Chapter 2
String Languages

THIS chapter introduces the notion of a grammar as an algebra. We shall describe how context free grammars and adjunction grammars fit the format described here. Then we shall study syntactic categories as they arise implicitly in the formulation of a grammar and then turn to the relationship between languages, grammars and surface tests to establish structure. We shall meet our first principle: the *Principle of Preservation*.

2.1 Languages and Grammars

Languages in the way they appear to us seem to consist in strings. The text in front of you is an example. It is basically a long chain of symbols, put one after the other. Yet, linguists stress over and over that there is *structure* in this chain and that this structure comes from a *grammar* that generates this language. I shall assume that the reader is familiar with this standard view on language. In this chapter I shall rehearse some of the definitions, though taking a slightly different view. While standard syntactic textbooks write rules in the form of replacement rules (S $\rightarrow$ NP VP) to be thought of as replacing what is to the left by what is to the right, here we take a bottom up view: we define grammars as devices that *combine* expressions. The reasons for this shift have already been discussed; in the next chapter, it will become apparent why there is no alternative to this view. This is also the way in which Montague defined his formation rules.

Although I shall have very little to say about phonology I should make it clear that when I use the terms "alphabet" and "letter" you may replace them by "phoneme inventory" and "phoneme". Likewise, we may decide to include tone and other characteristics into the representation. All this can be done. The only reason that I do not do it is, apart from the fact that I am not a phonologist, that it would distract attention from the central issues. The reader is however asked to keep in mind that the discussion is largely independent of the actual nature and manifestation of the alphabet.

I said that languages are sets of strings. Clearly, there is more to languages, as they also give meanings to the strings. Yet, if we disregard this latter aspect—and maybe some more—we retain as the simplest of all manifestations of a language: that of a *set of strings*. The topic of string languages is very rich since it has been

M. Kracht, *Interpreted Languages and Compositionality*, Studies in Linguistics and Philosophy 89, DOI 10.1007/978-94-007-2108-1_2,
© Springer Science+Business Media B.V. 2011

thoroughly studied in formal language theory. We start therefore by discussing string languages.

Recall that a **string** over some alphabet A is a sequence of letters from A; for example, /abcbab/ is a string over $\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$. It is also a string over the alphabet $\{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}$ but not over $\{\mathtt{a}, \mathtt{b}\}$. Alternatively, a string over A is a function $\vec{x} : n \to A$ for some natural number n (see Appendix A); n is the **length** of $\vec{x}$. If $n = 0$ we get the **empty string**; it is denoted by ε. We write $\vec{x}, \vec{y}$ (with an arrow) for arbitrary strings. Concatenation is either denoted by $\vec{x}^\frown\vec{y}$ or by juxtaposition. In running text, to enhance explicitness, I enclose material strings (or exponents in general) in slashes, like this: /dog/. This carries no theoretical commitment of any sort.

Definition 2.1 Let A be a finite set, the so-called **alphabet**. A^* denotes the set of strings over A, A^+ the set of nonempty strings. A **language over** A is a subset of A^*.

Following Unix convention, we shall enclose names for sets of symbols by colons (for example, :digit:). This way they cannot be confused with sets of strings, for which we use ordinary notation.

Definition 2.2 The union of two sets is alternatively denoted by $S \cup T$ and $S \mid T$. Given two sets S and T we write

$$S \cdot T := \{\vec{x}^\frown\vec{y} : \vec{x} \in S, \vec{y} \in T\}. \tag{2.1}$$

Furthermore, S^n is defined inductively by

$$S^0 := \{\varepsilon\}, \quad S^{n+1} := S^n \cdot S. \tag{2.2}$$

Finally we put

$$S^* := \bigcup_{n\in\mathbb{N}} S^n \tag{2.3}$$

as well as

$$S^+ := S \cdot S^*. \tag{2.4}$$

Typically, we write ST in place of $S \cdot T$. * binds stronger than $\cdot$ and $\cdot$ binds stronger than $\cup$. We shall write $\{x\}$ and x indiscriminately in case x is a single letter.

It is important to note that a language as defined here is a *set*, so it is unstructured. A grammar on the other hand is a *description* or specification of a language. There are two types of grammars: descriptive and generative. Descriptive grammars describe the strings of the language, while generative grammars describe a process that generates them. We shall delay a definition of descriptive grammars. Thus, for now a grammar is a system of rules (or rather functions). It is the grammar that imposes structure on a language. This point seems contentious; in fact, many linguists think differently. They think that the language itself possesses a structure that needs to be described using the grammar. Some are convinced that some descriptions (maybe even a single one) are better than all the others (see Tomalin (2006) on the origin of this view). I consider this belief unfounded. That we know the right grammar when we see it is wishful thinking. It is clear that regularities need accounting for. However, that accounting for them in a particular way will make the rule apparatus more transparent needs to be demonstrated. The most blatant defect of such claims is that no one knows how to define simplicity in an unambiguous way. One exception is perhaps Kolmogorov complexity, which is, however, difficult to use in practice (see Kornai (2007) on that subject). In absence of a unique notion of simplicity we are left with the intuition that a language "calls" for a particular description in the form of a certain grammar. But it may well be that there are different descriptions of the same facts, none of which need to be essentially better than the other. Indeed, if one looks around and studies various frameworks and the way they like to deal with various phenomena, one finds that there is little fundamental consensus; nor is there a criterion by which to judge who is right. Thus, a language may possess various quite different grammars. These grammars in turn impose different structures on the language and it may be impossible to say which one is "correct". Thus a distinction must be made between the set of acceptable strings and the structure that we see in them.

Example 2.1 (See also Example 2.6 below.) The language of *unbracketed* additive arithmetical terms (or *ua-terms* for short) is defined as follows. Consider the set

$$\text{:digit:} := \{\texttt{0}, \texttt{1}, \cdots, \texttt{9}\}. \tag{2.5}$$

An **ua-term** is a string over this alphabet plus the additional symbol /+/ such that it neither ends nor begins with /+/. So it is a member of the following set.

$$\mathrm{UA} := \text{:digit:}^{+}(\texttt{+}\text{:digit:}^{+})^{*} \tag{2.6}$$

Examples are the following.

$$\texttt{0},\ \texttt{10},\ \texttt{010+7},\ \texttt{00+01+31},\ \texttt{1001+000+9} \tag{2.7}$$

In practice we think of such a string as consisting in blocks of digits separated by /+/. This is so far just a matter of convenience. We shall see below however what may justify this view.

In contrast to the unbracketed arithmetical terms, the bracketed arithmetical terms (**a-terms**) always have brackets. They are technically strings over a different alphabet, namely :digit: $\cup$ {+, `(`, `)`}. Thus, it is not that we do not write ua-terms with brackets; they do not contain any brackets in the first place. An a-term, by contrast, has them everywhere. (A precise definition of a-terms will be given in Example 2.6.) There are many ways to "analyse" a given ua-term as arising from some a-term. For example, we can think of the ua-term

$$\vec{x}_0\texttt{+}\vec{x}_1\texttt{+}\vec{x}_2\texttt{+}\cdots\texttt{+}\vec{x}_n \tag{2.8}$$

as being derived, among others, in a left bracketed (2.9) or a right bracketed (2.10) way:

$$(\vec{x}_0+(\vec{x}_1+(\vec{x}_2+\cdots(\vec{x}_{n-1}+\vec{x}_n)\cdots))) \tag{2.9}$$

$$((\cdots((\vec{x}_0+\vec{x}_1)+\vec{x}_2)+\cdots\vec{x}_{n-1})+\vec{x}_n) \tag{2.10}$$

Similarly, the ua-term

$$\texttt{3+1+7+5} \tag{2.11}$$

can be derived from the following a-terms by deleting brackets:

$$\texttt{(((3+1)+7)+5), ((3+(1+7))+5), (3+((1+7)+5)), (3+(1+(7+5))).} \tag{2.12}$$

There is no way to decide which analysis is correct. ⚽

Example 2.2 The formation of the third singular present of the English verb is identical to the plural of nouns. It consists—irregular forms and idiosyncrasies of spelling aside—in the addition of /s/, /es/ or /ses/, depending on the end of the word. Is there a formal identity between the two or do they just accidentally happen to be the same? ⚽

This last example will be important also when discussing identity of modes in Section 3.1. Let me briefly go into some details. Ordinary languages contain—apart from the obvious alphabetic characters—also punctuation marks; in addition to punctuation marks we find the digits and the blank, written here /␣/ throughout when quoting material language strings and, finally, some less obvious characters such as "newline" or "new paragraph". These should be counted into the alphabet A for the purposes of writing serious grammars for languages. There is, for example, a difference in English between /black␣bird/ and /blackbird/. In written English the only difference is the presence or absence of the blank; in spoken English this comes out as a different stress assignment. The same goes obviously for punctuation (the difference between restrictive and nonrestrictive relative clauses is signalled by the presence of a comma). Spoken language has intonation, which is absent from written language; punctuation is a partial substitute for intonation. In what is to follow, we will concentrate on written language to avoid having to deal with issues that are irrelevant for the purpose of this book. Writing systems however introduce their own problems. For matters concerning the intricacies of alphabets I refer the reader to Korpela (2006).

Intermission 1 Some interesting facts about punctuation. In general, there is something of a syntax of punctuation marks. Writing no blank is different from writing one blank, while one blank is the same as two (consecutive) blanks. Two periods are likewise the same as one, two commas the same as one and so on. In general, punctuation marks act as separators, not as brackets. This means that they avoid being put in sequence (with minor exceptions such as a period and a comma when the period signals an abbreviation). Separators come in different strengths. For example, a period is a stronger separator than a comma. This means that if a period and a comma will be in competition, the (sentence) period will win. ⚽

Anyone who is nowadays dealing with characters will know that there is a lot of structure in an alphabet, just as the set of phonemes of a language is highly structured. There is first and foremost a division into alphabetic characters, digits, and punctuation marks. However, there is an additional division into such characters that serve as separators and those that do not. Separators are there to define the units ("identifiers" or "words"). For ua-terms, /+/ is a separator. Separators could also be strings, of course. If we want to understand where the words are in a text we break a string at all those positions where we find a separator. Thus, the blank and also punctuation marks are typical separators. But this is not always the case. A hyphen, for example, is a punctuation mark but does not serve as a separator—or at least not always. In programming languages, brackets are separators; this means that the name of a variable may not contain brackets, since they would simply not be recognised as parts of the name. Anyone interested in these questions may consult, for example, books or manuals on regular expressions and search patterns.

While we often think of languages as being sets of strings over a given alphabet, there are occasions when we prefer to think of languages as somehow independent of the alphabet. These viewpoints are not easy to reconcile. We can introduce some abstractness as follows. Let A and B be alphabets and $m : A \to B^*$ a map. m induces a **homomorphism** $\overline{m} : A^* \to B^*$ in the following way.

$$\overline{m}(x_0x_1\cdots x_{n-1}) := m(x_0)^\frown m(x_1)^\frown\cdots^\frown m(x_{n-1}) \tag{2.13}$$

Then $\overline{m}[L]$ is the **realphabetization** of L.

Example 2.3 In German, the **umlaut** refers to the change of /a/, /o/ and /u/ to /ä/, /ö/ and /ü/, respectively. Standard German allows to replace the vowels with dots by a combination of the vowel with /e/ (historically, this is where the dots came from; they are the remnants of an /e/ written above the vowel). So, we have a map m : ä $\mapsto$ ae, ö $\mapsto$ oe, ü $\mapsto$ ue. For all other (small) letters, $m(x) = x$. Hence,

$$\overline{m}(\texttt{Rädelsführer}) = \texttt{Raedelsfuehrer} \tag{2.14}$$

⚽

We now say that we look at a language only up to realphabetization. In linguistics this is done by considering spoken language as primary and all written languages as realphabetizations thereof. Usually we will want to require that $\overline{m}$ is injective on L, but spelling reforms are not always like that. In Switzerland, the letter /ß/ is written /ss/, and this obliterates the contrast between /Maße/ "measures" and /Masse/ "mass". For this reason we shall not deal with realphabetisation except for theoretical purposes, where we do require that $\overline{m}$ be injective. Realphabetizations are not structurally innocent. What is segmentable in one alphabet may not be in another. Imagine an alphabet where /`downtown`/ is rendered by a single letter, say, /⌂/. The map sending /⌂/ to /`downtown`/ makes an indecomposable unit decomposable (/`down`/ + /`town`/). The dependency of the analysis on the alphabet is mostly left implicit throughout this work.

The division into units, which is so important in practical applications (witness the now popular art of tokenisation), is from a theoretical standpoint secondary. That is to say, it is the responsibility of a grammar to tell us what the units are and how to find them. Whether or not a symbol is a separator will be a consequence of the way the grammar works, not primarily of the language itself. This is why we may maintain, at least in the beginning, that the alphabet is an unstructured set in addition to the language. The structure that we see in a language and its alphabet is—as I emphasized above—imposed on it by a system of rules and descriptions, in other words *by a grammar*. This applies of course to phonemes and features in the same way.

In my view, a grammar is basically an interpretation of an abstract language. In computer science one often talks about **abstract** and **concrete syntax**. The abstract syntax talks about the ideal constitution of the syntactic items, while the concrete syntax specifies how the items are communicated. The terminology used here is that of "signature" (abstract) versus "grammar" (concrete).

Definition 2.3 Let F be a set, the set of **function symbols**. A **signature** is a function Ω from F to the set $\mathbb{N}$ of natural numbers. Given f, $\Omega(f)$ is called the **arity** of f. f is a **constant** if $\Omega(f) = 0$.

If f has arity 2, for example, this means that f takes two arguments and yields a value. If f is a function on the set S, then $f : S \times S \to S$. We also write $f : S^2 \to S$. The result of applying f to the arguments x and y in this order is denoted by $f(x, y)$. If f is partial then $f(x, y)$ need not exist. In this case we write $f : S^2 \hookrightarrow S$. We mention a special case, namely $\Omega(f) = 0$. By convention, $f : S^0 \hookrightarrow S$, but there is little gain in allowing a zeroary function to be partial. Now, $S^0 = \{\varnothing\}$, and so f yields a single value if applied to $\varnothing$. However, $\varnothing$ is simply the empty tuple in this connection, and we would have to write $f()$ for the value of f. However, we shall normally write f in place of $f()$, treating f as if it was its own value. The 0-ary functions play a special role in this connection, since they shall form the *lexicon*.

Definition 2.4 A **grammar over** A is a pair $\langle \Omega, \mathcal{I} \rangle$, where Ω is a signature and for every $f \in F, \mathcal{I}(f) : (A^*)^{\Omega(f)} \hookrightarrow A^*$. F is the set of **modes** of the grammar. $\mathcal{I}$ is called the **interpretation**. If $\Omega(f) = 0$, f is called **lexical**, otherwise **nonlexical**. The set $\{\mathcal{I}(f) : \Omega(f) = 0\}$ is called the **lexicon** of G, and the set $\{\mathcal{I}(f) : \Omega(f) > 0\}$ the set of **rules**. The **language** generated by G, in symbols $L(G)$, is defined to be the least set S satisfying the following condition for every $f \in F$ and all $\vec{x}_i \in A^*$, $i < \Omega(f)$.

$$\begin{aligned} &\text{If for all } i < \Omega(f) \vec{x}_i \in S \text{ and } \mathcal{I}(f)(\vec{x}_0, \cdots, \vec{x}_{\Omega(f)-1}) \\ &\qquad\qquad \text{exists then } \mathcal{I}(f)(\vec{x}_0, \cdots, \vec{x}_{\Omega(f)-1}) \in S. \end{aligned} \tag{2.15}$$

Example 2.4 Let $F := \{j, t, f\}$, and $\Omega(j) = \Omega(t) = 0$, $\Omega(f) = 2$. $\mathcal{I}$ is defined as follows. $\mathcal{I}(j)$ is a zeroary function and so $\mathcal{I}(j)()$ is a string, the string /`John`/. Likewise, $\mathcal{I}(t)() =$ `talks`. Finally, we look at $\mathcal{I}(f)$. Suppose first that $\mathcal{I}(f)$ is interpreted like this.

$$\mathcal{I}(f)(\vec{x}, \vec{y}) := \vec{x}^\frown\text{␣}^\frown\vec{y}^\frown\texttt{.} \tag{2.16}$$

Then the language contains strings like this one:

$$\texttt{John␣talks.␣talks.} \tag{2.17}$$

The function $\mathcal{I}(f)$ needs to be constrained. One obvious way is to restrict the first input to /`John`/ and the second to /`talks`/. An indirect way to achieve the same is this definition.

$$\mathcal{I}(f)(\vec{x}, \vec{y}) := \begin{cases} \vec{x}^\frown\text{␣}^\frown\vec{y}^\frown\texttt{.} & \text{if } \vec{x} \text{ ends with /n/} \\ & \quad\text{and } \vec{y} \text{ begins with /t/,} \\ \text{undefined} & \text{otherwise.} \end{cases} \tag{2.18}$$

This grammar generates the following language.

$$\{\texttt{John}, \texttt{talks}, \texttt{John␣talks.}\} \tag{2.19}$$

⚽

Example 2.5 Here is now a pathological example. A set S is called **countable** if it is infinite and there is an onto function $f : \mathbb{N} \to S$. If S is countable we can assume that f is actually bijective. Let $L \subseteq A^*$. L is countable, since A is finite. Let $f : \mathbb{N} \to L$ be bijective. Let now $F := \{b, s\}$, $\Omega(b) := 0$ and $\Omega(s) := 1$. This means that we get the following terms: b, $s(b)$, $s(s(b))$, $s(s(s(b)))$, ... The general element has the form $s^n(b)$, $n \in \mathbb{N}$. This is a familiar way to generate the natural numbers: start with zero and keep forming successors. Further, we put

$$\begin{aligned} \mathcal{I}(b)() &:= f(0) \\ \mathcal{I}(s)(\vec{x}) &:= f(f^{-1}(\vec{x}) + 1) \end{aligned} \tag{2.20}$$

So, we start with the first element in the enumeration f. The number of $\vec{x}$ in the enumeration is $f^{-1}(\vec{x})$. If we add 1 to this number and translate this via f we get the next element in the list. In other words, we have $\mathcal{I}(s)(f(n)) = f(n + 1)$.

This grammar generates L. It follows that every countable language has a grammar that generates it. ⚽

Evidently, any $f \in F$ (that is, every mode) is either lexical or nonlexical. Notice that there are no requirements on the functions, not even that they be computable. (Recently, Lasersohn (2009) has argued that computability may not even be an appropriate requirement for meanings. Without endorsing the argument that he presents I have dropped the requirement here.) We shall introduce restrictions on the functions as we go along. The lexicon is not always considered part of the grammar. I make no principled decision here; it is just easier not to have to worry about the rules and the lexicon separately.

Example 2.6 This is one of our main examples: it will be called *the language of equations*.

$$\text{:eq:} := \text{:digit:} \cup \{\texttt{+}, \texttt{-}, \texttt{(}, \texttt{)}, \texttt{=}\} \tag{2.21}$$

$F = \{f_0, f_1, f_2, f_3, f_4, f_5, f_6\}$. $\Omega(f_0) = \Omega(f_1) = 0$, $\Omega(f_2) = \Omega(f_3) = 1$, $\Omega(f_4) = \Omega(f_5) = \Omega(f_6) = 2$. $\vec{x}$ is **binary** if it only contains /`0`/ and /`1`/; $\vec{x}$ is an a-term if it does not contain /=/. The modes are shown in Table 2.1. The strings that this grammar generates are of the following form. They are either strings consisting in the letters /`0`/ and /`1`/, for example /`010`/, /`11101`/, or they are a-terms, like /`(1+(01-101))`/; or they are equations between two a-terms, like /`(1+10)=11`/. (A single numeral expression is also an a-term.) ⚽

Given a signature Ω, we define the notion of an Ω*-term*.

Definition 2.5 Let V be a set of variables disjoint from F. Let Ω be a signature over F. An Ω**-term over** V is a string t over $F \cup V$ satisfying one of the following.

❶ $t \in V$,
❷ $t = f$, where $\Omega(f) = 0$,
❸ $t = f^\frown t_0^\frown \cdots ^\frown t_{n-1}$, where $n = \Omega(f)$ and t_i is an Ω-term for every $i < n$.

Table 2.1 The modes of Example 2.6

$$\mathcal{I}(f_0)() := \texttt{0}$$
$$\mathcal{I}(f_1)() := \texttt{1}$$
$$\mathcal{I}(f_2)(\vec{x}) := \begin{cases} \vec{x}^\frown\texttt{0} & \text{if } \vec{x} \text{ is binary,} \\ \text{undefined} & \text{else.} \end{cases}$$
$$\mathcal{I}(f_3)(\vec{x}) := \begin{cases} \vec{x}^\frown\texttt{1} & \text{if } \vec{x} \text{ is binary,} \\ \text{undefined} & \text{else.} \end{cases}$$
$$\mathcal{I}(f_4)(\vec{x}, \vec{y}) := \begin{cases} \texttt{(}^\frown\vec{x}^\frown\texttt{+}^\frown\vec{y}^\frown\texttt{)} & \text{if } \vec{x}, \vec{y} \text{ are a-terms,} \\ \text{undefined} & \text{else.} \end{cases}$$
$$\mathcal{I}(f_5)(\vec{x}, \vec{y}) := \begin{cases} \texttt{(}^\frown\vec{x}^\frown\texttt{-}^\frown\vec{y}^\frown\texttt{)} & \text{if } \vec{x}, \vec{y} \text{ are a-terms,} \\ \text{undefined} & \text{else.} \end{cases}$$
$$\mathcal{I}(f_6)(\vec{x}, \vec{y}) := \begin{cases} \vec{x}^\frown\texttt{=}^\frown\vec{y} & \text{if } \vec{x}, \vec{y} \text{ are a-terms,} \\ \text{undefined} & \text{else.} \end{cases}$$

The symbol $\mathrm{Tm}_\Omega(V)$ denotes the set of all Ω-terms over V. The set $\mathrm{Tm}_\Omega(\varnothing)$ is of special importance. It is the set of **constant Ω-terms**. A term t is **constant** if $t \in F^+$, that is, if it contains no variables. Given a grammar $G = \langle \Omega, \mathcal{I} \rangle$, we also call an Ω-term a G-**term**.

See Fig. 2.2 on page 36 for an example of term. Notice that the second case is a subcase of the third (where $n = 0$). It is listed separately for better understanding. Some remarks are in order. Standardly, terms are considered abstract, but I thought it easier to let terms also be concrete objects, namely strings. The syntax chosen for these objects is Polish Notation. It has the advantage of using the alphabet itself and having the property of transparency (see page 46 for a definition). Exercises 2.6 and 2.7 show that the language enjoys unique readability. Delaying the justification for the terminology, let us make the following definition.

Definition 2.6 Let t be an Ω-term. s is a **subterm** of t if and only if s is an Ω-term and a substring of t.

Based on the exercises at the end of this section one can show that the language of terms is quite well behaved. A substring that looks like a term actually is a subterm under every analysis. (Consequently there can be only one analysis.)

Proposition 2.1 *Let s and t be Ω-terms and s a substring of t. Then either $s = t$ or $t = f^\frown t_0^\frown \cdots {}^\frown t_{n-1}$ for some f and $n = \Omega(f)$ and there is an $i < n$ such that s is a subterm of t_i.*

Given a grammar G we can define the interpretation $\iota_G(t)$ of a constant term t.

❶ $\iota_G(f) := \mathcal{I}(f)$ if $\Omega(f) = 0$,
❷ $\iota_G(f t_0 \cdots t_{n-1}) := \mathcal{I}(f)(\iota_G(t_0), \cdots, \iota_G(t_{n-1}))$, where $n = \Omega(f)$.

We call ι_G the **unfolding function** and say that t **unfolds in** G **to** $\vec{x}$ if $\iota_G(t) = \vec{x}$. If the grammar is clear from the context, we shall write $\iota(t)$ in place of $\iota_G(t)$. Continuing our example, we have

$$\begin{aligned} \iota(f_4 f_3 f_0 f_2 f_1) &= \texttt{(}\iota(f_3 f_0)\texttt{+}\iota(f_2 f_1)\texttt{)} \\ &= \texttt{(}\iota(f_0)\texttt{1+}\iota(f_2 f_1)\texttt{)} \\ &= \texttt{(}\iota(f_0)\texttt{1+}\iota(f_1)\texttt{0)} \\ &= \texttt{(01+}\iota(f_1)\texttt{0)} \\ &= \texttt{(01+10)} \end{aligned} \tag{2.22}$$

This establishes the interpretation of constant terms. Since the string functions may be partial not every constant term has a value. Thus, $\iota(t)$ may be undefined. We call

$$\mathrm{dom}(\iota) := \{t \in \mathrm{Tm}_\Omega(\varnothing) : \iota(t) \text{ is defined}\} \tag{2.23}$$

the set of **orthographically definite terms**. The term $f_4 f_3 f_0 f_2 f_1$ is orthographically definite, while the term $f_6 f_6 f_0 f_1 f_1$ is not. This is because once f_6 has been used, it introduces the symbol /=/, and none of the modes can apply further. If t is orthographically definite, so is any subterm of t. Notice that for a grammar G, the language can simply be defined as

$$L(G) := \{\iota(t) : t \in \mathrm{Tm}_{\Omega}(\varnothing)\}. \tag{2.24}$$

Notice that this is different from the standard concept. This difference will be of great importance later on. Standardly, grammars may contain symbols other than the terminal symbols. The nonterminal alphabet contains characters foreign to the language itself. While in formal languages the presence of such characters can be motivated from considerations of usefulness, in our context these symbols make no sense. This is because we shall later consider interpreted languages; and there is, as far as I know, no indication that the nonterminal symbols have any meaning. In fact, in the terminology of this book, by the definition of "language" and "nonterminal symbol" the latter have no meaning. All of this will follow from the principles defined in Section 2.6. The present requirement is weaker since it does not constrain the power of the rules. What it says, though, is that the generation of strings must proceed strictly by using strings of the language itself. Later we shall also require that the strings must be used with the meaning that the language assigns to them.

If we eliminate nonterminal symbols, however, a lot of things change as well. $L(G)$ not only contains the strings at the end of a derivation but every string that is built on the way. If, for example, we write our grammar using context free rules, $L(G)$ not only contains the sentences but the individual words and all constituents that any sentence of $L(G)$ has. Therefore, unlike in traditional linguistic theory, L is not simply assumed to contain sentences but *all constituents*. To distinguish these two notions we shall talk of a **language in the narrow sense** if we mean language as a set of *sentences*; and we speak of a **language in the wide sense**—or simply of a **language**—if we mean language as a set of *constituents*. Notice that the difference lies merely in the way in which the language defines its grammar. As objects both are sets. But a language in the narrow sense leaves larger room to define grammars as languages in the wide sense also fix the set from which constituents may be drawn. Our stance in the matter is that one should start with language in the wide sense. The reasons for this will I hope become clear in Chapter 3. At this moment I would like to point out that for all intents and purposes starting with language in the narrow sense makes the grammar radically underdetermined.

For the working linguist, the choice of L is a highly empirical matter and hence full of problems: in defining L we need to make decisions as to what the constituents of the language are. This means we need more input in the first place. On the other hand, we get a more direct insight into structure. A grammar can only analyse a string into parts that are already members of L. Of course there is still a question of whether a given string really occurs as a constituent (we shall discuss that point later). But it can only do so if it is in L. A side effect of this is that we can sometimes know which occurrences of symbols are syncategorematic. Basically, an occurrence of a symbol is syncategorematic in a string under a derivation if it is not part of any primitive string that the derivation uses. This is admittedly vague; a proper definition must be deferred to Section 2.6.

Example 2.7 I give two alternative formulations of Boolean logic. The alphabet is as follows.

$$:\text{bool}: := \{\mathtt{0}, \mathtt{1}, \mathtt{p}, \neg, \wedge, \vee, \mathtt{(}, \mathtt{)}\} \tag{2.25}$$

The first language is the smallest set S satisfying the equation (here, as in the sequel, $\cdot$ binds stronger than | or $\cup$):

$$S = (\mathtt{p} \cdot (\mathtt{0} \mid \mathtt{1})^*) \cup (\mathtt{(} \cdot \neg \cdot S \cdot \mathtt{)}) \cup (\mathtt{(} \cdot S \cdot (\vee \mid \wedge) \cdot S \cdot \mathtt{)}) \tag{2.26}$$

The other language is the union $D \cup S$, where D and S are the minimal solution of the following set of equations:

$$\begin{aligned} D &= D \cup (\mathtt{0} \mid \mathtt{1}) \cdot D \\ S &= \mathtt{p} \cdot D \cup (\mathtt{(} \cdot \neg \cdot S \cdot \mathtt{)}) \cup (\mathtt{(} \cdot S \cdot (\vee \mid \wedge) \cdot S \cdot \mathtt{)}) \end{aligned} \tag{2.27}$$

It turns out that in both cases S is the same set; however, in the first example the language defined is just S, in the second it is $S \cup D$. S contains `p01`, `(¬p0)`, `(p1∧(¬p1))`. D (but not S) also contains `0`, `111`. ⚽

Given a grammar G and a string $\vec{x}$, we call a term t an **analysis term** or simply an **analysis** of $\vec{x}$ if $\iota(t) = \vec{x}$. A string may have several analysis terms. In this case we say that it is **ambiguous**. If it has none it is called **ungrammatical**. A *grammar* is called **ambiguous** if it generates at least one ambiguous string and **unambiguous** otherwise.

Exercise 2.1 Consider a context free grammar (for a definition, see Section 2.3). Then the language of that grammar generated in the narrow sense is context free, by definition. Show that also the language generated in the wide sense is context free.

Exercise 2.2 Give examples of pairs (L, L') such that L' is a language in the wide sense, and L its narrow restriction, such that (i) L is context free but L' is not, (ii) L' is context free but L is not.

Exercise 2.3 Describe the set of orthographically definite terms for the language of equations.

Exercise 2.4 Write grammars for the unbracketed additive terms, the left and the right bracketed additive terms of Example 2.1, respectively.

Exercise 2.5 Terms are strings, by definition, and can therefore be looked at as members of a language. The methodology of this book can therefore also be applied to them. Consider, by way of example, the strings for terms in Example 2.6. Write a grammar for the set of all constant Ω-terms; then write a grammar for the set of all orthographically definite terms.

Exercise 2.6 The formal notation of terms must be accompanied by a proof that it is uniquely readable. We shall use this and the next exercise to deliver such a proof. Recall that terms are sequences of function symbols, no extra symbol is added. However, not every such sequence is a term. Let Ω be a signature. For $f \in F \cup V$ let $\gamma(f) := \Omega(f) - 1$ and for a string $\vec{x} = x_0 x_1 \cdots x_{n-1} \in F^*$ let $\gamma(\vec{x}) = \sum_{i<n} \gamma(x_i)$. Show the following: if $\vec{x} \in F^*$ is a term, then (i) $\gamma(\vec{x}) = -1$ and (ii) for every proper prefix $\vec{y} = x_0 x_1 \cdots x_{m-1}$, $m < n$, $\gamma(\vec{y}) \geq 0$. (It follows from this that no proper prefix of a term is a term.) *Hint.* Do an induction on the length.

Exercise 2.7 (Continuing the previous exercise). Let $\vec{x} = x_0 x_1 \cdots x_{n-1} \in F^*$ be a string. Then if $\vec{x}$ satisfies (i) and (ii) from the previous exercise, $\vec{x}$ is a term. *Hint.* Induction on n. The cases $n = 0, 1$ are straightforward. Now suppose that $n > 1$. Then $x = x_0 x_1 \cdots x_{n-1}$ and $\gamma(x_0) = p \geq 0$, by (ii). Show that there is a number $i > 1$ such that $\gamma(x_1 \cdots x_{i-1}) = -1$; choose i minimal with that property. Then $\vec{y}_0 = x_1 \cdots x_{i-1}$ is a term, by inductive assumption. If $p > 1$ we have $i < n$, and there is $i' > i$ such that $\vec{y}_0 = x_i x_{i+1} \cdots x_{i'}$ is a term. Choose i' minimal with that property. And so on, getting a decomposition $x_0 \vec{y}_0 \cdots \vec{y}_p$.

Exercise 2.8 Show Proposition 2.1. *Hint.* Assume that $s \neq t$. Then there is a decomposition $t = f^\frown t_0^\frown \cdots ^\frown t_{n-1}$. Now fix a substring occurrence of s in t. Assume that it starts in t_i. Then show that it must also end in t_i.

2.2 Parts and Substitution

We defined a language (in the wide sense) to be the set of all of its constituent expressions. Since we do not discriminate sentences from nonsentences the language contains not only sentences but also parts thereof. We would like to be able to say of some expressions whether one is a part of the other. In particular, we would like to say that /`The cat is on the mat.`/ contains /`on the mat`/ as its part but not, for example, /`cat is on`/ or /`dog`/. In the cases just given this is straightforward: /`cat is on`/ is not in our language (in the wide sense), for it has no meaning; /`dog`/ on the other hand is not a string part of the expression. In other cases, however, matters are not so easy. Is /`Mary ran`/ a part of /`John and Mary ran.`/ or is it not? It is a string part of the sentence and it is meaningful. As it turns out, there is no unique answer in this case. (Curiously enough even semantic criteria fail to give a unanimous answer.) More problems arise, making the notion of *part* quite elusive. One problem is that there are no conditions on the string functions; another is that a given string may have been composed in many different ways. Let us discuss these issues below. We begin however with a definition of *part*.

Definition 2.7 Let G be a grammar. $\vec{x}$ is a **part of** $\vec{y}$ if there are constant terms s and u such that s is a subterm of u and $\iota_G(s) = \vec{x}$ as well as $\iota_G(u) = \vec{y}$.

This definition of *part of* pays no attention to the strings. Instead it looks at the way the strings are obtained through the string functions of the grammar. Thus, any useful restriction will come from restricting the power of string functions.

The definition also pays no attention to the way in which the parts occur in the larger string. Occurrences will be defined in Definition 2.9 and then we shall review Definition 2.7. The examples of this section will show how broad the spectrum of grammars is and how it affects parthood.

Example 2.8 Consider a unary function f which forms the past tense, for example $\mathcal{I}(f)(\texttt{go}) = \texttt{went}$, $\mathcal{I}(f)(\texttt{sing}) = \texttt{sang}$, $\mathcal{I}(f)(\texttt{ask}) = \texttt{asked}$. In this grammar, /`go`/ is a part of /`went`/, /`sing`/ a part of /`sang`/, /`ask`/ a part of /`asked`/. ⚽

Generally, it is actually not assumed that /`went`/ is literally made from /`go`/; rather, it is assumed that the verb /`go`/ possesses different allomorphs and that the context decides which of them is going to be used. This is also my own intuition. Therefore, I shall propose that syntactic functions may not delete material. This will effectively exclude the grammar of Example 2.8. Let us now look at a second example.

Example 2.9 We present two ways of generating the nonempty strings over the alphabet :blet: := $\{\texttt{a}, \texttt{b}\}$ of "binary letters". C_1 consists in the zeroary functions f_{a}, f_{b} plus the unary functions f_0 and f_1. We have

$$\begin{aligned}
\mathcal{I}_1(f_{\text{a}})() &:= \texttt{a}\\
\mathcal{I}_1(f_{\text{b}})() &:= \texttt{b}\\
\mathcal{I}_1(f_0)(\vec{x}) &:= \vec{x}^\frown\texttt{a}\\
\mathcal{I}_1(f_1)(\vec{x}) &:= \vec{x}^\frown\texttt{b}
\end{aligned} \tag{2.28}$$

So, $\iota_{C_1}(f_1 f_0 f_0 f_{\text{a}}) = \texttt{aaab}$. This grammar is the "typewriter model" of strings. Strings are generated by appending letters one by one to the initial letter.

The grammar C_2 has the zeroary function symbols f_{a} and f_{b} and a binary symbol γ. We have

$$\begin{aligned}
\mathcal{I}_2(f_{\text{a}})() &:= \texttt{a}\\
\mathcal{I}_2(f_{\text{b}})() &:= \texttt{b}\\
\mathcal{I}_2(\gamma)(\vec{x}, \vec{y}) &:= \vec{x}^\frown\vec{y}
\end{aligned} \tag{2.29}$$

For example, $\iota_{C_2}(\gamma\gamma f_{\text{a}} f_{\text{a}} \gamma f_{\text{a}} f_{\text{b}}) = \texttt{aaab}$.

In C_1, $\vec{x}$ is part of $\vec{y}$ if and only if it is a nonempty prefix of $\vec{y}$. In C_2, $\vec{x}$ is a part of $\vec{y}$ if and only if it is a nonempty subword. ⚽

It is to be noted that both grammars generate the set A^+, so they are extensionally identical. Yet structurally they are different. According to C_2 strings can have many more parts than according to C_1. For example, in C_2 /`aaab`/ possesses (apart from itself) the parts /`a`/, /`aa`/, /`aaa`/, /`b`/, /`ab`/, /`aab`/. In addition, the string /`aa`/ has two occurrences in /`aaab`/, which we may denote as follows: /$\underline{\texttt{aa}}\texttt{ab}$/ and /$\texttt{a}\underline{\texttt{aa}}\texttt{b}$/. (More on occurrences in Definition 2.9 and Section 2.5.) Both occurrences are actually parts of the string. It turns out, though, that not all parts can be parts in one and the same derivation. A more useful notion is in fact defined for a particular analysis term. The relation "is part of" is then the union of the relations "is a t-part of" for all terms t.

Definition 2.8 Let G be a grammar, t a constant term and $\vec{x}$ and $\vec{y}$ strings. We say that $\vec{x}$ is a **t-part of** $\vec{y}$ if $\iota_G(t) = \vec{y}$ and there is a subterm s of t such that $\iota_G(s) = \vec{x}$. In this case there is $t'(x)$ such that $t = t'(s)$.

With certain adaptations we can say that the relation "is a t-part of" is transitive. (If $\vec{y}$ is a t-part of $\vec{z}$ and is the unfolding of s, s a subterm of t, then parts of $\vec{y}$ must be s-parts of $\vec{y}$ in order to be t-parts of $\vec{z}$.) Here is a somewhat surprising result given that the union of transitive relations need not be transitive.

Proposition 2.2 *The relation* ***is part of*** *is transitive.*

Proof Let $\vec{x}$ be a part of $\vec{y}$ and $\vec{y}$ a part of $\vec{z}$. Then there are terms r and s such that r unfolds to $\vec{x}$ and s unfolds to $\vec{y}$ and r is a subterm of s. Furthermore there are t and u that unfold to $\vec{y}$ and $\vec{z}$, respectively, and t is a subterm of u. Since they unfold to the same string, we may replace t in u by s, giving us a new term u', of which s is a subterm. Since r is a subterm of s, it is also a subterm of u. □

Given a single C_2-term t for /`aaab`/, the substring occurrences that correspond to the subterms actually form a tree. This is essentially because the grammar encodes a context free analysis. However, C_2 is ambiguous: /`aaab`/ has several analysis terms and they provide different constituent analyses. The analysis terms are as follows: $\gamma\gamma\gamma f_a f_a f_a f_b$, $\gamma\gamma f_a \gamma f_a f_a f_b$, $\gamma\gamma f_a f_a \gamma f_a f_b$, $\gamma f_a \gamma\gamma f_a f_a f_b$ and $\gamma f_a \gamma f_a \gamma f_a f_b$. On the other hand, C_1 is unambiguous.

Standard tests for constituency in textbooks include the *substitution test*. Before we look in detail at the test let us first say a few words about string substitution.

Definition 2.9 A **(1-)context** is a pair $C = \langle \vec{x}, \vec{z} \rangle$ of strings. Inserting $\vec{y}$ into C results in the string $C(\vec{y}) := \vec{x}\,\vec{y}\,\vec{z}$. We say that $\vec{y}$ **occurs in** $\vec{u}$ if there is a context C such that $\vec{u} = C(\vec{y})$. We also say then that C is **an occurrence of** $\vec{y}$ **in** $\vec{u}$. The result of substituting $\vec{w}$ for $\vec{y}$ in its occurrence C is $C(\vec{w}) = \vec{x}\,\vec{w}\,\vec{z}$.

For example, $C := \langle \texttt{s}, \texttt{ish} \rangle$ is a 1-context. $C(\texttt{elf}) = \texttt{s}^\frown\texttt{elf}^\frown\texttt{ish} = \texttt{selfish}$. Notice that for any 1-context $C = \langle \vec{x}, \vec{y} \rangle$, $C(\varepsilon) = \vec{x}^\frown\vec{y}$. The substitution test runs as follows.

`John likes cricket.` (2.30)

Given a sentence, say, (2.30), we look for the string occurrences that can be substituted for /`cricket`/ such that the result is once again a sentence. These include /`chess`/, /`vegetables`/, /`his new home`/ and so on. Similarly, we try to substitute for other sequences such as /`likes`/, /`John likes`/ and /`likes cricket`/. The underlying idea is that nonconstituents cannot be substituted for (for example /`John likes`/) while constituents can. In practice, this test is not without problems, as it often turns out that nonconstituents can be substituted for (as is the case with /`John` `likes`/, for which we can substitute /`Peter dislikes`/). In fact, it sometimes turns out that the alleged nonconstituent passes all tests and we must be prepared to either strengthen our tests or admit that these really *are* constituents (as some claim is the case with /`John likes`/, see Steedman (1990)). In this section we shall look in some detail at the formal underpinnings of the substitution test.

First of all, we have to ask what we actually mean by substitution and second how it can possibly show us something about the grammars for our language. The answer to the first question is in fact not trivial. In the absence of a grammar the substitution we should be performing is simply *string substitution*. The underlying claim of the constituency test is that it shows us when string substitution is actually *constituent substitution*. This is the case if it can be performed without affecting grammaticality. Here I have *defined* constituent substitution to be substitution on the level of the analysis terms: it is the substitution of one subterm by another. The syntactic tests assume that constituent substitution if defined is always string substitution. This is problematic for two reasons. One is that the two need not be identical because the string functions of the grammar may be different from string polynomials (see the end of this section for a definition). The second is that the substitution can give misleading evidence. We start with some examples to show the point.

Definition 2.10 Let L be a language. Write $\vec{x} \sim_L \vec{y}$ if for all 1-contexts C: $C(\vec{x}) \in L \Leftrightarrow C(\vec{y}) \in L$. The set $\mathrm{Cat}_L(\vec{x}) := \{C : C(\vec{x}) \in L\}$ is called the **string category of $\vec{x}$ in L**.

Obviously, $\vec{x} \sim_L \vec{y}$ if and only if $\mathrm{Cat}_L(\vec{x}) = \mathrm{Cat}_L(\vec{y})$. If string substitution is constituent substitution then the definition above defines exactly the syntactically relevant classes of English. However, mostly this is not a realistic assumption. Let us review how the notion of part can depart from that of a constituent.

Example 2.10 We look at three grammars to form the plural of English. Let F_0 be a list of functions $f_{\vec{x}}$, where in all grammars below $f_{\vec{x}}$ will be evaluated to the string $\vec{x}$. To keep it simple, let $F_0 = F_R \cup F_I$, where $F_R = \{f_{\mathrm{cat}}, f_{\mathrm{dog}}\}$, $F_I = \{f_{\mathrm{sheep}}, f_{\mathrm{mouse}}, f_{\mathrm{ox}}\}$. F_R contains the regular nouns, F_I the irregular nouns. Thus, with $\mathcal{I}_i$ the interpretation function of the grammar P_i we have $\mathcal{I}_i(f_{\mathrm{cat}})() = \texttt{cat}$, $\mathcal{I}_i(f_{\mathrm{mouse}})() = \texttt{mouse}$ and so on. Now, put $\Omega_0(f_{\vec{x}}) := 0$. We denote by R_s the set $\{\vec{x} : f_{\vec{x}} \in F_R\}$, R_p the corresponding plural forms, likewise $I_s := \{\vec{x} : f_{\vec{x}} \in F_I\}$, I_p the corresponding plural forms.

The first grammar, $P_1 = \langle \Omega_1, \mathcal{I}_1 \rangle$, is as follows. $F_1 := F_0 \cup \{p\}$, $\Omega_1(p) = 1$, $\Omega_1 \restriction F_0 = \Omega_0$. $\mathcal{I}_1(p)$ is defined on $R_s \cup I_s$, which is all strings that are singular nouns (/`cat`/, /`mouse`/, /`ox`/, but not /`oxen`/) and its output is the corresponding plural. So we have

$$\mathcal{I}_1(p) = \{\langle \texttt{cat}, \texttt{cats}\rangle, \langle \texttt{dog}, \texttt{dogs}\rangle, \langle \texttt{sheep}, \texttt{sheep}\rangle, \langle \texttt{mouse}, \texttt{mice}\rangle, \langle \texttt{ox}, \texttt{oxen}\rangle\}. \tag{2.31}$$

The second grammar, $P_2 = \langle \Omega_2, \mathcal{I}_2 \rangle$, has instead $F_2 := F_0 \cup \{g, f_{\mathrm{mice}}, f_{\varepsilon}, f_{\mathrm{s}}, f_{\mathrm{es}}, f_{\mathrm{en}}\}$, where g is a binary function symbol. We put

$$\mathcal{I}_2(g)(\vec{x},\vec{y}) := \begin{cases} \vec{x}{}^\frown\vec{y} & \text{if } \vec{x} \in R_s \text{ and } \vec{y} = \mathtt{s} \\ & \quad \text{or } \vec{x} = \mathtt{sheep} \text{ and } \vec{y} = \varepsilon \\ & \quad \text{or } \vec{x} = \mathtt{ox} \text{ and } \vec{y} = \mathtt{en}, \\ \text{undefined} & \text{else.} \end{cases} \tag{2.32}$$

In short, $\mathcal{I}_2(g)(\vec{x},\vec{y})$ is defined only if $\vec{x}$ is a noun root and $\vec{y}$ a proper plural suffix for $\vec{x}$. Since the plural of /mouse/ is not obtained by affixation, it has been added to the lexicon. A variation of this grammar would be to set $\mathcal{I}_2(g)(\mathtt{mouse},\varepsilon) := \mathtt{mice}$. Thus, the plural is formed by a zero affix to a different stem.

The third grammar, $P_3 = \langle \Omega_3, \mathcal{I}_3\rangle$ is a mixture between the two. $F_3 := F_0 \cup \{p, g, f_\text{s}, f_\text{es}\}$. For regular nouns it uses g, for irregular nouns it uses f.

$$\mathcal{I}_3(g)(\vec{x},\vec{y}) = \begin{cases} \mathcal{I}_2(g)(\vec{x},\vec{y}) & \text{if } \vec{x} \in R_s, \\ \text{undefined} & \text{otherwise.} \end{cases} \tag{2.33}$$

$$\mathcal{I}_3(p)(\vec{x}) = \begin{cases} \mathcal{I}_1(p)(\vec{x}) & \text{if } \vec{x} \in I_s, \\ \text{undefined} & \text{otherwise.} \end{cases} \tag{2.34}$$

First of all notice that we can distinguish between these grammars in terms of the generated language. It turns out that P_1 generates all and only the singular and plural noun forms. P_2 in addition contains the plural morphs (like /s/, /es/, /en/ and ε). P_3 contains only the regular plural morphs and not ε, for example (though that depends on the exact distribution of the work between f and g). P_1 realizes a model called *item and process*, while P_2 realises a model called *item and arrangement* (see Matthews (1978) for a discussion of these models).

Next we need to look at how constituent substitution works in these examples. Here is an example: in P_2, the string /cats/ is the value of the term $gf_\text{cat}f_\text{s}$. Replace f_cat by f_dog and you get the term $gf_\text{dog}f_\text{s}$, which unfolds to /dogs/. Replace it by f_mouse and you get $gf_\text{mouse}f_\text{s}$, which is undefined. Similarly, replace f_s by f_en and you get $gf_\text{cat}f_\text{en}$, which is also undefined.

In P_2, the plural morph is a constituent, so it should be substitutable. Likewise, the root noun is a constituent, so we should be able to substitute for it. Sometimes we can successfully perform such a substitution, as certain nouns accept two plural endings: we have /formulas/ next to /formulae/. Most of the time the substitution will fail though. In P_1 on the other hand the substitution of the plural morph is illicit for a different reason: it is not a constituent. The form /cats/ is the value of pf_cat, so the only constituent substitution we can perform is to replace f_cat by f_mouse and in this case the result is /mice/.

In P_3 string substitution of the plural morph by something else is sometimes licit sometimes not. Let us look now at the substitution of the root noun by another root noun. In P_2 we may exchange /dog/ for /cat/ but not /mouse/. This is because $\mathcal{I}_2(g)(\mathtt{dog},\mathtt{s}) = \mathtt{dogs}$, which is the result of substituting the substring /cat/ of /cats/ by /dog/, but $\mathcal{I}_2(g)(\mathtt{mouse},\mathtt{s})$ is undefined, while applying the string substitution gives /mouses/. Trying the same in P_1 we find that the string substitution facts are similar; however, $\mathcal{I}_2(f)(\texttt{mouse})$ is defined and it gives /`mice`/. Thus, the difference between P_1 and P_2 is that the substitution of the subconstituent /`mouse`/ for /`cat`/ in the derivation is licit in P_1 but illicit in P_2. In P_1, the result of this substitution is different from string substitution, though. ⚽

The grammar P_2 actually uses straight concatenation and the string categories of English actually do tell us about the necessary distinctions we need to make in the paradigms. (Note though that the grammars here do not explicitly mention paradigms. There is no need to do so. The classes are just defined indirectly via the partiality.)

Example 2.11 The next example is a variation of the previous theme. The first grammar, Q_1, has constants for the names /`John`/, /`Alex`/ and /`Pete`/ and for the verb forms, /`sings`/ and /`sing`/, /`runs`/ and /`run`/. It has two binary functions, c and g. Call a sequence $\vec{x}$ an *NP* if it has the form $/x_1␣\texttt{and}␣x_2␣\texttt{and}␣x_3\cdots/$. It is *singular* if it does not contain /`and`/ and plural otherwise.

$$\mathcal{I}(c)(\vec{x},\vec{y}) := \begin{cases} \vec{x}^\frown␣\texttt{and}␣{}^\frown\vec{y} & \text{if } \vec{x} \text{ and } \vec{y} \text{ are NPs,} \\ \text{undefined} & \text{else.} \end{cases} \tag{2.35}$$

g combines NPs with verb forms. The chosen verb form must agree in number with the sequence. This is done as follows.

$$\mathcal{I}(g)(\vec{x},\vec{y}) := \begin{cases} \vec{x}^\frown␣^\frown\vec{y} & \text{if either } \vec{x} \text{ is a singular NP} \\ & \quad\text{and } \vec{y} \text{ is a singular verb form} \\ & \text{or } \vec{x} \text{ is a plural NP} \\ & \quad\text{and } \vec{y} \text{ is a plural verb form,} \\ \text{undefined} & \text{else.} \end{cases} \tag{2.36}$$

This grammar generates /`John sings`/ (it is the value of $gf_{\text{John}}f_{\text{sings}}$) and /`John and Mary and Alex sing`/ (the value of $gccf_{\text{John}}f_{\text{Mary}}f_{\text{Alex}}f_{\text{sing}}$) but not /`Mary and Alex sings`/. For the second grammar, Q_2, we assume we have only verb roots (form identical with the singulars of the previous grammar) and change the interpretation of g as follows:

$$\mathcal{K}(g)(\vec{x},\vec{y}) := \begin{cases} \vec{x}^\frown␣^\frown\vec{y} & \text{if } \vec{x} \text{ is a plural NP and } \vec{y} \text{ is a verb root,} \\ \vec{x}^\frown␣^\frown\vec{y}^\frown\texttt{s} & \text{if } \vec{x} \text{ is a singular NP and } \vec{y} \text{ is a verb root,} \\ \text{undefined} & \text{else.} \end{cases} \tag{2.37}$$

In Q_1, we can string substitute /`John and Mary`/ for /`John`/ only if the verb form is already plural but not, for example, in /`John sings`/, for we would get /`John and Mary sings`/, which the grammar does not generate. For the same reason it is also not possible to constituent substitute. In Q_2, the constituent substitution gives us different results. Namely, constituent substitution of /`John and Mary`/ for /`John`/

in /`John sings`/ yields /`John and Mary sing`/! This is because the sentence is the value (under $\mathcal{K}$) of $gf_{\text{John}}f_{\text{sing}}$, and we replace f_{John} by $cf_{\text{John}}f_{\text{Mary}}$. This yields the term $gcf_{\text{John}}f_{\text{Mary}}f_{\text{sing}}$, which unfolds to /`John and Mary sing`/. ⚽

The previous examples established two things: first, it may be the case that certain affixes are introduced by the derivation. In this case the string substitution has nothing to do with constituent substitution, since there is no constituent to begin with. Second, there is a difference between string substitution and constituent substitution. It is the latter notion that is dependent on the grammar. It is defined as follows.

We have seen in the previous section how to evaluate constant terms. Now we shall introduce variables over constituents. Thus, we shall allow to write fx and gxy but also $gfxx$, where f is unary and g binary and x and y are variables over terms. For terms containing such variables the interpretation must be a function from values of these variables to strings. Here is a way to implement this idea. The interpretation of a term is a partial function from $(A^*)^{\mathbb{N}}$ to A^*. Here, an infinite sequence $\overline{s} := \langle s_0, s_1, \cdots\rangle$ codes the assignment of strings to the variables that maps x_i to the string s_i. Now put

❶ $\iota_G(x_i)(\overline{s}) := s_i$,
❷ $\iota_G(f)(\overline{s}) := \mathcal{I}(f)$ if $\Omega(f) = 0$,
❸ $\iota_G(ft_0 \cdots t_{n-1})(\overline{s}) := \mathcal{I}(f)(\iota_G(t_0)(\overline{s}), \cdots, \iota_G(t_{n-1})(\overline{s}))$, where $n = \Omega(f)$.

Again, if G is clear from the context, ι_G will be simplified to ι. Notice that if the string functions are partial some of the $\iota_G(t)$ may also be partial functions. In the sequel I shall not use x_0 and x_1 but the usual x, y instead. (ι has been defined in Section 2.1 for constant terms slightly differently. On constant terms the valuation is irrelevant.)

Example 2.12 We continue Example 2.9. Apart from the constant, C_1 has only unary functions, so the terms we can create have at most one variable. Examples are f_1x_0, $f_0f_1x_1$, and so on. These describe functions from assignments to strings. The first defines a function from $\overline{s}$ to A^*: $\overline{s} \mapsto s_0{}^\frown\mathsf{a}$. The second is $\overline{s} \mapsto s_1{}^\frown\mathsf{b}$. I shall simplify this by eliminating reference to the entire valuation and replacing s_0 and s_1 by metavariables. This way we get the somewhat simpler expression $\vec{x} \mapsto \vec{x}{}^\frown\mathsf{a}$, $\vec{x} \mapsto \vec{x}{}^\frown\mathsf{b}$. It is possible to describe the totality of definable functions. They all have the form $\vec{x} \mapsto \vec{x}{}^\frown\vec{y}$ for some $\vec{y} \in A^*$ (which may be empty, since we generally also have the term x, which denotes the identity function on A^*).

C_2 has many more functions. In fact, the terms that we can define in C_2 are all the definable string polynomials using constants from A. ⚽

It is the simplifications of the preceding example that I shall adopt throughout. If we have a term $t(x, y, z)$ then the result of filling in the values $\vec{x}$, $\vec{y}$ and $\vec{z}$ for x, y and z, respectively, is denoted as usual by $t(\vec{x}, \vec{y}, \vec{z})$, or—if we want to be very explicit which value is assigned to which variable—by $t(x, y, z)[\vec{x}/x, \vec{y}/y, \vec{z}/z]$. The latter notation is more practical when we suppress the variables in the term itself by writing $t[\vec{x}/x, \vec{y}/y, \vec{z}/z]$. Now let $f : (A^*)^n \to A^*$. Say that it is a **term function** of G if there is a term $t(x_0, x_1, \cdots, x_{n-1})$ such that

$$f(\vec{x}_0, \cdots, \vec{x}_{n-1}) = \iota_G(t[\vec{x}_0/x_0, \cdots, \vec{x}_{n-1}/x_{n-1}]). \tag{2.38}$$

A **polynomial** (over A) is a term in the signature expanded by f_a (with value a) for every $a \in A$. f is a **polynomial function** of G if there is a polynomial $p(x_0, x_1, \cdots, x_{n-1})$ such that

$$f(\vec{x}_0, \cdots, \vec{x}_{n-1}) = \iota_G(p[\vec{x}_0/x_0, \cdots, \vec{x}_{n-1}/x_{n-1}]). \tag{2.39}$$

A particular sort of polynomial is the **string polynomial**. Let A be an alphabet. Then the string polynomials over A are the polynomials defined over the signature $\Omega : \cdot \mapsto 2, \varepsilon \mapsto 0$ in the algebra $\langle A^*, \varepsilon, ^\frown \rangle$. The interpretation is fixed: $\cdot$ is interpreted by concatenation, ε by the empty string and a by a constant yielding the letter a itself. (Bracketing is therefore eliminable since string concatenation is associative.) For example, $p(x_0, x_1) := x_1 \cdot \mathtt{a} \cdot x_1 \cdot x_0 \cdot \mathtt{b}$ is a polynomial. It is interpreted by the following function over A^*.

$$p^{A^*}(\vec{x}, \vec{y}) := \iota_G(t[\vec{x}/x_0, \vec{y}/x_1]) := \vec{y}^\frown \mathtt{a}^\frown \vec{y}^\frown \vec{x}^\frown \mathtt{b} \tag{2.40}$$

Typically, we do not even write the dot, so that $x_0 \cdot x_1$ reduces to $x_0 x_1$.

I close this section with an observation concerning the method of substitution, using Definition 2.10. This test is supposed to reveal something about the structure of the language provided that the grammar for it is some constituent grammar: parts are assumed to be substrings. (If the grammar is not of that form, another form of test is needed.) There are two ways to understand this test, ultimately deriving from two different definitions of language; one is to start with a language as the set of sentences and try to define the constituents smaller than sentences via substitution classes. Another, less ambitious method, starts with a language in the wide sense and tries to find out the constituent *occurrences* in a given string. We shall look here at the first of these interpretations; the other interpretation shall be looked at in more detail later.

Let $L \subseteq A^*$ be a language in the narrow sense and $\vec{x}$ a string. Evidently, there are two cases. Either $\vec{x}$ is not a substring of any string in L, and so $\mathrm{Cat}_L(\vec{x}) = \varnothing$, or it is and then $\mathrm{Cat}_L(\vec{x}) \neq \varnothing$. Apart from this there is nothing of substance one can say about the distribution of categories. There is no theoretical instrument to tell us from the substitution possibilities which are the constituents. This is reflected also in some grammars. In the Lambek Calculus all substrings of a string of the language are given a category.

There is a little bit that we can say about the relationship between the number of categories and L itself. It turns out that if the set of string categories is finite the language is regular. The following proof is based on the Myhill-Nerode Theorem (see Harrison (1978) for a proof).

Theorem 2.1 *A language has finitely many string categories if and only if it is regular.*

Proof Suppose that L has finitely many categories. Intersect the categories with the set $\{\langle\varepsilon, \vec{x}\rangle : \vec{x} \in A^*\}$. This yields a finite set of occurrences of prefixes. By the Myhill-Nerode Theorem, the language is regular. Now assume that the language is regular, and accepted by a finite automaton $\mathfrak{A}$. Let I_i be the language of all strings that lead from the initial state to state i; and let A_j be the language of all strings that lead from j to some accepting state. Then the categories coincide with the sets of pairs $I_i \times A_j$ for all states i and j such that j can be reached from i. □

Exercise 2.9 Describe all unary term functions of C_2, that is, all actions of C_2-terms in one variable.

Exercise 2.10 Verify that the language of ua-terms is defined by the following grammar:

$$\begin{aligned}
\mathcal{I}(n_0)() &:= \mathtt{0} \\
&\cdots\cdots \\
\mathcal{I}(n_9)() &:= \mathtt{9} \\
\mathcal{I}(c_0)(\vec{x}) &:= \vec{x}^\frown\mathtt{0} \\
&\cdots\cdots \\
\mathcal{I}(c_9)(\vec{x}) &:= \vec{x}^\frown\mathtt{9} \\
\mathcal{I}(a_0)(\vec{x}) &:= \vec{x}^\frown\mathtt{+}^\frown\mathtt{0} \\
&\cdots\cdots \\
\mathcal{I}(a_9)(\vec{x}) &:= \vec{x}^\frown\mathtt{+}^\frown\mathtt{9}
\end{aligned} \tag{2.41}$$

Exercise 2.11 (Continuing the previous exercise.) In the grammar of the previous exercise /10+1/ is a part of /10+12/. Simply choose the analysis $n_1c_0a_1c_2$. However, /12/ is not a part of /10+12/ although intuitively it should be. Begin by specifying when a given string is a substring of another. Then write a grammar where only those substring occurrences are parts that should be.

Exercise 2.12 The language of ua-terms is regular. Nevertheless, show that there is no regular grammar that generates exactly this language in the wide sense; this means that L is taken to be the union of all expressions that belong to some nonterminal of the grammar. *Hint.* Regular grammars allow to add only one symbol at a time.

2.3 Grammars and String Categories

In the previous section we looked at string categories defined by replacing substrings by other substrings. In this section we look at a similar but different definition where replacement is done only of constituent occurrences. This definition presupposes a grammar.

Definition 2.11 Let G be a grammar and $\vec{x}, \vec{y} \in L(G)$. We write $\vec{x} \sim_G \vec{y}$ if for every term $t(x_0)$, $\iota_G(t(\vec{x}))$ is defined if and only if $\iota_G(t(\vec{y}))$ is defined. We write $[\vec{x}]_G := \{\vec{y} : \vec{x} \sim_G \vec{y}\}$. These sets are called the **syntactic categories** of G.

We have restricted the definition to strings in $L(G)$. Thus, categories are defined only on the strings of the language. Strings outside the language have no category. An alternative formulation is this: $\vec{x}$ and $\vec{y}$ have the same category if for every pair of terms s_0 and s_1 that unfold to $\vec{x}$ and $\vec{y}$ respectively, $t(s_0)$ is orthographically definite if and only if $t(s_1)$ is. (It is easy to see that if this holds for one pair of terms s_0 and s_1 then it holds for all. See also Definition 2.23.)

Notice that the set of strings on which *no* function is defined is also a syntactic category. For example, in Example 2.1 this category is empty, in Example 2.6 it contains all equations.

There need not be finitely many equivalence classes as the following example shows.

Example 2.13 Let $A := \{\mathsf{a}\}$. $G = \langle \Omega, \mathcal{I}\rangle$ is defined by $\Omega(e) = 0$, $\Omega(f) = \Omega(g) = 1$ and

$$\begin{aligned} \mathcal{I}(e)() &:= \varepsilon \\ \mathcal{I}(f)(\mathsf{a}^n) &:= \begin{cases} \mathsf{a}^{n-1} & \text{if } n > 0, \\ \text{undefined} & \text{else.} \end{cases} \\ \mathcal{I}(g)(\mathsf{a}^n) &:= \mathsf{a}^{2n} \end{aligned} \tag{2.42}$$

G generates a^* in a somewhat unconventional way. Namely if $m > n$, then $\mathcal{I}(f)^n(\mathsf{a}^m) = \mathsf{a}^{m-n}$ and $\mathcal{I}(f)^m(\mathsf{a}^m) = \varepsilon$. However, for $n > m$, $\mathcal{I}(f)^n(\mathsf{a}^m)$ is undefined. Thus, $\mathsf{a}^m \sim_G \mathsf{a}^n$ if and only if $m = n$, and so there are infinitely many equivalence classes.

Now, the grammar $H = \langle \Omega', \mathcal{J}\rangle$ with $F' := \{e, h\}$ where $\mathcal{J}(e)() := \varepsilon$ and $\mathcal{J}(h)(\vec{x}) := \vec{x}^\frown\mathsf{a}$ has exactly one class of strings. It is checked that $\mathsf{a}^m \sim_H \mathsf{a}^n$ for all $m, n \in \mathbb{N}$. ⚽

It is linguistic practice not to leave the categories implicit (in the form of domain restrictions) but to make them part of the representation. If we so wish this can be implemented as follows. Let C be a set. A **c-string** is a pair $s = \langle \vec{x}, c\rangle$ where $\vec{x} \in A^*$ and $c \in C$. Given s, we put

$$\varepsilon(s) := \vec{x}, \quad \kappa(s) := c. \tag{2.43}$$

For a set S of c-strings write $\varepsilon[S] := \{\varepsilon(s) : s \in S\}$, $\kappa[S] := \{\kappa(s) : s \in S\}$. A **c-string language** is a subset of $A^* \times C$. A **c-string grammar** is a pair $\langle \Omega, \mathcal{C}\rangle$ where Ω is a signature (with domain F) and $\mathcal{C}$ an interpretation function such that for all $f \in F$ $\mathcal{C}(f) : (A^* \times C)^{\Omega(f)} \hookrightarrow (A^* \times C)$. We define $\iota_G(t)$ for an Ω-term t by

$$\iota_G(f s_0 \cdots s_{\Omega(f)-1}) := \mathcal{C}(f)(\iota_G(s_0), \cdots, \iota_G(s_{\Omega(f)-1})). \tag{2.44}$$

We write t^ε in place of $\varepsilon(\iota_G(t))$ and t^κ in place of $\kappa(\iota_G(t))$. Thus we have

$$\iota_G(t) = \langle t^\varepsilon, t^\kappa \rangle. \tag{2.45}$$

We also use the notation f^ε for the function $\varepsilon \circ \mathcal{C}(f)$ and f^κ for $\kappa \circ \mathcal{C}(f)$. A more detailed discussion can be found in Chapter 3. The categories will be most useful when the string operations of the grammar are independent. We shall deal with grammars acting on several components in Chapter 3.

Example 2.14 The shift to categories is not as innocent as it first appears, for we lose certain properties. Here is an example. The relation "is part of" is no longer transitive. Let $F := \{f_0, f_1, g\}$, $\Omega(f_0) := \Omega(f_1) := 0$ and $\Omega(g) := 1$. $C := \{\alpha, \beta\}$ and $A := \{\mathtt{a}\}$.

$$\begin{aligned}
\mathcal{I}(f_0)() &:= \langle \mathtt{a}, \alpha \rangle \\
\mathcal{I}(f_1)() &:= \langle \mathtt{aa}, \alpha \rangle \\
\mathcal{I}(g)(\langle \vec{x}, c \rangle) &:= \begin{cases} \langle \vec{x}^\frown \mathtt{a}, \beta \rangle & \text{if } c = \alpha, \\ \text{undefined} & \text{else.} \end{cases}
\end{aligned} \tag{2.46}$$

This grammar generates the language $\{\langle \mathtt{a}, \alpha \rangle, \langle \mathtt{aa}, \beta \rangle, \langle \mathtt{aa}, \alpha \rangle, \langle \mathtt{aaa}, \beta \rangle\}$. It turns out that /a/ is a part of /aa/, and /aa/ a part of /aaa/, but /a/ is not a part of /aaa/. ⚽

As the example shows we can no longer simply say that a string occurs as a substring; it occurs in a c-string as a c-string and so the category that it has in that occurrence may also be fixed. For example, /`I see John fly.`/ contains /`fly`/ as a verb and not as a noun.

An important class of (c-string) grammars are the *bottom up context free (c-) grammars*. These are not the same as ordinary CFGs. We shall recall the definition of standard CFGs first and then turn to the bottom up version. Recall that a context free grammar is standardly taken to be a quadruple $G = \langle A, N, S, R \rangle$, where A and N are disjoint sets, $S \in N$ and R a set of replacement rules. They have the form $X \to \vec{y}$, where $X \in N$ and $\vec{y} \in (A \cup N)^*$ is a sequence over $A \cup N$. The rules define a replacement relation in the following way.

Definition 2.12 Let $\rho = \vec{x} \to \vec{y}$ be a rule. We say that $\vec{u}\,\vec{y}\,\vec{w}$ is **1-step derivable** via ρ from $\vec{u}\,\vec{x}\,\vec{v}$, in symbols $\vec{u}\,\vec{x}\,\vec{v} \Rightarrow_\rho \vec{u}\,\vec{y}\,\vec{v}$. For a set R of rules we write $\vec{u}\,\vec{x}\,\vec{v} \Rightarrow_\rho \vec{u}\,\vec{y}\,\vec{v}$ and say that $\vec{u}\,\vec{y}\,\vec{v}$ is **1-step derivable** from $\vec{u}\,\vec{x}\,\vec{v}$ if there is a rule $\rho \in R$ such that $\vec{u}\,\vec{x}\,\vec{v} \Rightarrow_\rho \vec{u}\,\vec{y}\,\vec{v}$. Furthermore, we say that $\vec{w}$ is n**-step derivable** in R from $\vec{v}$ and write $\vec{v} \Rightarrow_R^n \vec{w}$ if either $n = 0$ and $\vec{w} = \vec{v}$ or $n > 0$ and there is a $\vec{u}$ such that $\vec{u}$ is $n-1$-step derivable from $\vec{v}$ and $\vec{w}$ is 1-step derivable from $\vec{u}$.

Notice that $\vec{v} \Rightarrow_R^1 \vec{w}$ and $\vec{v} \Rightarrow_R \vec{w}$ are synonymous, and that $\vec{v} \Rightarrow_{\{\rho\}} \vec{w}$ and $\vec{v} \Rightarrow_\rho \vec{w}$ are also synonymous; R or ρ will be dropped when the context makes clear which rules are being used. Notice furthermore that it may happen that a rule can be applied to a given string in several ways. The rule $\mathtt{A} \to \mathtt{aa}$ can be applied to the string /AcAb/ to yield either /aacAb/ or /Acaab/. Therefore, if we want to know what the result will be after applying the rule we need to identify the occurrence of the left-hand side that is being replaced. When can do this by underlining as follows: $\underline{\texttt{A}}\texttt{cAb} \Rightarrow \texttt{aacAb}$ and $\texttt{Ac}\underline{\texttt{A}}\texttt{b} \Rightarrow \texttt{Acaab}$. If the occurrence is underlined then the rule *must* be applied to that occurrence. Hence we do not have $\underline{\texttt{A}}\texttt{cAb} \Rightarrow \texttt{Acaab}$. Now, suppose we have such a marked string; then the result is still not unique unless we know which rule is being applied. This follows from the fact that several rules may replace a given string. For example, if we also have the rule $\texttt{A} \to \texttt{cd}$ then from /$\underline{\texttt{A}}\texttt{cAb}$/ we may proceed to /cdcAb/ in addition to /aacAb/. However, if also the resulting string is given, the rule that has been applied can be inferred. Thus, in order to show that a given string $\vec{w}$ is n-step derivable from a string $\vec{v}$ we need to produce a sequence $\langle \vec{v}_i : i < n\rangle$ of length n of marked strings such that $\vec{v}_i \Rightarrow \vec{v}_{i+1}$ for $i < n-1$ and $\vec{v}_{n-1} \Rightarrow \vec{w}$. Such a sequence is called a **derivation**. Notice that the sequence contains marked strings not just strings, though we shall often not show the marks. The derived string is by definition not marked, though it is often added at the end of the derivation sequence so that one can infer the choice of rules in each step.

Given a nonterminal A and a string $\vec{x}$ we write $A \vdash_G \vec{x}$ and say that G **derives** $\vec{x}$ from A if there is an n such that $A \Rightarrow^n_R \vec{x}$.

Definition 2.13 Let $G = \langle S, N, A, R\rangle$ be a CFG. The language of G in the narrow sense is defined by

$$L(G) := \{\vec{x} \in A^* : S \vdash_G \vec{x}\}. \tag{2.47}$$

The language in the wide sense is defined by

$$L^w(G) := \{\vec{x} \in A^* : \text{for some } X \in N : X \vdash_G \vec{x}\}. \tag{2.48}$$

A language L in the narrow (wide) sense is **context free** if there is a context free grammar G such that $L = L(G)$ ($L = L^w(G)$).

Also, write $[A]_G := \{\vec{x} : A \vdash_G \vec{x}\}$. Then $L(G) = [S]_G$. This notion of grammar is top down and nondeterministic. It generates the strings from a single string (consisting in the single letter S).

Example 2.15 Let G be defined as follows.

$$G := \langle\{\texttt{a}, \cdots, \texttt{z}, \texttt{␣}\}, \{\texttt{<S>}, \texttt{<NP>}, \texttt{<VP>}, \texttt{<N>}, \texttt{<D>}, \texttt{<VI>}, \texttt{<VT>}\}, \texttt{<S>}, R\rangle \tag{2.49}$$

The alphabet consists in all lower case letters plus the space.

$$\begin{aligned} R = \{&\texttt{<S>} \to \texttt{<NP><VP>} \\ &\texttt{<NP>} \to \texttt{<D><N>} \\ &\texttt{<D>} \to \texttt{the␣} \mid \texttt{a␣} \\ &\texttt{<N>} \to \texttt{cat␣} \mid \texttt{dog␣} \mid \texttt{mouse␣} \\ &\texttt{<VP>} \to \texttt{<VI>} \mid \texttt{<VT><NP>} \\ &\texttt{<VI>} \to \texttt{runs␣} \mid \texttt{sleeps␣} \\ &\texttt{<VT>} \to \texttt{sees␣} \mid \texttt{chases␣}\} \end{aligned} \tag{2.50}$$

This grammar generates among others the following strings:

$$\begin{array}{l}\texttt{<S>}\\ \texttt{<NP><VP>}\\ \texttt{<D>dog␣<VI>}\\ \texttt{a␣dog␣chases␣the␣cat␣}\end{array} \tag{2.51}$$

Only the last of the four strings is meaningful. A derivation of the third string is as follows.

$$\langle \underline{\texttt{<S>}}, \underline{\texttt{<NP>}}\texttt{<VP>}, \texttt{<D><N>}\underline{\texttt{<VP>}}, \texttt{<D>}\underline{\texttt{<N>}}\texttt{<VI><D>dog␣<VI>}\rangle \tag{2.52}$$

⚽

Let us now look at a bottom up version of CFGs. Obviously, to get such a grammar we simply turn the rules around. Rather than assuming a rule, say, $\rho = A \to BC$, we define a string function f_ρ of arity 2 such that f_ρ is interpreted as concatenation, which is to say $\mathcal{I}(f_\rho)(\vec{x}, \vec{y}) = \vec{x}{}^\frown\vec{y}$. However, this function is only defined if $\vec{x}$ is a B-string, that is, if we can derive $\vec{x}$ from B in the grammar and if $\vec{y}$ is a C-string. In this way we guarantee that $\vec{x}{}^\frown\vec{y}$ is an A-string. In general, for each rule ρ we assume a function symbol f_ρ and an interpretation $\mathcal{I}(f_\rho)$. A rule is of the form $A \to \vec{x}$ for some $\vec{x} \in (A \cup N)^*$.

This means that there is n and $\vec{x}_i \in A^*$, $i < n+1$, and $B_i \in N$, $i < n$, such that

$$\rho = A \to \vec{x}_0 B_0 \vec{x}_1 B_1 \cdots B_{n-1} \vec{x}_n \tag{2.53}$$

Then $\Omega(f_\rho) := n$ and its interpretation is

$$\mathcal{I}(f_\rho)(\vec{y}_0, \cdots, \vec{y}_{n-1}) := \begin{cases} \vec{x}_0\vec{y}_0\vec{x}_1\vec{y}_1 \cdots \vec{y}_{n-1}\vec{x}_n & \text{if for all } i < n\text{:} \\ & \quad \vec{y}_i \text{ is a } B_i\text{-string,} \\ \text{undefined} & \text{else.} \end{cases} \tag{2.54}$$

We do this for all ρ that do not have the form $A \to B$. It is an easy matter to transform G into a grammar that has no such rules. But this transformation is actually unnecessary. This defines the grammar $G^\blacklozenge$.

Example 2.16 I transform the grammar from Example 2.15. Let us note that the constituents generate only finitely many strings, so we can list them all.

$$\begin{array}{ll} [\texttt{<D>}]_G & = \{/\texttt{a␣}/, /\texttt{the␣}/\} \\ [\texttt{<N>}]_G & = \{/\texttt{cat␣}/, /\texttt{dog␣}/, /\texttt{mouse␣}/\} \\ [\texttt{<VI>}]_G & = \{/\texttt{runs␣}/, /\texttt{sleeps␣}/\} \\ [\texttt{<VT>}]_G & = \{/\texttt{sees␣}/, /\texttt{chases␣}/\} \\ [\texttt{<VP>}]_G & = (\texttt{runs␣} \mid \texttt{sleeps␣}) \mid (\texttt{sees␣} \mid \texttt{chases␣})(\texttt{the␣} \mid \texttt{a␣}) \\ & \quad (\texttt{cat␣} \mid \texttt{dog␣} \mid \texttt{mouse␣}) \end{array} \tag{2.55}$$

Before the transformation we need to consider the rule `<VP>` $\to$ `<VI>`. This is a unary rule. We eliminate it and add instead the rule

$$\texttt{<S>} \to \texttt{<NP><VI>} \tag{2.56}$$

Now we begin the transformation. The grammar $G^\blacklozenge$ is based on the set $\{f_1, f_2, \cdots, f_{11}\}$ with $\Omega(f_i) = 0$ for $i < 9$ and $\Omega(f_i) = 2$ otherwise. We have

$$\begin{array}{ll}
\mathcal{I}(f_0)() & := \texttt{a␣} \\
\mathcal{I}(f_1)() & := \texttt{the␣} \\
\mathcal{I}(f_2)() & := \texttt{cat␣} \\
\mathcal{I}(f_3)() & := \texttt{dog␣} \\
\mathcal{I}(f_4)() & := \texttt{mouse␣} \\
\mathcal{I}(f_5)() & := \texttt{runs␣} \\
\mathcal{I}(f_6)() & := \texttt{sleeps␣} \\
\mathcal{I}(f_7)() & := \texttt{sees␣} \\
\mathcal{I}(f_8)() & := \texttt{chases␣} \\
\mathcal{I}(f_9)(\vec{x}, \vec{y}) & := \begin{cases} \vec{x}^\frown\vec{y} & \text{if } \vec{x} \in [\texttt{<D>}]_G \text{ and } \vec{y} \in [\texttt{<N>}]_G, \\ \text{undefined} & \text{otherwise.} \end{cases} \\
\mathcal{I}(f_{10})(\vec{x}, \vec{y}) & := \begin{cases} \vec{x}^\frown\vec{y} & \text{if } \vec{x} \in [\texttt{<VT>}]_G \text{ and } \vec{y} \in [\texttt{<NP>}]_G, \\ \text{undefined} & \text{otherwise.} \end{cases} \\
\mathcal{I}(f_{11})(\vec{x}, \vec{y}) & := \begin{cases} \vec{x}^\frown\vec{y} & \text{if } \vec{x} \in [\texttt{<NP>}]_G \text{ and } \vec{y} \in [\texttt{<VI>}]_G, \\ \text{undefined} & \text{otherwise.} \end{cases}
\end{array} \tag{2.57}$$

The reader is asked to check that these modes correspond exactly to the rules of the grammar (in its slight modification). The string /`a␣cat␣sees␣the␣dog␣`/ is derived by the term $f_{11} f_9 f_0 f_2 f_{10} f_7 f_9 f_1 f_3$, as can be checked. ⚽

$G^\blacklozenge$ is a grammar in the sense of Section 2.1. The grammar $G^\blacklozenge$ generates the language of G in the wide sense, as the following theorem documents.

Proposition 2.3 *Let G be a context free grammar. Then $\vec{x} \in L(G^\blacklozenge)$ if and only if there is a nonterminal X such that $X \vdash_G \vec{x}$. In other words, $L(G^\blacklozenge) = L^w(G)$.*

The proof is a relatively easy induction on the length of derivations. I shall relegate this to the exercises.

Example 2.17 The elimination of unary rules is not as innocent as it first appears. In natural languages there are plenty of examples of zero-derivation. One example is the conversion of adjectives to nouns in Hungarian. Typically, adjectives do not inflect. However, in the absence of a noun they can inflect just as nouns and hence should be regarded as such. Thus, the form /`fehéret`/ (accusative of /`fehér`/) must be translated as "a white one". Critically, also the nominative form /`fehér`/ can be so regarded and hence can be translated as either "white" or "a white one". Given a bottom up grammar these two are now confused. However, as long as we do not treat meaning in addition there is no harm in this. This theme will be picked up in Section 3.4. ⚽

Notice that there is no way to generate *only* the language $L(G)$, which is, all and only the S-strings for the start symbol S. When we do a top down generation we can simply choose to start with the start symbol and all the strings we generate are sentences. However, in the bottom up process we cannot restrict ourselves to generating just the sentences. We must generate all intermediate strings. On the other hand there is no need to generate strings with extraneous symbols. In the c-string grammar we can make up for this defect as follows. For a CFG in the standard sense let

$$L^c(G) := \{\langle \vec{x}, X\rangle : X \in N, X \vdash_G \vec{x}\}. \tag{2.58}$$

So, $L^c(G)$ contains strings together with their categorial information; it does not however single out a particular category. We can derive $L(G)$ from $L^c(G)$ by picking all $\vec{x}$ for which $\langle \vec{x}, S\rangle \in L^c(G)$. This is a different notion of language than the generated language in the wide sense. In the latter we do not know what the categories of the strings are; we just know that they have *some* category. On the other hand, for a language in the wide sense there is no need to construct the categories from the input data (as languages mostly do not always mark their expressions for category). The arity of f_ρ equals the number of nonterminals on the right-hand side of the rule.

The string based version presented above is not an exact equivalent of the grammar G. In the exercises we shall show that these grammars may have quite different derivations. To get a more exact correspondence we turn to c-strings. In the case at hand we choose $C := N$. Thus c-strings are pairs $\langle \vec{x}, X\rangle$ where $\vec{x} \in A^*$ and $X \in N$. The interpretation of the function symbol f_ρ is now the partial function

$$\begin{aligned}\mathcal{C}(f_\rho)(\langle \vec{y}_0, c_0\rangle, \langle \vec{y}_1, c_1\rangle, \cdots, \langle \vec{y}_{n-1}, c_{n-1}\rangle) \\ := \langle \vec{x}_0\vec{y}_0\vec{x}_1\vec{y}_1 \cdots \vec{y}_{n-1}\vec{x}_n, f^\kappa_\rho(c_0, c_1, \cdots, c_{n-1})\rangle\end{aligned} \tag{2.59}$$

where

$$f^\kappa_\rho(c_0, \cdots, c_{n-1}) := \begin{cases} A & \text{if for all } i < n\text{: } c_i = B_i, \\ \text{undefined} & \text{else.}\end{cases} \tag{2.60}$$

Then $L(G)$ is a set of pairs $\langle \vec{x}, c\rangle$. We say that $\vec{x}$ **has category** c **in** G if some G-term unfolds to $\langle \vec{x}, c\rangle$. A given string can have several categories.

Example 2.18 We continue the language of equations (Example 2.6 on page 16). The grammar G_Q consists in the alphabet of terminals

$$\text{:bt:} := \{\mathtt{0}, \mathtt{1}, \mathtt{+}, \mathtt{-}, \mathtt{(}, \mathtt{)}, \mathtt{=}\}. \tag{2.61}$$

The alphabet of nonterminals is $N := \{\mathtt{E}, \mathtt{B}, \mathtt{T}\}$, the start symbol /E/ and the set of rules is as follows.

$$\begin{aligned} &\texttt{E} \rightarrow \texttt{T=T} \\ &\texttt{T} \rightarrow \texttt{(T+T)} \mid \texttt{(T-T)} \mid \texttt{B} \\ &\texttt{B} \rightarrow \texttt{B0} \mid \texttt{B1} \mid \texttt{0} \mid \texttt{1} \end{aligned} \tag{2.62}$$

By default, a derivation starts with the letter /E/. Thus

$$C = \langle :\text{bt}:, N, \texttt{E}, R \rangle. \tag{2.63}$$

Recall that "|" is an abbreviation. It allows to group together rules with the same left-hand side. Figure 2.1 shows an example of a derivation in G_Q. In each step we replace a single occurrence of a nonterminal by a corresponding right-hand side of (2.62). ⚽

An X-**derivation** is a sequence of strings starting with the nonterminal X, where each nonfirst member is obtained from the previous by replacing a nonterminal symbol in the appropriate way. A **derivation** is an X-derivation with X the top symbol. For our purposes, however, the best objects to deal with are not the derivations but the analysis terms. The analysis term of a derivation is obtained as follows. Assign to each rule ρ with $n(\rho)$ nonterminals on the right a function symbol f_ρ of arity $n(\rho)$. This defines the signature. Start with the variable x_0. A step in the derivation consists in the replacement of an occurrence of a variable x_i by a term of the form $f_\rho(x_{i_0}, x_{i_1}, \cdots, x_{i_{n(\rho)-1}})$ where the x_{i_j} so that in the end no variable occurs twice. This procedure is best explained with the derivation above.

Example 2.19 Continuing Example 2.18. We give the following names to the rules.

$$\begin{array}{ll} a & \texttt{E} \rightarrow \texttt{T=T} \\ b & \texttt{T} \rightarrow \texttt{(T+T)} \\ c & \texttt{T} \rightarrow \texttt{(T-T)} \\ d & \texttt{T} \rightarrow \texttt{B} \\ e & \texttt{B} \rightarrow \texttt{B0} \\ f & \texttt{B} \rightarrow \texttt{B1} \\ g & \texttt{B} \rightarrow \texttt{0} \\ h & \texttt{B} \rightarrow \texttt{1} \end{array} \tag{2.64}$$

$$\begin{array}{l} \underline{\texttt{E}} \\ \underline{\texttt{T}}\texttt{=T} \\ \underline{\texttt{B}}\texttt{=T} \\ \texttt{0=}\underline{\texttt{T}} \\ \texttt{0=(T-}\underline{\texttt{T}}\texttt{)} \\ \texttt{0=(T-}\underline{\texttt{B}}\texttt{)} \\ \texttt{0=(}\underline{\texttt{T}}\texttt{-0)} \\ \texttt{0=(}\underline{\texttt{B}}\texttt{-0)} \\ \texttt{0=(}\underline{\texttt{B}}\texttt{0-0)} \\ \texttt{0=(10-0)} \end{array}$$

Fig. 2.1 A derivation in G_Q

$\underline{\mathtt{E}}$	$\boxed{x_0}$
$\underline{\mathtt{T}}\mathtt{=T}$	$f_a \boxed{x_0} x_1$
$\underline{\mathtt{B}}\mathtt{=T}$	$f_a f_d \boxed{x_0} x_1$
$\mathtt{0=}\underline{\mathtt{T}}$	$f_a f_d f_g \boxed{x_1}$
$\mathtt{0=(T-}\underline{\mathtt{T}}\mathtt{)}$	$f_a f_d f_g f_c x_0 \boxed{x_1}$
$\mathtt{0=(T-}\underline{\mathtt{B}}\mathtt{)}$	$f_a f_d f_g f_c x_0 f_d \boxed{x_1}$
$\mathtt{0=(}\underline{\mathtt{T}}\mathtt{-0)}$	$f_a f_d f_g f_c \boxed{x_0} f_d f_g$
$\mathtt{0=(}\underline{\mathtt{B}}\mathtt{-0)}$	$f_a f_d f_g f_c f_d \boxed{x_0} f_d f_g$
$\mathtt{0=(}\underline{\mathtt{B}}\mathtt{1-0)}$	$f_a f_d f_g f_c f_d f_e \boxed{x_0} f_d f_g$
$\mathtt{0=(10-0)}$	$f_a f_d f_g f_c f_d f_e f_h f_d f_g$

Fig. 2.2 Deriving the term

Thus the symbols are called f_a, f_b, f_c (binary), f_d, f_e, f_f (unary), f_g and f_h (zeroary). The derivation is translated to a term as shown in Fig. 2.2. The variable that is being replaced is surrounded by a box. The exact recipe is this: if the derivation replaces the nth nonterminal counting from the left, then it is the nth variable from the left that is being replaced irrespective of its index. ⚽

Now we shall supply the term symbols with interpretations that match the effect of the rules. Call $\vec{x}$ an X-**string** if $X \vdash_G \vec{x}$. Write $L_X(G)$ for the set of X-strings of G. In our example $L_{\mathtt{E}}(G_Q)$ is the set of equations; these are strings of the form /$\vec{x}$=$\vec{y}$/, where both $\vec{x}$ and $\vec{y}$ are T-strings. T-strings are terms; these are strings of the form (a) $\vec{x}$, where $\vec{x}$ consists in 0 and 1 only (a number expression, or a B-string), (b) /($\vec{x}$+$\vec{y}$)/ where $\vec{x}$ and $\vec{y}$ are T-strings, or (c) /($\vec{x}$-$\vec{y}$)/ where $\vec{x}$ and $\vec{y}$ are T-strings. Finally, the B-strings are exactly the strings from $\{\mathtt{0}, \mathtt{1}\}^+$.

For example, $\rho = \mathtt{B} \to \mathtt{B1}$ is a rule of G_Q, and so we have a symbol f_ρ with $\Omega(f_\rho) = 1$. The function takes a B-string $\vec{x}$ and appends /1/. Hence:

$$\iota(f_\rho)(\vec{x}) := \begin{cases} \vec{x}^\frown\mathtt{1} & \text{if } \vec{x} \text{ is a B-string,} \\ \text{undefined} & \text{else.} \end{cases} \tag{2.65}$$

Similarly, if $\rho' = \mathtt{T} \to \mathtt{(T+T)}$ we postulate a symbol $f_{\rho'}$ with $\Omega(f_{\rho'}) = 2$ and which acts as follows:

$$\iota(f_{\rho'})(\vec{x}, \vec{y}) := \begin{cases} \mathtt{(}^\frown\vec{x}^\frown\mathtt{+}^\frown\vec{y}^\frown\mathtt{)} & \text{if } \vec{x} \text{ and } \vec{y} \text{ are T-strings,} \\ \text{undefined} & \text{else.} \end{cases} \tag{2.66}$$

As we have briefly noted above, the properties "B-string", "T-string" and so on can actually be defined without making reference to the grammar.

We can use Example 2.19 to show that the transformation $(-)^\blacklozenge$ of CFGs preserves the strings but not the set of terms if applied also to unary rules. The rule d has the form $\mathtt{T} \to \mathtt{B}$. It is converted into the string function $\mathcal{I}(f_d)(\vec{x}) = \vec{x}$, in other words the identity function. This function is iterable, while the rule is not. Thus the term $f_d f_d f_g$ would evaluate in $G_Q^\blacklozenge$ to /0/.

$$\iota(f_d f_d f_g) = \mathcal{I}(f_d)(\mathcal{I}(f_d)(\mathcal{I}(f_g)())) = \mathcal{I}(f_d)(\mathcal{I}(f_d)(\mathtt{0})) = \mathcal{I}(f_d)(\mathtt{0}) = \mathtt{0} \quad (2.67)$$

However, there is no derivation with term $f_d f_d f_g$. Try to start with the symbol T, for example:

$$\begin{array}{ll} \mathtt{T} & x_0 \\ \mathtt{B} & f_d x_0 \\ ? & f_d f_d x_0 \end{array} \quad (2.68)$$

Similarly if we start with /B/. (If you want a derivation beginning with the start symbol, take the term $f_a f_d f_d f_g f_d f_h$.) It might be deemed that all we have to do is to exclude unary rules. That this is not so is shown in Exercise 2.18.

We can characterize in more exact terms the connection between the two kinds of grammars. Here is a characterization of context free languages in terms of the generating functions. It shows that if the functions are partial functions of a certain kind and such that ranges of functions are subsets of domains (or disjoint) then the generated language is context free (and conversely).

Definition 2.14 Let $G = \langle \Omega, \mathcal{I} \rangle$ be a grammar. G is called a **concatenation grammar** if for all modes f, $\mathcal{I}(f)$ is the restriction of a polynomial function of the string algebra to some arbitrary set of sequences of strings.

This definition says the following. In a concatenation grammar a mode f interpreted as a partial function $\mathcal{I}(f) : (A^*)^{\Omega(f)} \hookrightarrow A^*$. While the domain is some arbitrary set $D \subseteq (A^*)^{\Omega(f)}$, there must exist a polynomial function p such that $\mathcal{I}(f) = p \restriction D$. Notice namely that the string polynomials are total. These polynomials may be arbitrarily restricted. However, as we shall see, in context free grammars there are tight restrictions on these domains. Say a polynomial $p(\vec{x})$ is a **linear string polynomial** if it is composed from the variables x_i and constants such that each x_i occurs exactly once. If p is a polynomial, we denote the induced function by p^{A^*}. $f : (A^*)^n \to A^*$ is a **rectangularly restricted linear string polynomial** if there is a linear string polynomial $p(x_0, \cdots, x_{n-1})$ such that $f \subseteq p^{A^*}(\vec{x})$ and there are subsets $P_i \subseteq A^*$, $i < n$, such that $\mathrm{dom}(f) = \mathsf{X}_{i<n} P_i$. Now recall that the grammar $G^{\blacklozenge}$ uses precisely such functions. Thus we have

Proposition 2.4 *If a language $L \subseteq A^*$ is context free then it has a grammar G in which all function symbols are interpreted by rectangularly restricted linear string polynomials.*

For the converse, a little more is needed. Namely, let H be a grammar such that all $\mathcal{I}(f)$ are rectangularly restricted linear polynomials. So for each f there are sets Q_i^f, $i < \Omega(f)$, such that the domain of $\mathcal{I}(f)$ is $\mathsf{X}_{i<\Omega(f)} Q_i^f$. Assume moreover that for every g and $i < \Omega(g)$: either $\mathrm{rng}(\mathcal{I}(f)) \subseteq Q_i^g$ or $\mathrm{rng}(\mathcal{I}(f)) \cap Q_i^g = \varnothing$. We call this the **connectivity property** for H. For each domain Q we choose a nonterminal N_Q (notice that $N_Q = N_P$ if $P = Q$ as sets). Further, for a function symbol f such that $\mathrm{dom}(\mathcal{I}(f)) = \mathsf{X}_{i<\Omega(f)} Q_i^f$ and $\mathrm{rng}(\mathcal{I}(f)) \subseteq Q_i^g$ we create a rule

$$\rho_f : N_{Q_i^g} \to \vec{x}_0 N_{Q_0^f} \vec{x}_1 N_{Q_1^f} \vec{x}_2 \cdots \vec{x}_{\Omega(f)-1} N_{Q^f_{\Omega(f)-1}} \vec{x}_{\Omega(f)} \tag{2.69}$$

where the $\vec{x}_i$ are chosen such that $\mathcal{I}(f)$ is the restriction of the polynomial

$$p^{A^*}(y_0, \cdots, y_{\Omega(f)-1}) := \vec{x}_0 y_0 \vec{x}_1 y_1 \vec{x}_2 \cdots \vec{x}_{\Omega(f)-1} y_{\Omega(f)-1} \vec{x}_{\Omega(f)}. \tag{2.70}$$

This grammar is such that $\vec{z}$ is an N_Q-string for some Q if and only if $\vec{z} \in L(H)$.

Proposition 2.5 *If H is a grammar such that all $\mathcal{I}(f)$ are rectangularly restricted linear string polynomials and $\mathcal{I}$ has the connectivity property then $L(H)$ is context free.*

Example 2.20 I give some examples to show that none of the conditions can be dropped. First, the functions must be linear string polynomials. Take $f(\vec{x}) := \vec{x}\,\vec{x}$ on the alphabet $\{\mathsf{a}\}$. This function is induced by the polynomial $p(x_0) := x_0 x_0$. It is not linear as the variable x_0 occurs twice on the right. As it happens the function generates the language $\{\mathsf{a}^{2^n} : n \in \mathbb{N}, n > 0\}$ from a. (Assuming we have a single constant c in the signature with interpretation a.) One may be tempted to eliminate the nonlinearity by using the following function instead.

$$f(\vec{x}, \vec{y}) := \begin{cases} \vec{x}\,\vec{y} & \text{if } |\vec{x}| = |\vec{y}|, \\ \text{undefined} & \text{otherwise.} \end{cases} \tag{2.71}$$

This (binary) function is the restriction of the polynomial $p(x_0, x_1) := x_0 x_1$ to the set of all pairs of strings of equal length. Unfortunately, this function is not rectangularly restricted. There are no sets H, K such that the domain of f is $H \times K$ and the set of strings generable from a with this function is again the set $\{\mathsf{a}^{2^n} : n \in \mathbb{N}, n > 0\}$. Finally, consider the following two functions. The first is a modification of f:

$$f(\vec{x}, \vec{y}) := \begin{cases} \vec{x}\,\vec{y} & \text{if } \vec{x}, \vec{y} \in \mathsf{a}^*, \\ \text{undefined} & \text{otherwise.} \end{cases} \tag{2.72}$$

The second is a unary function g defined by

$$g(\vec{x}) := \begin{cases} \vec{x}\,\mathsf{b} & \text{if } |\vec{x}| = 2^n \text{ for some } n, \\ \text{undefined} & \text{otherwise.} \end{cases} \tag{2.73}$$

Both functions are restrictions of linear polynomial functions to some rectangles. Only the connectivity property is lacking. For $Q^g = \{\vec{x} : |\vec{x}| = 2^n \text{ for some } n \in \mathbb{N}\}$, and we have both $\text{rng}(\mathcal{I}(g)) \nsubseteq Q^g$ and $\text{rng}(\mathcal{I}(g)) \cap Q^g \neq \varnothing$. The generated language is

$$\mathsf{a}^+ \cup \{\mathsf{a}^{2^n}\mathsf{b} : n \in \mathbb{N}, n > 0\}. \tag{2.74}$$

This is not context free. Hence all the conditions are really necessary and independent of each other. ⚽

This gives rise to the following definition.

Definition 2.15 A string grammar is called (**bottom up**) **context free** if it is a concatenation grammar with rectangularly restricted linear string polynomials with the connectivity property.

Notice that "context free" is applied not only to rule based grammars but also to c-string grammars and string grammars alike. Whenever there is risk of confusion, the context free grammars in the sense of this book are called "bottom up context free".

I close this section with some remarks concerning the use of categories as discriminatory devices. Suppose two strings are such that in a language they have the same category. Then we will want to say that they should also be of the same category in the analysing grammar. Recall that in a context free language, the formal concept of identity of category was substitutability in all 1-contexts, written $\vec{x} \sim_L \vec{y}$.

Principle 1 (Identity of Indiscernibles) *Let G be a context free c-grammar. If $\vec{x} \sim_L \vec{y}$ and $\langle \vec{x}, c\rangle \in L(G)$ then also $\langle \vec{y}, c\rangle \in L(G)$.*

We shall not spell out the generalisation to other kinds of grammars, though it is straightforward to do.

Exercise 2.13 In Example 2.14 is was shown that the relation *is a part of* is not transitive. Find an example to show that it is also not antisymmetric. (A relation R is **antisymmetric** if from $x \ R \ y$ and $y \ R \ x$ follows $x = y$.)

Exercise 2.14 A grammar is left regular if the functions are zeroary or unary and the unary functions all have the form $f(\vec{x}) := \vec{x}^\frown a$ for some a. Let L be a language. Define $\vec{x}/L := \{\vec{y} : \vec{x}^\frown\vec{y} \in L\}$. Show that for a regular grammar G generating L: $\vec{x} \sim_G \vec{y}$ if and only if $\vec{x}/L = \vec{y}/L$.

Exercise 2.15 Why does the bottom up grammar $G^\blacklozenge$ not contain any f_ρ for rules of the form $\rho = A \to B$?

Exercise 2.16 Let G be a context free grammar and A a nonterminal. Let $H_A := \{\vec{x} : A \vdash_G \vec{x}\}$. Show that for every $\vec{x} \in H_A$ $H_A \subseteq [\vec{x}]_G$. Give an example to show that equality need not hold!

Exercise 2.17 Prove Proposition 2.3.

Exercise 2.18 Context free grammars allow to tune derivations more finely than grammars in the sense of Definition 2.4. Here is an example, due to Ben George. Let G consist in the rules

$$\begin{aligned} &\mathtt{S} \to \mathtt{L} \mid \mathtt{R} \\ &\mathtt{L} \to \mathtt{La} \mid \mathtt{a} \\ &\mathtt{R} \to \mathtt{aR} \mid \mathtt{a} \end{aligned} \tag{2.75}$$

Construct the corresponding grammar and show that it allows for more analysis terms for the string /aaaa/ than does G.

2.4 Indeterminacy and Adjunction

In the previous section we have constructed a "bottom up" version $G^\blacklozenge$ of a context free grammar G. (I should stress here, though, that only $G^\blacklozenge$, not G, is a grammar in the sense of this book.) In addition to the differences between the types of grammars that I mentioned earlier there is a major difference between G and $G^\blacklozenge$. It is that by definition $L(G^\blacklozenge)$ is the set of all strings that are constituents for *some* nonterminals as opposed to just the strings corresponding to the start symbol. Thus the standard definition of $L(G)$ for a CFG is contained in $L(G^\blacklozenge)$ but the two need not be identical (cf. Proposition 2.3). The difference is exactly between language in the wide sense and language in the narrow sense. Since I insist that the language of a grammar must be taken in the wide sense we must ask if there is a kind of grammar that generates the sentences all by themselves so that the two notions actually coincide for this type of grammar. Such grammars do exist. The adjunction grammars are of this kind. Unfortunately, these grammars turn out to be somewhat different from the grammars previously defined in that the defining operations generally are *relations*. Grammars of this form shall be called *indeterminate grammars* (the label *relational grammar* has already been taken). I shall return to indeterminate grammars again in Section 3.7 in connection with interpreted languages.

Definition 2.16 An **indeterminate grammar over** A is a pair $\langle \Omega, \mathcal{I} \rangle$, where Ω is a signature and for every $f \in F$, $\mathcal{I}(f) \subseteq (A^*)^{\Omega(f)+1}$. F is the set of **modes** of the grammar. The set $\{f : \Omega(f) = 0\}$ is called the **lexicon** of G and the set $\{f : \Omega(f) > 0\}$ the set of **rules**. The **language** generated by G, in symbols $L(G)$, is defined to be the least set S satisfying for every $f \in F$ and all $\vec{x}_i \in A^*, i < \Omega(f)$:

$$\text{If for all } i < \Omega(f) : \vec{x}_i \in S \text{ and if } \langle \vec{x}_0, \cdots, \vec{x}_{\Omega(f)-1}, \vec{y} \rangle \in \mathcal{I}(f) \text{ then } \vec{y} \in S. \tag{2.76}$$

Thus, the output of a rule is not assumed to be unique. In a grammar of the usual sort the output need not exist but if it exists, it is unique. In an indeterminate grammar it need not even be unique. Adjunction grammars are such grammars. They are popular since they generate more than context free languages and enjoy nevertheless quite a simple description. I point out that as soon as we move to interpreted languages it will turn out that the indeterminacy will have to be eliminated; see also the discussion in Section 3.7.

Definition 2.17 A **2-context** is a triple $\gamma = \langle \vec{u}, \vec{v}, \vec{w} \rangle$. The result of inserting a pair $\langle \vec{x}, \vec{y} \rangle$ into γ is defined as follows.

$$\gamma(\langle \vec{x}, \vec{y} \rangle) := \vec{u}\, \vec{x}\, \vec{v}\, \vec{y}\, \vec{w} \tag{2.77}$$

A **2-locale** is a set of 2-contexts. A **string adjunction rule** is a pair $\rho = \langle\langle\vec{x}, \vec{y}\rangle, \Lambda\rangle$, where Λ is a 2-locale.

According to the previous definition, the string relation associated with ρ is

$$\mathrm{Adj}(\rho) := \{\langle\vec{u}\,\vec{v}\,\vec{w}, \vec{u}\,\vec{x}\,\vec{v}\,\vec{y}\,\vec{w}\rangle : \langle\vec{u}, \vec{v}, \vec{w}\rangle \in \Lambda\}. \tag{2.78}$$

Definition 2.18 A **string adjunction grammar** is a pair $A = \langle S, R\rangle$, where S is a finite set of strings and R a finite set of string adjunction rules.

For a string adjunction grammar we define the following signature: let $f_{\vec{x}}$ be a symbol of arity 0 for every $\vec{x} \in S$; and let g_ρ be a symbol of arity 1 for every $\rho \in R$. This defines the signature. The interpretation is given by

$$\mathcal{I}(f_{\vec{x}}) := \vec{x}, \qquad \mathcal{I}(g_\rho) := \mathrm{Adj}(\rho). \tag{2.79}$$

With this definition, the formal apparatus of the previous sections can be used with minor adaptations.

We say that G **generates** $\vec{y}$ in n-steps if the following holds: $n = 0$ and $\vec{y} \in S$ or $n > 0$ and there is a $\vec{z}$ such that A generates $\vec{z}$ in $n - 1$ steps and there is a rule $\langle\langle\vec{x}_0, \vec{x}_1\rangle, \Lambda\rangle$ and $\gamma = \langle\vec{u}, \vec{v}, \vec{w}\rangle \in A^*$ such that $\vec{z}$ possesses the decomposition $\vec{z} = \gamma(\langle\varepsilon, \varepsilon\rangle) = \vec{u}\,\vec{v}\,\vec{w}$ and

$$\vec{y} = \gamma(\langle\vec{x}_0, \vec{x}_1\rangle) = \vec{u}\vec{x}_0\vec{v}\vec{x}_1\vec{w} \tag{2.80}$$

$L(G)$ denotes the set of strings that can be generated in a finite number of steps. An alternative way to define this notion is to define the value of terms to be sets. Namely, $\iota_G(f s_0 \cdots s_{\Omega(f)-1})$ would be the projection to the last component of the following set:

$$\left(\prod_{i<\Omega(f)} \iota_G(s_i)\right) \times A^* \cap \mathcal{I}(f) \tag{2.81}$$

For a zeroary mode $f_{\vec{x}}$ we have

$$\iota_G(f_{\vec{x}}) = (1 \times A^*) \cap \{\vec{x}\} = \{\vec{x}\}. \tag{2.82}$$

The other cases are similar.

Example 2.21 We shall now give a presentation of the E-strings of the grammar from Example 2.18 using a string adjunction grammar. We put

$$S := \{\texttt{0=0}, \texttt{0=1}, \texttt{1=0}, \texttt{1=1}\} \tag{2.83}$$

The rules are as follows. Let Λ_1 be the set of triples $\langle \vec{u}x, \vec{v}, \vec{w}\rangle$ such that x is either /𝟘/ or /1/ and $\vec{v}\,\vec{w}$ does not begin with /𝟘/ or /1/. (This is equivalent with the following: (1) $\vec{v} \neq \varepsilon$ and $\vec{v}$ does not begin with /𝟘/ or /1/, or (2) $\vec{v} = \varepsilon$ and $\vec{w}$ (!) does not begin with /𝟘/ or /1/.) Let Λ_2 be the set of triples of the form $\langle \vec{u}, \vec{v}, \vec{w}\rangle$, where both $\vec{u}$ does not end with /𝟘/ or /1/, $\vec{w}$ does not begin with /𝟘/ or /1/, while $\vec{v} \in \{\mathtt{0}, 1\}^*$.

$$\begin{aligned}
\rho_0 &:= \langle\langle \mathtt{0}, \varepsilon\rangle, \Lambda_1\rangle\\
\rho_1 &:= \langle\langle \mathtt{1}, \varepsilon\rangle, \Lambda_1\rangle\\
\rho_2 &:= \langle\langle \varepsilon, \mathtt{0}\rangle, \Lambda_1\rangle\\
\rho_3 &:= \langle\langle \varepsilon, \mathtt{1}\rangle, \Lambda_1\rangle\\
\rho_4 &:= \langle\langle \mathtt{(}, \mathtt{+0)}\rangle, \Lambda_2\rangle\\
\rho_5 &:= \langle\langle \mathtt{(}, \mathtt{+1)}\rangle, \Lambda_2\rangle\\
R &:= \{\rho_0, \cdots, \rho_5\}
\end{aligned} \tag{2.84}$$

The signature is $F := \{f_0, \cdots, f_3, g_0, \cdots, g_5\}$, where the f_i are zeroary and the g_i are unary. Further,

$$\begin{aligned}
\mathcal{I}(f_0) &:= \{\mathtt{0=0}\}\\
\mathcal{I}(f_1) &:= \{\mathtt{0=1}\}\\
\mathcal{I}(f_2) &:= \{\mathtt{1=0}\}\\
\mathcal{I}(f_3) &:= \{\mathtt{1=1}\}\\
\mathcal{I}(g_i) &:= \mathrm{Adj}(\rho_i)
\end{aligned} \tag{2.85}$$

Here is an example of a derivation:

$$\begin{array}{ll}
f_1 & \mathtt{0=1}\\
g_5 f_1 & \mathtt{(0+1)=1}\\
g_2 g_5 f_1 & \mathtt{(0+10)=1}\\
g_5 g_2 g_5 f_1 & \mathtt{(0+(10+1))=1}
\end{array} \tag{2.86}$$

The first line is in S. To get from the first line to the second we choose a decomposition of /𝟘=1/ as /$\varepsilon^\frown\mathtt{0}^\frown$=1/. Thus, choose $\gamma = \langle \varepsilon, \mathtt{0}, \mathtt{=1}\rangle$. This is in Λ_2 since ε does not end in /𝟘/ or /1/, the middle string is a binary string and /=1/ does not begin with /𝟘/ or /1/. Thus we can apply the rule $\langle\langle \mathtt{(}, \mathtt{+1)}\rangle, \Lambda_2\rangle$.

$$\gamma(\langle \mathtt{(}, \mathtt{1)}\rangle) = \varepsilon^\frown/\mathtt{(}/^\frown/\mathtt{0}/^\frown/\mathtt{+1)}/^\frown/\mathtt{=1}/ = /\mathtt{(0+1)=1}/ \tag{2.87}$$

It may be checked that

$$\begin{aligned}
&\iota_G(g_5 g_2 g_5 f_1)\\
&= \{\mathtt{((00+1)+1)=1},\ \mathtt{(00+1)=(1+1)},\ \mathtt{((0+1)+10)=1},\ \mathtt{(0+1)=(10+1)},\\
&\quad \mathtt{((0+1)+1)=10},\ \mathtt{(0+1)=(1+10)},\ \mathtt{(00+(1+1))=1},\ \mathtt{00=((1+1)+1)},\\
&\quad \mathtt{(0+(10+1))=1},\ \mathtt{0=((10+1)+1)},\ \mathtt{(0+(1+1))=10},\ \mathtt{0=((0+1)+10)},\\
&\quad \mathtt{(00+1)=(1+1)},\ \mathtt{00=(1+(1+1))},\ \mathtt{(0+10)=(1+1)},\ \mathtt{0=(10+(1+1))},\\
&\quad \mathtt{(0+1)=(10+1)},\ \mathtt{0=(1+(10+1))}\}.
\end{aligned} \tag{2.88}$$

⚽

Example 2.22 (Cf. Example 2.7.) We give another example: Boolean logic in Polish Notation. The alphabet is :bool: $= \{\wedge, \vee, \neg, \mathtt{p}, \mathtt{0}, \mathtt{1}\}$. A term in Polish Notation is either /p/ followed by an index (a sequence of /0/ and /1/) or it is a function symbol f ($\neg$, $\wedge$ or $\vee$) followed by $\Omega(f)$ many terms. The formation rules using adjunction grammars are as follows. The set of start strings is $S := \{\mathtt{p}\}$. The rules are

$$\begin{aligned}
R := \{&\langle\langle \mathtt{0}, \varepsilon\rangle, \langle A^* \cdot \mathtt{p}, \varepsilon, A^*\rangle\rangle,\\
&\langle\langle \mathtt{1}, \varepsilon\rangle, \langle A^* \cdot \mathtt{p}, \varepsilon, A^*\rangle\rangle,\\
&\langle\langle \neg, \varepsilon\rangle, \langle A^*, (\mathtt{p}|\wedge|\vee|\neg) \cdot A^*, \varepsilon\rangle\rangle,\\
&\langle\langle \wedge\mathtt{p}, \varepsilon\rangle, \langle A^*, (\mathtt{p}|\wedge|\vee|\neg) \cdot A^*, \varepsilon\rangle\rangle,\\
&\langle\langle \vee\mathtt{p}, \varepsilon\rangle, \langle A^*, (\mathtt{p}|\wedge|\vee|\neg) \cdot A^*, \varepsilon\rangle\rangle\}.
\end{aligned} \tag{2.89}$$

Using Exercises 2.6 and 2.7 we can see that this preserves termhood: the sum of the elements added in the string is 0, and the sum of the prefixes is positive. The original Polish Notation had no room for indices but they pose no problem here. It is easy to verify that any string in Polish Notation is derivable in this grammar. ⚽

It is easy to generalize the previous example to Polish Notation in general (see the Exercises). Furthermore, I describe in the exercises how one can derive an adjunction grammar for bracketed notation as well.

In the remainder of this section I shall describe two variants of adjunction grammars that have been discussed in the literature.

Definition 2.19 A 2-locale Λ is **factored** if there are sets $S \subseteq A^* \times A^*$ and $C \subseteq A^*$ such that $\Lambda = \{\langle \vec{u}, \vec{v}, \vec{w}\rangle : \langle \vec{u}, \vec{w}\rangle \in S, \vec{v} \in C\}$. A rule is factored if its 2-locale is. A **contextual grammar** is a string adjunction grammar in which every rule is factored.

See Martín-Vide and Păun (1998) for an overview.

The most popular variant of adjunction grammars are however the tree adjunction grammars (TAGs). These grammars can be explained by a method of coding trees into strings. We shall define certain strings that we call trees. Let N be a set, the set of **nonterminal labels**. Then N-**trees** over the alphabet A are strings from $A \cup N \cup \{\mathtt{(}, \mathtt{)}, \lightning\}$. (a) $x \in A^*$ is an N-tree; (b) if $\vec{u}_i$, $i < n$, are N-trees and $X \in N$, then $/\mathtt{(}X\vec{u}_0\vec{u}_1\cdots\vec{u}_{n-1}X\mathtt{)}/$ and $/\lightning\mathtt{(}X\lightning\vec{u}_0\vec{u}_1\cdots\vec{u}_{n-1}\lightning X\mathtt{)}\lightning/$ is an N-tree. The adjunction rules have the following form. Let $\langle\vec{x}_0,\vec{x}_1\rangle$ be a pair of strings such that $\vec{x}_0 = /\lightning\mathtt{(}X\lightning\cdots/$, $\vec{x}_1 = /\cdots\lightning X\mathtt{)}\lightning/$ and $\vec{x}_0\vec{x}_1$ is an N-tree. Such a pair shall be called an N-**adjunction tree**. Given this tree, let

$$\Lambda := \{\langle\vec{u},\vec{v},\vec{w}\rangle : \vec{u}\,\vec{v}\,\vec{w} \text{ is an } N\text{-tree}, \vec{v} = \mathtt{(}X\cdots X\mathtt{)}\}. \tag{2.90}$$

The pair $\langle\langle\vec{x}_0,\vec{x}_1\rangle,\Lambda\rangle$ is called a **tree adjunction rule**. Notice that the category X of the adjunction string must match the X in the locale. Also, the presence of $\lightning$ blocks adjunction at a node. (The symbol $\lightning$ is not needed to code the tree structure; its sole purpose was to restrict adjunction.) There are many variants of TAGs. We have picked the most common form for comparison. The language generated by a TAG G is however not the string language; rather it is the language of yields. This is defined as follows.

$$\begin{aligned} h^{\lightning}(x) &:= \begin{cases} \varepsilon & \text{if } x \in N \cup \{\mathtt{(},\mathtt{)},\lightning\}, \\ x & \text{else.} \end{cases} \\ h^{\lightning}(x_0x_1\cdots x_{n-1}) &:= h^{\lightning}(x_0)h^{\lightning}(x_1)\cdots h^{\lightning}(x_{n-1}) \end{aligned} \tag{2.91}$$

Then $L_Y(G) := h^{\lightning}[L(G)]$ is the language of the yields, which is by definition the language generated by G.

Exercise 2.19 Verify that the grammars from Examples 2.21 and 2.22 are contextual grammars.

Exercise 2.20 Let Ω be an arbitrary signature. Write an adjunction grammar for all terms in Polish Notation in this signature.

Exercise 2.21 Let Ω be an arbitrary signature. Terms in this signature are now written as follows. If f is binary and s and t are terms, then $/\mathtt{(}^\frown s^\frown f^\frown t^\frown\mathtt{)}/$ is a term. If f is unary then $/f^\frown\mathtt{(}^\frown s^\frown\mathtt{)}/$ is a term. If f is ternary and higher order, then $/f^\frown\mathtt{(}^\frown s_0^\frown\mathtt{,}^\frown\mathtt{,}^\frown\cdots^\frown\mathtt{,}^\frown s_{\Omega(f)-1}\mathtt{)}/$ is a term. Use the previous exercise to derive an adjunction grammar for this language.

2.5 Syntactic Structure

Contemporary linguistics insists that what matters is not the string that we see but rather its *structure*. Structure usually means tree structure. It has been stressed by Chomsky that rules operate on constituents and not on strings. Moreover, Transformational Grammar uses representations that contain the structure in them. Formally, however, it is not clear whether the structure needs to be represented. In this section I shall discuss a popular way of encoding structure into the string. Moreover, we shall investigate to what extent a context free language determines the grammar from which it is generated.

Let us take a look at CFGs and tree structures. Given a string $\vec{x}$ and a grammar G that generates it, G assigns a structure to $\vec{x}$ through a term in the following way. Assume a term t for the string $\vec{x}$. Then $t = f_\rho(s_0, \cdots, s_{n-1})$, where $n = \Omega(f_\rho)$.

$$\rho = A \to \vec{x}_0 B_0 \vec{x}_1 B_1 \cdots B_{n-1} \vec{x}_n \tag{2.92}$$

If $n = 0$ then $\rho = A \to \vec{x}_0$ and we just let the tree consist in two nodes, one with label $\vec{x}_0$, and a preterminal with label A. In general, we create a daughter for each B_i and attach the tree for s_i there and a daughter for every nonempty $\vec{x}_i$ whose label will be $\vec{x}_i$ (we avoid positing empty words).

We can code the derivation with a string. This is done by switching to a grammar that distributes brackets, called G^b. This grammar is defined as follows. We introduce for each nonterminal symbol X a pair of brackets $\mathtt{(}_X$ and $\mathtt{)}_X$. Let $\rho = X \to \vec{Y}$ be a rule. Then

$$\rho^b := X \to \mathtt{(}_X \vec{Y} \mathtt{)}_X \tag{2.93}$$

G^b contains in place of the rules R the set

$$R^b := \{\rho^b : \rho \in R\}. \tag{2.94}$$

Let $\vec{x}$ be a string. Each term t of $\vec{x}$ can be mapped to a term t^b, which is defined by replacing every occurrence of f_ρ by an occurrence of f_{ρ^b}, for every ρ. Thus mapping t into a term t^b of the bracketed grammar we find the string $\vec{x}_t$, which contains a record of t. $\vec{x}$ is obtained from $\vec{x}_t$ by deleting the brackets and the category symbols. More exactly, define a map d as follows.

$$\begin{aligned} d(a) &:= \begin{cases} \varepsilon & \text{if for some } X\text{: } a = \mathtt{(}_X \text{ or } a = \mathtt{)}_X, \\ a & \text{else.} \end{cases} \\ d(x_0 x_1 \cdots x_{n-1}) &:= d(x_0) d(x_1) \cdots d(x_{n-1}) \end{aligned} \tag{2.95}$$

Notice that the mapping d is many to one, since a given string can have many derivations. Notice also that there may be derivations that lead to the same bracketed string. Thus the structure is intermediate between the string and the derivation, adding detail to the string but not enough to recover the entire derivation.

Example 2.23 Let $G = \langle$:blet:, $\{\mathtt{E}, \mathtt{A}, \mathtt{B}\}, \mathtt{E}, R\rangle$ where R contains the following rules:

$$\begin{aligned} &\mathtt{E} \to \mathtt{AB} \mid \mathtt{BA} \mid \mathtt{EE} \\ &\mathtt{A} \to \mathtt{AE} \mid \mathtt{EA} \mid \mathtt{a} \\ &\mathtt{B} \to \mathtt{BE} \mid \mathtt{EB} \mid \mathtt{b} \end{aligned} \tag{2.96}$$

Now $G^b = \langle$:blet: $\cup \{(_{\mathtt{E}},)_{\mathtt{E}}, (_{\mathtt{A}},)_{\mathtt{A}}, (_{\mathtt{B}},)_{\mathtt{B}}\}, \{\mathtt{E}, \mathtt{A}, \mathtt{B}\}, \mathtt{E}, R^b\rangle$, with R^b consisting in

$$\begin{aligned} \mathtt{E} &\to (_{\mathtt{E}}\mathtt{AB})_{\mathtt{E}} \mid (_{\mathtt{E}}\mathtt{BA})_{\mathtt{E}} \mid (_{\mathtt{E}}\mathtt{EE})_{\mathtt{E}} \\ \mathtt{A} &\to (_{\mathtt{A}}\mathtt{AE})_{\mathtt{A}} \mid (_{\mathtt{A}}\mathtt{EA})_{\mathtt{A}} \mid (_{\mathtt{A}}\mathtt{a})_{\mathtt{A}} \\ \mathtt{B} &\to (_{\mathtt{B}}\mathtt{BE})_{\mathtt{B}} \mid (_{\mathtt{B}}\mathtt{EB})_{\mathtt{B}} \mid (_{\mathtt{B}}\mathtt{b})_{\mathtt{B}} \end{aligned} \tag{2.97}$$

The string /abab/ can be derived in G in several ways. One is given by the sequence /E/, /EE/, /ABE/, /ABAB/ and so on; another is given by the sequence /E/, /AB/, /AEB/, /ABAB/ and so on. These derivations give rise to the following bracketed strings:

$$\begin{gathered} (_{\mathtt{E}}(_{\mathtt{E}}(_{\mathtt{A}}\mathtt{a})_{\mathtt{A}}(_{\mathtt{B}}\mathtt{b})_{\mathtt{B}})_{\mathtt{E}}(_{\mathtt{E}}(_{\mathtt{A}}\mathtt{a})_{\mathtt{A}}(_{\mathtt{B}}\mathtt{b})_{\mathtt{B}})_{\mathtt{E}})_{\mathtt{E}} \\ (_{\mathtt{E}}(_{\mathtt{A}}\mathtt{a})_{\mathtt{A}}(_{\mathtt{B}}(_{\mathtt{E}}(_{\mathtt{B}}\mathtt{b})_{\mathtt{B}}(_{\mathtt{A}}\mathtt{a})_{\mathtt{A}})_{\mathtt{E}}(_{\mathtt{B}}\mathtt{b})_{\mathtt{B}})_{\mathtt{B}})_{\mathtt{E}} \end{gathered} \tag{2.98}$$

Erasing the brackets returns the original string. The derivation /E/, /EE/, /EAB/, /ABAB/ on the other hand yields the first string again. ⚽

Proposition 2.6 *G^b is unambiguous.*

The proof is straightforward. It rests on the usual bracket count of embeddings.

Notice however that the structure of $\vec{x}$ is a derived notion and the bracketed string just a theoretical construct. The structure is actually an *epiphenomenon*. It may be used in theoretical discourse but is in principle eliminable. This will have to be reassessed when we turn to interpreted grammars. We discuss the definitions and results first in the context of CFGs. We shall now discuss the notion of constituent occurrence without adding brackets. Recall the definition of an occurrence from Definition 2.9. Given a grammar G and a term t we can assign constituent occurrences of substrings in a straightforward way. Choose a subterm occurrence s and decompose t into $t = t'(s)$. This means that $t'(x_0)$ is a term with one free variable and it defines a function $\iota_G(t'(x_0)) : \vec{x} \mapsto \vec{u}\,\vec{x}\,\vec{v}$. This means that $\langle \vec{u}, \vec{v}\rangle$ is a 1-context and the substring that occurs in t is $\iota_G(s)$. For a constant term t, $\mathrm{occ}(\vec{y}, t)$ is the set of occurrences of $\vec{y}$ in $\iota_G(t)$.

This definition basically repeats what is intuitively known. Moreover, from the derivation we can uniquely assign a category to the string occurrence. The following formalizes the known substitution principle.

Definition 2.20 Let G be a CFG, t an A-analysis of the string $\vec{x}$ and C an occurrence of $\vec{y}$ in $\vec{x}$. If $C \in \mathrm{occ}(\vec{y}, t)$ then C is said to be a **constituent occurrence of $\vec{y}$ in $\vec{x}$ under the analysis** t. If $C \notin \mathrm{occ}(\vec{y}, t)$, the occurrence is said to be an **accidental B-occurrence under** t if $\vec{y} \in L_B(G)$. G is **transparent** if no constituent has an accidental occurrence in a string of $L(G)$. A language is **transparent** if it has a transparent grammar.

Notice that we look at occurrences under a given analysis term t. A given string $\vec{x}$ can in principle have several analyses. Suppose that a context free language L is transparent. Then given a string $\vec{x} \in L$ we know that every substring occurrence of $\vec{x}$ that is in L also is a constituent occurrence under every analysis. Thus any context free grammar will assign the same constituent tree to $\vec{x}$. This is very useful for languages of analysis terms, because we need to know that they are structurally unique. This is the case for $\mathrm{Tm}_\Omega(V)$, as the next theorem asserts.

Proposition 2.7 *The language* $\mathrm{Tm}_\Omega(V)$ *is transparent.*

Proof (For notation and facts see Exercises 2.6 and 2.7.) Let s and t be terms and $C = \langle \vec{u}, \vec{v} \rangle$ an occurrence of s in t. We shall show that s is actually a subterm occurrence of t by induction on t. For either (a) $\vec{u} = \varepsilon$ or (b) $\vec{u} \neq \varepsilon$. If (a) is the case then $\vec{v} = \varepsilon$, that is, $s = t$, or else s is a proper prefix. This cannot be, since this would imply $\gamma(s) \geq 0$. Now in case (b) there is an i such that the named occurrence begins in t_i. (Case 1) The occurrence is contained in t_i, that is, $t_i = \vec{x} s \vec{y}$ for some $\vec{x}$ and $\vec{y}$. Then we are done by inductive hypothesis. (Case 2) s overlaps with t_i. Then we have $\vec{x}$, $\vec{y}$ and $\vec{z}$ all nonzero such that $t_i = \vec{x}\,\vec{y}$ and $s = \vec{y}\,\vec{z}$. Now note that since $-1 = \gamma(t_i) = \gamma(\vec{x}) + \gamma(\vec{y})$ and $\gamma(\vec{x}) \geq 0$ (since t_i is a term) we must have $\gamma(\vec{y}) < 0$. But then s is not a term since $\gamma(\vec{y}) \geq 0$ if $\vec{y}$ is a proper prefix. So this case does not arise and we are done. □

Every constituent occurrence in $\vec{x}$ under t corresponds to a subterm occurrence in t. We use this for the following definition. A term is simple if it has no nontrivial subterms.

Definition 2.21 Let G be a CFG, t an A-analysis of the string $\vec{x}$ and C an occurrence of a letter b in $\vec{x}$. C is **syncategorematic** if the term to which b belongs is not simple. A substring occurrence is syncategorematic if every letter is syncategorematic and belongs to the same subterm. G is in **standard form** if no string has syncategorematic occurrences.

This definition can easily be generalized. For a CFG, being in standard form means that the right-hand side of a rule cannot contain both a nonterminal and a terminal letter. For example, the standard formulation of regular grammars is that they have rules of the form $A \to xB$ or $A \to x$. Such grammars are not standard. It is easy to convert a CFG into standard form. However, notice that this changes the language of the grammar, since for us the language contains all constituents.

Example 2.24 We continue Example 2.23 above. The first derivation given by the sequence /E/, /EE/, /ABE/, /ABAB/, /aBAB/, /abAB/, /abaB/, /abab/. In the string we have the constituent occurrences $\langle \varepsilon, \varepsilon \rangle$, $\langle \varepsilon, \texttt{ab} \rangle$, $\langle \texttt{ab}, \varepsilon \rangle$ of category E; the occurrences $\langle \varepsilon, \texttt{bab} \rangle$ and $\langle \texttt{ab}, \texttt{b} \rangle$ of category A; and the occurrences $\langle \texttt{a}, \texttt{ab} \rangle$ and $\langle \texttt{aba}, \varepsilon \rangle$ of category B. The string /ab/ has an accidental occurrence $\langle \texttt{a}, \texttt{b} \rangle$. The string /aa/ has no accidental occurrence although it is a substring of /aabb/. ⚽

Proposition 2.8 *Let G be a CFG, $\vec{x} \in L(G)$ and t an analysis term. Fix a constituent occurrence of $\vec{y}$ in $\vec{x}$ under t. If $\vec{y}$ occurs as A in the context C, and $\vec{z}$ is any string of category A of G, then $C(\vec{z}) \in L(G)$.*

Suppose now that we wish to give a syntactic analysis of a string language L. We assume that the analysis is given in terms of a CFG. If this is so, we know that the set of strings of L fall into finitely many classes, say, S_i for $i < n$ and that if $\vec{x}, \vec{y} \in S_i$ then each constituent occurrence of $\vec{x}$ can be substituted by $\vec{y}$ and each constituent occurrence of $\vec{y}$ can be substituted by $\vec{x}$. This superficially looks like a way to discover the grammar behind a given language.

The problem with this idea is that we do not know whether a given occurrence is a constituent occurrence. However there is one exception: a single letter wherever it occurs can only occur as a constituent on condition that the grammar contains no syncategorematic occurrences of symbols. It is easy to massage any CFG into such a form without losing anything.

Example 2.25 The language of equations. In the form presented in Example 2.18 on page 34. This grammar introduces /=/, the operation symbols and the brackets in a syncategorematic way. It can be reformulated as follows. The original rule set is

$$\begin{aligned} &\mathtt{E} \to \mathtt{T{=}T} \\ &\mathtt{T} \to \mathtt{(T{+}T)} \mid \mathtt{(T{-}T)} \mid \mathtt{B} \\ &\mathtt{B} \to \mathtt{B0} \mid \mathtt{B1} \mid \mathtt{0} \mid \mathtt{1} \end{aligned} \tag{2.99}$$

Now introduce a nonterminal for each symbol. For example, introduce /O/, /C/, /Q/ together with the unary rules

$$\begin{aligned} &\mathtt{O} \to \mathtt{(} \\ &\mathtt{C} \to \mathtt{)} \\ &\mathtt{P} \to \mathtt{+} \\ &\mathtt{M} \to \mathtt{-} \\ &\mathtt{Q} \to \mathtt{=} \end{aligned} \tag{2.100}$$

Next replace the occurrence of the syncategorematic symbols above by the corresponding nonterminal:

$$\begin{aligned} &\mathtt{E} \to \mathtt{TQT} \\ &\mathtt{T} \to \mathtt{OTPTC} \mid \mathtt{OTMTC} \mid \mathtt{B} \\ &\mathtt{B} \to \mathtt{B0} \mid \mathtt{B1} \mid \mathtt{0} \mid \mathtt{1} \end{aligned} \tag{2.101}$$

It is possible to simplify this grammar; we group /P/ and /M/ into just one symbol, say, /H/. Then we have the following rule set:

$$\begin{aligned} &\mathtt{O} \to \mathtt{(} \\ &\mathtt{C} \to \mathtt{)} \\ &\mathtt{H} \to \mathtt{+} \mid \mathtt{-} \\ &\mathtt{Q} \to \mathtt{=} \\ &\mathtt{E} \to \mathtt{TQT} \\ &\mathtt{T} \to \mathtt{OTHTC} \mid \mathtt{B} \\ &\mathtt{B} \to \mathtt{B0} \mid \mathtt{B1} \mid \mathtt{0} \mid \mathtt{1} \end{aligned} \tag{2.102}$$

⚽

Notice that the grammars (2.100) and (2.102) are not only different grammars; they in fact generate different languages. For example, the string `/(/` is in the language of (2.102) but not in (2.100). This is a consequence of the fact that we defined $L(G)$ to contain *all* constituents of G, not just the sentences.

Let us now turn to the idea of recovering the grammar from the set of strings. We start with the assumption that our language is generated by a context free grammar. This means that constituents are strings, and that a string is a part of another string only if it is a subword. The standard substitution method starts with the language L and establishes for every $\vec{x} \in L$ the set of contexts:

$$\mathrm{cnt}_L(\vec{x}) := \{\langle \vec{u}, \vec{v} \rangle : \vec{u}\,\vec{x}\,\vec{v} \in L\}. \tag{2.103}$$

The syntactic classes are the context sets so obtained. We present an example first.

Example 2.26 (Continuing Example 2.11.) The language M is generated by u, defined by

$$\begin{aligned} t &:= \texttt{Alex}_\sqcup \mid \texttt{Pete}_\sqcup \mid \texttt{Mary}_\sqcup \\ u &:= t(\texttt{and}_\sqcup t)^*(\texttt{sing}_\sqcup \mid \texttt{run}_\sqcup \mid \texttt{sings}_\sqcup \mid \texttt{runs}_\sqcup) \end{aligned} \tag{2.104}$$

We consider words as units together with a following blank. (This makes the calculations easier.) The context sets are as follows. Here is a more succinct definition of the language:

$$\begin{aligned} a &:= \texttt{and}_\sqcup \\ v &:= \texttt{sings}_\sqcup \mid \texttt{runs}_\sqcup \\ w &:= \texttt{sing}_\sqcup \mid \texttt{run}_\sqcup \\ u &= tv \mid t(at)^+w \end{aligned} \tag{2.105}$$

It turns out that the syntactic classes are the following: $\mathrm{cnt}_M(v)$, $\mathrm{cnt}_M(tv)$, $\mathrm{cnt}_M(w)$, $\mathrm{cnt}_M(a)$, $\mathrm{cnt}_M(t)$, $\mathrm{cnt}_M(ta)$, $\mathrm{cnt}_M(at)$, $\mathrm{cnt}_M(tat)$, $\mathrm{cnt}_M(atat)$, $\mathrm{cnt}_M(atw)$, $\mathrm{cnt}_M(tatw)$. These are more classes and more constituents than were present in the original grammar even if we massaged the syncategorematic occurrences away. ⚽

The exercises give one more example. The problem with the substitution method is that there is no way of telling whether an occurrence is accidental or not. Consequently, the method will return context sets that are the sets of nonconstituent occurrences. In fact, we may end up with infinitely many context sets (see the exercises). And this is not because of the finiteness of the data: even if we had all data in our hands, the grammar is still underdetermined. Thus, there is some art involved in establishing the subset of constituent occurrences. This set can be different from the one for the original grammar. However, in the absence of decisive evidence this is the best one can do.

Under certain circumstances we can know in advance that the set of nonterminals is going to be finite. A particular case is provided by *primitive languages*.

Definition 2.22 A language is called **primitive** if every substitution class contains a string of length 1, that is, consisting in a single letter.

Evidently, since the alphabet is finite, there are finitely many substitution classes. This does not guarantee the uniqueness of the solution (see the Exercises) but it narrows the choice considerably.

The language defined in Example 2.25 is not primitive. This is because the set of E-strings (which form a substitution class!) consists in strings of length of at least 3: an equation sign, and two terms on either side. Terms may not be empty, they have a length of at least 1.

Primitive languages can easily be turned into CFGs. Just observe that for each letter a there is a substitution class $[a]_L$. Let N_a be the nonterminal representing this class (if $[a]_G = [b]_G$ then also $N_a = N_b$). The rules are of the form

$$\begin{array}{ll} N_a \to a & a \in A \\ N_a \to N_{c_0} N_{c_1} \cdots N_{c_n} & c_0 c_1 \cdots c_n \in [a]_L \end{array} \tag{2.106}$$

This set is typically infinite but a finite subset is enough to generate L, by assumption on L.

We shall finally turn to the abstract case.

Definition 2.23 Let u and v be constant Ω-terms and G a grammar. We say that u and v are **categorially equivalent**, in symbols $u \sim_G v$, if for all terms $s(x)$: $s(u)$ is orthographically definite if and only if $s(v)$ is. They are **intersubstitutable**, in symbols $u \approx_G v$, if and only if they are categorially equivalent and $\iota(s(u)) \in L(G)$ if and only if $\iota(s(v)) \in L(G)$.

This definition does not talk about strings; it talks about terms. This is because the term may be very complex while the string is very simple. Moreover, in absence of any condition on the form of the rules it is not possible to assign any sensible structure to the string.

Example 2.27 Here is a context sensitive grammar, consisting in the following rules.

$$\begin{aligned} \mathtt{S} &\to \mathtt{ATB} \\ \mathtt{T} &\to \mathtt{x} \mid \mathtt{xT} \\ \mathtt{Ax} &\to \mathtt{xA} \\ \mathtt{AB} &\to \mathtt{y} \end{aligned} \tag{2.107}$$

In a derivation, first /A/ is generated to the left of the string. However, when the last rule applies, /A/ has to be to the right. The system of constituents formed by this grammar is quite confusing. It puts the occurrence of /y/ into a constituent with all occurrences of /x/ (for each occurrence of /x/ there is a separate constituent, though). ⚽

Notice also that adjunction grammars in the general form may fail to allow for an unequivocal assignment of structure. This is why tree adjunction grammars work differently from string adjunction grammars. In TAGs the constituent structure is by definition preserved while in string adjunction grammars it need not be.

Exercise 2.22 Show that the substitution classes of a context free grammar (construed as a grammar in the sense of this book in the straightforward way) are of the following form. Let N be the set of nonterminals, and $P \subseteq N$. Then a string $\vec{x}$ is said to be of *class* P if for all $Y \in N$: $Y \Rightarrow^* \vec{x}$ if and only if $Y \in P$.

Exercise 2.23 Apply the method of context sets to grammar C_1 of Example 2.9. Show that the grammar that this gives is C_2 (also from Example 2.9)! Show that the language generated by either grammar is primitive.

Exercise 2.24 Let G consist in the rules $\texttt{S} \to \texttt{ab} \mid \texttt{aSb}$. Establish the context sets of all substrings and show that there are infinitely many of them. Show that infinitely many context free grammars can be postulated on the basis of these sets.

Exercise 2.25 Let G be a context free grammar. Try to establish an inductive definition of $\mathrm{occ}(\vec{y}, t)$. *Hint.* This definition will have to be inductive in the length of $\vec{y}$ and t.

2.6 The Principle of Preservation

We have seen that the effect of substitution is unpredictable unless restrictions are placed on the nature of the string functions. We propose here two principles that simplify the situation. In the most ideal case, functions are only able to change a string by appending material to its left or right. If we required this we get something slightly more general than context free grammars. To get some more freedom we propose that grammars do not operate on the set A^* but on some slightly more general set of exponents, which we equate with $(A^*)^m$ for some m, or perhaps $\bigcup_{m \in \mathbb{N}} (A^*)^m$, as proposed in Kracht (2003).

Principle 2 (Structure) ***Exponents*** *are sequences of strings.*

This is a heavy restriction but it still allows substantial freedom, more than is immediately apparent. First of all, we have not said anything at all about the alphabet from which the strings are formed. In conjunction with the Principle of Structure Preservation this will simply be equivalent to saying that letters are alphabetic letters; but I think matters are not that easy. The problems of this viewpoint will be discussed below. Let us for the moment remain with the idea that the alphabet is simply the standard typographical alphabet. Then exponents are strings of that alphabet—or, as I proposed above, sequences thereof. This latter qualification is important. Consider the following principle.

Principle 3 (Structure Preservation) *A rule may not break any string of the exponent or delete any parts of it.*

This can be formalized as follows. The interpretation of a function symbol f is a function from $\Omega(f)$ many m-tuples to a single m-tuple of strings. So, $\mathcal{I}(f) = \langle t_0, t_1, \cdots, t_{\Omega(f)-1}\rangle$, where the t_i are terms in $m < \Omega(f)$ variables that are polynomial functions in the string algebra

$$\langle A^*, \varepsilon, {}^\frown\rangle \tag{2.108}$$

over the signature $\Omega_\frown := \{\langle\varepsilon, 0\rangle, \langle{}^\frown, 2\rangle\}$. This means further that t_i may use variables, constants for letters of A and for the empty string and concatenation.

What these principles rule out is deletion of any kind; they also rule out breaking a constituent. However, what they *do* allow is discontinuity. A constituent may consist in a bounded number of parts. Typically, we find that constituents consist in just 1 or 2 strings. Examples of the latter kind are the verbs of German (after verb-second has applied), the crossed dependencies in Dutch infinitives and split-DPs. Occasionally we find languages that seem to have arbitrarily fragmented DPs, like Warlpiri or Jiwarli. However, even in the case of these languages it is not entirely clear that the approach does not work; for these languages do not break embedded clauses either. This needs further work.

We have so far only spoken about breaking or deleting strings. The next principle talks about rules in the sense of nonconstant functions (see Definition 2.3).

Principle 4 (Syncategorematicity Prohibition) *A rule may not add any occurrence of a given symbol.*

Again, this can be formalized by saying that the interpretation of functions uses only definable term functions in $\Omega_\frown$, not polynomials. This allows for complete reduplication (as in Malay) and it also allows for partial reduplication (modulo a regular relation), as long as the parts can be represented as strings. The way it does so is by stipulating that a given string may be repeated. This in fact does *not* mean that a fixed symbol is introduced since the nature of the string to be reduplicated is unknown. An alternative to reduplication is the following. We allow to concatenate two strings $\vec{x}$ and $\vec{y}$ *on the condition that they are identical*. Thus, the formation of the plural in Malay can be expressed in two ways: by a reduplication rule, using a function

$$r(x) = x^\frown x \tag{2.109}$$

or by partial concatenation, using the function

$$c(x, y) = \begin{cases} x^\frown y & \text{if } x = y, \\ \text{undefined} & \text{else.} \end{cases} \tag{2.110}$$

The advantage of the latter is that every occurrence of a letter can be uniquely traced back to a leaf. The disadvantage is that it creates too many substitution classes.[1] Apart from this it is hard to distinguish this approach from the one based on duplication, the more so since the rule is completely general and the categories will anyway turn out to be eliminable from the formulation of a grammar.

Another hard case to treat is the so-called **tmesis**. This is the coordination of parts that are not words by themselves. For example, in German we have the words `/Urfeind/` and `/Erzfeind/`, both formed from `/Feind/` "enemy" and a prefix `/Ur/` "since very long ago" and `/Erz/` "arch-". What is striking is that while neither prefix can be on its own, it is possible to say

`Ur- und Erzfeind` (2.111)

Similarly, verbal prefixes can be separated

`auf- und abladen` "load and unload" (2.112)

Tmesis can be applied at the juncture of compounds and with certain prefixes. It is in particular not free to apply to any morphological part of the word. A proper formulation of tmesis under the conditions just sketched is not impossible but requires great care.

What the principle does *not* allow is the addition of any concretely specified symbol. For example, it may not say: "add an `/s/` at the end". This must be represented alternatively as a binary rule concatenating the string to $\vec{x}$. Again, requiring this we do not so much restrict *what* can be done but rather *how* it can be done. Yet, there is a problem with this requirement and it runs as follows. We practically assume that bound forms are also part of the language; that is, the plural `/s/` of English, even though it cannot occur on its own, is part of the English language. However, this might just be an artefact of the requirement that only words are free forms; and we may say that the language consists in more that just the free forms. The semantics of the plural on the other hand is unproblematic or at least not more problematic than that of any other item.

Now we turn to the question of the alphabet and the nature of the underlying strings. Here, as so often, no unique solution can be given. Two extremes exist: on the one hand we have alphabetic systems that are more or less sound based (with complications of their own). On the other we have ideographic systems like Chinese, which make a single letter correspond (again more or less) to a morpheme. Chinese presents a good example of the predicament we are facing: if we base our analysis on the *sounds* then there are about 100 letters (vowels in four tones plus

[1] If we look at this rule in combination with semantics (anticipating the next chapter) we find that the reduplication approach will form the plural in the semantics by performing the step from properties of individuals to properties of sets of individuals. The partial concatenation approach however makes the plurals appear more like *dvandva*-compounds. The idea is that in the Malay plural noun `/anak-anak/` "children", we get the plural meaning from extrapolating a dvandva from "child" and "child" rather than (the more natural) dvandva formed from different parts.

consonants), or maybe somewhat more, given that pauses and intonation contours must be taken on board as well. If, however, we base our analysis on the alphabet of *characters* then we have an alphabet of up to 50,000 "letters". (The Chinese Standard Interchange Code, the most comprehensive of the lists, has close to 50,000 characters.) The question that naturally arises is this: which of the two should we choose? In principle, it seems, we should be able to do both but writing systems can be so artificial that it seems we ought to exclude some of them from the analysis.[2] But even if we do, the sound based approach presents difficulties of its own. One is that the notion of part is somewhat obscure. For example, we say that a string $\vec{x}$ is *part* of a string $\vec{y}$ if it is a subword. Thus, we may for example say that /eel/ is part of /reel/, or /ice/ is part of /rice/. If we apply our substitution tests, however, we get quite a bizarre picture of the language. Thus, we would like to apply substitution only to constituents, or, as we have said above, study those strings (or sequences) that can be substituted for a single letter. If *letter* can be equated with *morph*, or *morpheme*, we would get a far more interesting grammar from our substitution tests than if we insisted on sounds (or alphabetic characters). The disadvantage of the method is that it presupposes what it ought to reveal: the primitive parts. However, as we shall see in the next chapter, the notion of a morph(eme) makes perfect sense, because once we add the meaning the alphabetic characters are in fact *not* the most basic elements but the morph(eme)s.

In stratificational linguistics we actually pursue *both* analyses at once. There are various strata at which we have structure. Such frameworks have been pursued among others by Lamb (1966) and Mel'čuk (1993–2000). In our view the various levels are mostly epiphenomenal and can be reconstructed on the basis of the language (as a set) itself. I shall briefly discuss the reconstruction of levels in Section 3.7.

Even if all this is granted, we still face a number of problems. Suppose, for example, that our language is based on morphemes, which are the letters of our alphabet. Then, by our principles above, these letters must surface in our strings (or sequences of sounds). It follows that morphemes are sequences of characters of the alphabet. If that is so, we must address exceptions to strict concatenation. I mention here as representatives: final devoicing (as found in Russian and German, for example), vowel harmony (as found, say, in Finnish, Hungarian and Turkish), consonant lenition in Welsh, or consonant gradation in Sami (Svenonius, 2007). Let us discuss the first case. Final devoicing is a process that turns any consonant in the coda of a syllable into a voiceless consonant. For example, there are two nouns in German, /Rad/ [ʁa:t] "bicycle" and /Rat/ [ʁa:t] "council". They sound exactly the same. On the other hand, their respective genitives, /Rades/ [ʁa:dəs] and /Rates/ [ʁa:təs], do not. The reason is that the rules of segmentation put the stop into the onset of the next syllable, where it does not undergo devoicing. If we base ourselves

2 There was once a way to write in Japan that used only Chinese characters and even Chinese word order. The characters were augmented with numbers so that one knew in which way to read the characters. Now, not only do the characters come out differently (the character for mountain is read "yama" in Japanese and "shān" in Chinese), but they are also arranged according to Chinese syntax.

on the written forms, no problem. The sounds however do pose a problem. What can be the solution?

One solution ultimately rests on the distinction between complete and incomplete forms. Suppose that the base form comes without word end markers. So they would be [ʁa:d] and [ʁa:t], respectively. Now, when we attempt to pronounce such a word, we must speak it in isolation, so we add a word boundary marker to its left and right: [#ʁa:d#] and [#ʁa:t#]. After that, there is a process that will produce the required form. This solution does explain the different outcomes but it falls short of complying with the Principle of Preservation. This applies to all other phenomena listed above, which is why we have mentioned them. We shall therefore relax this principle a little bit. We shall assume that it is not the actual surface forms that must be preserved but a more abstract form.

If we left matters at that we would basically remove all restrictions. We need to restrict the abstraction. This is done as follows. We operate now with two levels: SP (the surface phonological level) and DP (the deep phonological level). Each of the levels uses the same alphabet (tentatively). The principles apply only to DP. The actual strings of SP are obtained by applying a finite state transducer. In other terms, the relation between DP and SP is a *regular relation* (see Kracht (2003) for definitions and discussion). To account for German devoicing, we assume that at DP no devoicing applies. The relation to SP, however, is such that every consonant that happens to be syllable final is devoiced. This can be achieved using a finite state transducer.

Let us briefly touch on the question of c-languages. If one wishes to include categories into the language then the Principle of Preservation loses some of its bite. It would namely be possible to introduce material into the category part where it is invisible to the principles formulated above. I assume therefore that when categories are added they cannot introduce a finer distinction than already present in the functions.

Principle 5 (Categorial Granularity) *For a c-grammar G and the associated string grammar H, if $\langle \vec{x}, c\rangle \in L(G)$ and $\vec{y} \sim_H \vec{x}$ then also $\langle \vec{y}, c\rangle \in L(G)$.*

Thus, the set of categories cannot differentiate the exponents in a finer way than the string functions. The way this is phrased makes the principle somewhat circular. But you need to recall that the string categories are derived from the string functions and ultimately from the language itself. Thus, bringing in an extra set C of categories really is to serve the purpose of explicitly coding the categorial facts rather than bringing back a lost dimension. However, I should note that adding categories even with the Granularity Principle brings in extra power.

Exercise 2.26 German nouns are written with an initial upper case letter. However, in compounds only the first letter is in upper case. For example, /`Auto`/ "car" and /`Bahn`/ "way" result in the compound /`Autobahn`/ "highway". (Observe similarly /`Erzfeind`/ in the example above.) Propose a solution to this. *Hint.* There are (at least) two solutions. One uses the regular relations, the other proposes several forms for the same word.