mapsto \beta \circ \tau$. Further,

$$\begin{aligned}\widehat{\tau \circ \sigma}(\beta) &= \beta \circ (\tau \circ \sigma)\\ &= (\beta \circ \tau) \circ \sigma\\ &= \widehat{\sigma}(\widehat{\tau}(\beta))\\ &= (\widehat{\sigma} \circ \widehat{\tau})(\beta)\end{aligned} \tag{4.125}$$

Let $\sigma = \langle \alpha, \delta \rangle$, $\tau = \langle \alpha, [\iota, \mu][\delta] \rangle$ and $\upsilon = \langle \alpha, (\iota, \mu)[\delta] \rangle$.

$$[\mathrm{s}^\sigma_{\iota,\mu}]\colon C_\sigma \to C_\tau\colon f \mapsto f \circ \widehat{[\iota, \mu]} \tag{4.126}$$

$$[\mathrm{p}^\sigma_{\iota,\mu}]\colon C_\sigma \to C_\upsilon\colon f \mapsto f \circ \widehat{(\iota, \mu)} \tag{4.127}$$

Next, $\bullet^{\langle \alpha \to \beta, \delta \rangle, \langle \alpha, \delta' \rangle}$ is interpreted as follows.

$$\begin{aligned}[\bullet^{\langle \alpha \to \beta, \delta \rangle, \langle \alpha, \delta' \rangle}]\colon C_{\langle \alpha, \delta \rangle} \times C_{\langle \beta, \delta' \rangle} &\to C_{\langle \alpha, \delta \cup \delta' \rangle}\colon\\ \langle f, g \rangle &\mapsto \{\langle \beta, f(\beta) \bullet g(\beta) \rangle : \beta \in \mathcal{E}\}\end{aligned} \tag{4.128}$$

Finally, we define abstraction. Let $\iota = \langle \gamma, i \rangle$.

$$\begin{aligned}[\lambda^{\langle \alpha, \delta \rangle}_\iota]\colon C_{\langle \alpha, \delta \rangle} &\to C_{\langle \gamma \to \beta, \delta - \{\iota\} \rangle}\colon\\ \mathfrak{x} &\mapsto \{\langle \beta, \{\langle y, f([y/\beta(\iota)]\beta) \rangle : y \in A_\gamma\} \rangle : \beta \in \mathcal{E}\}\end{aligned} \tag{4.129}$$

It takes some time to digest this definition. Basically, given f, $g_f(\beta) := \{\langle y, f([y/\beta(\iota)]\beta) \rangle : y \in A_\gamma\}$ is a function from A_γ to A_α with parameter $\beta \in \mathcal{E}$. Hence it is a member of $A_{\gamma \to \alpha}$. It assigns to y the value of f on β', which is identical to β except that now $\beta(\iota)$ is replaced by y. This is the abstraction from y. Finally, for each $f \in C_{\langle \alpha, \delta \rangle}$, $[\lambda^{\langle \alpha, \delta \rangle}_\iota](f)$ assigns to $\beta \in \mathcal{E}$ the value $g_f(\beta)$. (Notice that the role of abstraction is now taken over by the set formation operator $\{x : \quad\}$.)

Theorem 4.78 *Let Ω a multisorted signature, and T an equational theory over Ω. Furthermore, let Ω^λ be the signature of the λ–calculus with 0–ary constants $f^\star$ for every $f \in F$. The theory T^λ consisting of the translation of T and the equations (vb1) — (vb14) is conservative over T. This means that an equation $s = t$ valid in the Ω–algebras satisfying T iff its translation is valid in all Ω^λ–algebras satisfying T^λ.*

Notes on this section. The theory of cylindric algebras has given rise to a number of difficult problems. First of all, the axioms shown above do not fully characterize the cylindric algebras that are representable, that is to say, have as their domain U^κ, κ the dimension, and where relation variables range over n–ary relations over U. Thus, although this kind of cylindric algebra was the motivating example, the equations do not fully characterize it. As Donald Monk [**?**] has shown, there is no finite set of schemes (equations using variables for members of set of variable indices) axiomatizing the class of representable cylindric algebras of dimension κ if $\kappa \geq \aleph_0$; moreover, for finite κ, the class of representable algebras is not finitely axiomatizable. J. S. Johnson has shown in [**?**] an analogue of the second result for polyadic algebras, Sain and Thompson [**?**] an analogue of the first.

The model construction for the model of the λ–calculus is called a syntactical model in [**?**]. It is due to Hindley and Longo from [**?**]. The approach of using functions from the set of variables into the algebra as the carrier set is called a *functional environment model*, and has been devised by Koymans (see [**?**]). A good overview over the different types of models is found in [**?**] and [**?**].

Exercise 160. For f an n–ary function symbol let R_f be an $n+1$–ary relation symbol. Define a translation from terms to formulae as follows. First, for a term t let x_t be a variable such that $x_t \neq x_s$ whenever $s \neq t$.

$$(\mathtt{x}_i)^\dagger := \mathtt{x}_{x_i} = \mathtt{x}_{x_i} \tag{4.130a}$$

$$f(t_0,\ldots,t_{n-1})^\dagger := R_f(x_{t_0},\ldots,x_{t_{n-1}},x_{f(\vec{t})}) \wedge \bigwedge_{i<n} t_i^\dagger \tag{4.130b}$$

Finally, extend this to formulae as follows.

$$R(t_0,\ldots,t_{n-1})^\dagger := R(x_{t_0},\ldots,x_{t_{n-1}}) \wedge \bigwedge_{i<n} t_i^\dagger \tag{4.131a}$$

$$(\neg\varphi)^\dagger := (\neg\varphi^\dagger) \tag{4.131b}$$

$$(\varphi\wedge\chi)^\dagger := (\varphi^\dagger\wedge\chi^\dagger) \tag{4.131c}$$

$$((\exists x)\,\varphi)^\dagger := (\exists x)\,\varphi^\dagger \tag{4.131d}$$

Now, let $\mathfrak{M} = \langle M, \Pi, \mathfrak{I}\rangle$ be a signature. We replace the function symbols by relation symbols, and let $\mathfrak{I}^+$ be the extension of $\mathfrak{I}$ such that

$$\mathfrak{I}^+(R_f) = \{\langle \vec{x}, y\rangle \in M^{\Xi(f)+1} : \Pi(f)(\vec{x}) = y\} \tag{4.132}$$

Then put $\mathfrak{M}^{\dagger} := \langle M, \mathfrak{I}^{+} \rangle$. Show that $\langle \mathfrak{M}, \beta \rangle \vDash \varphi$ iff $\langle \mathfrak{M}^{+}, \beta \rangle \vDash \varphi^{\dagger}$.

Exercise 161. Show that if $\mathfrak{A}$ is a cylindric algebra of dimension κ, every E_{λ}, $\lambda < \kappa$, satisfies the axioms of S5. Moreover, show that if $\mathfrak{A} \vDash \varphi$ then $\mathfrak{A} \vDash \neg \mathsf{E}_{\lambda} \neg \varphi$.

Exercise 162. Prove Lemma 4.71.

Exercise 163. Show Proposition 4.76.

Exercise 164. Show that $\mathcal{L}_{\kappa}/\equiv$ is a cylindric algebra of dimension κ and that it is locally finite dimensional.

Exercise 165. Prove Proposition 4.75.

6. Montague Semantics II

This section deals with the problem of providing a language with a compositional semantics. The problem is to say, which languages that are weakly context free are also strongly context free. The principal result of this section is that if a language is strongly context free, it can be given a compositional interpretation based on an AB–grammar. Recall that there are three kinds of languages: languages as sets of strings, interpreted languages, and finally, systems of signs. A linear system of signs is a subset of $A^{*} \times C \times M$, where C is a set of categories.

Definition 4.79 *A linear sign grammar is* ***context free*** *if (a) C is finite, (b) if f is a mode of arity $n > 0$ then $f^{\varepsilon}(x_0, \ldots, x_{n-1}) := \prod_{i<n} x_i$, (c) $f^{\mu}(m_0, \ldots, m_{n-1})$ is defined if there exist derivable signs $\sigma_i = \langle e_i, c_i, m_i \rangle$, $i < n$, such that $f^{\gamma}(\vec{c})$ is defined and (d) if $f \neq g$ then $f^{\gamma} \neq g^{\gamma}$. If only (a) — (c) are satisfied, the grammar is* ***quasi context free****. Σ is (****quasi****)* ***context free*** *if it is generated by a (quasi) context free linear sign grammar.*

This definition is somewhat involved. (a) says that if f is an n–ary mode, f^{γ} can be represented by a list of n–ary immediate dominance rules. It conjunction with (b) we get that we have a finite list of context free rules. Condition (c) says that the semantics does not add any complexity to this by introducing partiality. Finally, (d) ensures that the rules of the CFG uniquely define the modes. (For we could in principle have two modes which reduce to the same phrase structure rule.) The reader may verify the following simple fact.

Proposition 4.80 *Suppose that Σ is a context free linear system of signs. Then the string language of Σ is context free.*

An interpreted string language is a subset $\mathcal{I}$ of $A^* \times M$ where M is the set of (possible) (sentence) meanings. The corresponding string language is $S(\mathcal{I})$. An interpreted language is **weakly context free** iff the string language is.

Definition 4.81 *An interpreted language $\mathcal{I}$ is **strongly context free** if there is a context free linear system of signs Σ and a category* $\mathtt{S}$ *such that* $\mathtt{S}(\Sigma) = \mathcal{I}$.

For example, let L be the set of declarative sentences of English. M is arbitrary. We take the meanings of declarative sentences to be truth–values, here 0 or 1 (but see Section 4.7). A somewhat more refined approach is to let the meanings be functions from contexts to truth values. Next, we shall also specify what it means for a system of signs to be context free.

Obviously, a linear context free system of signs defines a strongly context free interpreted language. The converse does not hold, however. A counterexample is provided by the following grammar, which generates simple equality statements.

$$\begin{aligned} &\mathtt{E} \to \mathtt{C{=}C} \\ &\mathtt{C} \to \mathtt{D} \mid \mathtt{D{+}D} \\ &\mathtt{D} \to \mathtt{1} \mid \mathtt{2} \end{aligned} \tag{4.133}$$

Expressions of category $\mathtt{E}$ are called equations, and they have as their meaning either true or false. Now, assign the following meanings to the strings. $\mathtt{1}$ has as its $\mathtt{D}$– and $\mathtt{C}$–meaning the number 1, $\mathtt{2}$ the number 2, and $\mathtt{1{+}1}$ as its $\mathtt{C}$–meaning the number 2. The $\mathtt{E}$–meanings are as follows.

$$\begin{array}{llll} [\mathtt{2{=}2}]^{\mathtt{E}} & = \{\mathsf{true}\} & [\mathtt{1{+}1{=}2}]^{\mathtt{E}} & = \{\mathsf{false}\} \\ [\mathtt{1{=}2}]^{\mathtt{E}} & = \{\mathsf{false}\} & [\mathtt{1{=}1{+}1}]^{\mathtt{E}} & = \{\mathsf{true}\} \\ [\mathtt{2{=}1}]^{\mathtt{E}} & = \{\mathsf{false}\} & [\mathtt{1{+}1{=}1}]^{\mathtt{E}} & = \{\mathsf{true}\} \\ [\mathtt{1{=}1}]^{\mathtt{E}} & = \{\mathsf{true}\} & [\mathtt{2{=}1{+}1}]^{\mathtt{E}} & = \{\mathsf{false}\} \\ & & [\mathtt{1{+}1{=}1{+}1}]^{E} & = \{\mathsf{true}\} \end{array} \tag{4.134}$$

This grammar is unambiguous; and every string of category X has exactly one X–meaning for $X \in \{\mathtt{C}, \mathtt{D}, \mathtt{E}\}$. Yet, there is no CFG of signs for this language. For the string $\mathtt{1{+}1}$ has the same $\mathtt{T}$–meaning as $\mathtt{2}$, while substituting one for the other in an equation changes the truth value.

We shall show below that 'weakly context free' and 'strongly context free' coincide for interpreted languages. This means that the notion of an interpreted language is not a very useful one, since adding meanings to sentences does not help in establishing the structure of sentences. The idea of the proof is very simple. Consider an arbitrary linear sign grammar $\mathfrak{A}$ and a start symbol $\mathtt{S}$. We replace $\mathtt{S}$ throughout by $\mathtt{S}^\circ$, where $\mathtt{S}^\circ \notin C$. Now replace the algebra of meanings by the partial algebra of definite structure terms. This defines $\mathfrak{B}$. Then for $c \in C - \{\mathtt{S}\}$, $\langle \vec{x}, c, \mathfrak{s} \rangle$ is a sign generated by $\mathfrak{B}$ iff $\mathfrak{s}$ is a definite structure term such that $\mathfrak{s}^\varepsilon = \vec{x}$ and $\mathfrak{s}^\gamma = c$; and $\langle \vec{x}, \mathtt{S}^\circ, \mathfrak{s} \rangle$ is generated by $\mathfrak{B}$ iff $\mathfrak{s}$ is a definite structure term such that $\mathfrak{s}^\varepsilon = \vec{x}$, and $\mathfrak{s}^\gamma = \mathtt{S}$. Finally, we introduce the following unary mode $\mathtt{F}$.

(4.135) $\quad \mathtt{F}(\langle \vec{x}, \mathtt{S}^\circ, \mathfrak{s} \rangle) := \langle \vec{x}, \mathtt{S}, \mathfrak{s}^\mu \rangle$

This new grammar, call it $\mathfrak{A}^\circ$, defines the same interpreted language with respect to $\mathtt{S}$. So, if an interpreted language is strongly context free, it has a context free sign grammar of this type. Now, suppose that the interpreted language $\mathfrak{I}$ is weakly context free. So there is a CFG G generating $\mathtt{S}(\mathfrak{I})$. At the first step we take the trivial semantics: everything is mapped to 0. This is a strongly context free sign system, and we can perform the construction above. This yields a context free sign system where each $\vec{x}$ has as its C–denotations the set of structure terms that define a C–constituent with string $\vec{x}$. Finally, we have to deal with the semantics. Let $\vec{x}$ be an $\mathtt{S}$–string and let $|M_{\vec{x}}|$ be the set of meanings of $\vec{x}$ and $\mathtt{S}_{\vec{x}}$ the set of structure terms for $\vec{x}$ as an $\mathtt{S}$. If $|\mathtt{S}_{\vec{x}}| < |M_{\vec{x}}|$, there is no grammar for this language based on G. If, however, $|\mathtt{S}_{\vec{x}}| \geq |M_{\vec{x}}|$ there is a function $f_{\vec{x}} \colon \mathtt{S}_{\vec{x}} \to M_{\vec{x}}$. Finally, put

(4.136)
$$f^* := \bigcup \langle f_{\vec{x}} : \vec{x} \in \mathtt{S}(\mathfrak{I}) \rangle$$
$$\mathtt{F}^*(\langle \vec{x}, \mathtt{S}^\circ, \mathfrak{s} \rangle) := \langle \vec{x}, \mathtt{S}, f^*(\mathfrak{s}) \rangle$$

This defines the sign grammar $\mathfrak{A}^*$. It is context free and its interpreted language with respect to the symbol $\mathtt{S}$ is exactly $\mathfrak{I}$.

Theorem 4.82 *Let $\mathfrak{I}$ be a countable interpreted language.*

① *If the string language of $\mathfrak{I}$ is context free then $\mathfrak{I}$ is strongly context free.*

② *For every CFG G for $\mathtt{S}(\mathfrak{I})$ there exists a context free system of signs Σ and with a category $\mathtt{S}$ such that*

(a) $\mathtt{S}(\Sigma) = \mathcal{I}$,

(b) for every nonterminal symbol A

$$\{\vec{x} : \textit{ for some } m \in M : \langle \vec{x}, A, m\rangle \in \Sigma\} = \{\vec{x} : A \vdash_G \vec{x}\}$$

Proof. ② has been established. For ① it suffices to observe that for every CFL there exists a CFG in which every sentence is infinitely ambigous. Just replace $\mathtt{S}$ by $\mathtt{S}^\bullet$ and add the rules $\mathtt{S}^\bullet \to \mathtt{S}^\bullet \mid \mathtt{S}$. □

Notice that the use of unary rules is essential. If there are no unary rules, a given string can have only exponentially many analyses.

Lemma 4.83 *Let L be a CFL and $d > 0$. Then there is a CFG G and such that for all $\vec{x} \in L$ the set of nonisomorphic G–trees for $\vec{x}$ has at least $d^{|\vec{x}|}$ members.*

Proof. Notice that it is sufficient that the result be proved for almost all $\vec{x}$. For finitely many words we can provide as many analyses as we wish. First of all, there is a grammar in Chomsky normal form that generates L. Take two rules that can be used in succession.

$$A \quad \to \quad BC, \qquad C \quad \to \quad DE \tag{4.137}$$

Add the rules

$$A \quad \to \quad XE, \qquad X \quad \to \quad AD \tag{4.138}$$

Then the string ABC has two analyses: $[A\ [B\ C]]$ and $[[A\ B]\ C]$. We proceed similarly if we have a pair of rules

$$A \quad \to \quad XE, \qquad X \quad \to \quad AB \tag{4.139}$$

This grammar assigns exponentially many parses to a given string. To see this notice that any given string $\vec{x}$ of length n contains n distinct constituents. For $n \leq 3$, we use the 'almost all' clause. Now, let $n > 3$. Then $\vec{x}$ has a decomposition $\vec{x} = \vec{y}_0\vec{y}_1\vec{y}_2$ into constituents. By inductive hypothesis, for $\vec{y}_i$ we have $d^{|\vec{y}_i|}$ many analyses. Thus $\vec{x}$ has at least $2d^{|\vec{y}_0|}d^{|\vec{y}_1|}d^{|\vec{y}_2|} = 2d^{|\vec{x}|}$ analyses. □

The previous proof actually assumes exponentially many different structures to a string. We can also give a simpler proof of this fact. Simply replace N by $N \times d$ and replace in each rule every nonterminal X by any one of the $\langle X, k\rangle$, $k < d$.

Theorem 4.84 *Let $\mathcal{I}$ be a countable interpreted language. Then $\mathcal{I}$ is strongly context free for a CFG without unary rules iff $\mathtt{S}(\mathcal{I})$ is context free and there is some constant $c > 0$ such that for almost all strings of A^*: the number of meanings of $\vec{x}$ is bounded by $c^{|\vec{x}|}$.*

We give an example. Let $A := \{\mathtt{Paul}, \mathtt{Marcus}, \mathtt{sees}\}$ and

$$L := \{\mathtt{Paul\ sees\ Paul}, \mathtt{Paul\ sees\ Marcus}, \mathtt{Marcus\ sees\ Paul}, \mathtt{Marcus\ sees\ Marcus}\} \tag{4.140}$$

We associate the following truth values to the sentences.

$$\begin{aligned}\mathcal{I} = \{&\langle \mathtt{Paul\ sees\ Paul}, 0\rangle,\\ &\langle \mathtt{Paul\ sees\ Marcus}, 1\rangle,\\ &\langle \mathtt{Marcus\ sees\ Paul}, 0\rangle,\\ &\langle \mathtt{Marcus\ sees\ Marcus}, 1\rangle\}\end{aligned} \tag{4.141}$$

Furthermore, we fix a CFG that generates L:

$$\begin{array}{llll} \rho_0 & := \mathtt{S} & \to \mathtt{NP\ VP} & \rho_1 := \mathtt{VP} \to \mathtt{V\ NP} \\ \rho_2 & := \mathtt{NP} & \to \mathtt{Paul} & \rho_3 := \mathtt{V} \to \mathtt{sees} \\ \rho_4 & := \mathtt{NP} & \to \mathtt{Marcus} & \end{array} \tag{4.142}$$

We construct a context free system of signs Σ with $\mathtt{S}(\Sigma) = \mathcal{I}$. For every rule ρ of arity $n > 0$ we introduce a symbol $\mathtt{N}_\rho$ of arity n. In the first step the interpretation is simply given by the structure term. For example,

$$\mathtt{N}_1(\langle \vec{x}, \mathtt{V}, \mathfrak{s}\rangle, \langle \vec{y}, \mathtt{NP}, \mathfrak{t}\rangle) = \langle \vec{x} \diamond \vec{y}, \mathtt{VP}, \mathtt{N}_1{}^\frown \mathfrak{s}^\frown \mathfrak{t}\rangle \tag{4.143}$$

(To be exact, on the left hand side we find the unfolding of $\mathtt{N}_1$ rather than the symbol itself.) Only the definition of $\mathtt{N}_0^\mu$ is somewhat different. Notice that the meaning of sentences is fixed. Hence the following must hold.

$$\begin{aligned}(\mathtt{N}_0\mathtt{N}_4\mathtt{N}_1\mathtt{N}_3\mathtt{N}_4)^\mu &= 1\\ (\mathtt{N}_0\mathtt{N}_4\mathtt{N}_1\mathtt{N}_3\mathtt{N}_2)^\mu &= 0\\ (\mathtt{N}_0\mathtt{N}_2\mathtt{N}_1\mathtt{N}_3\mathtt{N}_4)^\mu &= 1\\ (\mathtt{N}_0\mathtt{N}_2\mathtt{N}_1\mathtt{N}_3\mathtt{N}_2)^\mu &= 0\end{aligned} \tag{4.144}$$

We can now do two things: we can redefine the action of the function $\mathtt{N}_0^\mu$. Or we can leave the action as given and factor out the congruence defined by

(4.144) in the algebra of the structure terms enriched by the symbols 0 and 1. If we choose the latter option, we have

$$M := \{0, 1, \mathtt{N}_2, \mathtt{N}_3, \mathtt{N}_4, \mathtt{N}_1\mathtt{N}_3\mathtt{N}_4, \mathtt{N}_1\mathtt{N}_3\mathtt{N}_2, \mathtt{N}_0\mathtt{N}_4\mathtt{N}_1\mathtt{N}_3\mathtt{N}_4, \mathtt{N}_0\mathtt{N}_4\mathtt{N}_1\mathtt{N}_3\mathtt{N}_2, \mathtt{N}_0\mathtt{N}_2\mathtt{N}_1\mathtt{N}_3\mathtt{N}_4, \mathtt{N}_0\mathtt{N}_2\mathtt{N}_1\mathtt{N}_3\mathtt{N}_2\} \tag{4.145}$$

Now let Θ be the congruence defined by (4.144).

$$\begin{aligned} M/\Theta := \{&\{0, \mathtt{N}_0\mathtt{N}_4\mathtt{N}_1\mathtt{N}_3\mathtt{N}_2, \mathtt{N}_0\mathtt{N}_2\mathtt{N}_1\mathtt{N}_3\mathtt{N}_2\}, \{\mathtt{N}_2\}, \{\mathtt{N}_3\}, \\ &\{\mathtt{N}_4\}, \{1, \mathtt{N}_0\mathtt{N}_4\mathtt{N}_1\mathtt{N}_3\mathtt{N}_4, \mathtt{N}_0\mathtt{N}_2\mathtt{N}_1\mathtt{N}_3\mathtt{N}_4\}, \\ &\{\mathtt{N}_1\mathtt{N}_3\mathtt{N}_4\}, \{\mathtt{N}_1\mathtt{N}_3\mathtt{N}_2\}\} \end{aligned} \tag{4.146}$$

Next we define the action on the categories. Let $\rho = B \to A_0 \cdots A_{n-1}$. Then

$$\mathtt{N}^{\gamma}_{\rho}(\gamma_0, \ldots, \gamma_n) := \begin{cases} B & \text{if for all } i < n : \gamma_i = A_i, \\ \star & \text{otherwise.} \end{cases} \tag{4.147}$$

The functions on the exponents are fixed.

Let us pause here and look at the problem of reversibility. Say that $f\colon S \to \wp(T)$ is **finite** if $|f(x)| < \omega$ for every $x \in S$. Likewise, f is **bounded** if there is a number $k < \omega$ such that $|f(x)| < k$ for all x.

Definition 4.85 *Let $\mathfrak{I} \subseteq E \times M$ be an interpreted language. $\mathfrak{I}$ is **finitely reversible (boundedly reversible)** if for every $x \in E$, $x^{\mathfrak{I}} := \{y : \langle x, y\rangle \in \mathfrak{I}\}$ is finite (bounded), and for every $y \in M$, $y_{\mathfrak{I}} := \{x : \langle x, y\rangle \in \mathfrak{I}\}$ is finite (bounded), and moreover, the functions $x \mapsto x^{\mathfrak{I}}$ and $y \mapsto y_{\mathfrak{I}}$ are computable.*

The conditions on $x^{\mathfrak{I}}$ are independent from the conditions on $x_{\mathfrak{I}}$ (see [**?**; **?**]).

Theorem 4.86 (Dymetman) *There are interpreted languages $\mathfrak{I}$ and $\mathfrak{K}$ such that (a) $x \mapsto x^{\mathfrak{I}}$ is finite and computable but $y \mapsto y_{\mathfrak{I}}$ is not, (b) $y \mapsto y_{\mathfrak{K}}$ is finite and computable but $x \mapsto x^{\mathfrak{I}}$ is not.*

In the present context, the problem of enumerating the analyses is trivial. So, given a context free sign grammar, the function $\vec{x} \mapsto \vec{x}^{\mathfrak{I}}$ is always computable, although not always finite. If we insist on branching, $\vec{x}^{\mathfrak{I}}$ grows at most exponentially in the length of $\vec{x}$. We have established nothing about the maps $m \mapsto m_{\mathfrak{I}}$.

Let us now be given a sign system Σ whose projection to $E \times C$ is context free. What conditions must be impose so that there exists a context free sign

grammar for Σ? Let $\vec{x} \in A^*$ and $A \in C$. We write $[\vec{x}]^A_\Sigma := \{m : \langle \vec{x}, A, m\rangle \in \Sigma\}$ and call this set the set of A–**meanings of** $\vec{x}$ **in** Σ. If Σ is given by the context, we omit it. If a system of has a CFG $\mathfrak{A}$ then it satisfies the following equations for every A and every $\vec{x}$.

$$\bigcup_{Z \in F} Z^\mu[[\vec{x}_0]^{B_0} \times \cdots \times [\vec{x}_{\Omega(Z)-1}]^{B_{\Omega(Z)-1}}] = [\vec{x}_0{}^\frown \cdots {}^\frown \vec{x}_{\Omega(Z)-1}]^A \tag{4.148}$$

where $Z^\tau(B_0, \cdots, B_{n-1}) = A$. This means simply put that the A–meanings of $\vec{x}$ can be computed directly from the meanings of the immediate subconstituents. From (4.148) we can derive the following estimate.

$$\sum_{Z \in F} \left(\prod_{i < \Omega(Z)} |[\vec{x}_i]^{B_i}| \right) \geq |[\vec{x}_0{}^\frown \cdots {}^\frown \vec{x}_{\Omega(Z)-1}]^A| \tag{4.149}$$

This means that a string cannot have more meanings than it has readings (= structure terms). We call the condition (4.149) the **count condition**. In particular, it implies that there is a constant c such that for almost all $\vec{x}$ and all A:

$$|[\vec{x}]^A| \leq c^{|\vec{x}|} \tag{4.150}$$

Even if the count condition is satisfied it need not be possible to construct a context free system of signs. Here is an example.

$$\Sigma := \{\langle \mathtt{0}^n, \mathtt{T}, 0\rangle : n \in \omega\} \cup \{\langle \mathtt{a0}^n\mathtt{a}, \mathtt{S}, n\rangle : n \in \omega\} \tag{4.151}$$

A CFG for Σ is

$$\mathtt{S} \to \mathtt{aTa} \qquad\qquad \mathtt{T} \to \varepsilon \mid \mathtt{T0} \tag{4.152}$$

The condition (4.148) is satisfied. But there is no context free sign grammar for Σ. The reason is that no matter how the functions are defined, they must produce an infinite set of numbers from just one input, 0. Notice that we can even define a boundedly reversible system for which no CFG exists. It consists of the signs $\langle \mathtt{0}^n, \mathtt{T}, n\rangle$, $n \in \omega$, the signs $\langle \mathtt{a0}^n\mathtt{a}, \mathtt{S}, 2^n\rangle$ and the signs $\langle \mathtt{a0}^n\mathtt{a}, \mathtt{S}, 3^n\rangle$ where n is prime. We have $\mathtt{a0}^n\mathtt{a}^\mathfrak{I} \subseteq \{2^n, 3^n\}$, whence $|\vec{x}^\mathfrak{I}| \leq 2$, and $|k_\mathfrak{I}| \leq 2$. However, suppose we allow the functions f^ε to be partial, but if defined $f^\varepsilon(\vec{x}_0, \ldots, \vec{x}_{\Omega(f)-1}) = \prod_{i<\Omega(f)} \vec{x}_i$. Then a partial context free grammar exists if the interpreted language defined by Σ is boundedly reversible. (Basically, the partiality allows any degree of sensitivity to the string of which the

expression is composed. Each meaning is represented by a bounded number of expressions.)

Let $\rho = A \to B_0 \cdots B_{n-1}$. Notice that the function $\mathtt{F}_\rho$ makes A–meanings from certain B_i–meanings of the constituents of the string. However, often linguists use their intuition to say what an A–string means under a certain analysis, that is to say, structure term. This — as is easy to see — is tantamount to knowing the functions themselves, not only their domains and ranges. For let us assume we have a function which assigns to every structure term $\mathfrak{t}$ of an A–string $\vec{x}$ an A–meaning. Then the functions $\mathtt{F}_\rho$ are uniquely determined. For let a derivation of $\vec{x}$ be given. This derivation determines derivations of its immediate constituents, which are now unique by assumption. For the tuple of meanings of the subconstituents we know what the function does. Hence, it is clear that for any given tuple of meanings we can say what the function does. (Well, not quite. We do not know what it does on meanings that are not expressed by a B_i–string, $i < \Omega(f)$. However, on any account we have as much knowledge as we need.)

Let us return to Montague Grammar. Let $\Sigma \subseteq A^* \times C \times M$ be strongly context free, with F the set of modes. We want to show that there is an AB–grammar which generates Σ. We have to precisify in what sense we want to understand this. We cannot expect that Σ is any context free system, since AB–grammars are always binary branching. This, however, means that we have to postulate other constituents than those of Σ. Therefore we shall only aim to have the same sentence meanings. In what way we can get more, we shall see afterwards. To start, there is a trivial solution of our problem. If $\rho = A \to B_0 B_1$ is a rule we add a 0–ary mode

$$\mathtt{N}_\rho = \langle \varepsilon, A/B_0/B_1, \lambda x_{B_0}.\lambda x_{B_1}.F_\rho(x_{B_0}, x_{B_1})\rangle \tag{4.153}$$

This allows us to keep our constituents. However, postulating empty elements does have its drawbacks. It increases the costs of parsing, for example. We shall therefore ask whether one can do without empty categories. This is possible. For, as we have seen, with the help of combinators one can liberate oneself from the straightjacket of syntactic structure. Recall from Section 2.2 the transformation of a CFG into Greibach Normal Form. This uses essentially the tool of skipping a rule and of eliminating left recursion. We leave it to the reader to formulate (and prove) an analogon of the skipping of rules for context free sign grammars. This allows us to concentrate on the elimination of left recursion. We look again at the construction of Lemma 2.33. Choose

a nonterminal X. Assume that we have the following X–productions, where $\vec{\alpha}_j$, $j < m$, and $\vec{\beta}_i$, $i < n$, do not contain X.

(4.154) $\rho_j := X \to X^\frown \vec{\alpha}_i, \quad i < m, \qquad \sigma_i := X \to \beta_i, \quad i < n$

Further let $\mathtt{F}^\mu_{\rho_j}$, $j < m$, and $\mathtt{F}^\mu_{\sigma_i}$, $i < n$, be given. To keep the proof legible we assume that $\vec{\beta}_j = Y_j$, $\vec{\alpha}_i = U_i$, are nonterminal symbols. (Evidently, this can be achieved by introducing some more nonterminal symbols.) We we have now these rules.

(4.155) $\rho_j = X \to X^\frown U_i, \quad i < m, \qquad \sigma_i = X \to Y_i, \quad i < n$

So, we generate the following structures.

(4.156) $[Y' \, [U_{i_0} \, [U_{i_1} \cdots [U_{i_{n-2}} \, U_{i_{n-1}}] \cdots]]]$

We want to replace them by these structures instead:

(4.157) $[[\cdots [[Y' \, U_{i_0}] \, U_{i_1}] \cdots U_{i_{n-2}}] \, U_{i_{n-1}}]$

Proceed as in the proof of Lemma 2.33. Choose a new symbol Z and replace the rules by the following ones.

(4.158a) $\lambda_j := X \to U_j, \qquad j < m, \qquad \nu_i := Z \to Y_i, \qquad i < n,$

(4.158b) $\mu_j := X \to Y_j{}^\frown Z, \qquad j < m, \qquad \xi := Z \to Y_i{}^\frown Z, \qquad i < n.$

Now define the following functions.

$$
\begin{aligned}
\mathtt{G}^\mu_{\lambda_i} &:= \mathtt{F}^\mu_{\sigma_i} \\
\mathtt{G}^\mu_{\mu_i}(x_0, x_1) &:= x_1(x_0) \\
\mathtt{G}^\mu_{\nu_i}(x_0, x_1) &:= \lambda x_2.\mathtt{F}^\mu_{\rho_i}(x_1(x_2), x_0) \\
\mathtt{G}^\mu_{\xi_i}(x_0) &:= \lambda x_0.\mathtt{F}^\mu_{\rho_i}(x_0, x_1)
\end{aligned}
\tag{4.159}
$$

Now we have eliminated all left recursion on X. We only have to show that we have not changed the set of X–meanings for any string. To this end, let $\vec{x}$ be an X–string, say $\vec{x} = \vec{y}^\frown \prod_{i<k} \vec{z}_i$, where $\vec{y}$ is a Y_j–string and $\vec{z}_i$ a U_{j_i}–string. Then in the transformed grammar we have the Z–strings

(4.160) $\vec{u}_p := \prod_{p \leq i < k} \vec{z}_i$

and $\vec{x}$ is an X–string. Now we still have to determine the meanings. Let $\mathfrak{m}$ be a meaning of $\vec{y}$ as a Y_i–string and $\mathfrak{n}_i$, $i < k$, a meaning of $\vec{z}_i$ as a U_{j_i}–string. The meaning of $\vec{x}$ as an X–string under this analysis is then

$$\mathtt{F}^{\mu}_{\rho_{j_{n-1}}}(\mathtt{F}^{\mu}_{\rho_{j_{n-2}}}(\cdots(\mathtt{F}^{\mu}_{\rho_{j_0}}(\mathfrak{m},\mathfrak{n}_0),\mathfrak{n}_1),\ldots,\mathfrak{n}_{n-2}),\mathfrak{n}_{n-1}) \tag{4.161}$$

As a Z–string $\vec{u}_{n-1}$ has the meaning

$$\mathfrak{u}_{n-1} := \mathtt{G}^{\mu}_{\nu_{i_{n-1}}}(\mathfrak{n}_{n-1}) = \lambda x_0.\mathtt{F}^{\mu}_{\rho_i}(x_0,\mathfrak{n}_{n-1}) \tag{4.162}$$

Then $\vec{u}_{n-2}$ has the meaning

$$\begin{aligned} \mathfrak{u}_{n-2} &:= \mathtt{G}^{\mu}_{\nu_{i_{n-2}}}(\mathfrak{n}_{n-2},\mathfrak{u}_{n-1}) \\ &= \lambda x_2.\mathtt{F}^{\mu}_{\rho_{n-2}}(\mathfrak{u}_{n-1}(x_2),\mathfrak{n}_{n-2}) \\ &= \lambda x_2.\mathtt{F}^{\mu}_{\rho_{n-2}}(\mathtt{F}^{\mu}_{\rho_i}(x_2,\mathfrak{n}_{n-1}),\mathfrak{n}_{n-2}). \end{aligned} \tag{4.163}$$

Inductively we get

$$\mathfrak{u}_{n-j} = \lambda x_0.\mathtt{F}^{\mu}_{\rho_{n-1}}(\mathtt{F}^{\mu}_{\rho_{n-2}}(\cdots\mathtt{F}^{\mu}_{\rho_{n-j}}(x_0,\mathfrak{n}_{n-j}),\ldots,\mathfrak{n}_{n-2}),\mathfrak{n}_{n-1}) \tag{4.164}$$

If we put $j = n$, and if we apply at last the function $\mathtt{F}^{\mu}_{\mu_j}$ on $\mathfrak{m}$ and the result we finally get that $\vec{x}$ has the same X–meaning under this analysis. The converse shows likewise that every X–analysis of $\vec{x}$ in the transformed grammar can be transformed back into an X–analysis in the old grammar, and the X–meanings of the two are the same.

The reader may actually notice the analogy with the semantics of the Geach rule. There we needed to get new constituent structures by bracketing $[A[B\,C]]$ into $[[A\,B]\,C]$. Supposing that A and B are heads, the semantics of the rule forming $[A\,B]$ must be function composition. This is what the definitions achieve here. Notice, however, that we have no categorial grammar to start with, so the proof given here is not fully analogous. Part of the semantics of the construction is still in the modes themselves, while categorial grammar requires that it be in the meaning of the lexical items.

After some more steps, consisting in more recursion elimination and skipping of rules we are finally done. Then the grammar is in Greibach normal form. The latter can be transformed into an AB–grammar, as we have already seen.

Theorem 4.87 *Let Σ be a context free linear system of signs. Then there exists an AB–grammar that generates Σ.*

The moral to be drawn is that Montague grammmar is actually quite powerful from the point of view of semantics. If the string languages are already context free, then if any context free analysis succeeds, so does an analysis in terms of Montague grammar (supposing here that nothing except linear concatenation is allowed in the exponents). We shall extend this result later to **PTIME**–languages.

Notes on this section. With suitable conditions (such as nondeletion) the set $x^{\mathfrak{I}}$ becomes enumerable for every given x, simply because the number of parses of a string is finite (and has an a priori bound based on the length of x). Yet, $x^{\mathfrak{I}}$ is usually infinite (as we discussed in Section 5.1), and the sets $y_{\mathfrak{I}}$ need not be recursively enumerable for certain y. [**?**] studies how this changes for categorial grammars if semantic representations are not formulae but linear formulae. In that case, the interpreted grammar becomes reversible, and generation is polynomial time computable.

Exercise 166. Let G be a quasi context free sign grammar. Construct a context free sign grammar which generates the same interpreted language.

Exercise 167. Let G be determined by the two rules $\mathtt{S} \to \mathtt{SS} \mid \mathtt{a}$. Show that the set of constituent structures of G cannot be generated by an AB–grammar. *Hint.* Let $d(\alpha)$ be the number of occurrences of slashes (\ or /) in α. If α is the mother of β and γ then either $d(\beta) > d(\alpha)$ or $d(\gamma) > d(\alpha)$.

Exercise 168. Let Σ be strongly context free with respect to a 2–standard CFG G with the following property: there exists a $k \in \omega$ such that for every G–tree $\langle T, <, \sqsubset, \ell \rangle$ and every node $x \in T$ there is a terminal node $y \in T$ with $[y,x] \leq k$. Then there exists an AB–grammar for Σ which generates the same constituent structures as G.

Exercise 169. As we have seen above, left recursion can be eliminated from a CFG G. Show that there exists a CCG(B) grammar which generates for every nonterminal X the same set of X–strings. Derive from this that we can write an AB–grammar which for every X generates the same X–strings as G. Why does it not follow that $L_B(G)$ can be generated by some AB–grammar? *Hint.* For the first part of the exercise consider Exercise 114.

Exercise 170. Let $\langle \mathtt{S}, C, A, \zeta \rangle$ be an AB–grammar. Put $\mathcal{C}^0 := \bigcup_{a \in A} \zeta(a)$. These are the 0th projections. Inductively we put for $\beta^i = \alpha/\gamma \in \mathcal{C}^i$, $\gamma \neq \alpha$, $\beta^{i+1} := \alpha$. In this way we define the projections of the symbols from $\mathcal{C}^0$. Show that by these definitions we get a grammar which satisfies the principles of $\overline{X}$–syntax.

Remark. The maximal projections are not necessarily in $\mathcal{C}^2$.

7. Partiality and Discourse Dynamics

After having outlined the basic setup of Montague Semantics, we shall deal with an issue that we have so far tacitly ignored, namely *partiality*. The name 'partial logic' covers a wide variety of logics that deal with radically different problems. We shall look at two of them. The first is that of partiality as undefinedness. The second is that of partiality as ignorance. We start with partiality as undefinedness.

Consider the assignment $y := (x+1)/(u^2-9)$ to y in a program. This clause is potentially dangerous, since u may equal 3, in which case no value can be assigned to y. Similarly, for a sequence $\mathfrak{a} = (a_n)_{n\in\omega}$, $\lim \mathfrak{a} := \lim_{n\to\infty} a_n$ is defined only if the series is convergent. If not, no value can be given. Or in type theory, a function f may only be applied to x if f has type $\alpha \to \beta$ for certain α and β x has type α. In the linguistic and philosophical literature, this phenomenon is known as **presupposition**. It is defined as a relation between propositions (see [**?**]).

Definition 4.88 *A proposition φ **presupposes** χ if both $\varphi \vdash \chi$ and $\neg\varphi \vdash \chi$. We write $\varphi \gg_{\vdash} \chi$ (or simply $\varphi \gg \chi$) to say that φ presupposes χ.*

The definition needs only the notion of a negation in order to be well–defined. Clearly, in boolean logic this definition gives pretty uninteresting results. φ presupposes χ in PC iff χ is a tautology. However, if we have more than two truth–values, interesting results appear. First, notice that we have earlier remedied partiality by assuming a 'dummy' element $\star$ that a function assumes as soon as it is not defined on its regular input. Here, we shall remedy the situation by giving the expression itself the truth–value $\star$. That is to say, rather than making functions themselves total, we make the assignment of truth–values a total function. This has different consequences, as will be seen. Suppose that we totalize the operator $\lim_{n\to\omega}$ so that it can be applied to all sequences. Then if $(a_n)_{n\in\omega}$ is not a convergent series, say $a_n = (-1)^n$, $3 = \lim_{n\to\infty} a_n$ is not true, since $\lim_{n\to\omega} a_n = \star$ and $3 \neq \star$. The negation of the statement will then be true. This is effectively what Russel [**?**] and Kempson [**?**] claim. Now suppose we say that $3 = \lim_{n\to\infty} a_n$ has no truth–value; then $3 \neq \lim_{n\to\infty} a_n$ also has no truth–value. To nevertheless be able to deal with such sentences rather than simply excluding them from discourse, we

introduce a third truth–value, $\star$. The question is now: how do we define the 3–valued counterparts of $-$, $\cap$ and $\cup$? In order to keep confusion at a minimum, we agree on the following conventions. $\vdash_3$ denotes the 3–valued consequence relation determined by a matrix $\mathfrak{M} = \langle \{0,1,\star\}, \Pi, \{1\} \rangle$, where Π is an interpretation of the connectives. We shall assume that F consists of a subset of the set of the 9 unary and 27 binary symbols, which represent the unary and binary functions on the three element set. This defines Ω. Then $\vdash_3$ is uniquely fixed by F, and the logical connectives will receive a distinct name every time we choose a different function on $\{0,1,\star\}$. What remains to be solved, then, is not what logical language to use but rather by what connective to translate the ordinary language connectors `not`, `and`, `or`, and `if`$\cdots$`then`. Here, we assume that whatever interprets them is a function on $\{0,1,\star\}$ (or $\{0,1,\star\}^2$), whose restriction to $\{0,1\}$ is its boolean counterpart, which is already given. For those functions, the 2–valued consequence is also defined and denoted by $\vdash_2$.

Now, if $\star$ is the truth–value reserved for the otherwise truth valueless statements, we get the following three valued logic, due to Bochvar [**?**]. Its characteristics is the fact that undefinedness is strictly hereditary.

(4.165)

	$-$
0	1
1	0
$\star$	$\star$

$\cap$	0	1	$\star$
0	0	0	$\star$
1	0	1	$\star$
$\star$	$\star$	$\star$	$\star$

$\cup$	0	1	$\star$
0	0	1	$\star$
1	1	1	$\star$
$\star$	$\star$	$\star$	$\star$

The basic connectives are $\neg$, $\wedge$ and $\vee$, which are interpreted by $-$, $\cap$ and $\cup$. Here is a characterization of presupposition in Bochvar's Logic. Call a connective $\clubsuit$ a **Bochvar–connective** if $\Pi(\clubsuit)(\vec{x}) = \star$ iff $x_i = \star$ for some $i < \Omega(\clubsuit)$.

Proposition 4.89 *Let Δ and χ be composed using only Bochvar–connectives. Then $\Delta \vdash_3 \chi$ iff (i) Δ is not classically satisfiable or (ii) $\Delta \vdash_2 \chi$ and* $\mathrm{var}(\chi) \subseteq \mathrm{var}[\Delta]$.

Proof. Suppose that $\Delta \vdash_3 \chi$ and that Δ is satisfiable. Let β be a valuation such that $\overline{\beta}(\delta) = 1$ for all $\delta \in \Delta$. Put $\beta^+(p) := \beta(p)$ for all $p \in \mathrm{var}(\Delta)$ and $\beta^+(p) := \star$ otherwise. Suppose that $\mathrm{var}(\chi) - \mathrm{var}(\Delta) \neq \varnothing$. Then $\overline{\beta^+}(\chi) = \star$, contradicting our assumption. Hence, $\mathrm{var}(\chi) \subseteq \mathrm{var}(\Delta)$. It follows that every valuation that satisfies Δ also satisfies χ, since the valuation does not assume $\star$ on its variables (and can therefore be assumed to be a classical valuation).

Now suppose that $\Delta \nvdash_3 \chi$. Then clearly Δ must be satisfiable. Furthermore, by the argument above either $\mathrm{var}(\chi) - \mathrm{var}(\Delta) \neq \varnothing$ or else $\Delta \vdash_2 \chi$. $\square$

This characterization can be used to derive the following corollary.

Corollary 4.90 *Let φ and χ be composed by Bochvar–connectives. Then $\varphi \gg \chi$ iff* $\mathrm{var}(\chi) \subseteq \mathrm{var}(\varphi)$ *and* $\vdash_2 \chi$.

Hence, although Bochvar's logic makes room for undefinedness, the notion of presupposition is again trivial. Bochvar's Logic seems nevertheless adequate as a treatment of the ι–operator. It is formally defined as follows.

Definition 4.91 *ι is a partial function from predicates to objects such that $\iota x.\chi(x)$ is defined iff there is exactly one b such that $\chi(b)$, and in that case $\iota x.\chi(x) := b$.*

Most mathematical statements which involve presuppositions are instances of a (hidden) use the ι–operator. Examples are the derivative, the integral and the limit. In ordinary language, ι corresponds to the definite determiner `the`. Using the ι–operator, we can bring out the difference between the bivalent interpretation and the three valued one. Define the predicate cauchy' on infinite sequences of real numbers as follows:

(4.166) $\mathsf{cauchy}'(\mathfrak{a}) := (\forall \varepsilon > 0)(\exists n)(\forall m \geq n)|\mathfrak{a}(m) - \mathfrak{a}(n)| < \varepsilon$

This is in formal terms the definition of a Cauchy sequence. Further, define a predicate cum' as follows.

(4.167) $\mathsf{cum}'(\mathfrak{a})(x) := (\forall \varepsilon > 0)(\exists n)(\forall m \geq n)|\mathfrak{a}(m) - x| < \varepsilon$

This predicate says that x is a cumulation point of $\mathfrak{a}$. Now, we may set $\lim \mathfrak{a} := \iota x.\mathsf{cum}'(\mathfrak{a})(x)$. Notice that $\mathsf{cauchy}'(\mathfrak{a})$ is equivalent to

(4.168) $(\exists x)(\mathsf{cum}'(\mathfrak{a})(x) \wedge (\forall y)(\mathsf{cum}'(\mathfrak{a})(y) \rightarrow y = x))$

This is exactly what must be true for $\lim \mathfrak{a}$ to be defined.

(4.169) `The limit of` $\mathfrak{a}$ `equals three.`

(4.170) $\iota x.\mathsf{cum}'(\mathfrak{a})(x) = 3$

(4.171) $(\exists x)(\mathsf{cum}'(\mathfrak{a})(x) \wedge (\forall y)(\mathsf{cum}'(\mathfrak{a})(y) \rightarrow y = x) \wedge x = 3)$.

Under the analysis (4.170) the sentence (4.169) presupposes that $\mathfrak{a}$ is a Cauchy–sequence. (4.171) does not presuppose that. However, the dilemma for the

translation (4.171) is that the negation of (4.169) is also false (at least in ordinary judgement). What this means is that the truth–conditions of (4.172) are not expressed by (4.174), but by (4.175) which in three valued logic is (4.173).

(4.172) `The limit of` $\mathfrak{a}$ `does not equal three.`

(4.173) $\neg(\iota x.\mathsf{cum}'(\mathfrak{a})(x) = 3)$

(4.174) $\neg(\exists x)(\mathsf{cum}'(\mathfrak{a})(x) \wedge (\forall y)(\mathsf{cum}'(\mathfrak{a})(y) \rightarrow y = x) \wedge x = 3)$

(4.175) $(\exists x)(\mathsf{cum}'(\mathfrak{a})(x) \wedge (\forall y)(\mathsf{cum}'(\mathfrak{a})(y) \rightarrow y = x) \wedge \neg(x = 3))$

It is difficult to imagine how to get the translation (4.175) in a bivalent approach, although a proposal is made below. The problem with a bivalent analysis is that it can be shown to be inadequate, because it rests on the assumption that the primitive predicates are bivalent. However, this is problematic. The most clear–cut case is that of the truth–predicate. Suppose we define the semantics of T on the set of natural numbers as follows.

(4.176) $\mathsf{T}(\ulcorner\varphi\urcorner) \leftrightarrow \varphi$

Here, $\ulcorner\varphi\urcorner$ is, say, the Gödel code of φ. It can be shown that there is a χ such that $\mathsf{T}(\ulcorner\chi\urcorner) \leftrightarrow \neg\chi$ is true in the natural numbers. This contradicts (4.176). The sentence χ corresponds to the following liar paradox.

(4.177) `This sentence is false.`

Thus, as Tarski has observed, a truth–predicate that is consistent with the facts in a sufficiently rich theory must be partial. As sentence (4.177) shows, natural languages are sufficiently rich to produce the same effect. Since we do not want to give up the correctness of the truth–predicate (or the falsity predicate), the only alternative is to assume that it is partial. If so, however, there is no escape from the use of three valued logic, since bivalence must fail.

Let us assume therefore that we three truth–values. What Bochvar's logic gives us is called the logic of hereditary undefinedness. For many reasons it is problematic, however. Consider the following two examples.

(4.178) `If` $\mathfrak{a}$ `and` $\mathfrak{b}$ `are convergent sequences,` $\lim(\mathfrak{a}+\mathfrak{b}) = \lim\mathfrak{a} + \lim\mathfrak{b}$.

(4.179) $\mathsf{if}\ u \neq 3\ \mathsf{then}\ y := (x+1)/(u^2-9)\ \mathsf{else}\ y := 0\ \mathsf{fi}$

By Bochvar's Logic, (4.178) presupposes that $\mathfrak{a}$, $\mathfrak{b}$ and $\mathfrak{a}+\mathfrak{b}$ are convergent series. (4.179) presupposes that $u \neq 3$. However, none of the two sentences have nontrivial presuppositions. Let us illustrate this with (4.178). Intuitively, the if–clause preceding the equality statement excludes all sequences from consideration where $\mathfrak{a}$ and $\mathfrak{b}$ are nonconvergent sequences. One can show that $\mathfrak{a}+\mathfrak{b}$, the pointwise sum of $\mathfrak{a}$ and $\mathfrak{b}$, is then also convergent. Hence, the if–clause covers all cases of partiality. The statement $\lim(\mathfrak{a}+\mathfrak{b}) = \lim\mathfrak{a}+\lim\mathfrak{b}$ never fails. Similarly, `and` has the power to eliminate presuppositions.

(4.180) $\mathfrak{a}$ `and` $\mathfrak{b}$ `are convergent series and` $\lim(\mathfrak{a}+\mathfrak{b}) = \lim\mathfrak{a}+\lim\mathfrak{b}$.

(4.181) $u := 4; y := (x+1)/(u^2-9)$

As it turns out, there is an easy fix for that. Simply associate the following connectives with `if`$\cdots$`then` and `and`.

(4.182)

$\rightarrow'$	0	1	$\star$
0	1	1	1
1	0	1	$\star$
$\star$	$\star$	$\star$	$\star$

$\wedge'$	0	1	$\star$
0	0	0	1
1	0	1	$\star$
$\star$	$\star$	$\star$	$\star$

$\vee'$	0	1	$\star$
0	0	1	$\star$
1	1	1	1
$\star$	$\star$	$\star$	$\star$

The reader may take notice of the fact that while $\wedge'$ and $\rightarrow'$ are reasonable candidates for `and` and `if`$\cdots$`then`, $\vee'$ is not as good for `or`.

In the linguistic literature, various attempts have been made to explain these facts. First, we distinguish the *presupposition* of a sentence from its *assertion*. The definition of these terms is somewhat cumbersome. The general idea is that the presupposition of a sentence is a characterization of those circumstances under which it is either true or false, and the assertion is what the sentence says when it is either true of false (that is to say, the assertion tells us when the sentence is true given that it is either true or false). Let us attempt to define this. Let φ be a proposition. Call χ a **generic presupposition of** φ if the following holds. (a) $\varphi \gg \chi$, (b) if $\varphi \gg \psi$ then $\chi \vdash_3 \psi$. If χ is a generic presupposition of φ, $\chi \rightarrow \varphi$ is called an **assertion** of φ. First, notice that presuppositions are only defined up to interderivability. This is not a congruence. We may have $\varphi \dashv\vdash_3 \chi$ without φ and χ receiving the same truth–value under all assignments. Namely, $\varphi \dashv\vdash_3 \chi$ iff φ and χ are truth–equivalent, that is, $\overline{\beta}(\varphi) = 1$ exactly when $\overline{\beta}(\chi) = 1$. In order to have full equivalence, we must also require $\neg\varphi \dashv\vdash_3 \neg\chi$. Second, notice that $\varphi \vee \neg\varphi$ satisfies (a) and (b). However, $\varphi \vee \neg\varphi$ presupposes itself, something that we wish to avoid. So,

Table 14. The Projection Algorithm

$$
\begin{array}{lll}
A(\neg\varphi) & = \neg A(\varphi) & P(\neg\varphi) = P(\varphi) \\
A(\varphi \wedge \chi) & = A(\varphi) \wedge A(\chi) & P(\varphi \wedge \chi) = P(\varphi) \wedge P(\chi) \\
A(\varphi \vee \chi) & = A(\varphi) \vee A(\chi) & P(\varphi \vee \chi) = P(\varphi) \wedge P(\chi) \\
A(\varphi \rightarrow \chi) & = A(\varphi) \rightarrow A(\chi) & P(\varphi \rightarrow \chi) = P(\varphi) \wedge P(\chi) \\
A(\varphi \wedge' \chi) & = A(\varphi) \wedge A(\chi) & P(\varphi \wedge' \chi) = P(\varphi) \\
 & & \qquad \wedge (A(\varphi) \rightarrow P(\chi)) \\
A(\varphi \vee' \chi) & = A(\varphi) \vee A(\chi) & P(\varphi \vee' \chi) = P(\varphi) \\
 & & \qquad \wedge (\neg A(\varphi) \rightarrow P(\chi)) \\
A(\varphi \rightarrow' \chi) & = (A(\varphi) \wedge P(\varphi)) & P(\varphi \rightarrow' \chi) = P(\varphi) \\
 & \qquad \rightarrow A(\chi) & \qquad \wedge (A(\varphi) \rightarrow P(\chi))
\end{array}
$$

we additionally require the generic presupposition to be **bivalent**. Here, φ is **bivalent** if for every valuation β into $\{0,1,\star\}$: $\overline{\beta}(\varphi) \in \{0,1\}$. Define the following connective.

(4.183)

$\downarrow$	0	1	$\star$
0	$\star$	0	$\star$
1	$\star$	1	$\star$
$\star$	$\star$	$\star$	$\star$

Definition 4.92 *Let φ be a proposition. χ is a **generic presupposition of** φ **with respect to** $\vdash_3$ if (a) $\varphi \gg \chi$, (b) χ is bivalent and (c) if $\varphi \gg \psi$ then $\chi \vdash_3 \psi$. χ is the **assertion** of φ if (a) χ is bivalent and (b) $\chi \dashv\vdash_3 \varphi$. Write $P(\varphi)$ for the generic presupposition (if it exists), and $A(\varphi)$ for the assertion.*

It is not a priori clear that a proposition has a generic presupposition. A case in point is the truth–predicate. Write $\varphi \equiv_3 \chi$ if $\beta(\varphi) = \beta(\chi)$ for all β.

Proposition 4.93 $\varphi \equiv_3 A(\varphi) \downarrow P(\varphi)$.

The **projection algorithm** is a procedure that assigns generic presuppositions to complex propositions by induction over their structure. Table 14 shows a projection algorithm for the connectives defined so far. It is an easy matter to define projection algorithms for all connectives. The prevailing intuition is that the three valued character of `and`, `or` and `if`$\cdots$`then` is best explained in terms of **context change**. A **text** is a sequence of propositions, say $\Delta = \langle \delta_i : i < n \rangle$. A text is **coherent** if for every $i < n$: $\langle \delta_j : j < i \rangle \vdash_3 \delta_i \vee \neg\delta_i$.

In other words, every member is either true or false given that the previous propositions are considered true. (Notice that the order is important now.) In order to extend this to parts of the δ_j we define the **local context** as follows.

Definition 4.94 *Let $\Delta = \langle \delta_i : i < n \rangle$. The **local context** of δ_j is $\langle \delta_i : i < j \rangle$. For a subformula occurrence of δ_j, the **local context** of that occurrence is defined as follows.*

① *If Σ is the local context of $\varphi \wedge' \chi$ then (a) Σ is the local context of φ and (b) $\Sigma; \varphi$ is the local context of χ.*

② *If Σ is the local context of $\varphi \rightarrow' \chi$ then (a) Σ is the local context of φ and (b) $\Sigma; \varphi$ is the local context of χ.*

③ *If Σ is the local context of $\varphi \vee' \chi$ then (a) Σ is the local context of φ and (b) $\Sigma; \neg\varphi$ is the local context of χ.*

*δ_j is **bivalent in its local context** if for all valuations that make all formulae in the local context true, δ_j is true or false.*

The presupposition of φ is the formula χ such that φ is bivalent in the context χ, and which implies all other formulae that make φ bivalent. It so turns out that the context dynamics define a three valued extension of a 2–valued connective, and conversely. The above rules are an exact match. Such formulations have been given by Irene Heim [**?**], Lauri Karttunen [**?**] and also Jan van Eijck [**?**]. It is easy to understand this in computer programs. A computer program may contain clauses carrying presuppositions (for example clauses involving divisions), but it need not fail. For if whenever a clause carrying a presupposition is evaluated, that presupposition is satisfied, no error ever occurs at runtime. In other words, the local context of that clause satisfies the presuppositions. What the local context is, is defined by the evaluation procedure for the connectives. In computer languages, the local context is always to the left. But this is not necessary. The computer evaluates α and β by first evaluating α and then β only if α is true — but it could also evaluate β first and then evaluate α whenever β is true. In [**?**] it is shown that in the definition of the local context all that needs to be specified is the directionality of evaluation. The rest follows from general principles. Otherwise one gets connectives that extend the boolean connectives in an improper way (see below on that notion).

The behaviour of presuppositions in quantification and propositional attitude reports is less straightforward. We shall only give a sketch.

(4.184) `Every bachelor of the region got a letter from that marriage agency.`

(4.185) `Every person in that region is a bachelor.`

(4.186) `John believes that his neighbour is a bachelor.`

We have translated `every` using the quantifier $\forall$ in predicate logic. We wish to extend it to a three–valued quantifier, which we also call $\forall$. (4.184) is true even if not everybody in the region is a bachelor; in fact, it is true exactly if there is no non–bachelor. Therefore we say that $(\forall x)\varphi$ is true iff there is no x for which $\varphi(x)$ is false. $(\forall x)\varphi$ is false if there is an x for which $\varphi(x)$ is false. Thus, the presupposition effectively restricts the range of the quantifier. $(\forall x)\varphi$ is bivalent. This predicts that (4.185) has no presuppositions. On (4.186) the intuitions vary. One might say that it does not have any presuppositions, or else that it presupposes that the neighbour is a man (or perhaps: that John believes that his neighbour is a man). This is deep water (see [**?**]).

Now we come to the second interpretation of partiality, namely *ignorance*. Let $\star$ now stand for the fact that the truth–value is not known. Also here the resulting logic is not unique. Let us take the example of a valuation β that is only defined on some of the variables. Now let us be given a formula φ. Strictly speaking $\overline{\beta}(\varphi)$ is not defined on φ if the latter contains a variable that is not in the domain of β. On the other hand, there are clear examples of propositions that receive a definite truth–value no matter how we extend β to a total function. For example, even if β is not defined on p, every extension of it must make $p \vee \neg p$ true. Hence, we might say that β also makes $p \vee \neg p$ true. This is the idea of **supervaluations** by Bas van Fraassen. Say that φ is **sv–true** (**sv–false**) under β if φ is true under every total $\gamma \supseteq \beta$. If φ is neither sv–true nor sv—false, call it **sv–indeterminate**. Unfortunately, there is no logic to go with this approach. Look at the interpretation of `or`. Clearly, if either φ or χ is sv–true, so is their disjunction. However, what if both φ and χ are sv–indeterminate?

(4.187)

$\cup$	0	1	$\star$
0	0	1	$\star$
1	1	1	1
$\star$	$\star$	1	?

The formula $p \vee q$ is sv–indeterminate under the empty valuation. It has no definite truth–value, because both p and q could turn out to be either true or false. On the other hand, $p \vee \neg p$ is sv–true under the empty valuation, even though both p and $\neg p$ are sv–indeterminate. So, the supervaluation approach is not so well–suited. Stephen Kleene actually had the idea of doing a worst case interpretation: if you can't always say what the value is, fill in $\star$. This gives the so–called **Strong Kleene Connectives** (the weak ones are like Bochvar's).

(4.188)

	$-$
0	1
1	0
$\star$	$\star$

$\cap^\diamond$	0	1	$\star$
0	0	0	0
1	0	1	$\star$
$\star$	0	$\star$	$\star$

$\cup^\diamond$	0	1	$\star$
0	0	1	$\star$
1	1	1	1
$\star$	$\star$	1	$\star$

These connectives can be defined in the following way. Put $1^\diamond := \{1\}$, $0^\diamond := \{0\}$ and $\star := \{0,1\}$. Now put

(4.189) $\quad f^\diamond(x_0^\diamond, x_1^\diamond) := f[x_0^\diamond \times x_1^\diamond]$

For example, $\cup^\diamond(\langle 0^\diamond, \star^\diamond\rangle) = \cup[\{0\} \times \{0,1\}] = \{0 \cup 0, 0 \cup 1\} = \{0,1\} = \star^\diamond$. So, we simply take sets of truth–values and calculate with them.

A more radical account of ignorance is presented by constructivism and intuitionism. A constructivist denies that the truth or falsity of a statement can always be assessed directly. In particular, an existential statement is true only if we produce an instance that satisfies it. A universal statement can be considered true only if we possess a proof that any given element satisfies it. For example, Goldbach's conjecture is that every even number greater than 2 is the sum of two primes. According to a constructivist, at present it is neither true nor false. For on the one hand we have no proof that it holds, on the other hand we know of no even number greater than 2 which is not the sum of two primes. Both constructivists and intuitionists unanimously reject axiom (a2). (Put $\bot$ for p_1. Then in conjunction with the other rules this gives $(\neg p_0 \to p_0) \to p_0$. This corresponds to the **Rule of Clavius**: from $\neg p_0 \to p_0$ conclude p_0.) They also reject $p \vee \neg p$, the so–called **Law of the Exluded Middle**. The difference between a constructivist and an intuitionist is the treatment of negative evidence. While a constructivist accepts basic negative evidence, for example, that this lemon is not green, for an intuitionist there is no such thing as direct evidence to the contrary. We only witness the absence of the fact that the lemon is green. Both, however, are reformist in

the sense that they argue that the mathematical connectives `and`, `or`, `not`, and `if`$\cdots$`then` have a different meaning. However, one can actually give a reconstruction of both inside classical mathematics. We shall deal first with intuitionism. Here is a new set of connectives, defined with the help of $\Box$, which satisfies S4.

$$\begin{aligned} \neg^i\varphi &:= \Box(\neg\varphi) \\ \varphi \vee^i \chi &:= \varphi \vee \chi \\ \varphi \wedge^i \chi &:= \varphi \wedge \chi \\ \varphi \to^i \chi &:= \Box(\varphi \to \chi) \end{aligned} \tag{4.190}$$

Call an I–proposition a proposition formed from variables and $\perp$ using only the connectives just defined.

Definition 4.95 *An **I–model** is a pair $\langle P, \leq, \beta\rangle$, where $\langle P, \leq\rangle$ is a partially ordered set and $\beta(p) = \uparrow\beta(p)$ for all variables p.*

Intuitively, the nodes of P represent stages in the development of knowledge. Knowledge develops in time along $\leq$. We say that x **accepts** φ if $\langle P, \leq, x, \beta\rangle \vDash \varphi$, and that x **knows** φ if $\langle P, \leq, x, \beta\rangle \vDash \Box\varphi$. By definition of β, once a proposition p is accepted, it is accepted for good and therefore considered known. Therefore Gödel simply translated variables p by the formula $\Box p$. Thus, intuitionistically the statement that p may therefore be understood as 'p is known' rather than 'p is accepted'. The systematic conflation of knowledge and simple temporary acceptance as true is the main feature of intuitionistic logic.

Proposition 4.96 *Let $\langle P, \leq, \beta\rangle$ be an I–model and φ an I–proposition. Then $\overline{\beta}(\varphi) = \uparrow\overline{\beta}(\varphi)$ for all φ.*

Constructivism in the definition by Nelson adds to intuitionism a second valuation for those variables that are definitely rejected, and allows for the possibility that neither is the case. (However, nothing can be both accepted and rejected.) This is reformulated as follows.

Definition 4.97 *A **C–model** is a pair $\langle P, \leq, \beta\rangle$, where $\langle P, \leq\rangle$ is a partially ordered set and $\beta\colon V \times P \to \{0, 1, \star\}$ such that if $\beta(p, v) = 1$ and $v \leq w$ then also $\beta(p, w) = 1$, and if $\beta(p, v) = 0$ and $v \leq w$ then $\beta(p, w) = 0$. We write $\langle P, \leq, x, \beta\rangle \vDash^+ p$ if $\beta(p, x) = 1$ and $\langle P, \leq, x, \beta\rangle \vDash^- p$ if $\beta(p, x) = 0$.*

We can interpret any propositional formula over 3–valued logic that we have defined so far. We have to interpret $\Box$ and $\Diamond$, however.

$$
\begin{aligned}
x \vDash^{+} \Box\varphi &:\Leftrightarrow \text{ for no } y \geq x : y \vDash^{-} \varphi \\
x \vDash^{-} \Box\varphi &:\Leftrightarrow \text{ there is } y \geq x : y \vDash^{-} \varphi \\
x \vDash^{+} \Diamond\varphi &:\Leftrightarrow \text{ there is } y \geq x : y \vDash^{+} \varphi \\
x \vDash^{-} \Diamond\varphi &:\Leftrightarrow \text{ for no } y \geq x : y \vDash^{+} \varphi
\end{aligned}
\tag{4.191}
$$

Now define the following new connectives.

$$
\begin{aligned}
\neg^{c}\varphi &:= \neg\varphi \\
\varphi \vee^{c} \chi &:= \varphi \vee^{\diamond} \chi \\
\varphi \wedge^{c} \chi &:= \varphi \wedge^{\diamond} \chi \\
\varphi \rightarrow^{c} \chi &:= \Box(\varphi \rightarrow^{\diamond} \chi)
\end{aligned}
\tag{4.192}
$$

In his data semantics (see [**?**]), Frank Veltman uses constructive logic and proposes to interpret `must` and `may` as $\Box$ and $\Diamond$, respectively. What is interesting is that the set of points accepting $\Diamond\varphi$ is lower closed but *not* necessarily upper closed, while the set of points rejecting it is upper but not necessarily lower closed. The converse holds with respect to $\Box$. This is natural, since if our knowledge grows there are less things that *may* be true but more that *must* be.

The interpretation of the arrow carries the germ of the relational interpretation discussed here. A different strand of thought is the theory of conditionals (see again [**?**] and also [**?**]). The conditional $\varphi > \chi$ is accepted as true under Ramsey's interpretation if, on taking φ as a hypothetical assumption (doing as if φ is the case) and performing the standard reasoning, we find that χ is true as well. Notice that after this routine of hypothetical reasoning we retract the assumption that φ. In Gärdenfors models there are no assignments in the ordinary sense. A proposition φ is mapped directly onto a function, the update function U_{φ}. The states in the Gärdenfors model carry no structure.

Definition 4.98 *A **Gärdenfors model** is a pair $\langle G, U\rangle$, where G is a set, and $U\colon \mathrm{Tm}_{\Omega} \rightarrow G^{G}$ subject to the following constraints.*

① *For all χ: $U_{\chi} \circ U_{\chi} = U_{\chi}$.*

② *For all φ and χ: $U_{\varphi} \circ U_{\chi} = U_{\chi} \circ U_{\varphi}$.*

*We say that $x \in G$ **accepts** φ if $U_{\varphi}(x) = x$.*

Put $x \leq y$ iff there is a finite set $\{\chi_i : i < n\}$ such that

(4.193) $$y = U_{\chi_0} \circ U_{\chi_1} \circ \cdots \circ U_{\chi_{n-1}}(x)$$

The reader may verify that this relation is reflexive and transitive. If we require that $x = y$ iff x and y accept the same propositions, then this ordering is also a partial ordering. We can define as follows. If $U_\chi = U_\varphi \circ U_\psi$ then we write $\chi = \varphi \wedge \psi$. Hence, if our language actually has a conjunction, the latter is a condition on the interpretation of it. To define $\rightarrow$, Gärdenfors does the following. Suppose that for all φ and χ there exists a δ such that $U_\varphi \circ U_\delta = U_\chi \circ U_\delta$. Then we simply put $U_{\varphi \leftrightarrow \psi} := U_\delta$. Finally, since $\varphi \rightarrow \psi$ is equivalent to $\varphi \leftrightarrow \varphi \wedge \psi$, once we have $\wedge$ and $\leftrightarrow$, we can also define $\rightarrow$. For negation we need to assume the existence of an inconsistent state. The details need not concern us here. Obviously, Gärdenfors models are still more general than data semantics. In fact, any kind of logic can be modelled by a Gärdenfors model (see the exercises).

Notes on this section. It is an often discussed problem whether or not a statement of the form $(\forall x)\varphi$ is true if there are no x at all. Equivalently, in three valued logic, it might be said that $(\forall x)\varphi$ is undefined if there is no x such that $\varphi(x)$ is defined.

Exercise 171. A three valued binary connective $\clubsuit$ satisfies the Deduction Theorem if for all Δ, φ and χ: $\Delta; \varphi \vdash_3 \chi$ iff $\Delta \vdash_3 \varphi \clubsuit \chi$. Establish all truth–tables for binary connectives that satisfy the Deduction Theorem. Does any of the implications defined above have this property?

Exercise 172. Let $\mathcal{L}$ be a language and $\vdash$ a structural consequence relation over $\mathcal{L}$. Let $G_\vdash$ be the set of theories of $\vdash$. For $\varphi \in \mathcal{L}$, let $U_\varphi : T \mapsto (T \cup \{\varphi\})^\vdash$. Show that this is a Gärdenfors model. Show that the set of formulae accepted by all $T \in G_\vdash$ is exactly the set of tautologies of $\vdash$.

Chapter 5
PTIME Languages

1. Mildly–Context Sensitive Languages

The introduction of the Chomsky hierarchy has sparked off a lot of research into the complexity of formal and natural languages. Chomsky's own position was that language was not even of Type 1. In transformational grammar, heavy use of context sensitivity and deletion has been made. However, Chomsky insisted that these grammars were not actually models of performance, neither of sentence production nor of analysis; they were just models of competence. They were theories of language or of languages, couched in algorithmic terms. In the next chapter we shall study a different type of theory, based on axiomatic descriptions of structures. Here we shall remain with the algorithmic approach. If Chomsky is right, the complexity of the generated languages is only of peripheral interest and, moreover, cannot even be established by looking at the strings of the language. Thus, if observable language data can be brought to bear on the question of the 'language faculty', we actually need to have

- ☞ a theory of the human language(s),
- ☞ a theory of human sentence production, and
- ☞ a theory of human sentence analysis (and understanding).

Namely, the reason that a language may fail to show its complexity in speech or writing is that humans simply are unable to produce the more complex sentences, even though given enough further means they would be able to produce any of them. The same goes for analysis. Certain sentences might be avoided not because they are illegitimate but because they are misleading or too difficult to understand. An analogy that might help is the notion of a programming language. A computer is thought to be able to understand every program of a given computer language if it has been endowed with an understanding of the syntactic primitives and knows how to translate them into executable routines. Yet, some programs may simply be too large for the computer to be translated let alone executed. This may be remedied by giving it more memory (to store the program) or a bigger processing unit (to

be able to execute it). None of the upgrading operations, however, seem to touch on the basic ability of the computer to understand the language: the translation or compilation program usually remains the same. Some people have advanced the thesis that certain monkeys possess the symbolic skills of humans but since they cannot handle recursion, so that their ability to use language is restricted to single clauses consisting of single word phrases (see for example [**?**] and [**?**], Pages 8 – 12).

One should be aware of the fact that the average complexity of spoken language is linear (= $O(n)$) for humans. We understand sentences as they are uttered, and typically we seem to be able to follow the structure and message word by word. To conclude that therefore human languages must be regular is premature. For one thing, we might just get to hear the easy sentences because they are also easy to generate: it is humans who talk to humans. Additionally, it is not known what processing device the human brain is. Suppose that it is a finite state automaton. Then the conclusion is certainly true. However, if it is a pushdown automaton, the language can be deterministically context free. More complex devices can be imagined giving rise to even larger classes of languages that can be parsed in linear time. This is so since it is not clear that what is one step for the human brain also is one step for, say, a Turing machine. It is known that the human brain works with massive use of parallelism, for example.

Therefore, the problem with the line of approach advocated by Chomsky is that we do not possess a reliable theory of human sentence processing let alone of sentence production (see [**?**] for an overview of the latter). Without them, however, it is impossible to assess the correctness of any proposed theory of grammar. Many people have therefore ignored this division of labour into three faculties (however reasonable that may appear) and tried to assess the complexity of the language as we see it. Thus let us ask once more:

How complex is human language (are human languages)?

While the Chomsky hierarchy has suggested measuring complexity in terms of properties of rules, it is not without interest to try to capture its complexity in terms of resources (time and space complexity). The best approximation that we can so far give is this.

Human languages are in **PTIME**.

In computer science, **PTIME** problems are also called ‘tractable’, since the

time consumption grows slowly. On the other hand, **EXPTIME** problems are called 'intractable'. Their time consumption grows too fast. In between the two lie the classes **NPTIME** and **PSPACE**. Still today it is not known whether or not **NPTIME** is contained in (and hence equal to) **PTIME**. Problems which are **NPTIME**–complete usually do possess algorithms that run (deterministically) in polynomial time — on the average.

Specifically, Aravind Joshi has advanced the claim that languages are what he calls 'mildly context sensitive' (see [**?**]). Mildly context sensitive languages are characterized as follows.

Definition 5.1 *$L \subseteq A^*$ has the **constant growth property** if it is finite or there is a number c_L such that for every $\vec{x} \in L$ there is a $\vec{y} \in L$ such that $|\vec{x}| < |\vec{y}| \leq |\vec{x}| + c_L$.*

① Every context free language is mildly context sensitive. There are mildly context sensitive languages which are not context free.

② Mildly context sensitive languages can be recognized in deterministic polynomial time.

③ There is only a finite number of crossed dependency types.

④ Mildly context sensitive languages have the constant growth property.

These conditions are not very strong except for the second. It implies that the mildly context sensitive languages form a proper subset of the context sensitive languages. ① needs no comment. ④ is quite weak. Moreover, it seems that for every natural language there is a number d_L such that for every $n \geq d_L$ there is a string of length n in L. Rambow [**?**] proposes to replace it with the requirement of semilinearity, but that seems to be too strong (see Michaelis and Kracht [**?**]). Also ③ is problematic. What exactly is a crossed dependency type? In this chapter we shall study grammars in which the notion of structure can be defined as with context free languages. Constituents are certain subsets of disjoint (occurrences of) subwords. If this definition is accepted, ③ can be interpreted as follows: there is a number n such that a given constituent has no more than n parts. This is certainly not what Joshi had in mind when formulating his conditions, but it is certainly not easy to come up with a definition that is better than this one and as clear.

So, the conditions are problematic with the exception of ②. Notice that ② implies ①, and, as we shall see, also ③ (if only weak equivalence counts

here). ④ shall be dropped. In order not to create confusion we shall call a language a **PTIME language** if it has a deterministic polynomial time recognition algorithm (see Definition 1.100). In general, we shall also say that a function $f\colon A^* \to B^*$ is in **PTIME** if there is a deterministic Turing machine which computes that function. Almost all languages that we have considered so far are **PTIME** languages. This shall emerge from the theorems that we shall prove further below.

Proposition 5.2 *Every context free language is in **PTIME**.*

This is a direct consequence of Theorem 2.57. However, we get more than this.

Proposition 5.3 *Let A be a finite alphabet and L_1, L_2 languages over A. If $L_1, L_2 \in$ **PTIME** then so is $A^* - L_1$, $L_1 \cap L_2$ and $L_1 \cup L_2$.*

The proof of this theorem is very simple and left as an exercise. So we get that the intersection of CFLs, for example $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n : n \in \omega\}$, are **PTIME** languages. Condition ② for mildly context sensitive languages is satisfied by the class of **PTIME** languages. Further, we shall show that the full preimage of a **PTIME** language under the Parikh–map is again a **PTIME** language. To this end we shall identify $M(A)$ with the set of all strings of the form $\prod_{i<n} \mathtt{a}_i^{p_i}$. The Parikh–map is identified with the function $\pi\colon A^* \to A^*$, which assigns to a string $\vec{x}$ the string $\mathtt{a}_0^{p_0} \frown \mathtt{a}_1^{p_1} \frown \cdots \frown \mathtt{a}_{n-1}^{p_{n-1}}$, where p_j is the number of occurrences of $\mathtt{a}_j$ in $\vec{x}$. Now take an arbitrary polynomial time computable function $g\colon A^* \to 2$. Clearly, $g \restriction M(A)$ is also in **PTIME**. The preimage of 1 under this function is contained in the image of π. $g^{-1}(1) \cap M(A)$ can be thought of in a natural way as a subset of $M(A)$.

Theorem 5.4 *Let $L \subseteq M(A)$ be in **PTIME**. Then $\pi^{-1}[L]$, the full preimage of L under π, also is in **PTIME**. If L is semilinear, $\pi^{-1}[L]$ is in **PTIME**.*

The reader is warned that there nevertheless are semilinear languages which are not in **PTIME**. For **PTIME** is countable, but there are uncountably many semilinear languages (see Exercise 74). Theorem 5.4 follows directly from

Theorem 5.5 *Let $f\colon B^* \to A^*$ be in **PTIME** and $L \subseteq A^*$ in **PTIME**. Then $M := f^{-1}[L]$ also is in **PTIME**.*

Proof. By definition $\chi_L \in$ **PTIME**. Then $\chi_M = \chi_L \circ f \in$ **PTIME**. This is the characteristic function of M. □

Another requirement for mildly context sensitive languages was the constant growth property. We leave it to the reader to show that every semilinear language has the constant growth property but that there are languages which have the constant growth property without being semilinear.

We have introduced the Polish Notation for terms in Section 1.2. Here we shall introduce a somewhat exotic method for writing down terms, which has been motivated by the study of certain Australian languages (see [**?**]). Let $\langle F, \Omega \rangle$ be a finite signature. Further, let

$$\Omega_{\mathsf{T}} := \max\{\Omega(f) : f \in F\} \tag{5.1}$$

Inductively, we assign to every term t a set $M(t) \subseteq (F \cup \{\mathtt{0}, \mathtt{1}, \ldots, \Omega_{\mathsf{T}} - 1\})^*$:

① If $t = f$ with $\Omega(f) = 0$ then put $M(f) := \{f\}$.

② If $t = f(s_0, \ldots, s_{\Omega(f)-1})$ then put

$$M(t) := \{f\} \cup \bigcup_{i < \Omega(f)} \{\vec{x}^\frown i : \vec{x} \in M(s_i)\}$$

An element of $M(t)$ is a product $f^\frown \vec{y}$, where $f \in F$ and $\vec{y} \in \Omega_{\mathsf{T}}^*$. We call f the **main symbol** and $\vec{y}$ its **key**. We choose a new symbol, `#`. Now we say that $\vec{y}$ is an **A–form** of t if $\vec{y}$ is the product of the elements of $M(t)$ in an arbitrarily chosen (nonrepeating) sequence, separated by `#`. For example, let $t :=$ `(((x+a)-y)+(z-c))`. Then

$$M(t) = \{\mathtt{+}, \mathtt{-0}, \mathtt{-1}, \mathtt{+00}, \mathtt{x000}, \mathtt{a001}, \mathtt{y01}, \mathtt{z10}, \mathtt{c11}\} \tag{5.2}$$

Hence the following string is an A–form of t:

`c11#z10#+00#-0#-1#y01#x000#a001#+` (5.3)

Theorem 5.6 *Let $\langle F, \Omega \rangle$ be a finite signature and L the language of A–forms of terms of this signature. Then L is in **PTIME**.*

Proof. For each A–form $\vec{x}$ there is a unique term t such that $\vec{x}$ is the A–form of t, and there is a method to calculate $M(t)$ on the basis of $\vec{x}$. One simply has to segment $\vec{x}$ into correctly formed parts. These parts are maximal sequences consisting of a main symbol and a key, which we shall now simply call **stalks**. The segmentation into stalks is unique. We store $\vec{x}$ on a read and write tape

τ_i. Now we begin the construction of t. t will be given in Polish Notation. t will be constructed on Tape τ_o in left–to–right order. We will keep track of the unfinished function symbols Tape on τ_s. We search through the keys (members of Ω_{T}^*) in lexicographic order. On a separate tape, τ_k, we keep note of the current key.

① $\tau_s := \varepsilon$, $\tau_k := \varepsilon$, $\tau_o := \varepsilon$.

② Match $\tau_i = \vec{y}\#f^\frown\tau_k\#\vec{z}$. If match succeeds, put $\tau_i := \vec{y}\#\vec{z}$, $\tau_s := \tau_s{}^\frown f$, $\tau_o := \tau_o{}^\frown f$. Else exit: 'String is not an A–form.'

③ Let g be the last symbol of τ_s, n the last symbol of τ_k.

- $*$ If $n = \Omega(g) - 1$, $\tau_s := \tau_s/g$, $\tau_k := (\tau_k/n)$.
- $*$ Else $\tau_s := \tau_s$, $\tau_k := (\tau_k/n)^\frown(n+1)$.

④ If $\tau_k = \tau_s = \varepsilon$, go to ⑤ else go to ②.

⑤ If $\tau_i = \varepsilon$ exit: 'String is an A–form.' Else exit: 'String is not an A–form.'

It is left to the reader to check that this algorithm does what it is supposed to do. Polynomial runtime is obvious. □

Ebert and Kracht [**?**] show that this algorithm requires $O(n^{3/2}\log n)$ time to compute. Now we shall start the proof of an important theorem on the characterization of **PTIME** languages. An important step is a theorem by Chandra, Kozen and Stockmeyer [**?**], which characterizes the class **PTIME** in terms of space requirement. It uses special machines, which look almost like Turing machines but have a special way of handling parallelism. Before we can do that, we introduce yet another class of functions. We say that a function $f\colon A^* \to B^*$ is in **LOGSPACE** if it can be computed by a so called deterministic logarithmically bounded Turing machine. Here, a Turing machine is called **logarithmically space bounded** if it has $k+2$ tapes (where k may be 0) such that the length of the tapes number 1 through k is bounded by $\log_2|\vec{x}|$. Tape 0 serves as the input tape, Tape $k+1$ as the output tape. Tape 0 is read only, Tape $k+1$ is write only. At the end of the computation, T has to have written $f(\vec{x})$ onto that tape. (Actually, a moment's reflection shows that we may assume that the length of the intermediate tapes is bounded by $c\log_2|\vec{x}|$, for some $c > 0$, cf. also Theorem 1.98.) This means that if $\vec{x}$ has length 12 the tapes 2 to $k+1$ have length 3 since $3 < \log_2 12 < 4$. It need

not concern us further why this restriction makes sense. We shall see in Section 5.2 that it is well–motivated. We emphasize that $f(\vec{x})$ can be arbitrarily large. It is not restricted at all in its length, although we shall see later that the machine cannot compute outputs that are too long anyway. The reader may reflect on the fact that we may require the machine to use the last tape only in this way: it moves strictly to the right without ever looking at the previous cells again. Further, we can see to it that the intermediate tapes only contain single binary numbers.

Definition 5.7 *Let A^* be a finite alphabet and $L \subseteq A^*$. We say that L is in* ***LOGSPACE*** *if χ_L is deterministically* ***LOGSPACE****–computable.*

Theorem 5.8 *Let $f\colon A^* \to B^*$ be* ***LOGSPACE****–computable. Then f is in* ***PTIME****.*

Proof. We look at the configurations of the machine (see Definition 1.84). A configuration is defined with the exception of the output tape. It consists of the positions of the read head of the first tape and the content of the intermediate tapes plus the position of the read/write heads of the intermediate tapes. Thus the configurations are k–tuples of binary numbers of length $\leq c\log_2|\vec{x}|$, for some c. A position on a string likewise corresponds to a binary number. So we have $k+1$ binary numbers and there are at most

$$2^{(k+1)c\log_2|\vec{x}|} = |\vec{x}|^{c(k+1)} \tag{5.4}$$

of them. So the machine can calculate at most $|\vec{x}|^{c(k+1)}$ steps. For if there are more the machine is caught in a loop, and the computation does not terminate. Since this was excluded, there can be at most polynomially many steps. □

Since f is polynomially computable we immediately get that $|f(\vec{x})|$ is likewise polynomially bounded. This shows that a space bound implies a time bound. (In general, if $f(n)$ is the space bound then $c^{f(n)}$ is the corresponding time bound for a certain c.)

We have found a subclass of **PTIME** which is defined by its space consumption. Unfortunately, these classes cannot be shown to be equal. (It has not been disproved but is deeemed unlikely that they are equal.) We have to do much more work. For now, however, we remark the following.

Theorem 5.9 *Suppose that $f\colon A^* \to B^*$ and $g\colon B^* \to C^*$ are* ***LOGSPACE*** *computable. Then so is $g \circ f$.*

Proof. By assumption there is a logarithmically space bounded deterministic $k+2$–tape machine T which computes f and a logarithmically space bounded deterministic $\ell+2$–tape machine U which computes g. We cascade these machines in the following way. We use $k+\ell+3$ tapes, of which the first $k+2$ are the tapes of T and the last $\ell+2$ the tapes of U. We use Tape $k+1$ both as the output tape of T and as the input tape of U. The resulting machine is deterministic but not necessarily logarithmically space bounded. The problem is Tape $k+1$. However, we shall now demonstrate that this tape is not needed at all. For notice that T cannot but move forward on this tape and write on it, while U on the other hand can only progress to read the input. Now rather than having Tape k ready, it would be enough for U if it can access the symbol number i on the output tape of T on request. Clearly, as T can compute that symbol, U only needs to communicate the request to T by issuing i in binary. (This takes only logarithmic space. For we have $|f(\vec{x})| \leq p(|\vec{x}|)$ for some polynomial p for the length of the output computed by T, so we have $\log_2 |f(\vec{x})| \leq \lambda \log_2 |\vec{x}|$ for some natural number λ.) The proof follows once we make this observation: there is a machine T' that computes the ith symbol of the output tape of T, given the input for T input and i, using only logarithmic space. The rest is simple: everytime U needs a symbol, it calls T' issuing i in binary. The global input T' reads from U's input tape. □

This proof is the key to all following proofs. We shall now show that there is a certain class of problems which are, as one says, *complete* with respect to the class **PTIME** modulo **LOGSPACE**–reductions.

An n–**ary boolean function** is an arbitrary function $f\colon 2^n \to 2$. Every such function is contained in the polynomial clone of functions generated by the functions $\cup$, $\cap$ and $-$ (see Exercise 176). We shall now assume that f is composed from projections using the functions $\cap$, $\cup$, and $-$. For example, let $f(x_0,x_1,x_2) := -(-x_2 \cap (x_1 \cup x_0))$. Now for the variables x_0, x_1 and x_2 we insert concrete values (either 0 or 1). Which value does f have? This problem can clearly be solved in **PTIME**. However, the formulation of the problem is a delicate affair. Namely, we want to think of f not as a string, but as a network. (The difference is that in a network every subterm needs to be represented only once.) To write down networks, we shall have to develop a more elaborate coding. Networks are strings over the alphabet $W := \{\texttt{(},\texttt{)},\texttt{,},\texttt{0},\texttt{1},\texttt{a},\texttt{b},\wedge,\vee,\neg\}$. A **cell** is a string of the form $\texttt{(}\vec{\alpha}\texttt{,}\vec{\varepsilon}\texttt{,}\vec{\eta}\texttt{,}\vee\texttt{)}$, $\texttt{(}\vec{\alpha}\texttt{,}\vec{\varepsilon}\texttt{,}\vec{\eta}\texttt{,}\wedge\texttt{)}$, or of the form $\texttt{(}\vec{\alpha}\texttt{,}\vec{\varepsilon}\texttt{,}\neg\texttt{)}$, where $\vec{\alpha}$ is a binary sequence — written in the alphabet $\{\texttt{a},\texttt{b}\}$ —, and $\vec{\varepsilon},\vec{\eta}$ either are binary strings (written down using the letters a and b in place of 0 and 1) or a single symbol of the form 0

or 1. $\vec{\alpha}$ is called the **number** of the cell and $\vec{\varepsilon}$ and $\vec{\eta}$ the **argument key**, unless it is of the form 0 or 1. Further, we assume that the number represented by $\vec{\varepsilon}$ and $\vec{\eta}$ is smaller than the number represented by $\vec{\alpha}$. (This makes sure that there are no cycles.) A sequence of cells is called a **network** if (a) there are no two cells with identical number, (b) the numbers of cells are the numbers from 1 to a certain number ν, and (c) for every cell with argument key $\vec{\varepsilon}$ (or $\vec{\eta}$) there is a cell with number $\vec{\varepsilon}$ ($\vec{\eta}$). The cell with the highest number is called the **goal** of the network. Intuitively, a network defines a boolean function into which some constant values are inserted for the variables. This function shall be evaluated. With the cell number $\vec{\alpha}$ we associate a value $w_{\vec{x}}(\vec{\alpha})$ as follows.

$$w_{\vec{x}}(\vec{\alpha}) := \begin{cases} w_{\vec{x}}(\vec{\varepsilon}) \cup w_{\vec{x}}(\vec{\eta}) & \text{if } \vec{x} \text{ contains } (\vec{\alpha},\vec{\varepsilon},\vec{\eta},\vee), \\ w_{\vec{x}}(\vec{\varepsilon}) \cap w_{\vec{x}}(\vec{\eta}) & \text{if } \vec{x} \text{ contains } (\vec{\alpha},\vec{\varepsilon},\vec{\eta},\wedge), \\ -w_{\vec{x}}(\vec{\varepsilon}) & \text{if } \vec{x} \text{ contains } (\vec{\alpha},\vec{\varepsilon},\neg). \end{cases} \tag{5.5}$$

We write $w(\vec{\alpha})$ in place of $w_{\vec{x}}(\vec{\alpha})$. The value of the network is the value of its goal (incidentally the cell with number ν). Let $\xi\colon W^* \to \{0,1,\star\}$ be the following function. $\xi(\vec{x}) := \star$ if $\vec{x}$ is not a network. Otherwise, $\xi(\vec{x})$ is the value of $\vec{x}$. We wish to define a machine calculating ξ. We give an example. We want to evaluate $f(x_0,x_1,x_2) = -(-x_2 \cap (x_1 \cup x_0))$ for $x_0 := 0$, $x_1 := 1$ and $x_2 := 0$. Then we write down the following network:

(5.6) `(a,0,¬)(b,1,0,∨)(ba,a,b,∧)(bb,ba,¬)`

Now $w(\mathtt{a}) = -0 = 1$, $w(\mathtt{b}) = 1 \cup 0 = 1$, $w(\mathtt{ba}) = w(\mathtt{a}) \cap w(\mathtt{b}) = 1 \cap 1 = 1$, and $v(\vec{x}) = w(\mathtt{bb}) = -w(\mathtt{ba}) = -1 = 0$.

Lemma 5.10 *The set of all networks is in **LOGSPACE**.*

Proof. The verification is a somewhat longwinded matter but not difficult to do. To this end we shall have to run over the string several times in order to check the different criteria. The first condition is that no two cells have the same number. To check that we need the following: for every two positions i and j that begin a number, if $i \neq j$, then the numbers that start there are different. To compare the numbers means to compare the strings starting at these positions. (To do that requires to memorize only one symbol at a time, running back and forth between the strings.) This requires memorizing two further positions. However, a position takes only logarithmic space. □

Theorem 5.11 *ξ is in **PTIME**.*

Proof. Let $\vec{x}$ be given. First we compute whether $\vec{x}$ is a network. This computation is in **PTIME**. If $\vec{x}$ is not a network, output $\star$. If it is, we do the following. Moving up with the number k we compute the value of the cell number k. For each cell we have to memorize its value on a separate tape, storing pairs $(\vec{\alpha}, w(\vec{\alpha}))$ consisting of the name and the value of that cell. This can also be done in polynomial time. Once we have reached the cell with the highest number we are done. $\square$

It is not known whether the value of a network can be calculated in **LOGSPACE**. The problem is that we may not be able to bound the number of intermediate values. Now the following holds.

Theorem 5.12 *Let $f\colon A^* \to \{0,1\}$ be in **PTIME**. Then there exists a function $N\colon A^* \to W^*$ in **LOGSPACE** such that for every $\vec{x} \in A^*$ $N(\vec{x})$ is a network and $f(\vec{x}) = (\xi \circ N)(\vec{x})$.*

Proof. First we construct a network and then show that it is in **LOGSPACE**. By assumption there exist numbers k and c such that $f(\vec{x})$ is computable in $\rho := c \cdot |\vec{x}|^k$ time using a deterministic Turing machine T. We define a construction algorithm for a sequence $\mathcal{C}(\vec{x}) := (C(i,j))_{i,j}$, where $0 \leq i, j \leq c \cdot |\vec{x}|^k$. $\mathcal{C}(\vec{x})$ is ordered in the following way:

$$\begin{array}{l} C(0,0), C(0,1), C(0,2), \ldots, \\ C(1,0), C(1,1), C(1,2), \ldots, \\ C(2,0), C(2,1), C(2,2), \ldots \end{array} \tag{5.7}$$

$C(i,j)$ contains the following information: (a) the content of the jth cell of the tape of T at time point i, (b) information, whether the read head is on that cell at time point i, (c) if the read head is on this cell at i also the state of the automaton. This information needs bounded length. Call the bound λ. We denote by $C(i,j,k)$, $k < \lambda$, the kth binary digit of $C(i,j)$. $C(i+1,j)$ depends only on $C(i,j-1)$, $C(i,j)$ and $C(i,j+1)$. (T is deterministic. Moreover, we assumed that T works on a tape that is bounded to the left. See Exercise 43 that this is no loss of generality.) $C(0,j)$ are determined by $\vec{x}$ alone. (A) We have $C(i+1,j) = C(i,j)$ if either (A1) at i the head is not at $j-1$ or else did not move right, or (A2) at i the head is not at $j+1$ or else did not move left; (B) $C(i+1,j)$ can be computed from (B1) $C(i,j-1)$ if the head was at i positioned at j and moved right, (B2) $C(i,j)$ if the head was at i positioned at j and did not move, (B3) $C(i,j+1)$ if the head at i was positioned at $j+1$

and moved left. Hence, for every $k < \lambda$ there exist boolean functions f_L^k, f_M^k and f_R^k such that $f_L^k, f_R^k \colon \{0,1\}^{2\lambda} \to \{0,1\}^{\lambda}$, $f_M^k \colon \{0,1\}^{3\lambda} \to \{0,1\}^{\lambda}$, and

$$\begin{aligned} C(i+1,0,k) &= f_L(C(i,0),C(i,1)) \\ C(i+1,j,k) &= f_M(C(i,j-1),C(i,j),C(i,j+1)) \\ C(i+1,\rho,k) &= f_R(C(i,\rho-1),C(i,\rho)) \end{aligned} \tag{5.8}$$

These functions can be computed from T in time independent of $\vec{x}$. Moreover, we can compute sequences of cells that represent these functions. Basically, the network we have to construct results in replacing for every $i > 0$ and appropriate j the cell $C(i,j)$ by a sequence of cells calling on appropriate other cells to give the value $C(i,j)$. This sequence is obtained by adding a fixed number to each argument key of the cells of the sequence computing the boolean functions.

Now let $\vec{x}$ be given. We compute a sequence of cells $\gamma(i,j)$ corresponding to $C(i,j)$. The row $\gamma(0,j)$ is empty. Then ascending in i, the rows $\gamma(i,j)$ are computed and written on the output tape. If row i is computed, the following numbers are computed and remembered: the length of the ith row, the position of the read head at $i+1$ and the number of the first cell of $\gamma(i,j')$, where j' is the position of the read head at i. Now the machine writes down $C(i+1,j)$ with ascending j. This is done as follows. If j is not the position of the read head at $i+1$, the sequence is a sequence of cells that repeats the value of the cells of $\gamma(i,j)$. So, $\gamma(j+1,i,k) = (\vec{\alpha},\vec{\varepsilon},1,\wedge)$ for $\vec{\alpha}$ the number of the actual cell and $\vec{\varepsilon}$ is $\vec{\alpha}$ minus some appropriate number, which is computed from the length of the ith row the length of the sequence $\gamma(i,j')$ and the length of the sequence $\gamma(i+1,j)$. If j is the position of the read head, we have to insert more material, but basically it is a sequence shifted by some number, as discussed above. The number by which we shift can be computed in **LOGSPACE** from the numbers which we have remembered. Obviously, it can be decided on the basis of this computation when the machine T terminates on $\vec{x}$ and therefore when to stop the sequence. The last entry is the goal of the network. □

One also says that the problem of calculating the value of a network is complete with respect to the class **PTIME**. A network is **monotone** if it does not contain the symbol $\neg$.

Theorem 5.13 *There exists a **LOGSPACE**–computable function M which transforms an arbitrary network into a monotone network with identical value.*

The proof is longwinded but rather straightforward, so we shall only sketch it. Call $g : 2^n \to 2$ **monotone** if for every $\vec{x}$ and $\vec{y}$ such that $x_i \leq y_i$ for all $i < n$, $g(\vec{x}) \leq g(\vec{y})$. (We write $\vec{x} \leq \vec{y}$ if $x_i \leq y_i$ for all $i < n$.)

Lemma 5.14 *Let f be an n–ary boolean function. Then there exists a $2n$–ary boolean function g which is monotone such that*

$$f(\vec{x}) = g(x_0, -x_0, x_1, -x_1, \ldots, x_{n-1}, -x_{n-1}) \tag{5.9}$$

Theorem 5.15 *Every monotone boolean function is a polynomial function over $\cap$ and $\cup$.*

Proof. One direction is easy. $\cap$ and $\cup$ is monotone, and every composition of monotone functions is again monotone. For the other direction, let f be monotone. Let M be the set of minimal vectors $\vec{x}$ such that $f(\vec{x}) = 1$. For every vector $\vec{x}$, put $p(\vec{x}) := \bigcap_{x_i=1} x_i$. (If $\vec{x} = 0\ldots0$, then $p(\vec{x}) := \mathtt{1}$.) Finally, put

$$\rho_M := \bigcup_{\vec{x} \in M} p(\vec{x}) \tag{5.10}$$

If $M = \varnothing$, put $\rho_f := \mathtt{0}$. It is easily seen that $f(\vec{x}) = \rho_f(\vec{x})$ for all $\vec{x} \in 2^n$. $\square$

What is essential (and left for the reader to show) is that the map $f \mapsto g$ translates into a **LOGSPACE** computable map on networks. So, if g is an arbitrary **PTIME**–computable function from A^* to $\{0,1\}$, there exists a **LOGSPACE**–computable function N^+ constructing monotone networks such that $g(\vec{x}) = (\xi \circ N^+)(\vec{x})$ for all $\vec{x} \in A^*$.

Now we shall turn to the promised new type of machines.

Definition 5.16 *An **alternating Turing machine** is a sextuple*

$$\langle A, \mathtt{L}, Q, q_0, f, \gamma \rangle \tag{5.11}$$

*where $\langle A, \mathtt{L}, Q, q_0, f \rangle$ is a Turing machine and $\gamma\colon Q \to \{\wedge, \vee\}$ an arbitrary function. A state q is called **universal** if $\gamma(q) = \wedge$, and otherwise **existential**.*

We tacitly generalize the concepts of Turing machines to the alternating Turing machines (for example an *alternating k–tape Turing machine*, and a *logarithmically space bounded alternative Turing machine*). To this end one has to add the function γ in the definitions. Now we have to define when a Turing machine accepts an input $\vec{x}$. This is done via configurations. A configuration is said to be **accepted** by T if one of the following is the case:

- ☞ T is in an existential state and one of the immediately subsequent configurations is accepted by T.
- ☞ T is in a universal state and all immediately subsequent configurations are accepted by T.

Notice that the machine accepts a configuration that has no immediately subsequent configurations if (and only if) it is in a universal state. The difference between universal and existential states is effective if the machine is not deterministic. Then there can be several subsequent configurations. Acceptance by a Turing machine is defined as for an existential state if there is a successor state, otherwise like a universal state. If in a universal state, the machine must split itself into several copies that compute the various subsequent alternatives. Now we define **ALOGSPACE** to be the set of functions computable by a logarithmically space bounded alternating multitape Turing machine.

Theorem 5.17 (Chandra & Kozen & Stockmeyer)

$$\textit{ALOGSPACE} = \textit{PTIME}.$$

The theorem is almost proved. First, notice

Lemma 5.18 *LOGSPACE* $\subseteq$ *ALOGSPACE*.

For every deterministic logarithmically space bounded Turing machine also is an alternating machine by simply letting every state be universal. Likewise the following claim is easy to show, if we remind ourselves of the facts concerning **LOGSPACE**–computable functions.

Lemma 5.19 *Let $f\colon A^* \to B^*$ and $g\colon B^* \to C^*$ be functions. If f and g are in* ***ALOGSPACE****, so is $g \circ f$.*

Lemma 5.20 *ALOGSPACE* $\subseteq$ *PTIME*.

Also this proof is not hard. We already know that there are at most polynomially many configurations. The dependency between these configurations can also be checked in polynomial time. (Every configuration has a bounded number of successors. The bound only depends on T.) This yields a computation tree which can be determined in polynomial time. Now we must determine in the last step whether the machine accepts the initial configuration. To this end we must determine by induction on the depth in a computation tree

whether the respective configuration is accepted. This can be done as well in polynomial time. This completes the proof.

Now the converse inclusion remains to be shown. For this we use the following idea. Let f be in **PTIME**. We can write f as $\xi \circ N^+$ where N^+ is a monotone network computing f. As remarked above we can construct N^+ in **LOGSPACE** and in particular because of Lemma 5.18 in **ALOGSPACE**. It suffices to show that ξ is in **ALOGSPACE**. For then Lemma 5.19 gives us that $f = \xi \circ N^+ \in$ **ALOGSPACE**.

Lemma 5.21 $\xi \in$ ***ALOGSPACE***.

Proof. We construct a logarithmically space bounded alternating machine which for an arbitrary given monotone network $\vec{x}$ calculates its value $w(\vec{x})$. Let a network be given. First move to the goal. Descending from it compute as follows.

① If the cell contains $\wedge$ change into the universal state q_1. Else change into the existential state q_2. Goto ③.

② Choose an argument key $\vec{\alpha}$ of the current cell and go to the cell number $\vec{\alpha}$.

③ If $\vec{\alpha}$ is not an argument key go into state q_f if $\vec{\alpha} = \mathtt{1}$ and into q_g if $\vec{\alpha} = \mathtt{0}$. Here q_f is universal and q_g existential and there are no transitions defined from q_f and q_g.

All other states are universal; however, the machine works nondeterministically only in one case, namely if it gets the values of the arguments. Then it makes a nondeterministic choice. If the cell is an $\vee$–cell then it will accept that configuration if one argument has value 1, since the state is existential. If the cell is a $\wedge$–cell then it shall accept the configuration if both arguments have value 1 for now the state is universal. The last condition is the termination condition. If the string is not an argument key then it is either 0 or 1 and its value can be computed without recourse to other cells. If it is 1 the automaton changes into a final state which is universal and so the configuration is accepted. If the value is 0 the automaton changes into a final state which is existential and the configuration is rejected. □

Notes on this section. The gap between **PTIME** and **NPTIME** is believed to be a very big one, but it is not known whether the two really are distinct. The fact that virtually all languages are in **PTIME** is good news, telling

us that natural languages are tractable, at least syntactically. Concerning the tractability of languages as sign systems not very much is known, however.

Exercise 173. Show Proposition 5.3: *With L_1 and L_2 also $L_1 \cup L_2$, $L_1 \cap L_2$ as well as $A^* - L_1$ are in **PTIME**.*

Exercise 174. Show the following. *If L_1 and L_2 are in **PTIME** then so is $L_1 \cdot L_2$.*

Exercise 175. Show that L has the constant growth property if L is semilinear. Give an example of a language which has the constant growth property but is not semilinear.

Exercise 176. Let $f\colon 2^n \to 2$ a boolean function. Show that it can be obtained from the projections and the functions $-$, $\cup$, and $\cap$. *Hint.* Start with the functions $g_{\vec{x}}\colon 2^n \to 2$ such that $g_{\vec{x}}(\vec{y}) := 1$ iff $\vec{x} = \vec{y}$. Show that they can be generated from $-$ and $\cap$. Proceed to show that every boolean function is either the constant 0 or can be obtained from functions of type $g_{\vec{x}}$ using $\cup$.

Exercise 177. Prove Lemma 5.14.

Exercise 178. Call a language $L \subseteq A^*$ **weakly semilinear** if every intersection with a semilinear language $\subseteq A^*$ has the constant growth property. Show that every semilinear language is also weakly semilinear. Let $M := \{\mathtt{a}^m\mathtt{b}^n : n \geq 2^m\}$. Show that $M \subseteq \{\mathtt{a},\mathtt{b}\}^*$ is weakly semilinear but not semilinear.

Exercise 179. Show that every function $f\colon A^* \to B^*$ which is computable using an alternating Turing machine can also be computed using a Turing machine. (It is not a priori clear that the class of alternating Turing machines is not more powerful than the class of Turing machines. This has to be shown.)

2. Literal Movement Grammars

The concept of a literal movement grammar — **LMG** for short — has been introduced by Annius Groenink in [**?**] (see also [**?**]). With the help of these grammars one can characterize the **PTIME** languages by means of a generating device. The idea to this characterization goes back to a result by William Rounds [**?**]. Many grammar types turn out to be special subtypes of LMGs. The central feature of LMGs is that the rules contain a context free skeleton which describes the abstract structure of the string and in addition to this a description of the way in which the constituent is formed from the basic parts.

The notation is different from that of CFGs. In an LMG, nonterminals denote properties of strings and therefore one writes '$Q(\vec{x})$' in place of just 'Q'. The reason for this will soon become obvious. If $Q(\vec{x})$ obtains for a given string $\vec{x}$ we say that $\vec{x}$ has the property Q or that $\vec{x}$ **is a** Q**–string**. The properties play the role of the nonterminals in CFGs, but technically speaking they are handled differently. Since $\vec{x}$ is metavariable for strings, we now need another set of (official) variables for strings in the formulation of the LMGs. To this end we use the plain symbols x, y, z and so on (possibly with subscripts) for these variables. In addition to these variables there are also constants $\mathtt{a}$, $\mathtt{b}$, for the symbols of our alphabet A. We give a simple example of an LMG. It has two rules.

$$S(xx) \leftarrow S(x).; \qquad S(\mathtt{a}) \leftarrow . \tag{5.12}$$

These rules are written in Horn–clause format, as in Prolog, and they are exactly interpreted in the same way: the left hand side obtains with the variables instantiated to some term if the right hand obtains with the variables instantiated in the same way. So, the rules correspond to more familar looking formulae:

$$S(\mathtt{a}), \quad (\forall x)(S(x) \to S(x^\frown x)) \tag{5.13}$$

(Just reverse the arrow and interpret the comma as conjunction.)

Definition 5.22 *A formulae φ of predicate logic is called a* ***Horn–formula*** *iff it has the form*

$$(\forall x_0)\dots(\forall x_{q-1})(\bigwedge_{i<n} \chi_i \to \varphi) \tag{5.14}$$

where the χ_i, $i < n$, and φ are atomic formulae.

Here, it is assumed that only the variables x_i, $i < q$, occur in the χ_i and in φ. We abbreviate $(\forall x_0)\dots(\forall x_{p-1})$ by $(\forall \vec{x})$. Now, consider the case where the language has the following functional signature: for every letter from A a zeroary function symbol (denoted by the same letter), ε (zeroary) and $^\frown$ (binary). Further, assume the following set of axioms:

$$S_G := \{(\forall xyz)(x^\frown(y^\frown z) = (x^\frown y)^\frown z), \\ (\forall x)(\varepsilon^\frown x = x), (\forall x)(x^\frown \varepsilon = x)\} \tag{5.15}$$

Then a Horn–clause is of the form

(5.16) $$(\forall\vec{x})(U_0(s_0) \wedge U_1(s_1) \wedge \ldots \wedge U_{n-1}(s_{n-1}) \to T(t))$$

where t and the s_i ($i < n$) are string polynomials. This we shall write as

(5.17) $$T(t) \leftarrow U_0(s_0), U_1(s_1), \ldots, U_{n-1}(s_{n-1}).$$

Definition 5.23 *A **literal movement grammar**, or **LMG** for short, is a quintuple $G = \langle A, R, \Xi, S, H\rangle$, where A is the alphabet of terminal symbols, R a set of so–called **predicates**, $\Xi : R \to \omega$ a signature, $S \in R$ a distinguished symbol such that $\Xi(S) = 1$, and H a set of Horn–formulae in the language consisting of constants for every letter of A, the empty string, concatenation, and the relation symbols of R. $\vec{x}$ is a G–**sentence** iff $S(\vec{x})$ is derivable from H and S_G:*

(5.18) $$L(G) := \{\vec{x} : \ S_G; H \vdash S(\vec{x})\}$$

We call G a k–**LMG** if $\max\{\Xi(Q) : Q \in R\} \leq k$. The grammar above is a 1–LMG.

Proposition 5.24 $L(G) = \{\mathtt{a}^{2^n} : 0 \leq n\}$.

Proof. Surely $\mathtt{a} \in L(G)$. This settles the case $n = 0$. By induction one shows $\mathtt{a}^{2^n} \in L(G)$ for every $n > 0$. For if $\mathtt{a}^{2^n}$ is a string of category S so is $\mathtt{a}^{2^{n+1}} = \mathtt{a}^{2^n}{}^\frown\mathtt{a}^{2^n}$. This shows that $L(G) \supseteq \{\mathtt{a}^{2^n} : n \geq 0\}$. On the other hand this set satisfies the formula φ. For we have $\mathtt{a} \in L(G)$ and with $\vec{x} \in L(G)$ we also have $\vec{x}\vec{x} \in L(G)$. For if $\vec{x} = \mathtt{a}^{2^n}$ for a certain $n \geq 0$ then $\vec{x}\vec{x} = \mathtt{a}^{2^n \cdot 2} = \mathtt{a}^{2^{n+1}} \in L(G)$. □

There is an inductive definition of $L(G)$ by means of generation. We write $\vdash_G S(\vec{x})$ (vector arrow!), if either $S(\vec{x}) \leftarrow .$ is a rule or $\vec{x} = \vec{y}\vec{y}$ and $\vdash_G S(\vec{y})$. Both definitions define the same set of strings. Let us elaborate the notion of a 1–LMG in more detail. The maximum of all n such that G has an n–ary rule is called the **branching number of** G. In the rule

(5.19) $$S(xx) \leftarrow S(x).$$

we have $n = 1$ and $T = U_0 = S$, $t = xx$ and $s_0 = x$. In the rule

(5.20) $$S(\mathtt{a}) \leftarrow .$$

we have $n = 0$, $T = S$ and $t = \mathtt{a}$.

Definition 5.25 *Let $G = \langle A, R, \Xi, S, H\rangle$ be an LMG. Then we write $\Gamma \vdash_G^n \gamma$ iff $\Gamma; H; S_G \vdash^n \gamma$ in predicate logic and $\Gamma \vdash_G \gamma$ iff $\Gamma; H; S_G \vdash \gamma$ in predicate logic.*

We shall explain in some detail how we determine whether or not $\vdash_G^n Q(\vec{x})$ (Q unary). Call a **substitution** a function α which associates a term to each variable; and a **valuation** a function that associates a string in A^* to each string variable. Given α we define s^α for a polynomial by homomorphic extension. For example, if $s = \mathtt{a}x^2\,\mathtt{b}\,y$ and $\alpha(x) = \mathtt{ac}$, $\alpha(y) = \mathtt{bba}$ then $s^\alpha = \mathtt{aacacbbba}$, as is easily computed. Notice that strings can be seen as constant terms modulo equivalence, a fact that we shall exploit here by confusing valuations with substitutions that assign constant terms to the string variable. (The so–called **Herbrand–universe** is the set of constant terms. It is known that any Horn–formula that is not valid can be falsified in the Herbrand–universe, in this case $\mathfrak{Z}(A)$.)

(5.21) $$T(\vec{x}) \leftarrow U_0(\vec{y}_0), U_1(\vec{y}_1), \ldots, U_{m-1}(\vec{y}_{m-1}).$$

is an **instance** of the rule

(5.22) $$T(t) \leftarrow U_0(s_0), U_1(s_1), \ldots, U_{m-1}(s_{m-1}).$$

if there is a valuation β such that $\vec{x} = t^\beta$ and $\vec{y}_i = s_i^\beta$ for all $i < m$. Similarly,

(5.23) $$T(t') \leftarrow U_0(s'_0), U_1(s'_1), \ldots, U_{m-1}(s'_{m-1}).$$

is an instance of

(5.24) $$T(t) \leftarrow U_0(s_0), U_1(s_1), \ldots, U_{m-1}(s_{m-1}).$$

if there is a substitution α such that $s'_i = (s_i)^\alpha$ for all $i < m$ and $t' = t^\alpha$. The notion of generation by an LMG can be made somewhat more explicit.

Proposition 5.26 *(a) $\vdash_G^0 Q(\vec{x})$ iff $Q(\vec{x}) \leftarrow .$ is a ground instance of a rule of G. (b) $\vdash_G^{n+1} Q(\vec{x})$ iff $\vdash_G^n Q(\vec{x})$ or there is a number m, predicates R_i, $i < m$, and strings $\vec{y}_i$, $i < m$, such that*

① *$\vdash_G^n R_i(\vec{y}_i)$, and*

② *$Q(\vec{x}) \leftarrow R_0(\vec{y}_0), \ldots, R_{m-1}(\vec{y}_{m-1})$ is a ground instance of a rule of G.*

We shall give an example to illustrate these definitions. Let K be the following grammar.

(5.25) $\quad S(vxyz) \leftarrow S(vyxz).; \qquad S(\mathtt{abc}^\frown x) \leftarrow S(x).; \quad S(\varepsilon) \leftarrow .$

Then $L(K)$ is that language which contains all strings that contain an identical number of a, b and c. To this end one first shows that $(\mathtt{abc})^* \subseteq L(K)$ and in virtue of the first rule $L(K)$ is closed under permutations. Here $\vec{y}$ is a **permutation of** $\vec{x}$ if $\vec{y}$ and $\vec{x}$ have identical image under the Parikh map. Here is an example (the general case is left to the reader as an exercise). We can derive $S(\mathtt{abc})$ in one step from $S(\varepsilon)$ using the second rule, and $S(\mathtt{abcabc})$ in two steps, using again the second rule. In a third step we can derive $S(\mathtt{aabbcc})$ from this, using the first rule this time. Put $\alpha(v) := \mathtt{a}$, $\alpha(x) := \mathtt{ab}$, $\alpha(y) := \mathtt{bc}$ and $\alpha(z) := \mathtt{c}$. Then

(5.26) $\quad (vxyz)^\alpha = \mathtt{aabbcc}, \quad (vyxz)^\alpha = \mathtt{abcabc}$

Let H be a CFG. We define a 1–LMG $H^\spadesuit$ as follows. (For the presentation we shall assume that H is already in Chomsky normal form.) For every nonterminal A we introduce a unary predicate $\underline{A}$. The start symbol is $\underline{\mathtt{S}}$. If $A \to BC$ is a rule from H then $H^\spadesuit$ contains the rule

(5.27) $\quad \underline{A}(xy) \leftarrow \underline{B}(x), \underline{C}(y).$

If $A \to a$ is a terminal rule then we introduce the following rule into $H^\spadesuit$:

(5.28) $\quad \underline{A}(a) \leftarrow .$

One can show relatively easily that $L(H) = L(H^\spadesuit)$.

The 1–LMGs can therefore generate all CFLs. Additionally, they can generate languages without constant growth, as we have already seen. Let us note the following facts.

Theorem 5.27 *Let L_1 and L_2 be languages over A which can be generated by 1–LMGs. Then there exist 1–LMGs generating the languages $L_1 \cap L_2$ and $L_1 \cup L_2$.*

Proof. Let G_1 and G_2 be 1–LMGs which generate L_1 and L_2, respectively. We assume that the set of nonterminals of G_1 and G_2 are disjoint. Let S_i be the start predicate of G_i, $i \in \{1,2\}$. Let $H_\cup$ be constructed as follows. We

form the union of the nonterminals and rules of G_1 and G_2. Further, let $S^\heartsuit$ be a new predicate, which will be the start predicate of $G_\cup$. At the end we add the following rules: $S^\heartsuit(x) \leftarrow S_1(x).$; $S^\heartsuit(x) \leftarrow S_2(x)$. This defines $G_\cup$. $G_\cap$ is defined similarly, only that in place of the last two rules we have a single rule, $S^\heartsuit(x) \leftarrow S_1(x), S_2(x)$. It is easily checked that $L(G_\cup) = L_1 \cup L_2$ and $L(G_\cap) = L_1 \cap L_2$. We show this for $G_\cap$. We have $\vec{x} \in L(G_\cap)$ if there is an n with $\vdash^n_{G_\cap} S^\heartsuit(\vec{x})$. This in turn is the case exactly if $n > 0$ and $\vdash^{n-1}_{G_\cap} S_1(\vec{x})$ as well as $\vdash^{n-1}_{G_\cap} S_2(\vec{x})$. This is nothing but $\vdash^{n-1}_{G_1} S_1(\vec{x})$ and $\vdash^{n-1}_{G_2} S_1(\vec{x})$. Since n was arbitrary, we have $\vec{x} \in L(G_\cap)$ iff $\vec{x} \in L(G_1) = L_1$ and $\vec{x} \in L(G_2) = L_2$, as promised. □

The 1–LMGs are quite powerful, as the following theorem shows.

Theorem 5.28 *Let A be a finite alphabet and $L \subseteq A^*$. $L = L(G)$ for a 1–LMG iff L is recursively enumerable.*

The proof is left to the reader as an exercise. Since the set of recursively enumerable languages is closed under union and intersection, Theorem 5.27 already follows from Theorem 5.28. It also follows that the complement of a language that can be generated by a 1–LMG does not have to be such a language again. For the complement of a recursively enumerable language does not have to be recursively enumerable again. (Otherwise every recursively enumerable set would also be decidable, which is not the case.)

In order to arrive at interesting classes of languages we shall restrict the format of the rules. Let ρ be the following rule.

(5.29) $$\rho := T(t) \leftarrow U_0(s_0), U_1(s_1), \ldots, U_{n-1}(s_{n-1}).$$

- ☞ ρ is called **upward nondeleting** if every variable which occurs in one of the s_i, $i < n$, also occurs in t.
- ☞ ρ is called **upward linear** if no variable occurs more than once in t.
- ☞ ρ is called **downward nondeleting** if every variable which occurs in t also occurs in one of the s_i.
- ☞ ρ is called **downward linear** if none of the variables occurs twice in the s_i. (This means: the s_i are pairwise disjoint in their variables and no variable occurs twice in any of the s_i.)

- ☞ ρ is called **noncombinatorial** if the s_i are variables.
- ☞ ρ is called **simple** if it is noncombinatorial, upward nondeleting and upward linear.

G has the property $\mathcal{P}$ if all rules of G possess $\mathcal{P}$. In particular the type of simple grammars shall be of concern for us. The definitions are not always what one would intuitively expect. For example, the following rule is called upward nondeleting even though applying this rule means deleting a symbol: $\mathtt{U}(x) \leftarrow \mathtt{U}(x^\frown \mathtt{a})$. This is so since the definition focusses on the variables and ignores the constants. Further, downward linearity could alternatively be formulated as follows. One requires any symbol to occur in t as often as it occurs in the s_i taken together. This, however, is too strong a requirement. One would like to allow a variable to occur twice to the right even though on the left it occurs only once.

Lemma 5.29 *Let ρ be simple. Further, let*

$$Q(\vec{y}) \leftarrow R_0(\vec{x}_0), R_1(\vec{x}_1), \ldots, R_{n-1}(\vec{x}_{n-1}). \tag{5.30}$$

be an instance of ρ. Then $|\vec{y}| \geq \sum_{i<n} |\vec{x}_i| \geq \max\{|\vec{x}_i| : i < n\}$. Further, $\vec{x}_i$ is a subword of $\vec{y}$ for every $i < n$.

Theorem 5.30 *Let $L \subseteq A^*$ be generated by some simple 1–LMG. Then L is in **PTIME**.*

Proof. Let $\vec{x}$ be an arbitrary string and $n := \sharp N \cdot |\vec{x}|$. Because of Lemma 5.29 for every predicate Q: $\vdash_G Q(\vec{x})$ iff $\vdash_G^n Q(\vec{x})$. From this follows that every derivation of $S(\vec{x})$ has length at most n. Further, in a derivation there are only predicates of the form $Q(\vec{y})$ where $\vec{y}$ is a subword of $\vec{x}$. The following chart–algorithm (which is a modification of the standard chart–algorithm) only takes polynomial time:

- ☞ For $i = 0, \ldots, n$: For every substring $\vec{y}$ of length i and every predicate Q check if there are subwords $\vec{z}_j$, $j < p$, of length $< i$ and predicates R_j, $j < p$, such that $Q(\vec{y}) \leftarrow R_0(\vec{z}_0), R_1(\vec{z}_1), \ldots, R_{p-1}(\vec{z}_{p-1})$. is an instance of a rule of G.

The number of subwords of length i is proportional to n. For given p, a string of length n can be decomposed in $O(n^{p-1})$ ways as product of p (sub)strings.

Thus for every i, $O(n^p)$ many steps are required, in total $O(n^{p+1})$ on a deterministic multitape Turing machine. $\square$

The converse of Theorem 5.30 is in all likelihood false. Notice that in an LMG, predicates need not be unary. Instead, we have allowed predicates of any arity. There sometimes occurs the situation that one wishes to have uniform arity for all predicates. This can be arranged as follows. For an i–ary predicate A (where $i < k$) we introduce a k–ary predicate A^* which satisfies

$$A^*(x_0,\ldots,x_{k-1}) \leftrightarrow A(x_0,\ldots,x_{i-1}) \wedge \bigwedge_{j=i}^{k-1} x_j = \varepsilon. \tag{5.31}$$

There is a small difficulty in that the start predicate is required to be unary. So we lift also this restriction and allow the start predicate to have any arity. Then we put

$$L(G) := \{ \prod_{i<\Omega(S)} \vec{x}_i : \vdash_G S(\vec{x}_0,\ldots,\vec{x}_{\Omega(S)-1})\} \tag{5.32}$$

This does not change the generative power. An important class of LMGs, which we shall study in the sequel, is the class of simple LMGs. Notice that in virtue of the definition of a simple rule a variable is allowed to occur on the right hand side several times, while on the left it may not occur more than once. This restriction however turns out not to have any effect. Consider the following grammar H.

$$\begin{array}{llll} E(\varepsilon,\varepsilon) & \leftarrow & . & \\ E(a,a) & \leftarrow & . & (a \in A) \\ E(xa,ya) & \leftarrow & E(x,y). & (a \in A) \end{array} \tag{5.33}$$

It is easy to see that $\vdash_H E(\vec{x},\vec{y})$ iff $\vec{x} = \vec{y}$. H is simple. Now take a rule in which a variable occurs several times on the left hand side.

$$S(xx) \leftarrow S(x). \tag{5.34}$$

We replace this rule by the following one and add (5.33).

$$S(xy) \leftarrow S(x), \quad E(x,y). \tag{5.35}$$

This grammar is simple and generates the same strings. Furthermore, we can see to it that no variable occurs more than three times on the right hand side,

and that $s_i^j \neq s_k^j$ for $i \neq k$. Namely, replace s_i^j by distinct variables, say x_i^j, and add the clauses $E(x_i^j, x_{i'}^{j'})$, if $s_i^j = s_{i'}^{j'}$. We do not need to introduce all of these clauses. For each variable we only need two. (If we want to have $A_i = A_j$ for all $i < n$ we simply have to require $A_i = A_j$ for all $j \equiv i+1 \pmod{n}$.)

With some effort we can generalize Theorem 5.30.

Theorem 5.31 *Let $L \subseteq A^*$ be generated by a simple LMG. Then L is in **PTIME**.*

The main theorem of this section will be to show that the converse also holds. We shall make some preparatory remarks. We have already seen that **PTIME = ALOGSPACE**. Now we shall provide another characterization of this class. Let T be a Turing machine. We call T **read only** if none of its heads can write. If T has several tapes then it will get the input on all of its tapes. (A read only tape is otherwise useless.) Alternatively, we may think of the machine as having only one tape but several read heads that can be independently operated.

Definition 5.32 *Let $L \subseteq A^*$. We say that L is in **ARO** if there is an alternating read only Turing machine which accepts L.*

Theorem 5.33 ***ARO** = **ALOGSPACE**.*

Proof. Let $L \in$ **ARO**. Then there exists an alternating read only Turing machine T which accepts L. We have to find a logarithmically space bounded alternating Turing machine that recognizes L. The input and output tape remain, the other tapes are replaced by read and write tapes, which are initially empty. Now, let τ be a read only tape. The actions that can be performed on it are: moving the read head to the left or to the right (and reading the symbol). We code the position of the head using binary coding. Evidently, this coding needs only $\log_2 |\vec{x}| + 1$ space. Calculating the successor and predecessor (if it exists) of a binary number is **LOGSPACE** computable (given some extra tapes). Accessing the ith symbol of the input, where i is given in binary code, is as well. This shows that we can replace the read only tapes by logarithmically space bounded tapes. Hence $L \in$ **ALOGSPACE**. Suppose now that $L \in$ **ALOGSPACE**. Then $L = L(U)$ for an alternating, logarithmically space bounded Turing machine U. We shall construct a read only alternating Turing machine which accepts the same language. To this end we shall replace every intermediate Tape τ by several read only tapes which together perform

the same actions. Thus, all we need to show is that the following operations are computable on read only tapes (using enough auxiliary tapes). (For simplicity, we may assume that the alphabet on the intermediate tapes is just 0 and 1.) (a) Moving the head to the right, (b) moving the head to the left, (c) writing 0 onto the tape, (d) writing 1 onto the tape. Now, we must use at least two read only tapes; one, call it τ_a, contains the content of Tape τ, τ_b contains the position of the head of τ. The position i, being bounded by $\log_2 |\vec{x}|$, can be coded by placing the head on the cell number i. Call i_a the position of the head of τ_a, i_b the position of the head of τ_b. Arithmetically, these steps correspond to the following functions: (a) $i_b \mapsto i_b + 1$, (b) $i_b + 1 \mapsto i_b$, (c) replacing the i_bth symbol in the binary code of i_a by 0, (d) replacing the i_bth symbol in the binary code of i_a by 1. We must show that we can compute (c) and (d). (It is easy to see that if we can compute this number, we can reset the head of τ_b onto the position corresponding to that number.) (A) The i_bth symbol in the binary code of i_a is accessed as follows. We successively divide i_a by 2, exactly i_b times, throwing away the remainder. If the number is even, the result is 0, otherwise it is 1. (B) 2^{i_b} is computed by doubling 1 i_b times. So, (c) is performed as follows. First, check the i_bth digit in the representation. If it is 0, leave i_a unchanged. Otherwise, substract 2^{i_b}. Similarly for (d). This shows that we can find an alternating read only Turing machine that recognizes L. □

Now for the announced proof. Assume that L is in **PTIME**. Then we know that there is an alternating read only Turing machine which accepts L. This machine works with k tapes. For simplicity we shall assume that the machine can move only one head in a single step. We shall construct a $2k+2$–LMG G such that $L(G) = L$. Assume for each $a \in A$ two binary predicates, L^a and R^a, with the following rules.

(5.36) $\quad L^a(\varepsilon, a) \leftarrow . \qquad\qquad L^a(xc, yc) \leftarrow L^a(x, y).$

(5.37) $\quad R^a(\varepsilon, a) \leftarrow . \qquad\qquad R^a(cx, cy) \leftarrow R^a(x, y).$

It is easy to see that $L^a(\vec{x}, \vec{y})$ is derivable iff $\vec{y} = a\vec{x}$ and $R^a(\vec{x}, \vec{y})$ is derivable iff $\vec{y} = \vec{x}a$.

If $\vec{w}$ is the input we can code the position of a read head by a pair $\langle \vec{x}, \vec{y} \rangle$ for which $\vec{x}\vec{y} = \vec{w}$. A configuration is simply determined by naming the state of the machine and k pairs $\langle \vec{x}_i, \vec{y}_i \rangle$ with $\vec{x}_i \vec{y}_i = \vec{w}$. Our grammar will monitor the actions of the machine step by step. To every state q we associate a predicate $q^\star$. If q is existential the predicate is $2k+2$–ary. If q changes to r when reading the letter a and if the machine moves to the left on Tape j then the following

rule is added to G.

$$q^\star(w, x_j y_j, x_0, y_0, \ldots, x_{j-1}, y_{j-1}, x'_j, y'_j, x_{j+1}, y_{j+1}, \ldots, x_{k-1}, y_{k-1}) \leftarrow r^\star(w, w, x_0, y_0, \ldots, x_{k-1}, y_{k-1}), L^a(y_j, y'_j), R^a(x'_j, x_j). \tag{5.38}$$

If the machine moves the head to the right we instead add the following rule.

$$q^\star(w, x_j y_j, x_0, y_0, \ldots, x_{j-1}, y_{j-1}, x'_j, y'_j, x_{j+1}, y_{j+1}, \ldots, x_{k-1}, y_{k-1}) \leftarrow r^\star(w, w, x_0, y_0, \ldots, x_{k-1}, y_{k-1}), R^a(x_j, x'_j), L^a(y'_j, y_j). \tag{5.39}$$

If the machine does not move the head, then the following rule is added.

$$q^\star(w, w', x_0, y_0, \ldots, x_{k-1}, y_{k-1}) \leftarrow r^\star(w, w', x_0, y_0, \ldots, x_{k-1}, y_{k-1}). \tag{5.40}$$

Notice that the first two argument places of the predicate are used to get rid of 'superfluous' variables. If the state q is universal and if there are exactly p transitions with successor states r_i, $i < p$, which do not need to be different, then $q^\star$ becomes $2k+2$–ary and we introduce symbols $q(i)^\star$, $i < p$, which are $2k+2$–ary. Now, first the following rule is introduced.

$$\begin{aligned} q^\star(w, w', x_0, y_0, \ldots, x_{k-1}, y_{k-1}) \leftarrow & \\ & q(0)^\star(w, w', x_0, y_0, \ldots, x_{k-1}, y_{k-1}), \\ & q(1)^\star(w, w', x_0, y_0, \ldots, x_{k-1}, y_{k-1}), \\ & \ldots, \\ & q(p-1)^\star(w, w', x_0, y_0, \ldots, x_{k-1}, y_{k-1}). \end{aligned} \tag{5.41}$$

Second, if the transition i consists in the state q changing to r_i when reading the symbol a and if the machine moves to the left on Tape j, G gets (5.38) with $q^\star(i)$ in place $q^\star$ and $r^\star_j$ in place of $r^\star$. If movement is to the right, instead we use (5.39). If the machine does not move the head, then (5.40) is added.

All of these rules are simple. If q is an accepting state, then we also take the following rule on board.

$$q^\star(w, w', x_0, y_0, \ldots, x_{k-1}, y_{k-1}) \leftarrow . \tag{5.42}$$

The last rule we need is

$$S(w) \leftarrow q^\star_0(w, w, \varepsilon, w, \varepsilon, w, \ldots, \varepsilon, w). \tag{5.43}$$

This is a simple rule. For the variable w occurs to the left only once. With this definition made we have to show that $L(G) = L$. Since $L = L(T)$ it suffices to show that $L(G) = L(T)$. We have $\vec{w} \in L(T)$ if there is an $n \in \omega$ such that T moves into an accepting state from the initial configuration for $\vec{w}$. Here the initial configuration is as follows. On all tapes we have $\vec{w}$ and the read heads are to the left of the input. An end configuration is a configuration from which no further moves are possible. It is accepted if the machine is in a universal state.

We say that $\zeta := q^\star(\vec{w}, \vec{w}, \vec{x}_0, \vec{y}_0, \ldots, \vec{x}_{k-1}, \vec{y}_{k-1})$ codes the configuration ζ^K where T is in state q, and for each $i < k$ (a) Tape i is filled by $x_i y_i$, and (b) the read head Tape i is on the symbol immediately following $\vec{x}_i$. Now we have:

① $\vdash_G^0 \zeta$ iff ζ^K is an accepting end configuration.

② If q is existential then $\zeta \leftarrow \eta$ is an instance of a rule of G iff T computes η^K from ζ^K in one step.

③ If q is universal then ζ is derivable from η_i, $i < p$, in two rule steps iff T computes the transitions $\zeta^K \to \eta_i^K$ $(i < p)$.

Let $\vec{w} \in L(G)$. This means that $\vdash_G^n S(\vec{w})$ and so

$$\vdash_G^{n-1} \zeta := q_0^\star(\vec{w}, \vec{w}, \varepsilon, \vec{w}, \varepsilon, \vec{w}, \ldots, \varepsilon, \vec{w}). \tag{5.44}$$

This corresponds to the initial configuration of T for the input $\vec{w}$. We conclude from what we have said above that if $\vdash_G^{n-1} \zeta$ there exists a $k \leq n$ such that T accepts ζ^K in k steps. Furthermore: if T accepts ζ^K in k steps, then $\vdash_G^{2k} \zeta$. Hence we have $L(G) = L(T)$.

Theorem 5.34 (Groenink) *L is accepted by a simple LMG iff $L \in$ **PTIME**.*

Notes on this section. LMGs are identical to **Elementary Formal Systems** (**EFS**s) defined in [**?**], Page 4. Smullyan used them to define recursion without the help of a machine. [**?**] did the same, however his rules are more akin to actions of a Turing machine than to rules of an EFS (or an LMG). There is an alternative characterization of **PTIME**–languages. Let $\mathsf{FOL(LFP)}$ be the expansion of first–order predicate logic (with constants for each letter and a single binary symbol $<$ in addition to equality) by the least–fixed point operator. Then the **PTIME**–languages are exactly those that can be defined in $\mathsf{FOL(LFP)}$. A proof can be found in [**?**].

Exercise 180. Prove Theorem 5.28. *Hint.* You have to simulate the actions of a Turing machine by the grammar. Here we code the configuration by means of the string, the states by means of the predicates.

Exercise 181. Prove Theorem 5.31.

Exercise 182. Construct a simple 1–LMG G such that $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n : n \in \omega\} = L(G)$.

Exercise 183. Let $G = \langle A, R, \Xi, S, H\rangle$ be an LMG which generates L. Furthermore, let U be the language of all $\vec{x}$ whose Parikh image is that of some $\vec{y} \in L$. (In other words: U is the permutation closure of L.) Let

$$G^p := \langle A, R \cup \{S^\heartsuit\}, \Xi^\heartsuit, S^\heartsuit, H^p\rangle \tag{5.45}$$

where $\Xi(S^\heartsuit) = 1$, and let

$$H^p := R \cup \{S^\heartsuit(x) \leftarrow S(x).; S^\heartsuit(vyxz) \leftarrow S^\heartsuit(vxyz).\} \tag{5.46}$$

Show that $L(G^p) = U$.

Exercise 184. Let L be the set of all theorems of intuitionistic logic. Write a 1–LMG that generates this set. *Hint.* You may use the Hilbert–style calculus here.

3. Interpreted LMGs

In this section we shall concern ourselves with interpreted LMGs. The basic idea behind interpreted LMGs is quite simple. Every rule is connected with a function which tells us how the meanings of the elements on the right hand side are used to construct the meaning of the item on the left. We shall give an example. The following grammar generates — as we have shown above — the language $\{\mathtt{a}^{2^n} : n \geq 0\}$.

$$S(xx) \leftarrow S(x). \qquad S(\mathtt{a}) \leftarrow . \tag{5.47}$$

We write a grammar which generates all pairs $\langle \mathtt{a}^{2^n}, n\rangle$. So, we take the number n to be the meaning of the string $\mathtt{a}^{2^n}$. For the first rule we choose the function $\lambda n.n+1$ as the meaning function and for the second the constant 0. We shall adapt the notation to the one used previously and write as follows.

$$\mathtt{aaaa} : S : 2 \qquad \text{or} \qquad \langle \mathtt{aaaa}, S, 2\rangle \tag{5.48}$$

Both notations will be used concurrently. (5.48) names a sign with exponent aaaa with category (or predicate) S and with meaning 2. The rules of the above grammar are written as follows:

$$\langle xx, S, n+1\rangle \leftarrow \langle x, S, n\rangle. \qquad \langle \mathtt{a}, S, 0\rangle \leftarrow . \tag{5.49}$$

This grammar is easily transformed into a sign grammar. We define a 0–ary mode $\mathtt{A}_0$ and a unary mode $\mathtt{A}_1$.

$$\begin{aligned} \mathtt{A}_0 &:= \langle \mathtt{a}, S, 0\rangle, \\ \mathtt{A}_1(\langle x, S, n\rangle) &:= \langle xx, S, n+1\rangle. \end{aligned} \tag{5.50}$$

The structure term $\mathtt{A}_1\mathtt{A}_1\mathtt{A}_1\mathtt{A}_0$ for example defines the sign $\langle \mathtt{a}^8, S, 3\rangle$.

It seems that one can always define a sign grammar from a LMGs in this way. However, this is not so. Consider adding the following rule to (5.47).

$$\langle x\mathtt{aby}, S, 3n\rangle \leftarrow \langle x\mathtt{aay}, S, n\rangle. \tag{5.51}$$

The problem with this rule is that the left hand side is not uniquely determined by the right hand side. For example, from $\langle \mathtt{aaaa}, S, 2\rangle$ we can derive in one step $\langle \mathtt{abaa}, S, 6\rangle$ as well as $\langle \mathtt{aaba}, S, 6\rangle$ and $\langle \mathtt{aaab}, S, 6\rangle$. We shall therefore agree on the following.

Definition 5.35 *Let*

$$\begin{aligned} \rho = T(t^0, t^1, \ldots, t^{p-1}) &\leftarrow U_0(s_0^0, s_0^1, \ldots, s_0^{q_0-1}), \\ & U_1(s_1^0, s_1^1, \ldots, s_1^{q_1-1}), \ldots, U_{n-1}(s_{n-1}^0, s_{n-1}^1, \ldots, s_{n-1}^{q_n-1}) \end{aligned} \tag{5.52}$$

*be a rule. ρ is called **definite** if for all instances of the rule the following holds: For all α, if the $(s_i^j)^\alpha$ are given, the $(t^j)^\alpha$ are uniquely determined. An LMG is called **definite** if each of its rules is definite.*

Clearly, to be able to transform an LMG into a sign grammar we need that it is definite. However, this is still a very general concept. Hence we shall restrict our attention to simple LMGs. They are definite, as is easily seen. These grammars have the advantage that the s_i^j are variables over strings and the t^j polynomials. We can therefore write them in λ–notation. Our grammar can therefore be specified as follows.

$$\begin{aligned} \mathtt{A}_0 &:= \langle \mathtt{a}, S, 0\rangle \\ \mathtt{A}_1 &:= \langle \lambda x.x^\frown x, S, \lambda n.n+1\rangle \end{aligned} \tag{5.53}$$

In certain cases the situation is not so simple. For this specification only works if a variable of the right hand side occurs there only once. If it occurs several times, we cannot regard the t^j as polynomials using concatenation. Namely, they are partial, as is easily seen. An easy example is provided by the following rule.

$$C(x) \leftarrow A(x), B(x). \tag{5.54}$$

Intuitively, one would choose $\lambda x.x$ for the string function; however, how does one ensure that the two strings on the right hand side are equal? For suppose we were to introduce a binary mode $\mathtt{C}$.

$$\mathtt{C}(\langle \vec{x}, \alpha, X\rangle, \langle \vec{y}, \beta, Y\rangle) := \langle \mathtt{C}^{\varepsilon}(\vec{x}, \vec{y}), \mathtt{C}^{\tau}(\alpha, \beta), \mathtt{C}^{\mu}(X, Y)\rangle \tag{5.55}$$

Then we must ensure that $\mathtt{C}^{\varepsilon}(\vec{x}, \vec{y})$ is only defined if $\vec{x} = \vec{y}$. So in addition to concatenation on A^* we also have to have a binary operation ι, which is defined as follows.

$$\iota(\vec{x}, \vec{y}) := \begin{cases} \vec{x} & \text{if } \vec{x} = \vec{y}, \\ \star & \text{otherwise.} \end{cases} \tag{5.56}$$

With the help of this operation we can transform the rule into a binary mode. Then we simply put $\mathtt{C}^{\varepsilon} := \lambda x.\lambda y.\iota(x, y)$.

We shall try out our concepts by giving a few examples. Let $\vec{x} \in \{\mathtt{L}, \mathtt{O}\}^*$ be a binary sequence. This is the binary code $n^\flat$ of a natural number n. This binary sequence we shall take as the meaning of the same number in Turing code. For the number n it is the sequence $n^\sharp := \mathtt{a}^{n+1}$. Here is a grammar for the language $\{\langle n^\sharp, S, n^\flat\rangle : n \in \omega\}$.

$$\begin{aligned} \langle x\,\mathtt{a}, S, n\rangle &\leftarrow \langle x, T, n\rangle. \\ \langle xx\,\mathtt{a}, T, n^\frown\mathtt{L}\rangle &\leftarrow \langle x, T, n\rangle. \\ \langle xx, T, n^\frown\mathtt{O}\rangle &\leftarrow \langle x, T, n\rangle. \\ \langle \varepsilon, T, \varepsilon\rangle &\leftarrow . \end{aligned} \tag{5.57}$$

Notice that the meanings are likewise computed using concatenation. In place of $\lambda n.2n$ or $\lambda n.2n+1$ we therefore have $\lambda x.x^\frown\mathtt{O}$ and $\lambda x.x^\frown\mathtt{L}$.

We can also write a grammar which transforms binary codes into Turing codes, by simply exchanging exponent and meaning.

A somewhat more complex example is a grammar which derives triples $\langle \vec{x}\otimes\vec{y}, S, \vec{z}\rangle$ of binary numbers where $\vec{z}$ is the binary code of the sum of the numbers represented by $\vec{x}$ and $\vec{y}$. (The symbol $\otimes$ serves to separate $\vec{x}$ from $\vec{y}$.)

(5.58a)
$$\begin{aligned}
\langle x\otimes y, S, z\rangle &\leftarrow \langle x\otimes y, A, z\rangle.\\
\langle \mathtt{0}x\otimes y, S, z\rangle &\leftarrow \langle x\otimes y, S, z\rangle.\\
\langle x\otimes \mathtt{0}y, S, z\rangle &\leftarrow \langle x\otimes y, S, z\rangle.\\
\langle x\otimes y, S, \mathtt{0}z\rangle &\leftarrow \langle x\otimes y, S, z\rangle.\\
\langle x\otimes y, S, \mathtt{L}z\rangle &\leftarrow \langle x\otimes y, U, z\rangle.
\end{aligned}$$

(5.58b)
$$\begin{aligned}
\langle \mathtt{0}x\otimes \mathtt{0}y, A, \mathtt{0}z\rangle &\leftarrow \langle x\otimes y, A, z\rangle.\\
\langle \mathtt{0}x\otimes \mathtt{0}y, A, \mathtt{L}z\rangle &\leftarrow \langle x\otimes y, U, z\rangle.\\
\langle \mathtt{0}x\otimes \mathtt{L}y, U, \mathtt{0}z\rangle &\leftarrow \langle x\otimes y, U, z\rangle.\\
\langle \mathtt{0}x\otimes \mathtt{L}y, A, \mathtt{L}z\rangle &\leftarrow \langle x\otimes y, A, z\rangle.\\
\langle \mathtt{L}x\otimes \mathtt{0}y, U, \mathtt{0}z\rangle &\leftarrow \langle x\otimes y, U, z\rangle.\\
\langle \mathtt{L}x\otimes \mathtt{0}y, A, \mathtt{L}z\rangle &\leftarrow \langle x\otimes y, A, z\rangle.\\
\langle \mathtt{L}x\otimes \mathtt{L}y, U, \mathtt{0}z\rangle &\leftarrow \langle x\otimes y, A, z\rangle.\\
\langle \mathtt{L}x\otimes \mathtt{L}y, U, \mathtt{L}z\rangle &\leftarrow \langle x\otimes y, U, z\rangle.
\end{aligned}$$

(5.58c)
$$\begin{aligned}
\langle \mathtt{0}\otimes\mathtt{0}, A, \mathtt{0}\rangle &\leftarrow . & \langle \mathtt{0}\otimes\mathtt{L}, A, \mathtt{L}\rangle &\leftarrow .\\
\langle \mathtt{L}\otimes\mathtt{0}, A, \mathtt{L}\rangle &\leftarrow . & \langle \mathtt{L}\otimes\mathtt{L}, U, \mathtt{0}\rangle &\leftarrow .
\end{aligned}$$

Now let us return to the specification of interpreted LMGs. First of all we shall ask how LMGs can be interpreted to become sign grammars. To this end we have to reconsider our notion of an exponent. Up to now we have assumed that exponents are strings. Now we have to assume that they are sequences of strings (we say rather 'vectors of strings', since strings are themselves sequences). This motivates the following definition.

Definition 5.36 *Let A be a finite set. We denote by $V(A) := \bigcup_{k<\omega}(A^*)^k$ the set of vectors of strings over A. Furthermore, let $F^V := \{\varepsilon, 0, \frown, \otimes, \triangleright, \triangleleft, \zeta, \iota\}$, $\Omega(\frown) = \Omega(\otimes) = \Omega(\iota) = 2$, $\Omega(\triangleright) = \Omega(\triangleleft) = \Omega(\zeta) = 1$; $\Omega(\varepsilon) = \Omega(0) = 0$. Here, the following is assumed to hold. (Strings are denoted by vector arrows, while $\mathfrak{x}$, $\mathfrak{y}$ and $\mathfrak{z}$ range over $V(A)$.)*

① $\mathfrak{z}\otimes(\mathfrak{y}\otimes\mathfrak{z}) = (\mathfrak{z}\otimes\mathfrak{y})\otimes\mathfrak{z}$

② $0 = \langle\,\rangle$ *is the empty sequence.*

③ $^\frown$ *is the usual concatenation of strings, so it is not defined on vectors of length* $\neq 1$.

④ ε *is the empty string.*

⑤ $\triangleright\bigotimes_{i<m}\vec{x} = \bigotimes_{i<m-1}\vec{x}_i$, *and* $\triangleleft\bigotimes_{i<m}\vec{x}_i = \bigotimes_{0<i<m}\vec{x}_i$.

⑥ $\zeta(\mathfrak{x}) = \star$ *if* $\mathfrak{x}$ *is not a string;* $\zeta(\vec{x}) = \vec{x}$ *otherwise.*

⑦ $\iota(\mathfrak{x},\mathfrak{y}) = \mathfrak{x}$ *if* $\mathfrak{x} = \mathfrak{y}$ *and* $\iota(\mathfrak{x},\mathfrak{y}) = \star$ *otherwise.*

The resulting (partial) algebra is called the ***algebra of string vectors over*** A *and is denoted by* $\mathfrak{V}(A)$.

In this algebra the following laws hold among other.

$$\begin{aligned}
\mathfrak{x}\otimes 0 &= \mathfrak{x}\\
0\otimes\mathfrak{x} &= \mathfrak{x}\\
\mathfrak{x}\otimes(\mathfrak{y}\otimes\mathfrak{z}) &= (\mathfrak{x}\otimes\mathfrak{y})\otimes\mathfrak{z}\\
\triangleright(\mathfrak{x}\otimes\zeta(\mathfrak{y})) &= \mathfrak{x}\\
\triangleleft(\zeta(\mathfrak{y})\otimes\mathfrak{x}) &= \mathfrak{x}
\end{aligned} \tag{5.59}$$

The fourth and fifth equation hold under the condition that $\zeta(\mathfrak{y})$ is defined. A vector $\mathfrak{x}$ has length m if $\zeta(\triangleright^{m-1}\mathfrak{x})$ is defined but $\zeta(\triangleright^{m}\mathfrak{x})$ is not. In this case $\zeta(\triangleright^{m-(i+1)}\triangleleft^{i}\mathfrak{x})$ is defined for all $i < m$ and they are the projection functions. Now we have:

$$\mathfrak{x} = \triangleright^{m-1}\mathfrak{x}\otimes\triangleright^{m-2}\triangleleft\mathfrak{x}\otimes\cdots\otimes\triangleright\triangleleft^{m-2}\mathfrak{x}\otimes\triangleleft^{m-1}\mathfrak{x}. \tag{5.60}$$

All polynomial functions that appear in the sequel can be defined in this algebra. The basis is the following theorem.

Theorem 5.37 *Let* $p\colon (A^*)^m \to A^*$ *be a function which is a polynomial in* $^\frown$ *and* ι. *Then there exists a vector polynomial* $\mathfrak{q}\colon V(A) \to V(A)$ *such that*

① $\mathfrak{q}(\mathfrak{x})$ *is defined only if* $\mathfrak{x} \in (A^*)^m$.

② *If* $\mathfrak{x} \in (A^*)^m$ *and* $\mathfrak{x} = \langle \vec{x}_i : i < m\rangle$ *then* $\mathfrak{q}(\mathfrak{x}) = p(\vec{x}_0,\ldots,\vec{x}_{m-1})$.

Proof. Let p be given. We assume that one of the variables appears at least once. (Otherwise $p = \varepsilon$ and then we put $q := \varepsilon$.) Let q arise from p by replacement of x_i by $\zeta(\triangleright^{m-(i+1)}\triangleleft^{i}\mathfrak{x})$, for all $i < m$. This defines q. (It is well

defined, for the symbols ε, $^\frown$, ι are in the signature F^V.) Let now $\mathfrak{x}$ be given. As remarked above, q is defined on $\mathfrak{x}$ only if $\mathfrak{x}$ has length m. In this case $\mathfrak{x} = \langle \vec{x}_i : i < n \rangle$ for certain $\vec{x}_i$, and we have $\vec{x}_i = \zeta(\triangleright^{m-(i+1)} \triangleleft^i \mathfrak{x})$. Since the symbols ε, $^\frown$ and ι coincide on the strings in both algebras (that of the strings and that of the vectors) we have $q(\mathfrak{x}) = q(\vec{x}_0, \ldots, \vec{x}_{m-1})$. $\square$

That $p\colon (A^*)^m \to (A^*)^n$ is a polynomial function means that there exist polynomials p_i, $i < n$, such that

$$p(\vec{x}_0, \ldots, \vec{x}_{m-1}) = \langle p_i(\vec{x}_0, \ldots, \vec{x}_{m-1}) : i < n \rangle. \tag{5.61}$$

We can therefore replace the polynomials on strings by polynomials over vectors of strings. Thise simplifies the presentation of LMGs considerably. We can now write down a rule as follows.

$$\begin{aligned} &\langle q(\mathfrak{x}_0, \ldots, \mathfrak{x}_{m-1}), A, f(X_0, \ldots, X_{m-1}) \rangle \\ &\qquad \leftarrow \langle \mathfrak{x}_0, B_0, X_0 \rangle, \ldots, \langle \mathfrak{x}_{m-1}, B_{m-1}, X_{m-1} \rangle. \end{aligned} \tag{5.62}$$

We shall make a further step and consider LMGs as categorial grammars. To this end we shall first go over to Chomsky Normal Form. This actually brings up a surprise. For there are k–LMGs for which no k–LMG in Chomsky Normal Form can be produced (see the exercises). However, there exists a k'–LMG in Chomsky Normal Form for a certain effectively determinable $k' \leq \pi k$, where π is the maximal productivity of a rule. Namely, look at a rule. We introduce new symbols Z_i, $i < m-2$, and replace this rule by the following rules.

$$\begin{aligned} &\langle \mathfrak{x}_0 \otimes \mathfrak{x}_1, Z_0, X_0 \times X_1 \rangle \quad \leftarrow \quad \langle \mathfrak{x}_0, B_0, X_0 \rangle, \quad \langle \mathfrak{x}_1, B_1, X_1 \rangle. \\ &\langle \mathfrak{y}_0 \otimes \mathfrak{x}_2, Z_1, Y_0 \times X_2 \rangle \quad \leftarrow \quad \langle \mathfrak{y}_0, Z_0, Y_0 \rangle, \quad \langle \mathfrak{x}_2, B_2, X_2 \rangle. \\ &\ldots \\ &\langle \mathfrak{y}_{m-4} \otimes \mathfrak{x}_{m-2}, Z_{m-3}, Y_{m-4} \times X_{m-2} \rangle \\ &\qquad \leftarrow \quad \langle \mathfrak{y}_{m-4}, Z_{m-4}, Y_{m-4} \rangle \quad \langle \mathfrak{x}_{m-2}, B_{m-2}, X_{m-2} \rangle. \\ &\langle q^*(\mathfrak{y}_{m-3} \otimes \mathfrak{x}_{m-1}), A, f^*(Y_{m-3} \times X_{m-1}) \rangle \\ &\qquad \leftarrow \quad \langle \mathfrak{y}_{m-3}, Z_{m-3}, Y_{m-3} \rangle, \quad \langle \mathfrak{x}_{m-1}, B_{m-1}, X_{m-1} \rangle. \end{aligned} \tag{5.63}$$

Here q^* and f^* are chosen in such a way that

$$\begin{aligned} q^*(\mathfrak{x}_0 \otimes \cdots \otimes \mathfrak{x}_{m-1}) &= q(\mathfrak{x}_0, \ldots, \mathfrak{x}_{m-1}) \\ f^*(X_0 \times \cdots \times X_{m-1}) &= f(X_0, \ldots, X_{m-1}) \end{aligned} \tag{5.64}$$

It is not hard to see how to define the functions by polynomials. Hence, in the sequel we may assume that we have at most binary branching rules. 0–ary rules are the terminal rules. A unary rule has the following form.

(5.65) $\langle q(\mathfrak{x}), C, f(X)\rangle \leftarrow \langle \mathfrak{x}, A, X\rangle.$

We keep the sign $\langle \mathfrak{x}, A, X\rangle$ and introduce a new sign $\mathtt{Z}_\rho$ which has the following form.

(5.66) $\mathtt{Z}_\rho := \langle \lambda\mathfrak{x}.q(\mathfrak{x}), C/A, \lambda x.f(x)\rangle$

There is only one binary mode, $\mathtt{C}$, which is defined thus:

(5.67) $\mathtt{C}(\langle p, A, X\rangle, \langle q, B, Y\rangle) := \langle p(q), A\cdot B, (XY)\rangle$

This is exactly the scheme of application in categorial grammar. One difference remains. The polynomial p is not necessarily concatenation. Furthermore, we do not have to distinguish between two modes, since we in the string case we have the possibility of putting p to be either $\lambda x.x^\frown\vec{y}$ or $\lambda x.\vec{y}^\frown x$. Application has in this way become independent of the accidental order. Many more operations can be put here, for example reduplication. The grammar that we have mentioned at the beginning of the section is defined by the following two modes.

(5.68)
$$\mathtt{D}_0 := \langle \mathtt{a}, S, 0\rangle$$
$$\mathtt{D}_1 := \langle \lambda x.x^\frown x, S/S, \lambda n.n+1\rangle$$

To the previous structure term $\mathtt{A}_1\mathtt{A}_1\mathtt{A}_1\mathtt{A}_0$ now corresponds the structure term

(5.69) $\mathtt{CD}_1\mathtt{CD}_1\mathtt{CD}_1\mathtt{D}_0$

In this way the grammar has become an AB–grammar, with one exception: the treatment of strings must be explicitly defined.

The binary rules remain. A binary rule has the following form.

(5.70) $\langle q(\mathfrak{x},\mathfrak{y}), C, f(X,Y)\rangle \leftarrow \langle \mathfrak{x}, A, X\rangle, \langle \mathfrak{y}, B, Y\rangle.$

We keep the sign on the right hand side and introduce a new sign.

(5.71) $\mathtt{Z}_\rho := \langle \lambda\mathfrak{y}.\lambda\mathfrak{x}.q(\mathfrak{x},\mathfrak{y}), (C/A)/B, \lambda y.\lambda x.f(x,y)\rangle$

Table 15. Arabic Binyanim: `ktb`

I	`katab`	to write
II	`kattab`	to make write
III	`kaatab`	to correspond
VI	`takaatab`	to write to each other
VIII	`ktatab`	to write, to be inscribed

	Perf. Act.	Perf. Pass.	Impf. Act.	Impf. Pass.
I	`katab`	`kutib`	`aktub`	`uktab`
II	`kattab`	`kuttib`	`ukattib`	`ukattab`
III	`kaatab`	`kuutib`	`ukaatib`	`ukaatab`
VI	`takaatab`	`tukuutib`	`atakattab`	`utakattab`
VIII	`ktatab`	`ktutib`	`aktatib`	`uktatab`

Exercise 185. Show that for any $k > 1$ there are simple k–LMGs G with branching number 3 such that for no simple k–LMG H with branching number 2, $L(G) = L(H)$.

Exercise 186. Here are some facts from Arabic. In Arabic a root typically consists of three consonants. Examples are `ktb` 'to write', `drs` 'to study'. There are also roots with four letters, such as `drhm` (from Greek *Drachme*), which names a numismatic unit. From a root one forms so–called *binyanim*, roughly translated as 'word classes', by inserting vowels or changing the consonantism of the root. In Table 15 we give some examples of verbs derived from the root `ktb`. Of these forms we can in turn form verbal forms in different tenses and voices.

We have only shown the transparent cases, there are other classes whose forms are not so regular. Write an interpreted LMG that generates these forms. For the meanings, simply assume unary operators, for example caus' for II, pass' for passive, and so on.

Exercise 187. In Mandarin (a Chinese language) a yes–no–question is formed as follows. A simple assertive sentence has the form (5.72) and the corresponding negative sentence the form (5.73). Mandarin is an SVO–language, and so the verb phrase follows the subject. The verb phrase is negated by

prefixing bu. (We do not write tones.)

(5.72) `Ta zai jia.`
He/She/It (is) at home

(5.73) `Ta bu zai jia.`
He/She/It (is) not at home

The yes–no–question is formed by concatenating the subject phrase with the positive verb phrase and then the negated verb phrase.

(5.74) `Ta zai jia bu zai jia?`
Is he/she/it at home?

As Radzinski [**?**] argues, the verb phrases have to be completely identical (with the exception of bu). For example, (5.75) is grammatical, (5.76) is ungrammatical. However, (5.77) is again grammatical and means roughly what (5.75) means.

(5.75) `Ni xihuan ta-de chenshan bu xihuan ta-de`
You like his shirt not like his
`chenshan?`
shirt?
Do you like his shirt?

(5.76) *`Ni xihuan ta-de bu xihuan ta-de chenshan?`
You like his not like his shirt?

(5.77) `Ni xihuan bu xihuan ta-de chenshan?`
You like not like his shirt?

Write an interpreted LMG generating these examples. Use $? : \varphi$ to denote the question, whether or not φ is the case.

4. Discontinuity

In this section we shall study a very important type of grammars, the so–called **Linear Context–Free Rewrite Systems — LCFRS** for short (see [**?**]). These grammars are weakly equivalent to what we call linear LMGs.

Definition 5.38 *A k–LMG is called* ***linear*** *if it is a simple k–LMG and every rule which is not 0–ary is downward nondeleting and downward linear, while 0–ary rules have the form* $X(\vec{x}_0,\dots,\vec{x}_{\Xi(X)-1}) \leftarrow .$ *with* $\vec{x}_i \in A$ *for all* $i < \Xi(X)$.

In other words, if we have a rule of this form

$$A(t_0,\dots,t_{k-1}) \leftarrow B_0(s_0^0,\dots,s_{k-1}^0),\dots,B_{n-1}(s_0^{n-1},\dots,s_{k-1}^{n-1}). \tag{5.78}$$

then for every $i < k$ and every $j < n$: $s_i^j = x_i^j$ and $x_j^i = x_{j'}^{i'}$ implies $i = i'$ and $j = j'$. Finally, $\prod_{i<k} t_i$ is a term containing each of these variables exactly once, in addition to occurrences of constants.

It is easy to see that the generative capacity is not diminished if we disallowed constants. In case $k = 1$ we get exactly the CFGs, though in somewhat disguised form: for now — if there are no constants — a rule is of the form

$$A(\prod_{i<n} x_{\pi(i)}) \leftarrow B_0(x_0),B_1(x_1),\dots,B(x_{n-1}). \tag{5.79}$$

where π is a permutation of the numbers $< n$. If $\rho := \pi^{-1}$ is the permutation inverse to π we can write the rule in this way.

$$A(\prod_{i<n} x_i) \leftarrow B_{\rho(0)}(x_0),B_{\rho(1)}(x_1),\dots,B_{\rho(n-1)}(x_{n-1}). \tag{5.80}$$

(To this end we replace the variable x_i by the variable $x_{\rho(i)}$ for every i. After that we permute the B_i. The order of the conjuncts is anyway insignificant in an LMG.) This is as one can easily see exactly the form of a context free rule. For we have

$$\prod_{i<n} x_i = x_0 x_1 \cdots x_{n-1} \tag{5.81}$$

This rule says therefore that if we have constituents $\vec{u}_i$ of type $B_{\rho(i)}$ for $i < n$, then $\prod_{i<n} \vec{u}_i$ is a constituent of type A.

The next case is $k = 2$. This defines a class of grammars which have been introduced before, using a somewhat different notation and which have been shown to be powerful enough to generate non CFLs such as Swiss German. In linear 2–LMGs we may have rules of this kind.

$$\begin{aligned}
&A(x_1x_2,y_1y_2) \leftarrow B(x_1,y_1),C(x_2,y_2).\\
&A(x_2x_1,y_1y_2) \leftarrow B(x_1,y_1),C(x_2,y_2).\\
&A(y_1x_1y_2,x_2) \leftarrow B(x_1,y_1),C(x_2,y_2).\\
&A(x_1,y_1x_2y_2) \leftarrow B(x_1,y_1),C(x_2,y_2).
\end{aligned} \tag{5.82}$$

The following rules, however, are excluded.

$$A(x_1, y_1y_2) \leftarrow B(x_1, y_1), C(x_2, y_2). \tag{5.83}$$

$$A(x_2x_2x_1, y_1y_2) \leftarrow B(x_1, y_1), C(x_2, y_2). \tag{5.84}$$

The first is upward deleting, the second not linear. We shall see that the language $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n\mathtt{d}^n : n \in \omega\}$ can be generated by a linear 2–LMG, the language $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n\mathtt{d}^n\mathtt{e}^n : n \in \omega\}$ however cannot. The second fact follows from Theorem 5.45. For the first language we give the following grammar.

$$\begin{aligned}
\mathtt{S}(y_0x_0y_1, z_0x_1z_1) &\leftarrow \mathtt{S}(x_0, x_1), \mathtt{A}(y_0, y_1), \mathtt{B}(z_0, z_1).\\
\mathtt{S}(\varepsilon, \varepsilon) &\leftarrow .\\
\mathtt{A}(\mathtt{a}, \mathtt{b}) &\leftarrow .\\
\mathtt{B}(\mathtt{c}, \mathtt{d}) &\leftarrow .
\end{aligned} \tag{5.85}$$

This shows that 2–linear LMGs are strictly stronger than CFGs. As a further example we shall look again at Swiss German (see Section 2.7). We define the following grammar. (Recall that $x \diamond y := x^\frown \square^\frown y$.)

$$\begin{array}{lll}
\mathtt{NPa(d'chind)} & \leftarrow . & \mathtt{NPd(em\ Hans)} \leftarrow .\\
\mathtt{NPsm(Jan)} & \leftarrow . & \mathtt{NPa(es\ huus)} \leftarrow .\\
\mathtt{Vdr(hälfe)} & \leftarrow . & \mathtt{Vf(lönd)} \leftarrow .\\
\mathtt{Var(laa)} & \leftarrow . & \\
\mathtt{Van(aastriche)} & \leftarrow . & \mathtt{C(das)} \leftarrow .\\
\mathtt{NPss(mer)} & \leftarrow . & \mathtt{Vc(säit)} \leftarrow .\\
\mathtt{S}(x \diamond y^\frown \mathtt{.}) & \leftarrow \mathtt{NPsm}(x), \mathtt{VP}(y). & \\
\mathtt{VP}(x^\frown \mathtt{,} \diamond y) & \leftarrow \mathtt{Vc}(x), \mathtt{CP}(y). & \\
\mathtt{CP}(x \diamond y \diamond z_0 \diamond z_1 \diamond u) & \leftarrow \mathtt{C}(x), \mathtt{NPss}(y), \mathtt{VI}(z_0, z_1), \mathtt{Vf}(u). & \\
\mathtt{VI}(x \diamond z_0, y \diamond z_1) & \leftarrow \mathtt{NPa}(x), \mathtt{Var}(y), \mathtt{VI}(z_0, z_1). & \\
\mathtt{VI}(x \diamond z_0, y \diamond z_1) & \leftarrow \mathtt{NPd}(x), \mathtt{Vdr}(y), \mathtt{VI}(z_0, z_1). & \\
\mathtt{VI}(x, y) & \leftarrow \mathtt{NPa}(x), \mathtt{Van}(y). &
\end{array} \tag{5.86}$$

This grammar is pretty realistic also with respect to the constituent structure, about which more below. For simplicity we have varied the arities of the predicates. Notice in particular the last two rules. They are the real motor of

the Swiss German infinitive constructions. For we can derive the following.

(5.87) $\texttt{VI}(\texttt{d'chind}\ z_0, \texttt{laa}\ z_1) \leftarrow \texttt{VI}(z_0, z_1).$

(5.88) $\texttt{VI}(\texttt{em Hans}\ z_0, \texttt{hälfe}\ z_1) \leftarrow \texttt{VI}(z_0, z_1).$

(5.89) $\texttt{VI}(\texttt{d'chind}, \texttt{aastriche}) \leftarrow .$

(5.90) $\texttt{VI}(\texttt{es huus}, \texttt{aastriche}) \leftarrow .$

(5.91) $\texttt{VI}(\texttt{d'chind em Hans}\ z_0, \texttt{laa hälfe}\ z_1) \leftarrow \texttt{VI}(z_0, z_1).$

(5.92) $\texttt{VI}(\texttt{em Hans es huus}\ z_0, \texttt{hälfe laa}\ z_1) \leftarrow \texttt{VI}(z_0, z_1).$

However, we do not have

(5.93) $\texttt{VI}(\texttt{em Hans}, \texttt{laa}) \leftarrow .$

The sentences of Swiss German as reported in Section 2.7 are derivable and some further sentences, which are all grammatical.

Linear LMGs can also be characterized by the vector polynomials which occur in the rules. We shall illustrate this by way of example with linear 2–LMGs and here only for the at most binary rules. We shall begin with the unary rules. They can make use of these vector polynomials.

(5.94)
$$\begin{aligned}
\mathfrak{i}(x_0, x_1) &:= \langle x_0, x_1 \rangle \\
\mathfrak{p}_X(x_0, x_1) &:= \langle x_1, x_0 \rangle \\
\mathfrak{p}_F(x_0, x_1) &:= \langle x_0 x_1, \varepsilon \rangle \\
\mathfrak{p}_G(x_0, x_1) &:= \langle x_1 x_0, \varepsilon \rangle \\
\mathfrak{p}_H(x_0, x_1) &:= \langle \varepsilon, x_0 x_1 \rangle \\
\mathfrak{p}_K(x_0, x_1) &:= \langle \varepsilon, x_1 x_0 \rangle
\end{aligned}$$

Then the following holds.

(5.95)
$$\begin{aligned}
\mathfrak{p}_X(\mathfrak{p}_X(x_0, x_1)) &= \mathfrak{i}(x_0, x_1) \\
\mathfrak{p}_G(x_0, x_1) &= \mathfrak{p}_F(\mathfrak{p}_X(x_0, x_1)) \\
\mathfrak{p}_K(x_0, x_1) &= \mathfrak{p}_H(\mathfrak{p}_X(x_0, x_1))
\end{aligned}$$

This means that one has $\mathfrak{p}_G$ at one's disposal if one also has $\mathfrak{p}_X$ and $\mathfrak{p}_F$, and that one has $\mathfrak{p}_F$ if one also has $\mathfrak{p}_X$ and $\mathfrak{p}_G$ and so on. With binary rules already the situation gets quite complicated. Therefore we shall assume that we have all unary polynomials. A binary vector polynomial is of the form

$\langle p_0(x_0,x_1,y_0,y_1), p_1(x_0,x_1,y_0,y_1)\rangle$ such that $q := p_0{}^\frown p_1$ is linear. Given q there exist exactly 5 choices for p_0 and p_1, determined exactly by the cut–off point. So we only need to list q. Here we can assume that in $q(x_0,x_1,y_0,y_1)$ x_0 always appears to the left of x_1 and y_0 to the left of y_1. Further, one may also assume that x_0 is to the left of y_0 (otherwise exchange the x_i with the y_i). After simplification this gives the following polynomials.

$$\begin{aligned} q_C(x_0,x_1,y_0,y_1) &:= x_0x_1y_0y_1 \\ q_W(x_0,x_1,y_0,y_1) &:= x_0y_0x_1y_1 \\ q_Z(x_0,x_1,y_0,y_1) &:= x_0y_0y_1x_1 \end{aligned} \tag{5.96}$$

Let us take a look at q_W. From this polynomial we get the following vector polynomials.

$$\begin{aligned} \mathtt{q}_{W0}(\langle x_0,x_1\rangle,\langle y_0,y_1\rangle) &:= \langle \varepsilon, x_0y_0x_1y_1\rangle \\ \mathtt{q}_{W1}(\langle x_0,x_1\rangle,\langle y_0,y_1\rangle) &:= \langle x_0, y_0x_1y_1\rangle \\ \mathtt{q}_{W2}(\langle x_0,x_1\rangle,\langle y_0,y_1\rangle) &:= \langle x_0y_0, x_1y_1\rangle \\ \mathtt{q}_{W3}(\langle x_0,x_1\rangle,\langle y_0,y_1\rangle) &:= \langle x_0y_0x_1, y_1\rangle \\ \mathtt{q}_{W4}(\langle x_0,x_1\rangle,\langle y_0,y_1\rangle) &:= \langle x_0y_0x_1y_1, \varepsilon\rangle \end{aligned} \tag{5.97}$$

We say that a linear LMG has polynomial basis Q if in the rules of this grammar only vector polynomials from Q have been used. It is easy to see that if $\mathtt{q}$ is a polynomial that can be presented by means of polynomials from Q, then one may add $\mathtt{q}$ to Q without changing the generative capacity. Notice also that it does not matter if the polynomial contains constants. If we have, for example,

$$\mathtt{X}(\mathtt{a}x\mathtt{bc}y) \leftarrow \mathtt{X}(x), \mathtt{Z}(y). \tag{5.98}$$

we can replace this by the following rules.

$$\begin{aligned} \mathtt{X}(uxvwy) &\leftarrow \mathtt{A}(u), \mathtt{X}(x), \mathtt{B}(v), \mathtt{C}(w), \mathtt{Z}(y). \\ \mathtt{A}(\mathtt{a}) &\leftarrow . \\ \mathtt{B}(\mathtt{b}) &\leftarrow . \\ \mathtt{C}(\mathtt{c}) &\leftarrow . \end{aligned} \tag{5.99}$$

This is advantageous in proofs. We bring to the attention of the reader some properties of languages that can be generated by linear LMGs. The following is established in [**?**], see also [**?**].

Proposition 5.39 (Vijay–Shanker & Weir & Joshi) *Let G be a linear k–LMG. Then $L(G)$ is semilinear.*

A special type of linear LMGs are the so–called head grammars. These grammars have been introduced by Carl Pollard in [**?**]. The strings that are manipulated are of the form $\vec{x}a\vec{y}$ where $\vec{x}$ and $\vec{y}$ are strings and $a \in A$. One speaks in this connection of a in the string as the distinguished **head**. This head is marked here by underlining it. Strings containing an underlined occurrence of a letter are called **marked**. The following rules for manipulating marked strings are now admissible.

$$\begin{aligned}
h_{C1}(\vec{v}\underline{a}\vec{w},\vec{y}\underline{b}\vec{z}) &:= \vec{v}\underline{a}\vec{w}\vec{y}b\vec{z}\\
h_{C2}(\vec{v}\underline{a}\vec{w},\vec{y}\underline{b}\vec{z}) &:= \vec{v}a\vec{w}\vec{y}\underline{b}\vec{z}\\
h_{L1}(\vec{v}\underline{a}\vec{w},\vec{y}\underline{b}\vec{z}) &:= \vec{v}\underline{a}\vec{y}b\vec{z}\vec{w}\\
h_{L2}(\vec{v}\underline{a}\vec{w},\vec{y}\underline{b}\vec{z}) &:= \vec{v}a\vec{y}\underline{b}\vec{z}\vec{w}\\
h_{R1}(\vec{v}\underline{a}\vec{w},\vec{y}\underline{b}\vec{z}) &:= \vec{v}\vec{y}b\vec{z}\underline{a}\vec{w}\\
h_{R2}(\vec{v}\underline{a}\vec{w},\vec{y}\underline{b}\vec{z}) &:= \vec{v}\vec{y}\underline{b}\vec{z}a\vec{w}
\end{aligned} \tag{5.100}$$

(Actually, this definition is due to [**?**], who showed that Pollard's definition is weakly equivalent to this one.) Notice that the head is not allowed to be empty. In [**?**] the functions are partial: for example, $h_{C1}(\varepsilon,\vec{w})$ is undefined. Subsequently, the definition has been changed slightly, basically to allow for empty heads. In place of marked strings one takes 2–vectors of strings. The marked head is the comma. This leads to the following definition. (This is due to [**?**]. See also [**?**].)

Definition 5.40 *A **head grammar** is a linear 2–LMG with the following polynomial basis.*

$$\begin{aligned}
\mathfrak{p}_{C1}(\langle x_0,x_1\rangle,\langle y_0,y_1\rangle) &:= \langle x_0,x_1y_0y_1\rangle\\
\mathfrak{p}_{C2}(\langle x_0,x_1\rangle,\langle y_0,y_1\rangle) &:= \langle x_0x_1y_0,y_1\rangle\\
\mathfrak{p}_{L1}(\langle x_0,x_1\rangle,\langle y_0,y_1\rangle) &:= \langle x_0,y_0y_1x_1\rangle\\
\mathfrak{p}_{L2}(\langle x_0,x_1\rangle,\langle y_0,y_1\rangle) &:= \langle x_0y_0,y_1x_1\rangle\\
\mathfrak{p}_{R1}(\langle x_0,x_1\rangle,\langle y_0,y_1\rangle) &:= \langle x_0y_0y_1,x_1\rangle
\end{aligned} \tag{5.101}$$

It is not difficult to show that the following basis of polynomials is sufficient: $\mathfrak{p}_{C1}$, $\mathfrak{p}_{C2}$ and

$$\mathfrak{p}_W(\langle x_0,x_1\rangle,\langle y_0,y_1\rangle) := \langle x_0y_0,y_1x_1\rangle \tag{5.102}$$

Notice that in this case there are *no* extra unary polynomials. However, some of them can be produced by feeding empty material. These are exactly the polynomials $\mathfrak{i}$, $\mathfrak{p}_F$ and $\mathfrak{p}_H$. The others cannot be produced, since the order of the component strings must always be respected. For example, one has

$$\mathfrak{p}_{C2}(\langle x_0, x_1\rangle, \langle \varepsilon, \varepsilon\rangle) = \langle x_0 x_1, \varepsilon\rangle = \mathfrak{p}_F(x_0, x_1) \tag{5.103}$$

We shall now turn to the description of the structures that correspond to the trees for CFGs. Recall the definitions of Section 1.4.

Definition 5.41 *A sequence $C = \langle \vec{w}_i : i < n+1\rangle$ of strings is called an n–**context**. A sequence $\mathfrak{v} = \langle \vec{v}_i : i < n\rangle$ **occurs in** $\vec{x}$ **in the context** C if*

$$\vec{x} = \vec{w}_0{}^\frown \prod_{i<n} \vec{v}_i \vec{w}_{i+1} \tag{5.104}$$

We write $C(\mathfrak{v})$ in place of $\vec{x}$.

Notice that an n–sequence of strings can alternatively be regarded as an $n-1$–context. Let G be a k–linear LMG. If $\vdash_G A(\vec{x}_0, \ldots, \vec{x}_{k-1})$ then this means that the k–tuple $\langle \vec{x}_i : i < k\rangle$ is a constituent of category A. If $\vdash_G S(\vec{x})$, then the derivation will consist in deriving statements of the form $A(\langle \vec{y}_i : i < \Xi(A)\rangle)$ such that there is an $n+1$–context for $\langle \vec{y}_i : i < \Xi(A)\rangle$ in $\vec{x}$.

The easiest kinds of structures are trees where each nonterminal node is assigned a tuple of subwords of the terminal string. Yet, we will not follow this approach as it generates stuctures that are too artificial. Ideally, we would like to have something analogous to constituent structures, where constituents are just appropriate subsets of the terminal nodes.

Definition 5.42 *An **labelled ordered tree of discontinuity degree** k is a labelled, ordered tree such that $[x]$ has at most k discontinuous pieces.*

If G is given, the labels are taken from A and R, and A is the set of labels of leaves, while R is the set of labels for nonleaves. Also, if Q is a k–ary predicate, it must somehow be assigned a unique k–tuple of subwords of the terminal string. To this end, we segment $[x]$ into k continuous parts. The way this is done exactly shall be apparent later. Now, if x is assigned B and if $[x]$ is the disjoint union of the continuous substrings $\vec{y}_i$, $i < k$, and if $\vec{y}_i$ precedes in $\vec{y}_j$ in $\vec{x}$ iff $i < j$ then $B(\langle \vec{y}_i : i < k\rangle)$.

However, notice that the predicates apply to sequences of substrings. This is to say that the linear order is projected from the terminal string. Additionally, the division into substrings can be read off from the tree (though not

from $[x]$ alone). There is, however, one snag. Suppose G contains a rule of the form

(5.105) $$A(x_1x_0y_0, y_1) \leftarrow B(x_0, x_1), C(y_0, y_1)$$

Then assuming that we can derive $B(\vec{u}_0, \vec{u}_1)$ and $C(\vec{v}_0, \vec{v}_1)$, we can also derive $A(\vec{u}_1\,\vec{u}_0\,\vec{v}_0, \vec{v}_1)$. However, this means that $\vec{u}_1$ must precede $\vec{u}_0$ in the terminal string, which we have excluded. We can prevent this by introducing a nonterminal B^* such that $B^*(x_0, x_1) \leftrightarrow B(x_1, x_0)$, and then rewrite the rule as

(5.106) $$A(x_0x_1y_0, y_1) \leftarrow B^*(x_0, x_1), C(y_0, y_1).$$

The problem with the rule (5.105) is that it switches the order of the x_i's. Rules that do this (or switch the order of the y_i's) are called nonmonotone in the sense of the following definition.

Definition 5.43 *Let $\rho = L \leftarrow M_0 \cdots M_{n-1}$ be a linear rule, $L = B(\langle t^j : j < k\rangle)$ and $M_i = A_i(\langle x_i^j : j < k_i\rangle)$, $i < n$. ρ is called **monotone** of for every $i < n$ and every pair $j < j' < k_i$ the following holds: if x_i^j occurs in t^q and $x_i^{j'}$ in $t^{q'}$ then either $q < q'$ or $q = q'$ and x_i^j occurs before $x_i^{j'}$ in the polynomial t_q. An LMG is monotone if all of its rules are.*

Now, for every LCFRS there exists a monotone LCFRS that generates the same strings (and modulo lexical rules also the same structures). For every predicate A and every permutation $\pi : k \to k$, $k := \Xi(A)$ assume a distinct predicate A^π with the intended interpretation that $A(x_0, \ldots, x_{k-1})$ iff $A^\pi(x_{\pi^{-1}(0)}, \ldots, x_{\pi^{-1}(k-1)})$. Every rule $A(\vec{s}) \leftarrow B_0 \ldots B_{p-1}$ is replaced by all possible rules $A^\pi(\pi^{-1}(\vec{s})) \leftarrow B_0 \ldots B_{p-1}$.

Let $A \leftarrow B_0 \ldots B_{p-1}$ be a rule. Now put $\rho_i(j) := k$ iff x_i^k is the jth variable from the variables x_i^q, $q < \Xi(B_i)$, which occurs in $\prod_{i<\Xi(A)} t_i$, counting from the left. Then $A' \leftarrow B_0^{\rho_0} \ldots B_{p-1}^{\rho_{p-1}}$ will replace the rule $A \leftarrow B_0 \ldots B_{p-1}$, where A' results from A by applying the substitution $x_i^j \mapsto x_i^{\rho_i^{-1}(j)}$ (while the variables of the B_i^ρ remain in the original order). This rule is monotone. For example, assume that we have the rule

(5.107) $$A(x_2y_1x_1, x_0y_2y_0y_3) \leftarrow B_0(x_0, x_1, x_2), C(y_0, y_1, y_2, y_3).$$

Then we put $\rho_0 : 0 \mapsto 2, 1 \mapsto 1, 2 \mapsto 0$, and $\rho_1 : 0 \mapsto 2, 1 \mapsto 0, 2 \mapsto 1, 3 \mapsto 3$. So we get

(5.108) $$A(x_0y_0x_1, x_2y_1y_2y_3) \leftarrow B^{\rho_0}(x_0, x_1, x_2), C^{\rho_1}(y_0, y_1, y_2, y_3).$$

Every terminal rule is monotone. Thus, we can essentially throw out all nonmonotone rules. Thus, for nonmonotone LCFRS there is a monotone LFCRS generating the same strings.

We can a derive useful theorem on LCFRSs.

Definition 5.44 *A language L is called k–**pumpable** if there is a constant p_L such that for all $\vec{x}$ of length $\geq p_L$ there is a decomposition $\vec{x} = \vec{u}_0 \prod_{i<k}(\vec{v}_i\vec{u}_{i+1})$ such that*

$$\{\vec{u}_0 \prod_{i<k}(\vec{v}_i{}^n\vec{u}_i) : n \in \omega\} \subseteq L \tag{5.109}$$

Theorem 5.45 (Groenink) *Suppose that L is a k–LCFRL. Then L is $2k$–pumpable.*

We provide a proof sketch based on derivations. Transform the language into a language of signs, by adding the trivial semantics. Then let $\mathfrak{A}$ be a sign grammar based on a monotone k–LCFRS for it. Observe that if $\mathfrak{t}$ is a structure term containing x free exactly once, and if $\mathfrak{t}(\mathfrak{u})$ and $\mathfrak{u}$ are definite and unfold to signs of identical category, then with $\mathfrak{s}(\mathfrak{t}(\mathfrak{u}))$ also $\mathfrak{s}(\mathfrak{t}^n(\mathfrak{u}))$ is definite for every n (the rule skeleton is context free). Now, if $\vec{x}$ is large enough, its structure term will be of the form $\mathfrak{s}(\mathfrak{t}(\mathfrak{u}))$ such that $\mathfrak{t}(\mathfrak{u})$ and $\mathfrak{u}$ have the same category. Finally, suppose that the grammar is monotone. Then $\mathfrak{t}^\varepsilon$ is a monotone, linear polynomial function on k–tuples; hence there are $\vec{v}_i$, $i < 2k$, such that

$$p(x_0, \ldots, x_{k-1}) = \langle \vec{v}_{2i}x_i\vec{v}_{2i+1} : i < k\rangle \tag{5.110}$$

Thus let us now focus on monotone LCFRSs. We shall define the structures that are correspond to derivations in monotone LCFRSs, and then show how they can be generated using graph grammars.

Definition 5.46 *Suppose that $\mathfrak{T} = \langle T, <, \sqsubset, \ell\rangle$ is an ordered labelled tree with degree of discontinuity k and $G = \langle A, R, \Xi, S, H\rangle$ a monotone LCFRS. We say that $\mathfrak{T}$ is a G–**tree** if*

① *$\ell(v) \in A$ iff v is a leaf,*

② *for every nonleaf v: there is a set $\{H(v)_i : i < k\}$ of leaves such that $k = \Xi(\ell(v))$, and $H(v)_i \subseteq [x]$ is continuous,*

③ *if v has daughters w_i, $i < n$, then there is a rule*

$$A(t_0,\ldots,t_{\Xi(A)}) \leftarrow B_0(x_0^0,\ldots,x_0^{\Xi(B_0)}),\ldots,B_{n-1}(x_{n-1}^0,\ldots,x_{n-1}^{\Xi(B_{n-1})})$$

such that

- $*$ *v has label A, w_i has label B_i ($i < n$),*
- $*$ *if $t_j = \prod_{i<v} x_{g(i)}^{h(i)}$ then $H(v)_j = \bigcup_{i<v} H(w_{g(i)})_{h(i)}$.*

The last clause is somewhat convoluted. It says that whatever composition we assume of the leaves associated with v, it must be compatible with the composition of the w_i, and the way the polynomials are defined. Since the preterminals are unary, it can be shown that $H(v)_i$ is unique for all v and i.

The following grammar is called $G^{\heartsuit}$.

$$\begin{array}{ll}
\mathtt{S}(y_0x_0y_1,z_0x_1z_1) \leftarrow \mathtt{S}(x_0,x_1),\mathtt{X}(y_0,y_1),\mathtt{Y}(z_0,z_1). & \\
\mathtt{S}(\varepsilon,\varepsilon) \leftarrow . & \\
\mathtt{X}(x,y) \leftarrow \mathtt{A}(x),\mathtt{B}(y). & \mathtt{Y}(x,y) \leftarrow \mathtt{C}(x),\mathtt{D}(y). \\
\mathtt{A}(\mathtt{a}) \leftarrow . & \mathtt{C}(\mathtt{c}) \leftarrow . \\
\mathtt{B}(\mathtt{b}) \leftarrow . & \mathtt{D}(\mathtt{d}) \leftarrow .
\end{array} \tag{5.111}$$

$G^{\heartsuit}$ derives aabbccdd with the structure shown in Figure 12. This tree is not exhaustively ordered. This is the main difference with CFGs. Notice that the tree does not reflect the position of the empty constituent. Its segments are found between the second and the third as well as between the sixth and the seventh letter. One can define the structure tree also in this way that it explicitly contains the empty strings. To this end one has to replace also occurrences of ε by variables. The rest is then analogous.

We shall now define a context free graph grammar that generates the same structures as a given monotone LCFRS. For the sake of simplicity we assume that all predicates are k–ary, and that all terminal rules are of the form $Y(a,\varepsilon,\ldots,\varepsilon)$, $a \in A$. Monotonicity is not necessary to assume, but makes life easier. The only problem that discontinuity poses is that constituents cannot be ordered with respect to each other in a simple way. The way they are related to each other shall be described by means of special matrices.

Assume that $\vec{u}$ is a string, $C = \langle \vec{v}_i : i < k+1 \rangle$ a $k+1$–context for $\vec{x}_i$, $i < k$, in $\vec{u}$ and $D = \langle \vec{w}_i : i < k+1 \rangle$ a $k+1$–context for $\vec{y}_j$, $j < k$, in $\vec{u}$. Now, write $M(C,D) := (\mu_{ij})_{ij}$ for the following matrix:

$$\mu_{pq} = 1 \text{ iff } \prod_{i<p} \vec{v}_i\vec{x}_i \text{ is a prefix of } \vec{w}_0{}^{\frown} \prod_{i<q} \vec{y}_i\vec{w}_{i+1} \tag{5.112}$$

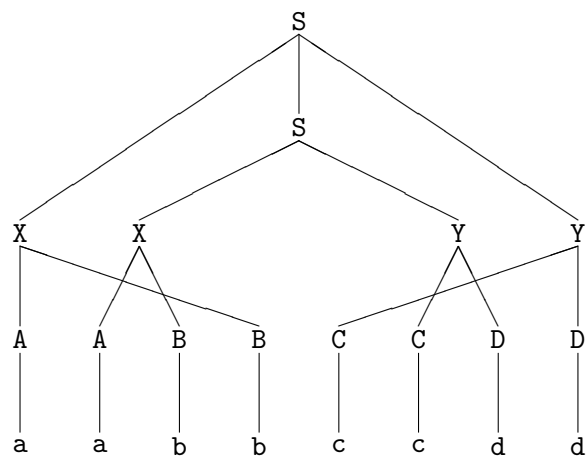


Figure 12. A Structure Tree for $G^{\heartsuit}$

We call $M(C,D)$ the **order scheme of** C and D. Intuitively, the order scheme tells us which of the $\vec{x}_p$ precede which of the $\vec{y}_q$ in $\vec{u}$. We say that C and D **overlap** if there are $i, j < k$ such that the (occurrences of) $\vec{x}_p$ and $\vec{y}_q$ overlap. This is the case iff $\mu_{pq} = \mu_{qp} = 0$.

Lemma 5.47 *Assume that $\mathfrak{B}$ is a labelled ordered tree for a monotone LCFRS with yield $\vec{u}$. Further, let x and y be nodes. Then $\ell(x)$ and $\ell(y)$ overlap iff x and y are comparable.*

Notice that in general $\mu_{qp} = 1 - \mu_{pq}$ in the nonoverlapping case, which is what we shall assume from now on. We illustrate this with an example. Figure 13 shows some 2–schemes for monotone rules together with the orders which define them. (We omit the vector arrows; the ordering is defined by 'is to the left of in the string'.) For every k–scheme M let ξ_M be a relation. Now we define our graph grammar. $N^\circ := \{A^\circ : A \in N\}$ is a set of fresh nonterminals. The set of vertex colours $F_V := N \cup N^\circ \cup A$, the set of terminal vertex colours is $F_V^T := N \cup A$, the set of edge colours is $\{<\} \cup \{\xi_M : M \text{ a } k\text{–scheme}\}$. (As we will see immediately, $\sqsubset$ is the relation which fits to the relation $\xi_{\mathbb{I}}$ where $\mathbb{I} = (1)_{ij}$ is the matrix which consists only of 1s.) The start graph is the one–element graph $\mathfrak{S}$ which has one edge. The vertex has colour S, the edge

$$\begin{array}{ccc} \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} & \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} & \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix} \\ x_0x_1y_0y_1 & x_0y_0x_1y_1 & x_0y_0y_1x_1 \\ \\ \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} & \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \\ y_0x_0x_1y_1 & y_0x_0y_1x_1 & y_0y_1x_0x_1 \end{array}$$

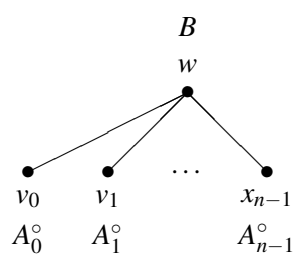
Figure 13. Order Schemes

has only one colour, ξ_K, where $K = (\kappa_{pq})_{pq}$ with $\kappa_{pq} = 1$ iff $p < q$. For every rule ρ we add a graph replacement rule. Let

$$\rho = B(\langle t^j : j < k\rangle) \leftarrow A_0(\langle x_0^j : j < k\rangle)\cdots A_{n-1}(\langle x_{n-1}^j : j < k\rangle) \tag{5.113}$$

be given. Let $p := \prod_{i<k} t^i$ be the characteristic polynomial of the rule. Then the graph replacement ρ^γ replaces the node B° by the following graph.

(5.114)



Furthermore, between the nodes v_i and w_j, $i \neq j$, the following relations hold (which are not shown in the picture). Put $\mu(i,j)_{i'j'} := 1$ if $x_i^{i'}$ is to the left of $x_j^{j'}$ in p. Otherwise, put $\mu(i,j)_{i'j'} := 0$. Then put $H_{ij} := (\mu(i,j)_{i'j'})_{i'j'}$. The relation from v_i to v_j is $\xi_{H_{ij}}$. Notice that by definition always either $x_i^{i'}$ is to the left of $x_j^{j'}$ or to the right of it. Hence the relation between v_j and v_i is exactly ξ_{1-H}. This is relevant insofar as it allows us to concentrate on one colour functional only: $\mathfrak{II}$. Now supppose that w is in relation ξ_M to a node u. (So, there is an edge of colour ξ_M from v to u.) We need to determine the relation (there is only one) from v_i to u. This is ξ_N, where $N = (\nu_{pq})_{pq}$ and $\nu_{pq} = 1$ iff $\mu_{p'q} = 1$, where x_i^p occurs in $t^{p'}$. The map that sends M to N and $<$ to $<$ is the desired colour functional $\mathfrak{II}$. The other functionals can be straightforwardly defined.

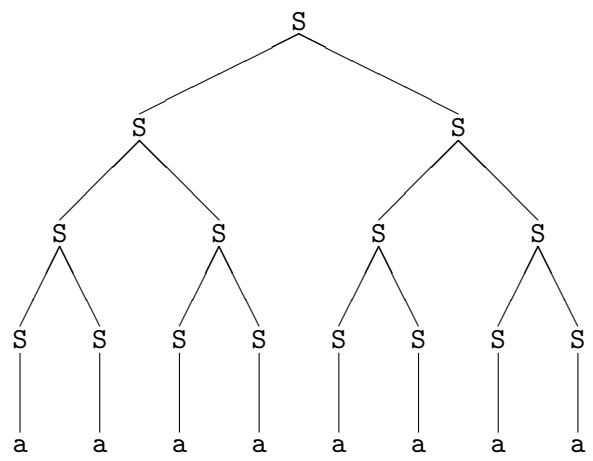


Figure 14. An Exponentially Growing Tree

There is a possibility of defining structure in some restricted cases, namely always when the right hand sides do not contain a variable twice. This differs from linear grammars in that variables are still allowed to occur several times on the left, but only once on the right. An example is the grammar

(5.115) $\mathtt{S}(xx) \leftarrow \mathtt{S}(x).; \quad \mathtt{S}(\mathtt{a}) \leftarrow .$

The notion of structure that has been defined above can be transferred to this grammar. We simply do as if the first rule was of this form

(5.116) $\mathtt{S}(xy) \leftarrow \mathtt{S}(x), \mathtt{S}(y).$

where it is clear that x and y always represent the same string. In this way we get the structure tree for `aaaaaaaa` shown in Figure 14.

Notes on this section. In [**?**], a slightly more general type of grammars than the LCFRSs is considered, which are called *Multiple Context Free Grammars* (MCFGs). In our terminology, MCFGs maybe additionally upward deleting. In [**?**] weak equivalence between MCFGs and LCFRs is shown. See also [**?**]. [**?**] also defines monotone rules for MCFGs and shows that any MCFL can be generated by a monotone MCFG. For the relevance of these grammars in

parsing see [**?**; **?**]. In his paper [**?**], Edward Stabler describes a formalisation of minimalist grammars akin to Noam Chomsky's Minimalist Program (outlined in [**?**]). Subsequently, in [**?**; **?**; **?**] and [**?**] it is shown that the languages generated by this formalism are exactly those that can be generated by simple LMGs, or, for that matter, by LCFRSs.

Exercise 188. Show that the derivation Γ' is determined by Γ up to renaming of variables.

Exercise 189. Prove Proposition 5.39.

Exercise 190. Let $A_k := \{\mathtt{a}_i : i < k\}$, and let $W_k := \{\prod_{i<k} \mathtt{a}_i^n : n \in \omega\}$. Show that W_k is a m–LCFRL iff $k \leq 2m$.

Exercise 191. Determine the graph grammar $\gamma G^{\heartsuit}$.

Exercise 192. Show the following. *Let $N = \{x_i : i < k\} \cup \{y_i : i < k\}$ and $<$ a linear ordering on N with $x_i < x_j$ as well as $y_i < y_j$ for all $i < j < k$. Then if $m_{ij} = 1$ iff $x_i < y_j$ then $M = (m_{ij})_{ij}$ is a k–scheme. Conversely: let M be a k–scheme and $<$ defined by (1) $x_i < x_j$ iff $i < j$, (2) $y_i < y_j$ iff $i < j$, (3) $x_i < y_j$ iff $m_{ij} = 1$. Then $<$ is a linear ordering. The correspondence between orderings and schemes is biunique.*

Exercise 193. Show the following: *If in a linear k–LMG all structure trees are exhaustively ordered, the generated tree set is context free.*

5. Adjunction Grammars

In this and the next section we shall concern ourselves with some alternative types of grammars which are all (more or less) equivalent to head grammars. These are the tree adjoining grammars (TAGs), CCGs (which are some refined version of the adjunction grammars of Section 1.4 and the grammars CCG($\mathbf{Q}$) of Section 3.4, respectively) and the so–called linear index grammars.

Let us return to the concept of tree adjoining grammars. These are pairs $G = \langle \mathbb{C}, N, A, \mathbb{A} \rangle$, where $\mathbb{C}$ is the set of centre trees and $\mathbb{A}$ a set of adjunction trees. In an adjunction tree a node is called **central** if it is above the distinguished leaf or identical to it.

It is advantageous to define a naming scheme for nodes in an adjunction tree. Let $\mathfrak{T} = \langle T, <_T, \sqsubset_T, \ell_T \rangle$ be a centre tree. Then put $N(T) := T$. If adjoin-

ing $\mathfrak{A} = \langle A, <_A, \sqsubset_A, \ell_A \rangle$ at x to $\mathfrak{T}$ yields $\mathfrak{U} = \langle U, <_U, \sqsubset_U, \ell_U \rangle$ then

$$N(\mathfrak{U}) := (N(T) - \{x\}) \cup \{x^\frown \mathfrak{A}^\frown v : v \in A\} \tag{5.117}$$

Let H be the set of all nodes of centre or adjunction trees. Then $N(\mathfrak{T}) \subseteq H \cdot (\mathfrak{A} \cdot H)^*$. Furthermore, as is inductively verified, $N(\mathfrak{T})$ is prefix free. Thus, as x gets replaced by strings of which x is a suffix, no name that is added equals any name in $N(\mathfrak{U})$. So, the naming scheme is unique. Moreover, it turns out that the structure of $\mathfrak{T}$ is uniquely determined by $N(\mathfrak{T})$. The map that identifies $x \in T$ with its name is called ν. Put $N := N(\mathfrak{T})$ and

$$\mathfrak{N}(\mathfrak{T}) := \langle N, <^N, \sqsubset^N, \ell^N \rangle \tag{5.118}$$

If $\vec{\sigma} = \vec{\gamma}^\frown \mathfrak{A}^\frown j$, let $\ell^N(\vec{\sigma}) := X$ for the unique X which is the label of the node j in $\mathfrak{A}$. Second, put $\vec{\sigma} \sqsubset_N \vec{\tau}$ if $\vec{\sigma} = \vec{\gamma}^\frown j^\frown \mathfrak{A}^\frown \vec{\eta}$ and $\vec{\tau} = \vec{\gamma}^\frown j'^\frown \mathfrak{A}^\frown \vec{\theta}$ for certain $\vec{\gamma}$, $\vec{\eta}$, $\vec{\theta}$, $\mathfrak{A}$ and $j \neq j'$ such that $j \sqsubset j'$. Third, $\vec{\sigma} < \vec{\tau}$ if $\vec{\sigma} = \vec{\gamma}^\frown j^\frown \mathfrak{A}^\frown \vec{\eta}$ and $\vec{\tau} = \vec{\gamma}^\frown j'^\frown \mathfrak{A}^\frown \vec{\theta}$ for certain $\vec{\gamma}$, $\vec{\eta}$, $\vec{\theta}$, $\mathfrak{A}$ and $j \neq j'$, such that (a) $j < j'$, (b) j' and every node of $\vec{\theta}$ is central in its corresponding adjunction tree.

Proposition 5.48 *ν is an isomorphism from $\mathfrak{T}$ onto $\mathfrak{N}(\mathfrak{T})$.*

Thus, we basically only need to define $N(\mathfrak{T})$. It turns out that the sets $N(\mathfrak{T})$ are the sets of leaves of a CFG. For the sake of simplicity we assume that the set of nodes of $\mathfrak{B}$ is the set of numbers $j(\mathfrak{B}) = \{0, 1, \ldots, j(\mathfrak{B}) - 1\}$. $j_* := \max\{j(\mathfrak{B}) : \mathfrak{B} \in \mathbb{A} \cup \mathbb{C}\}$. The terminals are the numbers $< j_*$. The nonterminals are pairs $(i, \mathfrak{B})$ where $\mathfrak{B}$ is a tree and $i < j_*$. The start symbols are $(0, \mathfrak{C})$, $\mathfrak{C} \in \mathbb{C}$. First, we shall define the grammar* $D(G)$. The rules are of this form.

$$(j, \mathfrak{A}) \to X_0 \quad X_1 \cdots X_{j(\mathfrak{A})-1} \tag{5.119}$$

where (5.119) is to be seen as rule a scheme: for every $\mathfrak{A}$ and every admissible j we may choose whether X_i is i or $(i, \mathfrak{B}_i)$ $(i < j(\mathfrak{A}))$ for some tree $\mathfrak{B}_i$ which can be adjoined at i in $\mathfrak{A}$. This grammar* we denote by $D(G)$ and call it the **derivation grammar**. A **derivation** for G is simply a tree generated by $D(G)$. The following is clear: trees from $D(G)$ are in one–to–one correspondence with their sets of leaves, which in turn define tree of the adjunction grammar. It should be said that the correspondence is not always biunique. (This is so since any given tree may have different derivations, and these get matched with nonisomorphic trees in $D(G)$. However, each derivation tree maps to exactly one tree of G modulo isomorphism.)

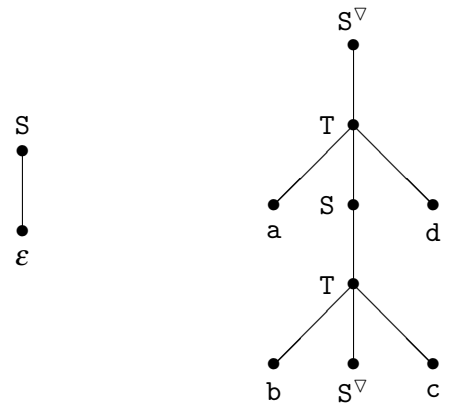


Figure 15. A Tree Adjoining Grammar for $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n\mathtt{d}^n : n \in \omega\}$

TAGs differ from the unregulated tree adjunction grammars in that they allow to specify

① whether adjunction at a certain node is licit,

② which trees may be adjoined at which node, and

③ whether adjunction is obligatory at certain nodes.

We shall show that ① increases the generative power but ② and ③ do not in presence of ①. To establish control over derivations, we shall have to change our definitions a little bit. We begin with ①. To control for the possiblity of adjunction, we assume that the category symbols are now of the form a, $a \in A$, or X and $X^{\triangledown}$ respectively, where $X \in N$. Centre and adjunction trees are defined as before. Adjunction is also defined as before. There is a leaf i which has the same label as the root (all other leaves carry terminal labels). However, no adjunction is licit at nodes with label $X^{\triangledown}$. Notice that root and distinguished leaf must carry the same nonterminal, it is not admitted that one carries X while the other has $X^{\triangledown}$. Even if we admitted that, this would not increase the generative capacity. Using such grammars one can generate the language $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n\mathtt{d}^n : n \in \omega\}$. Figure 15 shows such a grammar. The centre tree is shown to the left, the adjunction tree to the right. It is not hard to show that one can generally reduce such grammars to those where both the root

and the leaf carry labels of the form $X^{\triangledown}$. Namely, if the root does not carry an adjunction prohibition, but, say, a label $X \in N$, then add a new root which has label $X^{\triangledown}$, and similarly for the distinguished leaf. Also, notice that adjunction prohibition for interior nodes of adjunction trees can be implemented by changing their label to a newly added nonterminal. Trivially, no tree can be adjoined there.

Definition 5.49 *A **standard tree adjoining grammar** (or simply a **TAG**) is an adjunction grammar in which the adjunction trees carry an adjunction prohibition at the root and the distinguished leaf.*

Now let us turn to ② and ③. It is possible to specify in standard TAGs whether adjunction is obligatory and which trees may be adjoined. So, we also have a function f which maps all nodes with nonterminal labels to sets of adjunction trees. (If for some i $f(i) = \varnothing$ then that node effectively has an adjunction prohibition.) We can simulate this as follows. Let $\mathbb{A}$ be the set of adjunction trees. We think of the nonterminals as labels of the form $\langle X, \mathfrak{T} \rangle$ and $\langle X, \mathfrak{T} \rangle^{\triangledown}$, respectively, where $X \in N$ and $\mathfrak{T} \in \mathbb{A}$. A (centre or adjunction) tree $\mathfrak{T}$ is replaced by all trees $\mathfrak{T}'$ on the same set of nodes, where i carries the label $\langle X, \mathfrak{U} \rangle$ if i had label X in $\mathfrak{T}$ if $\mathfrak{U} \in f(i)$, and $\langle X, \mathfrak{U} \rangle^{\triangledown}$ if i has the label $X^{\triangledown}$ in $\mathfrak{T}$. However, if i is the root, it will only get the label $\langle i, \mathfrak{T} \rangle^{\triangledown}$. The second element says nothing but which tree is going to be adjoined next. This eliminates the second point from the list, as we can reduce the grammars by keeping the tree structure.

Now let us turn to the last point, the obligation for adjunction. We can implement this by introducing labels of the form $X^{\bullet}$. (Since obligation and prohibition to adjoin are exclusive, $\bullet$ occurs only when $\triangledown$ does not.) A tree is complete only if there are no nodes with label $X^{\bullet}$ for any X. Now we shall show that for every adjunction grammar of this kind there exists a grammar generating the same set of trees where there is no obligation for adjunction. We adjoin to a centre tree as often as necessary to eliminate the obligation. The same we do for adjunction trees. The resulting trees shall be our new centre and adjunction trees. Obviously, such trees exist (otherwise we may choose the set of centre trees to be empty). Now we have to show that there exists a finite set of minimal trees. Look at a tree without adjunction obligation and take a node. This node has a history. It has been obtained by successive adjunction. If this sequence contains an adjunction tree twice, we may cut the cycle. (The details of this operation are left to the reader.) This grammar still generates the same trees. So, we may remain with the standard form

of TAGs.

Now we shall first prove that adjunction grammars cannot generate more languages as linear 2–LMGs. From this it immediately follows that they can be parsed in polynomial time. The following is from [**?**], who incidentally show the converse of that theorem as well: head grammars are weakly equivalent to TAGs.

Theorem 5.50 (Vijay–Shanker & Weir & Joshi) *For every TAG G there exists a head grammar K such that $L(K) = L(G)$.*

Proof. Let G be given. We assume that the trees have pairwise disjoint sets of nodes. We may also assume that the trees are at most binary branching. (We only need to show weak equivalence.) Furthermore, we can assume that the nodes are strictly branching if not preterminal. The set of all nodes is denoted by M. The alphabet of nonterminals is $N' := \{i^a : i \in M\} \cup \{i^n : i \in M\}$. The start symbol is the set of all i^a and i^n where i is the root of a centre tree. By massaging the grammar somewhat one can achieve that the grammar contains only one start symbol. Now we shall define the rules. For a local tree we put

(5.120) $\quad i(a, \varepsilon) \leftarrow .$

if i is a leaf with terminal symbol a. If i is a distinguished leaf of an adjunction tree we also take the rule

(5.121) $\quad i^n(\varepsilon, \varepsilon) \leftarrow .$

Now let $i \to j \quad k$ be a branching local tree. Then we add the following rules.

(5.122) $\quad i^a(x_0x_1, y_0y_1) \leftarrow j^n(x_0, x_1), k^n(y_0, y_1).$

Further, if i is a node to which a tree with root j can be adjoined, then also this is a rule.

(5.123) $\quad i^n(x_0y_0, y_1x_1) \leftarrow j^n(x_0, x_1) \quad i^a(y_0, y_1).$

If adjunction is not necessary or prohibited at i, then finally the following rule is added.

(5.124) $\quad i^n(x_0, x_1) \leftarrow i^a(x_0, x_1).$

This ends the definition of K. In view of the rules (5.122) it is not entirely clear that we are dealing with a head grammar. So, replace the rules (5.122) by the following rules:

$$i^a(x_0, x_1y_0y_1) \leftarrow j^{n\bullet}(x_0, x_1), k^{n\bullet}(y_0, y_1). \tag{5.125}$$

$$j^{n\bullet}(x_0x_1y_0, y_1) \leftarrow j^n(x_0, x_1), \mathtt{L}(y_0, y_1). \tag{5.126}$$

$$k^{n\bullet}(x_0, x_1y_0y_1) \leftarrow \mathtt{L}(x_0, x_1), k^n(y_0, y_1). \tag{5.127}$$

$$\mathtt{L}(\varepsilon, \varepsilon) \leftarrow . \tag{5.128}$$

These are rules of a head grammar; (5.122) can be derived from them. For this reason we remain with the rules (5.122).

It remains to show that $L(K) = L(G)$. First the inclusion $L(G) \subseteq L(K)$. We show the following. Let $\mathfrak{T}$ be a local tree which contains exactly one distinguished leaf and nonterminal leaves x_i, $i < n$, with labels k_i. Let therefore $j < i$ be distinguished. We associate with $\mathfrak{T}$ a vector polynomial $\mathfrak{p}(\mathfrak{T})$ which returns

$$\langle \prod_{i<j} \vec{y}_i\vec{z}_i, \prod_{j<i<n} \vec{y}_i\vec{z}_i \rangle \tag{5.129}$$

for given pairs of strings $\langle \vec{y}_i, \vec{z}_i \rangle$. It is possible to show by induction over $\mathfrak{T}$ that there is a K–derivation

$$i^n(\mathfrak{p}(\mathfrak{T})(\langle \langle \vec{y}_i, \vec{z}_i \rangle : i < n \rangle)) \leftarrow^* k_0^n(\langle \vec{y}_0, \vec{z}_0 \rangle), \ldots, k_{n-1}^n(\langle \vec{y}_{n-1}, \vec{z}_{n-1} \rangle). \tag{5.130}$$

If no leaf is distinguished in $\mathfrak{T}$ the value of $p(\mathfrak{T})$ is exactly

$$\langle \vec{y}_0\vec{z}_0, \prod_{0<i<n} \vec{y}_i\vec{z}_i \rangle \tag{5.131}$$

This claim can be proved inductively over the derivation of $\mathfrak{T}$ in G. From this it follows immediately that $\vec{x} \in L(K)$ if $\vec{x} \in L(G)$. For the converse inclusion one has to choose a different proof. Let $\vec{x} \in L(K)$. We choose a K–derivation of $\vec{x}$. Assume that no rule of type (5.123) has been used. Then $\vec{x}$ is the string of a centre tree as is easily seen. Now we assume that the claim has been shown for derivations with fewer than n applications of (5.123) and that the proof has exactly n applications. We look at the last application. This is followed only by applications of (5.122), (5.120) and (5.121). These commute if they

belong to different subtrees. We can therefore rearrange the order such that our application of (5.123) is followed exactly by those applications of (5.122), (5.120) and (5.121) which belong to that subtree. They derive

(5.132) $\quad i^a(\vec{x}_0, \vec{x}_1).$

where i is the left hand side of the application of (5.123), and $\langle \vec{x}_0, \vec{x}_1 \rangle$ is the pair of the adjunction tree whose root is i. ($\vec{x}_0$ is to the left of the distinguished leaf, $\vec{x}_1$ to the right.) Before that we have the application of our rule (5.123):

(5.133) $\quad j^a(\vec{x}_0\vec{y}_0, \vec{y}_1\vec{x}_1) \leftarrow i^a(\vec{x}_0, \vec{x}_1), j^n(\vec{y}_0, \vec{y}_1).$

Now we eliminate this part of the derivation. This means that in place of $j^a(\vec{x}_0\vec{y}_0, \vec{y}_1\vec{x}_1)$ we only have $j^n(\vec{y}_0, \vec{y}_1)$. This however is derivable (we already have the derivation). But on the side of the adjunction this corresponds exactly to the disembedding of the corresponding adjunction tree. □

The converse also holds. However, the head grammars do not exhaust the 2–LMGs. For example look at the following grammar G.

$$
\begin{aligned}
\mathtt{S}(y_0x_0y_1, x_1) &\leftarrow \mathtt{T}(x_0, x_1), \mathtt{H}(y_0, y_1).\\
\mathtt{T}(x_0, \mathtt{c}x_1\mathtt{d}) &\leftarrow \mathtt{U}(x_0, y_1).\\
\mathtt{U}(\mathtt{a}x_0\mathtt{b}, x_1) &\leftarrow \mathtt{S}(x_0, x_1).\\
\mathtt{S}(\mathtt{ab}, \mathtt{cd}) &\leftarrow .\\
\mathtt{H}(\mathtt{t}x_0\mathtt{u}, x_1) &\leftarrow \mathtt{K}(x_0, x_1).\\
\mathtt{K}(x_0, \mathtt{v}x_1\mathtt{w}) &\leftarrow \mathtt{H}(x_0, x_1).\\
\mathtt{H}(\varepsilon, \varepsilon) &\leftarrow .
\end{aligned}
\tag{5.134}
$$

To analyze the generated language we remark the following facts.

Lemma 5.51 $\mathtt{H}(\vec{x}, \vec{y})$ *iff* $\langle \vec{x}, \vec{y} \rangle = \langle \mathtt{t}^n\mathtt{u}^n, \mathtt{v}^n\mathtt{w}^n \rangle$ *for some* $n \in \omega$.

As a proof one may reflect that first of all $\vdash_G \mathtt{H}(\varepsilon, \varepsilon)$ and secondly

(5.135) $\quad \vdash_G \mathtt{H}(\mathtt{t}x_0\mathtt{u}, \mathtt{v}x_1\mathtt{w})$ iff $\vdash_G \mathtt{H}(x_0, x_1)$

From this the following characterization can be derived.

Lemma 5.52 *Let* $\vec{x}_n := \mathtt{t}^n\mathtt{u}^n$ *and* $\vec{y}_n := \mathtt{v}^n\mathtt{w}^n$. *Then*

$$
\begin{aligned}
L(G) = \{\mathtt{a}\vec{x}_{n_0}\mathtt{a}\vec{x}_{n_1}\mathtt{a}\cdots\vec{x}_{n_{k-1}}\mathtt{ab}\vec{y}_{n_{k-1}}\mathtt{b}\cdots\mathtt{b}\vec{y}_{n_1}\mathtt{b}\vec{y}_{n_0}\mathtt{bc}^k\mathtt{d}^k :\\
k \in \omega, n_i \in \omega \textit{ for all } i < k\}
\end{aligned}
\tag{5.136}
$$

In particular, for every $\vec{x} \in L(G)$

(5.137) $\mu(\vec{x}) = m(\mathtt{a}+\mathtt{b}+\mathtt{c}+\mathtt{d}) + n(\mathtt{t}+\mathtt{u}+\mathtt{v}+\mathtt{w})$

for certain natural numbers m and n.

For example

$$\mathtt{aabbccdd},\ \mathtt{atuabvwbccdd},\ \mathtt{attuuatuabbvwbvvwwbcccddd},\ \ldots$$

are in $L(G)$ but not

$$\mathtt{atuabbcd},\ \mathtt{attuuatuabvwbvvwwbccdd}$$

Now for the promised proof that there is no TAG which can generate this language.

Lemma 5.53 *Let H be a TAG with* $L(H) = L(G)$ *and* $\mathfrak{B}$ *a centre or adjunction tree. Then*

(5.138) $\mu(\mathfrak{B}) = m_{\mathfrak{B}}(\mathtt{a}+\mathtt{b}+\mathtt{c}+\mathtt{d}) + n_{\mathfrak{B}}(\mathtt{t}+\mathtt{u}+\mathtt{v}+\mathtt{w})$

for certain natural numbers $m_{\mathfrak{B}}$ *and* $n_{\mathfrak{B}}$.

We put $\rho_{\mathfrak{B}} := n_{\mathfrak{B}}/m_{\mathfrak{B}}$. (This is ∞, if $m = 0$.) Certainly, there exists the minimum of all $\rho_{\mathfrak{B}}$ for all adjunction trees. It is easy to show that it must be 0. So there exists an adjunction tree which consists only of t, u, v and w, in equal number. Further there exists an adjunction tree which contains a.

Let $\vec{x}$ be a string from $L(G)$ such that

(5.139) $\mu(\vec{x}) = m(\mathtt{a}+\mathtt{b}+\mathtt{c}+\mathtt{d}) + n(\mathtt{t}+\mathtt{u}+\mathtt{v}+\mathtt{w})$

for certain natural numbers m and n such that (a) m is larger than any $m_{\mathfrak{B}}$, and (b) n/m is smaller than any $\rho_{\mathfrak{B}}$ that is not equal to 0. It is to be noticed that such a $\vec{x}$ exists. If m and n are chosen, the following string does the job.

(5.140) $\mathtt{a}\mathtt{t}^n\mathtt{u}^n\mathtt{a}^{m-1}\mathtt{b}^{n-1}\mathtt{v}^n\mathtt{w}^n\mathtt{b}\mathtt{c}^m\mathtt{d}^m$

This string results from a centre tree by adjoining (a′) an $\mathfrak{A}$ in which a occurs, by adjoining (b′) a $\mathfrak{B}$ in which a does not occur. Now we look at points in which $\mathfrak{B}$ has been inserted in (5.140). These can only be as follows.

(5.141) $\mathtt{a}\mathtt{t}^n \bullet \mathtt{u}^n\mathtt{a}^{m-1}\mathtt{b}^{n-1}\mathtt{v}^n \bullet \mathtt{w}^n\mathtt{b}\mathtt{c}^m\mathtt{d}^m$

However, let us look where the adjunction tree $\mathfrak{A}$ has been inserted.

(5.142) $\mathtt{at}^n\mathtt{u}^n\mathtt{a}^{m-1} \circ \mathtt{b}^{n-1}\mathtt{v}^n\mathtt{w}^n\mathtt{bc}^m \circ \mathtt{d}^m$

If we put this on top of each other, we get

(5.143) $\mathtt{at}^n \bullet \mathtt{u}^n\mathtt{a}^{m-1} \circ \mathtt{b}^{n-1}\mathtt{v}^n\mathtt{w}^n \bullet \mathtt{bc}^m \circ \mathtt{d}^m$

Now we have a contradiction. The points of adjunction may not cross! For the subword between the two $\bullet$ must be a constituent, likewise the part between the two $\circ$. However, these constituents are not contained in each other. (In order for this to become a real proof one has to reflect over the fact that the constituent structure is not changed by adjunction. This is Exercise 194.)

So we have a 2–LMG which generates a language that cannot be generated by a TAG. This grammar is 2–branching. In turn, 2–branching 2–LMGs are weaker than full linear 2–LMGs. Some parts of the argumentation shall be transferred to the exercises, since they are not of central concern.

Definition 5.54 *A linear LMG is called n–**branching** if the polynomial base consists of at most k–ary vector polynomials.*

The reason for this definition is the following fact.

Proposition 5.55 *Let $L = L(G)$ for some n–branching, k–linear LMG G. Then there exists a k–linear LMG H with $L(H) = L$ in which every rule is at most n–branching.*

To this end one has to see that a rule with more than n daughters can be replaced by a canonical sequence of rules with at most n daughters, if the corresponding vector polynomial is generated by at most n–ary polynomials. On the other hand it is not guaranteed that there is no n–branching grammar if higher polynomials have been used. Additionally, it is possible to construct languages such that essentially $n+1$–ary polynomials have been used and they cannot be reduced to at most n–ary polynomials. Define as before

(5.144) $\vec{x}_n := \mathtt{t}^n\mathtt{u}^n \qquad \vec{y}_n := \mathtt{v}^n\mathtt{w}^n$

The following polynomial is not generable using polynomials that are at most ternary.

(5.145) $\mathfrak{q}(\langle w_0, w_1\rangle, \langle x_0, x_1\rangle, \langle y_0, y_1\rangle, \langle z_0 z_1\rangle) := \langle w_0 x_0 y_0 z_0, y_1 w_1 z_1 x_1\rangle$

From this we can produce a proof that the following language cannot be generated by a 2–branching LMG.

(5.146) $L = \{\vec{x}_{n_0}\vec{x}_{n_1}\vec{x}_{n_2}\vec{x}_{n_3}\vec{y}_{n_2}\vec{y}_{n_0}\vec{y}_{n_3}\vec{y}_{n_1} : n_0, n_1, n_2, n_3 \in \omega\}$

We close this section with a few remarks on the semantics. Adjunction is an operation that takes complete trees as input and returns a complete tree. This concept is not easily coupled with a semantics that assembles the meanings of sentences from their parts. It is — at least in Montague semantics — impossible to recover the meaning components of a sentence after completion, which would be necessary for a compositional account. [**?**] only gives a modest sketch of how adjunction is done in semantics. Principally, for this to work one needs a full record of which items are correlated to which parts of meaning (which is assumed, for example, in many syntactic theories, for example LFG and HPSG).

Exercise 194. Let $\mathfrak{B}$ be a tree and $\mathfrak{A}$ an adjunction tree. Let $\mathfrak{C}$ be the result of adjoining $\mathfrak{A}$ to x in $\mathfrak{B}$. We view $\mathfrak{B}$ in a natural way as a subtree of $\mathfrak{C}$ with x the lower node of $\mathfrak{A}$ in $\mathfrak{C}$. Show the following: the constituents of $\mathfrak{B}$ are exactly the intersection of constituents of $\mathfrak{C}$ with the set of nodes of $\mathfrak{B}$.

Exercise 195. Show that the language $L := \{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n\mathtt{d}^n : n \in \omega\}$ cannot be generated by an unregulated TAG. *Hint.* Proceed as in the proof above. Take a string which is large enough so that a tree has been adjoined and analyze the places where it has been adjoined.

Exercise 196. Show that in the example above $\min\{\rho_{\mathfrak{B}} : \mathfrak{B} \in \mathbb{A}\} = 0$. *Hint.* Compare the discussion in Section 2.7.

Exercise 197. Show the following: *For every TAG G there is a TAG $G^\diamondsuit$ in standard form such that $G^\diamondsuit$ and G have the same constituent structures.* What can you say about the labelling function?

Exercise 198. Prove Proposition 5.55.

6. Index Grammars

Index grammars broaden the concept of CFGs in a very special way. They allow to use in addition of the nonterminals a sequence of indices; the manipulation of the sequences is however very limited. Therefore, we may consider these grammars alternatively as grammars that contain rule schemata rather

than individual rules. Let as usual A be our alphabet, N the set of nonterminals (disjoint with A). Now add a set I of **indices**, disjoint to both A and N. Furthermore, # shall be a symbol that does not occur in $A \cup N \cup I$. An **index scheme** σ has the form

(5.147) $$A^\frown\vec{\alpha} \to B_0{}^\frown\vec{\beta}_0 \quad \cdots \quad B_{n-1}{}^\frown\vec{\beta}_{n-1}$$

or alternatively the form

(5.148) $$A^\frown\vec{\alpha} \to a$$

where $\vec{\alpha}, \vec{\beta}_i \in I^* \cup \{\#\}$ for $i < n$, and $a \in A$. The schemata of the second kind are called **terminal schemata**. An **instantiation of** σ is a rule

(5.149) $$A^\frown\vec{x}\vec{\alpha} \to B_0{}^\frown\vec{y}_0\vec{\beta}_0 \quad \cdots \quad B_{n-1}{}^\frown\vec{y}_{n-1}\vec{\beta}_{n-1}$$

where the following holds.

① If $\vec{\alpha} = \#$ then $\vec{x} = \varepsilon$ and $\vec{y}_i = \varepsilon$ for all $i < n$.

② If $\vec{\alpha} \neq \#$ then for all $i < n$: $\vec{y}_i = \varepsilon$ or $\vec{y}_i = \vec{x}$.

③ For all $i < n$: if $\vec{\beta}_i = \#$ then $\vec{y}_i = \varepsilon$.

For a terminal scheme the following condition holds: if $\vec{\alpha} = \#$ then $\vec{x} = \varepsilon$. An index scheme simply codes the set of all of its instantiations. So we may also call it a **rule scheme**. If in a rule scheme σ we have $\vec{\alpha} = \#$ as well as $\vec{\beta}_i = \#$ for all $i < n$ then we have the classical case of a context free rule. We therefore call an index scheme **context free** if it has this form. We call it **linear** if $\vec{\beta}_i \neq \#$ for at most one $i < n$. Context free schemata are therefore also linear but the converse need not hold. One uses the following suggestive notation. $A[\,]$ denotes an A with an arbitrary stack; on the other hand, A is short for $A\#$. Notice for example the following rule.

(5.150) $$\mathtt{A[i]} \to \mathtt{B[\,]} \quad \mathtt{A} \quad \mathtt{C[ij]}$$

This is another form for the scheme

(5.151) $$\mathtt{Ai} \to \mathtt{BA\#Cij}$$

which in turn comprises all rules of the following form

(5.152) $$\mathtt{A}\vec{x}\mathtt{i} \to \mathtt{B}\vec{x} \quad \mathtt{A\#} \quad \mathtt{C}\vec{x}\mathtt{ij}$$

Definition 5.56 *We call an* ***index grammar*** *a sextuple* $G = \langle S, A, N, I, \#, R\rangle$ *where A, N, and I are pairwise disjoint finite sets not containing* #*,* $S \in N$ *the* ***start symbol*** *and R a finite set of index schemata over A, N, I and* #*. G is called* ***linear*** *or a* ***LIG*** *if all its index schemata are linear.*

The notion of a derivation can be formulated over strings as well as trees. (To this end one needs A, N and I to be disjoint. Otherwise the category symbols cannot be uniquely reconstructed from the strings.) The easiest is to picture an index grammar as a grammar $\langle S, N, A, R\rangle$, where in contrast to a context free rule set we have put an infinite set of rules which is specified by means of schemata, which may allow infinitely many instantiations. This allows us to transfer many notions to the new type of grammars. For example, it is easily seen that for an index grammar there is a 2–standard form which generates the same language.

The following is an example of an index grammar. Let $A := \{\mathtt{a}\}$, $N := \{\mathtt{S}, \mathtt{T}, \mathtt{U}\}$, $I := \{\mathtt{i}, \mathtt{j}\}$, and

$$
\begin{array}{lll}
\mathtt{S}[\,] \to \mathtt{T}[\mathtt{j}] & \mathtt{T}[\,] \to \mathtt{T}[\mathtt{i}] & \\
\mathtt{T}[\mathtt{i}] \to \mathtt{U}[\,] & \mathtt{U}[\mathtt{i}] \to \mathtt{U}[\,] & \mathtt{U}[\,] \\
\mathtt{U}[\mathtt{j}] \to \mathtt{a} & &
\end{array}
\tag{5.153}
$$

This defines the grammar G. We have $L(G) = \{\mathtt{a}^{2^n} : n \in \omega\}$. As an example, look at the following derivation.

$$
\begin{array}{lll}
\langle \mathtt{S}, & \mathtt{Tj} & \mathtt{Tji}, \\
\mathtt{Tjii}, & \mathtt{Tjiii}, & \mathtt{Ujii}, \\
\mathtt{UjiUji}, & \mathtt{UjiUjUj}, & \mathtt{UjUjUjUj}, \\
\mathtt{aUjUjUj}, & \mathtt{aaUjUj}, & \mathtt{aaaUj}, \\
\mathtt{aaaa}\rangle & &
\end{array}
\tag{5.154}
$$

Index grammars are therefore quite strong. Nevertheless, one can show that they too can only generate **PTIME**–languages. (For index grammars one can define a variant of the chart–algorithm This variant also needs only polynomial time.) Of particular interest are the linear index grammars.

Now we turn to the equality between LIGs and TAGs. Let G be an LIG; we shall construct a TAG which generates the same constituent structures. We shall aim for roughly the same proof as with CFGs. The idea is again to look for nodes x and y with identical label $X\vec{x}$. This however can fail. For on the one hand we can expect to find two nodes with identical label from N,

but they may have different index stack. It may happen that no such pair of nodes exists. Therefore we shall introduce the first simplification. We only allow rules of the following form.

(5.155a) $X[i] \to Y_0 \cdots Y_{j-1}\ Y_j[\,]\ Y_{j+1} \cdots Y_{n-1}$

(5.155b) $X[\,] \to Y_0 \cdots Y_{j-1}\ Y_j[i]\ Y_{j+1} \cdots Y_{n-1}$

(5.155c) $X \to Y_0 \cdots Y_{n-1}$

(5.155d) $X \to a$

In other words, we only admit rules that stack or unstack a single letter, or which are context free. Such a grammar we shall call **simple**. It is clear that we can turn G into simple form while keeping the same constituent structures. Then we always have the following property. If x is a node with label $X\vec{x}$ and if x immediately dominates the node x' with label $Y\vec{x}i$ then there exists a node $y' \leq x'$ with label $V\vec{x}i$ which immediately dominates a node with label $W\vec{x}$. At least the stacks are now identical, but we need not have $Y = V$. To get this we must do a second step. We put $N' := N^2 \times \{o, e, a\}$ (but write $\langle A, B\rangle^x$ in place of $\langle A, B, x\rangle$). The superscript keeps score of the fact whether at this point we stack an index (a), we unstack a letter (e) or we do nothing (o). The index alphabet is $I' := N^2 \times I$. The rules above are now reformed as follows. (For the sake of perspicuity we assume that $n = 3$ and $j = 1$.) For a rule of the form (5.155b) we add all rules of the form

(5.156) $\langle X, X'\rangle^a \to \langle Y_0, Y_0'\rangle^{a/o}\ \langle Y_1, Y_1'\rangle^{a/o}[\langle X, X', i\rangle]\ \langle Y_2, Y_2'\rangle^{a/o}$

So we stack in addition to the index i also the information about the label with which we have started. The superscript a is obligatory for $\langle X, X'\rangle$! From the rules of the form (5.155a) we make rules of the following form.

(5.157) $\langle X, X'\rangle^{a/o}[\langle W, Y_1', i\rangle] \to \langle Y_0, Y_0'\rangle^{a/o} \quad \langle W, Y_1'\rangle^e \quad \langle Y_2, Y_2'\rangle^{a/o}$

However, we shall also add these rules:

(5.158) $\langle Y_1, Y_1'\rangle^e \to \langle Y_1', Z\rangle^{a/o}$

for all $Y_1, Y_1', Z \in N$. The rules of the form (5.155c) are replaced thus.

(5.159) $\langle X, X'\rangle^o \to \langle Y_0, Y_0'\rangle^{a/o} \quad \langle Y_1, Y_1'\rangle^{a/o} \quad \langle Y_2, Y_2'\rangle^{a/o}$

Finally, the rules of the form (5.155d) are replaced by these rules.

(5.160) $\langle X, X' \rangle^o \to a$

We call this grammar $G^\spadesuit$. We shall at first see why G and $G^\spadesuit$ generate the same constituent structures. To this end, let us be given a $G^\spadesuit$–derivation. We then get a G–derivation as follows. Every symbol of the form $\langle X, X' \rangle^{a/e/o}$ is replaced by X, every stack symbol $\langle X, X', i \rangle$ by i. Subsequently, the rules of type (5.158) are skipped. This yields a G–derivation, as is easily checked. It gives the same constituent structure. Conversely, let a G–derivation be given with associated ordered labelled tree $\mathfrak{B}$. Then going from bottom to top we do the following. Suppose a rule of the form (5.155b) has been applied to a node x and that i has been stacked. Then look for the highest node $y < x$ where the index i has been unstacked. Let y have the label B, x the label A. Then replace A by $\langle A, B \rangle^a$ and the index i on all nodes up to y by $\langle A, B, i \rangle$. In between x and y we insert a node y^* with label $\langle A, B \rangle^e$. y^* has y as its only daughter. y keeps at first the label B. If however no symbol has been stacked at x then exchange the label A by $\langle A, A' \rangle^o$, where A' is arbitrary. If one is at the bottom of the tree, one has a $G^\spadesuit$–tree. Again the constituent structures have been kept, since only unary rules have been inserted.

Now the following holds. If at x the index $\langle A, B, i \rangle$ has been stacked then x has the label $\langle A, B \rangle^a$ and there is a node y below x at which this index is again removed. It has the label $\langle A, B \rangle^e$. We say that y is **associated to** x. Now define as in the case of CFLs centre trees as trees whose associated string is a terminal string and in which no pair of associated nodes exist. It is easy to see that in such trees no symbol is ever put on the stack. No node carries a stack symbol and therefore there are only finitely many such trees. Now we define the adjunction trees. These are trees in which the root has label $\langle A, B \rangle^a$ exactly one leaf has a nonterminal label and this is $\langle A, B \rangle^e$. Further, in the interior of the tree no pair of associated nodes shall exist. Again it is clear that there are only finitely many such trees. They form the basic set of our adjunction trees. However, we do the following. The labels $\langle X, X' \rangle^o$ we replace by $\langle X, X' \rangle$, the labels $\langle X, X' \rangle^a$ and $\langle X, X' \rangle^e$ by $\langle X, X' \rangle^\triangledown$. (Root and associated node get an adjunction prohibition.) Now the proof is as in the context free case.

Now let conversely a TAG $G = \langle \mathbb{C}, N, A, \mathbb{A} \rangle$ be given. We shall construct a LIG which generates the same constituent structures. To this end we shall assume that all trees from $\mathbb{C}$ and $\mathbb{A}$ are based on pairwise disjoint sets of nodes. Let K be the union of all sets of nodes. This is our set of nonterminals. The set $\mathbb{A}$ is our set of indices. Now we formulate the rules. Let $i \to j_0 \quad j_1 \cdots j_{n-1}$

be a local subtree of a tree.
(A) i is not central. Then add

(5.161) $\quad i \to j_0 \quad j_1 \cdots j_{n-1}$

(B) Let i be root of $\mathfrak{T}$ and j_k central (and therefore not a distinguished leaf). Then add

(5.162) $\quad i[\,] \to j_0 \quad j_{k-1} \quad j_k[\mathfrak{T}] \quad j_{k+1} \cdots j_{n-1}$

(C) Let j_k be a distinguished leaf of $\mathfrak{T}$. Then add

(5.163) $\quad i[\mathfrak{T}] \to j_0 \quad j_{k-1} \quad j_k[\,] \quad j_{k+1} \cdots j_{n-1}$

(D) Let i be central in $\mathfrak{T}$, but not a root and j_k central but not a distinguished leaf. Then let

(5.164) $\quad i[\,] \to j_0 \cdots j_{k-1} \quad j_k[\,] \quad j_{k+1} \cdots j_{n-1}$

be a rule. Nothing else shall be a rule. This defines the grammar G^I. (This grammar may have start trees over distinct start symbols. This can be remedied.) Now we claim that this grammar generates the same constituent structures over A. This is done by induction over the length of the derivation. Let $\mathfrak{T}$ be a centre tree, say $\mathfrak{T} = \langle B, <, \sqsubset, \ell \rangle$. Then let $\mathfrak{T}^I := \langle B, <, \sqsubset, \ell^I \rangle$, where $\ell^I(i) := i$ if i is nonterminal and $\ell^I(i) := \ell(i)$ otherwise. One establishes easily that this tree is derivable. Now let $\mathfrak{T} = \langle B, <, \sqsubset, \ell \rangle$ and $\mathfrak{T}^I = \langle B, <, \sqsubset, \ell^I \rangle$ already be constructed; let $\mathfrak{U} = \langle C, <', \sqsubset', \ell' \rangle$ result from $\mathfrak{T}$ by adjoining a tree $\mathfrak{B}$ to a node x. By making x into the root of an adjoined tree we get $B \subseteq C$, $<' \cap B^2 = <$, $\sqsubset' \cap B^2 = \sqsubset$ and $\ell' \restriction B = \ell$. Now $\mathfrak{U}^I = \langle C, <', \sqsubset', \ell'^I \rangle$. Further, there is an isomorphism between the adjunction tree $\mathfrak{B}$ and the local subtree induced on $C \cup \{x\}$. Let $\pi \colon C \cup \{x\} \to B$ be this isomorphism. Put $\ell'^I(y) := \ell^I(y)$ if $y \in B - C$. Put $\ell'^I(y) := \pi(y)$ if $\pi(y)$ is not central; and put $\ell'^I(y) := \ell'^I(x) := \ell^I(x)$ if y is a distinguished leaf. Finally, assume $\ell^I(x) = X\vec{x}$, where X is a nonterminal symbol and $\vec{x} \in I^*$. If y is central but not root or leaf then put

(5.165) $\quad \ell'^I(y) := \pi(y)\vec{x}\mathfrak{B}$

Now it is easily checked that the so–defined tree is derivable in G^I. We have to show likewise that if $\mathfrak{U}$ is derivable in G^I there exists a tree $\mathfrak{U}^A$ with $(\mathfrak{U}^A)^I \cong \mathfrak{U}$

which is derivable in G. To this end we use the method of disembedding. One looks for nodes x and y such that they have the same stack, $x > y$, there is no element between the two that has the same stack. Further, there shall be no such pair in $\downarrow x - (\downarrow y \cup \{x\})$. It is easily seen that this tree is isomorphic to an adjunction tree. We disembed this tree and gets a tree which is strictly smaller. (Of course, the existence of such a tree must still be shown. This is done as in the context free case. Choose x of minimal height such that such there exists a $y < x$ with identical stack. Subsequently, choose y maximal with this property. In $\downarrow x - (\downarrow y \cup \{x\})$ there can then be no pair x', y' of nodes with identical stack such that $y' < x'$. Otherwise, x would not be minimal.) We summarize.

Theorem 5.57 *A set of constituent structures is generated by a linear index grammar iff it is generated by a TAG.*

We also say that these types of grammars are equivalent in constituent analysis.

A rule is called **right linear** if the index is only passed on to the right hand daughter. So, the right hand rule is right linear, the left hand rule is not:

(5.166) $\quad \mathtt{A}[\,] \to \mathtt{B} \quad \mathtt{C}[\mathtt{i}] \quad \mathtt{B}, \qquad \mathtt{A}[\,] \to \mathtt{B} \quad \mathtt{C} \quad \mathtt{B}[\mathtt{i}]$

An index grammar is called **right linear** if all of its rules are right linear. Hence it is automatically linear. The following is from [**?**; **?**].

Theorem 5.58 (Michaelis & Wartena) *A language is generated by a right linear index grammar iff it is context free.*

Proof. Let G be right linear, $X \in N$. Define H_X as follows. The alphabet of nonterminals has the form $T := \{X^\diamond : X \in N\}$. The alphabet of terminals is the one of G, likewise the alphabet of indices. The start symbol is X. Now for every rule

(5.167) $\quad A[\,] \to B_0 \cdots B_{n-1} \quad B_n[i]$

we add the rule

(5.168) $\quad A^\diamond[\,] \to A \quad B^\diamond[i]$

This grammar is right regular and generates a CFL (see the exercises). So there exists a CFG $L_X := \langle S^L_X, N^L_X, N, R^L_X \rangle$ which generates $L(H_X)$. (Here N is

the alphabet of nonterminals of G but the terminal alphabet of L_X.) We assume that N_L^X is disjoint to our previous alphabets. We put $N' := \bigcup N_X^L \cup N$ as well as $R' := \bigcup R_X^L \cup R \cup R^-$ where R is the set of context free rules of G and R^- the set of rules $A[\,] \to B_0 \cdots B_{n-1}$ such that $A[\,] \to B_0 \cdots B_{n-1} \quad B_n[i] \in R$. Finally, let $G' := \langle S_L, N', A, R' \rangle$. G' is certainly context free. It remains to show that $L(G') = L(G)$. To this end let $\vec{x} \in L(G)$. There exists a tree $\mathfrak{B}$ with associated string $\vec{x}$ which is derived from G. By induction over the height of this tree one shows that $\vec{x} \in L(G')$. The inductive hypothesis is this: *For every G–tree $\mathfrak{B}$ with associated string $\vec{x}$ there exists a G'–tree $\mathfrak{B}'$ with associated string $\vec{x}$; and if the root of $\mathfrak{B}$ carries the label $X\vec{x}$ then the root of $\mathfrak{B}'$ carries the label X*. If $\mathfrak{B}$ contains no stack symbols, this claim is certainly true. Simply take $\mathfrak{B}' := \mathfrak{B}$. Further, the claim is easy to see if the root has been expanded with a context free rule. Now let this not be the case; let the tree have a root with label U. Let P be the set of right hand nodes of $\mathfrak{B}$. For every $x \in P$ let $B(x)$ be that tree which contains all nodes which are below x but not below any $y \in P$ with $y < x$. It is easy to show that these sets form a partition of $\mathfrak{B}$. Let $u \prec x$, $u \notin P$. By induction hypothesis, the tree dominated by u can be restructured into a tree $\mathfrak{T}_u$ which has the same associated string and the same root label and which is generated by G'. The local tree of x in $B(x)$ is therefore an instance of a rule of R^-. We denote the tree obtained from x in such a way by $\mathfrak{B}'_x$. $\mathfrak{B}'_x$ is a G'–tree. Furthermore: if $y < x$, $y \in P$, and if $u < x$ then $u \sqsubset y$. Therefore we have that $P = \{x_i : i < n\}$ is an enumeration with $x_i > x_{i+1}$ for all $i < n-1$. Let A_i be the root label of x_i in $\mathfrak{B}'_{x_i}$. The string $\prod_{i<n} A_i$ is a string of H_U. Therefore it is generated by L_U. Hence it is also generated by G'. So, there exists a tree $\mathfrak{C}$ associated to this string. Let the leaves of this tree be exactly the x_i and let x_i have the label A_i. Then we insert $\mathfrak{B}'_{x_i}$ at the place of x_i for all $i < n$. This defines $\mathfrak{D}$. $\mathfrak{D}$ is a G'–tree with associated string $\vec{x}$. The converse inclusion is left to the reader. □

We have already introduced Combinatory Categorial Grammars (CCGs) in Section 3.4. The concept of these grammars was very general. In the literature, the term CCG is usually fixed — following Mark Steedman — to a particular variant where only those combinators may be added that perform function application and generalized function composition. In order to harmonize the notation, we revise it as follows.

(5.169) $\alpha|^+\beta$ replaces α/β, $\alpha|^-\beta$ replaces $\beta\backslash\alpha$

We take p_i as a variable for elements from $\{+,-\}$. A **category** is a well formed string over $\{B, (,), |^+, |^-\}$. We agree on *obligatory* left associative bracket-

ing. That means that the brackets that we do not need to write assuming left associativity actually are *not* present in the string. Hence $\mathtt{a}|^{+}\mathtt{b}|^{-}\mathtt{c}$ is a category, as is $\mathtt{a}|^{+}(\mathtt{b}|^{-}\mathtt{c})$. However, $((\mathtt{a}|^{+}\mathtt{b})|^{-}\mathtt{c})$ and $(\mathtt{a}|^{+}(\mathtt{b}|^{-}\mathtt{c}))$ are not. A **block** is a sequence of the form $|^{+}a$ or $|^{-}a$, a basic, or of the form $|^{+}(\beta)$ or $|^{-}(\beta)$, where β is a complex category symbol. (Often we ignore the details of the enclosing brackets.) A **p–category** is a sequence of blocks, seen as a string. With this a category is simply a string of the form $\alpha^{\frown}\Delta$ where Δ is a p–category. If Δ and Δ' are p–categories, so is $\Delta^{\frown}\Delta'$. For a category α we define by induction the head, α, $K(\alpha)$, as follows.

① $K(b) := b$.

② $K(\alpha|^{+}\beta) := K(\alpha|^{-}\beta) := K(\alpha)$.

Lemma 5.59 *Every category α can be uniquely segmented as $\alpha = K(\alpha)^{\frown}\Delta$ where Δ is a p–category.*

If we regard the sequence simply as a string we can use $^{\frown}$ as the concatenation symbol of blocks as well as of sequences of blocks. We admit the following operations. (If β is basic, omit the additional enclosing brackets.)

$$\alpha|^{+}(\beta) \circ_1 \beta := \alpha \tag{5.170}$$

$$\beta \circ_2 \alpha|^{-}(\beta) := \alpha \tag{5.171}$$

$$\alpha|^{+}(\beta) \circ_3^n \beta^{\frown}\Delta^n := \alpha^{\frown}\Delta^n \tag{5.172}$$

$$\beta^{\frown}\Delta^n \circ_4^n \alpha|^{-}(\beta) := \alpha^{\frown}\Delta^n \tag{5.173}$$

Here Δ^n is a variable for p–categories consisting of n blocks. In addition it is possible to restrict the choice of heads for α and β. This means that we define operations $\circ_i^{F,A,n}$ in such a way that

$$\alpha \circ_i^{L,R,n} \beta := \begin{cases} \alpha \circ^n \beta & \text{if } K(\alpha) \in L, K(\beta) \in R, \\ \star & \text{otherwise.} \end{cases} \tag{5.174}$$

This means that we have to step back from our ideal to let the categories be solely determined by the combinators.

Definition 5.60 *A **combinatory categorial grammar** (or **CCG**) is a categorial grammar which uses finitely many operations from*

$$\{\circ_1^{L,R}, \circ_2^{L,R} : L, R \subseteq B\} \cup \{\circ_3^{L,R,n}, \circ_4^{L,R,n} : n \in \omega, L, R \subseteq B\} \tag{5.175}$$

Notice by the way that $\circ_1 = \circ_3^0$ and $\circ_2 = \circ_4^n$. This simplifies the calculations.

Lemma 5.61 *Let G be a CCG over A and M the set of categories which are subcategories of some $\alpha \in \zeta(a)$, $a \in A$. Then the following holds. If $\vec{x}$ is a string of category α in G then $\alpha = \beta^\frown\Delta$ where $\alpha \in M$ and Δ is a p–category over M.*

The proof is by induction over the length of $\vec{x}$ and is left as an exercise.

Theorem 5.62 *For every CCG G there exists a linear index grammar H which generates the same trees.*

Proof. Let G be given. In particular, G associates with every letter $a \in A$ a finite set $\zeta(a)$ of categories. We consider the set M of subterms of categories from $\bigcup\langle\zeta(a) : a \in A\rangle$. This is a finite set. We put $N := M$ and $I := M$. By Lemma 5.61, categories can be written as pairs $\alpha[\Delta]$ where $\alpha \in N$ and Δ is a p–category over I. Further, there exist finitely many operations which we write as rules. Let for example $\circ_1^{L,R}$ be an operation. This means that we have rules of the form

(5.176a) $\alpha \to \alpha\,|^+a \quad a$

(5.176b) $\alpha \to \alpha\,|^+(\beta) \quad \beta$

where $K(\beta) \in L$ and $K(\alpha) \in R$, and β not basic. We write this into linear index rules. Notice that in any case $\beta \in M$ because of Lemma 5.61. Furthermore, we must have $\alpha \in M^+$. So we write down all the rules of the form

(5.177) $\alpha \to \delta[\Delta] \quad \beta$

where $\delta[\Delta] = \alpha\,|^+\beta$ for certain $\alpha, \Delta \in M^*$ and $\delta, \beta \in M$. We can group these into finitely many rule schemata. Simply fix β where $K(\beta) \in R$. Let $\mathbb{B}$ be the set of all sequences $\langle\gamma_i : i < p\rangle \in M^*$ whose concatenation is $|^+\beta$. $\mathbb{B}$ is finite. Now put for (5.177) all rules of the form

(5.178) $\alpha'[] \to \alpha'[\Delta] \quad \beta$

where $\alpha' \in M$ is arbitrary with $K(\alpha) \in L$ and $\Delta \in \mathbb{B}$. Now one can see easily that every instance of (5.177) is an instance of (5.178) and conversely.

Analogously for the rules of the following form.

(5.179) $\alpha \to \beta \quad \alpha\,|^-\beta$

In a similar way we obtain from the operations $\circ_3^{L,R,n}$ rules of the form

(5.180) $\quad \alpha^\frown\Delta^n \to \alpha\,|^+\beta \quad \beta^\frown\Delta^n$

where $K(\alpha) \in L$ and $K(\beta) \in R$. Now it turns out that, because of Lemma 5.61 $\Delta^n \in M^n$ and $\beta \in M$. Only α may again be arbitrarily large. Nevertheless we have $\alpha \in M^+$, because of Lemma 5.61. Therefore, (5.180) only corresponds to finitely many index schemata. □

The converse does not hold: for the trees which are generated by an LIG need not be 3–branching. However, the two grammar types are weakly equivalent.

Notes on this section. There is a descriptive characterization of indexed languages akin to the results of Chapter 6 which is presented in [**?**]. The idea there is to replace the index by a so–called contingency function, which is a function on the nodes of the constituent structure that codes the adjunction history.

Exercise 199. Show the following claim. *For every index grammar G there is an index grammar H in 2–standard form such that $L(G) = L(H)$. If G is linear (context free) H can be chosen linear (context free) as well.*

Exercise 200. Prove the Lemma 5.61.

Exercise 201. Write an index grammar that generates the sentences of predicate logic. (See Section 2.7 for a definition.)

Exercise 202. Let *NB* be the set of formulae of predicate logic with = and ∧ in which every quantifier binds at least one occurrence of a variable. Show that there is no index grammar that generates *NB*. *Hint.* It is useful to concentrate on formulae of the form QM, where Q is a sequence of quantifiers and M a formula without quantifiers (but containing any number of conjuncts). Show that in order to generate these formulae from *NB*, a branching rule is needed. Essentially, looking top down, the index stack has to memorize which variables have been abstracted over, and the moment that there is a branching rule, the stack is passed on to both daughters. However, it is not required that the left and right branch contain the same variables.

7. Compositionality and Constituent Structure

In this section we return to the discussion of compositionality, which we started in Chapter 3. Our concern is how constituent structure and compo-

sitionality constrain each other. In good cases this shows that the semantics of a language does not allow a certain syntactic analysis. This will allow to give substance to the distinction between weak and strong generative capacity of grammar types.

Recall once again Leibniz' Principle. It is defined on the basis of constituent substitution and truth equivalence. However, constituent substitution is highly problematic in itself, for it hardly ever is string substitution. If we do string substitution of `fast` by `good` in (5.181), we get (5.182) and not the correct (5.183).

(5.181) `Simon is faster than Paul.`

(5.182) *`Simon is gooder than Paul.`

(5.183) `Simon is better than Paul.`

Notice that substitution in λ–calculus and predicate logic also is not string substitution but something more complex. (This is true even when variables are considered simple entities.) What makes matters even more difficult in natural languages is the fact that there seems to be no uniform algorithm to perform such substitution. Another, related problem is that of determining occurrences. For example, does `cater` occur as a subexpression in the word `caterpillar`, `berry` as a subexpression of `cranberry`? What about `kick` in `kick the bucket`? Worse still, does `tack` occur as a subexpression of `stack`, `rye` as a subexpression of `rice`? Obviously, no one would say that `tack` occurs in `stack`, and that by Leibniz' Principle its meaning is distinct from `needle` since there is no word `sneedle`. Such an argument is absurd. Likewise, in the formula $(p0 \wedge p01)$, the variable `p` does not occur, even though the string `p` is a substring of the formula as a string.

To be able to make progress on these questions we have to resort to the distinction between language and grammar. As the reader will see in the exercises, there is a tight connection between the choice of constituents and the meanings these constituents can have. If we fix the possible constituents and their meanings this eliminates some but not all choices. However it does settle the question of identity in meaning and can then lead to a detection of subconstituents. For if two given expressions have the same meaning we can conclude that they can be substituted for each other without change in the truth value in any given sentence on condition that they also have the same category. (Just an aside: sometimes substitution can be blocked by the exponents so that substitution is impossible even when the meanings and the

category are the same. These facts are however generally ignored. See below for further remarks.) So, Leibniz' Principle does tell us something about which substitutions are legitimate, and which occurrences of substrings are actually occurrences of a given expression. If `sneedle` does not exist we can safely conclude that `tack` has no proper occurrence in `stack` if substitution is simply string substitution. Moreover, Leibniz' Principle also says that if two expressions are intersubstitutable everywhere without changing the truth value, then they have the same meaning.

Definition 5.63 *Let* $\mathfrak{s}$ *be a structure term with a single free variable, x, and* $\mathfrak{u}$ *a structure term unfolding to* τ*. If* $[\mathfrak{u}/x]\mathfrak{s}$ *unfolds to* σ *we say that the sign* τ ***occurs in*** σ ***under the analyses*** $\mathfrak{s}$ ***and*** $\mathfrak{u}$*. Suppose now that* $\mathfrak{u}'$ *is a structure term unfolding to* τ'*, and that* $[\mathfrak{u}'/x]\mathfrak{s}$ *is definite and unfolds to* σ'*. Then we say that* σ' ***results from*** σ ***by replacing the occurrence of*** τ ***by*** τ' ***under the analyses*** $\mathfrak{s}$***,*** $\mathfrak{u}$ ***and*** $\mathfrak{u}'$*.*

This definition is complicated since a given sign may have different structure terms, and before we can define the substitution operation on a sign we must fix a structure term for it. This is particularly apparent when we want to define simultaneous substitution. Now, in ordinary parlance one does not usually mention the structure term. And substitution is typically defined not on signs but on exponents (which are called **expressions**). This, however, is dangerous and the reason for much confusion. For example, we have proved in Section 3.1 that almost every recursively enumerable sign system has a compositional grammar. The proof used rather tricky functions on the exponents. Consequently, there is no guarantee that if $\vec{y}$ is the exponent of τ and $\vec{x}$ the exponent of σ there is anything in $\vec{x}$ that resembles $\vec{y}$. Contrast this with CFGs, where a subexpression is actually also a substring. To see the dangers of this we discuss the theory of compositionality of [**?**]. Hodges discusses in passim the following principle, which he attributes to Tarski (from [**?**]). The original formulation in [**?**] was flawed. The correct version according to Hodges (p.c.) is this.

> *Tarski's Principle.* If there is a μ–meaningful structure term $[\mathfrak{s}/x]\mathfrak{u}$ unequal to $\mathfrak{u}$ and $[\mathfrak{s}'/x]\mathfrak{u}$ also is a μ–meaningful structure term with $[\mathfrak{s}'/x]\mathfrak{u} \sim_\mu [\mathfrak{s}/x]\mathfrak{u}$ then $\mathfrak{s} \sim_\mu \mathfrak{s}'$.

Notice that the typed λ–calculus satisfies this condition. Hodges dismisses

Tarski's Principle on the following grounds.

(5.184) `The beast ate the meat.`

(5.185) `The beast devoured the meat.`

(5.186) `The beast ate.`

(5.187) *`The beast devoured.`

Substituting `the beast ate` by `the beast devoured` in (5.184) yields a meaningful sentence, but it does not with (5.186). (As an aside: we consider the appearance of the upper case letter as well as the period as the result of adding a sign that turns the proposition into an assertion. Hence the substitution is performed on the string beginning with a lower case `t`.) Thus, substitutability in one sentence does not imply substitutability in another, so the argument goes. The problem with this argument is that it assumes that we can substitute `the beast ate` for `the beast devoured`. Moreover, it assumes that this is the effect of replacing a structure term u by u' in some structure term for (5.184). Thirdly, it assumes that if we perform the same substitution in a structure term for (5.186) we get (5.187). Unfortunately, none of these assumptions is justified. (The pathological examples of Section 3.1 should suffice to destroy this illusion.) What we need is a strengthening of the conditions concerning admissible operations on exponents. In the example sentence, the substituted strings are actually nonconstituents, so even under standard assumptions they do not constitute counterexamples. We can try a different substitution, for example replacing `the meat` by ε. This is a constituent substitution under the ordinary analysis. But this does not save the argument, for we do not know which grammar underlies the examples. It is not clear that Tarski's Principle is a good principle. But the argument against it is fallacious.

Obviously, what is needed is a restriction on the syntactic operations. In this book, we basically present two approaches. One is based on polynomials (noncombinatorial LMGs), the other on λ–terms for strings. In both cases the idea is that the functions should not destroy any material (ideally, each rule should *add* something). In this way the notion of composition does justice to the original meaning of the word. (Compositionality derives from Latin compositiō 'the putting together'.) Thus, every derived string is the result of applying some polynomial applied to certain vectors, and this polynomial determines the structure as well as — indirectly — the meaning and the category. Both approaches have a few abstract features in common. For example,

that the application of a rule is progressive with respect to some progress measurse. (For LMGs this measure is the combined length of the parts of the string vector.)

Definition 5.64 *A **progress measure** is a function $\mu : E \to \omega$. A function $f : E^n \to E$ is **progressive with respect to** μ if*

(5.188) $\mu(f(\vec{e})) > \max\{\mu(e_i) : i < n\}$

*f is **strictly progressive** if*

(5.189) $\mu(f(\vec{e})) > \sum_{i<n} \mu(e_i)$

*A sign grammar is (**strictly**) **progressive with respect to** μ if for all modes f, f^{ε} is (strictly) progressive.*

For example, a CFG is progressive with respect to length if it has no unary rules, and no empty productions (and then it is also strictly progressive). Let μ be a progress measure, $\mathfrak{A}$ a progressive grammar generating Σ. Then a given exponent e can be derived with a term that has depth at most $\mu(e)$. This means that its length is $\leq \Omega_{\mathsf{T}}^{\mu(e)}$, where $\Omega_{\mathsf{T}} := \max\{\Omega(f) : f \in F\}$. The number of such terms is $\leq |F|^{\Omega_{\mathsf{T}}^{\mu(e)}}$, so it is doubly exponential in $\mu(e)$! If $\mathfrak{A}$ is strictly progressive, the length of the structure term is $\leq \mu(e)$, so we have at most $|F|^{\mu(e)}$ many. Now, finally, suppose that the unfolding of a term is at most exponential in its length, then we can compute for strictly progressive grammars in time $O(2^{c\mu(e)})$ whether e is in $\pi_0[\Sigma]$.

Theorem 5.65 *Suppose that $\mathfrak{A}$ is strictly progressive with respect to the progress measure μ. Assume that computing the unfolding of a term can be done in time exponential in the length. Then for every e, '$e \in \pi_0[\Sigma]$' can be solved in time $O(c^{\mu(e)})$ for some $c > 0$. If $\mathfrak{A}$ is only progressive, '$e \in \pi_0[\Sigma]$' can be solved in time $O(c^{c^{\mu(e)}})$ for some $c > 0$.*

Notice that this one half of the finite reversibility for grammars (see Definition 4.85). The other half requires a similar notion of progress in the semantics. This would correspond to the idea that the more complex a sentence is the more complex its meaning. (Unlike in classical logic, where for example `(p∨(¬p))` is simpler than `p`.)

We still have not defined the notion of intersubstitutability that enters Leibniz' Principle. We shall give a definition based on a grammar. Recall

that for Leibniz' Principle we need a category of sentences, and a notion of truth. So, let $\mathfrak{A}$ be a sign grammar and $\mathtt{S}$ a category. Put

(5.190) $\quad \mathrm{Sent}_{\mathfrak{A},\mathtt{S}} := \{\mathfrak{s} : \mathfrak{s}^{\mu} = \mathtt{S}\}$

This defines the set of sentential terms (see also Section 6.1). Let $\mathfrak{t}$ be a structure term with a single occurrence of a free variable, say x. Then given $\mathfrak{s}$, $[\mathfrak{s}/x]\mathfrak{t}$ if definite is the result of putting $\mathfrak{s}$ in place of x. Thus $\mathfrak{t}^{\varepsilon} \in \mathrm{Pol}_1(\mathfrak{E})$. We define the **context set** of e as follows.

(5.191) $\quad \mathrm{Cont}_{\mathfrak{A},\mathtt{S}}(e) := \{\mathfrak{t}^{\varepsilon} : \text{for some } \mathfrak{s} \text{ such that } \mathfrak{s}^{\varepsilon} = e :$
$[\mathfrak{s}/x]\mathfrak{t} \in \mathrm{Sent}_{\mathfrak{A},\mathtt{S}}\}$

We shall spell this out for a CFG. In a CFG, $\mathrm{Cont}_{\mathfrak{A},\mathtt{S}} \in \mathrm{Pol}_1(\mathfrak{Z}(A))$. Moreover, if x occurs only once, the polynomials we get are quite simple: they are of the form $p(x) = \vec{u}^{\frown}x^{\frown}\vec{v}$ for certain strings $\vec{u}$ and $\vec{v}$. Putting $C := \langle \vec{u}, \vec{v} \rangle$, $p(\vec{x}) = C(\vec{x})$. Thus, the context set for $\vec{x}$ defined in (5.191) is the set of all C such that $C(\vec{x})$ is a sentence, and C is a constituent occurrence of $\vec{x}$ in it. Thus, (5.191) defines the substitution classes of the exponents. We shall also define what it means to be syntactically indistinguishable in a sign system.

Definition 5.66 *e and e' are **syntactically indistinguishable** — we write $e \simeq_{\Sigma} e'$ — iff*

① *for all $c \in C$ and all $m \in M$: if $\langle e, c, m \rangle \in \Sigma$ then there is an $m' \in M$ such that $\langle e', c, m' \rangle \in \Sigma$ and*

② *for all $c \in C$ and all $m' \in M$: if $\langle e', c, m' \rangle \in \Sigma$ then there is an $m \in M$ such that $\langle e, c, m \rangle \in \Sigma$.*

This criterion defines which syntactic objects should belong to the same substitution category. Obviously, we can also use Husserl's criterion here. However, there is an intuition that certain sentences are semantically but not syntactically well formed. Although the distinction between syntactic and semantic well–formedness breaks the close connection between syntactic categories and context sets, it seems intuitively justified. Structural linguistics, following Zellig Harris and others, typically defines categories in this way, using context sets. We shall only assume here that categories may not distinguish syntactic objects finer than the context sets.

Definition 5.67 *Let Σ be a system of signs. A sign grammar $\mathfrak{A}$ that generates Σ is* ***natural with respect to*** $\mathtt{S}$ *if* $\mathrm{Cont}_{\mathfrak{A},\mathtt{S}}(e) = \mathrm{Cont}_{\mathfrak{A},\mathtt{S}}(e')$ *implies* $e \simeq_\Sigma e'$.

A context free sign grammar is natural iff the underlying CFG is reduced. Here is an example. Let

(5.192) $\Sigma := \{\langle \mathtt{a},\mathtt{A},0\rangle, \langle \mathtt{b},\mathtt{B},0\rangle, \langle \mathtt{c},\mathtt{C},3\rangle, \langle \mathtt{ac},\mathtt{S},5\rangle, \langle \mathtt{bc},\mathtt{S},7\rangle\}$

Let $\mathfrak{A}$ be the sign grammar based on the following rules.

(5.193)
$$\mathtt{S} \to \mathtt{AC} \mid \mathtt{BC} \qquad \mathtt{A} \to \mathtt{a}$$
$$\mathtt{B} \to \mathtt{b} \qquad \mathtt{C} \to \mathtt{c}$$

This corresponds to choosing five modes, $\mathtt{F}_0$, $\mathtt{F}_1$, $\mathtt{F}_2$ all unary, and $\mathtt{F}_3$, $\mathtt{F}_4$ both binary.

(5.194)
$$\mathtt{F}_0 = \langle \mathtt{a},\mathtt{A},0\rangle$$
$$\mathtt{F}_1 = \langle \mathtt{b},\mathtt{B},0\rangle$$
$$\mathtt{F}_2 = \langle \mathtt{c},\mathtt{C},3\rangle$$

Further, $\mathtt{F}_3^\gamma$ is a two place function defined only on $\langle \mathtt{A},\mathtt{C}\rangle$ with result $\mathtt{S}$, $\mathtt{F}_3^\mu$ a two place function defined only on $\langle 0,3\rangle$ with value 5. Similarly, $\mathtt{F}_4^\gamma(\mathtt{A},\mathtt{C}) = \mathtt{S}$, and is undefined elsewhere, and $\mathtt{F}_4^\mu(0,3) = 7$, and is undefined elsewhere. Then the only definite structure terms are $\mathtt{F}_0$, $\mathtt{F}_1$, $\mathtt{F}_2$, $\mathtt{F}_3\mathtt{F}_0\mathtt{F}_2$, and $\mathtt{F}_4\mathtt{F}_1\mathtt{F}_2$. Together they unfold to exactly Σ.

This grammar is, however, not natural. We have

(5.195) $\mathrm{Cont}_{\mathfrak{A},\mathtt{S}}(\mathtt{a}) = \mathrm{Cont}_{\mathfrak{A},\mathtt{S}}(\mathtt{b}) = \{\langle \varepsilon, \mathtt{c}\rangle\}$

However, $\mathtt{a}$ and $\mathtt{b}$ do not have the same category.

Now look at the grammar $\mathfrak{B}$ based on the following rules:

(5.196)
$$\mathtt{S} \to \mathtt{ac} \mid \mathtt{BC} \qquad \mathtt{A} \to \mathtt{a}$$
$$\mathtt{B} \to \mathtt{b} \qquad \mathtt{C} \to \mathtt{c}$$

Here we compute that

(5.197) $\mathrm{Cont}_{\mathfrak{B},\mathtt{S}}(\mathtt{a}) = \varnothing \neq \{\langle \varepsilon, \mathtt{c}\rangle\} = \mathrm{Cont}_{\mathfrak{B},\mathtt{S}}(\mathtt{b})$

Notice that in this grammar, $\mathtt{a}$ has no constituent occurrence in a sentence. Only $\mathtt{b}$ has. So, $\mathtt{ac}$ is treated as an idiom.

Definition 5.68 *A vectorial system of signs is* ***strictly compositional*** *if there is a natural sign grammar for* Σ *in which for every* $f \in F$ *the function* f^3 *is a vector term which is stricly progressive with respect to the combined lengths of the strings.*

The definition of compositionality is approximately the one that is used in the literature (modulo adaptation to systems of signs) while the notion of strict compositionality is the one which we think is the genuine notion reflecting the intuitions concerning compositionality.

A particularly well–known case of noncompositionality in the strict sense is the analysis of quantification by Montague.

(5.198) `Nobody has seen Paul.`

(5.199) `No man has seen Paul.`

In the traditional, pre–Fregean understanding the subject of this sentence is `nobody` and the remainder of the sentence is the predicate; further, the predicate is predicated of the subject. Hence, it is said of nobody that he has seen Paul. Now, who is this 'nobody'? Russell, following Frege's analysis has claimed that the syntactic structure is deceptive: the subject of this sentence is contrary to all expectations *not* the argument of its predicate. Many, including Montague, have endorsed that view. For them, the subject denotes a so–called *generalized quantifier*. Type theoretically the generalized quantifier of a subject has the type $e \to (e \to t)$. This is a set of properties, in this case the set of properties that are disjoint to the set of all humans. Now, `has seen Paul` denotes a property, and (5.199) is true iff this property is in the set denoted by `no human`, that is to say, if it is disjoint with the set of all humans.

The development initiated by Montague has given rise to a rich literature. Generalised quantifiers have been a big issue for semantics for quite some time (see [**?**]). Similarly for the treatment of intensionality that he proposed. Montague systematically assigned intensional types as meanings, which allowed to treat world or situation dependencies. The general ideas were laid out in the semiotic program and left room for numerous alternatives. This is what we shall discuss here. However, first we scrutinize Montague's analysis of quantifiers. The problem that he chose to deal with was the ambiguity of sentences that were unambiguous with respect to the type assignment. In-

structive examples are the following.

(5.200) `Some man loves every woman.`

(5.201) `Jan is looking for a unicorn.`

Both sentences are ambiguous. (5.200) may say that there is a man such that he loves all women. Or it may say that for every woman there is a man who loves her. In the first reading the universal quantifier is in the scope of the existential, in the second reading the existential quantifier is in the scope of the universal quantifier. Likewise, (5.201) may mean two things. That there is a real unicorn and Jan is looking for it, or Jan is looking for something that is in his opinion a unicorn. Here we are dealing with scope relations between an existential quantifier and a modal operator. We shall concentrate on example (5.200). The problem with this sentence is that the universal quantifier `every woman` may not take scope over `some man loves`. This is so since the latter does not form a constituent. (Of course, we may allow it to be a constituent, but then this option creates problems of its own. In particular, this does not fit in with the tight connections between the categories and the typing regime.) So, if we insist on our analysis there is only one reading: the universal quantifier is in the scope of the existential. Montague solved the problem by making natural language look more like predicate logic. He assumed an infinite set of pronouns called $\mathtt{he}_n$. These pronouns exist in inflected forms and in other genders as well, so we also have $\mathtt{him}_n$, $\mathtt{her}_n$ and so on. (We shall ignore gender and case inflection at this point.) The other reading is created as follows. We feed to the verb the pronoun $\mathtt{he}_0$ and get the constituent $\mathtt{loves\ he}_0$. This is an intransitive verb. Then we feed another pronoun, say $\mathtt{he}_1$ and get $\mathtt{he}_1\ \mathtt{loves\ he}_0$. Next we combine this with `every man` and then with `a woman`. These operations substitute genuine phrases for these pronouns in the following way. Assume that we have two signs:

(5.202) $\sigma = \langle \vec{x}, e\backslash t, m\rangle, \quad \sigma' = \langle \vec{y}, t, m'\rangle$

Further, let Q_n be the following function on signs:

(5.203) $Q_n(\sigma, \sigma') := \langle \mathrm{sub}_n(\vec{x}, \vec{y}), t, Q(m, \lambda x_n.m')\rangle$

Here $\mathrm{sub}_n(\vec{x}, \vec{y})$ is defined as follows.

① For some k: $\vec{x} = \mathtt{he}_k$. Then $\mathrm{sub}_n(\vec{x}, \vec{y})$ is the result of replacing all occurrences of $\mathtt{he}_n$ by $\mathtt{he}_k$.

② For all k: $\vec{x} \neq \mathtt{he}_k$. Then $\mathrm{sub}_n(\vec{x},\vec{y})$ is the result of replacing the first occurrence of $\mathtt{he}_n$ by $\vec{x}$ and deleting the index n on all other occurrences of $\mathtt{he}_n$.

At last we have to give the signs for the pronouns. These are

(5.204) $\mathtt{P}_n := \langle \mathtt{he}_n, \mathtt{NP}, \lambda x_{\mathcal{P}}.x_{\mathcal{P}}(x_n)\rangle$

Depending on case, $x_{\mathcal{P}}$ is a variable of type $e \to t$ (for nominative pronouns) or of type $e \to (e \to t)$ (for accusative pronouns). Starting with the sign

(5.205) $\mathtt{L} := \langle \mathtt{loves}, (e\backslash t)/e, \lambda x_0.\lambda x_1.\mathsf{love}'(x_1,x_0)\rangle$

we get the sign

(5.206) $\mathtt{A}_{>}\mathtt{P}_1\mathtt{A}_{>}\mathtt{P}_0\mathtt{L}$
$= \langle \mathtt{he}_1\ \mathtt{loves}\ \mathtt{he}_0, t, \mathsf{loves}'(x_1,x_0))\rangle$

Now we need the following additional signs.

(5.207) $\mathtt{E}_n := \langle \lambda x.\lambda y.\,\mathrm{sub}_n(\mathtt{every} \diamond x, y), (t/t)/(e\backslash t),$
$\lambda x.\lambda y.\forall x_n.(x(x_n) \to y)\rangle$

(5.208) $\mathtt{S}_n := \langle \lambda x.\lambda y.\,\mathrm{sub}_n(\mathtt{some} \diamond x, y), (t/t)/(e\backslash t),$
$\lambda x.\lambda y.\exists x_n.(x(x_n) \wedge y)\rangle$

(5.209) $\mathtt{M} := \langle \mathtt{man}, e\backslash t, \lambda x.\mathsf{man}'(x)\rangle$

(5.210) $\mathtt{W} := \langle \mathtt{W}, e\backslash t, \lambda x.\mathsf{woman}'(x)\rangle$

If we feed the existential quantifier first we get the reading $\forall\exists$, and if we feed the universal quantifier first we get the reading $\exists\forall$. The structure terms are as follows.

(5.211) $\mathtt{A}_{>}\mathtt{A}_{>}\mathtt{E}_0\mathtt{W}\mathtt{A}_{>}\mathtt{A}_{>}\mathtt{S}_1\mathtt{M}\mathtt{A}_{>}\mathtt{P}_0\mathtt{A}_{>}\mathtt{P}_1\mathtt{L}$

(5.212) $\mathtt{A}_{>}\mathtt{A}_{>}\mathtt{S}_1\mathtt{M}\mathtt{A}_{>}\mathtt{A}_{>}\mathtt{E}_0\mathtt{W}\mathtt{A}_{>}\mathtt{P}_0\mathtt{A}_{>}\mathtt{P}_1\mathtt{L}$

We have not looked at the morphological realization of the phrase $\vec{x}$. Number and gender must be inserted with the substitution. So, the case is determined by the local context, the other features are not. We shall not go into this here. (Montague had nothing to say about morphology as English has very little. We can only speculate what would have been the case if Montague had

spoken, say, an inflecting language.) Notice that the present analysis makes quantifiers into sentence adjuncts.

Recall from Section 2.7 that the grammar of sentences is very complex. Hence, since Montague defined the meaning of sentences to be closed formulae, it is almost unavoidable that something had to be sacrificed. In fact, the given analysis violates several of our basic principles. First, there are infinitely many lexical elements. Second, the syntactic structure is not respected by the translation algorithm, and this yields the wrong results. Rather than taking an example from a different language, we shall exemplify the problems with the genitive pronouns. We consider `his` as the surface realization of `him's`, where `'s` is the so–called Anglo Saxon genitive. Look for example at (5.213), which resulted from (5.214). Application of the above rules gives (5.215), however.

(5.213) `with a hat on his head every man looks better`

(5.214) sub_0(`every man`, `with a hat on` him_0`'s head` he_0 `looks better`)

(5.215) `with a hat on every man's head he looks better`

This happens not because the possessive pronouns are also part of the rules: of course, they have to be part of the semantic algorithm. It is because the wrong occurrence of the pronoun is being replaced by the quantifier phrase. This is due to the fact that the algorithm is ignorant about the syntactic structure (which the string reveals only partly) and second because the algorithm is order sensitive at places where it should better not be. See Section 6.5 on GB, a theory that has concerned itself extensively with the question which NP may be a pronoun, or a reflexive pronoun or empty. Fiengo and May [**?**] speak quite plastically of *vehicle change*, to name the phenomenon that a variable appears sometimes as a pronoun, sometimes as pro (an empty pronoun, see Section 6.5), sometimes as a lexical NP and so on. The synchronicity between surface structure and derivational history which has been required in the subsequent categorial grammar, is not found with Montague. He uses instead a distinction proposed by Church between **tectogrammar** (the inner structure, as von Humboldt would have called it) and **phenogrammar** (the outer structure, which is simply what we see). Montague admits quite powerful phenogrammatical operations, and it seems as if only the label distinguishes him from GB theory. For in principle his maps could be interpreted as transformations.

We shall briefly discuss the problems of blocking and other apparent failures of compositionality. In principle, we have allowed the exponent functions to be partial. They can refuse to operate on certain items. This may be used in the analysis of defective words, for example `courage`. This word exists only in the singular (though it arguably also has a plural meaning). There is no form `courages`. In morphology, one says that each word has a root; in this case the root may simply be `courage`. The singular is formed by adding ε, the plural by adding `s`. The word `courage` does not let the plural be formed. It is defective. If that is so, we are in trouble with Leibniz' Principle. Suppose we have a word X that is synonymous with `courage` but exists in the singular and the plural (or only in the plural like `guts`). Then, by Leibniz' Principle, the two roots can never have the same meaning, since it is not possible to exchange them for each other in all contexts (the context where X appears in the plural is a case in point). To avoid this, we must actually assume that there is no root form of `courage`. The classical grammar calls it a **singulare tantum**, a 'singular only'. This is actually more appropriate. If namely this word has no root and exists only as a singular form, one simply cannot exchange the root by another. We remark here that English has **pluralia tanta** ('plural only' nouns), for example `troops`. In Latin, `tenebrae` 'darkness', `indutiae` 'cease fire' are examples. Additionally, there are words which are only formwise derived from the singular counterpart (or the root, for that matter). One such example is `forces` in its meaning 'troops', in Latin `fortunae` 'assets', whose singular `fortuna` means 'luck'. Again, if both forms are assumed to be derived from the root, we have problems with the meaning of the plural. Hence, some of these forms (typically — but not always — the plural form) will have to be part of the lexicon (that is, it constitutes a 0–ary mode).

Once we have restricted the admissible functions on exponents, we can show that weak and strong generative capacity do not necessarily coincide. Recall the facts from Exercise 187, taken from [**?**]. In Mandarin yes–no–questions are formed by iterating the statement with the negation word in between. Although it is conceivable that Mandarin is context free as a string language, Radzinski argues that it is not strongly context free. Now, suppose we understand by strongly context free that there is a context free sign grammar. Then we shall show that under mild conditions Mandarin is not strongly context free. To simplify the dicussion, we shall define a somewhat artificial counterpart of Mandarin. Start with a context free language G and a meaning

function μ defined on G. Then put $M := G \cup G \diamond \mathtt{bu} \diamond G$. Further, put

$$\nu(\vec{x} \diamond \mathtt{bu} \diamond \vec{y}) := \begin{cases} \mu(\vec{x}) \vee \neg\mu(\vec{y}) & \text{if } \vec{x} \neq \vec{y}, \\ \mu(\vec{x})? & \text{if } \vec{x} = \vec{y}. \end{cases} \tag{5.216}$$

Here, ? forms questions. We only need to assume that it is injective on the set $\mu[G]$ and that $?[\mu[G]]$ is disjoint from $\{\mu(\vec{x}) \vee \mu(\vec{y}) : \vec{x}, \vec{y} \in L(G)\}$. (This is the case in Mandarin.) Assume that there are two distinct expressions $\vec{u}$ and $\vec{v}$ of equal category in G such that $\mu(\vec{u}) = \mu(\vec{v})$. Then they can be substituted for each other. Now suppose that G has a sublanguage of the form $\{\vec{r}\vec{z}^{\,i}\vec{s} : i \in \omega\}$ such that $\mu(\vec{r}\vec{z}^{\,i}\vec{s}) = \mu(\vec{r}\vec{z}^{\,j}\vec{s})$ for all i, j. We claim that M together with ν is not context free. Suppose otherwise. Then we have a context free grammar H together with a meaning function that generates it. By the Pumping Lemma, there is a k such that $\vec{z}^{\,k}$ can be adjoined into some $\vec{r}\vec{z}^{\,i}\vec{s}$ any number of times. (This is left as an exercise.) Now look at the expressions

$$\vec{r}^{\frown}\vec{z}^{\,i}{}^{\frown}\vec{s} \diamond \mathtt{bu} \diamond \vec{r}^{\frown}\vec{z}^{\,j}{}^{\frown}\vec{s} \tag{5.217}$$

Adjunction is the result of substitution. However, the ν–meaning of these expressions is $\top$ if $i \neq j$ and a yes–no question if $i = j$. Now put $j = i + k$. If we adjoin $\vec{z}^{\,k}$ on the left side, we get a yes–no question, if we substitute it to the right, we do not change the meaning, so we do not get a yes–no question. It follows that one and the same syntactic substitution operation defines two different semantic functions, depending on where it is performed. Contradiction. Hence this language is not strongly context free. It is likely that Mandarin satisfies the additional assumptions. For example, colour words are extensional. So, `blue shirt` means the same as `blue blue shirt`, `blue blue blue shirt`, and so on.

Next we look at Bahasa Indonesia. Recall that it forms the plural by reduplication. If the lexicon is finite, we can still generate the set of plural expressions. However, we must assume a distinct syntactic category for each noun. This is clearly unsatisfactory. For every time the lexicon grows by another noun, we must add a few rules to the grammar (see [**?**]). However, let us grant this point. Suppose, we have two nouns, $\vec{m}$ and $\vec{n}$, which have identical meaning. If there is no syntactic or mophological blocking, by Leibniz' Principle any constituent occurrence of the first can be substituted by the second and vice versa. Therefore, if $\vec{m}$ has two constituent occurrences in $\vec{m}$-$\vec{m}$, we must have a word $\vec{m}$-$\vec{n}$ and a word $\vec{n}$-$\vec{m}$, and both mean the same as the first. This is precisely what is not the case. Hence, no such pair of words can exist if

Bahasa Indonesia is strongly context free. This argument relies on a stronger version of Leibniz' Principle: that semantic identity enforces substitutability *tout court*. Notice that our previous discussion of context sets does not help here. The noun $\vec{m}$ has a different context set as the noun $\vec{n}$, since it occurs in a plural noun $\vec{m}$–$\vec{m}$, where $\vec{n}$ does not occur. However, notice that the context set of $\vec{m}$ contains occurrences of $\vec{m}$ itself. If that circularity is removed, $\vec{m}$ and $\vec{n}$ become indistinguishable.

These example might suffice to demonstrate that the relationship between syntactic structure and semantics is loose but not entirely free. One should be extremely careful, though, of hidden assumptions. Many arguments in the literature showing that this or that language is not strongly context free rest on particular assumptions that are not made explicit.

Notes on this section. The idea that syntactic operations should more or less be restricted to concatenation give or take some minor manipulations is advocated for in [**?**], who calls this *surface compositionality*. Hausser also noted that Montague did not actually define a surface compositional grammar. Most present day categorial grammars are, however, surface compositional.

Exercise 203. Suppose that $A = \{\mathtt{0}, \mathtt{1}, \ldots, \mathtt{9}\}$, with the following modes.

$$\mathtt{0} := \langle \mathtt{0}, \mathtt{Z}, 0 \rangle \tag{5.218}$$

$$\mathtt{S}(\langle \vec{x}, \mathtt{Z}, n \rangle) := \langle \mathrm{suc}(\vec{x}), \mathtt{Z}, n+1 \rangle \tag{5.219}$$

Here, $\mathrm{suc}(\vec{x})$ denotes the successor of $\vec{x}$ in the decimal notation, for example, $\mathrm{suc}(\mathtt{19}) = \mathtt{20}$. Let a string be given. What does a derivation of that string look like? When does a sign σ occur in another sign τ? Describe the exponent of $[\mathfrak{s}'/\mathfrak{s}]\mathfrak{t}$, for given structure terms $\mathfrak{s}$, $\mathfrak{s}'$, $\mathfrak{t}$. Define a progress measure for which this grammar is progressive.

Exercise 204. Let $A := \{\mathtt{0}, \ldots, \mathtt{9}, \mathtt{+}, \mathtt{(}, \mathtt{)}, \mathtt{*}\}$. We shall present two ways for generating ordinary arithmetical terms. Recall that there is a convention to drop brackets in the following circumstances. (a) When the same operation symbol is used in succession (`5+7+4` in place of `(5+(7+4))`), (b) when the enclosed term is multiplicative (`3*4+3` in place of `(3*4)+3`). Moreover, (c) the outermost brackets are dropped. Write a sign grammar that generates triples $\langle \vec{x}, T, n \rangle$, where $\vec{x}$ is a term and n its value, where the conventions (a), (b) and (c) are optionally used. (So you should generate $\langle \mathtt{5+7+4}, T, 16 \rangle$ as well as $\langle \mathtt{(5+(7+4))}, T, 16 \rangle$). Now apply Leibniz' Principle to the pairs `(5+7)` and `5+7`, `5+(7+4)` and `5+7+4`. What problems arise? Can you suggest a solution?

`aio`	`inquam`	`dico`	'I say'
`ais`	`inquis`	`dicis`	'you(sg) say'
`ait`	`inquit`	`dicit`	'he says'
		`dicimus`	'we say'
		`dicitis`	'you(pl) say'
`aiunt`	`inquiunt`	`dicunt`	'they say'

Figure 16. Latin Verbs of Saying

Exercise 205. (Continuing the previous exercise.) Write a grammar that treats every accidental occurrence of a term as a constituent occurrence in some different parse. For example, the occurrence of `3+4` in `3+4*7` is in the grammar of the previous exercise a nonconstituent occurrence, now however it shall be a constituent occurrence under some parse. Apply Leibniz' Principle. Show that `5+7` is not identical to `7+5`, and `2+5+7` is not identical to `2+7+5` and so on. Which additive terms without brackets are identical in meaning, by Leibniz Principle?

Exercise 206. The Latin verbs `aio` and `inquam` ('I say') are highly defective. They exist only in the present. Apart from one or two more forms (which we shall ignore for simplicity), Figure 16 gives a synopsis of what forms exist of these verbs and contrast them with the forms of `dico`. The morphology of `inquam` is irregular in that form (we expect `inquo`); also the syntax of `inquit` is somewhat peculiar (it is used parenthetically). Discuss whether `inquit` and `dicit` can be identical in meaning by Leibniz' Principle or not. Further, the verb `memini` is formwise in the perfect, but it means 'I remember'; similarly `odi` 'I hate'.

8. de Saussure Grammars

In his famous Cours de Linguistique Générale, de Saussure speaks about linguistic signs and the nature of language as a system of signs. In his view, a sign is constituted by two elements: its **signifier** and its **signified**. In our terms, these are the exponent and the meaning, respectively. Moreover, de Saussure says that signifiers are *linear*, without further specifying what he means by that. To a modern linguist all this seems obviously false: there are categories, and linguistic objects are structured, they are not linear. Notably

Chomsky has repeatedly offered arguments to support this view. He believed that structuralism was fundamentally mistaken. In this section we shall show that the rejection of de Saussure's ideas is ill–founded. To make the point, we shall look at a few recalcitrant syntactic phenomena and show how they can be dealt with using totally string based notions.

Let us return to the idea mentioned earlier, that of λ–terms on strings. We call a **string term** a λ–term over the algebra of strings (consisting of constants for every $a \in A$, ε, and $^\frown$). We assume here that strings are typed, and that we have strings of different type. Assume for the moment that there is only one type, that of a string, denoted by s. Then $\lambda x.\lambda y.y^\frown x$ is the function of reverse concatenation, and it is of type $s \to (s \to s)$. Now we wish to implement restrictions on these terms that make sure we do not lose any material. Call a λ–term **relevant** if for all subterms $\lambda x.N$, x occurs at least once free in N. $\lambda x.\lambda y.y \diamond x$ is relevant, $\lambda x.\lambda y.x$ is not. Clearly, relevance is a necessary restriction. However, it is not sufficient. Let $\mathcal{P}$ and $\mathcal{Q}$ be variables of type $s \to s$, x a variable of type x. Then function composition, $\lambda\mathcal{P}.\lambda\mathcal{Q}.\lambda x.\mathcal{P}(\mathcal{Q}(x))$, is a relevant λ–term. But this is problematic. Applying this term leaves no visible trace on the string, it just changes the analysis. Thus, we shall also exclude *combinators*. This means, an admissible λ–term is a relevant term that contains $^\frown$ or an occurrence of a constant at least once.

Definition 5.69 *A string term τ is **weakly progressive** if it is relevant and not a combinator. τ is **progressive** if it is weakly progressive and does not contain ε.*

Definition 5.70 *A **de Saussure sign** or simply **dS–sign** is a pair $\delta = \langle e, m\rangle$, where e is a progressive string term and m a λ–term over meanings. The **type** of δ is the pair $\langle \sigma, \tau\rangle$, where σ is the type of e and τ the type of m. If $\delta' = \langle e', m'\rangle$ is another de Saussure sign then $\delta(\delta')$ is defined iff ee' is defined and mm' is defined, and then*

$$\delta(\delta') := \langle ee', mm'\rangle \tag{5.220}$$

*In this situation we call δ the **functor sign** and δ' the **argument sign**. A **de Saussure grammar** is a finite set of dS–signs.*

So, the typing regime of the strings and the typing regime of the meanings do all the work here.

Proposition 5.71 *Let δ and δ' be dS–signs of type $\langle\sigma,\tau\rangle$ and $\langle\sigma',\tau'\rangle$, respectively. Then $\delta(\delta')$ is defined iff there are μ, ν such that $\sigma = \sigma' \to \mu$, $\tau = \tau' \to \nu$, and then $\delta(\delta')$ has type $\langle\mu,\nu\rangle$.*

The rest is actually the same as in AB–grammars. Before we shall prove any results, we shall comment on the definition itself. In Montague Grammar and much of Categorial Grammar there is a conflation of information that belongs to the realm of meaning and information that belongs to the realm of exponents. The category β/α, for example, tells us that the meaning must be a function of type $\sigma(\alpha) \to \sigma(\beta)$, and that the exponent giving us the argument must be found to the right. $\alpha\backslash\beta$, is different only in that the exponent is to be found to the left. While this seems to be reasonable at first sight, it is already apparent that the syntactic categories simply elaborate the semantic types. (This is why σ is a homomorphism.) The information concerning the semantic types is however not necessary, since the merger would fail anyhow if we did not supply signs with the correct types. So, we could leave it to syntax to specify only the directionality. However, syntax is not well equipped for that. There are discontinuous constituents and this is not easily accommodated in categorial grammar. Much of the research can be seen as an attempt to upgrade the string handling potential in this direction. Notice further that the original categorial apparatus created distinctions that are nowhere attested. For example, adjectives in English are of category n/n. In order to modify a relational noun, however, they must be lifted to the category of a relational noun. The lifting will have to specify whether the noun is looking for its complement on its right or on its left. Generally, however, modifiers and functors do not care very much about the makeup of their arguments. However, in AB and L, categories must be explicit about these details.

De Saussure grammars do away with some of the problems that beset CGs. They do not require to iterate the semantic types in the category, and the string handling has more power than in standard categorial grammar. We shall discuss a few applications of de Saussure grammars. These will illustrate both the strength as well as certain deficiencies.

A striking fact about de Saussure grammars is that they allow for word order variation in the most direct way. Let us take a transitive verb, see, with meaning $\mathsf{see}' = \lambda x.\lambda y.\mathsf{see}'(y)(x)$. Its first argument is the direct object and the second its subject. We assume no case marking, so that the following nouns will be either subject or object.

(5.221) JOHN $:= \langle \mathtt{John}, \mathsf{john}' \rangle$ MARY $:= \langle \mathtt{Mary}, \mathsf{mary}' \rangle$

Now we can give to the verb one of the following six signs. of which each corresponds to a different word order pattern. Recall that $x \diamond y = x^\frown \Box^\frown y$.

(5.222) $\text{SEES}_0 := \langle \lambda x.\lambda y.y \diamond x \diamond \texttt{sees}, \mathsf{see}' \rangle$ SOV

(5.223) $\text{SEES}_1 := \langle \lambda x.\lambda y.y \diamond \texttt{sees} \diamond x, \mathsf{see}' \rangle$ SVO

(5.224) $\text{SEES}_2 := \langle \lambda x.\lambda y.\texttt{sees} \diamond y \diamond x, \mathsf{see}' \rangle$ VSO

(5.225) $\text{SEES}_3 := \langle \lambda x.\lambda y.x \diamond y \diamond \texttt{sees}, \mathsf{see}' \rangle$ OSV

(5.226) $\text{SEES}_4 := \langle \lambda x.\lambda y.x \diamond \texttt{sees} \diamond y, \mathsf{see}' \rangle$ OVS

(5.227) $\text{SEES}_5 := \langle \lambda x.\lambda y.\texttt{sees} \diamond x \diamond y, \mathsf{see}' \rangle$ VOS

The structure term for a basic sentence expressing that John sees Mary is in all cases the same. (Structure terms will be written using brackets, to avoid confusion. The convention is that bracketing is left–associative.) It is $\text{SEES}_i(\text{MARY})(\text{JOHN})$, $i < 6$. Only that the order of the words is different in each case. For example,

(5.228)
$$\begin{aligned}
&\text{SEES}_0(\text{MARY})(\text{JOHN}) \\
=&\langle \lambda x.\lambda y.y \diamond x \diamond \texttt{sees}, \mathsf{see}' \rangle (\langle \texttt{Mary}, \mathsf{mary}' \rangle)(\langle \texttt{John}, \mathsf{john}' \rangle) \\
=&\langle \lambda y.y \diamond \texttt{Mary sees}, \mathsf{see}'(\mathsf{mary}') \rangle (\langle \texttt{John}, \mathsf{john}' \rangle) \\
=&\langle \texttt{John Mary sees}, \mathsf{see}'(\mathsf{mary}')(\mathsf{john}') \rangle
\end{aligned}$$

(5.229)
$$\begin{aligned}
&\text{SEES}_4(\text{MARY})(\text{JOHN}) \\
=&\langle \lambda x.\lambda y.x \diamond \texttt{sees} \diamond y, \mathsf{see}' \rangle (\langle \texttt{Mary}, \mathsf{mary}' \rangle)(\langle \texttt{John}, \mathsf{john}' \rangle) \\
=&\langle \lambda y.\texttt{Mary sees} \diamond y, \mathsf{see}'(\mathsf{mary}') \rangle (\langle \texttt{John}, \mathsf{john}' \rangle) \\
=&\langle \texttt{Mary sees John}, \mathsf{see}'(\mathsf{mary}')(\mathsf{john}') \rangle
\end{aligned}$$

Notice that this construction can be applied to heads in general, and to heads with any number of arguments. Thus, de Saussure grammars are more at ease with word order variation than categorial grammars. Moreover, in the case of OSV word order we see that the dependencies are actually crossing, since the verb does not form a constituent together with its subject.

We have seen in Section 5.3 how interpreted LMGs can be transformed into AB–grammars using vector polynomials. Evidently, if we avail ourselves of vector polynomials (for example by introducing pair formation and projections and redefining the notion of progressivity accordingly) this result can be reproduced here for de Saussure grammars. Thus, de Saussure grammars suitably generalized are as strong as interpreted LMGs. However, we shall actually not follow this path. We shall not use pair formation; instead, we shall

Table 16. Plural in English

Singular	Plural	
`tree`	`trees`	plain suffix
`bush`	`bushes`	e-insertion
`ox`	`oxen`	en–suffix
`fish`	`fish`	no change
`man`	`men`	vowel change

stay with the more basic apparatus. The examples that we shall provide below will give evidence that this much power is actually sufficient for natural languages, though some modifications will have to be made.

Next we shall look at plural in Bahasa Indonesia (or Malay). The plural is formed by reduplicating the noun. For example, the plural of `orang` 'man' is `orang-orang`, the plural of `anak` 'child' is `anak-anak`. To model this, we assume one type of strings, n.

(5.230) $\quad \text{PLU} := \langle \lambda x.x^\frown\texttt{-}^\frown x, \lambda \mathcal{P}.\{x : \mathcal{P}(x)\}\rangle$

The term $\lambda x.x^\frown\texttt{-}^\frown x$ is progressive. The plural operation can in principle be iterated; we shall see below how this can be handled. (We see no obvious semantical reason why it cannot, so it must be blocked morphologically.) Now let us turn to English. In English, the plural is formed by adding an `s`. However, some morphophonological processes apply, and some nouns form their plural irregularly. Table 16 gives an (incomplete) list of plural formation. Above the line we find regular plurals, below irregular plurals. As we have outlined in Section 1.3, these differences are explained by postulating different plural morphs, one for each noun class. We can account for that by introducing noun class distinctions in the semantic types. For example, we may introduce a semantic type for nouns endings in a nonsibilant, another for nouns ending in a sibilant, and so on. However, apart from introducing the distinction where it obviously does not belong, this proposal has another drawback. Recall, namely, that linguists speak of a plural *morpheme*, which abstracts away from the particular realizations of plural formation. Mel'čuk defines a morpheme as a set of signs that have identical category and identical meaning. So, for him the plural morpheme is simply the set of plural morphs. Now, suppose that we want the morpheme to be a (de Saussure) *sign*. Then

its meaning is that of any of its morphs, but the string function cannot be a λ–term. For it may act differently on identical strings of different noun class. A good example is German `Bank`. Like its English counterpart it can denote (i) a money institute, (ii) something to sit on, (iii) the bank of a river. However, in the first case its plural is `Banken` and in the other two it is `Bänke`. Now, since the function forming the plural cannot access the meaning we must distinguish two different string classes, one for nouns that form the plural by umlaut plus added `e`, and the other for nouns that form the plural by adding `en`. Further, we shall assume that German `Bank` is in both, but with different meanings. Thus, we have two signs with exponent `Bank`, one to mean money institute and the other to mean something to sit on or the bank of a river. This is the common practice. The classes are morphological, that is, they do not pertain to meaning, just to form.

Thus we are led to the introduction of string types. We assume that types are ordered by some partial ordering $\leq$, so that if α and β are string types and $\alpha \leq \beta$ then any string of type α is a string of type β. Moreover, we put $\alpha \to \beta \leq \alpha' \to \beta'$ iff $\alpha \leq \alpha'$ and $\beta \leq \beta'$. No other relations hold between nonbasic types. The basic type s is the largest basic type. Returning now to English, we shall split the type n into various subtypes. In particular, we need the types ni, nr, of irregular and regular nouns. We shall first treat the regular nouns. The rule is that if a noun ends in a sibilant, the vowel `e` is inserted, otherwise not. Since this is a completely regular phenomenon, we can only define the string function if we have a predicate sib that is true of a string iff it ends in a sibilant. Further, we need to be able to define a function by cases.

(5.231) $\quad \mathsf{rplu} := \lambda x.\mathsf{if}\ \mathsf{sib}(x)\ \mathsf{then}\ x^\frown\mathtt{es}\ \mathsf{else}\ x^\frown\mathtt{s}\ \mathsf{fi};$

Thus, we must have a basic type of booleans plus some functions. We shall not spell out the details here. Notice that definitions by cases are not necessarily unique, so they have to be used with care. Notice a further problem. The minute that we admit different types we have to be specific about the type of the resulting string. This is not an innocent matter. The operation $^\frown$ is defined on all strings. Suppose now that `tree` is a string of type nr, which type does `trees` have? Obviously, we do not want it to be just a string, and we may not want it to be of type nr again. (The difference between regular and irregular is needed only for plural formation.) Also, as we shall see below, there are operations that simply change the type of a string without changing the string itself. Hence we shall move from a system of implicit typing to one of explicit typing (see [**?**] for an overview). Rather than using variables for each type,

we use a single set of variables. λ–abstraction is now written $\lambda x : \sigma.M : \tau$ in place of $\lambda x.M$. Here x must be a variable of type σ, and the result will be of type τ. Thus, $\lambda x : nr.x^\frown \mathtt{s} : n$ denotes the function that turns and nr–string into an n–string by appending $\mathtt{s}$. The reader may recall from Section 4.1 the idea that strings can be taken to mean different things depending on what type they are paired with. Internally, a typed string term is represented by $\langle N, \sigma \rangle$, where N is the string term and σ its type. The operation $M : \tau$ does the following: it evaluates M on N, and gives it the type τ. Now, the function is also defined for all $\sigma' \leq \sigma$, so we finally have

$$(5.232) \qquad (\lambda x : \sigma.M : \tau)(N : \sigma') = \begin{cases} [N/x]M : \tau & \text{if } \sigma' \leq \sigma, \\ * & \text{otherwise.} \end{cases}$$

Now we turn to the irregular plural. Here we face two choices. We may simply take all singular and plural nouns as being in the lexicon; or we devise rules for all occurring subcases. The first is not a good idea since it does not allow us to say that `ox` and the plural morpheme actually occur in `oxen`. The sign is namely an unanalyzable unit. So we discard the first alternative and turn to the second. In order to implement the plural we again need a predicate of strings that tells us whether a string equals some given string. The minimum we have to do is to introduce an equality predicate on strings. This allows to define the plural by cases. However, suppose we add a binary predicate $\mathsf{suf}(x, y)$ which is true of x and y iff x is a suffix of y. Then the regular plural can be defined also as follows:

$$(5.233) \qquad \mathsf{rplu} := \lambda x : nr.\ \mathsf{if}\ (\mathsf{suf}(\mathtt{s}, x)\ \mathsf{or}\ \mathsf{suf}(\mathtt{sh}, x))$$
$$\mathsf{then}\ x^\frown \mathtt{es}\ \mathsf{else}\ x^\frown \mathtt{s}\ \mathsf{fi} : n;$$

Moreover, equality is definable from suf. Evidently, since we allow a function to be defined by cases, the irregular plural forms can be incorporated here as well, as long as they are additive (as is `oxen` but not `men`). For nonadditive plural formations see the remarks on umlaut in Section 6.3.

Now take another case, causatives. Many English verbs have causative forms. Examples are `laugh`, `drink`, `clap`.

(5.234) `The audience laughed the conductor off the stage.`

(5.235) `The manager drank his friends under the table.`

(5.236) `The audience was clapping the musician back onto the stage.`

In all these cases the meaning of the causative is regularly formed so that we may actually assume that there is a sign that performs the change. But it leaves no visible trace. Thus, we must at least allow operators that perform type conversion even when they change nothing in the semantics. In the type system we have advocated above they can be succinctly represented by

(5.237) $\lambda x : \sigma . x : \tau$

Now, the conversion of a string of one type into another is often accompanied by morphological marking. For example, the gerund in English turns a verb into a noun (`singing`). It is formed regularly by suffixing `ing`. So, it has the following sign:

(5.238) $\text{GER} := \langle \lambda x : v . x^\frown \texttt{ing} : n, \lambda x . x \rangle$

The semantics of these nominalizations is rather complex (see [**?**]), so we have put the identity for simplicity here. Signs that consist of nothing more than a type conversion are called **conversionemes** in [**?**]. Obviously, they are not progressive in the intuitive sense. For we can in principle change a string from σ to τ and back; and we could do this as often as we like. However, there is little harm in admitting such signs. The additional complexity can be handled in much the same way as unproductive context free rules.

Another challenge is Swiss German. Since we do not want to make use of products, it is not obvious how we can instrumentalize the λ–terms to get the word order right. Here is how this can be done. We distinguish the main (inflected) verb from its subordinate verbs, and raising from nonraising verbs. (See Table 17. We use '±i' short for '±inflected', '±r' for '±raising', and '±t' for '±transitive'. We have suppressed the type information as it is of marginal relevance here.) Here, v, x, z are variables over NP–cluster strings, w, y variables over verb–cluster strings, and $\mathcal{P}$ a variable for functions from NP–cluster strings to functions from verb–cluster strings to strings. NPs are by contrast very basic:

(5.239) $\text{MER} := \langle \texttt{mer}, \mathsf{we}' \rangle$ $\qquad$ $\text{HUUS} := \langle \texttt{es huus}, \mathsf{house}' \rangle$

We ignore case for the moment. The lowest clause is translated as follows.

(5.240) $\text{AASTE}(\text{HUUS})$
$= \langle \lambda y . \lambda z . y \diamond \texttt{es huus} \diamond z \diamond \texttt{aastriche}, \mathsf{paint}'(\mathsf{house}') \rangle$

Table 17. Swiss German Verbs

–i–r+t	AASTE	$:= \langle \lambda x.\lambda y.\lambda z.y \diamond x \diamond z \diamond \texttt{aastriche}, \mathsf{paint}' \rangle$
+i–r+t	AAST	$:= \langle \lambda x.\lambda y.y \diamond x \diamond \texttt{aastricht}, \mathsf{paint}' \rangle$
–i–r–t	SCHWE	$:= \langle \lambda y.\lambda z.y \diamond z \diamond \texttt{schwimme}, \mathsf{sim}' \rangle$
+i–r–t	SCHW	$:= \langle \lambda x.x \diamond \texttt{schwimmt}, \mathsf{swim}' \rangle$
–i+r+t	LAA	$:= \langle \lambda x.\lambda \mathcal{P}.\lambda v.\lambda w.\mathcal{P}(v \diamond x)(w \diamond \texttt{laa}), \mathsf{let}' \rangle$
+i+r+t	LAAT	$:= \langle \lambda \mathcal{P}.\lambda x.\mathcal{P}(x)(\texttt{laa}), \mathsf{let}' \rangle$

The recursive part, raising verb plus object, is translated as follows:

$$
\begin{aligned}
&\textsc{hälfe}(\textsc{chind}) \\
=&\langle (\lambda x.\lambda \mathcal{P}.\lambda v.\lambda w.\mathcal{P}(v \diamond x)(w \diamond \texttt{hälfe}))(\texttt{em chind}), \\
&\quad \mathsf{help}'(\mathsf{children}') \rangle \\
=&\langle (\lambda \mathcal{P}.\lambda v.\lambda w.\mathcal{P}(v \diamond \texttt{em chind})(w \diamond \texttt{hälfe})), \\
&\quad \mathsf{help}'(\mathsf{children}') \rangle
\end{aligned}
\tag{5.241}
$$

If we combine the two we get something that is of the same kind as the lower infinitive, showing that the recursion is adequately captured:

$$
\begin{aligned}
&\textsc{hälfe}(\textsc{chind})(\textsc{aaste}(\textsc{huus})) \\
=&\langle \lambda v.\lambda w.v \diamond \texttt{em chind es huus} \diamond w \diamond \texttt{hälfe aastriche}, \\
&\quad \mathsf{help}'(\mathsf{children}')(\mathsf{paint}'(\mathsf{house}'))) \rangle
\end{aligned}
\tag{5.242}
$$

Had we inserted a finite verb, the second 'hole' would have been closed. There would have been just a place for the subject. Once that is inserted, there are no more holes left. The recursion is finished. Notice that the structure term has the form of the corresponding English structure. The λ–terms simply transliterate it into Swiss German. Let us briefly speak about case. We insert only the bare nouns and let the verb attach the appropriate case marker. For example, if DAT is the function that turns a DP into a dative marked DP, the sign HÄLFE will be

$$
\textsc{hälfe} := \langle \lambda x.\lambda \mathcal{P}.\lambda v.\lambda w.\mathcal{P}(v \diamond \textsc{dat}(x))(w \diamond \texttt{laa}), \mathsf{help}' \rangle \tag{5.243}
$$

Next, we shall deal with case agreement inside a noun phrase. In many languages, adjectives agree in case with the noun they modify. We take our

example from Finnish. The phrase `iso juna` 'a/the big train' inflects in the singular as follows. (We show only a fraction of the case system.)

(5.244)

nominative	`iso juna`
genitive	`ison junan`
allative	`isolle junalle`
inessive	`isossa junassa`

In the present case, it is the same suffix that is added to the adjective as well as the noun. Now, suppose we analyze the allative as a suffix that turns a caseless noun phrase into a case marked noun phrase. Then we want to avoid analyzing the allative `isolle junalle` as consisting of occurrences of the allative case. We want to say that it occurs once, but is spelled out twice. To achieve this, we introduce two types: ν, the type of case marked nouns, and κ, the type of case markers. Noun roots will be of type $\kappa \to \nu$, adjectives of type $(\kappa \to \nu) \to (\kappa \to \nu)$.

(5.245) $\textsc{juna} := \langle \lambda x : \kappa.\texttt{juna}^\frown x : \nu, \mathsf{train}' \rangle$

(5.246) $\textsc{iso} := \langle \lambda \mathcal{P} : \kappa \to \nu.\lambda x : \kappa.\texttt{iso}^\frown x \diamond \mathcal{P}(x) : \nu, \mathsf{big}' \rangle$

So, x has the type κ, $\mathcal{P}$ the type $\kappa \to \nu$. These signs combine to

(5.247) $\textsc{iso}(\textsc{juna}) = \langle \lambda x : \kappa.\texttt{iso}^\frown x \diamond \texttt{juna}^\frown x : \nu, \mathsf{big}'(\mathsf{train}') \rangle$

Finally, assume the following sign for the allative.

(5.248) $\textsc{all} := \langle \texttt{lle} : \kappa, \mathsf{move\text{-}to}' \rangle$

Then the last two signs combine to

(5.249) $\textsc{all}(\textsc{iso}(\textsc{juna}))$
$= \langle \texttt{isolle junalle} : \nu, \mathsf{move\text{-}to}'(\mathsf{big}'(\mathsf{train}')) \rangle$

This has the advantage that the tectogrammatical structure of signs is much like their semantic structure, and that we can stack as many adjectives as we like: the case ending will automatically be distributed to all constituents. Notice that LMGs put a limit on the number of occurrences that can be controlled at the same time, and so they cannot provide the same analysis for agreeing adjectives. Thus, de Saussure grammars sometimes provide more adequate analyses than do LMGs. We remark here that the present analysis conforms

to the idea proposed in [**?**], who considers agreement simply as a multiple manifestation of a single morpheme. Case assignment can also be handled in a rather direct way. Standardly, a verb that takes a case marked noun phrase is assumed to select the noun phrase as a noun phrase of that case. Instead, however, we may assume that the sign for a case marking verb actually carries the case marker and attaches it to the NP. The Finnish verb `tuntua` 'to resemble' selects ablative case. Assume that it has an ablative marked argument that it takes directly to its right. Then its sign may be assumed to be like this (taking the 3rd person singular present form).

(5.250) $\textsc{tuntuu} := \langle \lambda x.\texttt{tuntuu} \diamond x(\texttt{lta}), \mathsf{resemble}' \rangle$

The reason is that if we simply insist that the noun phrase comes equipped with the correct case, then it enters with its ablative case meaning rather than with its typical NP meaning. Notice namely that the ablative has an unmotivated appearance here given the semantics of the ablative case in Finnish. (Moreover, it is the only case possible with this verb.) So, semantically the situation is the same as if the verb was transitive. Notice that the fact that `tuntua` selects an ablative NP is a coincidence in this setup. The ablative form is directly added to the complement selected. This is not the best way of arranging things, and in [**?**] a proposal has been made to remedy the situation.

There is a list of potential problems for de Saussure grammars. We mention a few of them. The plural in German is formed with some stems by umlauting them (see Section 1.3). This is (at least on the surface) an operation that is not additive. As mentioned earlier, we shall discuss this phenomenon in Section 6.3. Another problem is what is known as **suppletion**. We exemplify this phenomenon with the gradation of Latin adjectives. Recall that adjectives in many languages possess three forms: a positive (`happy`) a comparative (`happier`) and a superlative (`happiest`). This is so in Latin. Table 18 gives some examples. Adjectives above the line are regularly formed, the ones below are irregular. Interesting is the fact that it is not the comparative or superlative suffix that is irregular: it is the root form itself. The expected form *`bonior` is replaced by `melior`: the root changes from `bon` to `mel`. (The English adjective `good` is also an example.) A different phenomenon is exhibited by English `worse`. The comparative is formed either by adding `er` (`better`, `faster`) or by adding `more`. The form `worse` resists a decomposition into a stem and a suffix. In the case of `worse` we speak of a **portmanteau morph**. Portmanteau morphs can be treated in de Saussure

Table 18. Gradation of Latin Adjectives

Positive	Comparative	Superlative
`parvus`	`parvior`	`parvissimus`
`beatus`	`beatior`	`beatissimus`
`bonus`	`melior`	`optimus`
`malus`	`peior`	`pessimus`

grammars only as lexical items (since we only allow additive phonological processes).

Notes on this section. A proposal similar to de Saussure grammars one has been made by Philippe de Groote [**?**].

Exercise 207. Recall from Section 3.5 the notion of a combinatory extension of categorial grammar. We may attempt the same for de Saussure grammars. Define a new mode of combination, B, as follows.

(5.251) $\mathtt{B}(\langle e,m\rangle)(\langle e',m'\rangle) := \langle \lambda x.e(e'(x)), \lambda y.m(m'(y))\rangle$

Here, e is of type $\mu \to \nu$, e' of type $\lambda \to \mu$ and x of type λ, so that the string term $\lambda x.e(e'(x))$ is of type $\lambda \to \nu$. Likewise for the semantics. Show that this extension does not generate different signs, it just increases the set of structure terms. Contrast this with the analogous extension of AB–grammars. Look especially at mixed composition rules.

Exercise 208. Review the facts from Exercise 186 on Arabic. Write a de Saussure grammar that correctly accounts for them. *Hint.* This is not so simple. First, define *schemes*, which are functions of type

(5.252) $s \to (s \to (s \to (s \to (s \to (s \to s)))))$

They provide a way of combining consonantism (root) and vocalism. The first three arguments form the consonantism, the remaining three the vocalism. The change in consonantism or vocalism can be defined on schemes before inserting the actual consonantism and vocalism.

Exercise 209. Write a de Saussure grammar that generates the facts of Mandarin shown in Exercise 187.

Exercise 210. In European languages, certain words inside a NP do not inflect for case (these are adverbs, relative clauses and other) and moreover, no

word can more than two cases. Define case marking functions that take care of this. (If you need concrete examples, you may elaborate the Finnish example using English substitute words.)

Exercise 211. We have talked briefly in Section 5.1 about Australian case marking systems. We shall simplify the facts (in particular the word order) as follows. We define a recursive translation from PN_Ω (Ω–terms t in Polish notation) inductively as follows. We assume case markers c_i, $i < \Omega$. For a constant term c, put $c^\diamond := c$. If F is an n–ary function symbol and t_i, $i < n$, terms then put

$$(5.253) \qquad (Ft_0 \cdots t_{n-1})^\diamond := F \diamond t_0^{\diamond\frown} \mathsf{c}_0 \diamond t_1^{\diamond\frown} \mathsf{c}_1 \diamond \cdots \diamond t_{n-1}^{\diamond\frown} \mathsf{c}_{n-1}$$

Write a de Saussure grammar that generates the set $\{\langle t^\diamond, t\rangle : t \in \mathrm{PN}_\Omega\}$.

Exercise 212. In many modern theories of grammar, so–called **functional heads** play a fundamental role. Functional elements are elements that are responsible for the correct shape of the structures, but have typically very little — if any — content. A particularly useful idea is to separate the content of an element from its syntax. For example, we may introduce the morphological type of a transitive verb (*tv*) without specifying any selectional behaviour.

$$(5.254) \qquad \mathrm{SEE} := \langle \mathsf{see} : tv, \mathsf{see}' \rangle$$

Then we assume one or two functional elements that turn this sign into the signs SEE_i, $i < 6$. Show how this can be done for the particular case of the signs SEE_i. Can you suggest a general recipe for words of arbitrary category? Do you see a solution of the problem of ablative case selection in Finnish?

Chapter 6
The Model Theory of Linguistic Structures

1. Categories

Up to now we have used plain nonterminal symbols in our description of syntactic categories — symbols with no internal structure. For many purposes this is not a serious restriction. But it does not allow to capture important regularities of language. We give an example from German. The sentences (6.1) – (6.6) are grammatical.

(6.1) `Ich sehe.`
I see-1.SG

(6.2) `Du siehst.`
You.SG see-2.SG

(6.3) `Er/Sie/Es sieht.`
He/She/It see-3.SG

(6.4) `Wir sehen.`
We see-1.PL

(6.5) `Ihr seht.`
You.PL see-2.PL

(6.6) `Sie sehen.`
They see-3.PL

By contrast, the following sentences are ungrammatical.

(6.7) *`Ich siehst/sieht/sehen/seht.`
I see-2.SG/see-3.SG/see-1/3.PL/see-2.PL

(6.8) *`Du sehe/sieht/sehen/seht.`
You.SG see-1.SG/see-3.SG/see-1/3.PL/see-2.PL

One says that the finite verb of German agrees with the subject in person and number. This means that the verb has different forms depending on whether the subject is in the 1st, 2nd or 3rd person, and whether it is singular or plural.

How can we account for this? On the one hand, we may simply assume that there are six different kinds of subjects (1st, 2nd or 3rd person, singular or plural) as well as five different kinds of verb forms (since two are homophonous, namely 1st and 3rd person plural). And the subjects of one kind can only cooccur with a matching verb form. But the grammars we looked at so far do not allow to express this fact at this level of generality; all one can do is provide lists of rules. A different way has been proposed among other in **Generalized Phrase Structure Grammar** (**GPSG**, see [**?**]). Let us start with the following basic rule.

(6.9) $\mathsf{S} \rightarrow \mathsf{NP} \quad \mathsf{VP}$

Here the symbols S, NP and VP are symbols not for a single category but for a whole set of them. (This is why we have not used typewriter font.) In fact, the labels are taken to be *descriptions of categories*. They are not string anymore. This means that these 'labels' can be combined using boolean connectives such as negation, conjunction and disjunction. For example, if we introduce the properties $\mathsf{1}$, $\mathsf{2}$ and $\mathsf{3}$ as well as Sg and Pl then our rule (6.9) can be refined as follows:

(6.10) $\mathsf{S} \rightarrow \mathsf{NP} \wedge \mathsf{1} \wedge \mathsf{Sg} \quad \mathsf{VP} \wedge \mathsf{1} \wedge \mathsf{Sg}$

Furthermore, we have the following terminal rules.

(6.11) $\mathsf{NP} \wedge \mathsf{1} \wedge \mathsf{Sg} \rightarrow \texttt{ich}, \quad \mathsf{VP} \wedge \mathsf{1} \wedge \mathsf{Sg} \rightarrow \texttt{sehe}$

Here $\mathsf{NP} \wedge \mathsf{1} \wedge \mathsf{Sg}$ is the description of a category which is a noun phrase (NP) in the first person ($\mathsf{1}$) singular (Sg). This means that we can derive the sentence (6.1). In order for the sentences (6.7) and (6.8) not to be derivable we now have to eliminate the rule (6.9). But this excludes the sentences (6.2) – (6.6). To get them back again we still have to introduce five more rules. These can however be fused into a single schematic rule. In place of NP we now write $[\textsc{cat} : \mathit{np}]$, in place of $\mathsf{1}$ we write $[\textsc{pers} : \mathit{1}]$, and in place of Pl we write $[\textsc{num} : \mathit{pl}]$. Here, we call CAT, PER and NUM **attributes**, and *np*, *vp*, *1*, and so on **values**. In the pair $[\textsc{cat} : \mathit{np}]$ we say that the attribute CAT has the value *np*. A set of pairs $[A : v]$, where A is an attribute and v a value is called an **attribute–value structure** or simply an **AVS**.

The rule (6.9) is now replaced by the schematic rule (6.12).

(6.12) $$\begin{bmatrix} \textsc{cat} & : & s \end{bmatrix} \rightarrow \begin{bmatrix} \textsc{cat} & : & \mathit{np} \\ \textsc{per} & : & \alpha \\ \textsc{num} & : & \beta \end{bmatrix} \quad \begin{bmatrix} \textsc{cat} & : & \mathit{vp} \\ \textsc{per} & : & \alpha \\ \textsc{num} & : & \beta \end{bmatrix}$$

Here, α and β are variables. However, they have different value range; α may assume values from the set $\{1,2,3\}$ β values from the set $\{sg,pl\}$. This fact shall be dealt with further below. One has to see to it that the properties inducing agreement are passed on. This means that the following rule also has to be refined in a similar way.

(6.13) $\quad \mathsf{VP} \rightarrow \mathsf{V} \quad \mathsf{NP}$

This rule says that a VP may be a constituent comprising a (transitive) verb and an NP. The agreement features have to be passed on to the verb.

(6.14) $$\begin{bmatrix} \text{CAT} & : & vp \\ \text{PER} & : & \alpha \\ \text{NUM} & : & \beta \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT} & : & v \\ \text{PER} & : & \alpha \\ \text{NUM} & : & \beta \end{bmatrix} \quad \begin{bmatrix} \text{CAT} & : & np \end{bmatrix}$$

Now, there are languages in which the verb not only agrees with the subject but also with the object in the same categories. This means that it does not suffice to simply write [PER : α]; we also have to say whether α concerns the subject or the object. Hence the structure relating to agreement has to be further embedded into the structure.

(6.15) $$\begin{bmatrix} \text{CAT} & : vp \\ \text{PER} & : \alpha \\ \text{NUM} & : \beta \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT} & : v \\ \text{AGRS} & : \begin{bmatrix} \text{PER} & : \alpha \\ \text{NUM} & : \beta \end{bmatrix} \\ \text{AGRO} & : \begin{bmatrix} \text{PER} & : \alpha' \\ \text{NUM} & : \beta' \end{bmatrix} \end{bmatrix} \quad \begin{bmatrix} \text{CAT} & : np \\ \text{PER} & : \alpha' \\ \text{NUM} & : \beta' \end{bmatrix}$$

It is clear that this rule does the job as intended. One can make it look even nicer by assuming also for the NP an embedded structure for the agreement complex. This is what we shall do below. Notice that the value of an attribute is now not only a single value but may in turn be an entire AVS. Thus, two kinds of attributes are distinguished. *1*, *sg* are called **atomic values**. In the present context, all basic expressions are either (atomic) values or attributes. Attributes which have only atomic values are called **Type 0 attributes**, all others are **Type 1 attributes**. This is the basic setup of [**?**]. In the so–called **Head Driven Phrase–Structure Grammar** by Carl Pollard and Ivan Sag (**HPSG**, see [**?**]) this has been pushed much further. In HPSG, the entire structure is encoded using AVSs of the kind just shown. Not only the bare linguistic features but also the syntactic structure itself is coded into AVSs.

We shall study these structures from a theoretical point of view in Section 6.6. Before we enter this investigation we shall move one step further. The rules that we have introduced above use variables for values of attributes. This certainly is a viable option. However, HPSG has gone into a different direction here. It introduces what are in fact structure variables, whose role it is to share entire AVSs between certain members of an AVS. To see how this works we continue with our example. Let us now write an NP not as a flat AVS, but let us instead embed the agreement related attribute value pairs as the value of an attribute AGR. A 3rd person NP in the plural is now represented as follows.

$$\text{(6.16)} \qquad \begin{bmatrix} \text{CAT} & : & \mathit{np} \\ \text{AGR} & : & \begin{bmatrix} \text{NUM} & : & \mathit{pl} \\ \text{PER} & : & \mathit{3} \end{bmatrix} \end{bmatrix}$$

The value of AGR is now structured in the same way as the values of AGRS and AGRO. Now we can rewrite our rules with the help of structure variables as follows. The rule (6.12) now assumes the form

$$\text{(6.17)} \qquad \begin{bmatrix} \text{CAT} & : & \mathit{s} \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT} & : & \mathit{np} \\ \text{AGR} & : & \boxed{1} \end{bmatrix} \quad \begin{bmatrix} \text{CAT} & : & \mathit{vp} \\ \text{AGRS} & : & \boxed{1} \end{bmatrix}$$

The rule that introduces the object now has this shape.

$$\text{(6.18)} \qquad \begin{bmatrix} \text{CAT} & : \mathit{vp} \\ \text{AGRS} & : \boxed{1} \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT} & : \mathit{v} \\ \text{AGRS} & : \boxed{1} \\ \text{AGRO} & : \boxed{2} \end{bmatrix} \quad \begin{bmatrix} \text{CAT} & : \mathit{np} \\ \text{AGR} & : \boxed{2} \end{bmatrix}$$

The labels $\boxed{1}$ and $\boxed{2}$ are variables for AVSs. If some variable occurs several times in a rule then every occurence stands for the same AVS. This is precisely what is needed to formulate agreement. AVS variables help to avoid that agreement blows up the rule apparatus beyond recognition. The rules have become once again small and perspicuous. (However, the agreement facts of languages are full of tiny details and exceptions, which make the introduction of more rules unavoidable.)

Now if AVSs are only the description, then what are categories? In a nutshell, it is thought that categories are *Kripke–frames*. One assumes a set of vertices and associates with each attribute a binary relation on this set. So, attributes are edge colours, atomic values turn into vertex colours. And a syntactic tree is no longer an exhaustively ordered tree with simple labels but an

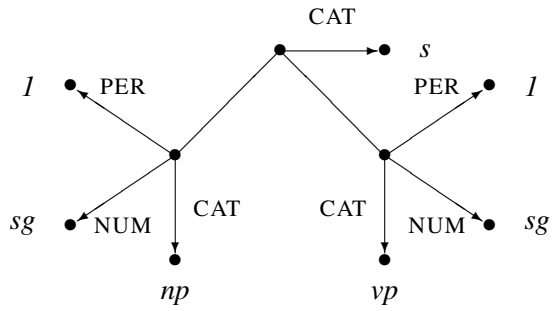


Figure 17. The Kripke–frame of an AVS

exhaustively ordered tree with labels having complex structure. Or, as it is more convenient, we shall assume that the tree structure itself also is coded by means of AVSs. The Figure 17 shows an example of a structure which — as one says — is **licensed** by the rule (6.17). The literature on AVSs is rich (see the books [**?**] and [**?**]). In its basic form, however, it is quite simple. Notice that it is a mistake to view attributes as objects. In fact, AVSs are not objects, they are descriptions of objects. Moreover, they can be the values of attributes. Therefore we treat values like *np*, *1* as properties which can be combined with the usual boolean operations, for example $\neg$, $\wedge$, $\vee$ or $\rightarrow$. This has the advantage that we are now able to represent the category of the German verb form `sehen` in either of the following ways.

(6.19) $$\begin{bmatrix} \text{CAT} & : & v \\ \text{PER} & : & 1 \vee 3 \\ \text{NUM} & : & pl \end{bmatrix} \quad \begin{bmatrix} \text{CAT} & : & v \\ \text{PER} & : & \neg 2 \\ \text{NUM} & : & pl \end{bmatrix}$$

The equivalence between these two follows only if we assume that the values of PER can be only *1*, *2* or *3*. This fact, however, is a fact of German, and will be part of the grammar of German. (In fact, it seems to hold pretty universally across languages.) Notice that the collocation of attribute–value pairs into an attribute–value structure is nothing but the logical conjunction. So the left hand AVS can also be written down as follows.

(6.20) $$[\text{CAT} : v] \wedge [\text{PER} : 1 \vee 3] \wedge [\text{NUM} : pl]$$

One calls **underspecification** the fact that a representation does not fix an object in all detail but that it leaves certain properties unspecified. Disjunctive

specifications are a case in point. However, they do not in fact provide the most welcome case. The most ideal case is when certain attributes are not contained in the AVS so that their actual value can be anything. For example, the category of the English verb form `saw` may be (partially!) represented thus.

(6.21) $$\begin{bmatrix} \text{CAT} & : & v \\ \text{TEMP} & : & past \end{bmatrix}$$

This means that we have a verb in the past tense. The number and person are simply not mentioned. We can — but need not — write them down explicitly.

(6.22) $$\begin{bmatrix} \text{CAT} & : & v \\ \text{TEMP} & : & past \\ \text{NUM} & : & \top \\ \text{PER} & : & \top \end{bmatrix}$$

Here $\top$ is the maximally unspecified value. We have — this is a linguistical, that is to say, an empirical, fact —:

(6.23) $$[\text{PER} : 1 \vee 2 \vee 3]$$

From this we can deduce that the category of `saw` also has the following representation.

(6.24) $$\begin{bmatrix} \text{CAT} & : & v \\ \text{TEMP} & : & past \\ \text{NUM} & : & \top \\ \text{PER} & : & 1 \vee 2 \vee 3 \end{bmatrix}$$

Facts of language are captured by means of axioms. More on that later.

Since attribute–value pairs are propositions, we can combine them in the same way. The category of the English verb form `see` has among other the following grammatical representation.

(6.25) $$\neg \begin{bmatrix} \text{CAT} & : & v \\ \text{PER} & : & 3 \\ \text{NUM} & : & sg \end{bmatrix} \vee \begin{bmatrix} \text{CAT} & : & v \\ \text{NUM} & : & pl \end{bmatrix}$$

This can alternatively be written as follows.

(6.26) $$[\text{CAT} : v] \wedge (\neg([\text{PER} : 3] \wedge [\text{NUM} : sg]) \vee [\text{NUM} : pl])$$

In turn, this can be simplified.

(6.27) $[\text{CAT} : v] \wedge (\neg[\text{PER} : 3] \vee [\text{NUM} : pl])$

This follows on the basis of the given interpretation. Since AVSs are not the objects themselves but descriptions thereof, we may exchange one description of an object or class of objects by any other description of that same object or class of objects. We call an AVS **universally true** if it is always true, that is, if it holds of every object.

① If φ is a tautology of propositional logic then φ holds for all replacements of AVSs for the propositional variables.

② If φ is universally true, then so is $[\mathrm{X} : \varphi]$.

③ $[\mathrm{X} : \varphi \to \chi]. \to .[\mathrm{X} : \varphi] \to [\mathrm{X} : \chi]$.

④ If φ and $\varphi \to \chi$ are universally true then so is χ.

In order to avoid having to use $\to$, we shall write $\varphi \leq \chi$ if $\varphi \to \chi$ is universally true. Most attributes are **definite**, that is, they can have at most one value in any object. For such attributes we also have

(6.28) $[\mathrm{X} : \varphi] \wedge [\mathrm{X} : \chi]. \to .[\mathrm{X} : \varphi \wedge \chi]$

Definite attributes are the norm. Sometimes, however, one needs nondefinite attributes; they are called **set valued** to distinguish them from the definite ones.

The AVSs are nothing but an alternative notation for formulae of some logical language. In the literature, two different kinds of logical languages have been proposed. Both serve the purpose equally well. The first is the so–called **monadic second order predicate logic** (MSO), which is a fragment of **second order logic** (SO). Second order logic extends standard first order predicate logic as follows. There additionally are variables and quantifiers for predicates of any given arity $n \in \omega$. The quantifiers are also written $\forall$ and $\exists$ and the variables are P_i^n, $n, i \in \omega$. Here, n tells us that the variable is a variable for n–ary relations. So, PdV $:= \{P_i^n : n, i \in \omega\}$ is the set of predicate variables for unary predicates and $V := \{x_i : i \in \omega\}$ the set of object variables. We write $P_i^n(\vec{x})$ to say that P_i^n applies to (the n–tuple) $\vec{x}$. If φ is a formula so are $(\forall P_i^n)\varphi$ and $(\exists P_i^n)\varphi$. The set of (M)SO–formulae defined over a given signature Ω is denoted by $\mathsf{SO}(\Omega)$ and $\mathsf{MSO}(\Omega)$, respectively. The

structures are the same as those of predicate logic (see Section 3.8): triples $\mathfrak{M} = \langle M, \{f^{\mathfrak{M}} : f \in F\}, \{r^{\mathfrak{M}} : r \in R\}\rangle$, where M is a nonempty set, $f^{\mathfrak{M}}$ the interpretation of the function f in $\mathfrak{M}$ and $r^{\mathfrak{m}}$ the interpretation of the relation r. A **model** is a triple $\langle \mathfrak{M}, \gamma, \beta\rangle$ where $\mathfrak{M}$ is a structure $\beta : V \to M$ a function assigning to each variable an element from M and γ: $P \to \wp(M)$ a function assigning to each n–ary predicate variable an n–ary relation on M. The relation $\langle \mathfrak{M}, \gamma, \beta\rangle \vDash \varphi$ is defined inductively.

(6.29) $\quad \langle \mathfrak{M}, \gamma, \beta\rangle \vDash P_i^n(\vec{x}) \quad :\Leftrightarrow \quad \beta(\vec{x}) \in \gamma(P_i^n)$

We define $\gamma \sim_P \gamma'$ if $\gamma'(Q) = \gamma(Q)$ for all $Q \neq P$.

(6.30) $\quad \langle \mathfrak{M}, \gamma, \beta\rangle \vDash (\forall P)\varphi :\Leftrightarrow$ for all $\gamma' \sim_P \gamma$: $\quad \langle \mathfrak{M}, \gamma', \beta\rangle \vDash \varphi$

(6.31) $\quad \langle \mathfrak{M}, \gamma, \beta\rangle \vDash (\exists P)\varphi :\Leftrightarrow$ for some $\gamma' \sim_P \gamma$: $\quad \langle \mathfrak{M}, \gamma', \beta\rangle \vDash \varphi$

We write $\mathfrak{M} \vDash \varphi$ iff for all γ and β $\langle \mathfrak{M}, \gamma, \beta\rangle \vDash \varphi$. MSO is that fragment of SO which has only predicate variables for unary relations ($n = 1$). When using MSO we drop the superscript '1' in the variables P_i^1.

Another type of languages that have been proposed are modal languages (see [**?**] and [**?**]). We shall pick out one specific language that is actually an extension of the ones proposed in the quoted literature, namely **quantified modal logic** (QML). This language possesses a denumerably infinite set $PV := \{p_i : i \in \omega\}$ of proposition variables, a set Md of so–called **modalities**, and a set Cd of propositional constants. And finally, there are the symbols $\neg$, $\wedge$, $\vee$, $\to$, $[-]$, $\langle - \rangle$, $\forall$ and $\exists$. Formulas (called propositions) are defined inductively in the usual way. Moreover, if φ is a proposition, so is $(\forall p_i)\varphi$ and $(\exists p_i)\varphi$. The notions of Kripke–frame and Kripe–model remain the same. A **Kripke–frame** is a triple $\langle F, R, C\rangle$, where $R : \mathrm{Md} \to \wp(F^2)$ and $C : \mathrm{Cd} \to \wp(F)$. If m is a modality, $R(m)$ is the accessibility relation associated with m. In particular, we have

(6.32) $\quad \langle \mathfrak{F}, \beta, x\rangle \vDash \langle m\rangle\varphi \quad :\Leftrightarrow \quad$ there is $y : x\, R(m)\, y$ and $\langle \mathfrak{F}, \beta, y\rangle\varphi$

For the constants we put

(6.33) $\quad \langle \mathfrak{F}, \beta, x\rangle \vDash c \quad :\Leftrightarrow \quad x \in C(c)$

We define

(6.34)
$$\begin{array}{llll} \langle \mathfrak{F}, \beta, x\rangle & \vDash (\forall p)\varphi & :\Leftrightarrow & \text{for all } \beta' \sim_p \beta : \quad \langle \mathfrak{F}, \beta', x\rangle \vDash \varphi \\ \langle \mathfrak{F}, \beta, x\rangle & \vDash (\exists p)\varphi & :\Leftrightarrow & \text{for some } \beta' \sim_p \beta : \quad \langle \mathfrak{F}, \beta', x\rangle \vDash \varphi \end{array}$$

Table 19. Translating QML into MSO

$$
\begin{array}{rlrl}
p_i^\dagger & := P_i(x_0) & c^\dagger & := Q^c(x_0) \\
(\neg\varphi)^\dagger & := \neg\varphi^\dagger & (\varphi_1 \wedge \varphi_2)^\dagger & := \varphi_1^\dagger \wedge \varphi_2^\dagger \\
(\varphi_1 \vee \varphi_2)^\dagger & := \varphi_1^\dagger \vee \varphi_2^\dagger & (\varphi_1 \to \varphi_2)^\dagger & := \varphi_1^\dagger \to \varphi_2^\dagger \\
((\forall p_i)\varphi)^\dagger & := (\forall P_i)\varphi^\dagger & ((\exists p_i)\varphi)^\dagger & := (\exists P_i)\varphi^\dagger \\
([m]\varphi)^\dagger & := (\forall x_0)(r^m(x_0,x_i) \to [x_i/x_0]\varphi^\dagger) & & \\
(\langle m\rangle\varphi)^\dagger & := (\exists x_0)(r^m(x_0,x_i) \wedge [x_i/x_0]\varphi^\dagger) & &
\end{array}
$$

We write $\langle \mathfrak{F}, \beta\rangle \vDash \varphi$ if for all $x \in F$ $\langle \mathfrak{F}, \beta, x\rangle \vDash \varphi$; we write $\mathfrak{F} \vDash \varphi$, if for all β we have $\langle \mathfrak{F}, \beta\rangle \vDash \varphi$.

We define an embedding of $\mathsf{QML}(\Omega)$ into $\mathsf{MSO}(\Omega^m)$, where Ω^m is defined as follows. Let $R := \{r^m : m \in \mathrm{Md}\}$ and $\mathrm{Cd} := \{Q^c : c \in K\}$. $\Omega^m(r^m) := 2$, $\Omega^m(Q^c) := 1$. Then define $\varphi^\dagger$ as in Table 19. Here in the last two clauses x_i is a variable that does not already occur in $\varphi^\dagger$. Finally, if $\mathfrak{F}$ is a Kripke–frame, we define an MSO–structure $\mathfrak{F}^m$ as follows. The underlying set is F, $(r^m)^{\mathfrak{F}^m} := R(m)$ for every $m \in \mathrm{Md}$, and $(Q^c)^{\mathfrak{F}^m} := C(c)$. Now we have

Theorem 6.1 *Let $\varphi \in \mathsf{QML}(\Omega)$. Then $\varphi^\dagger \in \mathsf{MSO}(\Omega^m)$. And for every Kripke–frame $\mathfrak{F}$: $\mathfrak{F} \vDash \varphi$ iff $\mathfrak{F}^m \vDash \varphi^\dagger$.*

Proof. We shall show the following. Assume $\beta\colon PV \to \wp(F)$ is a valuation in $\mathfrak{F}$ and that $x \in F$ and γ: PdV $\to \wp(F)$ and $\delta\colon V \to F$ valuations for the predicate and the object variables. Then if $\gamma(P_i) = \beta(p_i)$ for all $i \in \omega$ and $\delta(x_0) = x$ we have

$$\langle \mathfrak{F}, \beta, x\rangle \vDash \varphi \quad \Leftrightarrow \quad \langle \mathfrak{F}^m, \gamma, \delta\rangle \vDash \varphi^\dagger \tag{6.35}$$

It is not hard to see that (6.35) allows to derive the claim. We prove (6.35) by induction. If $\varphi = p_i$ then $\varphi^\dagger = P_i(x_0)$ and the claim holds in virtue of the fact that $\beta(p_i) = \gamma(P_i)$ and $\gamma(x_0) = x$. Likewise for $\varphi = c \in C$. The steps for $\neg$, $\wedge$, $\vee$ and $\to$ are routine. Let us therefore consider $\varphi = (\exists p_i)\eta$. Let $\langle \mathfrak{F}, \beta, x\rangle \vDash \varphi$. Then for some β' which differs from β at most in p_i: $\langle \mathfrak{F}, \beta', x\rangle \vDash \eta$. Put γ' as follows: $\gamma'(P_i) := \beta'(p_i)$ for all $i \in \omega$. By induction hypothesis $\langle \mathfrak{F}^m, \gamma', \delta\rangle \vDash \eta^\dagger$ and γ' differs from γ at most in P_i. Therefore we have $\langle \mathfrak{F}^m, \gamma, \delta\rangle \vDash (\exists P_i)\eta^\dagger = \varphi^\dagger$, as desired. The argument can be reversed, and the case is therefore settled. Analogously for $\varphi = (\forall P_i)\eta$. Now for $\varphi = \langle m\rangle\eta$. Let $\langle \mathfrak{F}, \beta, x\rangle \vDash \varphi$. Then there exists a y with $x\, r^m\, y$ and $\langle \mathfrak{F}, \beta, y\rangle \vDash \eta$.

Choose δ' such that $\delta'(x_0) = y$ and $\delta'(x_i) = \delta(x_i)$ for every $i > 0$. Then by induction hypothesis $\langle \mathfrak{F}^m, \gamma, \delta' \rangle \vDash \eta^\dagger$. If x_i is a variable that does not occur in $\eta^\dagger$ then let $\delta''(x_i) := \delta'(x_i)$, $\delta''(x_0) := x$ and $\delta''(x_j) := \delta'(x_j)$ for all $j \notin \{0, i\}$. Then $\langle \mathfrak{F}^m, \gamma, \delta'' \rangle \vDash r^m(x_0, x_i); [x_i/x_0]\eta^\dagger = \varphi^\dagger$. Hence we have $\langle \mathfrak{F}^m, \gamma, \delta'' \rangle \vDash \varphi^\dagger$. Now it holds that $\delta''(x_0) = x = \delta(x_0)$ and x_i is bound. Therefore also $\langle \mathfrak{F}^m, \gamma, \delta \rangle \vDash \varphi^\dagger$. Again the argument is reversible, and the case is proved. Likewise for $\varphi = [m]\eta$. □

Exercise 213. Let 1, 2 and 3 be modalities. Show that a Kripke–frame satisfies the following formula iff $R(3) = R(1) \cup R(2)$.

(6.36) $\langle 3 \rangle p \leftrightarrow \langle 1 \rangle p \vee \langle 2 \rangle p$

Exercise 214. Let 1, 2 and 3 be modalities. Show that a Kripke–frame satisfies the following formula iff $R(3) = R(1) \circ R(2)$.

(6.37) $\langle 3 \rangle p \leftrightarrow \langle 1 \rangle \langle 2 \rangle p$

Exercise 215. In HPSG one writes $[\text{CAT} : \alpha \oplus \beta]$ if CAT has at least two values: α and β. (If $\alpha \wedge \beta$ is consistent, CAT can take also one value, $\alpha \wedge \beta$, but not necessarily.) Devise a translation into QML for $\oplus$. What if $[\text{cat} : \alpha \oplus \beta]$ also means that CAT can have no other value than α and β?

Exercise 216. Let r be a binary relation symbol. Show that in a model of MSO the following holds: $\langle \mathfrak{M}, \gamma, \beta \rangle \vDash Q(x, y)$ iff $x\ (r^{\mathfrak{M}})^*\ y$ (this means that y can be reached from x in finitely many r–steps).

(6.38) $Q(x, y) := (\forall P)(P(x) \wedge (\forall yz)(P(y) \wedge y\ r\ z. \rightarrow .P(z)). \rightarrow .P(y))$

2. Axiomatic Classes I: Strings

For the purposes of this chapter we shall code strings in a new way. This will result in a somewhat different formalization than the one discussed in Section 1.4. The differences are, however, marginal.

Definition 6.2 *A **Z–structure over the alphabet** A is a triple of the form $\mathfrak{L} = \langle L, \prec, \{Q_a : a \in A\} \rangle$, where L is an arbitrary finite set, $\{Q_a : a \in A\}$ a partition of L and $\prec$ a binary relation on L such that both $\prec^+$ and its inverse are linear, irreflexive, total orderings of L.*

Z–structures are not strings. However, it is not difficult to define a map which assigns a string to each Z–structure. However, if $L \neq \varnothing$ there are infinitely many Z–structures which have the same string associated with them and they form a proper class.

Fix A and denote by MSO the MSO–language of the binary relation symbol $\preceq$ as well as a unary predicate constant $\underline{a}$ for every $a \in A$.

Definition 6.3 *Let $\mathcal{K}$ be a set or class of Z–structures over an alphabet A. Then $\mathsf{Th}\,\mathcal{K}$ denotes the set $\{\varphi \in \mathsf{MSO} : \text{ for all } \mathfrak{L} \in \mathcal{K} : \mathfrak{L} \vDash \varphi\}$, called the **MSO–theory of** $\mathcal{K}$. If Φ is a set of sentences from MSO then let $\mathsf{Mod}\,\Phi$ be the set of all $\mathfrak{L}$ which satisfy every sentence from Φ. $\mathsf{Mod}\,\Phi$ is called the **model class of** Φ.*

Recall from Section 1.1 the notion of a context. It is easy to see that $\vDash$ together with the class of Z–structures and the MSO–formulae form a context. From this we directly get the following

Theorem 6.4 *The map $\rho : \mathcal{K} \mapsto \mathsf{Mod}\,\mathsf{Th}\,\mathcal{K}$ is a closure operator on the class of of classes Z–structures over A. Likewise, $\lambda : \Phi \mapsto \mathsf{Th}\,\mathsf{Mod}\,\Phi$ is a closure operator on the set of all subsets of* MSO.

(We hope that the reader does not get irritated by the difference between classes and sets. In the usual set theory one has to distinguish between sets and classes. Model classes are except for trivial exception always classes while classes of formulae are always sets, because they are subclasses of the set of formulae. This difference can be neglected in what is to follow.) We now call the sets of the form $\lambda(\mathcal{K})$ **logics** and the classes of the form $\rho(\Phi)$ **axiomatic classes**. A class is called **finitely MSO–axiomatizable** if it has the form $\mathsf{Mod}(\Phi)$ for a finite Φ, while a logic is **finitely MSO–axiomatizable** if it is the logic of a finitely axiomatizable class. We call a class of Z–structures over A **regular** if it is the class of all Z–structures of a regular language. Formulae are called **valid** if they hold in all structures. The following result from [**?**] is the central theorem of this section.

Theorem 6.5 (Büchi) *A class of Z–structures is finitely MSO–axiomatizable iff it corresponds to a regular language which does not contain ε.*

This sentence says that with the help of MSO we can only define regular classes of Z–structures. If one wants to describe nonregular classes, one has to use stronger logical languages (for example SO). The proof of this theorem requires a lot of work. Before we begin, we have to say something about

Table 20. The Formula $\delta(\mathfrak{A})$

$$
\begin{array}{lll}
\delta(\mathfrak{A}) := & (\forall xyz)(x \preceq y \wedge x \preceq z. \rightarrow .y = z) & \text{(a)} \\
\wedge & (\forall xyz)(y \preceq x \wedge z \preceq x. \rightarrow .y = z) & \text{(b)} \\
\wedge & (\forall P)\{(\forall xy)(x \preceq y. \rightarrow .P(x) \rightarrow P(y)) \wedge (\exists x)P(x). & \\
 & \quad \rightarrow .(\exists x)(P(x) \wedge (\forall y)(x \preceq y \rightarrow \neg P(y)))\} & \text{(c)} \\
\wedge & (\forall P)\{(\forall xy)(x \preceq y. \rightarrow .P(y) \rightarrow P(x)) \wedge (\exists x)P(x). & \\
 & \quad \rightarrow .(\exists x)(P(x) \wedge (\forall y)(y \preceq x \rightarrow \neg P(y)))\} & \text{(d)} \\
\wedge & (\forall P)\{(\forall xy)(x \preceq y \rightarrow (P(x) \leftrightarrow P(y))) & \\
 & \quad \wedge(\exists x)P(x). \rightarrow .(\forall x)P(x)\} & \text{(e)} \\
\wedge & (\forall x) \bigvee_{a \in A} \underline{a}(x) & \text{(f)} \\
\wedge & (\forall x) \bigwedge_{a \neq b} \underline{a}(x) \rightarrow \neg \underline{b}(x) & \text{(g)} \\
\wedge & (\exists P_0 P_1 \cdots P_{n-1})\{(\forall x)((\forall y)\neg(y \preceq x). \rightarrow .P_0(x)) & \\
 & \quad \wedge(\forall x)((\forall y)\neg(x \preceq y). \rightarrow . \bigvee_{i \in F} P_i(x)) & \\
 & \quad \wedge(\forall xy)(x \preceq y \rightarrow \bigwedge_{a \in A}[\underline{a}(y) \wedge P_i(x). & \\
 & \quad\quad \rightarrow . \bigvee_{j \in \delta(i,a)} P_j(y)])\} & \text{(h)}
\end{array}
$$

the formulation of the theorem. By definition, models are only defined on nonempty sets. This is why a model class always defines a language not containing ε. It is possible to change this but then the Z–structure of ε (which is actually unique) is a model of every formula, and then $\varnothing$ is regular but not MSO–axiomatizable. So, complete correspondence cannot be expected. But this is the only exception.

Let us begin with the simple direction. This is the claim that every regular class is finitely MSO–axiomatizable. Let $\mathcal{K}$ be a regular class and L the corresponding regular language. Then there exists a finite state automaton $\mathfrak{A} = \langle A, Q, i_0, F, \delta \rangle$ with $L(\mathfrak{A}) = L$. We may choose $Q := n$ for a natural number n and $i_0 = 0$. Look at the sentence $\delta(\mathfrak{A})$ defined in Table 20.

Lemma 6.6 *Let $\mathfrak{L}$ be an MSO–structure. Then $\mathfrak{L} \models \delta(\mathfrak{A})$ iff $\mathfrak{L}$ is a Z–structure and its associated string is in $L(\mathfrak{A})$.*

Proof. Let $\mathfrak{L} \models \delta(\mathfrak{A})$ and let $\mathfrak{L}$ be an MSO–structure. Then there exists a binary relation $\prec$ (the interpretation of $\preceq$) and for every $a \in A$ a subset $Q_a \subseteq L$. By (a) and (b) an element x has at most one $\prec$–successor and at most one $\prec$–predecessor. By (c), every nonempty subset which is closed under $\prec$–successors contains a last element, and by (d) every nonempty subset which is

closed under $\prec$–predecessors contains a first element. Since L is not empty, it has a least element, x_0. Let $H := \{x_i : i < \kappa\}$ be a maximal set such that x_{i+1} is the (unique) $\prec$–successor of x_i. H cannot be infinite, for otherwise $\{x_i : i < \omega\}$ would be a successor closed set without last element. So, H is finite. H is also closed under predecessors. So, H is a maximal connected subset of L. By (e), every maximal connected nonempty subset of L is identical to L. So, $H = L$, and hence L is finite, connected, and linear in both directions.

Further, by (f) and (g) every $x \in L$ is contained in exactly one set Q_a. Therefore $\mathfrak{L}$ is a Z–structure. We have to show that its string is in $L(\mathfrak{A})$. (h) says that we can find sets $H_i \subseteq L$ for $i < n$ such that if x is the first element with respect to $\prec$ then $x \in H_0$, if x is the last element with respect to $\prec$ then $x \in H_j$ for some $j \in F$ and if $x \in H_i$, $y \in Q_a$, $x \prec y$ then $y \in H_j$ for some $j \in \delta(i,a)$. This means that the string is in $L(\mathfrak{A})$. (There is namely a biunique correspondence between accepting runs of the automaton and partitions into H_i. Under that partition $x \in H_i$ means exactly that the automaton is in state i at x in that run.) Now let $\mathfrak{L} \not\models \delta(\mathfrak{A})$. Then either $\mathfrak{L}$ is not a Z–structure or there exists no accepting run of the automaton $\mathfrak{A}$. Hence the string is not in $L(\mathfrak{A})$. This concludes the proof. □

Notice that we can define $<$, the transitive closure of $\prec$, and $>$, the transitive closure of $\succ$ (and converse of $<$) by an MSO–formula (see Exercise 216). Further, we will write $x \leq y$ for $x < y \vee x = y$, and $x \prec y$ in place of $x \preceq y$. Now, given a Z–structure $\mathfrak{L} = \langle L, \prec, \{Q_a : a \in A\}\rangle$ put

$$M(\mathfrak{L}) := \langle L, \prec, \succ, <, >, \{Q_a : a \in A\}\rangle \tag{6.39}$$

A structure of the form $M(\mathfrak{L})$ we call an **MZ–structure**.

Now we shall prove the converse implication of Theorem 6.5. To this end we shall make a detour. We put $M := \{+, -, \prec, \succ\}$ and $C := \{c_a : a \in A\}$. Then we call QML the language of quantified modal logic with basic modalities from M and propositional constants from C. Now we put

$$\begin{array}{lll} \sigma := & \langle\prec\rangle p \to \langle+\rangle p & \wedge\langle\succ\rangle p \to \langle-\rangle p \\ & \wedge p \to [\succ]\langle\prec\rangle p & \wedge p \to [\prec]\langle\succ\rangle p \\ & \wedge\langle\prec\rangle p \to [\prec]p & \wedge\langle\succ\rangle p \to [\succ]p \\ & \wedge\langle+\rangle\langle+\rangle p \to \langle+\rangle p & \wedge\langle-\rangle\langle-\rangle p \to \langle-\rangle p \\ & \wedge[+]([+]p \to p) \to [+]p & \wedge[-]([-]p \to p) \to [-]p \\ & \wedge\bigwedge_{a\neq b} c_a \to \neg c_b & \wedge\bigvee_{a\in A} c_a \end{array} \tag{6.40}$$

We call a structure **connected** if it is not of the form $\mathfrak{F} \oplus \mathfrak{G}$ with nonempty

$\mathfrak{F}$ and $\mathfrak{G}$. As already mentioned, QML–formulae cannot distinguish between connected and nonconnected structures.

Theorem 6.7 *Let $\mathfrak{F}$ be a connected Kripke–frame for* QML. *Put $Z(\mathfrak{F}) := \langle F, R(\prec), R(\succ), R(+), R(-), \{K(c^a) : a \in A\}\rangle$. $\mathfrak{F} \vDash \sigma$ iff $Z(\mathfrak{F})$ is an MZ–structure over A.*

Proof. The proof is not hard but somewhat lengthy. It consists of the following facts (the others are dual to these).

① $\mathfrak{F} \vDash \langle\prec\rangle p \to \langle+\rangle p$ *iff* $R(\prec) \subseteq R(+)$.

② $\mathfrak{F} \vDash p \to [\succ]\langle\prec\rangle p$ *iff* $R(\succ) \subseteq R(\prec)^{\smile}$.

③ $\mathfrak{F} \vDash \langle\prec\rangle p \to [\prec]p$ *iff every point has at most one* $R(\prec)$*–successor.*,

④ $\mathfrak{F} \vDash \langle+\rangle\langle+\rangle p \to \langle+\rangle p$ *iff* $R(+)$ *is transitive.*

⑤ $\mathfrak{F} \vDash [+](p \to [+]p) \to [+]p$ *iff* $R(+)$ *is transitive and free of infinite ascending chains (or cycles).*

We show only ① and ⑤. ① For this let $\mathfrak{F} \nvDash \langle\prec\rangle p \to \langle+\rangle p$. Then there exists β and x such that $\langle\mathfrak{F}, \beta, x\rangle \vDash \langle\prec\rangle p; \neg\langle+\rangle p$. This means that there is a $y \in F$ with $y \in \beta(p)$ and $x\, R(\prec)\, y$. If $x\, R(+)\, y$ then $x \vDash \langle+\rangle p$, contradiction. This shows $R(\prec) \nsubseteq R(+)$. Assume conversely $R(\prec) \nsubseteq R(+)$. Then there exist x and y such that $x\, R(\prec)\, y$ but not $x\, R(+)\, y$. Now set $\beta(p) := \{y\}$. Then we have $\langle\mathfrak{F}, \beta, x\rangle \vDash \langle\prec\rangle p; \neg\langle+\rangle p$. ⑤ Because of ④ we restrict ourselves to proving this for transitive $R(+)$. Assume $\mathfrak{F} \nvDash [+]([+]p \to p) \to [+]p$. Then there exists a β and a x_0 with $\langle\mathfrak{F}, \beta, x\rangle \vDash [+](p \to [+]p); \langle+\rangle\neg p$. So there exists a x_1 with $x_0\, R(+)\, x_1$ and $x_1 \notin \beta(p)$. Then $x_1 \vDash [+]p \to p$ and therefore $x_1 \vDash \langle+\rangle\neg p$. Because of the transitivity of $R(+)$ we also have $x_1 \vDash [+]([+]p \to p)$. Repeating this argument we find an infinite chain $\langle x_i : i \in \omega\rangle$ such that $x_i\, R(+)\, x_{i+1}$. Therefore, $R(+)$ contains an infinite ascending chain. Conversely, assume that $R(+)$ has an infinite ascending chain. Then there exists a set $\langle x_i : i \in \omega\rangle$ with $x_i\, R(+)\, x_{i+1}$ for all $i \in \omega$. Put $\beta(p) := \{y :$ there is $i \in \omega : y\, R(+)\, x_i\}$. Then it holds that $\langle\mathfrak{F}, \beta, x_0\rangle \vDash [+]([+]p \to p); \langle+\rangle\neg p$. For let $x\, R(+)\, y$ and suppose that $y \vDash [+]p$. Then there exists an i with $y\, R(+)\, x_i$ (for $R(+)$ is transitive). Hence $y \vDash p$, whence $y \vDash [+]p \to p$. Since y was arbitrary, we have $x \vDash [+]([+]p \to p)$. Also $x_1 \nvDash p$ and $x_0\, R(+)\, x_1$. Hence $x_0 \vDash \langle+\rangle\neg p$, as required. □

Notice that a finite frame has an infinite ascending chain iff it has a cycle. Now we define an embedding of MSO into QML. To this end we need some preparations. As one convinces oneself easily the following laws hold.

① $(\forall x)\varphi \leftrightarrow \neg(\exists x)(\neg\varphi)$, $\neg\neg\varphi \leftrightarrow \varphi$.

② $(\forall x)(\varphi_1 \wedge \varphi_2) \leftrightarrow (\forall x)\varphi_1 \wedge (\forall x)\varphi_2$,
$(\exists x)(\varphi_1 \vee \varphi_2) \leftrightarrow (\exists x)\varphi_1 \vee (\exists x)\varphi_2$.

③ $(\forall x)(\varphi_1 \vee \varphi_2) \leftrightarrow (\varphi_1 \vee (\forall x)\varphi_2)$, $(\exists x)(\varphi_1 \wedge \varphi_2) \leftrightarrow (\varphi_1 \wedge (\exists x)\varphi_2)$, if x does not occur freely in φ_1.

Finally, for every variable $y \neq x$:

$$(\forall x)\varphi(x) \leftrightarrow \quad (\forall x)(x < y \to \varphi(x)) \wedge (\forall x)(y < x \to \varphi(x)) \wedge \varphi(y) \tag{6.41}$$

We now define following quantifiers.

$$\begin{aligned} (\forall x < y)\varphi &:= (\forall x)(x < y \to \varphi) \\ (\forall x > y)\varphi &:= (\forall x)(x > y \to \varphi) \\ (\exists x < y)\varphi &:= (\exists x)(x < y \to \varphi) \\ (\exists x > y)\varphi &:= (\exists x)(x > y \to \varphi) \end{aligned} \tag{6.42}$$

We call these quantifiers **restricted**. Evidently, we can replace an unrestricted quantifier $(\forall x)\varphi$ by the conjunction of restricted quantifiers.

Lemma 6.8 *For every MSO–formula φ with at least one free object variable there is an MSO–formula φ^g with restricted quantifiers such that in all Z–structures $\mathfrak{L}$: $\mathfrak{L} \vDash \varphi \leftrightarrow \varphi^g$.*

We define the following functions f, f^+, g and g^+ on unary predicates.

$$\begin{aligned} (f(\varphi))(x) &:= (\exists y \prec x)\varphi(y) \\ (g(\varphi))(x) &:= (\exists y \succ x)\varphi(y) \\ (f^+(\varphi))(x) &:= (\exists y < x)\varphi(y) \\ (g^+(\varphi))(x) &:= (\exists y > x)\varphi(y) \end{aligned} \tag{6.43}$$

A somewhat more abstract approach is provided by the notion of a *universal modality*.

Table 21. Mimicking the Variables in QML

$\{P_x/x\}(x=y)$	$:= P_x(y)$	$\{P_x/x\}(y=x)$	$:= P_x(y)$
$\{P_x/x\}(v=w)$	$:= v=w$	$\{P_x/x\}(x \prec y)$	$:= (g(P_x))(y)$
$\{P_x/x\}(y \prec x)$	$:= (f(P_x))(y)$	$\{P_x/x\}(v \prec w)$	$:= v \prec w$
$\{P_x/x\}(\underline{a}(x))$	$:= \underline{a}(x)$	$\{P_x/x\}(\neg\varphi)$	$:= \neg\{P_x/x\}\varphi$
$\{P_x/x\}(\varphi_1 \wedge \varphi_2)$	$:= \{P_x/x\}\varphi_1 \wedge \{P_x/x\}\varphi_2$		
$\{P_x/x\}(\varphi_1 \vee \varphi_2)$	$:= \{P_x/x\}\varphi_1 \vee \{P_x/x\}\varphi_2$		
$\{P_x/x\}((\exists y)\varphi)$	$:= (\exists y)\{P_x/x\}\varphi$		
$\{P_x/x\}((\exists x)\varphi)$	$:= (\exists x)\varphi$		
$\{P_x/x\}((\forall P)\varphi)$	$:= (\forall P)\{P_x/x\}\varphi$		
$\{P_x/x\}((\exists P)\varphi)$	$:= (\exists P)\{P_x/x\}\varphi$		

Definition 6.9 *Let M be a set of modalities and $\omega \in M$. Further, let L be a* QML(M)*–modal logic. ω is called a* ***universal modality of*** *L if the following formulae are contained in L:*

① $[\omega]p \to p$, $[\omega]p \to [\omega][\omega]p$, $p \to [\omega]\langle\omega\rangle p$.

② $[\omega]p \to [m]p$, *for all* $m \in M$.

Proposition 6.10 *Let L be a* QML(M)*–logic, $\omega \in M$ a universal modality and $\mathfrak{F} = \langle F, R\rangle$ a connected Kripke–frame with $\mathfrak{F} \vDash L$. Then $R(\omega) = F \times F$.*

The proof is again an exercise. The logic of Z–structures allows to define a universal modality. Namely, set

(6.44) $$[\omega]\varphi := \varphi \wedge [+]\varphi \wedge [-]\varphi$$

This satisfies the requirements above.

The obvious mismatch between MSO and QML is that the former allows for several object variables to occur freely, while the latter only contains one free object variable (the world variable), which is left implicit. However, given that φ contains only one free object variable, we can actually massage it into a form suitable for QML. Let P_x be a predicate variable which does not occur in φ. Define $\{P_x/x\}\varphi$ inductively as in Table 21. (Here, assume that $x \notin \{v, w\}$ and $x \neq y$.) Let $\gamma(P_x) = \{\beta(x)\}$. Then

(6.45) $$\langle\mathfrak{M}, \gamma, \beta\rangle \vDash \varphi \quad \Leftrightarrow \quad \langle\mathfrak{M}, \gamma, \beta\rangle \vDash \{P_x/x\}\varphi$$

Lemma 6.11 *Let*

$$\begin{aligned}\nu(P) := (\exists x)(P(x)) \wedge (\forall x)(P(x) \to \neg(f^+(P))(x)) \\ \wedge (\forall x)(P(x) \to \neg(g^+(P))(x))\end{aligned} \tag{6.46}$$

Then $\langle \mathfrak{M}, \gamma, \beta \rangle \vDash \nu(P)$ *iff* $\gamma(P) = \{x\}$ *for some* $x \in M$.

This is easy to show. Notice that $\nu(P)$ contains no free occurrences of x. This is crucial, since it allows to directly translate $\nu(P)$ into a QML–formula.

Now we define an embedding of MSO into QML. Let $h\colon P \to PV$ be a bijection from the set of predicate variables of MSO onto the set of propositional variables of QML.

$$\begin{array}{rlrl}
(\underline{a}(x))^\diamond := c_a & & (\neg\varphi)^\diamond := \neg\varphi^\diamond \\
(P(y))^\diamond := h(P) & & (\varphi_1 \wedge \varphi_2)^\diamond := \varphi_1^\diamond \wedge \varphi_2^\diamond \\
(f(\varphi))^\diamond := \langle \succ \rangle \varphi^\diamond & & (\varphi_1 \vee \varphi_2)^\diamond := \varphi_1^\diamond \vee \varphi_2^\diamond \\
(g(\varphi))^\diamond := \langle \prec \rangle \varphi^\diamond & & ((\exists P)\varphi)^\diamond := (\exists h(P))\varphi^\diamond \\
(f^+(\varphi))^\diamond := \langle - \rangle \varphi^\diamond & & ((\forall P)\varphi)^\diamond := (\forall h(P))\varphi^\diamond \\
(g^+(\varphi))^\diamond := \langle + \rangle \varphi^\diamond & &
\end{array} \tag{6.47}$$

For the first–order quantifiers we put

$$\begin{aligned}((\forall x)\varphi(x))^\diamond := (\forall p_x)((\langle \omega \rangle p_x \wedge (\forall p'_x)([\omega](p'_x \to p_x) \\ \to ([\omega]\neg p_x. \vee .[\omega](p'_x \leftrightarrow p_x)))) \to \{p_x/p\}\varphi^\diamond)\end{aligned} \tag{6.48}$$

and

$$\begin{aligned}((\exists x)\varphi(x))^\diamond := (\forall p_x)((\langle \omega \rangle p_x \wedge (\forall p'_x)([\omega](p'_x \to p_x) \\ \to ([\omega]\neg p_x. \vee .[\omega](p'_x \leftrightarrow p_x)))) \wedge \{p_x/x\}\varphi^\diamond)\end{aligned} \tag{6.49}$$

The correctness of this translation follows from the fact that

$$\begin{aligned}\langle \mathfrak{F}, \beta, x \rangle \vDash \langle \omega \rangle p_x \wedge (\forall p'_x)([\omega](p'_x \to p_x) \\ \to ([\omega]\neg p_x) \vee [\omega](p'_x \leftrightarrow p_x)))\end{aligned} \tag{6.50}$$

exactly if $\beta(p_x) = \{v\}$ for some $v \in F$.

Theorem 6.12 *Let* $\varphi \in$ MSO *contain at most one free variable, the object variable* x_0. *Then there exists a QML–formula* φ^M *such that for all Z–structures* $\mathfrak{L}$*:*

$$\langle \mathfrak{L}, \beta \rangle \vDash \varphi(x_0) \text{ iff } \langle M(\mathfrak{L}), \beta(x_0) \rangle \vDash \varphi(x_0)^M \tag{6.51}$$

Corollary 6.13 *Modulo the identification $\mathfrak{L} \mapsto M(\mathfrak{L})$, MSO and QML define the same classes of connected nonempty and finite Z–structures. Further: $\mathcal{K}$ is a finitely MSO–axiomatizable class of Z–structures iff $M(\mathcal{K})$ is a finitely QML–axiomatizable class of MZ–structures.*

This shows that it is sufficient to prove that finitely QML–axiomatizable classes of MZ–structures define regular languages. This we shall do now. Notice that for the proof we only have to look at grammars with rules of the form $X \to a \mid aY$ and no rules of the form $X \to \varepsilon$. Furthermore, instead of regular grammars we can work with regular grammars*, where we have a set of start symbols.

Let $G = \langle \Sigma, N, A, R \rangle$ be a regular grammar* and $\vec{x}$ a string. For a derivation of $\vec{x}$ we define a Z–structure over $A \times N$ (!) as follows. We consider the grammar* $G^\times := \langle \Sigma, N, A \times N, R^\times \rangle$ which consists of the following rules.

$$(6.52) \qquad \begin{aligned} R^\times := \quad & \{X \to \langle a, X \rangle\, Y : X \to aY \in R\} \\ & \cup \{X \to \langle a, X \rangle : X \to a \in R\} \end{aligned}$$

The map $h\colon A \times N \to A\colon \langle a, X \rangle \mapsto a$ defines a homomorphism from $(A \times N)^*$ to A^*, which we likewise denote by h. It also gives us a map from Z–structures over $A \times N$ to Z–structures over A. Every G–derivation of $\vec{x}$ uniquely defines a $G^\times$–derivation of a string $\vec{x}^\times$ with $h(\vec{x}^\times) = \vec{x}$ and this in turn defines a Z–structure

$$(6.53) \qquad \mathfrak{M} = \langle L, \prec, \{Q^{\mathfrak{M}}_{\langle a, X \rangle} : \langle a, X \rangle \in A \times N\} \rangle$$

From $\mathfrak{M}$ we define a model over the alphabet $A \cup N$, also denoted by $\vec{x}^\times$.

$$(6.54) \qquad \vec{x}^\times := \langle L, \prec, \{Q^{\mathfrak{L}}_a : a \in A\}, \{Q^{\mathfrak{L}}_X : X \in N\} \rangle$$

Here $w \in Q^{\mathfrak{L}}_a$ iff $w \in Q^{\mathfrak{M}}_{\langle a, X \rangle}$ for some X and $w \in Q^{\mathfrak{L}}_X$ iff $w \in Q^{\mathfrak{M}}_{\langle a, X \rangle}$ for some $a \in A$.

Definition 6.14 *Let $G = \langle \Sigma, N, A, R \rangle$ be a regular grammar* and $\varphi \in \mathsf{QML}$ a constant formula (with constants for elements of A). We say, G is **faithful to** φ if there is a subset $H \subseteq N$ such that for every string $\vec{x}$, every $\vec{x}^\times$ and every w: $\langle \vec{x}^\times, w \rangle \vDash \varphi$ iff there exists $X \in H$ with $w \in Q_X$. We say, H **codes** φ **with respect to** G.*

The idea behind this definition is as follows. Given a set H and a formula φ, H codes φ with respect to G if in every derivation of a string $\vec{x}$ φ is true in $\vec{x}^{\times}$ at exactly those nodes where the nonterminal H occurs. The reader may convince himself of the following facts.

Proposition 6.15 *Let G be a regular grammar* and let H code φ and K code χ with respect to G. Then the following holds.*

① *$N - H$ codes $\neg\varphi$ with respect to G.*

② *$H \cap K$ codes $\varphi \wedge \chi$ with respect to G.*

③ *$H \cup K$ codes $\varphi \vee \chi$ with respect to G.*

We shall inductively show that every QML–formula can be coded in a regular grammar* on condition that one suitably extends the original grammar.

Definition 6.16 *Suppose that $G_1 = \langle \Sigma_1, N_1, A, R_1 \rangle$ and $G_2 = \langle \Sigma_2, N_2, A, R_2 \rangle$ are regular grammars*. Then put*

$$G_1 \times G_2 := \langle \Sigma_1 \times \Sigma_2, N_1 \times N_2, A, R_1 \times R_2 \rangle \tag{6.55}$$

where

$$\begin{aligned} R_1 \times R_2 := \quad & \{\langle X_1, X_2 \rangle \to a \, \langle Y_1, Y_2 \rangle : X_i \to aY_i \in R_i\} \\ & \cup \{\langle X_1, X_2 \rangle \to a : X_i \to a \in R_i\} \end{aligned} \tag{6.56}$$

*We call $G_1 \times G_2$ the **product** of the grammars* G_1 and G_2.*

We have

Proposition 6.17 *Let G_1 and G_2 be grammars* over A. Then $L(G_1 \times G_2) = L(G_1) \cap L(G_2)$.*

The following theorem is not hard to show and therefore left as an exercise.

Lemma 6.18 *Let φ be coded in G_2 by H. Then φ is coded in $G_1 \times G_2$ by $N_1 \times H$ and in $G_2 \times G_1$ by $H \times N_1$.*

Definition 6.19 *Let $\varphi \in \mathsf{QML}$. A **code** for φ is a pair $\langle G, H \rangle$ where $L(G) = A^*$ and H codes φ with respect to G. φ is called **codable** if it has a code.*

Assume $\langle G,H\rangle$ is a code for φ and let G' be given. Then we have $L(G'\times G) = L(G')$ and φ is coded in $G'\times G$ by $N'\times H$. Therefore, it suffices to name just one code for every formula. Moreover, the following fact makes life simpler for us.

Lemma 6.20 *Let Δ be a finite set of codable formulae. Then there exists a grammar* G and sets H_φ, $\varphi\in\Delta$, such that $\langle G,H_\varphi\rangle$ is a code of φ.*

Proof. Let $\Delta := \{\delta_i : i<n\}$ and let $\langle G_i,M_i\rangle$ be a code of δ_i, $i<n$. Put $G := \mathsf{X}_{i<n}G_i$ and $H_i := \mathsf{X}_{j<i}N_i\times H_i\times\mathsf{X}_{i<j<n}N_j$. Iterated application of Lemma 6.18 yields the claim. □

Theorem 6.21 (Coding Theorem) *Every constant QML–formula is codable.*

Proof. The proof is by induction over the structure of the formula. We begin with the code of $\underline{a}$, $a\in A$. Define the following grammar* G_a. Put $N := \{X,Y\}$ and $\Sigma := \{X,Y\}$. The rules are

$$X\to a \mid aX \mid aY,\quad Y\to b\mid bX\mid bY \tag{6.57}$$

where b ranges over all elements from $A-\{a\}$. The code is $\langle G_a,\{X\}\rangle$ as one easily checks. The inductive steps for $\neg$, $\wedge$, $\vee$ and $\to$ are covered by Proposition 6.15. Now for the case $\varphi = \langle\prec\rangle\eta$. We assume that η is codable and that $C_\eta = \langle G_\eta,H_\eta\rangle$ is a code. Now we define G_φ. Let $N_\varphi := N_\eta\times\{0,1\}$ and $\Sigma_\varphi := \Sigma_\eta\times\{0,1\}$. Finally, let the rules be of the form

$$\langle X,1\rangle\to a\,\langle Y,0\rangle,\quad \langle X,1\rangle\to a\,\langle Y,1\rangle, \tag{6.58}$$

where $X\to aY\in R_\eta$ and $Y\in H_\eta$ and of the form

$$\langle X,0\rangle\to a\,\langle Y,0\rangle,\quad \langle X,0\rangle\to a\,\langle Y,1\rangle, \tag{6.59}$$

where $X\to aY\in R_\eta$ and $Y\notin H_\eta$. Further, we take all rules of the form

$$\langle X,0\rangle\to a \tag{6.60}$$

for $X\to a\in R_\eta$. One easily checks that for every rule $X\to aY$ or $X\to a$ there is a rule G_φ. The code of φ is now $\langle G_\varphi,N_\eta\times\{1\}\rangle$. Now to the case $\varphi = \langle\succ\rangle\eta$. Again we assume that η is codable and that the code is $C_\eta = \langle G_\eta,H_\eta\rangle$. Now we define G_φ. Let $N_\varphi := N_\eta\times\{0,1\}$, $\Sigma_\varphi := \Sigma_\eta\times\{0\}$. Finally, let R_φ be the set of rules of the form

$$\langle X,0\rangle\to a\,\langle Y,1\rangle,\quad \langle X,1\rangle\to a\,\langle Y,1\rangle, \tag{6.61}$$

where $X \to aY \in R_\eta$ and $X \in H_\eta$ and of the form

(6.62) $\langle X,0\rangle \to a\,\langle Y,0\rangle, \quad \langle X,1\rangle \to a\,\langle Y,0\rangle,$

where $X \to aY \in R_\eta$ and $X \notin H_\eta$; and finally for every rule $X \to a$ we take the rule

(6.63) $\langle X,0\rangle \to a, \quad \langle X,1\rangle \to a$

on board. The code of φ is now $\langle G_\varphi, N_\eta \times \{1\}\rangle$. Now we look at $\varphi = \langle + \rangle \eta$. Again we put $N_\varphi := N_\eta \times \{0,1\}$ as well as $\Sigma_\varphi := \Sigma_\eta \times \{0,1\}$. We take all rules of the form

(6.64) $\langle X,0\rangle \to a\,\langle Y,0\rangle$

where $X \to aY \in R_\eta$. Further, the rules have the form

(6.65) $\langle X,1\rangle \to a\,\langle Y,1\rangle$

where $X \to aY \in R_\eta$ and $Y \notin H_\eta$. Moreover, we take the rules

(6.66) $\langle X,1\rangle \to a\,\langle Y,0\rangle$

for $X \to aY \in R_\eta$ and $Y \in H_\eta$, as well as all rules

(6.67) $\langle X,0\rangle \to a$

where $X \to a \in R_\eta$. The code of φ is then $\langle G_\varphi, N_\eta \times \{1\}\rangle$. Now we look at $\varphi = \langle - \rangle \eta$. Let $N_\varphi := N_\eta \times \{0,1\}$, and $\Sigma_\varphi := \Sigma_\eta \times \{0\}$. The rules are of the form

(6.68) $\langle X,1\rangle \to a\,\langle Y,1\rangle$

for $X \to aY \in R_\eta$. Further there are rules of the form

(6.69) $\langle X,0\rangle \to a\,\langle Y,1\rangle$

for $X \to aY \in R_\eta$ and $X \in H_\eta$. Moreover, we take the rules

(6.70) $\langle X,0\rangle \to a\,\langle Y,0\rangle$

where $X \to aY \in R_\eta$ and $X \notin H_\eta$, and, finally, all rules of the form

(6.71) $\langle X,0\rangle \to a, \qquad \langle X,1\rangle \to a,$

where $X \to a \in R_\eta$. The code of φ is then $\langle G_\varphi, N_\eta \times \{1\}\rangle$. The biggest effort goes into the last case, $\varphi = (\forall p_i)\eta$. To start, we introduce a new alphabet, namely $A \times \{0,1\}$, and a new constant, c. Let $\underline{a} := \underline{\langle a,0\rangle} \vee \underline{\langle a,1\rangle}$. Further, assume that $\mathsf{c} \leftrightarrow \bigvee_{a\in A} \underline{\langle a,1\rangle}$ holds. Then $\underline{\langle a,1\rangle} \leftrightarrow \underline{a} \wedge \mathsf{c}$ and $\underline{\langle a,0\rangle} \leftrightarrow \underline{a} \wedge \neg\mathsf{c}$. Then let $\eta' := \eta[\mathsf{c}/p_i]$. We can apply the inductive hypothesis to this formula. Let Δ be the set of subformulae of η'. For an arbitrary subset $\Sigma \subseteq \Delta$ let

$$L_\Sigma := \bigwedge_{\delta\in\Sigma} \delta \wedge \bigwedge_{\delta\notin\Sigma} \neg\delta \tag{6.72}$$

We can find a grammar* G and for each $\delta \in \Delta$ sets H_δ such that $\langle G, H_\delta\rangle$ codes δ. Hence for every $\Sigma \subseteq \Delta$ there exist H_Σ such that $\langle G, H_\Sigma\rangle$ codes L_Σ. Now we first form the grammar* G^1 with the nonterminals $N \times \wp(\Delta)$ and the alphabet $A \times \{0,1\}$. The set of rules is the set of all

$$\langle X,\Sigma\rangle \to \langle a,i\rangle \quad \langle X',\Sigma'\rangle \tag{6.73}$$

where $X \to aX' \in R$, $X \in H_\Sigma$ and $X' \in H_{\Sigma'}$; further all rules of the form

$$\langle X,\Sigma\rangle \to \langle a,i\rangle \tag{6.74}$$

where $X \to a \in R$ and $X \in H_\Sigma$. Put $H^1_\Sigma := H_\Sigma \times \{\Sigma\}$. Again one easily sees that $\langle G^1, H^1_\Sigma\rangle$ is a code for Σ for every $\Sigma \subseteq \Delta$. We now step over to the grammar* G^2 with $N^2 := N \times \{0,1\} \times \wp(\Delta)$ and $A^2 := A$ as well as all rules

$$\langle X,i,\Sigma\rangle \to a \quad \langle X',i',\Sigma'\rangle \tag{6.75}$$

where $\langle X,\Sigma\rangle \to \langle a,i\rangle \quad \langle X',\Sigma'\rangle \in R^1$ and

$$\langle X,i,\Sigma\rangle \to a \tag{6.76}$$

where $\langle X,\Sigma\rangle \to \langle a,i\rangle \in R^1$. Finally, we define the following grammar*. $N^3 := N \times \wp(\wp(\Delta))$, $A^3 := A$, and let

$$\langle X,\mathbb{A}\rangle \to a \quad \langle Y,\mathbb{B}\rangle \tag{6.77}$$

be a rule iff $\mathbb{B}$ is the set of all Σ' for which $\Sigma \in \mathbb{A}$ and there are $i,i' \in \{0,1\}$ such that

$$\langle X,i,\Sigma\rangle \to a \quad \langle Y,i',\Sigma'\rangle \in R^2 \tag{6.78}$$

Likewise

(6.79) $\langle X, \mathbb{A}\rangle \to a \in R^3$

iff there is a $\Sigma \in \mathbb{A}$ and some $i \in \{0,1\}$ with

(6.80) $\langle X, i, \Sigma\rangle \to a \in R^2$

Put $H_\varphi := \{\Sigma : \eta' \in \Sigma\}$. We claim: $\langle G^3, H_\varphi\rangle$ is a code for φ. For a proof let $\vec{x}$ be a string and let a G^3–derivation of $\vec{x}$ be given. We construct a G^1–derivation. Let $\vec{x} = \prod_{i<n} x_i$. By assumption we have a derivation

(6.81) $\langle X_i, \mathbb{A}_i\rangle \to x_i \quad \langle X_{i+1}, \mathbb{A}_{i+1}\rangle$

for $i < n-1$ and

(6.82) $\langle X_{n-1}, \mathbb{A}_{n-1}\rangle \to x_{n-1}$

By construction there exists a j_{n-1} and a $\Sigma_{n-1} \in \mathbb{A}_{n-1}$ such that

(6.83) $\langle X_{n-1}, j_{n-1}, \Sigma_{n-1}\rangle \to a \in R^2$

Descending we get for every $i < n-1$ a j_i and a Σ_i with

(6.84) $\langle X_i, j_i, \Sigma_i\rangle \to a \quad \langle X_{i+1}, j_{i+1}, \Sigma_{i+1}\rangle \in R^2$

We therefore have a G^2–derivation of $\vec{x}$. From this we immediately get a G^1–derivation. It is over the alphabet $A \times \{0,1\}$. By assumption η' is coded in G^1 by $H_{\eta'}$. Then η' holds in all nodes i with $X_i \in H_{\eta'}$. This is the set of all i with $X_i \in H_\Sigma$ for some $\Sigma \subseteq \Delta$ with $\eta' \subseteq \Delta$. This is exactly the set of all i with $\langle X_i, \mathbb{A}_i\rangle \in H_\varphi$. Hence we have $\langle X_i, \mathbb{A}_i\rangle \in H_\varphi$ iff the Z–structure of $\vec{x}$ satisfies φ in the given G^3–derivation at i. This however had to be shown. Likewise from a G^1–derivation of a string a G^3–derivation can be constructed, as is easily seen. □

Now we are almost done. As our last task we have to show that from the fact that a formula is codable we also get a grammar which only generates strings that satisfy this formula. So let $\Phi \subseteq \mathsf{MSO}$ be finite. We may assume that all members are sentences (if not, we quantify over the free variables with a universal quantifier). By Corollary 6.13 we can assume that in place of MSO–sentences we are dealing with QML–formulae. Further, a finite conjunction of QML–formulae is again a QML–formula so that we are down to

the case where Φ consists of a single QML–formula φ. By Theorem 6.21, φ has a code $\langle G,H\rangle$, with $G = \langle \Sigma,N,A,R\rangle$. Put $G^{\varphi} := \langle \Sigma\cap H, N\cap H, A, R_H\rangle$, where

$$(6.85)\qquad \begin{aligned} R_H := \quad & \{X \to aY \in R : X,Y \in H\} \\ & \cup \{X \to a \in R : X \in H\} \end{aligned}$$

Now there exists a G^{φ}–derivation of $\vec{x}$ iff $\vec{x}^{\times} \vDash \varphi$.

Notes on this section. Some remarks are in order about FOL–definable classes of Z–structures over the signature containing $<$ (!) and $\underline{a}$, $a \in A$. A regular term is $*$**–free** if it does not contain *, but may contain occurrences of $-$, which is a unary operator forming the complement of a language. Then the following are equivalent.

① The class of Z–structures for L are finitely FO–axiomatizable.

② $L = L(t)$ for a $*$–free regular term t.

③ There is a $k \geq 1$ such that for every $\vec{y} \in A^+$ and $\vec{x},\vec{z} \in A^*$: $\vec{x}\vec{y}^k\vec{z} \in L$ iff $\vec{x}\vec{y}^{k+1}\vec{z} \in L$.

See [**?**] and references therein.

Exercise 217. Show that every (!) language is an intersection of regular languages. (This means that we cannot omit the condition of finite axiomatizability in Theorem 6.5.)

Exercise 218. Let Φ be a finite MSO–theory, L the regular language which belongs to Φ. L is recognizable in $O(n)$–time using a finite state automaton. Give upper bounds for the number of states of a minimal automaton recognizing L. Use the proof of codability. Are the derived bounds optimal?

Exercise 219. An MSO–sentence is said to be in Σ^1_1 if it has the form

$$(6.86)\qquad (\exists P_0)(\exists P_1)\cdots(\exists P_{n-1})\varphi(P_0,\ldots,P_{n-1})$$

where φ does not contain second order quantifiers. φ is said to be in Π^1_1 if it has the form $(\forall P_0)(\forall P_1)\cdots(\forall P_{n-1})\varphi(P_0,\ldots,P_{n-1})$ where φ does not contain second order quantifiers. Show the following: *Every MSO–axiomatizable class $\mathcal{K}$ of Z–structures is axiomatizable by a set of Σ^1_1–sentences. If $\mathcal{K}$ is finitely MSO–axiomatizable then it is axiomatizable by finitely many Σ^1_1–sentences.*

3. Categorization and Phonology

In this section we shall deal with syllable structure and phonological rules. We shall look at the way in which discrete entities, known as *phonemes*, arise from a continuum of sounds or phones, and how the mapping between the sound continuum and the discrete space of language particular phonemes is to be construed. The issue is far from resolved; moreover, it seems that it depends on the way we look at language as a whole. Recall that we have assumed sign grammars to be completely additive: there is no possibility to remove something from an exponent that has been put there before. This has a number of repercussions. Linguists often try to define representations such that combinatory processes are additive. If this is taken to be a definition of linguistic processes (as in our definition of compositionality) the organisation of phonology and the phonetics–to–phonology mapping have to have a particular form. We shall discuss a few examples, notably umlaut and final devoicing.

For example, the plural of the German noun `Vater` is `Väter`. How can this be realized in an additive way? First, notice that the plural is not formed from the singular; rather, both forms are derived from an underlying form, the root. Notice right away that the root cannot be a string, it must be a string where at most one vowel is marked for umlaut. (Not all roots will undergo umlaut and if so only one vowel is umlauted!) Technically, we can implement this by writing the root as a string vector: $\mathtt{V}\otimes\mathtt{a}\otimes\mathtt{ter}$. This allows us to restrict our attention to the representation of the vowel alone.

Typically, in grammar books the root is assumed to be just like the singular: $\mathtt{V}\otimes\mathtt{a}\otimes\mathtt{ter}$. Early phonological theory on the other hand would have posited an abstract phoneme in place of `a` or `ä`, a so–called **archiphoneme**. Write `A` for the archiphoneme that is underspecified between `a` and `ä`. Then the root is $\mathtt{V}\otimes\mathtt{A}\otimes\mathtt{ter}$, and the singular adds the specification, say an element `x`, that makes `A` be like `a`, while the plural adds something, say `y` that makes `A` be like `ä`. In other words, in place of `a` and `ä` we have `Ax` and `Ay`.

(6.87) $\quad \text{sing}\colon x\otimes y\otimes z \mapsto z\cdot y\cdot \mathtt{x}\cdot z\colon \mathtt{V}\otimes\mathtt{A}\otimes\mathtt{ter} \mapsto \mathtt{VAxter}$

(6.88) $\quad \text{plur}\colon x\otimes y\otimes z \mapsto z\cdot y\cdot \mathtt{y}\cdot z\colon \mathtt{V}\otimes\mathtt{A}\otimes\mathtt{ter} \mapsto \mathtt{VAyter}$

This solution is additive. Notice, however, that `A` cannot be pronounced, and so the root remains an abstract element. In certain representations, however, `ä` is derived from `a`. Rather than treating the opposition between `a` and `ä` as

equipollent, we may treat it as privative: ä is a plus something else. One specific proposal is that ä differs from a in having the symbol **i** in the i–tier (see [**?**] and references therein). So, rather than writing the vowels using the Latin alphabet, we should write them as sequences indicating the decomposition into primitive elements, and the process becomes literally additive. Notice that the alphabet that we use actually *is* additive. ä differs from a by just two dots — and this is the same with ü and ö. Historically, the dots derive from an 'e' that was written above the vowel to indicate umlaut. (This does not always work; in Finnish ü is written y, blocking for us this cheap way out for Finnish vowel harmony.) Final devoicing could be solved similarly by positing a decomposition of voiced consonants into voiceless consonant plus an abstract voice element (or rather: being voiceless is being voiced plus having a devoicing–feature). All these solutions, however, posit two levels of phonology: a surface phonology and a deep phonology. At the deep level, signs are again additive. This allows us to say that languages are compositional from the deep phonological level onwards.

The most influential model of phonology, by Chomsky and Halle [**?**], is however *not* additive. The model of phonology they favoured — referred to simply as the **SPE–model** — transforms deep structures into surface structures using context sensitive rewrite rules. We may illustrate these rules with German final devoicing. The rule says, roughly, that syllable final consonants (those following the vowel) are voiceless in German. However, as we have noted earlier (in Section 1.3), there is evidence to assume that some consonants are voiced and only become voiceless exactly when they end up in syllable final position. Hence, instead of viewing this as a constraint on the structure of the syllable we may see this as the effect of a rule that devoices consonants. Write + for the syllable boundary. Sidestepping a few difficulties, we may write the rule of final devoicing as follows.

(6.89) $\mathtt{C}[+voiced] \Longrightarrow \mathtt{C}[-voiced]/\_\_[-voiced]\ or\ \_\_+$

(Phonologists write +*voiced* what in attribute–value notation is [VOICED : +].) This says that a consonant preceding a voiceless sound or a syllable boundary becomes voiceless. Using such rules, Chomsky and Halle have formulated a theory of the sound structure of English. This is a Type 1 grammar for English. It has been observed, however, by Ron Kaplan and Martin Kay [**?**] and Kimmo Koskenniemi [**?**] that for all that language really needs the relation between deep level and surface level is a regular relation and can be effected by a finite state transducer. Before we go into the details, we shall

explain something about the general abstraction process in structural linguistics, exemplified here with phonemes, and on syllable structure.

Phonetics is the study of phones (= linguistic sounds) whereas phonology is the study of the phonemes of the languages. We may simply define a phoneme as a set of phones. Different languages group different phones into different phonemes, so that the phonemes of languages are typically not comparable. The grouping into phonemes is far from trivial. A good exposition of the method can be found in [**?**]. We shall look at the process of phonemicization in some detail. Let us assume for simplicity that words or texts are realized as sequences of discrete entities called phones. This is not an innocent assumption: it is for example often not clear whether the sequence [t] plus [ʃ], resulting in an affricate [tʃ], is to be seen as one or as two phones. (One can imagine that this varies from language to language.) Now, denote the set of phones by Σ. A word is not a single sequence of phones, but rather a set of such sequences.

Definition 6.22 *L is a **language*** **over** Σ if L is a subset of $\wp(\Sigma^*)$ such that $\varnothing \notin \Sigma$ and if $W, W' \in L$ and $W \cap W' \neq \varnothing$ then $W = W'$. We call the members of L **words**. $\vec{x} \in W$ is called a **realization** of W. For two sequences $\vec{x}$ and $\vec{y}$ we write $\vec{x} \sim_L \vec{y}$ if they belong to (or realize) the same word.*

One of the aims of phonology is to simplify the alphabet in such a way that words are realized by as few as possible sequences. (That there is only one sequence for each word in the written system is an illusion created by orthographical convention. English orthography often has little connection with actual pronunciation.) We proceed by choosing a new alphabet, P, and a mapping $\pi\colon \Sigma \to P$. The map π induces a partition on Σ. If $\pi(s) = \pi(s')$ we say that s and s' are **allophones**. π induces a mapping of L onto a subset of $\wp(P^*)$ in the following way. For a word W we write $\overline{\pi}[W] := \{\overline{\pi}(\vec{x}) : \vec{x} \in W\}$. Finally, $\pi^*(L) := \{\overline{\pi}[W] : W \in L\}$.

Definition 6.23 *Let $\pi\colon P \to \Sigma$ be a map and $L \subseteq \wp(\Sigma^*)$ be a language*. π is called **discriminating for** L if whenever $W, W' \in L$ are distinct then $\overline{\pi}[W] \cap \overline{\pi}[W'] = \varnothing$.*

Lemma 6.24 *Let $L \subseteq \wp(\Sigma^*)$ be a language* and $\pi\colon \Sigma \to P$. If π is discriminating for L, $\pi^*(L)$ is a language* over P.*

Definition 6.25 *A **phonemicization** of L is a discriminating map $v\colon A \to B$ such that for every discriminating $w\colon A \to C$ we have $|C| \geq |B|$. We call the members of B **phonemes**.*

If phonemes are sets of phones, they are clearly infinite sets. To account for the fact that speakers can manipulate them, we must assume that they are finitely specified. Typically, phonemes are defined by means of articulatory gestures, which tell us (in an effective way) what basic motor program of the vocal organs is associated with what phoneme. For example, English [p] is voiceless. This says that the chords do not vibrate while it is being pronounced. It is further classified as an obstruent. This means that it obstructs the air flow. And thirdly it is classified as a bilabial: it is pronounced by putting the lips together. In English, there is exactly one voiceless bilabial obstruent, so these three features characterize English [p]. In Hindi, however, there are two phonemes with these features, an aspirated and an unaspirated one. (In fact, the actual pronunciation of English [p] for a Hindi speaker oscillates between two different sounds, see the discussion below.) As sounds have to be perceived and classified accordingly, each articulatory gesture is identifiable by an auditory feature that can be read off its spectrum.

The analysis of this sort ends in the establishment of an alphabet P of abstract sounds classes, defined by means of some features, which may either be called articulatory or auditory. (It is not universally agreed that features must be auditory or articulatory. We shall get to that point below.) These can be modeled in the logical language by means of *constants*. For example, the feature +*voiced* corresponds to the constant voiced. Then ¬voiced is the same as being unvoiced.

The features are often interdependent. For example, vowels are always voiced and continuants. In English and German voiceless plosives are typically aspirated, while in French this is not the case; so [t] is pronounced with a subsequent [h]. (In older German books one often finds `Theil` ('part') in place of the modern `Teil`.) The aspiration is lacking when [t] is preceded within the syllable by a sibilant, which in standard German always is [ʃ], for example in `stumpf` [ˈʃʊmpf]. In German, vowels are not simply long or short. Also the vowel quality changes with the length. Long vowels are tense, short vowels are not. The letter `i` is pronounced [ɪ] when it is short and [i:] when it is long (the colon indicates a long sound). (For example, `Sinn` ('sense') [ˈzɪn] as opposed to `Tief` ('deep') [ˈtʰi:f].) Likewise for the other vowels. Table 22 shows the pronunciation of the long and short vowels, drawn from [**?**], Page 87 (written by Klaus Kohler). Only the pairs [a:]/[a] and [ɛ:]/[ɛ] are pronounced in the same way, differing only in length. It is therefore not easy to say which feature is distinctive: is length distinctive in German for vowels, or is it rather the tension? This is interesting in particular when speakers learn a

Table 22. Long and short vowels of German

long	short	long	short
iː	ɪ	yː	ʏ
aː	a	eː	ə
oː	ɔ	øː	œ
uː	ʊ	ɛː	ɛ

new language, because they might be forced to keep distinct two parameters that are cogradient in their own. For example, in Finnish vowels are almost purely distinct in length, there is no cooccurring distinction in tension. If so, tension cannot be used to differentiate a long vowel from a short one. This is a potential source of difficulty for Germans if they want to learn Finnish.

If L is a language* in which every word has exactly one member, L is uniquely defined by the language $L^{\diamond} := \{\vec{x} : \{\vec{x}\} \in L\}$. Let us assume after suitable reductions that we have such a language*; then we may return to studying languages in the customary sense. It might be thought that languages do not possess nontrivial phonemicization maps. This is, however, not so. For example, English has two different sounds, [p] and [p^{h}]. The first occurs after [s], while the second appears for example word initially before a vowel. It turns out that in English [p] and [p^{h}] are not two but one phoneme. To see why, we offer first a combinatorial and then a logical analysis. Recall the definition of a context set. For regular languages it is simply

(6.90) $\quad \mathrm{Cont}_L(a) := \{\langle \vec{x}, \vec{y} \rangle : \vec{x}^\frown a^\frown \vec{y} \in L\}$

If $\mathrm{Cont}_L(a) \cap \mathrm{Cont}_L(a') = \varnothing$, a and a' are said to be in **complementary distribution**. An example is the abovementioned [p] and [p^{h}]. Another example is [x] versus [χ] in German. Both are written `ch`. However, `ch` is pronounced [x] if occurring after [a], [o] and [u], while it is pronounced [ç] if occurring after other vowels and [r], [n] or [l]. Examples are `Licht` [ˈlɪçt], `Nacht` [ˈnaxt], `echt` [ˈeçt] and `Furcht` [ˈfuʁçt]. (If you do not know German, here is a short description of the sounds. [x] is pronounced at the same place as [k] in English, but it is a fricative. [ç] is pronounced at the same place as `y` in English `yacht`, however the tongue is a little higher, that is, closer to the palatum and also the air pressure is somewhat higher, making it sound harder.) Now, from Definition 6.25 we extract the following.

Definition 6.26 *Let A be an alphabet and L a language over A. $\pi\colon A \to B$ is a **pre–phonemicization** if $\overline{\pi}$ is injective on L. $\pi\colon A \twoheadrightarrow B$ is a **phonemicization** if for all pre–phonemicizations $\pi'\colon A \twoheadrightarrow C$, $|C| \geq |B|$.*

The map sending [x] and [ç] to the same sound is a pre–phonemicization in German. However, notice the following. In the language $L_0 := \{\mathtt{aa}, \mathtt{bb}\}$, a and b are in complementary distribution. Nevertheless, the map sending both to the same element is not injective. So, complementary distribution is not enough to make two sounds belong to the same phoneme. We shall see below what is. Second, let $L_1 := \{\mathtt{ac}, \mathtt{bd}\}$. We may either send a and b to e and obtain the language $M_0 := \{\mathtt{ec}, \mathtt{ed}\}$, or we may send c and d to f and obtain the language $M_1 := \{\mathtt{af}, \mathtt{bf}\}$. Both maps are phonemicizations, as is easily checked. So, phonemicizations are not necessarily unique. In order to analyse the situation we have to present a few definitions. The general idea is this. Suppose that A is not minimal for L in the sense that it possesses a non-injective phonemicization. Then there is a pre–phonemicization that conflates exactly two symbols into one. The image M of this map is a regular language again. Now, given the latter we can actually recover for each member of M its preimage under this conflation. What we shall show now is that moreover if L is regular *there is an explicit procedure telling us what the preimage is*. This will be cast in rather abstract terms. We shall define here a modal language that is somewhat different from QML, namely PDL with converse.

The syntax of **propositional dynamic logic** (henceforth PDL) has the usual boolean connectives, the **program connectives** ;, $\cup$, *, further ? and the 'brackets' $[-]$ and $\langle - \rangle$. Further, there is a set Π_0 of **elementary programs**.

① Every propositional variable is a proposition.

② if φ and χ is a proposition, so are $\neg\varphi$, $\varphi \wedge \chi$, $\varphi \vee \chi$, and $\varphi \to \chi$.

③ If φ is a proposition, $\varphi?$ is a program.

④ Every elementary program is a program.

⑤ If α and β are programs, so are $\alpha;\beta$, $\alpha \cup \beta$, and α^*.

⑥ If α is a program and φ a proposition, $[\alpha]\varphi$ and $\langle\alpha\rangle\varphi$ are propositions.

A **Kripke–model** is a triple $\langle F, R, \beta\rangle$, where $R : \Pi_0 \to \wp(F^2)$, and $\beta : \mathrm{PV} \to$

$\wp(F)$. We extend the maps R and β as follows.

$$
\begin{aligned}
\overline{\beta}(\neg\varphi) &:= F - \overline{\beta}(\varphi)\\
\overline{\beta}(\varphi\wedge\chi) &:= \overline{\beta}(\varphi)\cap\overline{\beta}(\chi)\\
\overline{\beta}(\varphi\vee\chi) &:= \overline{\beta}(\varphi)\cup\overline{\beta}(\chi)\\
\overline{\beta}(\varphi\to\chi) &:= (-\overline{\beta}(\varphi))\cup\overline{\beta}(\chi)\\
\overline{R}(\varphi?) &:= \{\langle x,x\rangle : x\in\overline{\beta}(\varphi)\}\\
\overline{R}(\alpha\cup\beta) &:= \overline{R}(\alpha)\cup\overline{R}(\beta)\\
\overline{R}(\alpha;\beta) &:= \overline{R}(\alpha)\circ\overline{R}(\beta)\\
\overline{R}(\alpha^*) &:= \overline{R}(\alpha)^*\\
\overline{\beta}([\alpha]\varphi) &:= \{x : \text{for all } y : \text{if } x\,\overline{R}(\alpha)\,y \text{ then } y\in\overline{\beta}(\varphi)\}\\
\overline{\beta}(\langle\alpha\rangle\varphi) &:= \{x : \text{there is } y : x\,\overline{R}(\alpha)\,y \text{ and } y\in\overline{\beta}(\varphi)\}
\end{aligned}
\tag{6.91}
$$

We write $\langle F,R,\beta\rangle\vDash\varphi$ if $x\in\overline{\beta}(\varphi)$. **Elementary PDL** (**EPDL**) is the fragment of PDL that has no star. The elements of Π_0 are constants; they are like the modalities of modal logic. Obviously, it is possible to add also propositional constants.

In addition to PDL, it also has a program constructor $^\smile$. $\alpha^\smile$ denotes the converse of α. Hence, in a Kripke–frame $\overline{R}(\alpha^\smile)=\overline{R}(\alpha)^\smile$. The axiomatization consists in the axioms for PDL together with the axioms $p\to[\alpha]\langle\alpha^\smile\rangle p$, $p\to[\alpha^\smile]\langle\alpha\rangle p$ for every program α. The term *dynamic logic* will henceforth refer to an extension of $\mathsf{PDL}^\smile$ by some axioms. The fragment without * is called **elementary PDL** with converse, and is denoted by $\mathsf{EPDL}^\smile$. An analog of Büchi's Theorem holds for the logic $\mathsf{PDL}^\smile(\prec)$.

Theorem 6.27 *Let A be a finite alphabet. A class of MZ–structures over A is regular iff it is axiomatizable over the logic of all MZ–structures by means of constant formulae in* $\mathsf{PDL}^\smile(\prec)$ *(with constants for letters from A).*

Proof. By Kleene's Theorem, a regular language is the extension of a regular term. The language of such a term can be written down in $\mathsf{PDL}^\smile$ using a constant formula. Conversely, if γ is a constant $\mathsf{PDL}^\smile(\prec)$–formula it can be rewritten into an MSO–formula. □

The last point perhaps needs reflection. There is a straightforward translation of $\mathsf{PDL}^\smile$ into MSO. We only have to observe that the transitive closure

of an MSO–definable relation is again MSO–definable (see Exercise 216).

$$x\,R^*\,y \quad \Leftrightarrow \quad (\forall X)(X(x) \wedge (\forall z)(\forall z')(X(z) \wedge z\,R\,z'. \\ \rightarrow .X(z')) \rightarrow X(y)) \tag{6.92}$$

Notice also that we can eliminate $^\smile$ from complex programs using the following identities.

$$\overline{R}((\alpha \cup \beta)^\smile) = \overline{R}(\alpha^\smile \cup \beta^\smile) \tag{6.93a}$$

$$\overline{R}((\alpha;\beta)^\smile) = \overline{R}(\beta^\smile;\alpha^\smile) \tag{6.93b}$$

$$\overline{R}((\alpha^*)^\smile) = \overline{R}((\alpha^\smile)^*) \tag{6.93c}$$

$$\overline{R}((\varphi?)^\smile) = \overline{R}(\varphi?) \tag{6.93d}$$

Hence, $\mathsf{PDL}^\smile(\prec)$ can also be seen as an axiomatic extension of $\mathsf{PDL}(\prec;\succ)$ by the axioms $p \rightarrow [\prec]\langle\succ\rangle p$, $p \rightarrow [\succ]\langle\prec\rangle p$. Now let Θ be a dynamic logic. Recall from Section 4.3 the definition of $\Vdash_\Theta$, the global consequence associated with Θ.

Now, we shall assume that we have a language $\mathsf{PDL}^\smile(\prec;D)$, where D is a set of constants. For simplicity, we shall assume that for each letter $a \in A$, D contains a constant $\underline{a}$. However, there may be additional constants. It is those constants that we shall investigate here. We shall show (i) that these constants can be eliminated in an explicit way, (ii) that one can always add constants such that A can be be described purely by contact rules.

Definition 6.28 *Let Θ be a dynamic logic and $\varphi(q)$ a formula. $\varphi(q)$* ***globally implicitly defines*** *q* ***in*** *Θ if $\varphi(q);\varphi(q') \Vdash_\Theta q \leftrightarrow q'$.*

Features (or constants, for that matter) that are implicitly defined are called **inessential**. Here the leading idea is that an inessential feature does not constitute a distinctive phonemic feature, because removing the distinction that this feature induces on the alphabet turns out to induce an injective map. Formally, this is spelled out as follows. Let $A \times \{0,1\}$ be an alphabet, and assume that the second component indicates the value of the feature c. Let $\pi\colon A \times \{0,1\} \rightarrow A$ be the projection onto the first factor. Suppose that the language L can be axiomatized by the constant formula $\varphi(\mathsf{c})$. $\varphi(\mathsf{c})$ defines c implicitly if $\overline{\pi}\colon L' \rightarrow L$ is injective. This in turn means that the map π is a pre–phonemicization. For in principle we could do without the feature. Yet, it is not clear that we can simply eliminate it. In $\mathsf{PDL}^\smile \oplus \varphi(\mathsf{c})$ we call c **eliminable** if there is a formula χ provably equivalent to $\varphi(\mathsf{c})$ that uses only the

constants of φ without c. In the present case, however, an inessential feature is also eliminable. Notice first of all that a regular language over an alphabet B is definable by means a constant formula over the logic of all strings, with constants $\underline{b}$ for every element b of B. By Lemma 6.31, it is therefore enough to show the claim for the logic of all strings. Moreover, by a suitable replacement of other variables by new constants we may reduce the problem to the case where p is the only variable occurring in the formula. Now the language L is regular over the alphabet $A \times \{0,1\}$. Therefore, $\overline{\pi}[L]$ is regular as well. This means that it can be axiomatized using a formula without the constant c. However, this only means that we can make the representation of words more compact. Ideally, we also wish to describe for given $a \in A$, in which context we find $\langle a, 0\rangle$ (an a lacking c) and in which context we find $\langle a, 1\rangle$ (an a having c). This can be done. Let $\mathfrak{A} = \langle A, Q, q_0, F, \delta\rangle$ be a finite state automaton. Then $L_{\mathfrak{A}}(q) := \{\vec{x} : q_0 \xrightarrow{\vec{x}} q\}$ is a regular language (for $L_{\mathfrak{A}}(q) = L(\langle A, Q, q_0, \{q\}, \delta\rangle)$, and the latter is a finite state automaton). Furthermore, $A^* = \bigcup_{q \in Q} L_{\mathfrak{A}}(q)$. If $\mathfrak{A}$ is deterministic, then $L_{\mathfrak{A}}(q) \cap L_{\mathfrak{A}}(q') = \varnothing$ whenever $q \neq q'$. Now, let $\mathfrak{B}$ be a deterministic finite state automaton over $A \times \{0,1\}$ such that $\vec{x} \in L(\mathfrak{B})$ iff $\vec{x} \vDash \varphi(\mathsf{c})$. Suppose we have a constraint χ, where χ is a constant formula.

Definition 6.29 *The* ***Fisher–Ladner closure*** *of* χ*,* $\mathrm{FL}(\chi)$*, is defined as follows.*

(6.94a) $\mathrm{FL}(p_i) := \{p_i\}$

(6.94b) $\mathrm{FL}(\gamma) := \{\gamma\}$

(6.94c) $\mathrm{FL}(\chi \wedge \chi') := \{\chi \wedge \chi'\} \cup \mathrm{FL}(\chi) \cup \mathrm{FL}(\chi')$

(6.94d) $\mathrm{FL}(\langle \alpha \cup \beta\rangle\chi) := \{\langle \alpha \cup \beta\rangle\chi\} \cup \mathrm{FL}(\langle \alpha\rangle\chi) \cup \mathrm{FL}(\langle \beta\rangle\chi)$

(6.94e) $\mathrm{FL}(\langle \alpha;\beta\rangle\chi) := \{\langle \alpha;\beta\rangle\chi\} \cup \mathrm{FL}(\langle \alpha\rangle\langle \beta\rangle\chi)$

(6.94f) $\mathrm{FL}(\langle \alpha^*\rangle\chi) := \{\langle \alpha^*\rangle\chi\} \cup \mathrm{FL}(\langle \alpha\rangle\langle \alpha^*\rangle\chi) \cup \mathrm{FL}(\chi)$

(6.94g) $\mathrm{FL}(\langle \varphi?\rangle\chi) := \{\langle \varphi?\rangle\chi\} \cup \mathrm{FL}(\varphi) \cup \mathrm{FL}(\chi)$

(6.94h) $\mathrm{FL}(\langle \alpha\rangle\chi) := \{\langle \alpha\rangle\chi\} \cup \mathrm{FL}(\chi) \qquad \alpha$ *basic*

The Fisher–Ladner closure covers only $\mathsf{PDL}(\prec;\succ)$–formulae, but this is actually enough for our purposes. For each formula σ in the Fisher–Ladner closure of χ we introduce a constant $c(\sigma)$. In addition, we add the following

axioms.

$$
\begin{aligned}
c(\neg\sigma) &\leftrightarrow \neg c(\sigma)\\
c(\sigma\wedge\tau) &\leftrightarrow c(\sigma)\wedge c(\tau)\\
c(\langle\varphi?\rangle\sigma) &\leftrightarrow c(\varphi)\wedge c(\sigma)\\
c(\langle\alpha\cup\beta\rangle\sigma) &\leftrightarrow c(\langle\alpha\rangle\sigma)\vee c(\langle\beta\rangle\sigma)\\
c(\langle\alpha;\beta\rangle\sigma) &\leftrightarrow c(\langle\alpha\rangle\langle\beta\rangle\sigma)\\
c(\langle\alpha^*\rangle\sigma) &\leftrightarrow c(\sigma)\vee c(\langle\alpha\rangle\langle\alpha^*\rangle\sigma)\\
c(\langle\prec\rangle\sigma) &\leftrightarrow \langle\prec\rangle c(\sigma)\\
c(\langle\succ\rangle\sigma) &\leftrightarrow \langle\succ\rangle c(\sigma)
\end{aligned}
\tag{6.95}
$$

We call these formulae **cooccurrence restrictions**. After the introduction of these formulae as axioms $\sigma\leftrightarrow c(\sigma)$ is provable for every $\sigma\in\mathrm{FL}(\chi)$. In particular, $\chi\leftrightarrow c(\chi)$ is provable. This means that we can eliminate χ in favour of $c(\chi)$. The formulae that we have just added do not contain any of $?$, $\cup$, $^\smile$, * or ;. We only have the most simple axioms, stating that some constant is true before or after another. Now we construct the following automaton. Let ϑ be a subset of $\mathrm{FL}(\chi)$. Then put

$$
q_\vartheta := \bigwedge_{\gamma\in\vartheta} c(\gamma)\wedge\bigwedge_{\gamma\notin\vartheta}\neg c(\gamma)
\tag{6.96}
$$

Now let Q be the set of all consistent q_ϑ. Furthermore, put $q_\vartheta\xrightarrow{a}q_\eta$ iff $q_\vartheta\wedge\langle\prec;\underline{a}?\rangle q_\eta$ is consistent. Let $F:=\{q_\vartheta:[\prec]\bot\in\vartheta\}$ and $B:=\{q_\vartheta:[\succ]\bot\in\vartheta\}$. For every $b\in B$, $\langle A,Q,b,F,\delta\rangle$ is a finite state automaton. Then

$$
L:=\bigcup_{b\in B}L(\langle A,Q,b,F,\delta\rangle)
\tag{6.97}
$$

is a regular language. It immediately follows that the automaton above is well–defined and for every subformula α of χ the set of positions i such that $\langle\vec{x},i\rangle\vDash\alpha$ is uniquely fixed. Hence, for every $\vec{x}$ there exists exactly one accepting run of the automaton. $\langle\vec{x},i\rangle\vDash\psi$ iff ψ holds at the ith position of the accepting run.

We shall apply this to our problem. Let $\varphi(\mathsf{c})$ be an implicit definition of c. Construct the automaton $\mathfrak{A}(\varphi(\mathsf{c}))$ for $\varphi(\mathsf{c})$ as just shown, and lump together all states that do not contain $c(\varphi(\mathsf{c}))$ into a single state q' and put $q'\xrightarrow{a}q'$ for every a. All states different from q' are accepting. This defines the automaton $\mathfrak{B}$. Now let $C:=\{q_\vartheta:\mathsf{c}\in\vartheta\}$. The language $\bigcup_{c\in C}L_{\mathfrak{B}}(q)$ is regular, and it possesses a description in terms of the constants $\underline{a}$, $a\in A$, alone.

Definition 6.30 *Let Θ be a logic and $\varphi(q)$ a formula. Further, let δ be a formula not containing q. We say that δ* ***globally explicitly defines*** *q* ***in*** *Θ* ***with respect to*** *φ if $\varphi(q) \Vdash_\Theta \delta \leftrightarrow q$.*

Obviously, if δ globally explicitly defines q with respect to $\varphi(q)$ then $\varphi(q)$ globally implicitly defines q. On the other hand, if $\varphi(q)$ globally implicitly defines q then it is not necessarily the case that there is an explicit definition for it. It very much depends on the logic in addition to the formula whether there is. A logic is said to have the **global Beth–property** if for any global implicit definition there is a global explicit definition. Now suppose that we have a formula φ implicitly defining q. Suppose further that δ is an explicit definition. Then the following is valid.

(6.98) $\Vdash_\Theta \varphi(q) \leftrightarrow \varphi(\delta)$

The logic $\Theta \oplus \varphi$ defined by adding the formula φ as an axiom to Θ can therefore equally well be axiomatized by $\Theta \oplus \varphi(\delta)$. The following is relatively easy to show.

Lemma 6.31 *Let Θ be a modal logic, and γ a constant formula. Suppose that Θ has the global Beth–property. Then $\Theta \oplus \gamma$ also has the global Beth–property.*

Theorem 6.32 *Every logic of a regular string language has the global Beth–property.*

If the axiomatization is infinite, by the described procedure we get an infinite array of formulae. This does not have a regular solution in general, as the reader is asked to show in the exercises.

The procedure of phonemicization is inverse to the procedure of adding features that we have looked at in the previous section. We shall briefly look at this procedure from a phonological point of view. Assume that we have an alphabet A of phonemes, containing also the syllable boundary marker + and the word boundary marker #. These are not brackets, they are separators. Since a word boundary is also a syllable boundary, no extra marking of the syllable is done at the word boundary. Let us now ask what are the rules of syllable and word structure in a language. The minimal assumption is that any combination of phonemes may form a syllable. This turns out to be false. Syllables are in fact constrained by a number of (partly language dependent)

principles. This can partly be explained by the fact that vocal tract has a certain physiognomy that discourages certain phoneme combinations while it enhances others. These properties also lead to a deformation of sounds in contact, which is called **sandhi**, a term borrowed from Sanskrit grammar. A particular example of sandhi is assimilation ([np] > [mp]). Sandhi rules exist in nearly all languages, but the scope and character varies greatly. Here, we shall call *sandhi* any constraint that is posed on the occurrence of two phonemes (or sounds) next to each other. Sandhi rules are 2–templates in the sense of the following definition.

Definition 6.33 *Let A be an alphabet. An n–**template over** A (or **template of length** n) is a cartesian product of length n of subsets of A. A language L is an n–**template language** if there is a finite set $\mathcal{P}$ of length n such that L is the set of words $\vec{x}$ such that every subword of length n belongs to at least one template from $\mathcal{P}$. L is a **template language** if there is an n such that L is an n–template language.*

Obviously, an n–template language is an $n+1$–template language. Furthermore, 1–template languages have the form B^* where $B \subseteq A$. So the first really interesting class is that of the 2–template languages. It is clear that if the alphabet is finite, we may actually define an n–template to be just a member of A^n. Hence, a template language is defined by naming all those sequences of bounded length that are allowed to occur.

Proposition 6.34 *A language is a template language iff its class of A–strings is axiomatizable by finitely many positive EPDL–formulae.*

To make this more realistic we shall allow also boundary templates. Namely, we shall allow a set $\mathcal{P}^-$ of left edge templates and a set $\mathcal{P}^+$ of right edge templates. $\mathcal{P}^-$ lists the admissible n–prefixes of a word and $\mathcal{P}^+$ the admissible n–suffixes. Call such languages **boundary template languages**. Notice that phonological processes are conditioned by certain boundaries, but we have added the boundary markers to the alphabet. This effectively eliminates the need for boundary templates in the description here. We have not explored the question what would happen if they were eliminated from the alphabet.

Proposition 6.35 *A language is a boundary template language iff its class of A–strings is axiomatizable by finitely many EPDL–formulae.*

It follows from Theorem 6.5 that template languages are regular (which is easy to prove anyhow). However, the language $\mathtt{ca}^+\mathtt{c} \cup \mathtt{da}^+\mathtt{d}$ is regular but not a template language.

The set of templates effectively names the legal transitions of an automaton that uses the alphabet A itself as the set of states to recognize the language. We shall define this notion, using a slightly different concept here, namely that of a **partial finite state automaton**. This is a quintuple $\mathfrak{A} = \langle A, Q, I, F, \delta \rangle$, such that A is the **input alphabet**, Q the set of internal states, I the set of initial states, F the set of accepting states and $\delta : A \times Q \xrightarrow{p} Q$ a partial function. $\mathfrak{A}$ **accepts** $\vec{x}$ if there is a computation from some $q \in I$ to some $q' \in F$ with $\vec{x}$ as input. $\mathfrak{A}$ is a **2–template language** if $Q = A$ and $\delta(a,b)$ is either undefined or $\delta(a,b) = b$.

The reason for concentrating on 2–template languages is the philosophy of naturalness. Basically, grammars are natural if the nonterminal symbols can be identified with terminal symbols, that is, for every nonterminal X there is a terminal a such that for every X–string $\vec{x}$ we have $\mathrm{Cont}_L(\vec{x}) = \mathrm{Cont}_L(a)$. For a regular grammar this means in essence that a string beginning with a has the same distribution as the letter a itself. A moment's reflection reveals that this is the same as the property of being 2–template. Notice that the 2–template property of words and syllables was motivated from the nature of the articulatory organs, and we have described a parser that recognizes whether something is a syllable or a word. Although it seems prima facie plausible that there are also auditory constraints on phoneme sequences we know of no plausible constraint that could illustrate it. We shall therefore concentrate on the former. What we shall now show is that syllables are not 2–template. This will motivate either adding structure or adding more features to the description of syllables. These features are necessarily nonphonemic.

We shall show that nonphonemic features exist by looking at syllable structure. It is not possible to outline a general theory of syllable structure. However, the following sketch may be given (see [**?**]). The sounds are aligned into a so–called **sonoricity hierarchy**, which is shown in Table 23 (vd. = voiced, vl. = voiceless). The syllable is organized as follows.

> *Syllable Structure.* Within a syllable the sonoricity increases monotonically and then decreases.

This means that a syllable must contain at least one sound which is at least as sonorous as all the others in the syllable. It is called the **sonoricity peak**. We shall make the following assumption that will simplify the discussion.

> *Sonoricity Peak.* The sonoricity peak can be constituted by vowels only.

Table 23. The Sonoricity Hierarchy

dark vowels	> mid vowels	> high vowels
[a], [o]	[æ], [œ]	[i], [y]
> r–sounds	> nasals; laterals	> vd. fricatives
[r]	[m], [n]; [l]	[z], [ʒ]
> vd. plosives	> vl. fricatives	> vl. plosives
[b], [d]	[s], [ʃ]	[p], [t]

This wrongly excludes the syllable [krk], or [dn]. The latter is heard in the German `verschwinden` ('to disappear') [ˌfɛʁˈʃwɪndn]. (The second `e` that appears in writing is hardly ever pronounced.) However, even if the assumption is relaxed, the problem that we shall address will remain.

The question is: how can we implement these constraints? There are basically two ways of doing that. (a) We state them by means of $\mathsf{PDL}^{\smile}$–formulae. This is the descriptive approach. (b) We code them. This means that we add some features in such a way that the resulting restrictions become specifiable by 2–templates. The second approach has some motivation as well. The added features can be identified as states of a productive (or analytic) device. Thus, while the solution under (a) tells us what the constraint actually is, the approach under (b) gives us features which we can identify as (sets of) states of a (finite state) machine that actually parses or produces these structures. That this can be done is expressed in the following corollary of the Coding Theorem.

Theorem 6.36 *Any regular language is the homomorphic image of a boundary 2–template language.*

So, we only need to add features. Phonological string languages are regular, so this method can be applied. Let us see how we can find a 2–template solution for the sonoricity hierarchy. We introduce a feature α and its negation $-\alpha$. We start with the alphabet P, and let $C \subseteq P$ be the set of consonants. The new alphabet is

(6.99) $\quad \Xi := P \times \{-\alpha\} \cup C \times \{\alpha\}$

Let $\mathrm{son}(a)$ be the sonoricity of a. (It is some number such that the facts of Table 23 fall out.)

$$
\begin{aligned}
\nabla := \quad & \{\langle\langle a,\alpha\rangle,\langle a',\alpha\rangle\rangle : \mathrm{son}(a) \leq \mathrm{son}(a')\} \\
& \cup \{\langle\langle a,-\alpha\rangle,\langle a',-\alpha\rangle\rangle : \mathrm{son}(a) \geq \mathrm{son}(a')\} \\
& \cup \{\langle\langle a,\alpha\rangle,\langle a',-\alpha\rangle\rangle : a' \notin C, \mathrm{son}(a) \leq \mathrm{son}(a')\} \\
& \cup \{\langle\langle a,-\alpha\rangle,\langle a',\alpha'\rangle\rangle : a \in \{\mathtt{+},\mathtt{\#}\}\}
\end{aligned} \tag{6.100}
$$

As things are defined, any subword of a word is in the language. We need to mark the beginning and the end of a sequence in a special way, as described above. This detail shall be ignored here.

α has a clear phonetic interpretation: it signals the rise of the sonoricity. It has a natural correlate in what de Saussure calls 'explosive articulation'. A phoneme carrying α is pronounced with explosive articulation, a phoneme carrying $-\alpha$ is pronounced with 'implosive articulation'. (See [**?**].) So, α actually has an articulatory (and an auditory) correlate. But it is a nonphonemic feature; it has been introduced in addition to the phonemic features in order to constrain the choice of the next phoneme. As de Saussure remarks, it makes the explicit marking of the syllable boundary unnecessary. The syllable boundary is exactly where the implosive articulation changes to explosive articulation. However, some linguists (for example van der Hulst in [**?**]) have provided a completely different answer. For them, a syllable is structured in the following way.

(6.101) [onset [nucleus coda]]

So, the grammar that generates the phonological strings is actually not a regular grammar but context free (though it makes only very limited use of phrase structure rules). α marks the onset, while $-\alpha$ marks the nucleus together with the coda (which is also called **rhyme**). So, we have three possible ways to arrive at the constraint for the syllable structure: we postulate an axiom, we introduce a new feature, or we assume more structure.

We shall finally return to the question of spelling out the relation between deep and surface phonological representations. We describe here the simplest kind of a machine that transforms strings into strings, the *finite state transducer*.

Definition 6.37 *Let A and B be alphabets. A **(partial) finite state transducer from** A **to** B is a sextuple $\mathfrak{T} = \langle A, B, Q, i_0, F, \delta\rangle$ such that $i_0 \in Q$, $F \subseteq Q$ and*

*$\delta\colon Q \times A_\varepsilon \to \wp(Q \times B^*)$ where $\delta(q,\vec{x})$ is always finite for every $\vec{x} \in A_\varepsilon$. Q is called the set of **states**, i_0 is called the **initial state**, F the set of **accepting states** and δ the **transition function**. $\mathfrak{T}$ is called **deterministic** if $\delta(q,a)$ contains at most one element for every $q \in Q$ and every $a \in A$.*

We call A the **input alphabet** and B the **output alphabet**. The transducer differs from a finite automaton in the transition function. This function does not only say into which state the automaton may change but also what symbol(s) it will output on going into that state. Notice that the transducer may also output an empty string and that it allows for empty transitions. These are not eliminable (as they would be in the finite state automaton) since the machine may accompany the change in state by a nontrivial output. We write

$$q \xrightarrow{\vec{x}:\vec{y}} q' \tag{6.102}$$

if the transducer changes from state q with input $\vec{x}$ ($\in A^*$) into the state q' and outputs the string $\vec{y}$ ($\in B^*$). This is defined as follows.

$$q \xrightarrow{\vec{x}:\vec{y}} q', \quad \text{if} \begin{cases} & (q',\vec{y}) \in \delta(q,\vec{x}) \\ \text{or} & \text{for some } q'',\vec{u},\vec{u}_1,\vec{v},\vec{v}_1 : \\ & q \xrightarrow{\vec{u}:\vec{v}} q'' \xrightarrow{\vec{u}_1:\vec{v}_1} q' \\ & \text{and } \vec{x} = \vec{u}^\frown\vec{u}_1, \vec{y} = \vec{v}^\frown\vec{v}_1. \end{cases} \tag{6.103}$$

Finally one defines

$$L(\mathfrak{T}) := \{\langle \vec{x},\vec{y}\rangle : \text{ there is } q \in F \text{ with } i_0 \xrightarrow{\vec{x}:\vec{y}} q\} \tag{6.104}$$

Transducers can be used to describe the effect of rules. One can write, for example, a transducer $\mathfrak{Syl}$ that syllabifies a given input according to the constraints on syllable structure. Its input alphabet is $A \cup \{\texttt{+},\texttt{\#}\}$, where A is the set of phonemes, `+` the word boundary and `#` the syllable boundary. The output alphabet is $A \times \{\texttt{o},\texttt{n},\texttt{c}\} \cup \{\texttt{+},\texttt{\#}\}$. Here, `o` stands for 'onset', `n` for 'nucleus,' and `c` for 'coda'. The machine annotates each phoneme stating whether it belongs to the onset of a syllable, to its nucleus or its coda. Additionally, the machine inserts a syllable boundary wherever necessary. (So, one may leave the input partially or entirely unspecified for the syllable boundaries. The machine will look which syllable segmentation can or must be introduced.)

Now we write a machine $\mathfrak{AVh}$ which simulates the actions of final devoicing. It has one state, i_0, it is deterministic and the transition function consists in $\langle[\mathrm{b}],\mathtt{c}\rangle:\langle[\mathrm{p}],\mathtt{c}\rangle$, $\langle[\mathrm{d}],\mathtt{c}\rangle:\langle[\mathrm{t}],\mathtt{c}\rangle$, $\langle[\mathrm{g}],\mathtt{c}\rangle:\langle[\mathrm{k}],\mathtt{c}\rangle$ as well as $\langle[\mathrm{z}],\mathtt{c}\rangle:\langle[\mathrm{s}],\mathtt{c}\rangle$ and $\langle[\mathrm{v}],\mathtt{c}\rangle:\langle[\mathrm{f}],\mathtt{c}\rangle$. Everywhere else we have $\langle P,\alpha\rangle:\langle P,\alpha\rangle$, P a phoneme, $\alpha\in\{\mathtt{a},\mathtt{c},\mathtt{n}\}$.

The success of the construction is guaranteed by a general theorem known as the *Transducer Theorem*. It says that the image under transduction of a regular language is again a regular language. The proof is not hard. First, by adding some states, we can replace the function $\delta\colon Q\times A_\varepsilon\to\wp(Q\times B^*)$ by a function $\delta^\diamond\colon Q^\diamond\times A_\varepsilon\to\wp(Q^\diamond\times B_\varepsilon)$ for some set $Q^\diamond$. The details of this construction are left to the reader. Next we replace this function by the function $\delta^2\colon Q\times A_\varepsilon\times B_\varepsilon\to\wp(Q)$. What we now have is an automaton over the alphabet $A_\varepsilon\times B_\varepsilon$. We now take over the notation from the Section 5.3 and write $\vec{x}\otimes\vec{y}$ for the pair consisting of $\vec{x}$ and $\vec{y}$. We define

(6.105) $$(\vec{u}\otimes\vec{v})^\frown(\vec{w}\otimes\vec{x}):=(\vec{u}^\frown\vec{w})\otimes(\vec{v}^\frown\vec{x})$$

Definition 6.38 *Let R be a regular term. We define $L^2(R)$ as follows.*

(6.106a) $$L^2(0):=\varnothing$$

(6.106b) $$L^2(\vec{x}\otimes\vec{y}):=\{\vec{x}\otimes\vec{y}\}\qquad(\vec{x}\otimes\vec{y}\in A_\varepsilon\times B_\varepsilon)$$

(6.106c) $$L^2(R\cdot S):=\{\mathfrak{x}^\frown\mathfrak{y}:\mathfrak{x}\in L^2(R),\mathfrak{y}\in L^2(S)\}$$

(6.106d) $$L^2(R\cup S):=L^2(R)\cup L^2(S)$$

(6.106e) $$L^2(R^*):=L^2(R)^*$$

*A **regular relation on** A is a relation of the form $L^2(R)$ for some regular term R.*

Theorem 6.39 *A relation $Z\subseteq A^*\times B^*$ is regular iff there is a finite state transducer $\mathfrak{T}$ such that $L(\mathfrak{T})=Z$.*

This is essentially a consequence of the Kleene's Theorem. In place of the alphabets A we have chosen the alphabet $A_\varepsilon\times B_\varepsilon$. Now observe that the transitions $\varepsilon:\varepsilon$ do not add anything to the language. We can draw a lot of conclusions from this.

Corollary 6.40 (Transducer Theorem) *The following holds.*

① *Regular relations are closed unter intersection and converse.*

② *If $H \subseteq A^*$ is regular so is $H \times B^*$. If $K \subseteq B^*$ is regular so is $A^* \times K$.*

③ *If $Z \subseteq A^* \times B^*$ is a regular relation, so are the projections*

- $\pi_1[Z] := \{\vec{x} : \textit{there is } \vec{y} \textit{ with } \langle \vec{x}, \vec{y} \rangle \in Z\}$,
- $\pi_2[Z] := \{\vec{y} : \textit{there is } \vec{x} \textit{ with } \langle \vec{x}, \vec{y} \rangle \in Z\}$.

④ *If Z is a regular relation and $H \subseteq A^*$ a regular set then $Z[H]$ also is regular.*

$$Z[H] := \{\vec{y} : \textit{there is } \vec{x} \in H \textit{ with } \langle \vec{x}, \vec{y} \rangle \in Z\}$$

One can distinguish two ways of using a transducer. The first is as a machine which checks for a pair of strings whether they stand in a particular regular relation. The second, whether for a given string over the input alphabet there is a string over the output alphabet that stands in the given relation to it. In the first use we can always transform the transducer into a deterministic one that recognizes the same set. In the second case this is impossible. The relation $\{\langle \mathtt{a}, \mathtt{a}^n \rangle : n \in \omega\}$ is regular but there is no deterministic translation algorithm. One easily finds a language in which there is no deterministic algorithm in any of the directions. From the previous results we derive the following consequence.

Corollary 6.41 (Kaplan & Kay) *Let $R \subseteq A^* \times B^*$ and $S \subseteq B^* \times C^*$ be regular relations. Then $R \circ S \subseteq A^* \times C^*$ is regular.*

Proof. By assumption and the previous theorems, both $R \times C^*$ and $A^* \times S$ are regular. Furthermore, $(R \times C^*) \cap (A^* \times S) = \{\langle \vec{x}, \vec{y}, \vec{z} \rangle : \langle \vec{x}, \vec{y} \rangle \in R, \langle \vec{y}, \vec{z} \rangle \in S\}$ is regular, and so is its projection onto $A^* \times B^*$, which is exactly $R \circ S$. □

This theorem is important. It says that the composition of rules which define regular relations defines a regular relation again. Effectively, what distinguishes regular relations from Type 1 grammars is that the latter allow arbitrary iterations of the same process, while the former do not.

Notes on this section. There is every reason to believe that the mapping from phonemes to phones is not constant but context dependent. In particular, final devoicing is believed by some not to be a phonological process, rather, it is the effect of a contextually conditioned change of realization of the voice–feature (see [**?**]). In other words, on the phonological level nothing changes, but the realization of the phonemes is changed, sometimes so radically that they sound like the realization of a different phoneme (though in a

different environment). This simplifies phonological processes at the cost of complicating the realization map.

The idea of eliminating features was formulated in [**?**] and already brought into correspondence with the notion of implicit definability. Concerning long and short vowels, Hungarian is an interesting case. The vowels i, o, ö, u, ü show length contrast alone, while the long and short forms of a and e also differ in lip attitude and/or aperture. Sauvageot noted in [**?**] that Hungarian moved towards a system where length alone is not distinctive. Effectively, it moves to eliminate the feature short.

Exercise 220. Show that for every given string in a language there is a separation into syllables that conforms to the *Syllable Structure* constraint.

Exercise 221. Let $\Pi_0 := \{\zeta_i : i < n\}$ be a finite set of basic programs. Define $M := \{\zeta_i : i < n\} \cup \{\zeta_i^{\smile} : i < n\}$. Show that for every $\mathsf{EPDL}^{\smile}$ formula φ there is a modal formula δ over the set M of modalities such that $\mathsf{PDL}^{\smile} \vdash \delta \leftrightarrow \varphi$. *Remark.* A modal formula is a formula that has no test, and no $\cup$ and ;. Whence it can be seen as a $\mathsf{PDL}^{\smile}$–formula.

Exercise 222. The results of the previous section show that there is a translation $^{\heartsuit}$ of $\mathsf{PDL}^{\smile}(M)$ into $\mathsf{QML}(M)$. Obviously, the problematic symbols are * and $^{\smile}$. With respect to $^{\smile}$ the technique shown above works. Can you suggest a perspicuous translation of $[\alpha^*]\varphi$? *Hint.* $[\alpha^*]\varphi$ holds if φ holds in the smallest set of worlds closed under α–successors containing the current world. This can be expressed in QML rather directly.

Exercise 223. Show that in Theorem 6.32 the assumption of regularity is necessary. *Hint.* For example, show that the logic of $L = \{\mathtt{a}^{2^n}\mathtt{ca}^n : n \in \omega\}$ fails to have the global Beth–property.

Exercise 224. Prove Lemma 6.31.

Exercise 225. One of the aims of historical linguistics is to reconstruct the affiliation of languages, preferrably by reconstructing a parent language for a certain group of languages and showing how the languages of that group developed from that parent language. The success of the reconstruction lies in the establishment of so–called sound correspondences. In the easiest case they take the shape of correspondences between sounds of the various languages. Let us take the Indo–European (I.–E.) languages. The ancestor of this language, called Indo–European, is not known directly to us, if it at all existed. The proof of its existence is — among other — the successful estab-

lishment of such correspondences. Their reliability and range of applicability have given credibility to the hypothesis of its existence. Its sound structure is reconstructed, and is added to the sound correspondences. (We base the correspondence on the written language, viz. transcriptions thereof.)

I–E	Sanskrit	Greek	Latin	(meaning)
gʰermos	gharmaḥ	thermos	formus	‘warm’
ou̯is	aviḥ	ois	ovis	‘sheep’
su̯os	svaḥ	hos	suus	‘his’
septm̥	sapta	hepta	septem	‘seven’
dekᵘm̥	daśa	deka	decem	‘ten’
neu̯os	navaḥ	neos	novus	‘new’
ǵenos	janaḥ	genos	genus	‘gender’
su̯epnos	svapnaḥ	hypnos	somnus	‘sleep’

Some sounds of one language have exact correspondences in another. For example, I.–E. *p corresponds to p across all languages. (The added star indicates a reconstructed entity.) With other sounds the correspondence is not so clear. I.–E. *e and *a become a in Sanskrit. Sanskrit a in fact has multiple correspondences in other languages. Finally, sounds develop differently in different environments. In the onset, I.–E. *s becomes Sanskrit s, but it becomes ḥ at the end of the word. The details need not interest us here. Write a transducer for all sound correspondences displayed here.

Exercise 226. (Continuing the previous exercise.) Let L_i, $i < n$, be languages over alphabets A_i. Show the following: *Suppose R is a regular relation between L_i, $i < n$. Then there is an alphabet P, a proto–language $Q \subseteq P^*$, and regular relations $R_i \subseteq P^* \times A_i^*$, $i < n$, such that (a) for every $\vec{x} \in P$ there is exactly one $\vec{y}$ such that $\vec{x} R_i \vec{y}$ and (b) L_i is the image of P under R_i.*

Exercise 227. Finnish has a phenomenon called *vowel harmony*. There are three kinds of vowels: back vowels ([a], [o], [u], written a, o and u, respectively), front vowels ([æ], [ø], [y], written ä, ö and y, respectively) and neutral vowels ([e], [i], written e and i). The principle is this.

> *Vowel harmony (Finnish).* A word contains not both a back and a front harmonic vowel.

The vowel harmony only goes up to the word boundary. So, it is possible to combine two words with different harmony. Examples are osakeyhtiö ‘share

holder company'. It consists of the back harmonic word `osake` 'share' and the front harmonic word `yhtiö` 'society'. First, give an $\mathsf{PDL}^{\smile}$–definition of strings that satisfy Finnish vowel harmony. It follows that there is a finite automaton that recognizes this language. Construct such an automaton. *Hint.* You may need to explicitly encode the word boundary.

4. Axiomatic Classes II: Exhaustively Ordered Trees

The theorem by Büchi on axiomatic classes of strings has a very interesting analogon for exhaustively ordered trees. We shall prove it here; however, we shall only show those facts that are not proved in a completely similar way. Subsequently, we shall outline the importance of this theorem for syntactic theory. The reader should consult Section 1.4 for notation. Ordered trees are structures over a language that has two binary relation symbols, $\sqsubset$ and $<$. We also take labels from A and N (!) in the form of constants and get the language MSO^b. In this language the set of exhaustively ordered trees is a finitely axiomatizable class of structures. We consider first the postulates. $<$ is transitive and irreflexive, $\uparrow x$ is linear for every x, and there is a largest element, and every subset has a largest and a smallest element with respect to $<$. From this it follows in particular that below an arbitrary element there is a leaf. Here are now the axioms listed in the order just described.

(6.107a) $(\forall xyz)(x < y \wedge y < z. \rightarrow .x < z)$

(6.107b) $(\forall x)\neg(x < x)$

(6.107c) $(\forall xyz)(x < y \wedge x < z. \rightarrow .y < z \vee y = z \vee y > z)$

(6.107d) $(\exists x)(\forall y)(y < x \vee y = x)$

(6.107e) $(\forall P)(\exists x)(\forall y)(P(x) \wedge y < x. \rightarrow .\neg P(y))$

(6.107f) $(\forall P)(\exists x)(\forall y)(P(x) \wedge y < x. \rightarrow .\neg P(y))$

In what is to follow we use the abbreviation $x \leq y := x < y \vee x = y$. Now we shall lay down the axioms for the ordering. $\sqsubset$ is transitive and irreflexive, it is linear on the leaves, and we have $x \sqsubset y$ iff for all leaves u below x and all leaves $v \leq y$ we have $u \sqsubset v$. Finally, there are only finitely many leaves, a fact which we can express by requiring that every set of nodes has a smallest and

a largest member (with respect to $\sqsubset$). We put $b(x) := \neg(\exists y)(y < x)$.

(6.108a) $(\forall xyz)(x \sqsubset y \wedge y \sqsubset z. \rightarrow .x \sqsubset z)$

(6.108b) $(\forall x)\neg(x \sqsubset x)$

(6.108c) $(\forall xy)(b(x) \wedge b(y). \rightarrow .x \sqsubset y \vee x = y \vee y \sqsubset x)$

(6.108d) $(\forall xy)(x \sqsubset y. \leftrightarrow .(\forall uv)(b(u) \wedge u \leq x \wedge b(v) \wedge v \leq y. \rightarrow .u \sqsubset v))$

(6.108e) $(\forall P)\{(\forall x)(P(x) \rightarrow b(x)).$
$\rightarrow . \quad (\exists y)(P(y) \wedge (\forall z)(P(z) \rightarrow \neg(y \sqsubset z)))$
$\wedge (\exists y)(P(y) \wedge (\forall z)(P(z) \rightarrow \neg(z \sqsupset y)))\}$

Thirdly, we must regulate the distribution of the labels.

(6.109a) $(\forall x)(b(x) \leftrightarrow \bigvee\langle \underline{a}(x) : a \in A\rangle)$

(6.109b) $(\forall x)(\neg b(x) \rightarrow \bigvee\langle \underline{A}(x) : A \in N\rangle)$

(6.109c) $(\forall x)(\bigwedge\langle \underline{\alpha}(x) \rightarrow \neg\underline{\beta}(x) : \alpha \neq \beta\rangle)$

The fact that a tree is exhaustively ordered is described by the following formula.

(6.110) $(\forall xy)(\neg(x \leq y \vee y \leq x). \rightarrow .x \sqsubset y \vee y \sqsubset x)$

Proposition 6.42 *The following are finitely* MSO^b*–axiomatizable classes.*

① *The class of ordered trees.*

② *The class of finite exhaustively ordered trees.*

Likewise we can define a quantified modal language. However, we shall change the base as follows, using the results of Exercise 23. We assume 8 operators, $M_8 := \{\diamondsuit, \diamondsuit^+, \diamondsuit, \diamondsuit^+, \diamond, \diamond^+, \diamond, \diamond^+\}$, which correspond to the relations $\prec$, $<$, $\succ$, $>$, *immediate left sister of*, *left sister of*, *immediate right sister of*, as well as *right sister of*. These relations are MSO–definable from the original ones, and conversely the original relations can be MSO–defined from the present ones. Let $\mathfrak{T} = \langle T, <, \sqsubset\rangle$ be an exhaustively ordered tree.

Then we define $R\colon M_8 \to \wp(T)$ as follows.

$$
\begin{aligned}
x\,R(\diamondminus^{+})\,y &:= x \sqsubset y \wedge (\exists z)(x \prec z \wedge y \prec z)\\
x\,R(\diamondminus)\,y &:= x\,R(\diamondminus^{+})\,y \wedge \neg(x\,R(\diamondminus^{+}) \circ R(\diamondminus^{+})\,y)\\
x\,R(\diamondplus^{+})\,y &:= x \sqsupset y \wedge (\exists z)(x \prec z \wedge y \prec z)\\
x\,R(\diamondplus)\,y &:= x\,R(\diamondplus^{+})\,y \wedge \neg(x\,R(\diamondplus^{+}) \circ R(\diamondplus^{+})\,y)\\
x\,R(\diamond^{+})\,y &:= x < y\\
x\,R(\diamond)\,y &:= x \prec y\\
x\,R(\diamondtimes^{+})\,y &:= x > y\\
x\,R(\diamondtimes)\,y &:= x \succ y
\end{aligned}
\tag{6.111}
$$

The resulting structure we call $M(\mathfrak{T})$. Now if T as well as R are given, then the relations $\prec$, $\succ$, $<$, $>$, and $\sqsubset$, as well as $\sqsupset$ are definable. First we define $\diamond^{*}\varphi := \varphi \vee \diamond^{+}\varphi$, and likewise for the other relations. Then $R(\diamond^{*}) = \Delta \cup R(\diamond^{+})$.

$$
\begin{aligned}
&\prec = R(\diamond) && \succ = R(\diamondtimes)\\
&< = R(\diamond^{+}) && > = R(\diamondtimes^{+})\\
&\sqsubset = R(\diamond^{*}) \circ R(\diamondminus^{+}) \circ R(\diamondtimes^{*})\\
&\sqsupset = R(\diamond^{*}) \circ R(\diamondplus^{+}) \circ R(\diamondtimes^{*})
\end{aligned}
\tag{6.112}
$$

Analogously, as with the strings we can show that the following properties are axiomatizable: (a) that $R(\diamondplus^{+})$ is transitive and irreflexive with converse relation $R(\diamondminus^{+})$; (b) that $R(\diamondplus^{+})$ is the transitive closure of $R(\diamondplus)$ and $R(\diamondminus^{+})$ the transitive closure of $R(\diamondminus)$. Likewise for $R(\diamond^{+})$ and $R(\diamond)$, $R(\diamondtimes^{+})$ and $R(\diamondtimes)$. With the help of the axiom below we axiomatically capture the condition that $\uparrow x$ is linear:

$$\diamond^{+}p \wedge \diamond^{+}q. \to .\diamond^{+}(p \wedge q) \vee \diamond^{+}(p \wedge \diamond^{+}q) \vee \diamond^{+}(q \wedge \diamond^{+}p) \tag{6.113}$$

The other axioms are more complex. Notice first the following.

Lemma 6.43 *Let $\langle T, <, \sqsubset\rangle$ be an exhaustively ordered tree and $x, y \in T$. Then $x \neq y$ iff (a) $x < y$ or (b) $x > y$ or (c) $x \sqsubset y$ or (d) $x \sqsupset y$.*

Hence the following definitions.

$$\langle\neq\rangle\varphi := \diamond^{+}\varphi \vee \diamondtimes^{+}\varphi \vee \diamond^{*}\diamondminus^{+}\diamondtimes^{*}\varphi \vee \diamond^{*}\diamondplus^{+}\diamondtimes^{*}\varphi \tag{6.114}$$

$$\boxtimes\varphi := \varphi \wedge [\neq]\varphi \tag{6.115}$$

So we add the following set of axioms.

(6.116) $\{\boxtimes\varphi \to \boxminus^{+}\varphi, \boxtimes\varphi \to \boxminus^{+}\varphi, \boxtimes\varphi \to \boxminus^{+}\varphi, \boxtimes\varphi \to \boxminus^{+}\varphi$
$\boxtimes\varphi \to \boxtimes\boxtimes\varphi, \boxtimes\varphi \to \varphi, \varphi \to \boxtimes\neg\boxtimes\neg\varphi\}$

(Most of them are already derivable. The axiom system is therefore not minimal.) These axioms see to it that in a connected structure every node is reachable from any other by means of the basic relations, moreover, that it is reachable in one step using $R(\boxtimes)$. Here we have

(6.117) $R(\boxtimes) = \{\langle x, y\rangle : \textit{there is } z : x \leq z \geq y\}$

Notice that this always holds in a tree and that conversely it follows from the above axioms that $R(\diamondminus^{+})$ possesses a largest element. Now we put

(6.118) $b(\varphi) := \varphi \wedge \boxminus\bot \wedge [\neq]\neg\varphi$

$b(\varphi)$ holds at a node x iff x is a leaf and φ is true exactly at x. Now we can axiomatically capture the conditions that $R(\diamondminus^{+})$ must be linear on the set of leaves.

(6.119) $\boxminus\bot \wedge \langle\neq\rangle b(q). \to .\diamondminus^{+}p \vee \diamondminus^{+}p$

Finally, we have to add axioms which constrain the distribution of the labels. The reader will be able to supply them. A forest is defined here as the disjoint union of trees.

Proposition 6.44 *The class of exhaustively ordered forests is finitely* QML^b*–axiomatisable.*

We already know that QML^b can be embedded into MSO^b. The converse is as usual somewhat difficult. To this end we proceed as in the case of strings. We introduce an analogon of restricted quantifiers. We define functions $\diamondminus$, $\diamondminus^{+}$, $\diamondminus$, $\diamondminus^{+}$, $\diamondminus$, $\diamondminus^{+}$, $\diamondminus$, $\diamondminus^{+}$, as well as $\langle\neq\rangle$ on unary predicates, whose meaning should be self explanatory. For example

(6.120a) $(\diamondminus\varphi)(x) := (\exists y \succ x)\varphi(y)$

(6.120b) $(\diamondminus^{+}\varphi)(x) := (\exists y > x)\varphi(y)$

where $y \notin \mathrm{fr}(\varphi)$. Finally let O be defined by

(6.121) $O(\varphi) := (\forall x)\neg\varphi(x)$

$O(\varphi)$ says that $\varphi(x)$ is nowhere satisfiable. Let P_x be a predicate variable which does not occur in φ. Define $\{P_x/x\}\varphi$ inductively as described in Section 6.2. Let $\gamma(P_x) = \{\beta(x)\}$. Then we have

$$\langle \mathfrak{M}, \gamma, \beta \rangle \vDash \varphi \qquad \Leftrightarrow \qquad \langle \mathfrak{M}, \gamma, \beta \rangle \vDash \{P_x/x\}\varphi \tag{6.122}$$

Therefore put

$$(Ex)\varphi(x) := (\exists P_x)(\neg O(P_x) \wedge O(P_x \wedge \langle \neq \rangle P_x). \rightarrow .\{P_x/x\}\varphi) \tag{6.123}$$

Because of this we have for all exhaustively ordered trees $\mathfrak{T}$

$$\langle \mathfrak{T}, \gamma, \beta \rangle \vDash (\exists x)\varphi \qquad \Leftrightarrow \qquad \langle \mathfrak{T}, \gamma, \beta \rangle \vDash (Ex)\varphi \tag{6.124}$$

Let again $h\colon P \to PV$ be a bijection from the set of predicate variables of MSO^b onto the set of proposition variables or QML^b.

$$\begin{array}{rcl}
(\underline{a}(x))^\diamond := Q^a & & (P(y))^\diamond := h(P) \\
(\diamondplus\varphi)^\diamond := \diamondplus\varphi^\diamond & & (\diamondminus\varphi)^\diamond := \diamondminus\varphi^\diamond \\
(\diamondleftarrow\varphi)^\diamond := \diamondleftarrow\varphi^\diamond & & (\diamondrightarrow\varphi)^\diamond := \diamondrightarrow\varphi^\diamond \\
(\diamondplus^+\varphi)^\diamond := \diamondplus^+\varphi^\diamond & & (\diamondminus^+\varphi)^\diamond := \diamondminus^+\varphi^\diamond \\
(\diamondleftarrow^+\varphi)^\diamond := \diamondleftarrow^+\varphi^\diamond & & (\diamondrightarrow^+\varphi))^\diamond := \diamondleftarrow^+\varphi^\diamond \\
(\neg\varphi)^\diamond := \neg\varphi^\diamond & & (O(\varphi))^\diamond := \boxtimes\neg\varphi^\diamond \\
(\varphi_1 \wedge \varphi_2)^\diamond := \varphi_1^\diamond \wedge \varphi_2^\diamond & & (\varphi_1 \vee \varphi_2)^\diamond := \varphi_1^\diamond \vee \varphi_2^\diamond \\
((\exists P)\varphi)^\diamond := (\exists h(P))\varphi^\diamond & & ((\forall P)\varphi)^\diamond := (\forall h(P))\varphi^\diamond
\end{array} \tag{6.125}$$

Then the desired embedding of MSO^b into QML^b is shown.

Theorem 6.45 *Let φ be an MSO^b–formula with at most one free variable, the object variable x_0. Then there exists a QML^b–formula φ^M such that for all exhaustively ordered trees $\mathfrak{T}$:*

$$\langle \mathfrak{T}, \beta \rangle \vDash \varphi(x_0) \text{ iff } \langle M(\mathfrak{T}), \beta(x_0) \rangle \vDash \varphi(x_0)^M \tag{6.126}$$

Corollary 6.46 *Modulo the identification $\mathfrak{T} \mapsto M(\mathfrak{T})$ MSO^b and QML^b define the same model classes of exhaustively ordered trees. Further: $\mathcal{K}$ is a finitely axiomatizable class of MSO^b–structures iff $M(\mathcal{K})$ is a finitely axiomatizable class of QML^b–structures.*

For the purpose of definition of a code we suspend the difference between terminal and nonterminal symbols.

Definition 6.47 *Let $G = \langle \Sigma, N, A, R \rangle$ be a CFG* and $\varphi \in \mathsf{QML}^b$ a constant formula (with constants over A). We say, G is **faithful for** φ if there is a set $H_\varphi \subseteq N$ such that for every tree $\mathfrak{T}$ and every node $w \in T$: $\langle \mathfrak{T}, w \rangle \vDash \varphi$ iff $\ell(w) \in H_\varphi$. We also say that H_φ **codes** φ **with respect to** G. Let φ be a QML^b–formula and n a natural number. An n–**code for** φ is a pair $\langle G, H \rangle$ such that $L_B(G)$ is the set of all at most n–ary branching, finite, exhaustively ordered trees over $A \cup N$ and H codes φ in G. φ is called n–**codable** if there is an n–code for φ. φ is called **codable** if there is an n–code for φ for every n.*

Notice that for technical reasons we must restrict ourselves to at most n–branching trees since we can otherwise not write down a CFG* as a code. Let $G = \langle \Sigma, N, A, R \rangle$ and $G' = \langle \Sigma', N', A, R' \rangle$ be grammars* over A. The **product** is defined by

$$G \times G' = \langle \Sigma \times \Sigma', N \times N', A, R \times R' \rangle \tag{6.127}$$

where

$$\begin{aligned} R \times R' := \{ \langle X, X' \rangle \to \langle \alpha_0, \alpha'_0 \rangle \cdots \langle \alpha_{n-1}, \alpha'_{n-1} \rangle : \\ X \to \alpha_0 \cdots \alpha_{n-1} \in R, X' \to \alpha'_0 \cdots \alpha'_{n-1} \in R' \} \end{aligned} \tag{6.128}$$

To prove the analogon of the Coding Theorem (6.21) for strings we shall have to use a trick. As one can easily show the direct extension on trees is false since we have also taken the nonterminal symbols as symbols of the language. So we proceed as follows. Let $h\colon N \to N'$ be a map and $\mathfrak{T} = \langle T, <, \sqsubset, \ell \rangle$ a tree with labels in $A \cup N$. Then let $h[\mathfrak{T}] := \langle T, <, \sqsubset, h_A \circ \ell \rangle$ where $h_A \restriction N := h$ and $h_A(a) := a$ for all $a \in A$. Then $h[\mathfrak{T}]$ is called a **projection of** $\mathfrak{T}$. If $\mathcal{K}$ is a class of trees, then let $h[\mathcal{K}] := \{ h[\mathfrak{T}] : \mathfrak{T} \in \mathcal{K} \}$.

Theorem 6.48 (Thatcher & Wright, Doner) *Let A be a terminal alphabet, N a nonterminal alphabet and $n \in \omega$. A class of exhaustively ordered, at most n–branching finite trees over $A \cup N$ is finitely axiomatizable in MSO^b iff it is the projection onto $A \cup N$ of a context free* class of ordered trees over some alphabet.*

Here a class of trees is **context free*** if it is the class of trees generated by some CFG*. Notice that the symbol ε is not problematic as it was for regular

languages. We may look at it as an independent symbol which can be the label of a leaf. However, if this is to be admitted, we must assume that the terminal alphabet may be A_ε and not A. Notice that the union of two context free sets of trees is not necessarily itself context free. (This again is different for regular languages, since the structures did not contain the nonterminal symbols.)

From now on the proof is more or less the same. First one shows the codability of QML^b–formulae. Then one argues as follows. Let $\langle G,H\rangle$ be the code of a formula φ. We restrict the set of symbols (that is, both N as well as A) to H. In this way we get a grammar* which only generates trees that satisfy φ. Finally we define the projection $h\colon H \to A \cup N$ as follows. Put $h(a) := a$, $a \in A$, and $h(Y) := X$ if $L_B(G) \vDash (\forall x)(\underline{Y}(x) \to \underline{X}(x))$. In order for this to be well defined we must therefore have for all $Y \in H$ an $X \in N$ with this property. In this case we call the code **uniform**. Uniform codability follows easily from codability since we can always construct products $G \times G'$ of grammars* so that $G = \langle \Sigma, N, A, R\rangle$ and $L_B(G \times G') \vDash \underline{X}(\langle x,y\rangle)$ iff $L_B(G) \vDash \underline{X}(x)$. The map h is nothing but the projection onto the first component.

Theorem 6.49 *Every constant* QML^b*–formula is uniformly codable.*

Proof. We only deliver a sketch of the proof. We choose an n and show the uniform n–codability. For ease of exposition we illustrate the proof for $n = 2$. For the formulae $\underline{a}(x)$, $a \in A$, and $\underline{Y}(x)$, $Y \in N$, nothing special has to be done. Again, the booleans are easy. There remain the modal operators and the quantifiers. Before we begin we shall introduce a somewhat more convenient notation. As usual we assume that we have a grammar* $G = \langle \Sigma, N, A, R\rangle$ as well as some sets H_η for certain formulae. Now we take the product with a new grammar* and define H_φ. In place of explicit labels we now use the formulae themselves, where η stands for the set of labels from H_η.

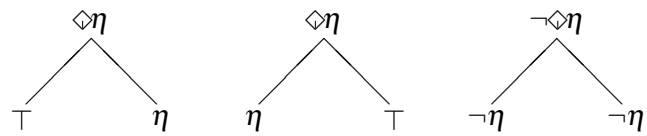
The basic modalities are as follows. Put

$$2 := \langle \{0,1\}, \{0,1\}, A, R_2\rangle \tag{6.129}$$

where R_2 consists of all possible n–branching rules of a grammar in standard form. To code $\diamondplus\eta$, we form the product of G with $\mathbf{2}$. However, we only choose a subset of rules and of the start symbols. Namely, we put $\Sigma' := \Sigma \times \{0,1\}$ and $H'_\eta := H_\eta \times \{0,1\}$, $H'_{\diamondplus\eta} := N \times \{1\}$. The rules are all rules of the

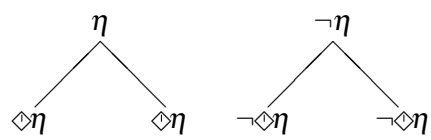
form

(6.130)



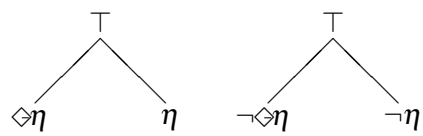
Now we proceed to $\diamondplus\eta$. Here $\Sigma'_{\diamondplus\eta} := N \times \{0\}$.

(6.131)



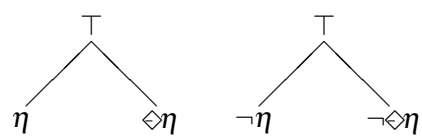
With $\diamondminus\eta$ we choose $\Sigma'_{\diamondminus\eta} := \Sigma \times \{0\}$.

(6.132)



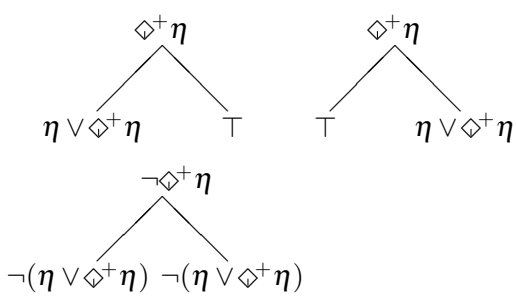
Likewise, $\Sigma'_{\diamondminus\eta}$ is the start symbol of G' in the case of $\diamondminus\eta$.

(6.133)



We proceed to the transitive relations. Notice that on binary branching trees $\diamondplus^{+}\eta \leftrightarrow \diamondplus\eta$ and $\diamondminus^{+}\eta \leftrightarrow \diamondminus\eta$. Now let us look at the relation $\diamondplus^{+}\eta$.

(6.134)



The set of start symbols is $\Sigma \times \{0,1\}$. Next we look at $\diamond^+\eta$.

(6.135)
$$\begin{array}{ccc} & \eta \vee \diamond^+\eta & \\ \diamond^+\eta & & \diamond^+\eta \end{array} \qquad \begin{array}{ccc} & \neg(\eta \vee \diamond^+\eta) & \\ \neg\diamond^+\eta & & \neg\diamond^+\eta \end{array}$$

The set of start symbols is $\Sigma' := \Sigma \times \{0\}$.

Finally we study the quantifier $(\exists p)\eta$. Let $\eta' := \eta[\mathsf{c}/p]$, where c is a new constant. Our terminal alphabet is now $A \times \{0,1\}$, the nonterminal alphabet $N \times \{0,1\}$. We assume that $\langle G^1, H^1_\theta \rangle$ is a uniform code for θ, θ an arbitrary subformula of η'. For every subset Σ of the set Δ of all subformulae of η' we put

(6.136) $$L_\Sigma := \bigwedge_{\theta \in \Sigma} \theta \wedge \bigwedge_{\theta \in \Delta - \Sigma} \neg\theta$$

Then $\langle G^1, H^1_\Sigma \rangle$ is a code for L_Σ where

(6.137) $$H^1_\Sigma := \bigcap_{\theta \in \Sigma} H^1_\theta \cap \bigcap_{\theta \in \Delta - \Sigma} (N - H^1_\theta)$$

Now we build a new CFG*, G^2. Put $N^2 := N \times \{0,1\} \times \wp(N^1)$. The rules of G^2 are all rules of the form

(6.138)
$$\begin{array}{ccc} & \langle X, i, \Sigma \rangle & \\ \langle Y_0, j_0, \Theta_0 \rangle & & \langle Y_1, j_1, \Theta_1 \rangle \end{array}$$

where $\langle X, i \rangle \in H^1_\Sigma$, $\langle Y_0, j_0 \rangle \in H^1_{\Theta_0}$, $\langle Y_1, j_1 \rangle \in H^1_{\Theta_1}$ and $\Sigma \to \Theta_0\Theta_1$ is a rule of G^1. (This in turn is the case if there are X, Y_0 and Y_1 as well as i, j_0 and j_1 such that $\langle X, i \rangle \to \langle Y_0, j_0 \rangle \langle Y_1, j_1 \rangle \in R$.) Likewise for unary rules. Now we go over to the grammar* G^3, with $N^3 := N \times \wp(\wp(N^1))$. Here we take all rules of the form

(6.139)
$$\begin{array}{ccc} & \langle X, \mathbb{A} \rangle & \\ \langle Y_0, \mathbb{B}_0 \rangle & & \langle Y_1, \mathbb{B}_1 \rangle \end{array}$$

where $\mathbb{A}$ is the set of all Σ for which there are Θ_0, Θ_1 and i, j_0, j_1 such that

(6.140)
$$\begin{array}{c} \langle X, i, \Sigma \rangle \\ \diagup \quad \diagdown \\ \langle Y_0, j_0, \Theta_0 \rangle \quad \langle Y_1, j_1, \Theta_1 \rangle \end{array}$$

is a rule of G^2. □

Notes on this section. From complexity theory we know that CFLs, being in **PTIME**, actually possess a description using first order logic plus inflationary fixed point operator. This means that we can describe the set of strings in $L(G)$ for a CFG by means of a formula that uses first order logic plus inflationary fixed points. Since we can assume G to be binary branching and invertible, it suffices to find a constituent analysis of the string. This is a set of subsets of the string, and so of too high complexity. What we need is a first order description of the constituency in terms of the string alone. The exercises describe a way to do this.

Exercise 228. Show the following: $<$ is definable from $\prec$, likewise $>$. Also, trees can be axiomatized alternatively with $\prec$ (or $\succ$). Show furthermore that in ordered trees $\prec$ is uniquely determined from $<$. Give an explicit definition.

Exercise 229. Let $x L y$ if x and y are sisters and $x \sqsubset y$. Show that in ordered trees L can be defined with $\sqsubset$ and conversely.

Exercise 230. Let $\mathfrak{T}$ be a tree over A and N such that every node that is not preterminal is at least 2–branching. Let $\vec{x} = x_0 \cdots x_{n-1}$ be the associated string. Define a set $C \subseteq n^3$ as follows. $\langle i, j, k \rangle \in C$ iff the least node above x_i and x_j is lower than the least node above x_i and x_k. Further, for $X \in N$, define $L_X \subseteq n^2$ by $\langle i, j \rangle \in L_X$ iff the least node above x_i and x_j has label X. Show that C uniquely codes the tree structure $\mathfrak{T}$ and L_X, $X \in N$, the labelling. Finally, for every $a \in A$ we have a unary relation $T_a \subseteq n$ to code the nodes of category a. Axiomatize the trees in terms of the relations C, L_X, $X \in N$, and T_a, $a \in A$.

Exercise 231. Show that a string of length n possesses at most 2^{cn^3} different constituent structures for some constant c.

5. Transformational Grammar

In this section we shall discuss the so–called **Transformational Grammar**, or **TG**. Transformations have been introduced by Zellig Harris. They were operations that change one syntactic structure into another without changing the meaning. The idea to use transformations has been adopted by Noam Chomsky, who developed a very rich theory of transformations. Let us look at a simple example, a phenomenon known as **topicalization**.

(6.141) `Harry likes trains.`

(6.142) `Trains, Harry likes.`

We have two different English sentences, of which the first is in normal serialization, namely SVO, and the second in OSV order. In syntactic jargon we say that in the second sentence the object has been topicalized. (The metaphor is by the way a dynamic one. Speaking statically, one would prefer to express that differently.) The two sentences have different uses and probably also different meanings, but the meaning difference is hard to establish. For the present discussion this is however not really relevant. A transformation is a rule that allows us for example to transform (6.141) into (6.142). Transformations have the form SD $\Longrightarrow$ SC. Here SD stands for **structural description** and SC for **structural change**. The rule **TOP**, for **topicalization**, may be formulated as follows.

(6.143) $\mathrm{NP}_1\ \mathrm{V}\ \mathrm{NP}_2\ \mathrm{Y} \Longrightarrow \mathrm{NP}_2\ \mathrm{NP}_1\ \mathrm{V}\ \mathrm{Y}$

This means the following. If a structure can be decomposed into an NP followed by a V and a second NP followed in turn by an arbitrary string, then the rule may be applied. In that case it moves the second NP to the position immediately to the left of the first NP. Notice that Y is a variable for arbitrary strings while NP and V are variables for constituents of category NP and V, respectively. Since a string can possess several NPs or Vs we must have for every category a denumerable set of variables. Alternatively, we may write $[W]_{\mathrm{NP}}$. This denotes an arbitrary string which is an NP–constituent. We agree that (6.143) can also be applied to a subtree of a tree (just as the string replacement rules of Thue–processes apply to substrings).

Analogously, we may formulate also the reversal of this rule:

(6.144) $\mathrm{NP}_2\ \mathrm{NP}_1\ \mathrm{V}\ \mathrm{Y} \Longrightarrow \mathrm{NP}_1\ \mathrm{V}\ \mathrm{NP}_2\ \mathrm{Y}$

However, one should be extremely careful with such rules. They often turn out to be too restrictive and often also too liberal. Let us look again at **TOP**. As formulated, it cannot be applied to (6.145) and (6.147), even though topicalization is admissible, as (6.146) and (6.148) show.

(6.145) `Harry might like trains.`

(6.146) `Trains, Harry might like.`

(6.147) `Harry certainly likes trains.`

(6.148) `Trains, Harry certainly likes.`

The problem is that in the SD V only stands for the verb, not for the complex consisting of the verb and the auxiliaries. So, we have to change the SD in such a way that it allows the examples above. Further, it must not be disturbed by eventually intervening adverbials.

German exemplifies a construction which is one of the strongest arguments in favour of transformations, namely the so–called V2–phenomenon. In German, the verb is at the end of the clause if that clause is subordinate. In a main clause, however, the part of the verb cluster that carries the inflection is moved to second position in the sentence. Compare the following sentences.

(6.149) `...,daß Hans sein Auto repariert.`
..., that Hans his car repairs.

(6.150) `Hans repariert sein Auto.`
Hans repairs his car.

(6.151) `...,daß Hans nicht in die Oper gehen will.`
..., that Hans not into the opera go wants.

(6.152) `Hans will nicht in die Oper gehen.`
Hans wants not into the opera go.

(6.153) `...,daß Hans im Unterricht selten aufpaßt.`
..., that Hans in class rarely PREF–attention.pay.

(6.154) `Hans paßt im Unterricht selten auf.`
Hans attention.pay in class rarely PREF.

As is readily seen, the auxiliaries and the verb are together in the subordinate clause, in the main clause the last of the series (which carries the inflection) moves into second place. Furthermore, as the last example illustrates, it can

happen that certain prefixes of the verb are left behind when the verb moves. In transformational grammar one speaks of **V2–movement**. This is a transformation that takes the inflection carrier and moves it to second place in a main clause. A similar phenomenon is what might be called **damit**- or **davor–split**, which is found mainly in northern Germany.

(6.155) `Da hat er mich immer vor gewarnt.`
DA has he me always VOR warned.
He has always warned me of that.

(6.156) `Da konnte ich einfach nicht mit rechnen.`
DA could I simply not MIT reckon.
I simply could not reckon with that.

We leave it to the reader to picture the complications that arise when one wants to formulate the transformations when V2–movement and damit- or davor–split may operate. Notice also that the order of application of these rules must be reckoned with.

A big difference between V2–movement and damit–split is that the latter is optional and may apply in subordinate clauses, while the former is obligatory and restricted to main clauses.

(6.157) `Er hat mich immer davor gewarnt.`

(6.158) *`Er mich immer davor gewarnt hat.`

(6.159) `Ich konnte einfach nicht damit rechnen.`

(6.160) *`Ich damit einfach nicht rechnen konnte.`

In (6.158) we have reversed the effect of both transformations of (6.155). The sentence is ungrammatical. If we only apply V2–movement, however, we get (6.157), which is grammatical. Likewise for (6.160) and (6.159). In contrast to Harris, Chomsky did not construe transformations as mediating between grammatical sentences (although also Harris did allow to pass through illegitimate structures). He insisted that there is a two layered process of generation of structures. First, a simple grammar (context free, preferrably) generates so–called **deep structures**. These deep structures may be seen as the canonical representations, like Polish Notation or infix notation, where the meaning can be read off immediately. However, these structures may not be legitimate objects of the language. For example, at deep structure, the verb

of a German sentence appears in final position (where it arguably belongs) but alas these sentences are not grammatical as main clauses. Hence, transformations must apply. Some of them apply optionally, for example damit- and davor–split, some obligatorily, for example V2–movement. At the end of the transformational cycle stands the **surface structure**. The second process is also called (somewhat ambiguously) **derivation**. The split between these two processes has its advantages, as can be seen in the case of German. For if we assume that the main clause is not the deep structure, but derived from a deep structure that looks like a surface subordinate clause, the entire process for generating German sentences is greatly simplified. Some have even proposed that all languages have universally the same deep structure, namely SVO in [**?**]; or right branching, allowing both SVO and SOV deep structure. The latter has been defended in [**?**] (dating from 1991) and [**?**; **?**]. Since the overwhelming majority of languages belongs to either of these types, such claims are not without justification. The differences that can be observed in languages are then caused not by the first process, generating the deep structure, but entirely by the second, the transformational component. However, as might be immediately clear, this is on the one hand theoretically possible but on the other hand difficult to verify empirically. Let us look at a problem. In German, the order of nominal constituents is free (within bounds).

(6.161) `Der Vater schenkt dem Sohn einen Hund.`

(6.162) `Einen Hund schenkt der Vater dem Sohn.`

(6.163) `Dem Sohn schenkt der Vater einen Hund.`

(6.164) `Dem Sohn schenkt einen Hund der Vater.`

The father gives a dog to the son.

How can we decide which of the serializations are generated at deep structure and which ones are not? (It is of course conceivable that all of them are deep structure serializations and even that none of them is.) This question has not found a satisfactory answer to date. The problem is what to choose as a diagnostic tool to identify the deep structure. In the beginning of transformational grammar it was thought that the meaning of a sentence is assigned at deep structure. The transformations are not meaning related, they only serve to make the structure ‘speakable’. This is reminiscent of Harris’ idea that transformations leave the meaning invariant, the only difference being that Harris’ conceived of transformations as mediating between sentences of the

language. Now, if we assume this then different meanings in the sentences suffice to establish that the deep structures of the corresponding sentences are different, though we are still at a loss to say which sentence has which deep structure. Later, however, the original position was given up (on evidence that surface structure did contribute to the meaning in the way that deep structure did) and a new level was introduced, the so–called **Logical Form** (**LF**), which was derived from surface structure by means of further transformations. We shall not go into this, however. Suffice it to say that this increased even more the difficulty in establishing with precision the deep structure(s) from which a given sentence originates.

Let us return to the sentences (6.161) – (6.164). They are certainly not identical. (6.161) sounds more neutral, (6.163) and (6.162) are somewhat marked, and (6.164) finally is somewhat unusual. The sentences also go together with different stress patterns, which increases the problem here somewhat. However, these differences are not exactly semantical, and indeed it is hard to say what they consist in.

Transformational grammar is very powerful. Every recursively enumerable language can be generated by a relatively simple TG. This has been shown by Stanley Peters and R. Ritchie [**?**; **?**]. In the exercises the reader is asked to prove a variant of these theorems. The transformations that we have given above are problematic for a simple reason. The place from which material has been moved is lost. The new structure is actually not distinguishable from the old one. Of course, often we can know what the previous structure was, but only when we know which transformation has been applied. However, it has been observed that the place from which an element has been moved influences the behaviour of the structure. For example, Chomsky has argued that `want to` can be contracted to `wanna` in American English; however, this happens only if no element has been placed between `want` and `to` during the derivation. For example, contraction is permitted in (6.166), in (6.168) however it is not, since `the man` was the subject of the lower infinitive (standing to the left of the verb), and had been raised from there.

(6.165) `We want to leave.`

(6.166) `We wanna leave.`

(6.167) `This is the man we want to leave us alone.`

(6.168) *`This is the man we wanna leave us alone.`

The problem is that the surface structure does not know that the element `the`

man has once been in between want and to. Therefore, one has assumed that the moved element leaves behind a so–called **trace**, written t. For other reasons the trace also got an index, which is a natural number, the same one that is given to the moved element (= antecedent of the trace). So, (6.142) 'really' looks like this.

(6.169) Trains$_1$, Harry likes t_1.

We have chosen the index 1 but any other would have done equally well. The indices as well as the t are not audible, and they are not written either (except in linguistic textbooks, of course). Now the surface structure contains traces and is therefore markedly different from what we actually hear or read. Whence one assumed — finally — that there is a further process turning a surface structure into a pronounceable structure, the so–called **Phonological Form** (**PF**). PF is nothing but the phonological representation of the sentence. On PF there are no traces and no indices, no (or hardly any) constituent brackets.

One of the most important arguments in favour of traces and the instrument of coindexing was the distribution of pronouns. In the theory one distinguishes **referential expressions** (like Harry or the train) from **anaphors**. To the latter belong pronouns (I, you, we) as well as reflexive pronouns (oneself). The distribution of these three is subject to certain rules which are regulated in part by structural criteria.

(6.170) Harry likes himself.

(6.171) Harry believed that John was responsible for himself.

(6.172) Harry believed to be responsible for himself.

Himself is a subject–oriented anaphor. Where it appears, it refers to the subject of the same sentence. Semantically, it is interpreted as a variable which is identical to the variable of the subject. As (6.171) shows, the domain of a reflexive ends with the finite sentence. The antecedent of himself must be taken to be John, not Harry. Otherwise, we would have to have him in place of himself. (6.172) shows that sometimes also phonetically empty pronouns can appear. In other languages they are far more frequent (for example in Latin or in Italian). Subject pronouns may often be omitted. One says that these languages have an empty pronoun, called pro ('little PRO').

Additionally to the question of the subject also structural factors are involved.

(6.173) `Nat was driving his car and Peter, too.`

(6.174) `His car made Nat happy and Peter, too.`

We may understand (6.173) in two ways: either Peter was driving his (= Peter's) car or Nat's car. (6.174) allows only one reading (on condition that 'his' refers to Nat, which it does not have to): Peter was happy about Nat's car, not Peter's. This has arguably nothing to do with semantical factors, but only with the fact that in the first sentence, but not in the second, the pronoun is bound by its antecedent. Binding is defined as follows.

Definition 6.50 *Let $\mathfrak{T}$ be a tree with labelling ℓ. x* ***binds*** *y if (a) the smallest branching node that properly dominates x dominates y, but x does not dominate y, and (b) x and y carry the same index.*

The structural condition (a) of the definition is called **c–command**. (A somewhat modified definition is found below.) The antedecent c–commands the pronoun in case of binding. In (6.173) the pronoun `his` is c–commanded by `Nat`. For the smallest constituent properly containing `Nat` is the entire sentence. In (6.174) the pronoun `his` is not c–commanded by `Nat`. (This is of course not entirely clear and must be argued for independently.)

There is a rule of distribution for pronouns that is as follows: the reflexive pronoun has to be bound by the subject of the sentence. A nonreflexive pronoun however may not be bound by the subject of the sentence. This applies to German as well as to English. Let us look at (6.175).

(6.175) `Himself, Harry likes.`

If this sentence is grammatical, then binding is computed not only at surface structure but at some other level. For the pronoun `himself` is not c–commanded by the subject `Harry`. The structure that is being assumed is [`Himself` [`Harry likes`]]. Such consideration have played a role in the introduction of traces. Notice however that none of the conclusions is inevitable. They are only inevitable moves within a certain theory (because it makes certain assumptions). It has to be said though that binding was *the* central diagnostic tool of transformational grammar. Always if it was diagnosed that there was no c–command relation between an anaphor and some element one has concluded that some movement must have taken place from a position, where c–command still applied.

In the course of time the concept of transformation has undergone revision. TG allowed deletions, but only if they were recoverable: this means that if one has the output structure and the name of the transformation that has been applied one can reconstruct the input structure. (Effectively, this means that the transformations are partial injective functions.) In the so–called **Theory of Government and Binding** (**GB**) Chomsky has banned deletion altogether from the list of options. The only admissible transformation was movement, which was later understood as copy and delete (which in effect had the same result but was theoretically a bit more elegant). The movement transformation was called **Move**–α and allowed to move any element anywhere (if only the landing site had the correct syntactic label). Everything else was regulated by conditions on the admissibility of structures.

Quite an interesting complication arose in the form of the so–called **parasitic gaps**.

(6.176) `Which papers did you file without reading?`

We are dealing here with two verbs, which share the same direct object (`to file` and `to read`). However, at deep structure only one them could have had the overt object phrase `which papers` as its object and so at deep structure we either had something like (6.177) or something like (6.178).

(6.177) `you did file which papers without reading?`

(6.178) `you did file without reading which papers?`

It was assumed that essentially (6.177) was the deep structure while the verb `to read` (in its form `reading`, of course) just got an empty coindexed object.

(6.179) `you did file which papers`$_1$ `without reading` e_1`?`

However, the empty element is not bound in this configuration. English does not allow such structures. The transformation that moves the wh–constituent at the beginning of the sentence however sees to it that a surface structure the pronoun is bound. This means that binding is not something that is decided at deep structure alone but also at surface structure. However, it cannot be one of the levels alone (see [**?**]). We have just come to see that deep structure alone gives the wrong result. If one replaces `which papers` by `which paper about yourself` then we have an example in which the binding

conditions apply neither exclusively at deep structure nor exclusively at surface structure. And the example shows that traces form an integral part of the theory.

A plethora of problems have since appeared that challenged the view and the theory had to be revised over and over again in order to cope with them. One problem area were the quantifiers and their scope. In German, quantifiers have scope more or less as in the surface structure, while in English matters are different (not to mention other languages here). Another problem is coordination. In a coordinative construction we may intuitively speaking delete elements. However, deletion is not an option any more. So, one has to assume that the second conjunct contains empty elements, whose distribution must be explained. The deep structure of (6.180) is for example (6.181). For many reasons, (6.182) or (6.183) would however be more desirable.

(6.180) `Karl hat Maria ein Fahrrad gestohlen und Peter ein Radio.`

Karl has Maria a bicycle stolen and Peter a radio.

Karl has stolen a bicycle from Maria and a radio from Peter.

(6.181) `Karl`$_1$ `Maria ein Fahrrad` [`gestohlen hat`]$_2$ `und` e_1 `Peter ein Radio` e_2`.`

(6.182) `Karl` [[`Maria ein Fahrrad`] `und` [`Peter ein Auto`]] `gestohlen hat.`

(6.183) `Karl` [[`Maria ein Fahrrad gestohlen hat`] `und` [`Peter ein Radio gestohlen hat.`]]

We shall conclude this section with a short description of GB. It is perhaps not an overstatement to say that GB has been the most popular variant of TG, so that it is perhaps most fruitful to look at this theory rather than previous ones (or even the subsequent **Minimalist Program**). GB is divided into several subtheories, so–called **modules**. Each of the modules is responsible for its particular set of phenomena. There is

① Binding Theory,

② the ECP (Empty Category Principle),

③ Control Theory,

④ Bounding Theory,

⑤ the Theory of Government,

⑥ Case Theory,

⑦ Θ–Theory,

⑧ Projection Theory.

The following four levels of representation were distinguished.

❶ D–Structure (formerly *deep structure*),

❷ S–Structure (formerly *surface structure*),

❸ Phonetic Form (PF) and

❹ Logical Form (LF).

There is only one transformation, called **Move**–α. It takes a constituent of category α (α arbitrary) and moves it to another place either by putting it in place of an empty constituent of category α (substitution) or by adjoining it to a constituent. Binding Theory however requires that trace always have to be bound, and so movement always is into a position c–commanding the trace. Substitution is defined as follows. Here X and Y are variables for strings α and γ category symbols. i is a variable for a natural number. It is part of the representation (more exactly, it is part of the label, which we may construe as a pair of a category symbol and a set of natural numbers). i may occur in the left hand side (SC) namely, if it figures in the label α. So, if $\alpha = \langle C, I\rangle$, C a category label and $I \subseteq \omega$, $\alpha \oplus i := \langle C, I \cup \{i\}\rangle$.

(6.184) Substitution: $[X\ [e]_\alpha\ Y\ [Z]_\alpha\ W] \Longrightarrow [X\ [Z]_{\alpha\oplus i}\ Y\ [t_i]_\alpha\ W]$

Adjunction is the following transformation.

(6.185) Adjunction: $[X\ [Y]_\alpha\ Z]_\gamma \Longrightarrow [[Y]_{\alpha\oplus i}\ [X\ [t_i]_\alpha\ Z]_\gamma]_\gamma$

Both rules make the constituent move leftward. Corresponding rightward rules can be formulated analogously. (In present day theory it is assumed that movement is always to the left. We shall not go into this, however.) In both cases the constituent on the right hand side, $[X]_{\alpha\oplus i}$, is called the **antecedent**

of the trace, t_i. This terminology is not arbitrary: traces in GB are considered as anaphoric elements. In what is to follow we shall not consider adjunction since it leads to complications that go beyond the scope of this exposition. For details we refer to [**?**]. For the understanding of the basic techniques (in particular with respect to Section 6.7) it is enough if we look at substitution.

As in TG, the D–structure is generated first. How this is done is not exactly clear. Chomsky assumes in [**?**] that it is freely generated and then checked for conformity with the principles. Subsequently, the movement transformation operates until the conditions for an S–structure are satisfied. Then a copy of the structure is passed on to the component which transforms it into a PF. (PF is only a level of representation, therefore there must be a process to arrive at PF.) For example, symbols like t_i, e, which are empty, are deleted together with all or part of the constituent brackets. The original structure meanwhile is subjected to another transformational process until it has reached the conditions of Logical Form and is directly interpretable semantically. Quantifiers appear in their correct scope at LF. This model is also known as the **T–model**.

We begin with the phrase structure, which is conditioned by the theory of projection. The conditions of theory of projection must in fact be obeyed at all levels (with the exception of PF). This theory is also known as $\overline{X}$–**syntax**. It differentiates between simple categorial labels (for example V, N, A, P, I and C, to name the most important ones) and a level of projection. The categorial labels are either **lexical** or **functional**. Levels of projection are natural numbers, starting with 0. The higher the number the higher the level. In the most popular version one distinguishes exactly 3 levels for all categories (while in [**?**] it was originally possible to specify the numbers of levels for each category independently). The levels are added to the categorial label as superscripts. So N^2 is synonymous with

(6.186) $$\begin{bmatrix} \text{CAT} & : & \text{N} \\ \text{PROJ} & : & 2 \end{bmatrix}$$

If X is a categorial symbol then XP is the highest projection. In our case NP is synonymous with N^2. The rules are at most binary branching. The non–branching rules are

(6.187) $$X^{j+1} \rightarrow X^j$$

X^j is the **head of** X^{j+1}. There are, furthermore, the following rules:

(6.188) $$X^{j+1} \rightarrow X^j \quad \text{YP}, \qquad X^{j+1} \rightarrow \text{YP} \quad X^j$$

Here, YP is called the **complement** of X^j if $j = 0$, and the **specifier** if $j = 1$. Finally, we have these rules.

(6.189) $X^j \to X^j \quad \mathrm{YP}, \qquad X^j \to \mathrm{YP} \quad X^j$

Here YP is called the **adjunct of** X^j. The last rules create a certain difficulty. We have two occurrences of the symbol X^j. This motivated the distinction between a **category** (= connected sets of nodes carrying the same label) and **segments** thereof. The complications that arise from this definition have been widely used by Chomsky in [**?**]. The relation **head of** is transitive. Hence x with category N^i is the head of y with N^j, if all nodes z with $x < z < y$ have category N^k for some k. By necessity, we must have $i \leq k \leq j$.

Heads possess in addition to their category label also a **subcategorization frame**. This frame determines which arguments the head needs and to which arguments it assigns case and/or a θ–role. θ–roles are needed to recover an argument in the semantic representation. For example, there are roles for **agent**, **experiencer**, **theme**, **instrument** and so on. These are coded by suggestive names such as θ_a, θ_e, θ_{th}, θ_{inst}, and so on. `see` gets for example the following subcategorization frame.

(6.190) `see` : $\langle \mathrm{NP}[\theta_e], \mathrm{NP}[\mathrm{ACC}, \theta_{th}] \rangle$

It is on purpose that the verb does not assign case to its subject. It only assigns a θ–role. The case is assigned only by virtue of the verb getting the finiteness marker. The subcategorization frames dictate how the local structure surrounding a head looks like. One says that the head **licenses** nodes in the deep structure, namely those which correspond to entries of its subcategorization frame. It will additionally determine that certain elements get case and/or a θ–role. Case- and Θ–Theory determine which elements need case/θ–roles and how they can get them from a head. One distinguishes between **internal** and **external arguments**. There is at most one external argument, and it is signalled in the frame by underlining it. It is found at deep structure outside of the maximal projection of the head (some theorists also think that it occupies the specifier of the projection of the head, but the details do not really matter here). Further, only one of the internal arguments is a complement. This is already a consequence of $\overline{X}$–syntax; the other arguments therefore have to be adjuncts at D–structure.

One of the great successes of the theory is the analysis of `seem`. The uninflected `seem` has the following frame.

(6.191) `seem` : $\langle \mathrm{INFL}^2[\theta_t] \rangle$

(INFL is the symbol of inflection. This frame is valid only for the variant which selects infinitives.) This verb has an internal argument, which must be realized by the complement in the syntactic tree. The verb assigns a θ–role to this argument. Once it is inflected, it has a subject position, which is assigned case but no θ–role. A caseless NP inside the complement must be moved into the subject position of `seem` in syntax, since being an NP it needs case. It can only appear in that position, however, if at deep structure it has been assigned a θ–role. The subject of the embedded infinitive however is a canonical choice: it only gets a θ–role, but still needs case.

(6.192) $\texttt{Jan}_1$ `seems` $[t_1$ `to sleep`$]$

(6.193) *`Seems [Jan to sleep]`

It is therefore possible to distinguish two types of intransitive verbs, those which assign a θ–role to their subject (`fall`) and those which do not (`seem`). There were general laws on subcategorization frames, such as

> *Burzio's Generalization.* A verb assigns case to its governed NP–argument if and only it assigns a θ–role to its external argument.

The Theory of Government is responsible among other for case assignment. It is assumed that nominative and accusative could not be assigned by heads (as we — wrongly, at least according to this theory — said above) but only in a specific configuration. The simplest configuration is that between head and complement. A verb having a direct complement licenses a direct object position. This position is qua structural property (being sister to an element licensing it) assigned accusative. The following is taken from [**?**], p. 293.

Definition 6.51 *x with label α* ***governs*** *y with label β iff (1) x and y are dominated by the same nodes with label XP, X arbitrary, and (2) either $\alpha = X^0$, where X is lexical or $\alpha = \mathrm{AGR}^0$ and (3) x c–commands y. x* ***governs*** *y* ***properly*** *if x governs y and either $\alpha = X^0$, X lexical, or x and y are coindexed.*

(Since labels are currently construed as pairs $\langle X^i, P\rangle$, where X^i is a category symbol with projection and P a set of natural numbers, we say that x and y are coindexed if the second component of the label of x and the second component of the label of y are not disjoint.) The ECP is responsible for the distribution of empty categories. In GB there is a whole army of different empty categories: e, a faceless constituent into which one could move, t, the

trace, PRO and pro, which were pronouns. The ECP says among other that t must always be properly governed, while PRO may never be governed. We remark that traces are not allowed to move. In Section 6.7 we consider this restriction more closely. The Bounding Theory concerns itself with the distance that syntactic processes may cover. It (or better: notions of distance) is considered in detail in Section 6.7. Finally, we remark that Transformational Grammar also works with conditions on derivations. Transformations could not be applied in any order but had to follow certain orderings. A very important one (which was the only one to remain in GB) was **cyclicity**. Let y be the antecedent of x after movement and $z \succ y$. Then let the interval $[x, z]$ be called the **domain** of this instance of movement.

Definition 6.52 *Let Γ be a set of syntactic categories. x is called a **bounding node** if the label of x is in Γ. A derivation is called **cyclic** if for any two instances of movement β_1 and β_2 and their domains B_1 and B_2 the following holds: if β_1 was applied before β_2 then every bounding node from B_1 is dominated (not necessarily properly) by some bounding node from B_2 and every bounding node from B_2 dominates (not necessarily properly) a bounding node from B_1.*

Principally, all finite sentences are bounding nodes. However, it has been argued by Rizzi (and others following him) that the choice of bounding categories is language dependent.

Notes on this section. This exposition may suffice to indicate how complex the theory was. We shall not go into the details of parametrization of grammars and learnability. We have construed transformations as acting on labelled (ordered) trees. No attempt has been made to precisify the action of transformations on trees. Also, we have followed common practice to write t_1, even though strictly speaking t is a symbol. So, it would have been more appropriate to write τ_1, say, to make absolutely clear that there is a symbol that gets erased. (In TG, deletion really erased the symbol. Today transformations may not delete, but deletion must take place on the way to PF, since there are plenty of 'empty' categories.)

Exercise 232. Coordinators like `and`, `or` and `not` have quite a flexible syntax, as was already remarked at the end of Section 3.5. We have `cat and dog`, `read and write`, `green and blue` and so on. What difficulties arise in connection with $\overline{X}$–syntax for these words? What solutions can you propose?

Exercise 233. A transformation is called *minimal* if it replaces at most two

adjacent symbols by at most two adjacent symbols. Let L be a recursively enumerable language. Construct a regular grammar G and a finite set of minimal transformations such that the generated set of strings is L. Here the criterion for a derivation to be finished is that no transformation can be applied. *Hint.* If L is recursively enumerable there is a Turing machine which generates L from a given regular set of strings.

Exercise 234. (Continuing the previous exercise.) We additionally require that the deep structure generated by G as well as all intermediate structures conform to $\overline{X}$–syntax.

Exercise 235. Write a 2–LMG that accommodates German V2 and damit– and davor–split.

Exercise 236. It is believed that if traces are allowed to move, we can create unbound traces by movement of traces. Show that this is not a necessary conclusion. However, the ambiguities that arise from allowing such movement on condition that it does not make itself unbound are entirely harmless.

6. GPSG and HPSG

In the 1980s, several alternatives to transformational grammar were being developed. One alternative was categorial grammar, which we have discussed in Chapter 3. Others were the grammar formalisms that used a declarative (or model theoretic) definition of syntactic structures. These are Generalised Phrase Structure Grammar (mentioned already in Section 6.1) and **Lexical–Functional Grammar** (**LFG**). GPSG later developed into HPSG. In this section we shall deal mainly with GPSG and HPSG. Our aim is twofold. We shall give an overview of the expressive mechanism that is being used in these theories, and we shall show how to translate these expressive devices into a suitable polymodal logic.

In order to justify the introduction of transformational grammar, Chomsky had given several arguments to show that traditional theories were completely inadequate. In particular, he targeted the theory of finite automata (which was very popular in the 1950s) and the structuralism. His criticism of finite automata is up to now unchallenged. His negative assessment of structuralism, however, was based on factual errors. First of all, Chomsky has made a caricature of Bloomfields structuralism by equating it with the claim that natural languages are strongly context free (see the discussion by Manaster–Ramer

and Kac [**?**]). Even if this was not the case, his arguments of the insufficiency of CFGs are questionable. Some linguists, notably Gerald Gazdar and Geoffrey Pullum, after reviewing these and other proofs eventually came to the conclusion that contrary to what has hitherto been believed all natural languages were context free. However, the work of Riny Huybregts and Stuart Shieber, which we have discussed already in Section 2.7 put a preliminary end to this story. On the other hand, as Rogers [**?**] and Kracht [**?**] have later shown, the theories of English proposed inside of GB actually postulated an essentially context free structure for it. Hence English is still (from a theoretical point of view) strongly context free.

An important argument against context free rules has been the fact that simple regularities of language such as agreement cannot be formulated in them. This was one of the main arguments by Paul Postal [**?**] against the structuralists (and other people), even though strangely enough TG and GB did not have much to say about it either. Textbooks only offer vague remarks about agreement to the effect that heads agree with their specifiers in certain features. Von Stechow and Sternefeld [**?**] are more precise in this respect. In order to formulate this exactly, one needs AVSs and variables for values (and structures). These tools were introduced by GPSG into the apparatus of context free rules, as we have shown in Section 6.1. Since we have discussed this already, let us go over to word order variation. Let us note that GPSG takes over $\overline{X}$–syntax more or less without change. It does, however, not insist on binary branching. (It allows even unbounded branching, which puts it just slightly outside of context freeness. However, the bound on branching may seem unnatural, see Section 6.4.) Second, GPSG separates the context free rules into two components: one is responsible for generating the dominance relation, the other for the precedence relation between sisters. The following rule determines that a node with label VP can have daughters, which may occur in any order.

(6.194) $\mathrm{VP} \rightarrow \mathrm{NP}[\mathit{nom}]\ \mathrm{NP}[\mathit{dat}]\ \mathrm{NP}[\mathit{acc}]\ \mathrm{V}$

This rule stands for no less than 24 different context free rules. In order to get for example the German word order of the subordinate clause we now add the following condition.

(6.195) $\mathrm{N} \prec \mathrm{V}$

This says that every daughter with label N is to the left of any daughter with label V. Hence there only remain 6 context free rules, namely those in which

the verb is at the end of the clause. (See in this connection the examples (6.161) – (6.164).) For German one would however not propose this analysis since it does not allow to put any adverbials in between the arguments of the verb. If one uses binary branching trees, the word order problems reappear again in the form of order of discharge (for which GPSG has no special mechanism). There are languages for which this is better suited. For example, Staal [**?**] has argued that Sanskrit has the following word orders: SVO, SOV, VOS and OVS. If we allow the following rules without specifying the linear order, these facts are accounted for.

(6.196) $\quad \mathrm{VP} \to \mathrm{NP}[nom] \quad \mathrm{V}^1, \qquad \mathrm{V}^1 \to \mathrm{NP}[acc] \quad \mathrm{V}^0$

All four possibilities can be generated — and no more.

Even if we ignore word order variation of the kind just described there remain a lot of phenomena that we must account for. GPSG has found a method of capturing the effect of a single movement transformation by means of a special device. It first of all defines **metarules**, which generate rules from rules. For example, to account for movement we propose that in addition to $[[\cdots Y \cdots]_W]_V$ also the tree $[Y_i\ [\cdots t_i \cdots]_W]_V$ will be a legitimate tree. To make this happen, there shall be an additional unary rule that allows to derive the latter tree whenever the former is derivable. The introduction of these rules can be captured by a general scheme, a metarule. However, in the particular case at hand one must be a bit more careful. It is actually necessary to do a certain amount of bookkeeping with the categories. GPSG borrows from categorial grammar the category W/Y, where W and Y are standard categories. In place of the rule $V \to W$ one writes $V \to Y\ W/Y$. The official notation is

(6.197) $$\begin{bmatrix} W \\ \textsc{slash} : Y \end{bmatrix}$$

How do we see to it that the feature $[\textsc{slash} : Y]$ is correctly distributed? Also here GPSG has tried to come up with a principled answer. GPSG distinguishes **foot features** from **head features**. Their behaviour is quite distinct. Every feature is either a foot feature or a head feature. The attribute SLASH is classified as a foot feature. (It is perhaps unfortunate that it is called a feature and not an attribute, but this is a minor issue.) For a foot feature such as SLASH, the SLASH–features of the mother are the **unification** of the SLASH–features of the daughters, which corresponds to the logical meet. Let us look more closely into that. If W is an AVS and f a feature then we denote by $f(W)$ the value of f in W.

Definition 6.53 *Let G be a set of rules over AVSs. f is a* ***foot feature in*** *G if for every maximally instantiated rule $A \to B_0 \cdots B_{n-1}$ the following holds.*

$$f(A) = \bigwedge_{i<n} f(B_i) \tag{6.198}$$

So, what this says is that the SLASH–feature can be passed on from mother to any number of its daughters. In this way [**?**] have seen to it that parasitic gaps can also be handled (see the previous section on this phenomenon). However, extreme care is needed. For the rules do not allow to count how many constituents of the same category have been extracted. **Head features** are being distributed roughly as follows.

Head Feature Convention. Let $A \to B_0 \cdots B_{n-1}$ be a rule with head B_i, and f a head feature. Then $f(A) = f(B_i)$.

The exact formulation of the distribution scheme for head features however is much more complex than for foot features. We shall not go into the details here.

This finishes our short introduction to GPSG. It is immediately clear that the languages generated by GPSG are context free if there are only finitely many category symbols and bounded branching. In order for this to be the case, the syntax of paths in an AVS was severely restricted.

Definition 6.54 *Let A be an AVS. A* ***path in*** *A is a sequence $\langle f_i : i < n \rangle$ such that $f_{n-1} \circ \cdots \circ f_0(A)$ is defined. The value of this expression is the* ***value*** *of the path.*

In [**?**] it was required that only those paths were legitimate in which no attribute occurs twice. In this way the finiteness is a simple matter. The following is left to the reader as an exercise.

Proposition 6.55 *Let A be a finite set of attributes and F a finite set of paths over A. Then every set of pairwise non–equivalent AVSs is finite.*

Subsequently to the discovery on the word order of Dutch and Swiss German this restriction finally had to fall. Further, some people had anyway argued that the syntactic structure of the verbal complex is quite different, and that this applies also to German. The verbs in a sequence of infinitives were argued to form a constituent, the so called **verb cluster**. This has been claimed in the GB framework for German and Dutch. Also, Joan Bresnan, Ron Kaplan,

Table 24. An LFG–Grammar

S	$\rightarrow$	NP	VP
		$(\uparrow \text{SUBJ} = \downarrow)$	$\uparrow = \downarrow$
NP	$\rightarrow$	Det	N
		$\uparrow = \downarrow$	$\uparrow = \downarrow$
VP	$\rightarrow$	V	PP
		$\uparrow = \downarrow$	$(\uparrow \text{OBJ} = \downarrow)$

Stanley Peters and Annie Zaenen argue in [**?**] for a different analysis, based on principles of LFG. Central to LFG is the assumption that there are three (or even more) distinct structures that are being built simultaneously:

- ☞ **c–structure** or constituent structure: this is the structure where the linear precedence is encoded and also the syntactic structure.
- ☞ **f–structure** or functional structure: this is the structure where the grammatical relations (subject, object) but also discourse relations (topic) are encoded.
- ☞ **a–structure** or argument structure: this is the structure that encodes argument relations (θ–roles).

A rule specifies a piece of c-, f- and a–structure together with correspondences between the structures. For simplicity we shall ignore a–structure from now on. An example is provided in Table 24. The rules have two lines: the upper line specifies a context free phrase structure rule of the usual kind for the c–structure. The lower line tells us how the c–structure relates to the f–structure. These correspondences will allow to define a unique f–structure (together with the universal rules of language). The rule if applied creates a local tree in the c–structure, consisting of three nodes, say 0, 00, 01, with label S, NP and VP, respectively. The corresponding f–structure is different. This is indicated by the equations. To make the ideas precise, we shall assume two sets of nodes in the universe, C and F, which are sets of c–structure and f–structure nodes, respectively. And we assume a function FUNC, which maps C to F. It is clear now how to translate context free rules into first–order

formulae. We directly turn to the f–structure statements. C–structure is a tree, F–structure is an AVS. Using the function UP to map a node to its mother, the equation $(\uparrow \text{SUBJ} = \downarrow)$ is translated as follows:

(6.199) $$(\uparrow \text{SUBJ} = \downarrow)(x) := \text{SUBJ} \circ \text{FUNC} \circ \text{UP}(x) = \text{FUNC}(x)$$

In simpler terms: I am my mother's subject. The somewhat simpler statement $\uparrow = \downarrow$ is translated by

(6.200) $$(\uparrow = \downarrow)(x) := \text{FUNC} \circ \text{UP}(x) = \text{FUNC}(x)$$

Here the f–structure does not add a node, since the predicate installs itself into the root node (by the second condition), while the subject NP is its SUBJ–value. Notice that the statements are local path equations, which are required to hold of the c–structure node under which they occur. LFG uses the fact that f–structure is flatter than c–structure to derive the Dutch and Swiss German sentences using rules of this kind, despite the fact that the c–structures are not context free.

While GPSG and LFG still assume a phrase structure skeleton that plays an independent role in the theory, HPSG actually offers a completely homogeneous theory that makes no distinction between the sources from which a structure is constrained. What made this possible is the insight that the attribute value formalism can also encode structure. A very simple possibility of taking care of the structure is the following. Already in GPSG there was a feature SUBCAT whose value was the subcategorization frame of the head. Since the subcategorization frame must map into a structure we require that in the rules

(6.201) $$X \to Y[\text{SUBCAT} : A] \quad B$$

where $B \leq A$. (Notice that the order of the constituents does not play any role.) This means nothing but that B is being subsumed under A that is to say that it is a special A. The difference with GPSG is now that we allow to stack the feature SUBCAT arbitrarily deep. For example, we can attribute to

the German word geben ('to give') the following category.

(6.202)
$$\begin{bmatrix} \text{CAT} & : v \\ \text{SUBCAT} : & \begin{bmatrix} \text{CAT} & : \text{NP} \\ \text{CASE} & : \textit{nom} \\ \text{SUBCAT} : & \begin{bmatrix} \text{CAT} & : \text{NP} \\ \text{CASE} & : \textit{dat} \\ \text{SUBCAT} : & \begin{bmatrix} \text{CAT} & : \text{NP} \\ \text{CASE} & : \textit{acc} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

The rules of combination for category symbols have to be adapted accordingly. This requires some effort but is possible without problems. HPSG essentially follows this line, however pushing the use of AVSs to the limit. Not only the categories, also the entire geometrical structure is now coded using AVSs. HPSG also uses structure variables. This is necessary in particular for the semantics, which HPSG treats in the same way as syntax. (In this it differs from GPSG. The latter uses a Montagovian approach, pairing syntactic rules with semantical rules. In HPSG — and LFG for that matter —, the semantics is coded up like syntax.) Parallel to the development of GPSG and related frameworks, the so called *constraint based approaches* to natural language processing were introduced. [**?**] provides a good reference.

Definition 6.56 *A **basic constraint language** is a finite set F of unary function symbols. A **constraint** is either an equation '$s(x) = t(x)$' or a statement '$s(x)\uparrow$'. A **constraint model** is a partial algebra $\mathfrak{A} = \langle A, \Pi\rangle$ for the signature. We write $\mathfrak{A} \vDash s = t$ iff for every a, $s^{\mathfrak{A}}(a)$ and $t^{\mathfrak{A}}(a)$ are defined and equal. $\mathfrak{A} \vDash s(x)\uparrow$ iff for every a, $s^{\mathfrak{A}}(a)$ is defined.*

Often, one has a particular constant $\underline{o}$, which serves as the root, and one considers equations of the form $s(\underline{o}) = t(\underline{o})$. Whereas the latter type of equation holds only at the root, the above type of equations are required to hold *globally*. We shall deal only with globally valid equations. Notice that we can encode atomic values into this language by interpreting an atom p as a unary function f_p with the idea being that f_p is defined at b iff p holds of b. (Of course, we wish to have $f_p(f_p(b)) = f_p(b)$ if the latter is defined, but we need not require that.) We give a straightforward interpretation of this language into modal logic. For each $f \in F$, take a modality, which we call by the same name. Every f satisfies $\langle f\rangle p \wedge \langle f\rangle q. \rightarrow .\langle f\rangle(p \wedge q)$. (This logic is known as $\mathsf{K.alt}_1$, see [**?**].) With each term t we associate a modality in the

obvious way: $f^\mu := f$, $(f(t))^\mu := f;t^\mu$. Now, the formula $s = t$ is translated by

$$(6.203)\qquad \begin{aligned} (s^\mu = t^\mu) &:= \langle s^\mu \rangle p \leftrightarrow \langle t^\mu \rangle p \\ (s\uparrow)^\mu &:= \langle s^\mu \rangle \top \end{aligned}$$

Finally, given $\mathfrak{A} = \langle A, \Pi \rangle$ we define a Kripke–frame $\mathfrak{A}^\mu := \langle A, R \rangle$ with $x\,R(f)\,y$ iff $f(x)$ is defined and equals y. Then

$$(6.204)\qquad \mathfrak{A} \vDash s = t \quad \Leftrightarrow \quad \mathfrak{A}^\mu \vDash (s = t)^\mu$$

Further,

$$(6.205)\qquad \mathfrak{A} \vDash (s\uparrow) \quad \Leftrightarrow \quad \mathfrak{A}^\mu \vDash (s\uparrow)^\mu$$

Now, this language of constraints has been extended in various ways. The attribute–value structures of Section 6.1 effectively extend this language by boolean connectives. $[C : A]$ is a shorthand for $\langle C \rangle A^\mu$, where A^μ is the modal formula associated with A. Moreover, following the discussion of Section 6.4 we use $\diamondminus$, $\diamondminus$, $\diamondplus$ and $\diamondplus$ to steer around in the phrase structure skeleton. HPSG uses a different encoding. It assumes an attribute called DAUGHTERS, whose value is a list. A list in turn is an AVS which is built recursively using the predicates FIRST and REST. (The reader may write down the path language for lists.) The notions of a Kripke–frame and a generalized Kripke–frame are then defined as usual. The Kripke–frames take the role of the actual syntactic objects, while the AVSs are simply formulae to talk about them.

The logic L_0 is of course not very interesting. What we want to have is a theory of the existing objects, not just all conceivable ones. A particular concern in syntactic theory is therefore the formulation of an adequate theory of the linguistic objects, be it a universal theory of all linguistic objects, or be it a theory of the linguistic objects of a particular language. We may cast this in logical terms in the following way. We start with a set (or class) $\mathcal{K}$ of Kripke–frames. The theory of that class is $\mathsf{Th}\,\mathcal{K}$. It would be most preferrable if for any given Kripke–frame $\mathfrak{F}$ we had $\mathfrak{F} \vDash \mathsf{Th}\,\mathcal{K}$ iff $\mathfrak{F} \in \mathcal{K}$. Unfortunately, this is not always the case. We shall see, however, that the situation is as good as one can hope for. Notice the implications of the setup. Given, say, the admissible structures of English, we get a modal logic $L_M(\text{Eng})$, which is an extension of L_0. Moreover, if $L_M(\text{Univ})$ is the modal logic of all existing

linguistic objects, then $L_M(\text{Eng})$ furthermore is an axiomatic extension of $L_M(\text{Univ})$. There are sets Γ and E of formulae such that

(6.206) $$L_0 \subseteq L_M(\text{Univ}) = L_0 \oplus \Gamma \subseteq L_M(\text{Eng}) = L_0 \oplus \Gamma \oplus E$$

If we want to know, for example, whether a particular formula φ is satisfiable in a structure of English it is not enough to test it against the postulates of the logic L_0, nor those of $L_M(\text{Univ})$. Rather, we must show that it is consistent with $L_M(\text{Eng})$. These problems can have very different complexity. While L_0 is decidable, this need not be the case for $L_M(\text{Eng})$ nor for $L_M(\text{Univ})$. The reason is that in order to know whether there is a structure for a logic that satisfies the axioms we must first guess that structure before we can check the axioms on it. If we have no indication of its size, this can turn out to be impossible. The exercises shall provide some examples. Another way to see that there is a problem is this. φ is a theorem of $L_M(\text{Eng})$ if it can be derived from $L_0 \cup \Gamma \cup E$ using modus ponens (MP), substitution and (MN). However, $\Gamma \cup E \Vdash_{L_0} \varphi$ iff φ can be derived from $L_0 \cup \Gamma \cup E$ using (MP) and (MN) alone. Substitution, however, is very powerful. Here we shall be concerned with the difference in expressive power of the basic constraint language and the modal logic. The basic constraint language allows to express that two terms (called paths for obvious reasons) are identical. There are two ways in which such an identity can be enforced. (a) By an axiom: then this axiom must hold of all structures under consideration. An example is provided by the agreement rules of a language. (b) As a datum: then we are asked to satisfy the equation in a particular structure. In modal logic, only equations as axioms are expressible. Except for trivial cases there is no formula $\varphi(s,t)$ in polymodal $\mathsf{K.alt_1}$ such that

(6.207) $$\langle \mathfrak{A}^\mu, \beta, x\rangle \vDash \varphi \Leftrightarrow s^{\mathfrak{A}}(x) = t^{\mathfrak{A}}(x)$$

Hence, modal logic is expressibly weaker than predicate logic, in which such a condition is easily written down. Yet, it is not clear that such conditions are at all needed in natural language. All that is needed is to be able to state conditions of that kind on all structures — which we can in fact do. (See [**?**] for an extensive discussion.)

HPSG also uses *types*. Types are properties of nodes. As such, they can be modelled by unary predicates in MSO, or by boolean constants in modal logic. For example, we have represented the atomic values by proposition constants. In GPSG, the atomic values were assigned only to Type 0 features.

HPSG goes further than that by typing AVSs. Since the AVS is interpreted in a Kripke–frame, this creates no additional difficulty. Reentrancy is modelled by path equations in constraint languages, and can be naturally expressed using modal languages, as we have seen. As an example, we consider the agreement rule (6.17) again.

$$\text{(6.208)} \qquad \begin{bmatrix} \text{CAT} : s \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT} : np \\ \text{AGR} : \boxed{1} \end{bmatrix} \qquad \begin{bmatrix} \text{CAT} : vp \\ \text{AGRS} : \boxed{1} \end{bmatrix}$$

In the earlier [**?**] the idea of reentrancy was motivated by *information sharing*. What the label $\boxed{1}$ says is that any information available under that node in one occurrence is available at any other occurrence. One way to make this true is to simply say that the two occurrences of $\boxed{1}$ are not distinct in the structure. (An analogy might help here. In the notation $\{a, \{a, b\}\}$ the two occurrences of a do not stand for different things of the universe: they both denote a, just that the linear notation forces us to write it down twice.) There is a way to enforce this in modal logic. Consider the following formula.

$$\text{(6.209)} \qquad \begin{aligned} &\langle \text{CAT} \rangle \mathsf{s} \wedge \diamondsuit(\diamond\bot \wedge \langle \text{CAT} \rangle \mathsf{np} \wedge \diamondsuit(\langle \text{CAT} \rangle \mathsf{vp} \wedge \diamondsuit\bot)) \\ &\quad \rightarrow \diamondsuit(\diamond\bot \wedge \langle \text{AGR} \rangle p \leftrightarrow \diamondsuit\langle \text{AGRS} \rangle p) \end{aligned}$$

This formula says that if we have an S which consists of an NP and a VP, then whatever is the value of AGR of the NP also is the value of AGRS of the VP.

The constituency structure that the rules specify can be written down using quantified modal logic. As an exercise further down shows, QML is so powerful that first–order ZFC can be encoded. (See Section 1.1 for the definition of ZFC.) In $\mathsf{MSO}(\in, =)$ one can write down an axiom that forces sets to be well–founded with respect to $\in$ and even write down the axioms of NBG (von Neumann–Gödel–Bernays Set Theory), which differs from ZFC in having a simpler scheme for set comprehension. In its place we have this axiom.

Class Comprehension. $(\forall P)(\forall x)(\exists y)(\forall z)(z \in y \leftrightarrow z \in x \wedge P(z))$.

It says that from a set x and an arbitrary subset of the universe P (which does not have to be a set) there is a *set* of all things that belong to both x and P. In presence of the results by Thatcher, Doner and Wright all this may sound paradoxical. However, the introduction of structure variables has made the structures into acyclic graphs rather than trees. However, our reformulation of HPSG is not expressed in QML but in the much weaker polymodal logic.

Thus, theories of linguistic objects are extensions of polymodal K. However, as [**?**] shows, by introducing enough modalities one can axiomatize a logic such that a Kripke–frame $\langle F, R\rangle$ is a frame for this logic iff $\langle F, R(\in)\rangle$ is a model of NGB. This means that effectively any higher order logic can be encoded into HPSG notation, since it is reducible to set theory, and thereby to polymodal logic. Although this is not per se an argument against using the notation, it shows that anything goes and that a claim to the effect that such and such phenomenon can be accounted for in HPSG is empirically vacuous.

Notes on this section. One of the seminal works in GPSG besides [**?**] is the study of word order in German by Hans Uszkoreit [**?**]. The constituent structure of the continental Germanic languages has been a focus of considerable debate between the different grammatical frameworks. The discovery of Swiss German actually put an end to the debate whether or not context free rules are appropriate. In GSPG it is assumed that the dominance and the precedence relations are specified separately. Rules contain a dominance skeleton and a specification that says which of the orderings is admissible. However, as Almerindo Ojeda [**?**] has shown, GPSG can also generate cross serial dependencies of the Swiss German type. One only has to relax the requirement that the daughters of a node must be linearly ordered to a requirement that the yield of the tree must be so ordered.

Exercise 237. Show that all axioms of ZFC and also *Class Comprehension* are expressible in $\mathsf{MSO}(\in, =)$.

Exercise 238. Show that the logic L_0 of any number of basic modal operators satisfying $\Diamond p \wedge \Diamond q \to \Diamond(p \wedge q)$ is decidable. This shows the decidability of L_0. *Hint.* Show that any formula is equivalent to a disjunction of conjunctions of statements of the form $\langle\delta\rangle\pi$, where δ is a sequence of modalities and π is either nonmodal or of the form $[m]\bot$.

Exercise 239. Write a grammar using LFG–rules of the kind described above to generate the crossing dependencies of Swiss German.

Exercise 240. Let A be an alphabet, T a Turing machine over A. The computation of T can be coded onto a grid of numbers $\mathbb{Z} \times \mathbb{N}$. Take this grid to be a Kripke–structure, with basic relations the immediate horizontal successor and predecessor, the transitive closure of these relations, and the vertical successor. Take constants c_a for every $a \in A \cup Q \cup \{\nabla\}$. c_∇ codes the position of the read write head. Now formulate an axiom φ_T such that a Kripke–structure satisfies φ_T iff it represents a computation of T.

7. Formal Structures of GB

We shall close this chapter with a survey of the basic mathematical constructs of GB. The first complex concerns constraints on syntactic structures. GB has many types of such constraints. It has for example many principles that describe the geometrical configuration within which an element can operate. A central definition is that of idc–command, often referred to as *c–command*, although the latter was originally defined differently.

Definition 6.57 *Let $\mathfrak{T} = \langle T, < \rangle$ be a tree, $x, y \in T$. x* ***idc–commands*** *y if for every $z > x$ we have $z \geq y$. A constituent $\downarrow x$* ***idc–commands*** *a constituent $\downarrow y$ if x idc–commands y.*

In [**?**], Jan Koster has proposed an attempt to formulate GB without the use of movement transformations. The basic idea was that the traces in the surface structure leave enough indication of the deep structure that we can replace talk of deep structure and derivations by talk about the surface structure alone. The general principle that Koster proposed was as follows. Let x be a node with label δ, and let δ be a so–called dependent element. (Dependency is defined with reference to the category.) Then there must exist a uniquely defined node y with label α which c–commands x, and is local to x. Koster required in addition that α and δ shared a property. However, in formulating this condition it turns out to be easier to constrain the possible choices of δ and α. In addition to the parameters α and δ it remains to say what locality is. Anticipating our definitions somewhat we shall say that we have $x\,R\,y$ for a certain relation R. [**?**] have surveyed the notions of locality that enter in the definition of R that were used in the literature and given a definition of command relation. Using this, [**?**] developed a theory of command relations that we shall outline here.

Definition 6.58 *Let $\langle T, < \rangle$ be a tree and $R \subseteq T^2$ a relation. R is called a* ***command relation*** *if there is a function $f_R \colon T \to T$ such that (1) – (3) hold. R is a* ***monotone command relation*** *if in addition it satisfies (4), and* ***tight*** *if it satisfies (1) – (5).*

① $R_x := \{y : x\,R\,y\} = \downarrow f_R(x)$.

② $x < f_R(x)$ *for all* $x < r$.

③ $f_R(r) = r$.

④ *If* $x \leq y$ *then* $f_R(x) \leq f_R(y)$.

⑤ *If* $x < f_R(y)$ *then* $f_R(x) \leq f_R(y)$.

The first class that we shall study is the class of tight command relations. Let $\mathfrak{T}$ be a tree and $P \subseteq T$. We say, x P–**commands** y if for every $z > x$ with $z \in P$ we have $z \geq y$. We denote the relation of P–command by $K(P)$. If we choose $P = T$ we exactly get idc–command. The following theorem is left as an exercise.

Proposition 6.59 *Let* R *be a binary relation on the tree* $\langle T, < \rangle$. R *is a tight command relation iff* $R = K(P)$ *for some* $P \subseteq T$.

Let $\mathfrak{T}$ be a tree. We denote by $\mathrm{MCr}(\mathfrak{T})$ the set of monotone command relations on $\mathfrak{T}$. This set is closed under intersection, union and relation composition. We even have

$$\begin{aligned} f_{R \cup S}(x) &= \max\{f_R(x), f_S(x)\} \\ f_{R \cap S}(x) &= \min\{f_R(x), f_S(x)\} \\ f_{R \circ S}(x) &= (f_S \circ f_R)(x) \end{aligned} \tag{6.210}$$

For union and intersection this holds without assuming monotonicity. For relation composition, however, it is needed. For suppose $x\, R \circ S\, y$. Then we can conclude that $x\, R\, f_R(x)$ and $f_R(x)\, S\, y$. Hence $x\, R \circ S\, y$ iff $y \leq f_S(f_R(x))$, from which the claim now follows. Now we set

$$\mathfrak{MCr}(\mathfrak{T}) := \langle \mathrm{MCr}(\mathfrak{T}), \cap, \cup, \circ \rangle \tag{6.211}$$

$\mathfrak{MCr}(\mathfrak{T})$ is a distributive lattice with respect to $\cap$ and $\cup$. What is more, there are additional laws of distribution concerning relation composition.

Proposition 6.60 *Let* $R, S, T \in \mathrm{MCr}(\mathfrak{T})$. *Then*

① $R \circ (S \cap T) = (R \circ S) \cap (R \circ T)$,
$(S \cap T) \circ R = (S \circ R) \cap (T \circ R)$.

② $R \circ (S \cup T) = (R \circ S) \cup (R \circ T)$,
$(S \cup T) \circ R = (S \circ R) \cup (T \circ R)$.

Proof. Let x be an element of the tree. Then

$$\begin{aligned} f_{R \circ (S \cap T)}(x) &= f_{S \cap T} \circ f_R(x) \\ &= \min\{f_S(f_R(x)), f_T(f_R(x))\} \\ &= \min\{f_{R \circ S}(x), f_{R \circ T}(x)\} \\ &= f_{(R \circ S) \cap (R \circ T)}(x) \end{aligned} \tag{6.212}$$

The other claims can be shown analogously. □

Definition 6.61 *Let $\mathfrak{T}$ be a tree, $R \in \mathrm{MCr}(\mathfrak{T})$. R is called **generated** if it can be produced from tight command relations by means of $\cap$, $\cup$ and $\circ$. R is called **chain like** if it can be generated from tight relations with $\circ$ alone.*

Theorem 6.62 *R is generated iff R is an intersection of chain line command relations.*

Proof. Because of Proposition 6.60 we can move $\circ$ to the inside of $\cap$ and $\cup$. Furthermore, we can move $\cap$ outside of the scope of $\cup$. It remains to be shown that the union of two chain like command relations is an intersection of chain like command relations. This follows from Lemma 6.66. □

Lemma 6.63 *Let $R = K(P)$ and $S = K(Q)$ be tight. Then*

$$R \cup S = (R \circ S) \cap (S \circ R) \cap K(P \cap Q) \tag{6.213}$$

Proof. Let x be given. We look at $f_R(x)$ and $f_S(x)$. Case 1. $f_R(x) < f_S(x)$. Then $f_{R\cup S}(x) = f_S(x)$. On the right hand side we have $f_S \circ f_R(x) = f_S(x)$, since S is tight. $f_R \circ f_S(x) \geq f_S(x)$, as well as $f_{K(P\cap Q)}(x) \geq f_S(x)$. Case 2. $f_S(x) < f_R(x)$. Analogously. Case 3. $f_S(x) = f_R(x)$. Then $f_{S\cup R}(x) = f_R(x) = f_S(x)$, whence $f_R \circ f_S(x), f_S \circ f_R(x) \geq f_{S\cup R}(x)$. The smallest node above x which is both in P and in Q is clearly in $f_S(x)$. Hence we have $f_{K(P\cap Q)}(x) = f_S(x)$. Hence equality holds in all cases. □

We put

$$K(P) \bullet K(Q) := K(P \cap P) \tag{6.214}$$

The operation $\bullet$ is defined only on tight command relations. If $\langle R_i : i < m\rangle$ is a sequence of command relations, then $R_0 \circ R_1 \circ \cdots \circ R_{n-1}$ is called its **product**. In what is to follow we shall characterize a union of chain like relations as the intersection of products. To this end we need some definitions. The first is that of a **shuffling**. This operation mixes two sequences in such a way that the liner order inside the sequences is respected.

Definition 6.64 *Let $\rho = \langle a_i : i < m\rangle$ and $\sigma = \langle b_j : j < n\rangle$ be sequences of objects. A **shuffling** of ρ and σ is a sequence $\langle c_k : k < m+n\rangle$ such that there are injective monotone functions $f\colon n \to m+n$ and $g\colon m \to m+n$ such that $\mathrm{im}(f) \cap \mathrm{im}(g) = \varnothing$ and $\mathrm{im}(f) \cup \mathrm{im}(g) = m+n$, as well as $c_{f(i)} = a_i$ for all $i < m$ and $c_{g(j)} = b_j$ for all $j < n$. f and g are called the **embeddings** of the shuffling.*

Definition 6.65 *Let $\rho = \langle R_i : i < m\rangle$ and $\sigma = \langle S_j : j < n\rangle$ be sequences of tight command relations. Then T is called* ***weakly associated with*** *ρ and σ if there is a shuffling $\tau = \langle T_i : i < m+n\rangle$ of ρ and σ together with embeddings f and g such that*

$$T = T_0 \circ^0 T_1 \circ^1 T_2 \cdots \circ^{n-2} T_{n-1} \tag{6.215}$$

where $\circ^i \in \{\circ, \bullet\}$ for $i < n-1$ and $\circ^i = \circ$ always if $\{i, i+1\} \subseteq \mathrm{im}(f)$ or $\{i, i+1\} \subseteq \mathrm{im}(g)$.

If $m = n = 2$, we have the following shufflings.

$$\begin{array}{lll} \langle R_0, R_1, S_0, S_1\rangle, & \langle R_0, S_0, R_1, S_1\rangle, & \langle R_0, S_0, S_1, R_1\rangle, \\ \langle S_0, R_0, R_1, S_1\rangle, & \langle S_0, R_0, S_1, R_1\rangle, & \langle S_0, S_1, R_0, R_1\rangle \end{array} \tag{6.216}$$

The sequence $\langle R_1, S_0, S_1, R_0\rangle$ is not a shuffling because the order of the R_i is not respected. In general there exist up to $\binom{m+n}{n}$ different shufflings. For every shuffling there are up to 2^{n-1} weakly associated command relations (if $n \leq m$). For example the following command relations are weakly associated to the third shuffling.

$$R_0 \bullet S_0 \circ S_1 \bullet S_1, \quad R_0 \circ S_0 \circ S_1 \circ R_1 \tag{6.217}$$

The relation $R_0 \circ S_0 \bullet S_1 \circ R_1$ is however not weakly associated to it since $\bullet$ may not occur in between two S.

Lemma 6.66 *Let $\rho = \langle R_i : i < m\rangle$ and $\sigma = \langle S_i : i < n\rangle$ be sequences of tight command relations with product T and U, respectively. Then $T \cup U$ is the intersection of all chain like command relations which are products of sequences weakly associated with a shuffling of ρ and σ.*

In practice one has restricted attention to command relations which are characterized by certain sets of nodes, such as the set of all maximal projections, the set of all finite sentences, the set of all sentences in the indicative mood and so on. If we choose P to be the set of nodes carrying a label subsuming the category of finite sentences, then we get the following: if x is a reflexive anaphor, it has to be c–commanded by a subject, which it in turn P–commands. (The last condition makes sure that the subject is a subject of the same sentence.) There is a plethora of similar examples where command relations play a role in defining the range of phenomena. Here, one took not just any old set of nodes but those that where *definable*. To precisify this, let $\langle \mathfrak{T}, \ell\rangle$ with $\ell\colon T \to N$ be a labelled tree and $Q \subseteq N$. Then $K(Q) := K(\ell^{-1}(Q))$ is called a **definable tight command relation**.

Definition 6.67 *Let $\mathfrak{T}$ be a tree and $R \subseteq T \times T$. P is called a* ***(definable) command relation*** *if it can be obtained from definable tight command relations by means of composition, union and intersection.*

In follows from the previous considerations that the union of definable relations is an intersection of chains of tight relations. A particular role is played by subjacency. The antecedent of a trace must be 1–subjacent to a trace. As is argued in [**?**] on the basis of [**?**] this relation is exactly

(6.218) $K(\mathsf{ip}) \circ K(\mathsf{cp})$

The movement and copy–transformations create so–called *chains*. Chains connect elements in different positions with each other. The mechanism inside the grammar is coindexation. For as we have said in Section 6.5 traces must be properly governed, and this means that an antecendent must c–command its trace in addition to being coindexed with it. This is a restriction on the structures as well as on the movement transformations. Using coindexation one also has the option of associating antecedent and trace without assuming that anything has ever moved. The transformational history can anyway be projected form the S–structure up to minor (in fact inessential) variations. This means that we need not care whether the S–structure has been obtained by transformations or by some other process introducing the indexation (this is what Koster has argued for). The association between antecedent and trace can also be done in a different way, namely by collecting sets of constituents. We call a chain a certain set of constituents. In a chain the members may be thought to be coindexed, but this is not necessary. Chomsky has once again introduced the idea in the 1990s that movement is the sequence of copying and deletion and made this one of the main innovations of the reform in the Minimalist Program (see [**?**]). Deletion here is simply marking as phonetically empty (so the copy remains but is marked). However, the same idea can be introduced into GB without substantial change. Let us do this here and introduce in place of Move–α the transformation **Copy**–α. It will turn out that it is actually not necessary to say which of the members of the chain has been obtained by copying from which other member. The reason is simple: the copy (= antecedent) c–commands the original (= trace) but the latter does not c–command the former. Knowing who is in a chain with whom is therefore enough. This is the central insight that is used in the theory of chains in [**?**] which we shall now outline. We shall see below that copying gives more

information on the derivation than movement, so that we must be careful in saying that nothing has changed by introducing copy–movement.

Recall that constituents are subtrees. In what is to follow we shall not distinguish between a set of nodes and the constituent that is based on that set. Say that x **ac–commands** y if x and y are incomparable, x idc–commands y but y does not idc–command x.

Definition 6.68 *Let $\mathfrak{T}$ be a tree. A set Δ of constituents of $\mathfrak{T}$ which is linearly ordered with respect to ac–command is called a* ***chain in*** *$\mathfrak{T}$. The element which is highest with respect to ac–command is called the* ***head of*** *Δ, the lowest the* ***foot****. Δ is a* ***copy chain*** *if any two members are isomorphic. Δ is a* ***trace chain*** *if all non heads are traces.*

The definition of chains can be supplemented with more detail in the case of copy chains. This will be needed in the sequel.

Definition 6.69 *Let $\mathfrak{T}$ be a tree. A* ***copy chain* in*** *$\mathfrak{T}$ is a pair $\langle\Delta,\Phi\rangle$ for which the following holds.*

① *Δ is a chain.*

② *$\Phi = \{\varphi_{\mathfrak{C},\mathfrak{D}} : \mathfrak{C},\mathfrak{D} \in \Delta\}$ is a family of isomorphisms such that for all $\mathfrak{C},\mathfrak{D},\mathfrak{A} \in \Delta$ we have*

(a) $\varphi_{\mathfrak{C},\mathfrak{C}} = 1_{\mathfrak{C}}$

(b) $\varphi_{\mathfrak{C},\mathfrak{A}} = \varphi_{\mathfrak{C},\mathfrak{D}} \circ \varphi_{\mathfrak{D},\mathfrak{A}}$

The ***chain associated with*** *$\langle\Delta,\Phi\rangle$ is Δ.*

Often we shall identify a chain* with its associated chain. The isomorphisms give explicit information which elements of the various constituents are counterparts of which others.

Definition 6.70 *Let $\mathfrak{T}$ be a tree and $\mathfrak{D} = \langle\Delta,\Phi\rangle$ a copy chain*. Then we put $x \approx_{\mathfrak{D}} y$ if there is a map $\varphi \in \Phi$ such that $\varphi(x) = y$. We put $[x]_{\mathfrak{D}} := \{y : x \approx_{\mathfrak{D}} y\}$. If C is a set of copy chains* then let $\approx_C$ be the smallest equivalence relation generated by all $\approx_{\mathfrak{D}}$, $\mathfrak{D} \in C$. Further, let $[x]_C := \{y : x \approx_C y\}$.*

Definition 6.71 *Let $\langle\Delta,\Phi\rangle$ be a copy chain*, $\mathfrak{C},\mathfrak{D} \in \Delta$. $\mathfrak{C}$ is said to be* ***immediately above*** *$\mathfrak{D}$ if there is no $\mathfrak{E} \in \Delta$ distinct from $\mathfrak{C}$ and $\mathfrak{D}$ which ac–commands $\mathfrak{D}$ and is ac–commanded by $\mathfrak{C}$. A* ***link of*** *Δ is a triple $\langle\mathfrak{C},\varphi_{\mathfrak{D},\mathfrak{C}},\mathfrak{D}\rangle$ where $\mathfrak{C}$ is immediately above $\mathfrak{D}$. φ is called a* ***link map*** *if it occurs in a link. An* ***ascending map*** *is a composition of link maps.*

Lemma 6.72 *Let φ be a link map. Then $t(\varphi(x)) < t(x)$.*

Proof. Let $\varphi = \varphi_{\mathfrak{C},\mathfrak{D}}$, $\mathfrak{C} = \downarrow v$, $\mathfrak{D} = \downarrow w$. Further, let $t_{\mathfrak{C}}(x)$ be the depth of x in $\mathfrak{C}$, $t_{\mathfrak{D}}(\varphi(x))$ the depth of $\varphi(x)$ in $\mathfrak{D}$. Then $t_{\mathfrak{C}}(x) = t_{\mathfrak{D}}(\varphi(x))$, since φ is an isomorphism. On the other hand $t(x) = t(v) + t_{\mathfrak{C}}(x)$ and $t(\varphi(x)) = t(w) + t_{\mathfrak{D}}(\varphi(x)) = t(w) + t_{\mathfrak{C}}(x)$. The claim now follows from the next lemma given the remark that v c–commands w, but w does not c–command v. □

Lemma 6.73 *Let $\mathfrak{T} = \langle T, < \rangle$ be a tree, $x, y \in T$. If x ac–commands y, $t(x) \leq t(y)$.*

Proof. There exists a uniquely defined z with $z \succ x$. By definition of c–command we have $z \geq y$. But $y \neq z$, since y is not comparable with x. Hence $y < z$. Now we have $t(x) = t(z) + 1$ and $t(y) > t(z) = t(x) - 1$. Whence the claim. □

We call a pair $\langle \mathfrak{T}, C \rangle$ a **copy chain tree** (**CCT**) if C is a set of copy chains* on $\mathfrak{T}$, $\mathfrak{T}$ a finite tree. We consider among other the following constraints.

Uniqueness. Every constituent of $\mathfrak{T}$ is contained in exactly one chain.
Liberation. Let Γ, Δ be chain, $\mathfrak{C} \in \Gamma$ and $\mathfrak{D}_0, \mathfrak{D}_1 \in \Delta$ with $\mathfrak{D}_0 \neq \mathfrak{D}_1$ such that $\mathfrak{D}_0, \mathfrak{D}_1 \subseteq \mathfrak{C}$. Then $\mathfrak{C}$ is the foot of Γ.

Lemma 6.74 *Let K be a CCT which satisfies Uniqueness and Liberation. Further, let φ and φ' be link maps with* $\mathrm{im}(\varphi) \cap \mathrm{im}(\varphi') \neq \varnothing$. *Then already $\varphi = \varphi'$.*

Proof. Let $\varphi \colon \mathfrak{C} \to \mathfrak{D}$, $\varphi' \colon \mathfrak{C}' \to \mathfrak{D}'$ be link maps. If $\mathrm{im}(\varphi) \cap \mathrm{im}(\varphi') \neq \varnothing$ then $\mathfrak{D} \subseteq \mathfrak{D}'$ or $\mathfrak{D}' \subseteq \mathfrak{D}$. Without loss of generality we may assume the first. If $\mathfrak{D} \subsetneq \mathfrak{D}'$ then also $\mathfrak{C} \subseteq \mathfrak{D}'$, since $\mathfrak{D}$ c–commands $\mathfrak{C}$. By *Liberation* $\mathfrak{D}'$ is the foot of its chain, in contradiction to our assumption. Hence we have $\mathfrak{D} = \mathfrak{D}'$. By *Uniqueness*, $\mathfrak{C}$, $\mathfrak{C}'$ and $\mathfrak{D}$ are therefore in the same chain. Since φ and φ' are link maps, we must have $\mathfrak{C} = \mathfrak{C}'$. Hence $\varphi = \varphi'$. □

Definition 6.75 *Let K be a CCT. x is called a* ***root*** *if x is not in the image of a link map.*

Then proof of the following theorem is now easy to provide. It is left for the reader.

Proposition 6.76 *Let K be a CCT which satisfies* Uniqueness *and* Liberation. *Let x be an element and τ_i, $i < m$, φ_j, $j < n$, link maps, and y, z roots such that*

$$x = \tau_{m-1} \circ \tau_{m-2} \circ \cdots \circ \tau_0(y) = \varphi_{n-1} \circ \varphi_{n-2} \circ \cdots \circ \varphi_0(z) \tag{6.219}$$

Then we have $y = z$, $m = n$ and $\tau_i = \varphi_i$ for all $i < n$.

Hence, for given x there is a uniquely defined root x_r with $x \approx_C x_r$. Further, there exists a unique sequence $\langle \varphi_i : i < n \rangle$ of link maps such that x is the image of $\varphi_{n-1} \circ \cdots \circ \varphi_0$. This sequence we call the **canonical decomposition of** x.

Proposition 6.77 *Let K be a CCT satisfying* Uniqueness *and* Liberation. *Then the following are equivalent.*

① $x \approx_C y$.

② $x_r = y_r$.

③ *There exist two ascending maps χ and τ with $y = \tau \circ \chi^{-1}(x)$.*

Proof. ① $\Rightarrow$ ③. Let $x \approx_C y$. Then there exists a sequence $\langle \sigma_i : i < p \rangle$ of link maps or inverses thereof such that $y = \sigma_{p-1} \circ \cdots \circ \sigma_0(x)$. Now if σ_i is a link map and σ_{i+1} an inverse link map, then $\sigma_{i+1} = \sigma_i^{-1}$. Hence we may assume that for some $q \leq p$ all σ_i, $i < q$, are inverse link maps and all σ_i, $p > i \geq q$, are link maps. Now put $\tau := \sigma_p \circ \sigma_{p-1} \cdots \circ \sigma_q$ and $\chi := \sigma_0 \circ \sigma_1 \circ \cdots \circ \sigma_{q-1}$. χ and τ are ascending maps. So, ③ obtains. ③ $\Rightarrow$ ②. Let ascending maps χ and τ be given with $y = \tau \circ \chi^{-1}(x)$. Put $u := \chi^{-1}(x)$. Then $u = \rho(u_r)$ for some ascending map ρ. Further, $x = \chi(u) = \chi \circ \rho(u_r)$ and $y = \tau(u) = \tau \circ \rho(u_r)$. Now, u_r is a root and x as well as y are images of u_r under ascending maps. Hence u_r is a root of x and y. This however means that $u_r = x_r = y_r$. Hence, ② obtains. ② $\Rightarrow$ ① is straightforward. □

The proof also establishes the following fact.

Lemma 6.78 *Every ascending map is a canonical decomposition. Every composition of maps equals a product $\tau \circ \chi^{-1}$ where τ and χ are ascending maps. A minimal composition of link maps and their inverses is unique.*

Let x be an element and $\langle \varphi_i : i < n \rangle$ its canonical decomposition. Then we call

$$T_K(x) := \{\varphi_{j-1} \circ \varphi_{j-2} \circ \cdots \circ \varphi_0(x) : j \leq n\} \tag{6.220}$$

the **trajectory of** x. The trajectory mirrors the history of x in the process of derivation. We call **root line** of x the set

(6.221) $$W_K(x) := \{y : y \in T_K(x), y \text{ idc–commands } x_r\}$$

Notice that x_r idc–commands itself. The **peak of** x is the element of $W_K(x)$ of smallest depth. We write x_π for the peak of x and π_x for the ascending map which sends x to x_π.

Definition 6.79 *Let K be a CCT satisfying Uniqueness and Liberation. If r is the root of the tree then r is the **zenith of** r, the **zenith map** is $\zeta_r := 1_T$. If $x \neq r$ then the **zenith map** is the composition $\zeta_y \circ \pi_x$, where $y \succ x_\pi$. The **zenith of** x equals $\zeta_y \circ \pi_x(x)$. We write x_ζ for the zenith of x.*

Definition 6.80 *A link map is called **orbital** if it occurs in a minimal decomposition of the zenith map.*

At last we can formulate the following restriction on CCTs.

No Recycling. All link maps are orbital.

The effect of a copy transformation is that (1) it adds a new constituent and (2) this constituent is added to an already existing chain as a head. Hence the whole derivation can be thought of as a process which generates a tree together with its chains. These can be explicitly described and this eliminates the necessity of talking about transformations.

Definition 6.81 *A **copy chain structure (CCS)** is a CCT $K = \langle \mathfrak{T}, C \rangle$ which satisfies Uniqueness, Liberation and No Recycling.*

Everything that one wants to say about transformations and derivations can be said also about copy chain structures. The reason for this is the following fact. We call a CCT simply a **tree** if every chain consists of a single constituent. Then also this tree is a CCS. A transformation can naturally be defined as an operation between CCSs. It turns out that Copy–α turns a CCS into a CCS. The reason for this is that traces have to be bound and may not be moved. (Only in order to reflect this in the definition of the CCSs the condition *No Recycling* has been introduced. Otherwise it was unnecessary.) The following now holds.

Theorem 6.82 *A CCT is a CCS iff it is obtained from a tree by successive application of Copy–α.*

Transformational grammar and HPSG are not as different as one might think. The appearance to the contrary is created by the fact that TG is written up using trees, while HPSG has acyclic structures, which need not be trees. In this section we shall show that GB actually defines structures that are more similar to acyclic graphs than to trees. The basis for the alternative formulation is the idea that instead of movement transformations we define an operation that changes the dominance relation. If the daughter constituent z of x moves and becomes a daughter constituent of y then we can simply add to the dominance relation the pair $\langle z, y\rangle$. This rather simple idea has to be worked out carefully. For first we have to change from using the usual transitive dominance relation the immediate dominance relation. Second one has to take care of the linear order of the elements at the surface since it is now not any more represented.

Definition 6.83 *A **multidominance structure (MDS)** is a triple $\langle M, \prec, r\rangle$ such that $\langle M, \prec\rangle$ is a directed acyclic graph with root r and for every $x < r$ the set $M(x) := \{y : x \prec y\}$ is linearly ordered by $<$.*

With an MDS we only have coded the dominance relation between the constituents. In order to include order we cannot simply add another relation as we did with trees. Depending on the branching number, a fair number of new relations will have to be added, which represent the relations *the ith daughter of* (where $i < n$, the maximum branching number). Since we are dealing with binary branching trees we need only two of these relations.

Definition 6.84 *An **ordered (binary branching) multidominance structure (OMDS)** is a quadruple $\langle M, \prec_0, \prec_1, r\rangle$ such that the following holds:*

① *$\langle M, \prec_0 \cup \prec_1, r\rangle$ is an MDS.*

② *From $x \succ_0 y$ and $x \succ_0 z$ follows $y = z$.*

③ *From $x \succ_1 y$ and $x \succ_1 z$ follows $y = z$.*

④ *If $x \succ_1 z$ for some z then there exists a $y \neq z$ with $x \succ_0 y$.*

(The reader may verify that ② and ④ together imply that $\succ_0 \cap \succ_1 = \varnothing$.) Let $\langle \mathfrak{T}, <, \sqsubset\rangle$ be a binary branching ordered tree. Then we put $x \prec_0 y$ if x is a daughter of y and there is no daughter z of y with $z \sqsubset x$. Further, we write $x \prec_1 y$ if x is a daughter of y but not $x \prec_0 y$.

Theorem 6.85 *Let $K = \langle \mathfrak{T}, C \rangle$ be a CCS over an ordered binary branching tree with root r. Put $M := [x]_C$, $x \in T$, as well as for $i = 0, 1$, $[x]_C \prec_i [y]_C$ iff there is an $x' \approx_C x$ and an $y' \approx_C y$ with $x' \prec_i y'$. Finally let*

$$M(K) := \langle M, \prec_0, \prec_1, [r]_K \rangle \tag{6.222}$$

Then $M(K)$ is an OMDS.

Now we want to deal with the problem of finding the CCS from the OMDS.

Definition 6.86 *Let $\langle M, \prec_0, \prec_1, r \rangle$ be an OMDS. An **identifier** is a sequence $I = \langle x_i : i < n \rangle$ such that $r \succ x_0$ and $x_i \succ x_{i+1}$ for all $i \in n$. $\mathcal{I}(\mathfrak{M})$ denotes the set of all identifiers of $\mathfrak{M}$. The **address of** I is that sequence $\langle \gamma_i : i < n \rangle$ such that for all $i < n$ one has $x_i \prec_{\gamma_i} x_{i-1}$.*

The following is easy to see.

Proposition 6.87 *The set of addresses of an OMDS is a tree domain.*

This means that we have already identified the tree structure. What remains to do is to find the chains. The order is irrelevant, so we ignore it. At first we want to establish which elements are overt. In a CCS an element x is called **overt** if for every $y \geq x$ the constituent $\downarrow y$ is the head of its chain. This we can also describe in the associated MDS. We say a pair $\langle x, y \rangle$ is a **link in** $\langle M, \prec, r \rangle$ if $x \prec y$. The link is **maximal** if y is maximal with respect to $<$ in $M(x)$. An **S–identifier** is an identifier $I = \langle x_i : i < n \rangle$ where $\langle x_{i-1}, x_i \rangle$ is a maximal link for all $i < n$. (For the purpose of this definition, x_{-1} is the root.) The overt elements are exactly the S–identifiers.

Definition 6.88 *Let $\mathfrak{M} = \langle M, \prec, r \rangle$ and $\mathfrak{M}' = \langle M', \prec', r' \rangle$ be MDSs. Then $\mathfrak{M}'$ is called a **link extension of** $\mathfrak{M}$ if $M' = M$, $r' = r$ and $\prec' = \prec \cup \{\langle x, y \rangle\}$, where $\langle x, y \rangle$ is maximal in $\mathfrak{M}'$.*

One finds out easily that if K' is derived from K by simple copying then $M(K')$ is isomorphic to a link extension of $M(K)$. Let conversely $\mathfrak{M}'$ be a link extension of $\mathfrak{M}$ and K a CCS such that $M(K) \cong \mathfrak{M}$. Then we claim that there is a CCS K' for which $M(K') \cong \mathfrak{M}'$ and which results by copying from K. This is unique up to isomorphism. The tree is given by $\mathcal{I}(\mathfrak{M}')$. Further, let the tree of K be exactly $\mathcal{I}(\mathfrak{M})$. First we have $\mathcal{I}(\mathfrak{M}) \subset \mathcal{I}(\mathfrak{M}')$, and the identity is an embedding whose image contains all identifiers which do not contain the subsequence $x; y$. Let now y' be maximal with respect to $<$ in

$\mathfrak{M}$. Further, let I be the S–identifier of y and I' the S–identifier of y' in $\mathfrak{M}$. Then $I' = I;J$ for some J since $y' < y$. Define $\varphi : I;J;x;K \mapsto I;x;K$. This is an isomorphism of the constituent $\downarrow I;J;x$ onto the constituent $\downarrow I;x$. Now we define the chains* on $\mathcal{I}(\mathfrak{M}')$. Let $\mathcal{D} = \langle \Delta, \Phi \rangle$ be the chain of K which contains the constituent $\downarrow I;J;xK$. Then let $\mathcal{D}' := \langle \Delta \cup \{\mathrm{im}(\varphi)\}, \Phi' \rangle$, where $\Phi' := \Phi \cup \{\varphi \circ \chi : \chi \in \Phi\} \cup \{\chi \circ \varphi^{-1} : \chi \in \Phi\}$. For every other chain $\mathcal{C}$ let $\mathcal{C}' := \mathcal{C}$. Finally for an identifier $L < I;J;x;K$ we put $\mathcal{K}_L := \langle \{\downarrow L\}, \{1_{\downarrow L}\} \rangle$. Then we put

$$K' := \langle \mathcal{I}(\mathfrak{M}'), <, \varepsilon, \{\mathcal{C}' : \mathcal{C} \in C\} \cup \{\mathcal{K}_L : L < I;J;x;K\} \rangle \tag{6.223}$$

This is a CCS. Evidently it satisfies *Uniqueness*. Further, *Liberation* is satisfied as one easily checks. For *No Recycling* it suffices that the new link map is orbital. This is easy to see.

Now, how does one define the kinds of structures that are common in GB? One approximation is the following. We say a **trace chain structure** is a pair $\langle \mathfrak{T}, C \rangle$ where C is a set of trace chains. If we have a CCS we get the trace chain structure relatively easily. To this end we replace all maximal nonovert constituents in a tree by a trace (which is a one node tree). This however deletes some chain members! Additionally it may happen that some traces are not any more bound. Hence we say that a trace chain structure is a pair $\langle \mathfrak{T}, C \rangle$ which results from a CCS by deleting overt constituents. Now one can define trace chain structures also from MDSs, and it turns out that if two CCSs K and K' have isomorphic MDSs then their trace chain structures are isomorphic. This has the following reason. An MDS is determined from $\mathfrak{T}$ and $\approx_C$ alone. We can determine the root of every element from $\mathfrak{T}$ and $\approx_C$, and further also the root line. From this we can define the peak of every element and therefore also the zenith. The overt elements are exactly the elements in zenith position. Except for the overt element, the trace chain structure contains also the traces. These are exactly the overt daughters of the overt elements.

Let us summarize. There exists a biunique correspondence between derivations of trace chain structures, derivations of CCSs and derivations of MDSs. Further, there is a biunique correspondence between MDSs and trace chain structures. Insofar the latter two structures are exactly equivalent. CCSs contain more information over the derivation (see the exercises).

Exercise 241. This example shows why we cannot use the ordering $<$ in the MDSs. Let $\mathfrak{M} = \langle \{0,1,2\}, \prec, 0 \rangle$ and $\mathfrak{M}' = \langle \{0,1,2\}, \prec', 0 \rangle$ with $\prec = \{\langle 2,1 \rangle, \langle 1,0 \rangle\}$ and $\prec' = \prec \cup \{\langle 2,0 \rangle\}$. Evidently $\prec^+ = \prec'^+$. Construct $\mathcal{I}(\mathfrak{M})$

and $\mathcal{I}(\mathfrak{M}')$ as well as the connected CCS.

Exercise 242. Prove Proposition 6.59.

Exercise 243. Show Lemma 6.66.

Exercise 244. Show that ac–command is transitive.

Exercise 245. Show Proposition 6.76.

Exercise 246. Let the CCS in Figure 18 be given. The members of a chain are annotated by the same upper case Greek letter. Trivial chains are not shown. Let the link maps be $\varphi_\Gamma\colon 2 \mapsto 4$, $\varphi_\Delta\colon i \mapsto i+6 \quad (i<6)$, and also $\varphi_\Theta\colon i \mapsto i+13 \quad (i<13)$. Compute $[i]_C$ for every i. If instead of φ_Δ we take the map φ'_Δ how do the equivalence classes change?

$$\varphi'_\Delta\colon 1 \mapsto 8, 2 \mapsto 7, 3 \mapsto 9, 4 \mapsto 10, 5 \mapsto 11 \tag{6.224}$$

Determine the peak and the zenith of every element and the maps.

Exercise 247. Let $d(n)$ be the largest number of nonisomorphic CCSs which have (up to isomorphism) the same MDS. Show that $d(n) \in O(2^n)$.

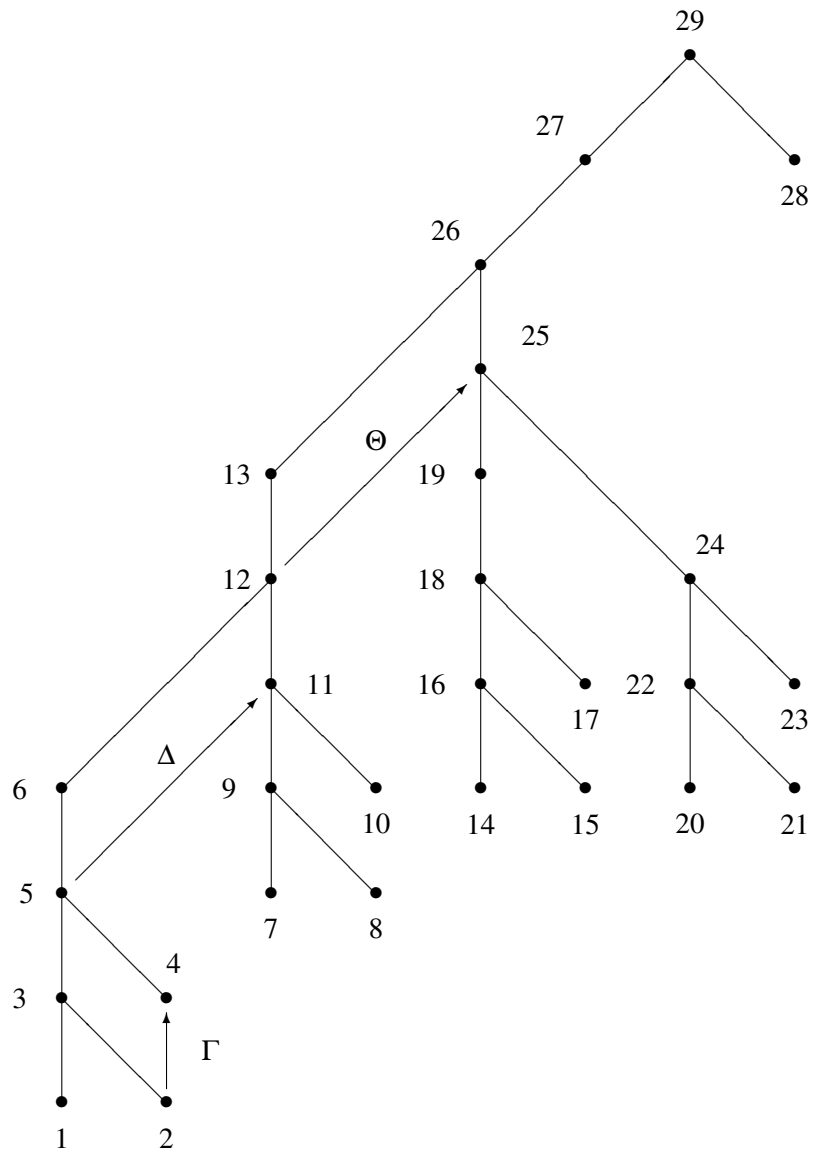


Figure 18. A Copy–Chain Structure

Bibliography

Ajdukiewicz, Kazimierz
1936 Die syntaktische Konnexität [The syntactic connectivity]. *Studia Philosophica* 1:1 – 27.

Barendregt, Henk
1985 *The Lambda Calculus. Its Syntax and Semantics*. Studies in Logic and the Foundations of Mathematics 103. 2nd ed. Amsterdam: Elsevier.

Barker, Chris, and Pullum, Geoffrey.
1990 A theory of command relations. *Linguistics and Philosophy* 13:1–34.

Bauer, Brigitte L. M.
1995 *The Emergence and Development of SVO Patterning in Latin and French*. Oxford: Oxford University Press.

Bird, Steven, and Ellison, Mark
1994 One–level phonology: Autosegmental representations and rules as finite automata. *Computational Linguistics* 20:55 – 90.

Blackburn, Patrick
1993 Modal logic and attribute value structures. In *Diamonds and Defaults*, Maarten de Rijke (ed.), 19 – 65. (Synthese Library 229.) Dordrecht: Kluwer.

Blok, Wim J., and Pigozzi, Don J.
1990 Algebraizable logics. *Memoirs of the Americal Mathematical Society*, 77(396).

Bochvar, D. A.
1938 On a three–valued logical calculus and its application to the analysis of contradictions. *Mathematicheskii Sbornik* 4:287 – 308.

Böttner, Michael, and Thümmel, Wolf (eds.)
2000 *Variable–free Semantics*. Artikulation und Sprache 3. Osnabrück: secolo Verlag.

Bresnan, Joan, Kaplan, Ronald M., Peters, Stanley, and Zaenen, Annie
1987 Cross–Serial Dependencies in Dutch. In *The Formal Complexity of Natural Language*, Walter Savitch, Emmon Bach, William Marsch, and Gila Safran–Naveh (eds.), 286 – 319. Dordrecht: Reidel.

Büchi, J.
1960 Weak second–order arithmetic and finite automata. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik* 6:66 – 92.

Burmeister, Peter
1986 *A Model Theoretic Oriented Approach to Partial Algebras*. Berlin: Akademie Verlag.
2002 Lecture Notes on Universal Algebra. Many Sorted Partial Algebras. Manuscript available via internet.

Burris, Stanley, and Sankappanavar, H. P.
1981 *A Course in Universal Algebra*. Graduate Texts in Mathematics 78. Berlin/New York: Springer.

Buszkowski, Wojciech
1997 Mathematical linguistics and proof theory. In *Handbook of Logic and Language*, Johan van Benthem and Alice ter Meulen (eds.), 683 – 736. Amsterdam: Elsevier.

Carpenter, Bob
1992 *The Logic of Typed Feature Structures*. Cambridge Tracts in Theoretical Computer Science 32. Cambridge: Cambridge University Press.

Chandra, A. K., Kozen, D. C., and Stockmeyer, L. J.
1981 Alternation. *Journal of the Association for Computing Machinery* 28:114 – 133.

Chomsky, Noam and Halle, Morris
1968 *The sound pattern of English*. New York: Harper and Row.

Chomsky, Noam
1959 On certain formal properties of grammars. *Information and Control* 2:137 – 167.
1962 Context–free grammars and pushdown storage. *MIT Research Laboratory of Electronics Quarterly Progress Report* 65.
1981 *Lecture Notes on Government and Binding*. Dordrecht: Foris.
1986 *Barriers*. Cambridge (Mass.): MIT Press.
1993 A minimalist program for linguistic theory. In *The View from Building 20: Essays in Honour of Sylvain Bromberger*, Ken Hale and Samuel J. Keyser (eds.), 1 – 52. Cambridge (Mass.): MIT Press.

Church, Alonzo
1933 A set of postulates for the foundation of logic. *Annals of Mathematics* 2:346 – 366.
1940 A formulation of the simple theory of types. *Journal of Symbolic Logic* 5:56 – 68.

Coulmas, Florian
2003 *Writing Systems. An introduction to their linguistic analysis*. Cambridge: Cambridge University Press.

Culy, Christopher
1987 The Complexity of the Vocabulary of Bambara. In *The Formal Complexity of Natural Language*, Walter Savitch, Emmon Bach, William Marsch, and Gila Safran–Naveh (eds.), 345 – 351. Dordrecht: Reidel.

Curry, Haskell B.
1930 Grundlagen der kombinatorischen Logik [Foundations of combinatory logic]. *American Journal of Mathematics* 52:509 – 536, 789 – 834.
1977 *Foundations of Mathematical Logic*. 2nd ed. New York: Dover Publications.

Davey, B. A., and Priestley, H. A.
1990 *Lattices and Order*. Cambridge: Cambridge University Press.

Deutsch, David, Ekert, Artur, and Luppacchini, Rossella
2000 Machines, logic and quantum physics. *Bulletin of Symbolic Logic* 6:265 – 283.

Doner, J. E.
1970 Tree acceptors and some of their applications. *Journal of Computer and Systems Sciences* 4:406 – 451.

Dowty, David R., Wall, Robert E., and Peters, Stanley
1981 *Introduction to Montague Semantics*. Synthese Library 11. Dordrecht: Reidel.

Dresner, Eli
2001 Tarski's Restricted Form and Neale's Quantificational Treatment of Proper Names. *Linguistics and Philosophy* 24:405 – 415.
2002 Holism, Language Acquisition, and Algebraic Logic. *Linguistics and Philosophy* 25:419 – 452.

Dymetman, Marc
1991 Inherently reversible grammars, logic programming and computability. In *Proceedings of the ACL Workshop: Reversible Grammars in Natural Language Processing*.
1992 *Transformations de Grammaires Logiques et Réversibilité* [Transformations of Logical Grammars and Reversibility]. Ph. D. diss., Université Joseph Fourier, Grenoble.

Ebbinghaus, Hans-Dieter, and Flum, Jörg
1995 *Finite Model Theory*. Perspectives in Mathematical Logic. Berlin/New York: Springer.

Ebert, Christian, and Kracht, Marcus
2000 Formal syntax and semantics of case stacking languages. In *Proceedings of the EACL 2000*.

van Eijck, Jan
1994 Presupposition failure: a comedy of errors. *Formal Aspects of Computing* 3.

Eisenberg, Peter
1973 A Note on 'Identity of Constituents'. *Linguistic Inquiry* 4:417 – 420.

Ewen, Colin J., and van der Hulst, Harry
2001 *The Phonological Structure of Words*. Cambridge: Cambridge University Press.

Ferreirós, José
2001 The road to modern logic — an interpretation. *The Bulletin of Symbolic Logic* 7:441 – 484.

Fiengo, Robert, and May, Robert
1994 *Indices and Identity*. Linguistic Inquiry Monographs 24. Cambridge (Mass.): MIT Press.

Fine, Kit
1992 Transparency, Part I: Reduction. Unpublished manuscript, UCLA.

Frege, Gottlob
1962 Funktion und Begriff [Function and Concept]. In *Funktion, Begriff, Bedeutung. Fünf logische Studien* [Function, Concept, Meaning. Five logical Studies], Günther Patzig (ed.), 17 – 39. Göttingen: Vandenhoeck & Ruprecht.

Frey, Werner
1993 *Syntaktische Bedingungen für die semantische Interpretation* [Syntactic Conditions for the Semantic Interpretation]. Number 35 in Studia Grammatica. Berlin: Akademie Verlag.

Fromkin, V. (ed.)
2000 *Linguistics: An Introduction to linguistic theory*. London: Blackwell.

Gamut, L. T. F.
1991a *Logic, Language and Meaning. Vol. 1: Introduction to Logic*. Chicago: The University of Chicago Press.
1991b *Logic, Language and Meaning. Vol. 2: Intensional Logic and Logical Grammar*. Chicago: The University of Chicago Press.

Gärdenfors, Peter
1988 *Knowledge in Flux*. Cambridge (Mass.): MIT Press.

Gazdar, Gerald, Klein, Ewan, Pullum, Geoffrey, and Sag, Ivan.
1985 *Generalized Phrase Structure Grammar*. London: Blackwell.

Gazdar, Gerald, Pullum, Geoffrey, Carpenter, Bob, Hukari, T., and Levine, R.
1988 Category structures. *Computational Linguistics* 14:1 – 19.

Geach, Peter
1972 A Program for Syntax. In *Semantics for Natural Language*, Donald Davidson and Gilbert Harman (eds.). (Synthese Library 40.) Dordrecht: Reidel.

Geller, M. M., and Harrison, M. A.
1977 On $LR(k)$ grammars and languages. *Theoretical Computer Science* 4:245 – 276.

Geurts, Bart
1998 Presupposition and Anaphors in Attitude Contexts. *Linguistics and Philosophy* 21:545 – 601.

Ginsburg, Seymour and Spanier, Edwin H.
1964 Bounded ALGOL–Like Languages. *Transactions of the American Mathematical Society* 113:333 – 368.
1966 Semigroups, Presburger Formulas, and Languages. *Pacific Journal of Mathematics* 16:285 – 296.

Ginsburg, Seymour
1975 *Algebraic and Automata–Theoretic Properties of Formal Languages*. Amsterdam: North–Holland.

Goldstern, Martin, and Judah, Haim
1995 *The Incompleteness Phenomenon*. Wellesley (Mass.): AK Peters.

Grätzer, George
1968 *Universal Algebra*. New York: van Nostrand.
1971 *Lattice Theory: First Concepts and Distributive Lattices*. Freeman.

Greibach, Sheila A.
1967 A new normal–form theorem for context–free phrase structure grammars. *Journal of the Association for Computing Machinery* 13:42 – 52.

Grewendorf, Günter, Hamm, Friedrich, and Sternefeld, Wolfgang
1987 *Sprachliches Wissen. Eine Einführung in moderne Theorien der grammatischen Beschreibung* [Knowledge of Language. An Introduction to Modern Theories of Grammatical Description]. Number 695 in suhrkamp taschenbuch wissenschaft. Frankfurt a.M.: Suhrkamp Verlag.

Groenink, Annius V.
1997a Mild context–sensitivity and tuple–based generalizations of context–grammar. *Linguistics and Philosophy* 20:607–636.
1997b *Surface without Structure. Word Order and Tractability Issues in Natural Language Analysis*. Ph. D. diss., University of Utrecht.

de Groote, Philippe
2001 Towards Abstract Categorial Grammars. In *Association for Computational Linguistics, 39th Annual Meeting and 10th Conference of the European Chapter*, 148 – 155, Toulouse.

Haider, Hubert
1991 Die menschliche Sprachfähigkeit — exaptiv und kognitiv opak [The human language faculty — exaptive and cognitively opaque]. *Kognitionswissenschaft* 2:11 – 26.
1993 *Deutsche Syntax — generativ. Vorstudien zur Theorie einer projektiven Grammatik* [German Syntax — generative. Preliminary studies towards a theory of projective grammar]. Tübingen: Gunter Narr Verlag.
1995 Downright down to the right. In *On Extraction and Extraposition*, U. Lutz and J. Pafel (eds.), 145 – 271. Amsterdam: John Benjamins.
1997 Extraposition. In *Rightward Movement*, D. Beerman, D. LeBlanc, and H. van Riemsdijk (eds.), 115 – 151. Amsterdam: John Benjamins.
2000 Branching and Discharge. In *Lexical Specification and Insertion*, Peter Coopmans, Martin Everaert, and Jane Grimshaw (eds.), 135 – 164. (Current Issues in Linguistic Theory 197.) Amsterdam: John Benjamins.

Halmos, Paul
1956 Homogeneous locally finite polyadic boolean algebras of infinite degree. *Fundamenta Mathematicae* 43:255 – 325.

Hamm, Friedrich, and van Lambalgen, Michiel
2003 Event Calculus, Nominalization and the Progressive. *Linguistics and Philosophy* 26:381 – 458.

Harkema, Henk
2001 A Characterization of Minimalist Languages. In *Logical Aspects of Computational Linguistics (LACL '01)*, Philippe de Groote, Glyn Morrill, and Christian Retoré (eds.), 193 – 211. (Lecture Notes in Artificial Intelligence 2099.) Berlin/New York: Springer.

Harris, Zellig S.
1963 *Structural Linguistics*. The University of Chicago Press.
1979 *Mathematical Structures of Language*. Huntington (NY): Robert E. Krieger Publishing Company.

Harrison, Michael A.
1978 *Introduction to Formal Language Theory*. Reading (Mass.): Addison Wesley.

Hausser, Roland R.
1984 *Surface Compositional Grammar*. München: Wilhelm Finck Verlag.

Heim, Irene
1983 On the projection problem for presuppositions. In *Proceedings of the 2nd West Coast Conference on Formal Linguistics*, M. Barlow and D. Flickinger, D. Westcoat (eds.), 114 – 126, Stanford University.

Hendriks, Herman
2001 Compositionality and Model–Theoretic Interpretation. *Journal of Logic, Language and Information* 10:29 – 48.

Henkin, Leon, Monk, Donald, and Tarski, Alfred
1971 *Cylindric Algebras. Part 1*. Studies in Logic and the Foundation of Mathematics 64. Amsterdam: North–Holland.

Hindley, J. R., Lercher, B., and Seldin, J. P.
1972 *Introduction to Combinatory Logic*. London Mathematical Society Lecture Notes 7. Oxford: Oxford University Press.

Hindley, J. R. and Longo, L.
1980 Lambda calculus models and extensionality. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik* 26:289 – 310.

Hintikka, Jaakko
1962 *Knowledge and Belief. An Introduction into the logic of the two notions*. Ithaca: Cornell University Press.

Hodges, Wilfrid
2001 Formal features of compositionality. *Journal of Logic, Language and Information* 10:7 – 28.

Hopcroft, John E., and Ullman, Jeffrey D.
1969 *Formal Languages and their Relation to Automata*. Reading (Mass.): Addison Wesley.

van der Hulst, Harry
1984 *Syllable Structure and Stress in Dutch*. Dordrecht: Foris.

Huybregts, Riny
1984 Overlapping Dependencies in Dutch. *Utrecht Working Papers in Linguistics* 1:3 – 40.

IPA
1999 *Handbook of the International Phonetic Association*. Cambridge: Cambridge University Press.

Jackendoff, Ray
1977 *$\overline{X}$–Syntax: A Study of Phrase Structure*. Linguistic Inquiry Monographs 2. Cambridge (Mass.): MIT Press.

Jacobson, Pauline
1999 Toward a Variable–Free semantics. *Linguistics and Philosophy* 22:117 – 184.

2002 The (Dis)Organisation of the Grammar: 25 Years. *Linguistics and Philosophy* 25:601 – 626.

Johnson, J. S.
1969 Nonfinitizability of classes of representable polyadic algebras. *Journal of Symbolic Logic* 34:344 – 352.

Johnson, Mark
1988 *Attribute–Value Logic and the Theory of Grammar*. CSLI Lecture Notes 16. Stanford: CSLI.

Jones, Burton W.
1955 *The Theory of Numbers*. New York: Holt, Rinehart and Winston.

Joshi, Aravind, Levy, Leon S., and Takahashi, Masako
1975 Tree Adjunct Grammars. *Journal of Computer and System Sciences* 10:136 – 163.

Joshi, Aravind K.
1985 Tree adjoining grammars: How much context–sensitivity is required to provide reasonable structural descriptions? In *Natural Language Parsing. Psychological, Computational, and Theoretical Perspectives*, David Dowty, Lauri Karttunen, and Arnold Zwicky (eds.), 206–250. Cambridge: Cambridge University Press.

Just, Winfried and Weese, Martin
1996 *Discovering Modern Set Theory. Vol. I: The Basics*. Graduate Studies in Mathematics 8. AMS.
1997 *Discovering Modern Set Theory. Vol. II: Set–Theoretic Tools for Every Mathematician*. Graduate Studies in Mathematics 18. AMS.

Kac, Michael B., Manaster–Ramer, Alexis, and Rounds, William C.
1987 Simultaneous–Distributive Coordination and Context–Freeness. *Computational Linguistics* 13:25 – 30.

Kaplan, Ron M., and Kay, Martin
1994 Regular Models of Phonological Rule Systems. *Computational Linguistics* 20:331 – 378.

Karttunen, Lauri
1974 Presuppositions and linguistic context. *Theoretical Linguistics* 1:181 – 194.

Kasami, Tadao, Seki, Hiroyuki, and Fujii, Mamoru
1987 Generalized context–free grammars, multiple context–free grammars and head grammars. Technical report, Osaka University.

Kasami, Tadao
1965 An efficient recognition and syntax–analysis algorithm for context–free languages. Technical report, Air Force Cambridge Research Laboratory, Bedford (Mass.). Science Report AFCRL–65–758.

Kayne, Richard S.
1994 *The Antisymmetry of Syntax*. Linguistic Inquiry Monographs 25. Cambridge (Mass.): MIT Press.

Keenan, Edward L., and Faltz, Leonard L.
1985 *Boolean Semantics for Natural Language*. Dordrecht: Reidel.

Keenan, Edward L., and Westerståhl, Dag
1997 Generalized quantifiers. In *Handbook of Logic and Language*, Johan van Benthem and Alice ter Meulen (eds.), 835 – 893. Amsterdam: Elsevier.

Kempson, Ruth
1975 *Presupposition and the delimitation of semantics*. Cambridge: Cambridge University Press.

Kleene, Stephen C.
1956 Representation of events in nerve nets. In *Automata Studies*, C. E. Shannon and J. McCarthy (eds.), 3 – 40. Princeton: Princeton University Press.

Knuth, Donald
1956 On the translation of languages from left to right. *Information and Control* 8:607 – 639.

Koskenniemi, Kimmo
1983 Two–level morphology. A general computational model for word–form recognition. Technical Report 11, Department of General Linguistics, University of Helsinki.

Koster, Jan
1986 *Domains and Dynasties: The Radical Autonomy of Syntax*. Dordrecht: Foris.

Koymans, J. P. C.
1982 Models of the Lambda Calculus. *Information and Control* 52:306 – 332.

Kracht, Marcus
1993 Mathematical aspects of command relations. In *Proceedings of the EACL 93*, 241 – 250.
1994 Logic and Control: How They Determine the Behaviour of Presuppositions. In *Logic and Information Flow*, Jan van Eijck and Albert Visser (eds.), 88 – 111. Cambridge (Mass.): MIT Press.

1995a Is there a genuine modal perspective on feature structures? *Linguistics and Philosophy* 18:401 – 458.

1995b Syntactic Codes and Grammar Refinement. *Journal of Logic, Language and Information* 4:41 – 60.

1997 Inessential Features. In *Logical Aspects of Computational Linguistics (LACL '96)*, Christian Retoré (ed.), 43 – 62. (Lecture Notes in Artificial Intelligence 1328.) Berlin/New York: Springer.

1998 Adjunction Structures and Syntactic Domains. In *The Mathematics of Sentence Structure. Trees and Their Logics*, Uwe Mönnich and Hans-Peter Kolb (eds.), 259 – 299. (Studies in Generative Grammar 44.) Berlin: Mouton de Gruyter.

1999 *Tools and Techniques in Modal Logic*. Studies in Logic and the Foundations of Mathematics 142. Amsterdam: Elsevier.

2001a Modal Logics That Need Very Large Frames. *Notre Dame Journal of Formal Logic* 42:141 – 173.

2001b Syntax in Chains. *Linguistics and Philosophy* 24:467 – 529.

2003 Against the feature bundle theory of case. In *New Perspectives on Case Theory*, Eleonore Brandner and Heike Zinsmeister (eds.), 165 – 190. Stanford: CSLI.

Kuroda, S. Y.

1964 Classes of languages and linear bounded automata. *Information and Control* 7:207 – 223.

Lamb, Sydney M.

1966 *Outline of Stratificational Grammar*. Washington: Georgetown University Press.

Lambek, Joachim

1958 The Mathematics of Sentence Structure. *The American Mathematical Monthly* 65:154 – 169.

Landweber, Peter S.

1963 Three theorems on phrase structure grammars of type 1. *Information and Control* 6:131 – 137.

Langholm, Tore

2001 A Descriptive Characterisation of Indexed Grammars. *Grammars* 4:205 – 262.

Lehmann, Winfred P.

1993 *Theoretical Bases of Indo–European Linguistics*. London: Routledge.

Leibniz, Gottfried Wilhelm

2000 *Die Grundlagen des logischen Kalküls (Lateinisch–Deutsch)* [The Foundations of the Logical Calculus (Latin–German)]. Philosophische Bibliothek 525. Hamburg: Meiner Verlag.

Levelt, Willem P.
1991 *Speaking. From Intention to Articulation*. 2nd ed. Cambridge (Mass.): MIT Press.

Lyons, John
1968 *Introduction to Theoretical Linguistics*. Cambridge: Cambridge University Press.
1978 *Semantics. Vol. 1*. Cambridge: Cambridge University Press.

Manaster–Ramer, Alexis, and Kac, Michael B.
1990 The Concept of Phrase Structure. *Linguistics and Philosophy* 13:325 – 362.

Manaster–Ramer, Alexis, Moshier, M. Andrew, and Zeitman, R. Suzanne
1992 An Extension of Ogden's Lemma. Manuscript. Wayne State University, 1992.

Manaster–Ramer, Alexis
1986 Copying in natural languages, context–freeness and queue grammars. In *Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics*, 85 – 89.

Markov, A. A.
1947 On the impossibility of certain algorithms in the theory of associative systems (Russian). *Doklady Akadémii Nauk SSSR* 55:587 – 590.

Marsh, William, and Partee, Barbara H.
1987 How Non–Context Free is Variable Binding? In *The Formal Complexity of Natural Language*, Walter Savitch, Emmon Bach, William Marsch, and Gila Safran–Naveh (eds.), 369 – 386. Dordrecht: Reidel.

Mel'čuk, Igor
1988 *Dependency Syntax: Theory and Practice*. SUNY Linguistics Series. Albany: State University of New York Press.
2000 *Cours de Morphologie Générale* [General Morphology. A Coursebook]. Volume 1 – 5. Montréal: Les Presses de l'Université de Montréal.

Meyer, A. R.
1982 What is a model of the lambda calculus? *Information and Control* 52:87 – 122.

Michaelis, Jens, and Kracht, Marcus
1997 Semilinearity as a syntactic invariant. In *Logical Aspects of Computational Linguistics (LACL '96)*, Christian Retoré (ed.), 329 – 345. (Lecture Notes in Artificial Intelligence 1328.) Heidelberg: Springer.

Michaelis, Jens, and Wartena, Christian
1997 How linguistic constraints on movement conspire to yield languages analyzable with a restricted form of LIGs. In *Proceedings of the Conference on Formal Grammar (FG '97),* Aix en Provence, 158–168.
1999 LIGs with reduced derivation sets. In *Constraints and Resources in Natural Language Syntax and Semantics*, Gosse Bouma, Geert–Jan M. Kruijff, Erhard Hinrichs, and Richard T. Oehrle (eds.), 263–279. Stanford: CSLI.

Michaelis, Jens
2001a Derivational minimalism is mildly context–sensitive. In *Logical Aspects of Computational Linguistics (LACL '98)*, Michael Moortgat (ed.), 179 – 198. (Lecture Notes in Artificial Intelligence 2014.) Heidelberg: Springer.
2001b *On Formal Properties of Minimalist Grammars*. Ph. D. diss., Universität Potsdam.
2001c Transforming linear context–free rewriting systems into minimalist grammars. In *Logical Aspects of Computational Linguistics (LACL '01)*, Philippe de Groote, Glyn Morrill, and Christian Retoré (eds.), 228 – 244. (Lecture Notes in Artificial Intelligence 2099.) Heidelberg: Springer.

Miller, Philip H.
1991 Scandinavian Extraction Phenomena Revisited: Weak and Strong Generative Capacity. *Linguistics and Philosophy* 14:101 – 113.
1999 *Strong Generative Capacity. The Semantics of Linguistic Formalisms*. Stanford: CSLI.

Mitchell, J. C.
1990 Type systems for programming languages. In *Handbook of Theoretical Computer Science, Vol B. Formal Models and Semantics*, Jan van Leeuwen (ed.), 365 – 458. Amsterdam: Elsevier.

Monk, Donald J.
1969 Nonfinitizability of classes of representable cylindric algebras. *Journal of Symbolic Logic* 34:331 – 343.
1976 *Mathematical Logic*. Berlin, Heidelberg: Springer.

Mönnich, Uwe
1999 On cloning context–freeness. In *The Mathematics of Sentence Structure. Trees and their Logics*, Hans-Peter Kolb and Uwe Mönnich (eds.), 195 – 229. (Studies in Generative Grammar 44.) Berlin: Mouton de Gruyter.

Moschovakis, Yannis
1994 Sense and denotation as algorithm and value. In *Proceedings of the ASL Meeting 1990, Helsinki*, Juha Oikkonen and Jouko Väänänen (eds.), 210 – 249. (Lecture Notes in Logic 2.) Berlin and Heidelberg: Springer.

Myhill, John
1960 Linear bounded automata. Technical report, Wright–Patterson Air Force Base.

Ogden, R. W., Ross, R, J., and Winkelmann, K.
1985 An "Interchange Lemma" for Context Free Languages. *SIAM Journal of Computing* 14:410 – 415.

Ogden, R. W.
1968 A helpful result for proving inherent ambiguity. *Mathematical Systems Sciences* 2:191 – 194.

Ojeda, Almerindo E.
1988 A Linear Precedence Account of Cross–Serial Dependencies. *Linguistics and Philosophy* 11:457 – 492.

Pentus, Mati
1995 Models for the Lambek calculus. *Annals of Pure and Applied Logic* 75:179 – 213.
1997 Product–Free Lambek–Calculus and Context–Free Grammars. *Journal of Symbolic Logic* 62:648 – 660.

Peters, Stanley P., and Ritchie, R. W.
1971 On restricting the base component of transformational grammars. *Information and Control* 18:483 – 501.
1973 On the generative power of transformational grammars. *Information Sciences* 6:49 – 83.

Pigozzi, Don J., and Salibra, Antonino
1995 The Abstract Variable–Binding Calculus. *Studia Logica* 55:129 – 179.

Pigozzi, Don J.
1991 Fregean Algebraic Logic. In *Algebraic Logic*, Hajnal Andréka, Donald Monk, and István Németi (eds.), 475 – 504. (Colloquia Mathematica Societatis János Bolyai 54.), Budapest and Amsterdam: János Bolyai Matematikai Társulat and North–Holland.

Pogodalla, Sylvain
2001 *Réseaux de preuve et génération pour les grammaires de type logiques* [Proof nets and generation for type logical grammars]. Ph. D. diss., Institut National Polytechnique de Lorraine.

Pollard, Carl J., and Sag, Ivan
1987 *Information–Based Syntax and Semantics. Vol. 1*. CSLI Lecture Notes 13. Stanford: CSLI.
1994 *Head–Driven Phrase Structure Grammar*. Chicago: The University of Chicago Press.

Pollard, Carl J.
1984 *Generalized Phrase Structure Grammar, Head Grammars and Natural Language*. Ph. D. diss., Stanford University.

Port, R. F., and O'Dell, M. L.
1985 Neutralization of syllable–final voicing in German. Technical Report 4, Indiana University, Bloomington.

Post, Emil L.
1936 Finite combinatory processes – formulation. *Journal of Symbolic Logic* 1:103 – 105.
1943 Formal reductions of the combinatorial decision problem. *Americal Journal of Mathematics* 65:197 – 215.
1947 Recursive unsolvability of a problem of Thue. *Journal of Symbolic Logic* 11:1 – 11.

Postal, Paul
1964 *Constituent Structure: A Study of Contemporary Models of Syntax*. The Hague: Mouton.

Prucnal, T., and Wroński, A.
1974 An algebraic characterization of the notion of structural completeness. *Bulletin of Section Logic of the Polish Academy of Sciences* 3:20 – 33.

Quine, Willard van Orman
1960 Variables explained away. *Proceedings of American Philosophical Society* 104:343 – 347.

Radzinski, Daniel
1990 Unbounded Syntactic Copying in Mandarin Chinese. *Linguistics and Philosophy* 13:113 – 127, 1990.

Rambow, Owen
1994 *Formal and Computational Aspects of Natural Language Syntax*. Ph. D. diss., University of Pennsylvania.

Recanati, François
2000 *Oratio Obliqua, Oratio Recta. An Essay on Metarepresentation*. Cambridge (Mass.): MIT Press.

Roach, Kelly
1987 Formal properties of head grammars. In *Mathematics of Language*, Alexis Manaster–Ramer (ed.), 293–347. Amsterdam: John Benjamins.

Rogers, James
1994 *Studies in the Logic of Trees with Applications to Grammar Formalisms*. Ph. D. diss., University of Delaware, Department of Computer & Information Sciences.

Rounds, William C.
1988 LFP: A Logic for Linguistic Description and an Analysis of its Complexity. *Computational Linguistics* 14:1 – 9.

Russell, Bertrand
1905 On denoting. *Mind* 14:479 – 493.

Sain, Ildikó, and Thompson, Richard S.
1991 Finite Schema Axiomatization of Quasi–Polyadic Algebras. In *Algebraic Logic*, Hajnal Andréka, Donald Monk, and István Németi (eds.), 539 – 571. (Colloquia Mathematica Societatis János Bolyai 54.) Budapest and Amsterdam: János Bolyai Matematikai Társulat and North–Holland.

Salomaa, Arto K.
1973 *Formal Languages*. New York: Academic Press.

van der Sandt, Rob A.
1988 *Context and Presupposition*. London: Croom Helm.

de Saussure, Ferdinand
1965 *Course in General Linguistics*. Columbus: McGraw–Hill.

Sauvageot, Aurélien
1971 *L'Edification de la Langue Hongroise* [The building of the Hungarian language]. Paris: Klincksieck.

Schönfinkel, Moses
1924 Über die Bausteine der mathematischen Logik [On the building blocks of mathematical logic]. *Mathematische Annalen* 92:305 – 316.

Seki, Hiroyuki, Matsumura, Takashi, Fujii, Mamoru, and Kasami, Tadao.
1991 On multiple context–free grammars. *Theoretical Computer Science* 88:191 – 229.

Sestier, A.
1960 Contributions à une théorie ensembliste des classifications linguistiques [Contributions to a set–theoretical theory of classifications]. In *Actes du Ier Congrès de l'AFCAL*, 293 – 305, Grenoble.

Shieber, Stuart
1985 Evidence against the Context–Freeness of Natural Languages. *Linguistics and Philosophy* 8:333 – 343.

1992 *Constraint–Based Grammar Formalisms*. Cambridge (Mass.): MIT Press.

Smullyan, Raymond M.
1961 *Theory of Formal Systems*. Annals of Mathematics Studies 47. Princeton: Princeton University Press.

Staal, J. F.
1967 *Word Order in Sanskrit and Universal Grammar*. Foundations of Language, Supplementary Series No. 5. Dordrecht: Reidel.

Stabler, Edward P.
1997 Derivational Minimalism. In *Logical Aspects of Computational Linguistics (LACL '96)*, Christian Retoré (ed.), 68 – 95. (Lecture Notes in Artificial Intelligence 1328.) Heidelberg: Springer.

von Stechow, Arnim, and Sternefeld, Wolfgang
1987 *Bausteine syntaktischen Wissens. Ein Lehrbuch der generativen Grammatik* [Building blocks of syntactic knowledge. A textbook of generative grammar]. Opladen: Westdeutscher Verlag.

Steedman, Mark
1990 Gapping as constituent coordination. *Linguistics and Philosophy* 13:207 – 263.
1996 *Surface Structure and Interpretation*. Linguistic Inquiry Monographs 30. Cambridge (Mass.): MIT Press.

Tarski, Alfred
1983 The concept of truth in formalized languages. In *Logic, Semantics, Metamathematics*, J. Corcoran (ed.), 152 – 178. Indianapolis: Hackett Publishing.

Tesnière, Lucien
1982 *Eléments de syntaxe structurale* [Elements of structural syntax]. 4th ed. Paris: Klincksieck.

Thatcher, J. W., and Wright, J. B.
1968 Generalized finite automata theory with an application to a decision problem of second–order logic. *Mathematical Systems Theory* 2:57 – 81.

Thue, Axel
1914 Probleme über Veränderungen von Zeichenreihen nach gegebenen Regeln [Problems concerning changing strings according to given rules]. *Skrifter utgit av Videnskapsselkapet i Kristiania, I. Mathematisk–naturvidenskabelig klasse* 10.

Trakhténbrodt, B. A.
1950 Névozmožnost' algorifma dlá problémy razréšimosti na konéčnyh klassah [Impossibility of an algorithm for the decision problem of finite classes]. *Doklady Akadémii Nauk SSSR*, 569 – 572.

Turing, Alan M.
1936 On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society* 42:230 – 265.

Uszkoreit, Hans
1987 *Word Order and Constituent Structure in German*. CSLI Lecture Notes 8. Stanford: CSLI.

Vaught, Robert L.
1995 *Set Theory. An Introduction*. 2nd ed. Basel: Birkhäuser.

Frank Veltman.
1985 *Logics for Conditionals*. Ph. D. diss., Department of Philosophy, University of Amsterdam.

Vijay–Shanker, K., Weir, David J., and Joshi, Aravind K.
1986 Tree adjoining and head wrapping. In *Proceedings of the 11th International Conference on Computational Linguistics (COLING '86)*, Bonn, 202–207.
1987 Characterizing structural descriptions produced by various grammar formalisms. In *Proceedings of the 25th Meeting of the Association for Computational Linguistics (ACL '87)*, Stanford, CA, 104 – 111.

Villemonte de la Clergerie, Eric
2002a Parsing MCS Languages with Thread Automata. In *Proceedings of the Sixth International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+6),* Venezia.
2002b Parsing mildly context–sensitive languages with thread automata. In *Proceedings of the 19th International Conference on Computational Linguistics (COLING 02)*, Taipei.

Weir, David J.
1988 *Characterizing Mildly Context–Sensitive Grammar Formalisms*. Ph. D. diss., University of Pennsylvania, Philadelphia.

Younger, D. H.
1967 Recognition and parsing of context–free languages in time n^3. *Information and Control* 10:189 – 208.

Zadrozny, Wlodek
1994 From Compositional Semantics to Systematic Semantics. *Linguistics and Philosophy* 17:329 – 342.

Zimmermann, Thomas Ede
1999 Meaning Postulates and the Model–Theoretic Approach to Natural Language Semantics. *Linguistics and Philosophy* 22:529 – 561.