


2. Syntactics

1. Subdivisions of syntactics
2. Formal aspects of signs
3. Relations between signs
4. Complex signs
   4.1. Analysis and synthesis
   4.2. The concept of a string
   4.3. String codes
   4.4. Production grammars
   4.5. Regular grammars
   4.6. Context-free grammars
   4.7. Context-sensitive grammars
   4.8. Grammars and automata
   4.9. Syntactic structures and structural descriptions
   4.10. Transformational grammars
5. Recent developments in formal syntactics
   5.1. X-bar syntax, subcategorization, and thematic roles
   5.2. Graphs, markedness, and unification
   5.3. The direct formation of structure trees
   5.4. Multi-dimensional formation of sign systems and graph grammars
6. Selected references

1. Subdivisions of syntactics

Syntactics studies the conditions an entity must fulfil if it is to represent meaning for interpreters in semiosis (cf. Art.1). With respect to the traditional problem areas of semiotics, it is not always easy to decide whether they belong to the subject matter of syntactics or not (cf. Morris 1946: 219 ff = 1971: 303). In many cases the answer will be different according to which conception of syntactics it is based on:

- syntactics\(_1\) as the study of the formal aspects of signs (Morris 1929, 1937, and 1938: 13 ff = 1971: 27 ff)
- syntactics\(_2\) as the study of the relations of signs to other signs (Morris 1937, and 1938: 7 ff = 1971: 23 ff), or
- syntactics\(_3\) as the study of the way in which signs of various classes are combined to form complex signs (Morris 1938: 14 = 1971: 28 f and 1946: 354 ff = 1971: 367).

The areas of research defined by the three characterizations overlap but are not identical. In what follows we will therefore use the term “syntactics” without a subscript only when we speak of syntactics proper, i.e., the science whose subject matter is the intersection of the subject matters of syntactics\(_1\), syntactics\(_2\), and syntactics\(_3\).
2. **Formal aspects of signs**

The question as to what constitutes the formal aspects of signs has been given alternative answers by different traditions of semiotics (cf. Art.1 § 2). While logic-oriented semioticians like Wittgenstein (1922: 3.33), Carnap (1934: 1, 208 and 1938: 16), and Morris (1938: 13ff = 1971: 27ff) equate the distinction between form and substance with the distinction between signs (B's) and meanings (C's), linguistics-oriented semioticians like Saussure (1916: 155–157) and Hjelmslev (1943: 69–73 and 98ff = 1963: 76–81 and 110ff) use it to differentiate two kinds of aspects within signs (B's) as well as within meanings (C's). Yet the underlying conceptions of form are quite similar and are based on the theory of invariants as developed in geometry; cf. the “Erlangen Program” of F. Klein (1872: 463f): “A generalization of geometry raises the following problem: Let there be a multiplicity of transformations defined on them; now the configurations of elements in that multiplicity should be studied with respect to those properties that remain unchanged when transformations of that group are applied to them […]. The task is to develop the theory of invariants for that group.” In such configurations of elements two different types of entities may vary. On the one hand, certain properties of the configurations can change while others remain constant, as is the case in the movement of an object in space, where the location varies but all spatial proportions of the elements involved stay the same. On the other hand, the elements themselves can also change, as when geometrical figures are replaced by sequences of letters or numbers. What remains invariant here is the abstract structure of the figure (cf. Holenstein 1975: 30ff and Wunderlich 1971). More complicated cases in point are mirroring in visual space, where the location and orientation of a figure changes but all of its spatial proportions remain the same, and the transposition of a musical chord in auditive space, where the absolute pitches of the tones change but the structure of the pitch relations is taken to remain the same, thus determining the pitch space of tonal music.

The structure of a relation was extensionally defined (in a type-theoretic setting) by Russell (1919: 59ff) and Carnap (1928: 13ff) as the class of relations isomorphic with that relation. This approach was developed further (within set theory) in the mathematical theory of structures published by the group of French mathematicians Bourbaki (1935ff). It is related to Hilbert’s doctrine of implicit definitions (cf. 1918 and 1925; cf. Art. 30 § 1.6). Hilbert proposed to define basic technical terms of a theory implicitly by specifying the axioms in which they occur. Discussing the development of mathematics, Hilbert wrote in 1925: “In intuitive number theory formulas were always exclusively used for communication. The letters stood for numbers and an equation communicated the fact that the two numbers coincided. In algebra, on the other hand, we regard expressions containing letters as independent structures which formalize the material theorems of number theory. In place of statements about numbers, we have formulas which are themselves the concrete objects of intuitive study. In place of number-theoretic material proof we have the derivation of a formula from another formula according to determinate rules” (cf. Putnam and Benacerraf 1964: 145). Carnap (1934) applied this idea to the study of language in general. He was convinced that for every language one can work out “a formal theory of the forms of that language” (1934: 1). It was the task of that theory to specify the rules that determine the sign forms of a language. For Carnap, the formal nature of the concrete individual sign was no problem, as it was for Hjelmslev (see below). Therefore he was content to characterize the sign forms of a language by specifying “the classes and the serial relations” of their components (1934: 1). Analogous to Hilbert’s program of Metamathematics, Carnap’s “formal theory” was to be formulated in a special metalanguage whose expressions refer to the object-language in question (1939: 5). Morris (1938: 9 = 1971: 23ff and 1946: 178ff = 1971: 256f) generalized this conception for sign systems of all kinds with all their aspects. If we call a sign system under study “an object-code”, Morris (1938: 13ff, 21ff, and 29ff = 1971: 28ff, 35ff, and 43ff) envisaged three different metalanguages dealing with three different dimensions of semioses involving signs in the object-code. The metalanguages differ in what their expressions refer to. While the pragmatic metalanguage refers to the interpreters (cf. Art. 4) and the semantic metalanguage refers to the meanings (cf. Art. 3) of the signs in the object-code, a metalanguage whose descriptive terms all refer only to the sign forms of the
object-code is called “a syntactic metalanguage”.

It is worth noting that each metalanguage has itself all properties of a sign system. Thus the syntactic metalanguage not only has a syntactic dimension insofar as it contains signs of various classes with various serial relations holding among them, it also has a semantic dimension since its signs are interpreted to represent meaning and refer to something, i.e., to the signs of the object-code, and it has a pragmatic dimension insofar as it can be used by special types of interpreters, i.e., by syntacticians. This being the case, it is the semantic dimension of the syntactic metalanguage that has to do with the syntactic dimension of the object-code. And it is the semantic metalanguage of the syntactic metalanguage that deals with the relations between signs of the syntactic metalanguage and signs of the object-code. Viewed from this perspective, the text of this article can be understood to be written in the pragmatic metalanguage of the syntactic metalanguages devised by semioticians of various traditions to deal with object-codes.

What has been said about the syntactic metalanguage so far takes care of the terms “forms of a language” in Carnap's program and “formal aspects of signs” in Morris's version thereof. Understood in this way, syntactics becomes a subdiscipline of syntactics2. There are, however, two problems remaining; one is the task of setting up a “formal theory” within the syntactic metalanguage (it will be treated in § 4 and 5; cf. also Art. 30 § 1.7.3.); the other is the problem of characterizing the formal aspects of a sign that is not a complex sign. The latter problem was approached by Hjelmslev, who, like Carnap (1928: 11), insisted “that a totality does not consist of things but of relationships, and that not substance but only its internal and external relationships have scientific existence […]” (Hjelmslev 1943: 22 = 1963: 23).

This approach had been successful in the theory of human speech sounds. Reflecting on Trubetzkoy's (1929) analysis of vowel systems, Bühler (1931) had distinguished the acoustic sign from the concrete sound event by saying that the former was connected with the latter as form is connected with uniformed matter. In 1934, Bühler described the complex relation between the properties of sign matter and sign form with reference to the areas of two overlapping geometrical figures (1934: 28 and 42 ff; cf. Art. 112): in order to recognize the sign form in the sign matter produced in semiosis, the interpreter must on the one hand concentrate on the relevant properties of the sign matter and abstract from the irrelevant ones (principle of abstractive relevance), and on the other hand complement properties of the sign form not manifested in the sign matter (principle of apperceptive complementation of the sign gestalt). The distinction between sign matter and sign form can be exemplified by the way different languages pattern the sign matter which Berliners produce in order to name their city: it becomes [ba:’ln] in English, [ber-li:n] in German, [ba’rl’n] in Danish, and [be-ul’nu] in Japanese. Hjelmslev, who uses this example (1943: 52 = 1963: 56), describes it by saying that one and the same sign matter (“purport”) is modeled into different sign substance through the sign forms supplied by the different languages. The sign forms comprise the language-specific invariants patterning sign matter. According to Hjelmslev, these invariants are independent of the medium involved; thus a given sign form can be realized by sign matter in various media, as when the speech chain of German [ber-li:n] is transformed into the written word Berlin. From this he concludes (1943: 91 f = 1963: 103 f): “Substance cannot itself be a definiens for a language [...]. Here is a graphic ‘substance’ which is addressed exclusively to the eye and which need not be transposed into a phonetic ‘substance’ in order to be grasped or understood.” What is true for language also holds for all other sign systems.

In phonology, the difference between substance and form is captured terminologically by the distinction between phonetic and phonemic entities studied by phonetics and phonemics, respectively. This distinction was generalized in American Structuralism by isolating the suffixes -etic and -emic (cf. Bloomfield 1933 and Pike 1954, 1966) and using them in the description of non-linguistic sign systems. Thus a description in terms of behavioremes considers all and only those properties of a given behavior that are relevant for it to represent meaning to interpreters of the kind in question, while a description including other aspects of behavior would be called “behavioretic”. The fruitfulness of this distinction became obvious in Western cultural anthropology in the 1950's and 60's when ethnomusicologists such as Bruno Nettl (cf. Nettl 1957 and 1983) discovered that the conceptual
framework of Western music theory was inappropriate for the description of music from other parts of the world (cf. Natieez 1987: 67–95). Chenoweth (1972) used methods of analysis developed by Pike (1954, cf. 1966) to determine the cultural relevance of musical intervals in culture-specific styles of singing.

If the members of a culture tolerate the replacement of an interval by an acoustically different one in a given melody without judging the resultant melody to be different from the original, the two intervals must be considered emically equivalent in that culture (cf. Goodman 1968 and Posner 1988: 916).

Occasionally, two etically different intervals are emically different in one context and emically equivalent in another. In phonology, such cases are known as neutralization (cf. Trubetzkoy 1939). Examples for neutralizing contexts in Western music are the minor scales, as pointed out by Monelle (1992: 60) with reference to Nettl (1957: 39): “In the key of C, the tones a and b are used as the minor variant of a pitch to be different from the original, the two intervals must be considered emically equivalent in that culture (cf. Goodman 1968 and Posner 1988: 916).”

What is true of tones and intervals also applies to duration and loudness so that one can postulate tonemes as well as rhythmemes and dynamemes in music (cf. Seeger 1960: 229ff). Analogously, chromemes, formemes and texturemes are postulated as elementary sign forms in visual signs (cf. Groupe μ 1992: 197). The general conclusion to be drawn from this is that the sign forms in the various physical media (for the concept of media cf. Art. 14) are constructs of categorical perception, which are learned by infant organisms through culture-specific deformation of initial categorizations (cf. Eimas 1980 for speech perception, Burns and Ward 1978 for music perception and Saint-Martin 1987: ch. 2 for visual perception; see also Art. 6–11). Categorical perception can be mathematically described by means of catastrophe theory, which explains how small changes in certain parameters of a stimulus can lead to radical changes in the categorization of the percept (cf. Thom 1972 and 1980 as well as Petitot-Cocordia 1985 a and b; see also Wildgen 1982).

In summary, it should be noted that syntactics, studies sign forms and disregards sign substance and sign matter. Within linguistics, syntactics1 includes phonemes but excludes physical phonetics; within musicology it includes tonemes, rhythmemes and dynamemes but excludes physical acoustics; within the study of visual signs it includes chromemes, formemes and texturemes but excludes physical optics.

This delimitation, which relies on the differences between concrete and abstract, should not be confused with a distinction introduced by Peirce (C. P. 4.537) and developed further by Reichenbach (1947: 4, 21, 284, 336), which is based on the difference between individual and general. For practical purposes, sign forms must be reproducible since we want to use them on more than one occasion. The individual sign form is called a token. Thus in the two sentences “Whatever happens, Berlin will remain Berlin” and “Berlin is situated in Germany” we have the same word Berlin, but appearing in three different tokens; and in giving the explanation, a fourth token of the word has been produced. This can be described by stating that the four sign forms are tokens of the same sign type. Thus the common formulation “the same sign occurs in different places” amounts to saying “(sign [form]) tokens of the same (sign [form]) type occur in different places”. The independence of the type/token-distinction from the form/substance/matter-distinction is demonstrated by the fact that one can also distinguish tokens from types in sign substance and in sign matter.

3. Relations between signs

The statement that in a complex sign form like “Berlin will remain Berlin” two component sign forms are tokens of the same type is a statement about a relation between these sign forms and thus falls into syntactics2, the study of the relations between signs.

Traditionally, there are said to be two kinds of relations between signs (cf. Morris 1938: 6f and 18ff = 1971: 21f and 32ff):

a. relations between signs occurring in a given complex sign,
b. relations between signs in a sign system (i.e., code; cf. Art. 16 and 17).

Relations of the first kind are usually exemplified by syntagmatic relations, relations of the second kind by paradigmatic relations (cf. Kruszewski 1884, Saussure 1916, and Hjelmslev 1943). The difference can best
be demonstrated with respect to the process of sign production. In producing a complex sign, the sender will make successive selections from the inventory of a sign system mastered by him and combine, according to certain rules, the chosen elements into an appropriate structure. Let us take a natural language like English as the sign system and an utterance of the sentence “The child sleeps” as an example. Then, “if child is the topic of the message the speaker selects one among the extant more or less similar nouns like child, kid, youngster, tot, all of them equivalent in a certain respect, and then, to comment on this topic, he may select one of the semantically cognate verbs sleeps, dozes, nods, naps. Both chosen words combine in the speech chain” (Jakobson 1960: 358). The set of elements that provides the basis of selection in each step of the sign production is called “a paradigm”, and the result of the combination of the elements selected is called “a syntagm”.

Paradigms need not consist of semantically equivalent signs, as in Jakobson’s example, but can consist of signs having the same distribution in syntax like sleeps, lies, stands, or of signs belonging to the same lexeme like sleep, sleeping, sleeps, slept, or of signs containing the same root like sleep, sleeper, sleepy, or of signs having the same phonemes in certain positions like sleep, sweep, sleep, or even of signs having the same subphonemic properties like German ich, nicht, Licht, where /f/ is pronounced differently from ach, Nacht, lacht. Semantic, syntactic, inflectional, derivational, phonemic, and phonetic paradigms all have the same structure: Each of them is a class of elements equivalent to one another in a certain respect. Paradigms that fulfill additional conditions such as that of mutual substitutability of all their elements in specified types of contexts salva congruitate, i.e., with well-formedness being preserved in the interchange, are called “syntactic categories” (cf. Bar-Hillel 1964; cf. also § 5.1. and 5.2. as well as Art. 4).

While paradigms are constituted by relations of (partial) equivalence, syntags are constituted by relations of contiguity. Contiguity can be conceived of as proximity in space or time, as a restriction determining the distribution of one constituent with respect to another, or as a dependency relation like agreement in number between a and child or agreement in person and number between child and sleeps in the above example. Since syntags are complex signs produced by some interpreter, syntagmatic relations are part of the surface structure of those signs. A syntagmatic relation is to be distinguished from a deep-structure relation and from a syntactic rule determining either of them (cf. Posner 1982: 129–159). Although originally defined as a term for linear configurations of signs in texts (“serial relations”, cf. Saussure 1916: 171 and Carnap 1934: 1; “precedence relations”, cf. § 5.1.4.), the term “syntagmatic relation” is also applied to signs combined in more than one dimension (cf. § 5.4.), as occurring, e.g., in vestimentary codes (cf. Barthes 1967: ch. 12), film (cf. Metz 1974: ch. 8 and 9), music (cf. Tarasti 1979: ch. 6.1), theater (cf. Fischer-Lichte 1983: ch. 8), and pictorial art (cf. Groupe μ 1992: 218 ff).

Syntagmatic and paradigmatic relations are conceptually distinct, but they can occur together, as in paradigms of syntags: The child sleeps, The youngster dozes. The kid nods, The tot naps constitute a paradigm of elements equivalent with respect to the syntagmatic relations holding within each of them; another case in point is the inflectional paradigm exemplified above.

Paradigms of sign forms need not be given in advance within a generally valid code, they can also be constructed in the perception of an individual complex sign such as a poem, a picture, a building, or a piece of music. They then constitute what is called the “aesthetic code” of the complex sign in question (cf. Posner 1982: 122 f).

In music, the listener’s search for œuvre-specific paradigms in melody, harmony and rhythm is an essential part of the interpretation process. Ruwet (1966) has attempted an operationalization of this process which is summarized here in the procedural formulation of Monelle (1992: 83 f): “1. One identifies […] the longest passages that are repeated fully, whether at once or after another passage, giving formulae like A + B + X + B + Y, where A and B are recurrent passages, X and Y non-recurrent passages. 2. Non-recurrent passages are considered units on the same level as recurrent passages in the respect of length. As repetitions of temporal proportion they may be written as primary units; thus A + X + A + Y becomes A + B + A + C. The resulting segmentation may now be checked by looking at pauses […] 3. If the above operations have not yielded a satisfactory analysis, the following misfunctions may be suspected: (i) Non-recurrent
passages are much shorter than recurrent ones and therefore cannot be considered units on the same level. (ii) Non-recurrent passages are longer than recurrent ones. Here the non-recurrent passage may have to be subdivided, either yielding several units of level 1 – thus A + A + X becomes A + A + B + C – or shifting the analysis on to level 2 where the recurrent passages are themselves subdivided into shorter figures which may be found to recur in the non-recurrent passages. Thus, for example, A + A + X may become \((a + b) + (a + b) + (x + b)\). If neither of these procedures is possible, it may be necessary to consider X as an unanalyzed unit on ‘level 0’. 4. Passages which seem initially not the same may be transformations of each other (rhythmic or melodic variants) according to certain principles of transformation. (i) If pitch and rhythm are separated, we may find similar contours with different rhythms or similar rhythms with different contours. (ii) There may be permutations, additions and suppressions of certain elements. (iii) In discerning the equivalence of certain passages it may be necessary to shift from a higher to a lower level or from a lower to a higher, which Ruwet calls ‘shunting’. For example, while units on level 2 \(- a + b + a + c\) may seem to constitute the realities of analysis, it may be necessary to move up to level 1 for the purposes of subsequent work, and to determine that \(a + b = A\) and \(a + c = A'\). (iv) In some cases units of level 1 may seem to group themselves into even longer units. Must we therefore evoke a ‘level 0’? Taking two typical formulae: \(A + X^* + A + Y, X + A^* + Y + A\), we find that each may become \(A + B\) or perhaps \(A + A'\) on level 0, on the fulfillment of one of the following conditions: (a) there is some special rhythmic marking of the point shown by an asterisk, either a pause or the prolongation of a note. (b) \(Y\) is a transformation of \(X^*\). Ruwet (1966) exemplifies this procedure with the 14th century German flagellant song *Maria muoter reine nait* and isolates the paradigmatically related passages by writing them vertically beneath each other in the score (cf. Fig. 2.1).


A complex sign may be the basis for setting up several alternative paradigms, and the choice between them can be a source of aesthetic delight for the interpreter, as in Brahms’ *Intermezzo opus 119 nr. 3* (cf. Fig. 2.2) where paradigms of recurrent rhythmical and melodic sequences (see the analysis beneath the score) compete with paradigms of metric and harmonic arrangement (see the analysis above the score).

Each of these alternative structurings can be emphasized or de-emphasized by a musician through the use of subtle diacritical signs. Monelle (1992: 98) provides the historical context: “The […] analysis [given above the score] conforms best to the harmonic arrangement of the tune and seems initially the most natural. Indeed, Julius Katchen, in his recording of the piece, seems to follow it. But another pianist, Walter Klein, stresses the melodic repetitions [of the analysis given beneath the score] […] and even more striking,
in the film Rendez-vous à Bray of André Delvaux this piece serves as *leitmotiv* and becomes the object of variations in different styles. It is the melodic unit [...] which serves as theme of these variations."

In mathematical music theory (cf. Mazzola 1985, 1990 and 1993 as well as Noll 1995), a paradigmatic relation is defined as a morphism between local compositions. Local compositions are regarded as pairs such as \((K, M)\) and \((L, N)\), where \(K\) and \(L\) are (finite) subsets of musical parameter spaces \(M\) and \(N\). (For the concept of parameter space cf., e.g., Nattiez 1987: ch. 9–12.) A morphism between two local compositions \(f: (K, M) \Rightarrow (L, N)\) is a mapping \(f: K \Rightarrow L\) which is induced by a structure-preserving mapping \(F: M \Rightarrow N\) between the two parameter spaces involved. A simple example is Mazzola’s (1990: 93ff) explication of the relationship between the major and minor tone scales in Western music: If these scales are studied as local compositions in a fifth-third-tone net, then it can be proven that there are exactly two isomorphisms which map them onto each other. In Fig. 2.3 the C-major scale and the C-minor scale are presented on the left and right, respectively. The isomorphism indicated by the three two-sided arrows is an axial mirroring along the central horizontal line connecting \(F\) and \(D\) which maps \(F, C, G\) and \(D\) into themselves and \(A\) into \(Ab\), \(E\) into \(Eb\), and \(B\) into \(Bb\). The isomorphism indicated by the curved arrows is a central mirroring around the center of the line connecting \(C\) and \(G\) which maps \(F\) into \(D\), \(C\) into \(G\), \(A\) into \(Bb\), \(E\) into \(Eb\), \(B\) into \(Ab\), and vice versa. This formal analysis is confirmed by

the fact that Hindemith (1940) discussed exactly two possible interpretations of the opposition between major and minor scales. The axial mirroring corresponds to what Hindemith called “the dimming” of \(E\) to \(Eb\) (as well as of \(A\) to \(Ab\) and \(B\) to \(Bb\)), while the central mirroring corresponds to what he called “the dualism” between the two scales. In mathematical music theory, isomorphisms of the kind used in our examples are applied to set up a classification of all possible local compositions into paradigms (cf. Frietteringer 1993).

In musicology, as in other semiotic disciplines, the study of paradigms is called “paradigmatics” and the study of syntagms is called “syntagmatics”. Paradigmatics and syntagmatics are both part of syntactics. While paradigmatics requires the comparison of given sign forms with virtual ones, syntagmatics analyzes the relations of sign forms within actually produced sign complexes. It will therefore be treated in more detail in the context of syntactics (cf. § 4. and 5.).

Fig. 2.2: A passage from Brahms’ Intermezzo opus 119 nr. 3, analyzed according to competing paradigms (after Nattiez 1975: 326 and Monelle 1992: 98).

Fig. 2.3: The two isomorphisms connecting the C-major and C-minor scales in Western music.
2. Syntactics

Syntactics as an intersection of syntactics\textsubscript{1}, syntactics\textsubscript{2} and syntactics\textsubscript{3} must not be confused with syntagmatics nor with syntax as defined in linguistics. As is obvious from the verbal examples discussed above, paradigmatic and syntagmatic relations can be found on all levels of language. This fact has been exploited by structuralist linguists to use these relations in the definition of the levels of language and of the disciplines studying them (cf. Bloomfield 1933, Harris 1947, as well as Hjelmslev 1943): phonetics studying the physical properties of linguistic sound matter; phonemics studying the relations between phonemes, i.e., the smallest sound forms used to distinguish the signs of a language; morphology studying the relations between morphemes, i.e., the smallest combinations of phonemes representing meaning to the language user, and their combinations into words (derivational morphology) and word forms (inflectional morphology); syntax studying the relations between phrases, i.e., combinations of word forms within and into sentences; lexicology studying the relations between paradigms of word forms having the same meaning. Of these disciplines, phonetics does not deal with formal aspects of signs and is therefore excluded from syntactics\textsubscript{1}; phonemics deals with formal aspects of signs, but not with combinations of signs and is therefore excluded from syntactics\textsubscript{2}; inflectional and derivational morphology and syntax deal with formal aspects of signs, their relations, and their combinations and are part of syntactics proper; lexicology deals with formal aspects of signs but not with their combinations and is therefore excluded from syntactics\textsubscript{1}.

As it turns out, syntactics proper includes only morphology and syntax from the linguistic disciplines, and it is no accident that this is exactly what linguists have traditionally called “grammar”. Thus it is justified to regard syntactics as a semiotic generalization of grammar.

In many contexts, the Carnapian identification of syntactics with syntax (cf. Carnap 1934 and 1939) is highly misleading. Only in sign systems which do not require a distinction between morphology and syntax is it unproblematic to equate syntactics with syntax. This is the case in sign systems such as numerals and in most of the formal languages constructed in logic so far.

In summary, it should be noted that syntactics proper contains both syntagmatics and syntax as subdisciplines, but syntagmatics overlaps with syntax, since syntax studies not only syntagmatic but also paradigmatic relations between phrases, and syntagmatics studies not only phrases but also morphemes and phonemes.

4. Complex signs

4.1. Analysis and synthesis

In his program for a “logical syntax of language”, Carnap (1934: 1) had envisioned “a formal theory of the forms of a language” to be formulated in a syntactic metalanguage. There have been various attempts to work out such a formal theory (cf. Art. 28 and Art. 30 § 1.7.). Most of them have been guided by the idea of a calculus, i.e., an axiomatic system that has the properties of an algorithm for specifying exactly the set of all signs belonging to the object-code under investigation (concerning the concept of algorithm cf. § 5.3. below).

The specification of a set of more or less complex objects can be given either by starting with the complex objects and introducing rules for their analysis into components, components of components, etc. until elementary objects are reached. Or one can proceed from elementary objects and introduce rules for their use in the synthesis of more and more complex signs. The two approaches are of different value for different kinds of sign systems. For many sign systems that are in use in human or animal societies or within organisms or machines, it is by no means clear what the basic elements are. The most controversial examples include dance, gestures, pictures, films, and architecture. However, there are also sign systems where it is clear what the elementary signs are and hard to decide for a given complex sign whether it is part of the sign system in question or not and what is its structure. Such cases occur in some of the richer artificial languages of logic and mathematics. For these reasons it has become customary to use the analytic approach in the study of natural sign systems that have evolved over time, and the synthetic approach in the study of artificial sign systems (cf. the distinction between text-oriented and grammar-oriented cultures in Lottman 1969 and 1971, which is discussed in Eco 1976: 137 ff; cf. also Pape 1980).

Linguists such as Bloomfield (1926 and 1933), Harris (1947), Wells (1947), Pike
(1954) and Hockett (1958), as well as Hjelmslev (1943), Prieto (1966), and Ruwet (1966; see the application to music in Fig. 2.1 above) have developed procedures for the step-by-step analysis of texts into components, components of components, etc. The formal theory for this approach has been discussed by authors such as Hjelmslev (1943), Marcus (1967), and Harris (1970). According to Hjelmslev, syntactic theory has to provide a general calculus containing rules of partition for complex signs in all sign systems. The application of such a calculus to a given complex sign involves a finite set of partition operations, the last of which will yield basic elements of the sign system in question. The syntactic structure of that complex sign is described by describing its analysis. The basic elements of the whole sign system are obtained by analyzing complex signs belonging to the system until no new basic elements are generated (cf. Hjelmslev 1943: 27–31 = 1963: 28–33).

In judging the value of this approach one must distinguish between the continental European tradition and the American tradition. The first relies on Hjelmslev’s commutation method (cf. Art. 4 § 4.), which is applicable to sign systems of all kinds but does not abstract from the meaning of the signs analyzed, in the way required by Carnap and Morris. The second abstracts from the meaning of the signs analyzed but relies on the method of parsing, which is applicable to languages only.

Philosophers and logicians such as Leibniz (cf. Knobloch 1973), Boole (1854), Frege (1879 and 1893–1903), Schröder (1890), Peano (1894), Peirce (1934), Whitehead and Russell (1910–1913), Carnap (1934, 1939, and 1954), and Curry (1963) were the first to develop step-by-step procedures for the construction of more and more complex signs out of basic elements. The formal theory for this approach has been given by authors such as Thue (1914), Post (1936), Turing (1937 and 1950), Hermes (1938 and 1961), Markov (1947), Lorenzen (1955), Davis (1958), Trakhtenbrot (1960), Chomsky and Miller (1963), Bar-Hillel (1964), and Lindemayer (cf. Lindemayer and Rozenberg 1976 and 1979 and Rozenberg and Salomaa 1974). According to Chomsky, syntactic theory has to specify the general form of a calculus that generates all the expressions, i.e., the simple and complex signs, of a given language, starting from a finite set of basic elements and using a finite set of rules of various types. The application of such a calculus to an initial string involves a finite set of production operations, the last of which yields an expression of the language in question. The syntactic structure of that expression is described by describing its production. The set of expressions of the language is obtained by applying the rules of the calculus to all its basic elements (cf. Chomsky 1957: 18–91).

Compared with the analytic approach, the calculi developed for the synthetic approach have reached a much higher sophistication. In addition, logicians like Carnap (1934: 8), Quine (1960), Montague (1970a and 1970b), and Cresswell (1973) and linguists like Chomsky (1965) and Shaumyan (1970) have shown that it is possible to apply the synthetic approach in the analysis of natural languages also by using the so-called “indirect method” (cf. Schnelle 1973). The point of this method is to introduce an artificial sign system into the metalanguage, which can be kept under tight control by means of the stipulative definitions with which it was constructed, to compare this sign system with the object-code, and to incorporate more and more features of the object-code into it so that in the end the set of signs belonging to the object-code becomes completely reconstructed in the metalanguage.

This strategy has worked well in the analysis of natural languages. If it is to be applied to the study of non-linguistic sign systems, an additional obstacle has to be overcome. In language, the relations between the components of a complex sign are generally thought to be based on one single serial relation, e.g., the relation “following in time” as in speech or “immediately to the right” as in European writing systems. Complex signs governed by serial relations can be produced through application of an associative non-commutative binary operation called “concatenation”. As noted in § 2. and 3., however, there are sign systems that have either additional serial relations or equivalence relations or relations of more complex types governing their complex signs.

If one wants to describe the syntactic structure of complex signs in sign systems using operations other than concatenation, one can again choose between a direct and an indirect strategy. The first strategy consists in defining appropriate operations of combination and describing the complex signs directly on this basis (cf. § 5.4. below). This strategy

In the second strategy, a system of notation is devised to represent the relevant features of the complex signs in question in a way that makes them more amenable to a linguistics-type syntactic analysis. Notational systems tend to reduce multidimensional sign configurations to two-dimensional ones (scores) or one-dimensional ones (strings). Examples are musical notations and the phonetic transcriptions of natural languages. Theoretically it is always possible to reduce a given \( n \)-dimensional sign complex to a complex with \( n-1 \) dimensions as long as the relations among its constituents are serial or equivalence relations (cf. Greenberg 1957: 95f, Curry 1963: 50–69, Goodman 1968: 127–224, Fu 1976; see also § 5.3. below).

To sum up, the best developed branch of syntactics is able to describe the syntactic structure of sign systems for which two conditions are fulfilled:

a. A set of discrete basic elements is given from which all well-formed signs of the system can be constructed by combinatory operations (for syntactic structures beyond discreteness cf. Marcus 1992: 1356f).

b. All combinatory operations can be reduced to or defined on the basis of one single binary operation, i.e., concatenation; all complex signs of the system therefore are, or are reducible to, strings (for syntactic structures beyond concatenation cf. § 5.4.).

Sign systems with these properties are called “string codes”. They include, among others, natural languages, writing systems, vestimental codes, culinary codes, and traffic signs.

As a technical device for the syntactic description of string codes, formal grammars have been developed which may be characterized as string production grammars. They are a special type of the so-called “generative grammars” (cf. Chomsky 1957 and 1965, Bar-Hillel 1964, Marcus 1968, and Herrmanns 1977). The rest of § 4. will give formal definitions for the concepts of a string (§ 4.2.) and a string code (§ 4.3.), introduce some basic types of string production grammars, (§ 4.4.–4.7.), compare them with automata (§ 4.8.), explicate the concepts of the syntactic structure and the structural description of a string (§ 4.9.), and discuss transformations between string structures (§ 4.10.).

4.2. The concept of a string

Let \( V \) be a finite set. Then a string of length \( n \) \((n \geq 1)\) over \( V \) is a finite sequence \( a_1a_2 \ldots a_n \) of elements of \( V \) \((a_i \in V \text{ for } i = 1, 2, \ldots, n)\). A null string, \( \Lambda \), can be defined as a string with no elements in it. We will use lower case Latin letters for elements of \( V \) and lower case Greek letters for strings.

Let \( \Sigma(V) \) be the set of all non-null strings over \( V \). There is a binary operation \( ^\ast \) in \( \Sigma(V) \) such that if \( a \) and \( \beta \) are members of \( \Sigma(V) \) and if \( a = a_1 \ldots a_n \) and \( \beta = b_1 \ldots b_m \), then \( a^\ast \beta = a_1 \ldots a_nb_1 \ldots b_m \) is also a member of \( \Sigma(V) \).

This operation is non-commutative, i.e., for \( a + \beta \) we generally have \( a^\ast \beta \neq \beta ^\ast a \), and associative, i.e., \( (a^\ast \beta)^\ast \gamma = a^\ast (\beta^\ast \gamma) = a^\ast \beta^\ast \gamma \).

Such an operation as well as each result of its application is called “concatenation”. In the following, concatenation of two strings will be represented by simple juxtaposition of these strings. Because of non-commutativity each concatenation result has a unique sequential order. Because of associativity the sequential order in which concatenation operations are performed has no effect on the concatenation results.

4.3. String codes

An algebraic system \( \langle A, \circ \rangle \) composed of a set \( A \) and a binary operation \( \circ \) is called “a semigroup” if \( \circ \) is associative in \( A \). Clearly \( \Sigma(V) , ^\ast \) is a semigroup. Since \( ^\ast \) is represented by juxtaposition, we will use \( \Sigma(V) \) to refer to it.

If \( \langle A, \circ \rangle \) is a semigroup which contains an element \( e \) such that \( a \circ e = e \circ a = a \) for all \( a \in A \), then \( \langle A, \circ \rangle \) is called “a monoid” and \( e \) is called “an identity of \( A \)”. If a semigroup contains an identity this identity is unique.

Let \( \Sigma(V) = \Sigma(V) \cup \{ \Lambda \} \); then clearly, \( \Sigma(V) \) is a monoid with \( \Lambda \) as its identity.

To exemplify an abstract system which fulfills the required conditions and is not a sign system, let us consider the set \( \mathbb{N} \) of natural numbers \( 1, 2, 3, \ldots \) under the operation of addition: The elements \( n_1, n_2, \) and \( n_3 \) in \( \mathbb{N} \) have the property \((n_1 + n_2) + n_3 = n_1 + (n_2 + n_3) \), i.e., addition is associative and \( \langle \mathbb{N}, + \rangle \) is a semigroup. However, \( \langle \mathbb{N}, \times \rangle \) is no monoid, since it contains no identity. Yet we
may add 0 as an identity to \(\mathbb{N}\) and thus obtain \(\mathbb{N}_0 = \{0, 1, 2, \ldots\}\). Then \(\langle \mathbb{N}_0, + \rangle\) is a monoid with identity 0, since we still have \((n_1 + n_2) + n_3 = n_1 + (n_2 + n_3)\) for all \(n_1, n_2, n_3 \in \mathbb{N}_0\) and moreover \(0 + n = n + 0 = n\) for any \(n \in \mathbb{N}_0\).

A semigroup \(\Sigma(V)\) or monoid \(\Sigma'(V)\) constructed from a finite vocabulary \(V\) is called “the free semigroup over \(V\)” or “the free monoid over \(V\)”, respectively. How is a structure like a free monoid \(\Sigma(V)\) related to a natural language? Let \(V\) be thought of as the set of phonemes (or of morphemes or of words) in a given language. Then \(\Sigma(V)\) would contain, among other strings, all the sentences in the language under discussion. Thus that language can be defined as a subset of \(\Sigma(V)\).

This idea applies to all string codes.

A string code is a set of signs that can be characterized as a subset of a free monoid.

If one wants to characterize a given string code \(L\) syntactically, the task is to set up a finite vocabulary \(V\), to define the free monoid \(\Sigma(V)\) over it, and to distinguish the strings of \(L\) from the rest of \(\Sigma(V)\). In language, this task corresponds to the language-user’s capacity to distinguish well-formed sentences from strings not belonging to his language. Analogous considerations apply to all other string codes.

4.4. Production grammars

A production grammar is a set of rules on the basis of which the strings belonging to a certain subset of a free monoid can be constructed. If the grammar produces all and only the strings of a given string code \(L\), then the grammar is called “observationally adequate for \(L\)” (cf. Chomsky 1964: 30ff and Wasow 1985 for other levels of adequacy).

As an example for a string code \(L\) we take the standard system of signs used for the representation of natural numbers, the Hindu-Arabic numerals \(L_{10}\). Let \(V_M\) be the set of the ten Hindu-Arabic digits \(0, 1, 2, \ldots, 9\). Evidently \(L_{10}\) is infinitely large; however, it is only a subset of \(\Sigma(V_M)\) since while it includes \(10, 100, 1000, \ldots, 20, 200, 2000, \ldots\), etc., it does not include \(0, 00, 000, \ldots, 01, 001, 0001, \ldots, 02, 002, 0002, \ldots\), etc. To be observationally adequate for \(L_{10}\), a grammar must construct the infinitely large set of Hindu-Arabic numerals on the basis of \(V_M\) and exclude strings like the ones mentioned.

This is accomplished through a set of stipulations such as the following:

(i) \(1, 2, \ldots, 9\) belong to \(L_{10}\).

(ii) If \(a\) belongs to \(L_{10}\), then \(a0, a1, \ldots, a9\) belong to \(L_{10}\).

(iii) Elements of \(L_{10}\) can be constructed in no other way than by employing (i) and (ii).

The stipulations (i) through (iii) are said to be the rules \(R_{10}\) of a grammar \(\theta_{10}^{10}\) for the string code \(L_{10}\). In \(R_{10}\), (iii) amounts to saying that \(L_{10}\) is the smallest string code over \(V_M\) which can be formed so as to satisfy rules (i) and (ii). Using (i) and (ii) one can construct all and only the strings that belong to \(L_{10}\), i.e., that are Hindu-Arabic numerals.

Rules can be given in various forms. An alternative formulation for (i) through (iii) would be to introduce \(Z\) as a starting symbol and stipulate:

\[
(i') \quad Z \rightarrow \begin{cases} 1 \\ 2 \\ 9 \\ 0 \end{cases} \quad (ii') \quad Z \rightarrow Z \begin{cases} 0 \\ 1 \\ 9 \end{cases}
\]

In applying these rules one would first write down the starting symbol \(Z\) and then replace it either by any one digit except 0, according to \((i')\), or by \(Z\) followed by any one digit, according to \((ii')\). Rules such as \((i')\) and \((ii')\) are called “rewrite rules” or “production rules”.

In order to construct the complex sign representing the number of the year 1983, the rules \((i')\) and \((ii')\) would have to be applied in the following way:

\[
(1') \quad Z \quad (2') \quad Z3 \quad \text{by rewriting } Z \text{ in } (1') \text{ as } Z3 \quad \text{according to } (ii')
\]

\[
(3') \quad Z83 \quad \text{by rewriting } Z \text{ in } (2') \text{ as } Z8 \quad \text{according to } (ii')
\]

\[
(4') \quad Z983 \quad \text{by rewriting } Z \text{ in } (3') \text{ as } Z9 \quad \text{according to } (ii')
\]

\[
(5') \quad 1983 \quad \text{by rewriting } Z \text{ in } (4') \text{ as } 1 \quad \text{according to } (i')
\]

The ordered set of formulas \((1')\) through \((5')\) is called “a derivation of 1983”. Strings like 1983 that do not allow the further application of a rule are called “terminal strings”.

A further alternative in the formulation of rewrite rules for the Hindu-Arabic numerals is obtained when we introduce \(M\) and \(N\) as special names for the classes of signs occurring in braces in \((i')\) and \((ii')\):

\[
(i'') \quad Z \rightarrow N, \quad N \rightarrow 1, \quad N \rightarrow 2, \ldots, \quad N \rightarrow 9,
\]

\[
(ii'') \quad Z \rightarrow NM, \quad M \rightarrow 0, \quad M \rightarrow 1, \ldots, \quad M \rightarrow 9.
\]

Here \(M\) and \(N\) are called “non-terminal symbols” since in derivations made on the basis
of (i') and (ii') they occur only in non-terminal strings. Thus a grammar consisting of rules (i') and (ii') contains not only a terminal vocabulary $V_T = V_H$, but also a non-terminal vocabulary $V_{NT}$ with $Z, M$, and $N$ as its members.

In order to make precise the terms we have used in describing this example, we introduce the following definitions: A production grammar is a 4-tuple $G = \langle V_{NT}, V_T, S, R \rangle$ where $V_{NT}$ and $V_T$ are two disjoint non-void finite sets, the non-terminal and terminal vocabularies of $G$; $S$ is an arbitrary fixed symbol out of $V_{NT}$, the so-called “axiom” or “starting symbol of $G$”; and $R$ is a finite set of rules of the form $a \rightarrow \beta$ (“rewrite $a$ as $\beta$”) such that $a$ and $\beta$ are elements of $\Sigma'(V_{NT} \cup V_T)$ and $a$ contains at least one element from $V_{NT}$.

Now if there is a string $\gamma = \gamma_1\alpha_2 \tau_2$, from $\Sigma'(V_{NT} \cup V_T)$, then, given the rewrite rule $a \rightarrow \beta$, the string $\gamma' = \gamma_1\beta_2 \tau_2$ is said to be “directly derivable from $\gamma$”. The relation of direct derivability in grammar $G$ is commonly designated by the double arrow $\Rightarrow_G$; thus we have $\gamma \Rightarrow_G \gamma'$. (The subscript $G$ is omitted if it is clear from the context which is the grammar at issue.) Direct derivability has derivability $\Rightarrow^*$ as its ancestral. This means that $\delta'$ is derivable from $\delta$ if and only if there is a sequence $e_0, e_1, \ldots, e_m (m \geq 0)$ such that (1) $e_0 = \delta$, (2) $e_m = \delta'$, and (3) for each $k (0 \leq k \leq m-1)$, $e_k \Rightarrow e_{k+1}$. In our sample grammar with the rewrite rules (i') and (i'') for instance, it holds true that, e.g., $ZM3 \Rightarrow NMM3$ and $NMM3 \Rightarrow NM3$ and $NM3 \Rightarrow N983$ and $N983 \Rightarrow 1983$; thus we have $ZM3 \Rightarrow^* 1983$. (Note that, under the definition given, $S \Rightarrow^* S$ holds in every grammar.)

The string code $L(G)$ generated by $G$ is the set of all strings over the vocabulary $V_T$ which is derivable from the axiom $S$; i.e., $L(G) = \{a \mid a \in \Sigma'(V_T)$ and $S \Rightarrow^* a\}$. String codes are also sometimes called “languages”, even if their vocabulary does not consist of phonemes, morphemes, or words.

Let us also introduce rewrite rules $a \rightarrow \beta$ of a special form here. If $a$ is a single non-terminal symbol and $\beta$ a terminal one, then $a \rightarrow \beta$ is called “a lexical rule” or “lexical entry for $\beta$” because it provides a classification of a word $\beta$ (e.g., “house”) as belonging to a syntactic category $a$ (e.g., a noun) and such information is typically given in dictionaries. In what follows, lexical rules are sometimes formulated by means of the inverted arrow (as in $a \leftarrow \beta$, which stands for, e.g., $9 \rightarrow N$). A rule such as $\textit{house} \rightarrow \textit{noun}$ amounts to the statement “the category Noun includes the item house”.

Let us call the grammar with rules (i) through (iii) “$G^0_H$”, the production grammar with rules (i') and (ii') “$G^1_H$”, and the production grammar with rules (i'') and (ii‘”) “$G^2_H$”. It is easily seen that $G^0_H$, $G^1_H$, and $G^2_H$ have the same output, i.e., produce the same set of terminal strings, i.e., $L_H$. In general we say that two production grammars $G_1$ and $G_2$ are equivalent if they have the same output (for other levels of equivalence cf. Chomsky 1963).

There are three main classes of production grammars which are interesting syntactically. These are, in increasing amount of generality, regular grammars, context-free grammars, and context-sensitive grammars.

4.5. Regular grammars

For obvious reasons it is necessary to distinguish two kinds of regular grammars, left regular grammars and right regular grammars. But it can easily be shown that for any given right regular grammar there exists a left regular grammar with the same output. A production grammar $\langle V_{NT}, V_T, S, R \rangle$ is right (left) regular if its rules take one of the following forms:

$A \rightarrow aB, \quad C \rightarrow c$

$A \rightarrow aBa, \quad C \rightarrow c$

where $A, B, C$ belong to $V_{NT}$ and $a, c$ to $V_T$.

Thus the grammar $G^0_H$ is a left regular grammar, while the following is right regular:

$G^0_D = \langle \{Z, P, Q\}, \{0, 1, 2, \ldots, 9\}, Z, R^0_D \rangle$

where $R^0_D$ contains the rules:

$\begin{align*}
(i^*) \quad &Z \rightarrow \begin{cases} l \rightarrow 0 & (P), \\
9 \rightarrow 1 \end{cases} \\
(ii^*) \quad &P \rightarrow \begin{cases} l \rightarrow 0 & (P), \\
9 \rightarrow 1 \end{cases}
\end{align*}$

$\begin{align*}
(iii^*) \quad &Z \rightarrow .Q, \\
(iv^*) \quad &P \rightarrow .Q,
\end{align*}$

$\begin{align*}
(v^*) \quad &Q \rightarrow \begin{cases} l \rightarrow 0 & (Q), \\
9 \rightarrow 1 \end{cases} \\
(vi^*) \quad &Q \rightarrow \begin{cases} l \rightarrow 0 & (Q), \\
9 \rightarrow 1 \end{cases}
\end{align*}$

and (i*) is short for the two rules
4.6. Context-free grammars

Context-free grammars are a less restrictive kind of production grammar. A production grammar \( \langle V_{NT}, T, S, R \rangle \) is context-free if each of its rules takes the form

\[
A \rightarrow \alpha
\]

where \( A \) belongs to \( V_{NT} \) and \( \alpha \) to \( \Sigma(V_{NT} \cup T) \). A context-free grammar is exemplified by \( \Theta^1_L \). However, \( L_{H} \) the output of \( \Theta^1_L \) can also be produced by \( \Theta^1_H \), and since \( \Theta^1_H \) is regular, \( L_{H} \) is not only context-free but regular as well.

An example of a string code that is context-free but not regular is the set \( L_{0} \) of English sentences connected by the logical connectives \( \ldots \text{and} \ldots, \ldots \text{or} \ldots, \) it is false that \( \ldots \), and if \( \ldots \) then \( \ldots \). If \( \alpha \) and \( \beta \) are sentences of English, then \( \alpha \) and \( \beta \), \( \alpha \) or \( \beta \), it is false that \( \alpha \), and if \( \alpha \), then \( \beta \) are also sentences of English. These considerations can be embodied in a context-free grammar \( \Theta^0_p \) as follows:

Let the non-terminal vocabulary \( \Theta^0_p \) as follows:

Let the terminal vocabulary be composed of the symbols \( S, E, N, I, T \) and let the terminal vocabulary be composed of \( \text{and}, \text{or}, \text{it}, \text{is}, \text{false}, \text{that}, \text{if}, \text{then}, \text{and} \sigma \), where \( \sigma \) would have to be expanded into an English sentence in a larger grammar containing the present fragment). Let the rules be

\[
\begin{align*}
(i^*) & \quad S \rightarrow SES, \\
(ii^*) & \quad S \rightarrow ISTS, \\
(iii^*) & \quad S \rightarrow NS, \\
(iv^*) & \quad S \rightarrow \sigma, \\
(v^*) & \quad E \rightarrow \text{and}, \\
(vi^*) & \quad E \rightarrow \text{or}, \\
(vii^*) & \quad N \rightarrow \text{it is false that}, \\
(viii^*) & \quad I \rightarrow \text{if}, \\
(ix^*) & \quad T \rightarrow \text{then}.
\end{align*}
\]

\( \Theta^0_p \) will produce strings like if \( \sigma \) then \( \sigma \), which become normal English sentences when every occurrence of \( \sigma \) is replaced by an English sentence. However, \( \Theta^0_p \) will not produce such strings for all English sentences containing the logical connectives specified. Strings such as \( \sigma \sigma \) and \( \sigma \), which have the syntactic structure of listing expressions like

\[
(1) \ a \ b
\]

do not belong to the output of \( \Theta^0_p \).

Another problem is the fact that, when interpreted in the usual way, some of the strings produced by \( \Theta^0_p \) will be ambiguous (cf. the music example of Fig. 2.2 in § 3. above). For example, it will be left open whether \( \sigma \) and \( \sigma \) or \( \sigma \) should be read as \( \sigma \) and \( \sigma \) or as \( \sigma \) and \( \sigma \). Depending on what sentences are substituted for \( \sigma \), these two readings can have rather different consequences. The two readings are mirrored in two different derivational histories for \( \sigma \) and \( \sigma \) or \( \sigma \):

\[
\begin{align*}
(1.1) & \quad S \\
(1.2) & \quad S E S \quad \text{by (i^*)} \\
(1.3) & \quad S E S E S \quad \text{by (i^*)} \\
(1.4) & \quad \sigma E \quad \text{by (iv^*)} \\
(1.5) & \quad \sigma and \quad \sigma E S \quad \text{by (iv^*)} \\
(1.6) & \quad \sigma and \quad \sigma E S \quad \text{by (iv^*)} \\
(1.7) & \quad \sigma and \quad \sigma or \quad S \quad \text{by (v^*)} \\
(1.8) & \quad \sigma and \quad \sigma or \quad \sigma \quad \text{by (iv^*)} \\
(2.1) & \quad S \\
(2.2) & \quad S E S \quad \text{by (i^*)} \\
(2.3) & \quad S E S E S \quad \text{by (i^*)} \\
(2.4) & \quad \sigma E \quad \text{by (iv^*)} \\
(2.5) & \quad \sigma and \quad \sigma E S \quad \text{by (iv^*)} \\
(2.6) & \quad \sigma and \quad \sigma E S \quad \text{by (iv^*)} \\
(2.7) & \quad \sigma and \quad \sigma or \quad S \quad \text{by (v^*)} \\
(2.8) & \quad \sigma and \quad \sigma or \quad \sigma \quad \text{by (iv^*)}
\end{align*}
\]

The two derivations (1.1) through (1.8) and (2.1) through (2.8) use the same rules, but apply them differently: In \( S E S \) of (1.2) the first occurrence of \( S \) is rewritten as \( S E S \) according to rule (i^*). In \( S E S \) of (2.2) the second occurrence of \( S \) is rewritten as \( S E S \) according to rule (i^*). However, this difference cannot be reflected by the formal properties of the derivations as defined so far.

Logicians solve the problem of ambiguity by introducing brackets into the terminal vocabulary and restating the first three rules above as follows:

\[
\begin{align*}
S \rightarrow & \quad [SES], \\
S \rightarrow & \quad [ISTS], \\
S \rightarrow & \quad [NS].
\end{align*}
\]

The resulting grammar \( \Theta^1_H \) will al-
low the following derivations: $S$, $[SES]$, $[[SES][ES]]$, etc. and $S$, $[SES]$, $[SE[SES]]$, etc.

The brackets can be dispensed with if we employ what is often called "Polish notation" (cf. Bar-Hillel 1964). A grammar avoiding ambiguity by using the Polish notation (for all sentence connectives except if–then) can be obtained if we restate the rules of $\Theta^0_p$ as follows: $S \rightarrow KS$, $S \rightarrow ESS$, $S \rightarrow ISTS$, $S \rightarrow NS$, $S \rightarrow \sigma$, $K \rightarrow \text{and}$, $E \rightarrow \text{or}$, $N \rightarrow \text{false that}$, $I \rightarrow \text{if}$, $T \rightarrow \text{then}$. The resulting Grammar $\Theta^2_p$ will allow the following derivations: $S$, $ESS$, $EKSS$, etc. and $S$, $KSS$, $KSESS$, etc. Here the terminal strings are $o$ and $\sigma$, and $\sigma$ or $\sigma$, respectively.

As is seen from the form of the rules, each of the grammars $\Theta^0_p$, $\Theta^1_p$, and $\Theta^2_p$ is context-free but not regular. Strings like the ones produced by $\Theta^2_p$ and $\Theta^0_p$ fulfill the requirements of logic since they have only one syntactic structure each, but they do not correspond to sentences of ordinary English since they either contain additional elements or are ordered in a different way. Strings like the ones produced by $\Theta^0_p$ do not have these disadvantages, but they are ambiguous. Neither grammar accounts for listing expressions like (1 a). While context-free grammars (as well as regular ones) are capable of generating such expressions, they are not able to generate them in such a way that their structural relatedness with expressions like (1 b) is indicated.

(1 b) Ann played the piano, and Peter beat the drums, and Mary plucked the bass, and Paul blew the horn.

This could be done through a grammar deriving expressions such as (1 a) from expressions such as (1 b) by dropping unnecessary connectives like the first two tokens of and in (1 b). (For discussion of the question whether a natural language like English can be regarded as a regular or context-free string code, cf. Chomsky and Miller 1963: 297–300, Bar-Hillel 1964: 94–98 and 114f, Postal 1964, Hiz 1968, Brainerd 1971: 176–181 and 186–195, and Pullum and Gazdar 1982; see also § 4.10. below).

4.7. Context-sensitive grammars

A production grammar $\langle V_C, V_T, S, R \rangle$ is context-sensitive if its rules take the form $\alpha_1 A \alpha_2 \rightarrow \alpha_3\alpha_4\alpha_5$ where $A$ belongs to $V_N$, $\alpha_1$ to $\Sigma(V_N \cup V_T)$, and $\alpha_3$ and $\alpha_5$ to $\Sigma(V_C \cup V_T)$. As an example, consider the grammar $\Theta^0_p$, which differs from $\Theta^0_p$ in that its non-termina-

ment $Q$ and in that the rule (i) $S \rightarrow SES$ is replaced by the three rules $S \rightarrow QE$, $QE \rightarrow QQE$, and $Q \rightarrow S$, where $QE \rightarrow QQE$ is context-sensitive. $\Theta^0_p$ easily produces sentences of the form $\sigma\varepsilon\sigma$, $\sigma\sigma\varepsilon\sigma$, etc.

Context-sensitive rules are sometimes presented in the form $A \rightarrow o\alpha_1 \ldots \alpha_n$, read "re-write $A$ as $o$ in the context $\alpha_1 \ldots \alpha_n$". On certain occasions this is preferable to the form $A_1 \alpha_2 \ldots \alpha_n$ because the $\alpha_i$s may also contain $A$'s, which renders the actual context that the writer had in mind unclear. For example, the rule $QE \rightarrow QQE$ can be read as $a_1 = Q$ and $a_2 = A$ or as $a_1 = A$ and $a_2 = E$. If we employ the above convention, the former intent is rendered $E \rightarrow QE / Q$ and the latter $Q \rightarrow QQ / E$. In case either $a_i$ is empty, the corresponding side of the bar is left empty. (For further details, including alternative terminol-

In special fields such as computer programming, string production grammars are used that hold an intermediate position between context-free and context-sensitive grammars. Such grammars are usually obtained on the basis of context-free rules by imposing special restrictions on the process of derivation. The most well-known are matrix grammars, controlled grammars, ordered grammars, scattered grammars, programmed grammars, and valence grammars (for dis-

2. Syntactics

While all these grammars can be regarded as special types of context-sensitive grammars, there are also string production grammars that do not fit in the hierarchy of gram-
mars discussed above. The most important are configurational grammars (cf. Gladkij 1963 and 1968), Lindenmayer systems (cf. Lindenmayer and Rozenberg 1976 and 1979, Herman and Rozenberg 1975, and Rozenberg and Salomaa 1974), and contextual grammars (cf. Păun 1982). Lindenmayer sys-
tems and contextual grammars are both char-
acterized by the fact that they do not have a non-terminal vocabulary. Moreover, Linden-
mayer systems require that every element of a given string be processed in each step of a derivation. Grammars with such properties have been successfully used in the description of cell structures and their development (cf. Claus, Ehrig, and Rozenberg 1979).
4.8. Grammars and automata

As pointed out in § 4.1. above, sets of complex signs can be specified in two complementary ways: through synthesis of elementary signs and through analysis of complex signs. Synthesis is an activity characteristic of a sign producer constructing a complex sign, whereas analysis is characteristic of a sign recipient decoding a given sign. While synthesis is simulated by the functioning of a production grammar which generates complex strings through a step-by-step rewriting of the starting symbol, analysis can be simulated by an automaton in the sense of the mathematical theory of automata (cf. Art. 78 § 5.3.). In addition to providing a given complex string with a syntactic analysis (which is called “parsing”; see § 4.1. above and cf. Aho and Ullman 1972: I, ch. 4–6), an automaton must distinguish well-formed strings from ill-formed ones (see § 4.3. above). Automata capable of solving this problem are known as “recognizers” (cf. Aho and Ullman 1972: I, 93) or “acceptors” (cf. Ginsburg 1966: 46).

If a string \( a \) over a vocabulary \( V \) occurs on a tape and is fed into an automaton \( \mathcal{A} \), it will start a computation which results in a classification of \( a \) either as well- or as ill-formed. The string \( a \) is said to be “accepted by \( \mathcal{A} \)” if it is classified as well-formed.

A very simple kind of automata are the “finite-state automata” (cf. Rabin and Scott 1959, Ginsburg 1966: 47, Maurer 1969: 86, Aho and Ullman 1972: I, 116 f.). A finite-state automaton \( \mathcal{A} \) consists of five components: (1) a finite, non-empty set \( A \) of states, (2) a finite vocabulary \( V \), (3) a transition function \( M \) (from \( A \times V \) to \( A \)), (4) an initial state \( s \), (5) a set \( F \) of distinguished final states (\( F \subseteq A \)). Thus we have \( \mathcal{A} = \langle A, V, M, s, F \rangle \).

As an example for the concepts introduced above, let us consider the automaton \( \mathcal{A}_H = \langle A_H, V_H, M_H, s_H, F_H \rangle \) for the recognition of the Hindu-Arabic numerals. \( \mathcal{A}_H \) has only one further state \( s^* \) in addition to its initial state \( s_H : A_H = \{ s_H, s^* \} \). Only \( s_H \) is a final state; thus \( F_H = \{ s_H \} \). \( V_H \) is the vocabulary of digits known as “\( V_M \)” from § 4.4., i.e.: \( V_H = \{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 \} \). The transition function \( M_H \) is specified by Tab. 2.1.

<table>
<thead>
<tr>
<th>( s_H )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s_H )</td>
<td>( s^* )</td>
<td>( s^* )</td>
<td>( s^* )</td>
<td>( s^* )</td>
<td>( s^* )</td>
<td>( s^* )</td>
<td>( s^* )</td>
<td>( s^* )</td>
<td>( s^* )</td>
<td>( s^* )</td>
</tr>
</tbody>
</table>
Now assume $\Delta_H$, being in its initial state $s_{H1}$, is fed with the string $1983$ (cf. the treatment of this string by the grammar $G^1_{14}$ in § 4.4.). Then $\Delta_H$ starts a computation which may be described by the following equations:

$$M^*_{H1}(s_{H1}, 1983) = M^*_{H1}(M_{H1}(s_{H1}, 1), 9, 1983)$$

$$= M^*_{H1}(M_{H1}(M_{H1}(s_{H1}, 1), 9), 8, 3)$$

$$= M_{H1}(M_{H1}(M_{H1}(s_{H1}, 1), 9), 9, 3)$$

Using the transition table we obtain:

$$M^*_{H1}(s_{H1}, 1983) = M_{H1}(M_{H1}(M_{H1}(s_{H1}, 9), 8), 3)$$

$$= M_{H1}(M_{H1}(s_{H1}, 8), 3)$$

$$= M_{H1}(s_{H1}, 7)$$

Since $s_{H1}$ is a final state (indeed the only one), the string $1983$ is accepted (classified as well-formed) by $\Delta_H$. Furthermore, inspection of the transition table yields the result that there is only one possible way of reaching the non-final state $s^*$ from $s_{H1}$, namely by starting with a configuration $\langle s_{H1}, 00 \rangle$. Thus the strings $\theta_0a$ are the only ones which are rejected (i.e.: not accepted) by $\Delta_H$. The string code $L(\Delta_{H1})$ is identical with the string code $L(\theta^1_{H1})$ generated by the grammar $\theta^1_{H1}$ of § 4.3.

Now the grammar $\theta^1_{H1}$ is of the most restricted type: it is a regular grammar. And since $L(\Delta_{H1}) = L(\theta^1_{H1})$, the string code accepted by the automaton $\Delta_{H1}$ is a regular string code. This is no accident: every string code accepted by a finite-state automaton is a regular one; and conversely, every regular string code is also accepted by some such device. Regular grammars and finite-state automata are equivalent in the following sense: the string codes generated by regular grammars are exactly those which are accepted by finite-state automata (Ginsburg 1966: 52, Maurer 1969: § 2.2., Aho and Ullman 1972: I, 118–121). This raises the question as to whether there are other kinds of recognizers corresponding to the other kinds of production grammars. This is indeed so: context-free grammars are matched by “pushdown automata” (see also Art. 78 § 5.3.). While finite-state automata cannot keep track of auxiliary computations performed in testing a string for well-formedness, pushdown automata have a memory available for this purpose. The formal definition of a pushdown automaton resembles that of a finite-state automaton, but differs from it in providing an additional vocabulary with a designated starting symbol. This vocabulary is used to store information. Accordingly, a configuration of a pushdown automaton $\Phi$ is a triple consisting of the current state of the automaton, the tape under investigation and the content of its memory. Of course, the transition function of a pushdown automaton has to take care of this additional structure. We shall not go into the exact details of the formal definition here (cf. Aho and Ullman 1972: I, 167–176) but simply state the result that a string code is context-free only when it is accepted by a pushdown automaton. In this sense, pushdown automata and context-free grammars are equivalent (cf. Ginsburg 1966: 69, Aho and Ullman 1972: I, 184).

The next correspondence requires the notion of a Turing machine (cf. Hermes 1961 = 1971: ch. 2 and Rogers 1967: 13–16 as well as Art. 78, § 5.3.). A Turing machine $\Xi$ is an automaton which is capable of scanning a tape which extends infinitely in both directions. This tape is divided into an infinite number of cells of which, however, only a finite number contain a single letter from a vocabulary $V$. In each move during a computation, $\Xi$ is supposed to operate on the content of just one single cell; this is the cell being examined by $\Xi$ (in that move). $\Xi$ can carry out just one of four operations in each move:

1. It can print a symbol from $V$ on the cell it is examining.
2. It can go one cell to the right.
3. It can go one cell to the left.
4. It can stop its computation. After each move, $\Xi$ is in a certain state belonging to one of a fixed set of states $T$. Again we have an initial state, final states, configurations and a transition function (cf. Fig. 2.4; for exact definitions see the literature quoted above). Now, a string code $L$ is accepted by a Turing machine if and only if $L$ is generated by a production grammar (be it restricted or not; cf. Maurer 1969: 153, 158 and 164).

Fig. 2.4: A Turing machine (after Meschkowski 1967: 599). Naturally, only a finite part of the infinite tape can be represented.
A Turing machine has available in its computation the unlimited space of an infinitely long tape. This is a rather strong idealization in as much as a real machine never has more than a finite space at its disposal. We can, however, modify this idealization by requiring that the machine stay within the range of the (possibly disconnected) inscription which is written on its tape. The most restrictive requirement is that the automaton may scan only cells within the region between the leftmost cell which is filled with an input sign and the rightmost cell which is thus filled; such a machine is called “a linear bounded automaton” (cf. Maurer 1969: 135 f, Aho and Ullman 1972: I, 100). Linear bounded automata correspond to context-sensitive grammars; i.e., a string code is context-sensitive if and only if it is accepted by a linear bounded automaton (cf. Maurer 1969: 142).

The four correspondences described above – namely those between production grammars and Turing machines, between context-sensitive grammars and linear bounded automata, between context-free grammars and pushdown automata, and between regular grammars and finite-state automata – are the core of what is known as “mathematical linguistics” (or “formal language theory”). This is a subdivision of mathematics which originated with the early work of Chomsky (1959a, 1963) and the report on the programming language ALGOL edited by Naur in 1963. Obviously, it is very useful to know whether a string code has a grammar of one of the specified types since this indicates its computational complexity. We then know what structure a mechanical device must have if it is to accept the strings of this code as well-formed and to reject all other strings (over the same vocabulary) as ill-formed. Furthermore the stated correspondences (as well as others of a similar kind) may indicate the psychological capacity required of the sign users. If they manage to communicate by means of a string code of type X (e.g., a context-free one), their capacity must be at least as strong as that of an automaton of the corresponding type X′ (viz., a pushdown automaton). The relationship between mathematical linguistics and psycholinguistics is elaborated by Levelt (1974) and Berwick and Weinberg (1984).

Of course, the capability of distinguishing the well-formed strings from the ill-formed ones is not enough in order to know a code. Someone who commands a code should also have (at least implicit) knowledge of the syntactic structures of its well-formed strings. An automaton should, therefore, not only classify a string as well- or ill-formed; in doing so it should also provide us with a reasonable parsing of the string tested. It is interesting to note that this is exactly what is achieved in the proofs of equivalence between grammars of certain types and the corresponding automata. (As it is often the case in mathematics, only the proofs supply the theorems with their full information content.) For instance, in showing that a context-free code L is also a pushdown code one uses a context-free grammar Θ to generate L and constructs on the basis of Θ a pushdown automaton Π which accepts L. Thus in computing whether a given string belongs to L or not, Π parses this string in accordance with Θ.

4.9. Syntactic structures and structural descriptions

The procedure by which a production grammar specifies the syntactic structure of a string is based on that string’s derivation. As has been pointed out in § 4.5, and 4.6., the derivational history of a string can be used to disambiguate it. In § 4.6., brackets were introduced to keep track of each step of the derivation. By labeling the brackets with the appropriate elements of the non-terminal vocabulary, each string in the output of the grammar can be given a description of its syntactic structure which is complete with respect to this grammar. (For discussion of this point cf. Reichenbach 1947: 164–167, Hermes 1961, Wang 1968, Schnelle 1970, and Posner 1971.)

Let us consider once again our example of a syntactically ambiguous string:

\[ (2) \sigma \text{ and } \sigma \text{ or } \sigma. \]

Instead of introducing brackets by including them in the rewrite rules as in \( \Theta_1 \Theta_0 \), we can construct labeled bracketings in the course of derivations by using the following bracketing convention: If a rule \( \Delta \rightarrow \alpha \) is applied to a string \( \beta = \chi_1 A \chi_2 \), write the result as \( \chi_1 [\alpha] \chi_2 \), where \( [\alpha] \) is a bracketing of \( \alpha \) labeled by \( \Delta \).

(For other versions of the bracketing convention cf. Montague 1970b, Brainerd 1971: 212 ff, and Hermanns 1977: 160 ff.)

In order to obtain the two structural descriptions indicated for (2), we proceed as follows:
Each line in the above derivations contains a string within labeled bracketing indicating the syntactic structure of this string. The difference in structure between (1.8) and (2.8) is due to the structural difference between (1.3) and (2.3), which is caused by the application of rule (i) to different element tokens in the string $S|E|S$ in (1.2) and (2.2), respectively. While the unstructured strings (1.8) and (2.8) are identical, the structured strings (1.8') and (2.8') are different: (1.8') contains $\sigma$ and $\sigma$ as a constituent, (2.8') does not. (2.8') contains $\sigma$ or $\sigma$ as a constituent, (1.8') does not.

We are now in a position to give more exact definitions for some of the terms used in describing the ambiguous string (3) and its structure. If for a given context-free grammar $\mathcal{G}$ we apply the bracketing convention, then any of the bracketings assigned to a terminal string $a \in L(\mathcal{G})$ is a description of the syntactic structure (or in short, a structural description) of $a$ in $\mathcal{G}$. Each partial string of $a$ included within a labeled pair of brackets is a constituent of $a$. The non-terminal symbol occurring as a label of the brackets indicates the syntactic class or category which this partial string belongs to. These definitions explicate the central terms used in the characterization of syntactic given by Morris (1938: 14 = 1971: 281 and 1946: 354 f = 1971: 367).

Even for a simple string like (2), the bracketings assigned by the grammar are quite complex, leading to a certain room for error due to inaccurate transcription. This is why some syntacticians prefer to think of structural descriptions in terms of parsing trees (also called "derivation trees", "phrase structure trees", or "P-markers") rather than in terms of labeled bracketings. As the two methods are interchangeable, it makes sense to define the notion of a parsing tree as well at this point.

Let $\mathcal{G} = (V_N, V_T, S, R)$ be a context-free grammar. Then each derivation $S, a_1, a_2, \ldots, a_n$ has associated with it a unique parsing tree which is constructed as follows: $S$ is the root. If $a_i = a_{i-1}a_i \ldots a_n$ where $a_i \in V_N \cup V_T$ for $i = 1, 2, \ldots, n$, then the parsing tree has nodes $1, 2, \ldots$. In labeled respectively $a_1, a_2, \ldots, a_n$, which are connected to the root by lines as in Fig. 2.5. If $a_2$ is obtained from $a_1$ by application of a rule $a_1 \rightarrow a_2 \ldots a_n$ where $a_i \in V_N$ and $a_j \in V_N \cup V_T$ for $1 \leq j \leq k$, then the nodes $1, 2, \ldots, k$ are connected to the node labeled by lines and are labeled $b_1, b_2, \ldots, b_k$ respectively as in Fig. 2.6. If this process is continued until the terminal string is reached, we ultimately obtain a unique labeled tree corresponding to the derivation in question. This tree is the parsing tree which $\mathcal{G}$ assigns to the terminal string of that derivation.

Fig. 2.7 and Fig. 2.8 give the structural descriptions of (2) in the form of parsing trees corresponding to derivations (1.1') through (1.8') and (2.1') through (2.8'), respectively.
Trees such as those of Fig. 2.7 and 2.8 are special kinds of graphs and are investigated in graph theory. (For some basic concepts from graph theory cf. § 5.2, below.) However, most of the graph-theoretical concepts used in syntactics have such an intuitively clear geometric character that they may be introduced without recourse to this subdiscipline of mathematics. A single line which connects two nodes of a tree is called “an arc” or “an edge”; thus every arc connects exactly two nodes. The arcs of a tree are given an orientation by the top-bottom direction of the drawing. The higher node of an arc is said to “directly dominate” the lower one; alternatively, the lower one is said to be “a daughter” of the higher one, which is called its “mother”. Thus the mother node always directly dominates the daughter node. Daughter nodes of the same mother node are called “sister nodes”. A node k is said to dominate another node k’ if and only if there is a sequence k₀, k₁, ..., kₘ (m ≥ 1) such that for each 1 (0 ≤ l < m) kₗ directly dominates kₗ₊₁. The single node which is not dominated by another node but dominates all others is the root (see above). Nodes which do not dominate any other nodes are called “leaves”. Note that a tree has no isolated nodes; each node either dominates or is dominated by another one. Furthermore, each node is dominated by at most one single node (its mother node) and the root is the only node without a mother. In a labeled tree, the concatenation of the labels of the leaves (in their left-to-right order) is called the “yield” of the tree. Thus the yield of the derivation tree for a string α is just that string α.

For approaches to the construction of parsing trees that do not rely on the derivational history of the strings parsed, cf. § 5.3.

4.10. Transformational grammars

As was shown in § 4.8., it is very useful to know what type of grammar is sufficient to generate a given code. However, this information is often hard to obtain, especially when the code was not introduced through explicit stipulations but is empirically given through the semiosic behavior of its users. This is typically the case for natural languages and their grammatical status is therefore controversial (cf. § 4.6. above). Most linguists will nowadays agree that natural languages are too complex to be classified as regular and treated by finite-state automata. This conviction can certainly be traced back to Chomsky’s (1959 b) famous criticism of behaviorism. Finite-state automata simulate the stimulus-response-behavior learned through trial-and-error, which is the only way of learning recognized by radical behaviorists (cf. Suppes 1969). If one can show (as Chomsky is supposed to have done) that natural languages can never be acquired by way of trial-and-error alone, one has thus demonstrated that they are not within the reach of finite-state automata, and this means that they are not regular (cf. Wexler and Culicover 1980 for a classical exposition of the formal problems of language learning seen from the perspective of Generative Grammar).

Admittedly, this is a rather indirect way of arguing against the case of regular grammars for natural languages. A more direct approach which would settle the problem without any recourse to the theory of language learning would be to look for natural language constructions that can be demonstrated not to be regular. For instance, the nonregular character of the language L̅sym is known, where L̅sym = {αⁿ{b}ⁿ}, i.e., L̅sym comprises exactly those strings over the two-item vocabulary V = {a, b} which consist of a number of occurrences of the character a followed by an equal number of occurrences of b. In order to demonstrate the inadequacy of regular grammars, one could try to find a set of constructions in a natural language which resemble L̅sym. Sometimes the nested embedding of relative clauses is seen in this way; cf. the example in Fig. 2.9 and the discussion by, e.g., Kratzer, Pause and von Stechow (1973: 138–139). An analogous argument, concerning context-free languages, involves English sentences using several occurrences of the word respectively (cf. Postal 1964 and Kratzer, Pause and von Stechow 1973: 142). Based on his reasoning against linguistic behaviorism and on examples such as that of Fig. 2.9, Chomsky considered it necessary to supplement string production rules by what
he called “transformation rules”. Transformations had already been postulated by Chomsky’s teacher Harris, who motivated them with the needs of discourse analysis (cf. Harris 1952 as well as 1970 and 1981). But whereas Harrisian transformations are mappings from strings to strings, Chomskyan transformations are of a more abstract character (cf. Joshi 1973 for a technical comparison of the two transformationalist approaches). Chomsky proposed to divide the derivation of a well-formed string  \( a \) of a given language \( L \) into several stages: first a structural description of  \( a \) is produced which encodes the most basic syntactic information about  \( a \); this structural description specifies the so-called “deep-structure”, which is the basis for the derivation of both a semantic and a phonological representation of  \( a \).

Beginning with the late 1950s, Chomsky published studies of a series of grammar formats which followed this idea and were labeled either with the publication dates or with key words of the book titles or with special acronyms. Thus the grammar format described in Chomsky’s 1957 publication as well as in his 1975 book (which had been circulated in various mimeographed versions since 1955) is referred to as “the 1957 Model”; the model presented in 1965 is called “the Aspects theory” or “the Standard Theory (ST)”; it was followed by “the Extended Standard Theory (EST)” and by “the Revised Extended Standard Theory (REST)”. The version of Transformational Grammar that originated in Chomsky’s 1981 publication is called “Government and Binding Theory (GB)”; it has been reworked in further publications such as Chomsky 1986a and 1986b. Introductions to GB are provided, e.g., by Radford (1981 and 1988) and Riemsdijk and Williams (1986); some issues of GB are also discussed in § 5. below. Here we shall not deal with this recent development of Transformational Grammar but give a brief survey of the Standard Theory (and some of its amendments). This is done because of the historical importance of this theory, which dominated linguistic research for more than a decade. Introductions to the Standard Theory and its extensions are provided, e.g., by Akmajian and Heny (1975) and Baker (1978).

As already mentioned, the Aspects Theory calls the structure underlying a string its “deep structure”. Deep structures are postulated for a wealth of reasons, which cannot all be reported here. But one of the best arguments for their acceptance is that an adequate grammar should explain the systematic co-occurrence of active and passive sentences (cf. example 3) or of sentences with a prepositional object and corresponding ones with an indirect object construction; (cf. example 4).

(3a) John kissed Mary.
(3b) Mary was kissed by John.
(4a) I gave the money to my friend.
(4b) I gave my friend the money.

In Transformational Grammar pairs of strings such as (3a, b) and (4a, b) are said to have the same deep structure but differ in what is called their “surface structure”. Now deep structures are specified by means of the derivation trees of a context-free grammar (cf. § 4.4–4.6.), and the transformation rules have the task of mapping deep structures to surface structures in several steps. Formally, transformation rules are functions which map labeled trees to labeled trees again. Allowing only for unary functions of this kind (i.e., functions with only one argument) the Aspects Theory of 1965 differs from the account given by Chomsky in 1957.

Of course, not every unary function which takes labeled trees as its arguments and yields labeled trees as its values is recognized as a genuine transformation. However, before explaining the details of a transformation rule,
let us take a look at the overall structure of a transformational grammar \( \mathfrak{G} \) for a language \( L \). As already mentioned above, the items to which a transformation rule may be applied are specified by means of the derivation trees of a context-free syntax \( \mathfrak{B} = \langle V^B_{NT}, V^B_T, S, \mathcal{R} \rangle \). The letter “B” stands for “Base”, but the syntax \( \mathfrak{B} \) is only one constituent of the so-called “base component” of a transformational grammar \( \mathfrak{G} \), the others being a set of subcategorization rules and a lexicon (see below). The derivation trees of \( \mathfrak{B} \) are called “pre-terminal structures”. The elements of the vocabulary \( V^B_T \) are not yet the symbols of the strings of the language \( L \) to be analyzed but are strings of symbols for grammatical categories of \( L \). Thus the yields of the pre-terminal structures, i.e., the pre-terminal strings, are not strings of \( L \). In the case of a natural language the pre-terminal strings are strings of grammatical symbols like \( N(oun) \), \( N(oun) \ P(hrase) \), \( V(erb) \), \( V(erb) \ P(hrase) \), \( A(djective) \), \( A(djectival) \ P(hrase) \), \( P(reposition) \ P(hrase) \), \( A(rticle) \) etc. Within Transformational Grammar, the syntactical atoms which make up the strings of our language \( L \) are called “formatives”. The formatives have to be inserted into the pre-terminal structures generated by \( \mathfrak{B} \) by means of lexical rules (cf. §4.4.) in a very complicated process which is called “lexical insertion”. For illustration, let us consider the simple tree in Fig. 2.10 as an example of a pre-terminal structure:

![Fig. 2.10](image)

Having derived such a tree, let us substitute word forms of English for the pre-terminal symbols \( N \) and \( V \). (In Transformational Grammar, more abstract units than word forms are used, but for simplicity of exposition we neglect this further complication.) We might use tentative lexical rules like John\( \rightarrow \)NP, Mary\( \rightarrow \)NP and kissed\( \rightarrow \)V and attach the item John as a daughter to the leftmost NP, kissed to the V and Mary to the rightmost N. But obviously, we may (other things being equal) not insert an intransitive verb under \( V \) in this context. This would result in an ill-formed sentence such as *John behaved Mary. The verb form kissed is said to be “subcategorized” for a Noun Phrase as its direct object, and it is required that each item only appears together with elements for which it is subcategorized. Thus kissed takes a direct object, whereas behaved cannot be accompanied by such an additional phrase. Such restrictions are implemented by means of subcategorization rules. They expand the preterminal symbols into complex structures which provide information about the contexts in which the terminal symbols must appear. For instance, the \( V \) in the tree above is rewritten as a complex symbol which consists of the so-called “feature specifications” \( [V: +] \) (cf. §5.1. for more recent treatments of subcategorization and §5.2. for treatments of syntactic features).

The rewriting in our example is achieved by means of the following subcategorization rule, which has the form of a context-sensitive string production rule:

\[
V \rightarrow [V: +, \text{trans: +}] \rightarrow N
\]

Application of this rule to the derivation tree in Fig. 2.10 results in the tree of Fig. 2.11:

![Fig. 2.11](image)

The formatives of language \( L \) and their relevant grammatical properties are specified in the lexicon of the grammar \( \mathfrak{G} \). It will contain lexical entries which allow the rewriting of a complex category symbol through a word form as in:

\[
kissed \rightarrow [V: +, \text{trans: +}]
\]

\[
behaved \rightarrow [V: +, \text{trans: -}]
\]
These lexical entries guarantee that only those word forms are inserted into a pre-terminal structure which are compatible with the feature specifications of the complex symbol. Our lexical entry for behaved, for instance, contradicts the specification \( \text{trans: +} \) under the \( V \)-node in the tree of Fig. 2.11; thus the lexical item behaved cannot be inserted under the \( V \)-position. But obviously, the item kissed is allowed there. The rewriting of all non-terminal symbols in a pre-terminal structure of all non-terminal symbols in a pre-terminal structure and appropriate lexical entries results in a deep structure as in Fig. 2.12.

![Fig. 2.12 Deep structures are turned into surface structures by means of transformation rules. There are two sets of such rules: (1) the class \( R_{ob} \) of obligatory rules, which must be used where applicable, and (2) the set \( R_{fac} \) of facultative rules, which may or may not be applied where applicable. The class \( R \) of the transformation rules of our grammar \( \mathcal{G} \) then is the union of these two classes: \( R = R_{ob} \cup R_{fac} \).

A transformational derivation takes the form of a sequence \( \delta_0, \ldots, \delta_m \) where (a) \( \delta_0 \) is a deep structure resulting from a derivation tree \( \delta \) of \( B \) through subcategorization and lexical insertion and (b) each \( \delta_i \) (\( 0 < i < m \)) results from its predecessor \( \delta_{i+1} \) through the application of a rule \( \tau \) from \( R \), and (c) all members of \( R_{ob} \) are applicable at all in the course of the derivation are in fact applied. For an exact definition of the notion of a transformational derivation the reader is referred to Kimball (1967), Ginsburg and Partee (1969), Peters and Ritchie (1973) and Kratzer, Pause and von Stechow (1973: 272–281). Let us note here that the transformations are often postulated to be ordered in some way, so that the sequence of transformations to be applied in the course of a transformational derivation is not free but restricted by a rule ordering \( \Omega \). There are several possibilities as regards the exact order type of \( \Omega \) and the way \( \Omega \) achieves the ordering of rule applications. For instance, \( \Omega \) may specify some kind of either linear or cyclic order. Furthermore, rule ordering may be achieved through explicit stipulation (extrinsic order) or by defining the transformation rules in such a way that each rule has only one possible successor (intrinsic order). The linguistic implications of these matters are discussed, e.g., by Koutsoudas (1976). The transformation rules and the rule ordering make up the transformational component of the grammar.

The complete grammar \( \mathcal{G} \) then consists of the base component, which comprises the base syntax \( \mathcal{G} \) as well as subcategorization rules and the lexicon, and the transformational component, which comprises the sets \( R_{ob} \) and \( R_{fac} \) of obligatory and facultative transformation rules as well as the rule ordering \( \Omega \). The base component generates deep structures from which the surface structures are derived using transformation rules in accordance with the ordering \( \Omega \). The whole architecture of \( \mathcal{G} \) may then be depicted as in Fig. 2.13.

Since we now have an overall view of the functioning of a transformational grammar, let us turn to the structure of the transformation rules themselves. A transformation rule \( \tau \) consists of two finite sequences: (1) a sequence of “structural descriptions” (SDs) and (2) a sequence of “structural changes” (SCs). The structural descriptions delimit the range of those trees to which the transformation rule in question is applicable. They are usually given in the form of indexed cuts through the labeled trees. Consider, for instance, the tree in Fig. 2.14 with the cut indicated by 

\[
\begin{pmatrix}
X & V & NP & P & NP \\
1 & 2 & 3 & 4 & 5
\end{pmatrix}
\]

As the numbering of the category symbols shows, the cut in Fig. 2.14 leads through five nodes of the tree, labeled with the respective category symbols, with \( X \) serving as a variable. Thus “

\[
\begin{pmatrix}
X & V & NP & P & NP \\
1 & 2 & 3 & 4 & 5
\end{pmatrix}
\]

” says that the first node of the cut may be labeled with any category symbol whatsoever, the second, however, must be labeled with a \( V \), the third with an \( NP \), etc. The five subtrees which are dominated by the nodes of the cut are
required to have such yields that their concatenation (in the left-right-order of the nodes) equals the yield of the entire tree. For instance, the leftmost node numbered 1 in the above cut dominates a tree with the yield \(He\); the second node has the yield \(gave\), the third node the yield \(the\ money\), and the fourth node the yield \(to\), and the last node the yield \(me\). Concatenation of these five yields results in the yield of the entire tree, namely \(He\ gave\ the\ money\ to\ me\). (Structural descriptions in the form of cuts resemble the proper analysis predicates of § 5.3., differing from them only in permitting variables and requiring numerical indices.)

In the specification of the structural changes (SCs), the numerical indices of the structural descriptions (SDs) serve to indicate how the input tree must be modified. If the material dominated by the node of the input tree with index \(m\) is to be deleted, this is coded as \(m \rightarrow \emptyset\) within the specification of the structural change (SC). If the material dominated by node \(m\) is to be moved from its original position to the position right of the material dominated by node \(n\), this is coded as \(m \rightarrow n + m\). Such deletions and movements are the most important types of structural changes. (For other types the reader should consult the literature quoted above.) With respect to the SD in Fig. 2.14, we might, for instance, formulate the transformation rule \(\tau_{d\text{m}}\):

\[
\tau_{d\text{m}}: \quad \text{SD:} \quad \left(\frac{X}{1} \quad \frac{V}{2} \quad \frac{NP}{3} \quad \frac{P}{4} \quad \frac{NP}{5}\right) \quad \text{SC:} \quad \begin{align*}
2 \rightarrow 2 + 5 \\
4 \rightarrow \emptyset \\
5 \rightarrow \emptyset
\end{align*}
\]

Application of the transformation rule \(\tau_{d\text{m}}\) to the tree in Fig. 2.14 results in a tree which resembles the one in Fig. 2.15, with the exception of having an additional arc on the right-hand side which leads from \(VP\) to a node \(PP\) that does not dominate any lexical material.
The operation of omitting such void arcs is known as “tree pruning”; and a tree pruning convention ensures that void arcs will not appear in the surface structure. The transformation described above is known as “Dative Movement”. (An empirically more adequate formulation of Dative Movement for a transformational grammar of English is given in Akmajian and Heny 1980: 183–186; cf. also Larson 1988.)

As the reader will certainly agree, transformational grammars of the kind set out here provide a rather complex framework for the syntactic description of complex signs. Thus it came as a surprise when Peters and Ritchie (1973) proved that production grammars and transformational grammars are equal in weak generative capacity (cf. also Pause 1976). This means that every language which is generated by a transformational grammar (i.e.: whose well-formed strings are the yields of the surface structures of that grammar) can also be generated by a production grammar (and vice versa).

However, this result does not render superfluous the transformationalist approach to syntactic description of natural languages. The more complex transformational descriptions might take account of the structural facts more adequately and might provide better explanations, e.g., for the ways natural languages are acquired and for the ways they change (cf. the brief discussion of Peter and Ritchie’s result in Chomsky 1981: 129).

Nevertheless, the development of Transformational Grammar in the late 1970s and 1980s was characterized by the search for well-motivated theoretical restrictions of the power of possible transformation rules. The Aspects and the Standard Theories had allowed the structures derived in the base component to be considerably modified and even mutilated in the transformational component, but since Emonds (1976) transformationalist linguists tended to keep the deep structures more similar to the surface structures. While the earlier approaches had tried to characterize each kind of surface structure by postulating a specific type of transformation rule, such as the Passive Transformation, Dative Movement, there-Insertion, etc., more recent transformational theory has tended to explain such constructions by means of a limited number of rather general grammatical principles. Chomsky’s Government and Binding Theory (GB) of 1981 assumes only one transformation rule, called “move-a” and providing for constituents to be moved. At the same time, however, the syntactic structures postulated for the sentences of natural languages have become more abstract, with poor surface structures and rather remote D-structures (the recent equivalents of deep structures). This growing distance of present-day transformational theorizing from the practice of grammatical description (e.g., for the purposes of second language learning and translation) may be the reason why the more recent versions of Transformational Grammar never reached the degree of acceptance enjoyed by the Aspects and the Standard Theories.

5. Recent developments in formal syntactics

Quite independently of the fate of Transformational Grammar, formal syntactics has made considerable progress since the 1970s, and this applies both to the analytic acumen of its conceptual tools and its range of applications. Some of these developments are covered in the following paragraphs; § 5.1 and § 5.2. are devoted to refinements of the technical apparatus of formal grammars, including X-bar-syntax (§ 5.1.1.), complex features (§ 5.1.2.), subcategorization (§ 5.1.3.), and thematic roles (§ 5.1.4.) as well as graph theory (§ 5.2.1.), markedness theory (§ 5.2.2.), and unification theory (§ 5.2.3.); § 5.3. and § 5.4. deal with generalizations of formal grammars that allow their application in the syntactic analysis of non-string codes.

5.1. X-bar syntax, subcategorization, and thematic roles

The examples of production grammars in § 4.3.—4.5. are peculiar in employing very restricted non-terminal vocabularies. Thus, the
non-terminal vocabulary of the grammar \( \Theta^*_1 \), which generates the Hindu-Arabic numerals, includes only the three members \( Z \), \( M \) and \( N \). Furthermore, each syntactic category referred to by an item of a non-terminal vocabulary is treated as an atom without any internal structure; potential relationships between categories are left unconsidered. As an example, take the category \( N \) of \( \Theta^*_1 \), which is a proper subcategory of \( M \). Each terminal symbol belonging to \( N \) is also an element of \( M \) whereas \( M \) includes the terminal \( \theta \) which is not an element of \( N \). The two categories \( M \) and \( N \) in turn are (proper) subcategories of the starting category \( Z \). Also, \( N \), \( M \) and \( Z \) are all categories of numerals, but while \( Z \) is the universal category of all numerals, \( N \) and \( M \) are categories of digits. \( N \) is the category of digits designating natural numbers, and \( M \) the category of digits designating non-negative integers.

5.1.1. X-bar syntax

Relationships between syntactic categories may be described by means of syntactic features which are designated by feature symbols (cf. Brockhaus 1971, Kratzer, Pause and von Stechow 1973–74: II 5–14, as well as Gazdar, Klein, Pullum and Sag 1985: 17–42). We may, for instance, distinguish \( N \) and \( M \) by means of the feature \([ \text{zero} ]\) designated by \([ \text{zero} ]\). (Following a convention from phonology, feature symbols are written with square brackets.) This feature may assume one of two possible values, which are designated by \( + \) and \( - \). The symbols \( + \) and \( - \) themselves are also feature symbols, sometimes called “singleton symbols” (Smolka 1988: 9). No deeper distinction needs to be made between feature symbols that designate features, and feature symbols (such as singleton symbols) that designate values, because features can themselves occur as values. A feature specification is an expression of the form \( [f: v] \) where \( f \) and \( v \) are feature symbols; the specification \( [f: v] \) says that the feature designated by \( f \) takes the value designated by \( v \). Category symbols are now analyzed as consisting of such feature specifications (possibly besides other components such as sort symbols; cf. § 5.1.3. below); conversely, feature specifications are regarded as syntactic parts of category symbols. Thus, the symbol \( N \) is now analyzed as including, e.g., the feature specification \([ \text{zero: } - ]\) as its part, which indicates that the category \( N \) designated by \( N \) does not have the digit \( \theta \) as its element, whereas for \( M \) we have of course \([ \text{zero: } + ]\) because the category \( M \) designated by \( M \) does contain the digit \( \theta \). Features such as \([ \text{zero} ]\) which can take only one of two possible values are called “Boolean features”.

Complex category symbols consisting of feature specifications are also called “feature structures” (Shieber 1986: 12) or “feature terms” (Smolka 1988: 10); they are named “\( f \)-structures” in LFG (i.e., Lexical Functional Grammar; cf. the contributions in Bresnan 1982); in GPSG (i.e., Generalized Phrase Structure Grammar; cf. Gazdar 1982, and Gazdar, Klein, Pullum, and Sag 1985) they are simply called “categories”. This, however, obliterates the distinction between the category symbols and the categories designated by these symbols. While such a mixing of levels may be harmless and may even simplify the exposition of a formalism, it should be observed that real progress in the formal theory of features has only been made on the basis of a “clear distinction between descriptions and what they denote” (Smolka 1988: 7).

What remains to be treated in our reformulation of \( \Theta^*_1 \) is the difference between \( Z \) on the one hand and \( N \) and \( M \) on the other, which is not yet captured by the feature \([ \text{zero} ]\). Since \( Z \) does not include the digit \( \theta \), we have \([ \text{zero: } - ]\) as a part of the category symbol \( Z \), but this only sets \( Z \) apart from \( N \) and does not distinguish \( Z \) from \( M \). The difference between the categories \( Z \) versus \( M \) and \( N \) is fairly obvious, however. Expressions of category \( Z \) may be complex while those of \( M \) and \( N \) are always simple digits. In linguistics, the feature \([ \text{bar} ]\) is used for the description of varying degrees of syntactic complexity, and syntactic theories employing such a feature are known as versions of X-bar syntax (cf., e.g., Jackendoff 1977). The name of this theory derives from the convention of designating the syntactic complexity of a category through the number of bars written above the category symbol (alternatively, sometimes primes are used instead of bars). Thus, \( V \) (no bar = bar-level 0) designates the category of simple verbs (e.g., \( \text{swims} \), \( \text{loves} \), \( \text{gave} \), etc.), \( V^* \) (or, employing primes instead of bars, \( V' \)) designates the category of modified verb phrases (e.g., \( \text{swims slowly} \), \( \text{loves Mary ardently} \), \( \text{gave the book to Peter} \)).
based on the syntactic work of Harris (1947: ch. 16).

Before deciding on the bar-levels in our sample grammars, let us have a closer look at the two rules (i') and (ii) in § 4.4, parts of which are repeated here as (R1) and (R2) for easy reference.

(R1) \( Z \rightarrow N \)
(R2) \( Z \rightarrow NM \)

The intuition behind this simple rule system is that we may write a digit behind a numeral in order to produce a new numeral, but this process may never start with a zero (cf. R1) and the first digit of a complex numeral must be an element of category N (cf. R2). Thus we may say that a numeral consists of a non-zero digit followed by a string of further digits which may begin with a zero. Let us call the category of such strings (e.g., 0, 01, 02, 03, ..., 1, 11, 12, ..., 001, 002, ...) \( E \) (“ending”), and let us add E as a further category symbol to the non-terminal vocabulary of our rule system (cf. § 4.4.). Then, as a first step towards the grammar \( \Theta_{31}^{1} \), we may replace the rules (R1) and (R2) by the slightly more complex rules (R1') and (R2').

(R1') \( Z \rightarrow N (E) \)
(R2') \( Z \rightarrow M (E) \)

As usual, parentheses indicate optional elements. Thus (R1') abbreviates the two production rules: \( Z \rightarrow N \) and \( Z \rightarrow NE \). Of course, it is not necessary to postulate a category such as E for the generation of the Hindu-Arabic numerals, as the rule systems of our original grammars \( \Theta_{31}^{1} \) and \( \Theta_{31}^{2} \) show. But if we now adopt the feature [bar] with the two possible values - and +, the inclusion of E becomes very natural. Since we have the two features [zero] and [bar], taking two values each, there are 2·2 possible combinations of feature-value-pairs (whereas our original grammar has only 3 categories). These pairs can be used to characterize the categories Z, M, N, E as shown in Tab. 2.2:

<table>
<thead>
<tr>
<th></th>
<th>[zero: +]</th>
<th>[zero: -]</th>
</tr>
</thead>
<tbody>
<tr>
<td>[bar: 0]</td>
<td>M</td>
<td>N</td>
</tr>
<tr>
<td>[bar: 1]</td>
<td>E</td>
<td>Z</td>
</tr>
</tbody>
</table>

Following Tab. 2.2, the category symbol \( M \) is now construed as the collection of the feature specifications [bar : 0] and [zero: +], \( N \) as [bar : 0] and [zero : -], \( E \) as [bar : 1] and [zero : +] and \( Z \) as [bar : 1] and [zero : -]. Collections of feature specifications are written by means of long square brackets as shown in the first row of Tab. 2.3.

Obviously, the above complex category symbols come in pairs: the symbols for M and E differ only in their respective bar-specifications, as do those for N and Z. In cases like this, an abbreviation is used for the shared feature specifications, and such abbreviation symbols are then indexed with an numerical superscript in order to indicate the bar-level. We chose the category symbols \( Z \) for [zero : -] and \( E \) for [zero : +]. These conventions are employed in the following reformulation of the rules (R1') and (R2'):

(R1') \( Z^l \rightarrow Z^l(E^l) \)
(R2') \( E^l \rightarrow E^l(E^l) \)

As the reader will immediately recognize, the two rules (R1') and (R2') have a similar structure, which can be expressed by the very condensed rule scheme (X):

(X) \( X^m \rightarrow X^{m-l} (Y^m) \), for \( 0 \leq m \leq 1 \)

The possibility of economizing the formulation of rules by means of rule schemes is an

<table>
<thead>
<tr>
<th>The complex category symbol:</th>
<th>[bar: 0]</th>
<th>[bar: 0]</th>
<th>[bar: 1]</th>
<th>[bar: 1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>[zero: +]</td>
<td>E(^0)</td>
<td>Z(^0)</td>
<td>E(^l)</td>
<td>Z(^l)</td>
</tr>
<tr>
<td>is abbreviated by:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>and designates the category:</td>
<td>M</td>
<td>N</td>
<td>E</td>
<td>Z</td>
</tr>
</tbody>
</table>
additional reason for the introduction of syntactic features and complex category symbols. We shall return to rule schemes below. But let us complete the description of our new grammar \( \Theta_{IH} = \langle \mathcal{V}^{0}_{NT}, \mathcal{V}^{1}_{T}, \mathcal{S}^{1}, \mathcal{R}^{3} \rangle \) for the Hindu-Arabic numerals. So far we have only specified the following items: (i) the non-terminal vocabulary \( \mathcal{V}^{1}_{NT} \), which contains the four complex category symbols shown in Tab. 2.3, (ii) the terminal symbols of \( \mathcal{V}^{1}_{T} \), which are 0, 1, ..., 9, as before, and (iii) the starting symbol, which is now \( Z^{1} \).

(Note that the symbol \( Z^{1} \) simply stands for the complex symbol in the last field of the first row of Tab. 2.3.) But as members of \( \mathcal{R}^{3} \) we have only given the two rules \( \text{(R1)} \) and \( \text{(R2)} \) and, alternatively, the rule scheme (X). These are supplemented now by the following lexical entries for the members of \( \mathcal{V}^{1} \):

\[
\begin{align*}
\text{(R3')} & : 0 \rightarrow [\text{bar: } 0^\text{zero: +}] \\
\text{(R4')} & : \begin{cases} 
1 \\
2 \\
3 \\
9
\end{cases} \rightarrow [\text{bar: } 0^\text{zero: } x] \quad \text{where } x \text{ may be either + or } -
\end{align*}
\]

Note that the use of complex symbols permits us to condense eighteen of the lexical entries of \( \Theta_{IH} \) (cf. (i') and (ii') in § 4.4.) into the nine instances of (R4'). Thus instead of the twenty-one rules of \( \Theta_{IH} \) we only have the one rule (R3') plus the two schemes (X) and (R4') in \( \Theta_{IH} \). This gain in economy may not look very impressive in the case of our sample grammars, but this is due to the simplicity of the numeral code. Kratzer, Pause and von Stechow (1974: 18) provide a grammatical fragment of German formulated with complex category symbols where 62,958,336 rules formulated with atomic symbols can be replaced by one single rule scheme. On the basis of \( \Theta_{IH} \), the derivation of the numeral 1983 takes the form given in the parsing tree of Fig. 2.16.

5.1.2. Complex feature structures

The feature structures of our sample grammar \( \Theta_{IH} \) are extremely simple. To get a better grasp of the descriptive power of feature structures, let us consider the architectonic order of antique columns as a more complex example (cf. Fig. 2.17). For our restricted purposes, it will suffice to distinguish three orders: the Doric, Ionic, and Corinthian ones (a more detailed account of the types of columns in classical architecture is given in Art. 69; cf. also Eco 1972). A classical column consists of at most three parts: a base, a shaft, and a capital. A complex symbol \( F \) characterizing a category of columns should therefore include specifications of the three symbols [base], [shaft], and [capital]. Doric columns do not possess a base whereas Ionic and Corinthian ones do. Ignoring the finer differences between Ionic and Corinthian bases, we construe [base] as a Boolean feature symbol which admits the two value symbols + (base) and − (no base). In contrast to the Boolean feature symbol [base], the value symbols for [shaft] and [capital] are not atomic but consist in the complex symbols \( F_{s} \) and \( F_{c} \) that provide more detailed feature descriptions of the shaft and capital. Including these structures, the symbol \( F \) characterizing an entire column then is:

\[
F = \begin{cases} 
\text{base: } \pm \\
\text{shaft: } F_{s} \\
\text{capital: } F_{c}
\end{cases}
\]

We describe the shaft by means of two feature symbols: [flute] provides the number of flutes of the shaft; the two possible numbers will be designated by the singleton symbols \( 20 \) (Doric) and \( 24 \) (Ionic and Corinthian). The shaft of a column of the Doric order is more tapered than that of columns of the other two orders. We represent this by the Boolean feature symbol [taper]. Then the complex value symbol of the feature symbol [shaft] for the category of Doric columns is:

\[
\begin{cases} 
\text{flute: } 20 \\
\text{taper: } +
\end{cases}
\]
The capitals of the columns have three properties. Each column has an abacus (which is a narrow, square slab at the top of the capital) and an echinus (which is a convex molding at the bottom of the capital). We do not represent these properties in the feature symbol of a column because they do not differ for the three categories of columns, and there would be little point in having the Boolean feature symbols (abacus) and [echinus] if they could never receive a negative value. But Ionic and Corinthian capitals each have a volute, which Doric ones lack. Therefore we adopt a Boolean feature symbol [volut] which is specified negatively for the category of Doric column and positively for the two other categories. Furthermore, the capital of a Doric column is the most simply decorated of all three, while that of a Corinthian column with its characteristic garland of stylized acanthus leaves is more decorated than that of an Ionic one. So we adopt two further Boolean feature symbols [decor], which is specified negatively for the Doric but positively for the other two column categories, and [acant], which is specified positively only in the case of the Corinthian capital.

As a result we obtain the following description of the Corinthian column, which can be taken as a (modestly) complex example of a complex category symbol (complex category symbols characterizing the other categories of columns are given in § 5.2.):  

\[
\begin{align*}
\text{base:} & + \\
\text{shaft:} & \begin{cases} 
\text{flute:} & 24 \\
\text{taper:} & -
\end{cases} \\
\text{capital:} & \begin{cases} 
\text{volut:} & + \\
\text{decor:} & + \\
\text{acant:} & +
\end{cases}
\end{align*}
\]

\[(CC)\]

Obviously, there is a certain degree of redundancy in this feature description, because it is impossible for the feature [volut] to be specified negatively if [decor] takes the value +. The latter feature in turn must be specified positively if [acant] is. Thus we can predict the positive specifications for [volut] and [decor] from that of [acant]. We shall come back to such relationships between features in § 5.2.

5.1.3. Subcategorization

Let us return to our sample grammar \( \mathcal{G}^{14} \) and its scheme (X). This scheme describes a complex expression of any category (be it Z\(^1\) or E\(^1\)) as consisting of one obligatory constituent (belonging to category Z\(^2\) in the case of Z\(^1\) and to E\(^0\) in the case of E\(^1\)) which is facultatively followed by an expression belonging to category E\(^1\). Furthermore, the obligatory element differs from the whole category only with respect to its lower bar-level but not with respect to the value for the remaining feature [zero]. We may therefore say that the complex expression differs from its obligatory constituent in its complexity but not in its overall syntactic character (which in the case of numerals is exhausted by the [zero]-value). Traditionally, a construction (e.g., one belonging to Z\(^1\) or to E\(^1\)) with an obligatory constituent that has the same syntactic character as the whole is called “endocentric” — the obligatory constituent (e.g., of categories Z\(^0\) and E\(^0\)) being known as “head”, “center” or “nucleus” and the facultative constituent as “modifier” of the head (center of nucleus; cf. Bloomfield 1933: ch. 12.10 and Hockett 1958: ch. 21 for a treatment of endo-
centric constructions within the framework of structuralist syntax). However, the traditional concept of an endocentric construction differs in certain respects from the one in X-bar theory. For instance, X-bar theory requires an endocentric construction to have just one head (cf., e.g., Jackendoff 1977: chs. 2 and 3 and Sells 1985: 27–31), whereas Hockett admits Hydra-like constructions with several heads. As a consequence, syntactic coordination is classified as endocentric by Hockett but as exocentric by most X-bar theorists. Standard versions of X-bar theory require not only that each expression of category \(X^m\) possesses a unique head of category \(X^{m-1}\), they also postulate that the facultatively present additional expressions belong to categories of maximal bar-level. The scheme (X): \(X^m \rightarrow X^{m-1} (Y^m)\) of grammar \(\Theta_1\) conforms to these requirements of standard X-bar theory. Such a rule-scheme is therefore called an “X-bar scheme”. A category \(X^m\) is called a “projection” of category \(X = X^0\), and \(Y^{\text{max}}\) is the maximal projection of category \(Y\). In our simple case, max equals 1, but, of course, this need not be the case for more complex codes.

The X-bar scheme of our sample grammar \(\Theta_1\) is very simple, indeed. More complex codes such as natural languages pose more difficult problems for X-bar analysis. In this context, the question arises as to how many bar-levels are to be distinguished for a given category \(X\). In particular, how many degrees of complexity should be distinguished for verbal categories? Traditionally, there is agreement that simple verb forms constitute the basic level and that at least some languages (namely, those which are called “configurational languages”) contain a verb phrase (traditionally labeled \(VP\) consisting of a simple verb form and its objects. But what status do adverbial modifiers then have? Are they constituents of the verb phrase or do they belong to a new bar-level? Since they are traditionally viewed as modifiers they should constitute a new bar-level, but if the constituent consisting of the verb, its objects, and the adverbial modifiers is called “a verb phrase”, then we need a new label for the complex consisting only of the verb and its objects. We might use \(V^2\) for the verb, \(V^1\) for the verb plus its objects, and \(V^2\) for a constituent of category \(V^1\) plus its adverbial modifiers. However, what about sentences, whose category is traditionally labeled \(S\)? Is \(S\) a projection of \(V\) or do sentences belong to a quite different category? If category \(S\) is indeed a projection of category \(V\), is \(X = V^3\) or are there further categories intervening between \(V^1\) and \(S\) (due, e.g., to the complications of the auxiliary phrase)? The category \(S\) is analyzed as a projection of the verbal category in GPSG, as an exocentric category without a lexical head in LFG, and as a projection of a category INFL (which encodes information about the inflection of the verb) in GB (as presented in Chomsky 1982a: 18–19), which permits only \(S\)’s maximal projection (where \(\text{INFL} = \text{INFL}^0\), \(S = \text{INFL}^1\) and \(S = \text{INFL}^2\); cf. also Sells 1985: 30). Furthermore, if \(S = V^3\), are there also only three higher levels for all the other syntactic categories? Traditionally, nouns are labeled with \(N\), and noun phrases with \(NP\); thus, granting that \(S = V^3\), is \(NP = N^3\)? The hypothesis that all categories of expressions in a natural language share a unique common maximal bar-level max is known as the “uniform-level-hypothesis”. Jackendoff (1977) adopts the uniform-level-hypothesis and takes max to be 3 (starting from 0 as the lowest level); the version of GB presented in Sells (1985: ch. 2) assumes only three levels.

In order to illustrate further concepts from X-bar theory, let us now introduce a new grammar \(\Theta_1\), which describes the syntactic structure of arithmetic expressions and is an extension of \(\Theta_1\). First we enlarge the set of terminal symbols of \(\Theta_1\) by adding the new symbols: \(, (, , +, –, \cdot, :, EN\) and \(=\). The two parentheses \(\) and \(\) are called “auxiliary symbols”; they serve to indicate groupings in complex expressions. Each expression designating a natural number is called “a term”; examples of complex terms are \(\sqrt{\frac{1}{4}}\), \(\sqrt{\frac{5}{3}}\), \((3 \times 4) : 6\), etc. (In mathematical disciplines, “term” is used for any expression referring to an object of that discipline; thus, e.g., the terms of arithmetic refer to numbers. This, however, is not the basic meaning of this word which can be traced back to the Greek ὁρισμός meaning ‘frontier’, and, in the syntactic terminology of traditional logic, originally designated certain constituents in the premises and conclusions of syllogistic inferences; cf. Art. 41 for further details.) Starting with terms, we use the sign of equality \(\equiv\) to formulate sentences like \(5 = 7, \sqrt{4} = 2, (3 \times 4) : 6 = (1 + 1)\), etc. The terminal symbol \(EN\) is prefixed to a term in order to state that there exists a natural number which is the result of computing the term. For instance, \(EN\sqrt{4}\) is a true sentence, because the
(positive) square root of 4 is the number 2, but EN(3 : 6) is false since the division of 3 by 6 does not yield a natural number and EN(4 : 4) is false because 0, which would be the result of subtracting 4 from 4, is classified here as a non-negative integer but not as a natural number. The feature symbols which are employed by $\Theta^4_H$ are displayed in the following Tab. 2.4 together with their corresponding singleton symbols.

### Tab. 2.4

<table>
<thead>
<tr>
<th>feature symbol</th>
<th>corresponding singleton symbols</th>
</tr>
</thead>
<tbody>
<tr>
<td>bar</td>
<td>0, 1, 2</td>
</tr>
<tr>
<td>subcat</td>
<td>1, 2</td>
</tr>
<tr>
<td>left</td>
<td>+, −</td>
</tr>
</tbody>
</table>

We do not require any complex category symbol in $\Theta^4_H$ to contain a specification for each of the feature symbols listed in the first column of Tab. 2.4. With regard to $\Theta^4_H$ four sorts of expressions are distinguished: relations, operators, ciphers and auxiliaries. Relations are expressions which take terms in order to form sentences. Similarly, operators take terms in order to form other terms. For instance, the symbol EN is a unary (1-ary) relator and = is a binary (2-ary) relator, because two flanking terms $t_1$ and $t_2$ are required to form with it a complete sentence: $t_1 = t_2$. The feature [subcat] is used to encode the number of open slots (the “arity”) of operators and relators. Ciphers are either 0-ary operators (1, 2, …) or strings of digits of the type which we have called “endings” in $\Theta^4_H$ (e.g., 0, 1, 2, …, 9, 00, 01, …, 09, 10, 11, …, etc.). Note that the operators and ciphers intersect since the 0-ary operators $I$, $\bar{I}$, …, $9$, $\bar{9}$, … are also ciphers. But there are operators which are not ciphers (e.g., + and —) as well as ciphers like 0, 00, 01, …, etc. which are not 0-ary operators because they are not Hindu-Arabic numerals. Auxiliary symbols only include the two parentheses. In $\Theta^4_H$, we shall use the sort symbols REL, OP, CIPH and AUX in order to designate the four sorts of expressions, so that we have sort symbols in addition to feature specifications occurring in the complex category symbols of $\Theta^4_H$. We introduce the following restrictions: Category symbols containing REL or OP contain only [bar]- and [subcat]-specifications; complex category symbols with the component CIPH contain only a [bar]-specification, while symbols with the component AUX contain a [bar]- and a [left]-specification. In order to state our rules and lexical entries, we use the abbreviations given in Tab. 2.5.

Furthermore, we adopt the following convention with respect to underspecified abbreviations such as $S'$ and $Z^m$. If information must be displayed which is not provided in Tab. 2.5, we add the missing specifications to the respective abbreviation. For instance, $Z^m[subcat: 0]$ designates the category of 0-ary operators of bar-level m (cf. (LE4) below for examples), and $S'[subcat: 1]$ designates the unary relators of bar-level 1. Based on this convention, we may now list our lexical entries of $\Theta^4_H$ as in (LE4):

\begin{equation}
(LE4) \quad \begin{cases}
0 \\
1 \\
\vdots \\
9
\end{cases} \rightarrow E^0 \nonumber
\end{equation}

\begin{equation}
(LE4) \quad \begin{cases}
1 \\
2 \\
\vdots \\
9
\end{cases} \rightarrow Z^l[subcat: 0] \nonumber
\end{equation}

\begin{equation}
\sqrt{\quad} \rightarrow Z^l[subcat: 1] \nonumber
\end{equation}

\begin{equation}
+ \\
\rightarrow Z^l[subcat: 2] \nonumber
\end{equation}

\begin{equation}
\rightarrow EN \rightarrow S^0[subcat: 1] \nonumber
\end{equation}

\begin{equation}
\rightarrow S^0[subcat: 2] \nonumber
\end{equation}

\begin{equation}
( \rightarrow P, ) \nonumber
\end{equation}

\begin{equation}
\rightarrow P. \nonumber
\end{equation}

The non-lexical rules of $\Theta^4_H$ include the ones listed in (ID4):

\begin{equation}
(ID4) \quad H^l[subcat: 1] \rightarrow H^0, Z^2 \\
H^l[subcat: 2] \rightarrow H^0, Z^2, Z^2 \\
Z^2 \rightarrow P., Z^l, P_1 \\
Z^l[subcat: 1] \rightarrow Z^l, Z^l \\
Z^l[subcat: 0] \rightarrow Z^l, (E^2) \\
E^2 \rightarrow E^l, (E^2) \\
E^m \rightarrow E^{m-1}, \text{ where } m \text{ may be } 2 \text{ or } 1 \nonumber
\end{equation}

(The presence of the commas in these rules is explained below.) In the first two rules the symbol “$H$” stands for the head of an expression. More precisely the first rule states that every expression of bar-level 1 for which the
feature [subcat] takes the value 1 and which is characterized by the complex symbol $H$ consists of a head of bar-level 0 that is also characterized by $H$ and an expression of category $Z^2$. The rules of (ID4) do not accord with the uniform-level-hypothesis, but they all follow the very general scheme ($X'$):

$$(X') \ H^m \rightarrow \ldots, H^{m-1}, \ldots, \text{where } m \text{ may be } 0, 1 \text{ or } 2.$$ 

The dots “…” indicate that a number of further symbols may be present besides the one designating the head. Thus, in the first rule of (ID4) there is just one further constituent, in the second rule there are two, etc. In the grammar $\theta^4_{1H}$, a sentence such as $(2 + 2) = \sqrt{16}$ is assigned the parsing tree of Fig. 2.18.

As indicated by the commas in (ID4), the rules in question exclusively determine the hierarchical order of the constituents and are therefore called “immediate dominance rules (ID rules)”. The linear order of the constituents is left undetermined so that, e.g., the third rule of (ID4) does not prescribe that $P_+$ must precede $Z^1$ and that $Z^1$ must in turn precede $P_+$. Concerning the linearization of constituents the grammar $\theta^4_{1H}$ follows the

format of GPSG (Generalized Phrase Structure Grammar), which provides so-called “linear precedence rules” (“LP rules”; cf. Gazdar, Klein, Pullum and Sag 1985: 44–50

<table>
<thead>
<tr>
<th>Tab. 2.5</th>
<th>The complex category symbol:</th>
<th>where:</th>
<th>is abbreviated by:</th>
</tr>
</thead>
<tbody>
<tr>
<td>[REL bar: 1 subcat: n]</td>
<td>$n$ may be 1 or 2</td>
<td>$S^1$ (note that the symbol $n$ is neglected in this abbreviation)</td>
<td></td>
</tr>
<tr>
<td>[REL bar: 0 subcat: m]</td>
<td>$m$ may be 1 or 2</td>
<td>$S^0$</td>
<td></td>
</tr>
<tr>
<td>[OP bar: m subcat: n]</td>
<td>$m$ and $n$ may be 0, 1 or 2</td>
<td>$Z^m$ (note that the symbol $n$ is neglected in this abbreviation)</td>
<td></td>
</tr>
<tr>
<td>[CIPH bar: m]</td>
<td>$m$ may be 0, 1 or 2</td>
<td>$E^m$</td>
<td></td>
</tr>
<tr>
<td>[AUX bar: 2 left: x]</td>
<td>$x$ may be + or –</td>
<td>$P_x$</td>
<td></td>
</tr>
</tbody>
</table>
allows for trees with a root node labeled $S_1$.

For instance, every other sister symbol.

This amounts to the requirement for unary operators and relations that prefix notation is used, as in, e.g., $\sqrt{16}$ or $EN(1+3)$. The second rule of (LP4) postulates infix notation for binary operators and relations. The third rule states that a left parenthesis precedes each of its sister symbols while the fourth rule states that a right parenthesis is preceded by each of its sister symbols. The last rule says that the ending of a numeral follows every other sister symbol. The ID and LP rules collaborate in the construction of parsing trees. For instance, the second rule of (ID4) allows for trees with a root node labeled $S'$ which dominates three nodes, one of which is labeled $S^0$ and the other two both receiving the label $Z^2$. The second rule of (LP4) requires that the node labeled $S^0$ should occupy a position between the two other nodes.

Taken as a whole, the grammar $\mathcal{G}_{14}$ comprises the following components: (i) the non-terminal vocabulary $V^\text{NT}$ which contains all complex category symbols that may be constructed from the sort symbols $REL$, $OP$, $AUX$ and $CIPH$ and the feature symbols of Tab. 2.4 in accordance with the sort restrictions stated above, (ii) the enlarged terminal vocabulary $V^\text{t} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, EN, =, \sqrt{\cdot}, +, -, \times, :, ), ()\}$. (iii) the complex starting symbols $S'$ (note that there are two such symbols differing in the [subcat]-specification), (iv) the ID rules listed in (ID4), and (v) the LP rules listed in (LP4).

The grammar $\mathcal{G}_{14}$ allows for parsing trees which accord to the tree scheme in Fig. 2.19.

As pointed out above, a category $X^m$ is called a “projection” of the corresponding category $X^m$ (m < n) of lowest bar-level. Most versions of X-bar theory require 0 to be the lowest bar-level for each kind of category. For our arithmetic code, however, this requirement would only result in an undue complication of the grammar. Thus we have adopted 1 as the lowest level for E but 0 as the base level for $Z$; cf. Tab. 2.5 and Fig. 2.18. Furthermore, the scheme of Fig. 2.19 is to be understood in such a way that everything except the position labeled $X^0$ may be left out. This again is a deviation from some versions of X-bar theory, which postulate that every node labeled with an $X^2$ must be the origin of a path that leads via $X^i$ to $X^0$. We have not included this postulate because that would only lead to trees with long non-branching paths.

The tree scheme of Fig. 2.19 can now serve to answer questions of the type posed in the beginning of this section (§ 5.1.3.) by defining the grammatical functions of the constituents of complex expressions. Expressions of a category $Y^\text{max}$ which accompany an $X^{\text{max-1}}$ phrase and indicate that a constituent of the maximal projection $X^\text{max}$ has been completed are called “specifiers of $X$” (cf. Jackendoff 1977: 37 and Sells 1985: 28). In our arithmeti-
In any case, information about linear order need not be encoded in the subcategorization frames.

The treatment of subcategorization in $\Theta^\mathfrak{H}_1$ again follows the line taken in GPSG (cf. Gazdar, Klein, Pullum and Sag 1985: 33 ff). This approach to subcategorization is simple because the subcategorization properties are directly encoded by means of the [subcat]-specification. However, there is a certain arbitrariness in this procedure which is due to the fact that the singleton symbols which specify the values of the feature [subcat] are merely conventional code numbers. Of course, the choice of 1 and 2 in $\Theta^\mathfrak{H}_1$ is suggested by the fact that unary operators and relators subcategorize for one argument and binary operators and relators for two. But instead of 1 and 2 we could also have used such symbols as * and #. This arbitrariness becomes even more apparent when more complex codes are considered. E.g., Gazdar, Klein, Pullum and Sag (1985: 247) use the digit 8 as a code number for verbs like persuade, which subcategorize for a nominal object and an infinitival clause. In this case, the digit does not even specify the number of the argument and there is no reason for choosing 8 in this case instead of, say, 1056.

There are several other approaches to subcategorization, e.g., that of Categorial Grammar or Lexical Functional Grammar. Categorial Grammar takes subcategorization as its criterion for classifying expressions into different categories. This makes it a very appropriate basis for semantics (cf. its extensive use in Art. 4; for recent approaches to categorial grammar cf., e.g., Ades and Steedman 1982, Uszkoreit 1986, Zeevat, Klein and Calder 1987, Buszkowski, Marciszewski and van Benthem 1988, Moortgat 1988, Oehrle, Bach and Wheeler 1988, König 1990).

A very common approach to subcategorization is via grammatical relations. Traditionally, for instance, the arguments of a verb are called its “objects”. However, grammatical relations do not play an important role in syntactic theories which concentrate on constituent structure; instead, such theories try to define grammatical relations in terms of parsing tree configurations (cf., e.g., Chomsky 1965: ch. 2.2). This account of grammatical relations is in turn criticized by adherents of Relational Grammar (cf. Perlmutter 1982, Perlmutter 1983 as well as Perlmutter and Rosen 1984) and of Arc Pair Grammar, which is its formalized descendant (cf. Johnson and Postal 1980 as well as Postal 1982). A formally very elaborate theory which analyzes subcategorization in terms of grammatical relations is Lexical Functional Grammar (cf. Bresnan 1982).
5.1.4. Thematic roles

A further approach to subcategorization is based on the assumption of so-called “thematic roles” (abbreviated: “θ-roles”). This route is taken, e.g., by the influential Government and Binding Theory (GB) of Chomsky (1981); cf. § 4.10. above. The usefulness of thematic roles can again be illustrated with respect to our arithmetic code. Any adequate grammar of it must state that the operators − and : take two terms as their arguments. But, obviously, additional information is required. In subtraction it must be specified which of the two arguments is the minuend and which is the subtrahend; and in division it must be specified which of the two arguments is the dividend and which is the divisor. Relational concepts like these (minuend, subtrahend, dividend, divisor) are examples of θ-roles. We shall now explain the GB analysis of subcategorization in terms of θ-roles as well as some other notions from this theory by introducing the grammar $\Theta^5_H$ for our arithmetic code.

The list of feature symbols for $\Theta^5_H$ is an extension of that given for $\Theta^3_H$ in Tab. 2.4; cf. Tab. 2.6:

<table>
<thead>
<tr>
<th>feature symbol</th>
<th>corresponding singleton symbols</th>
</tr>
</thead>
<tbody>
<tr>
<td>bar</td>
<td>0, 1, 2</td>
</tr>
<tr>
<td>case</td>
<td>left, right</td>
</tr>
<tr>
<td>role</td>
<td>theme, radicand, summand, minuend, subtrahend, factor, dividend, divisor</td>
</tr>
<tr>
<td>zero</td>
<td>+, −</td>
</tr>
</tbody>
</table>

As in $\Theta^3_H$, we have the sort symbols REL, OP, CIPH and AUX. Complex category symbols which contain the sort symbol REL have only a [bar]-specification, those containing OP have specifications for the feature symbols [bar], [case] and [role] while those with the sort symbol AUX or CIPH only have a [bar]- and a [case]-specification. We shall explain the special role of the feature [case] only after having explained the version of X-bar theory applied in $\Theta^3_H$.

Tab. 2.7 provides a list of the abbreviatory conventions used in connection with $\Theta^3_H$.

In $\Theta^5_H$ the subcategorization properties of expressions are described by means of argument grids (cf. Sells 1985: 36 and Andrews 1988: 71–76). The argument grid of an expression $e$ consists of two lists: $\langle a_1, a_2, \ldots, a_m \rangle \langle \theta_1, \theta_2, \ldots, \theta_n \rangle$. The first list $\langle a_1, a_2, \ldots, a_m \rangle$ specifies the respective categories of the arguments for which the expression subcategorizes, and the second list $\langle \theta_1, \theta_2, \ldots, \theta_n \rangle$ provides the thematic roles which are assigned to the arguments of $e$. The second list may be longer than the first one because $e$ may have arguments for which it does not subcategorize. In GB for instance, the subject of a verb is analyzed as one of its arguments but verbs do not subcategorize for their subjects. Arguments like the subjects of verbs are called “external arguments”. There are, however, no external arguments in our code. The list (LE5) now specifies the lexical entries of $\Theta^5_H$.

$$
\text{(LE5)} \quad \begin{cases} 
0 & \leftarrow E^0 \\
1 & \leftarrow Z^0 \\
2 & \leftarrow Z^0 \\
\vdots & \leftarrow \vdots \\
9 & \leftarrow \vdots \\
\end{cases}
$$

As in $\Theta^3_H$, we have the sort symbols REL, OP, CIPH and AUX. Complex category symbols which contain the sort symbol REL have only a [bar]-specification, those containing OP have specifications for the feature symbols [bar], [case] and [role] while those with the sort symbol AUX or CIPH only have a [bar]- and a [case]-specification. We shall explain the special role of the feature [case] only after having explained the version of X-bar theory applied in $\Theta^3_H$.

Tab. 2.7 provides a list of the abbreviatory conventions used in connection with $\Theta^3_H$. 

In (X5) $X^m \rightarrow \ldots X^{m-1} \ldots$

In (X5) $X^{m-1}$ indicates the position of the head (at level $m-1$). The dots “…” may be filled with any sequence of category symbols for which the singleton symbol of the [bar]-specification designates the maximal number possible. If one compares (X5) to the rules of
Tab. 2.7

<table>
<thead>
<tr>
<th>The complex category symbol</th>
<th>where:</th>
<th>is abbreviated by:</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ REL \bar{m} ] [ bar: m ]</td>
<td>$m$ may be $0$, $1$</td>
<td>$S^m$</td>
</tr>
<tr>
<td>[ OP \bar{m} ] [ bar: m ] [ case: c ] [ role: 0 ]</td>
<td>$m$ may be $0$, $1$ or $2$, $c$ may be left or right, $\theta$ may be any of the role symbols of Tab. 2.6</td>
<td>$Z^m$ (note that the symbols $c$ and $\theta$ are neglected in this abbreviation)</td>
</tr>
<tr>
<td>[ NUM \bar{m} ] [ bar: m ] [ case: left ]</td>
<td>$m$ may be $0$, $1$ or $2$</td>
<td>$E^m$</td>
</tr>
<tr>
<td>[ AUX \bar{m} ] [ bar: m ] [ case: c ]</td>
<td>$m$ may be $0$, $1$ or $2$, $c$ may be left or right</td>
<td>$P^m$ (note that the symbol $x$ is neglected in this abbreviation)</td>
</tr>
</tbody>
</table>

$\Theta^4_{1H}$, one immediately recognizes that (X5) is highly overgenerative. This means that the scheme allows the derivation of many non well-formed expressions. For instance, (X5) admits a fragment of parsing trees such as the one given in Fig. 2.20:

![Fig. 2.20](image)

We already know from grammar $\Theta^4_{1H}$ that Fig. 2.20 cannot show the syntactic structure of any expression of our code. The left part of the tree in Fig. 2.20 is excluded by the fact that there cannot be any specifiers for terms whose head is not a binary operator. The subtree on the right in Fig. 2.20 is ruled out for the following two reasons: (i) If there are two specifiers for a term they must flank it and may not both precede it. (ii) Infix notation (instead of the prefix notation of Fig. 2.20) is obligatory for binary operators. In order to exclude at least some of the ill-formed expressions admitted by (X5), we adopt the following principles for the X-bar theory applied to $\Theta^4_{1H}$.

(S1) If in a structure [...] $[a]_{\text{\max}^{-1}} \ldots [a]_{\text{\max}}$ there are specifiers at all, then there must be two of them and they occupy the extreme left and right-hand positions in the structure.

(S2) In a structure [...] $[a]_{\text{\max}^{-1}} \ldots [a]_{\text{\max}}$, modifiers may only follow the modified element of the category designated by $\text{\max}^{-1}$.

(S3) In structures of the form [...] $[a]_{\text{\max}^{-1}} \ldots [a]_{\text{\max}}$, lexical heads always precede at least one argument, but if there are more than one, will not precede all of them.

The principle (S1) excludes structures like those in the right subtree of Fig. 2.20 but still admits for the left subtree. The second principle excludes ill-formed pseudo-terms like $0i$ but it still admits, e.g., the ill-formed $\sqrt{16} 0$, where $0$ modifies $\sqrt{16}$. Finally, (S3) excludes $11 22$ but it still admits the pseudo-term $11$.
+ 22 23, where + co-occurs with more than the appropriate number of arguments.

This last example takes us back to the problem of subcategorization. In order to eliminate pseudo-terms such as $11 + 22 23$, we adopt the following $0$-criterion, which postulates that lexical heads assign the roles of their argument grids to their arguments.

(0) The $n$-th argument (in left-to-right-order) $a$ of a lexical head $h$ must carry exactly that $0$-role which is designated by the $n$-th item in the list of roles in the argument grid of $h$.

The $0$-criterion excludes pseudo-terms like $11 + 22 23$ because the operator + can assign two but not three $0$-roles. This means that + must always be accompanied by exactly two arguments, and by extension (0) taken together with the above (S3) requires that + always occurs as an infix between exactly two arguments. Furthermore, the argument grid of + requires (by means of its first component $\langle Z^2, Z^2 \rangle$) these arguments to be terms of bar-level 2. Similarly, the $0$-criterion taken together with (S3) requires that a unary operator such as $\sqrt{}$ must always appear as a prefix of a single argument of category $Z^2$.

Pseudo-terms like $0 + 17, 17 + 0$ and $/0$ are ruled out by the following principle (A) of subcategorization.

(A) If the argument grid of an expression $h$ stipulates that $h$ takes $n$ arguments of the categories $C_1, C_2, \ldots, C_n$ respectively, then any occurrence of $h$ has to be accompanied by $n$ expressions of these categories.

Note that (A) does not fix the order of the arguments. That is (partly) done by means of (S3), and the semantic impact of the argument order is accounted for by (0). It is a peculiarity due to the simplicity of our code that only expressions of category $Z^2$ may carry $0$-roles so that one of the two lists of the argument grid is redundant in our case.

There are only two problems left in this context, and these concern the distribution of specifiers and modifiers. The distribution of these expressions is governed by the feature [case]. Clearly, a 2-ary operator requires the left parenthesis to occur to the left of its first argument and the right parenthesis to occur to the right of its second argument. Similarly, 0-ary operators require their modifiers to appear to the right of them. Thus the position of specifiers and modifiers are determined by their corresponding heads. We explain this by assuming the feature symbol [case] with the possible value symbols left and right, and say that heads assign case to their specifiers and modifiers. The case of a specifier or modifier determines the position of this expression; the left case determines the left position of the case marked expression, the right case the right position of this expression. If such an expression occurs at all, it must occur at a determinate position. But in order to occur there, it must carry a case which determines this position. It follows that a specifier or modifier cannot occur at all unless it has case. Again utilizing GB theory, we postulate the following case filter:

(C) The [case]-specifications of occurrences of symbols $P^1$ and $E^2$ in parsing trees are completely determined by the lexical heads of the expression to which these occurrences belong. Only heads which are digits or binary operators may assign case. A binary operator assigns right case (i.e., [case: right]) to the left specifier position and right case (i.e., [case: right]) to the right one. A digit assigns right case to just one following modifier position.

Trees like the left subtree in Fig. 2.20 are ruled out now because digits cannot assign case to parentheses, and $\sqrt{16}$ $0$ as well as $16 + 5 27$ are ill-formed because unary and binary operators cannot assign case to following modifiers. Furthermore, since a 0-ary operator may only assign case to the first modifier position, $17 18$ is also ruled out as ill-formed. We conclude this section by taking the sentence $(2 + 2) = \sqrt{16}$ and confronting the parsing tree provided for it in $\Theta_{\ast}^H$ (cf. Fig. 2.18) with the parsing tree assigned to it by the grammar $\Theta_{\ast}^H$ (cf. Fig. 2.21).

In Fig. 2.21, the cases and $0$-roles assigned to constituents by their heads are marked at the corresponding nodes.

5.2. Graphs, markedness, and unification

The previous section showed that feature structures and feature symbols play an important part in advanced syntactic theories. The present section is, therefore, devoted to further aspects of feature structures and feature symbols.

5.2.1. Graphs

Let us start by dealing with methods of representation for feature structures. Besides their representation by means of brackets, feature structures may also be represented by graphs, in particular by DAGs, i.e., directed acyclic
graphs (cf. Gazdar, Klein, Pullum and Sag 1985: 20–27 as well as Shieber 1986: 21–22; for the concept of a DAG cf. especially Aho and Ullman 1972: vol. 1, 39–45). As its name says, a DAG is a directed graph which has no cycles. There are several equivalent definitions of a graph, and since we shall need some concepts from graph theory in § 5.4., we shall be a little more explicit in our explanation of graphs than would otherwise be necessary at this point (cf. also the passage at the end of § 4.9.); we follow Ehrig (1979: 12–13) in matters of basic terminology (cf. also Halin 1980). A graph $\mathcal{G}$ consists of two collections: the collections $\mathcal{D}_N$ of its nodes and the collection $\mathcal{D}_A$ of its arcs (also called its “edges”). In undirected graphs, each arc $a$ from $\mathcal{D}_A$ connects two nodes $n_1$ and $n_2$ from $\mathcal{D}_N$ without assigning a direction to this link between the two nodes. Neglecting both the possibilities of arcs connecting nodes with themselves and that of several arcs between the same two nodes, we might identify $\mathcal{D}_A$ with a subcollection of the set of all classes which contain exactly two elements from $\mathcal{D}_N$. For ordered graphs, however, there exist two functions — source and target — which determine the direction of each arc. In this case, it is natural to admit “loops”, i.e., arcs $a$ with $s(a) = t(a)$, and several arcs (e.g., two $a_1 \neq a_2$ connecting the same nodes in the same (i.e., $s(a_1) = s(a_2)$ and $t(a_1) = t(a_2)$) or in the opposite direction (i.e., $s(a_1) = t(a_2)$ and $t(a_1) = s(a_2)$). Graphs with several arcs between the same nodes are sometimes called “multi-graphs”. If $a$ is an arc from $\mathcal{D}_A$ and $k_1$ and $k_2$ are nodes from $\mathcal{D}_N$, then $a$ is said to leave $k_1$ and to enter $k_2$ if $s(a) = k_1$ and $t(a) = k_2$; in this case, we say that $a$ is an arc from $k_1$ to $k_2$. A graph is finite if the number of its arcs and that of its nodes are both finite. A directed path (from $k_0$ to $k_n$) is a sequence $\langle k_0, k_1, \ldots, k_n \rangle$ such that for every two successive terms $k_m$ and $k_{m+1}$ ($0 \leq m < n$) there is an arc leaving $k_m$ and entering $k_{m+1}$; thus the shortest paths are the arcs themselves. A path $\langle k_0, k_1, \ldots, k_n \rangle$ is a cycle (or circuit), if $k_0 = k_n$; thus the shortest cycles are the loops themselves. A root is a node which (i) is not entered by any arc, but (ii) from which there is a path to every other node of the graph. A graph is called “rooted” if it has a root. (Obviously, a directed graph may have at most one root. If there were two or more, they should be connected by directed paths, but then there would be arcs entering one of them, which is impossible for roots.) Conversely, a node is a leaf if there is no arc leaving it. Two nodes $k_0$ and $k_n$ are called “connected” if there are intermediate
nodes $k_1, \ldots, k_{n-1}$ such that every two successive terms $k_m$ and $k_{m+1} \ (0 \leq m < n)$ of the sequence $\langle k_0, k_1, \ldots, k_n \rangle$ are connected by a path. (Each node is said to be connected to itself.) A graph is connected if every two of its nodes are connected. As we have already said, a DAG has no cycles. DAGs which represent feature structures are mostly considered to be finite, rooted, and connected. But there are exceptions from this as well as from the requirement that graphs representing feature structures should have no cycles (cf. Johnson 1988 and Smolka 1989). Graphs may carry annotations both on their nodes and their arcs; these are called “colors” (or “labels”). A graph with colors on its nodes and/or arcs is a colored graph. A color alphabet is a pair $C = \langle C_A, C_N \rangle$ of two sets: $C_A$ is the set of colors for arcs, $C_N$ the set of colors for nodes. Given a color alphabet, a colored graph is a graph together with an arc coloring map $m_A$ and a node coloring map $m_N$. The arc coloring map $m_A$ assigns colors from $C_A$ to (not necessarily all) arcs, as does the node coloring map $m_N$ for (again not necessarily all) nodes and $C_N$.

Given this terminology from graph theory, we may reformulate the complex feature symbol (CC) from § 5.1.2. (characterizing the Corinthian column) as a colored DAG in the following way: the DAG in question has a root from which three arcs leave. These arcs carry the feature symbols [base], [shaft], and [capital] as their respective colors. The arc which is colored with [base] centers a leaf colored with the value symbol +. The [shaft]-arc enters a node which is the source of two other arcs: The first is colored with [flute] and enters a leaf colored with the value symbol 24, while the second is colored with [taper] and enters a leaf colored with the value symbol −. The node entered by the [capital]-arc is the source of three further arcs colored by the feature symbols [volut], [decor] and [acant] and entering three leaves colored with three instances of the symbol +. Thus the complex feature symbol (CC) of § 5.1.2.:

\[
\begin{align*}
\text{base:} & + \\
\text{shaft:} & \begin{cases}
\text{flute:} & 24 \\
\text{taper:} & - 
\end{cases} \\
\text{capital:} & \begin{cases}
\text{volut:} & + \\
\text{decor:} & + \\
\text{acant:} & + 
\end{cases}
\end{align*}
\]

is equivalent with the diagram in Fig. 2.22.

![Fig. 2.22](image)

Among other things, the representation of a feature structure in a graph has the advantage of allowing for a very intuitive interpretation of a somewhat subtle distinction. For theoretical reasons, a difference should be made between two features having values of the same type and two features sharing their values. E.g., in the feature graph of Fig. 2.22 the feature symbols [volut] and [decor] both have a positive value specification. We seem to be dealing with two distinct value tokens of the same type, and yet a volute is also a form of decoration, such that we would expect [volut] and [decor] to share their value specification instead of having only value tokens of the same type. (In view of the Ionic column, which has [decor: +] but [acant: −], the feature symbol [acant] cannot be included in these considerations.) We may represent this by letting the two arcs which

![Fig. 2.23](image)
are colored with [volut] and [decor], respectively, enter the same node, which carries only one value specification + (cf. the graph in Fig. 2.23). If arcs enter the same node colored by a value specification, this is called “value sharing”.

Using the bracket notation, value sharing is indicated by coindexed boxes. Such a box is known as “a tag”. The value is written out only once, and repetition is avoided by the use of its associated tag (cf. Shieber 1986: 131). For the feature structure represented by the DAG in Fig. 2.23, this is exemplified by the complex symbol (CC'):

\[
\begin{bmatrix}
\text{base:} + \\
\text{shaft:} \\
\text{volut:} + [\text{flute:} 24] \\
\text{taper:} - \\
\text{capital:} \\
\text{decor:} [\text{acant:} +]
\end{bmatrix}
\]

Value sharing is not to be confused with a coincidence of the type of value symbols. This distinction is especially clear in the case of the graph notation, where value sharing corresponds to two arcs entering the same node while coincidence of value type corresponds to two arcs entering two different nodes which are colored with the same value symbol. Compare the graph of Fig. 2.22 (which corresponds to (CC)) and exemplifies coincidence of the type of the specifications of the feature symbols [volut] and [decor]) with the graph of Fig. 2.23 (which corresponds to (CC') and exemplifies value sharing, i.e., identity of the specifications of [volut] and [decor]). They are quite distinct graphs!

Value sharing can be used to analyze agreement phenomena. Traditional grammars for natural languages formulate agreement rules such as the one requiring the subject and predicate of a sentence to agree in number and person. Obviously, this may easily be formalized by making use of value sharing.

In the new structural description of Corinthian columns, which utilizes value sharing, the feature symbol [base] on the one hand and the feature symbols [volut] and [decor] on the other still have value specifications of the same type. (Note that [volut] and [decor] do not just agree in the type of their specifications but really do have the same specification.) We could, of course, let these merge, too, but there seem to be reasons to keep them apart just as in the case of [acant]. (For instance, one could claim that decor is exclusively a matter of the capital.) If these reasons are valid, one might nevertheless want to express the fact that the feature symbols [decor] and [base] always have the same type of value symbols.

The need to express coincidence of the type of value where there is no value sharing is even clearer in the case of the feature symbols [base], [flute] and [taper]. Here, the Boolean feature symbols must always have opposite value symbols, for, if a column has no base, then it is a Doric one and is specified positively for the feature symbol [taper]. The columns of the two other orders, however, have a base ([base: +]) but are not tapered ([taper: -]). This correspondence cannot be expressed by means of value sharing because there are no shared values. However, one might introduce functions on feature values in this case. Let \text{op} designate that function which reverses the values of Boolean features; i.e., it provides + on the basis of – and – on the basis of +. Then the correspondence between [base] and [taper] can be expressed in bracket notation as follows:

\[
\begin{bmatrix}
\text{base:} + [\text{taper:} \text{op} ([\text{taper:} - ])]
\end{bmatrix}
\]

Correspondences between feature values that are describable in this format are called “functional dependencies” in the Head-Driven Phrase Structure Grammar (HPSG) of Pollard and Sag (1987: 48). However, this solution of our problem is not quite satisfactory without additional assumptions: Why should the value specification of the feature symbol [taper] depend on that of the feature symbol [base] and not the other way round? And how should one account for the fact that the value symbol – for [base] must always be accompanied by the specification [flute: 20] whereas [base: +] co-occurs with [flute: 24]? It would, of course, be easy to introduce a symbol designating a function which maps – to 20 and + to 24; but this would be a completely \textit{ad hoc} definition since there is no intrinsic correspondence between the Boolean values and the numbers. A more adequate procedure would be the formulation of a feature co-occurrence restriction (FCR) of the type used in GPSG (cf. Gazdar, Klein, Pullum and Sag 1985: 27–29). A FCR is the statement of a relationship between the val-
ues of features within a feature structure and is formulated with the help of logical notation. In our case, we need the following FCR:

\[(\text{FCR1}) \quad [\text{base: +}] \leftrightarrow [\text{shaft: [flute: 24]}]\]

Instead of using a functional dependency in the case of \([\text{taper}]\), we could adopt the \((\text{FCR2})\):

\[(\text{FCR2}) \quad [\text{base: +}] \leftrightarrow [\text{shaft: [taper: -]}]\]

Two further biconditionals of this kind, which are also valid, are given below. \((\text{FCR3})\) is a logical consequence of \((\text{FCR1})\) and \((\text{FCR2})\) and thus may be omitted as redundant. \((\text{FCR4})\) states the above mentioned correspondence between positive value specifications for \([\text{decor}]\) and \([\text{base}]\).

\[(\text{FCR3}) \quad [\text{shaft: [taper: -]}] \leftrightarrow [\text{shaft: [flute: 24]}]\]

\[(\text{FCR4}) \quad [\text{capital: [decor: +]}] \leftrightarrow [\text{base: +}]\]

The next FCR states that \([\text{decor}]\) and \([\text{volut}]\) share their value specification.

\[(\text{FCR5}) \quad \begin{bmatrix} \text{capital: [volut: [ ]]} \\ \text{decor: [ ]} \end{bmatrix}\]

5.2.2. Markedness

There is another kind of restriction for feature structures which is motivated by the theory of markedness (cf. the contributions to Eckman, Moravcsik and Wirth 1986). Very often it is possible in linguistics to classify expressions (of various types, e.g., phonemes as well as morphemes, words, and constituents; cf. § 3.) into two groups — the group of the normal, typical, or paradigmatic items and the group of those which are remarkable, peculiar, or extraordinary in some respect. The members of the second group are then said to be “marked” for a special value of a particular feature (cf. Chomsky and Halle 1968: 405–407). Markedness theory goes back to the phonology of Trubekoy (1929, 1939), in which the marked items are said to be “merkmalhaltig”, i.e., to be specified positively for a certain feature. Given a feature formalism, this is sometimes interpreted in such a way that the unmarked value of a feature is “−”. Following this interpretation we need only state those values in the feature description of an expression which are positive because they are the features for which the expression is marked. All other features are specified negatively. However, there is often no obvious reason to give one of the values a special status with respect to markedness, and this is especially true of features with non-Boolean values. From a more general point of view, any statement about a feature structure which holds true for the normal cases is called a “feature specification default (FSD)” in GPSG (cf. Gazdar, Klein, Pullum, Sag 1985: 29–31). Let us again consider the example of columns. Two of the three types of columns have a base. Consequently, we adopt the following as a FSD:

\[(\text{FSD1}) \quad [\text{base: +}]\]

This means that we should normally find the feature specification above as a part of the feature symbol for a column. Furthermore, two of three orders do not require garlands of acanthus leaves in the capital. Thus one should expect the following as a part of a normal feature symbol:

\[(\text{FSD2}) \quad [\text{capital: [acant: -]}],\]

which implies that for the [acant]-feature the positive value is marked. Note that neither feature co-occurrence restrictions nor feature specification defaults have been newly invented by GPSG (or some other more recent approach in formal syntaxes) but had been employed by old-fashioned transformational grammar (cf. Chomsky and Halle 1968). To transformationalists FCRs are known as “redundancy rules” and FSDs as “marking conventions”. What is new, however, is that these tools of syntactic description are embedded in a coherent formal theory of features and feature structures.

Using FCRs and FSDs, we may considerably simplify our feature symbols. This is demonstrated by the three feature symbols below, which determine the column type in the three orders (as far as our analysis goes) and can therefore be used as lexical input in their syntactic descriptions:

\=[[base: -][ ] [capital: [acant: -]]\]

Doric Ionic Corinthian

The Ionic column is now described by the empty feature symbol \([\ ]\), which does not contain any feature specification! This extremely parsimonious description can be made more explicit in the following way: First, we use \((\text{FSD1})\) and \((\text{FSD2})\) to enrich \([\ ]\) with the unmarked specifications for \([\text{base}]\) and \([\text{acant}]\). This yields the following structure:

\=[[base: +][capital: [acant: -]]\]

Given this structure, we use \((\text{FCR1})\) and \((\text{FCR2})\) to add the complete value specifica-
two types as more securely as the least marked and taken together with the lexical Corinthian one. But should we not reconsider whether the Ionionic pointed out that one might well doubt similitarily in the framework of a semiotic analysis of Classical architecture (cf. Art. 44 and 69). In the present context it should only be pointed out that one might well doubt whether the Ionic column is the least marked one of all three types. Admittedly, it has a somewhat intermediate status between the simple Doric column and the richly decorated Corinthian one. But should we not regard the Doric column in its classical modesty as the least marked case and the other two types as more special forms? This question aims at a crucial problem of markedness theory: What exactly is it that renders an item marked? We motivated the choice of our FSDs by frequency considerations, but the question now is whether this is really adequate. Note also that the status of the Ionic column would presumably have turned out to be different had we taken into account the Tuscan and the Composite orders. The suggestion that the Doric column should be considered the least marked column type indicates that other criteria than just probabilistic ones might be playing a role here; potentially relevant aspects would include perceptual, aesthetic and historical considerations, but such questions cannot be answered here.

The completion of the structure for the Doric column is interesting for another reason. Since this structure must contain the specification [base: −]. (FSD1) cannot be applied here, whereas, of course, (FSD2) can and taken together with the lexical input it provides the following complex symbol:

\[
\begin{array}{c}
\text{base: −} \\
\text{capital: [acant: −]}
\end{array}
\]

Now, since the value of [base] is negative, it is not positive (i.e.: − [base: +]); so by (FCR1) we conclude − [shaft: [flute: 20]]. This leaves open two possibilities. Either, using graph terminology, there is no path [shaft: [flute]], or there is a path leading to a node which is colored by a value which differs from the value symbol 24. (The path [f1; f2; ...; fn] consists of n arcs which are colored with f1, f2, ..., fn respectively; if the last arc of this path enters a leaf colored with v we express this by "[f1; f2; ...; fn; v]".) Assuming that each of our structures has a path [shaft: [flute]] and that the only possible value symbols for [flute] are 20 and 24, we infer that the Doric structure must contain the path [shaft: [flute: 24]]. By a similar reasoning, (FC3) then completes the complex value of [shaft] by adding the specification [taper: +]. Note that again the feature co-occurrence restriction by itself does not suffice to reach the additional specifications: in addition we need the assumption that each of our structures has a [shaft: [taper]] path. The same assumption together with FCR4 adds [decor: −] to the values of [capital]. Finally, (FCR5) requires that the just added negative value symbol of [decor] is also the value symbol of [volat]. This completes our description of the Doric column, yielding the following complex category symbol (DC):

\[
\begin{array}{c}
\text{base: +} \\
\text{shaft: [flute: 20]} \\
\text{taper: +} \\
\text{capital: [volat: −]} \\
\text{[decor: −]} \\
\text{[acant: −]}
\end{array}
\]

5.2.3. Unification

Several times in building the complex structure designated by (DC) we relied upon the assumption that column structures are constructed in the same way; i.e., (i) that features defined for one structure also occur in the other structures and (ii) that there is a basic stock of possible values for each feature symbol which does not vary with the column structure in which the feature occurs. Correspondingly, all complex category symbols are built up with the same feature symbols. In other fields, however, it is sometimes natural to drop the assumption that all category symbols are similar in these ways. This leads to the use of different sorts of complex symbols which may not only differ with respect to their value symbols but also with respect to the feature symbols occurring in them. Examples for this have already been given with the grammars $\delta_1^1$ and $\delta_1^2$ in §5.1.3. and §5.1.4. There we distinguished several sorts of feature structures (cf. Pollard and Sag
Feature structures from different sorts need not contain specifications for the same feature, and, correspondingly, the complex category symbols are built up from different stocks of feature symbols for each sort. For instance, in $\mathcal{G}_4$ only structures from the sorts REL and OP contain specifications for the feature symbol [subcat], while only those of sort AUX may contain such a specification but need not do so. Sorts are marked by sort symbols in the complex feature symbols. Thus, a complex symbol which designates, e.g., a structure of sort REL contains the sort symbol REL as its part. Note that singleton symbols which designate ultimate values (i.e., values which are not complex feature structures) may be quite naturally considered as sort symbols. They designate sorts of simple structures for which no feature is defined. It seems natural to postulate that each such sort is exemplified by just one structure.

In the formal apparatus of $\mathcal{G}_4$ we distinguished four sorts of complex structures, which are designated by the symbols REL, OP, CIPH and AUX, and five sorts of atomic structures, which are designated by the singleton symbols $0, 1, 2, +$ and $-$. We may join the sorts REL, OP and CIPH to the more comprehensive sort DES of designators; designators are those expressions which have a designation (cf. Art. 3 § 2.). Furthermore, DES and AUX make up the sort EXP of expressions. If we now add a symbol $T$ for the universal sort $T$ and another symbol $\bot$ for the improper sort $\bot$, which does not contain anything, we may view the collection of sort symbols as a lattice whose structure is provided by the Hasse diagram of Fig. 2.24; cf. Art. 3 § 4.4.1. and § 7. for the concepts from lattice theory.

The ordering relation of this lattice of sort symbols is denoted by “$\leq$”; if $S_2 \leq S_1$, then the symbol $S_2$ is said to be at most as specific as the symbol $S_1$. Conversely, $S_1$ is said to subsume $S_2$ or to be at most as general as $S_2$. REL is at least as specific as EXP and therefore, since REL $\neq$ EXP, REL is more specific than EXP. The symbol $T$ is the most general and subsumes every feature symbol while $\bot$ is the most specific symbol subsumed by every other symbol. As a lattice ordering, the relation $\leq$ is reflexive, antisymmetric and transitive. The fact that $S_2 \leq S_1$ is represented in the diagram through a path from the higher positioned $S_1$ to the lower $S_2$. The meet-operation in the lattice of sorts is called “greatest common subsort symbol (gcs)”. For sort symbols $S_1$, $S_2$, we may find gcs($S_1, S_2$) by looking for the label of the highest node in the diagram which lies below the $S_1$ and $S_2$ nodes, and which may be reached from both these nodes by following the arcs of the Hasse graph. Thus, e.g., $\bot = \text{gcs(REL, OP)}$ and $REL = \text{gcs(REL, DES)}$.

In the intended model (cf. Art. 3, § 4.4.1.) of the feature formalism each sort symbol designates a sort of feature structures. We may simply identify such a sort with the set of structures exemplifying this sort. Then the universal sort $T$ designated by $T$ is the set of all feature structures, and the empty sort $\bot$ designated by $\bot$ is the empty set. Singleton symbols are interpreted by singleton (i.e., one-membered) subsets of $T$, $\leq$ corresponds to the subset relation ($\subseteq$) and the operation gcs corresponds to set intersection ($\cap$), cf.
e.g., Smolka 1988: 9f). Thus the lattice represented by Fig. 2.24 corresponds to a sublattice of the power set of all feature structures. Starting with this interpretation of the sort symbols as the most simple of our feature symbols, we may assign to each complex feature symbol a subset of T as its designation. Above we often spoke of feature symbols as characterizing a unique feature structure, but clearly, unique characterization is only a borderline case. More often a feature symbol is only a partial characterization of feature structures, and now we may say that a feature symbol designates the set of those feature structures which are partially characterized by it. In fact we already employed this view when presenting our sample grammars $\theta^1_H$ and $\theta^2_H$. The abbreviations introduced in Tab. 2.5 and Tab. 2.7 may be interpreted as standing for feature symbols which characterize classes of feature structures, e.g., instead of interpreting “$S^T$” of Tab. 2.5 as an abbreviation of a feature symbol which contains a variable (namely, the variable n), we may simply take it as an abbreviation for the following symbol (FS):

$$\text{(FS) \begin{array}{c} \text{REL} \\ \text{bar: 1} \end{array}}$$

The symbol (FS) characterizes feature structures containing [subcat: 1] as well as those containing [subcat: 2]. (Note that this re-interpretation requires a liberalization of the rules for constructing feature symbols for $\theta^1_H$, as they always require REL to co-occur with a [bar]- and a [subcat]-specification. Furthermore, the use of variables is essential for the above analysis of value sharing.)

A symbol such as (FS) is then explicated in the derivation of a sentence by means of the rules (ID4) of $\theta^2_H$. The first two rules of (ID4) require that (FS) be combined with one of the specifications [subcat: 1] or [subcat: 2] depending on whether the head of the sentence is a unary relator (namely, EN) or a binary one (namely, =). This kind of combination of feature symbols is known as “unification” (cf. Gazdar, Klein, Pullum and Sag 1985: 26f, Shieber 1986, Pollard and Sag 1987: 35–38, Smolka 1988: 38f). The unification of two feature symbols $F_1$ and $F_2$ is that symbol which designates the feature structures characterized by both $F_1$ and $F_2$. Clearly, if there is no feature structure characterized by $F_1$ and by $F_2$, the unification of these two symbols is $\perp$. Unification becomes particularly clear if we use graph notation. Consider the two graphs $D^1_i$ and $D^2_i$ below:

$$D^1_i \begin{array}{c} \text{base} \\ \text{capital} \end{array}$$

$$D^2_i \begin{array}{c} \text{shaft} \\ \text{capital} \end{array}$$

The two graphs $D^1_i$ and $D^2_i$ may be combined by making their common parts coincide. The resulting graph $D_i$ is their unification. As the reader will immediately verify, $D_i$ is exactly the (graph form of the) Ionic column’s feature symbol (IC).

$$\text{(IC) \begin{array}{c} \text{base}: + \\ \text{shaft}: \text{flute: 24} \\ \text{taper: } - \\ \text{capital}: \text{volut: } [\square] \\ \text{decor: } [\square] \\ \text{acant: } + \end{array}}$$

Non-trivial unification is only possible when the features which are to be combined are compatible, i.e., common paths must have identical continuations (or lead to identically colored leaves). As stated above, the unification of incompatible symbols yields $\perp$ as its trivial result.

Historically, Kay’s Functional Unification Grammar (FUG) seems to have used unification in an extensive and systematic manner for the first time (cf., e.g., the exposition in Kay 1985, where reference is given to Kay’s earlier work, as well as Bresnan and Kaplan 1982). As might be evident from the examples discussed, unification is a very powerful tool of syntactic description (and its use is by no means confined to syntax). Some syntac-
tic structures, however, seem to require even more efficient tools. Consequently, one of the main endeavors in the theory of syntactic features is to enrich the lattice of feature structures by additional operations. One such operation which is of special importance in order to express negative information is complementation, which indicates that a feature does not have a value of a certain kind (cf. Mosher and Rounds 1987, Johnson 1988, Smolka 1988, 1989). Several further tools are added to the usual feature formalism by HPSG (cf. Pollard and Sag 1987: ch. 2). This syntactic theory is also noteworthy for its application of the enriched feature formalism. All aspects of a linguistic unit — those which relate to its form as well as those which pertain to its content — are described in one and the same feature structure. In doing so, Pollard and Sag refer to Saussure (1916) and accordingly call the sort of feature structures ascribed to linguistic units “SIGN”.

We conclude this section with the remark that the theory of syntactic features has made considerable progress through its elaboration in the framework of logic. In contrast to earlier approaches, feature structures are strictly distinguished from their semiotic representation. Formal languages are developed for the description of feature structures, and these languages are then interpreted in the way described in Art. 3 § 4. so that feature structures (or sets of such structures) become their models. This way of viewing the matter is due especially to the work of Johnson (1988) and Smolka (1988 and 1989).

5.3. The direct formation of structure trees

Given a complex expression of a natural language, most linguists assume two kinds of syntactic structure (cf. § 3. above). Syntactic structure is based on the contiguity relations between the various items of the complex expression. Paradigmatic structure is based on the equivalence relations between each item of the complex expression and other items that could take its place within this expression without changing its well-formedness. A third kind of structure can be defined by constructing paradigms of syntagms for the items of the complex expression under investigation. These form a hierarchical structure which groups the items of the complex expression into larger constituents; it is called “constituent structure” (for classical treatments of this concept cf. Bloomfield 1933 and Wells 1947).

Each constituent structure can be depicted in a tree diagram which represents the relationship between the constituents. Using the graph-theoretic terminology introduced in § 5.1.3., we can define a tree as a directed rooted graph, each node of which is entered by not more than one arc. Seen in this way, constituent structure has been studied in a rather indirect manner in traditional linguistics: Languages are considered as string codes and analyzed by means of string production grammars (cf. § 4.2. – 4.7.). Each string production grammar $G$ aims at characterizing the well-formed strings of a language $L$ as a subset of all possible linear arrangements of syntactic items. This is achieved via the derivations of $G$: all and only the well-formed strings of $L$ over the terminal vocabulary $V$ should be derivable in $G$ (cf. § 4.4.). The constituent structures of these strings are constructed only as by-products of their derivations. In order to assign a string $l$ its constituent structure one projects onto $l$ its derivational history in $G$ (cf. § 4.9. for details).

If constituency is a central notion in syntactics, it should be possible to study it in a more direct way without tying it to derivational history and syntagmatic contiguity. This is why most recent theories in formal syntactics approach hierarchical structure separately from linear order (cf., e.g., the distinction between immediate dominance rules — ID-rules — and linear precedence rules — LP-rules in GPSG, see § 5.1.3.). Let us now introduce some technical means to provide an even more direct access to hierarchical structure. Note that tree structures are by no means limited to language codes. As examples for tree structures in music consider the diagram by Wankmüller (1987: 583) which (partially) represents the structure of a classical sonata (cf. Fig. 2.25) and the diagram by Lerdal and Jackendoff (1983: 144) which specifies the time-span reduction for Johann Sebastian Bach’s chorale O Haupt voll Blut und Wunden (cf. Fig. 2.26).

The derivations in a string production grammar proceed by means of rewrite rules. In the case of context-sensitive grammars they take the form $A \rightarrow \omega / a_1 \ldots a_n$, where $A$ is an item of the non-terminal vocabulary $V_N$, $\omega$ is a non-empty string of terminal and/ or non-terminal symbols (i.e., $\omega \in \Sigma (V_N \cup V_T)$), and $a_1, a_2, \ldots$ are possibly empty strings of the same kind (i.e., $a_1, a_2 \in \Sigma (V_N \cup V_T)$); cf. § 4.7. Context-free grammars (cf. § 4.6.) only admit rules where $a_i$ and $a_2$ are empty.
Now instead of using such rewrite rules in order to form strings which are then assigned constituent structures, one could utilize them directly as norms regarding the well-formedness of trees. These rules then play the role of so-called “node admissibility conditions”. This notion goes back to McCawley (1967), who considers the base component of a transformational grammar to be a set of colored trees instead of a rule system generating such trees. The members of this set, however, have to fulfill certain conditions which can be stated by means of the former rewrite rules. Peters and Ritchie (1969) elaborated McCawley’s idea, and it was later incorporated into GPSG (cf. Gazdar, Klein, Pullum and Sag 1985: 99–105). The results of Peters and Ritchie have been generalized in joint work by Joshi and Levy (1982), whose work is presented in what follows.

We first explain the notion of a proper analysis of a colored tree δ. Intuitively, a proper analysis is just a cut through a tree which crosses all of its paths (cf. § 4.10.). If k is a node of δ, then every proper analysis of δ cuts a path on which k lies. (Note that normally there are several paths that go through k.) Formally, this notion is defined by induction in the following way: (1) If δ is the empty tree θ, then the empty set Ø is the set P(θ) of all proper analyses of θ. (2) If δ is the tree [δ₁,…,δₙ]₄ (where A is a non-terminal symbol), then P(δ) = {A} ∪ P(δ₁)*…*P(δₙ) (where M*N is the set of strings whose heads come from M and whose tails come from N). As an example consider Fig. 2.27, which represents a parsing tree that δ₄₈ (cf. § 5.1.3.) assigns to the term √16 (this is the right subtree of the derivation tree shown in Fig. 2.18).

The dotted lines in Fig. 2.27 depict three of the nineteen possible proper analyses: Z₁Z₂, √Z₁E₂, and √16. The nineteen proper analyses of the parsing tree of Fig. 2.27 are: (Z₁, Z₂Z₂, Z₁Z₂E₂, Z₁Z₄E₁, Z₁Z₄E₂, Z₂Z₆, Z₁E₂, Z₁E₁, Z₁E₀, Z₁₁₆, √Z₁, √Z₁E₂, √Z₁E₁, √Z₁E₀, √1E₂, √1E₁, √1E₀, √1₁₁₆).

Given a colored tree δ, a node of δ colored with A, and a context-sensitive rule A → α₁ A α₂, we then say that “the rule holds for the node” if (1) the nodes immediately dominated by the node are each colored with a symbol from the string α and (2) there is a proper analysis β₁ A β₂ in P(δ) for β₁, β₂ ∈ Σ(VNT U VT). The context α₁ α₂ of the rule is now called “a proper analysis predicate”. In the tree of Fig. 2.27, for instance, the rule Z₁ → Z₁E₂ / Z₁₁₆ holds for the right daughter of the root. One might think that a language which is determined by trees satisfying proper analysis predicates must be context-sensitive because the rules employed in analyzing trees are context-sensitive. Surprisingly, this is not the case. Such a language is still context-free. We may even allow Boolean combinations of proper analysis predicates and include so-called “domination predicates” without leaving the realm of the context-free (cf. Joshi and Levy 1982: 4, Theorem 3.1). A domination predicate DOM(γ₁, γ₂) may be viewed as the vertical analogue of a proper analysis predicate. It holds for a node colored with A if there is a path from the root through A to the terminal string which has the form β₁ γ₁ A γ₂ β₂ (where β₁ and β₂ may be void).

Node admissibility conditions are rules for checking trees, and as such they presuppose that trees are available which can be checked.
Fig. 2.26

We may ask how these trees are to be constructed. One answer to this question is given by means of tree adjunct (or tree adjoining) grammars (TAGs), which were developed by Joshi and his colleagues (cf. Joshi and Levy 1975, Joshi 1985 as well as Weir 1988). Similar work has been done in computer science, and this proved to be a source of inspiration for Joshi (cf. Thatcher 1967, Brainerd 1969, Rounds 1970 and Dilger 1982). TAGs originate with Joshi’s attempt to formalize the string analysis and transformational theory of Harris (cf. Harris 1962 and 1970). Harris started with a small collection of very primitive sentence forms, the so-called “center strings”, into which adjunct strings may be inserted at certain adjunction points. Since the adjunct strings themselves may have adjunction points, the insertion may be repeated in a recursive manner. Obviously, this kind of syntactic analysis concentrates exclusively on what may be called “head-modifier relationships” (in view of § 5.1.3.). The relationship between head and argument is either an internal matter of the center string and the adjunct strings or is neglected. If a complex
string (e.g., a sentence) is embedded as an argument into another string, the matrix string, one would not consider the whole construction as a simple center string. Furthermore, the relationships between atomic components of the matrix string and those of the embedded string are of quite a different kind to those between items within the matrix string or those between items of the embedded string. Conversely, a complex argument cannot be regarded as an adjunct of any element of the matrix string into which it is embedded. Thus grammatical theories such as string analysis which concentrate on adjunction have problems with certain kinds of arguments. Traditional phrase structure approaches, on the other hand, (including the Standard Theory of Transformational Grammar with its base component) may deal with argument structure easily but have difficulties with adjuncts. These frameworks cannot, for example, distinguish between a relative clause (adjunct to a head noun) and a sentential object. Joshi tries to combine the merits of string analysis and phrase structure grammar in the notion of a “Mixed Adjunct Grammar (MAG)” which allows for both the adjunction operation of string analysis and the replacement rules of phrase structure grammar. MAGs could even be refined by adding transformation rules of various kinds; cf. Joshi (1972) and Sager (1981) for an application of such a framework.

In admitting phrase structure rules besides adjunction, Joshi assigns a vertical structure to the initially merely horizontal strings. It was therefore only consistent of him to generalize the domain of the adjunction operation from strings to trees (cf. Joshi, Levy and Takahashi 1975). This generalization allowed a unified treatment of adjunction and replacement operations and made the use of transformations superfluous. A tree adjunct grammar, then, is a pair \( \langle I, A \rangle \) consisting of a set \( I \) of initial trees (Joshi 1985: 211)—in Joshi, Levy and Takahashi (1975: 139) they are still called “center trees”—and a set \( A \) of auxiliary trees (Joshi 1985: 211); these are the “adjunct trees” of Joshi, Levy and Takahashi (1975: 139). An initial tree has its leaves colored with terminal symbols only; its root is colored with the start symbol. An auxiliary tree also has its leaves colored with terminal symbols, with one distinguished exception: the so-called “foot node”, which has the same color as its root. Thus an auxiliary tree looks like \( \delta_0 \) below.

\[
(\delta_0)
\]

Now let \( \delta_1 \) be a tree with a node \( n \) which is colored with \( X \) and let \( \delta_2 \) be the subtree of \( \delta_1 \) whose root is \( n \). Then \( \delta_0 \) is adjoined to \( \delta_1 \) at node \( n \) by ways of excising \( \delta_2 \), then inserting \( \delta_0 \) at the node \( n \), and finally placing the excised \( \delta_2 \) at the foot node of \( \delta_0 \).

Let us illustrate the procedure of a TAG with the example of our arithmetical code (cf. § 5). The tree adjunct grammar \( \Theta^6_{\text{H}} = \langle I^6_{\text{H}}, A^6_{\text{H}} \rangle \) is based on the same terminal vocabulary as our grammar \( \Theta^6_{\text{H}} \) (cf. § 5.1.3.), but as non-terminal symbols we replace the complex symbols of this grammar with \( S \) (sentences), \( Z \) (terms), \( F \) (operators), \( R \) (relators), \( P \) (parentheses), \( N \) and \( M \). The symbols \( N \) and \( M \) are used in the same sense as in the grammar \( G^1_{\text{H}} \); cf. § 4.4. The set \( I^6_{\text{H}} \) of initial trees of our TAG \( \Theta^6_{\text{H}} \) comprises the trees of the forms shown in Fig. 2.28, where \( d_1 \) and \( d \) are digits of category \( N \) (cf. § 4.4.).

In addition, there are three kinds of auxiliary trees in \( A^6_{\text{H}} \). Trees of the first kind have foot nodes \( N \) as shown in Fig. 2.29, where \( d_1 \) is now a digit from category \( M \).

Trees of the other two kinds have foot nodes \( Z \), as in Fig. 2.30, where in the right diagram \( \bigcirc \) is a binary relator (i.e., \( +, -, \times, \ldots \)). The sentence \( (2 + 2) = \sqrt{16} \), whose parsing trees in \( \Theta^6_{\text{H}} \) and \( \Theta^6_{\text{H}} \) are provided by Fig. 2.18 and Fig. 2.12, is now reached by
three adjunctions as shown in Fig. 2.31—Fig. 2.33. In these diagrams we use \( \oplus \) as adjunction operator and mark the node where the adjunction takes place by a box.

In a TAG there is no tree-building operation other than adjunction. As already pointed out, the application of this operation presupposes that there are already trees with some degree of complexity to be combined. Therefore the question arises whether one can give a still more direct account of trees which neither restricts the range of tree operations nor presupposes more than the most primitive trees consisting only of roots. Before we turn to such a direct syntactic construction of trees, let us again have a look at the syntagmatic contingency relation. The theory of concatenation was formally developed by Hermes in his classical booklet of 1938. However, Hermes did not start with concatenation but took a whole family of operations as basic instead: one for each atomic symbol of the code under investigation. This procedure is exemplified in the analysis of Roman numerals in Art. 3 § 2. Corresponding, for instance, to the operation \( J_I \) of adjoining \( I \) to the right of a numeral string; corresponding to \( V \) there is the operation \( J_V \) of right- adjoining this digit, etc. Each Roman numeral can be constructed by means of these operations, starting from the empty word \( \Lambda \). E.g., \( IV \) is \( J_I(J_I(\Lambda)) \) and \( VII \) is \( J_I(J_I(J_V(\Lambda))) \). Concatenation is shown by Hermes (1938: 14–15) to be definable on the basis of such operations which adjoin just one syntactic atom.

At first sight, this procedure seems to be a rather silly way of characterizing the class of strings that can be formed over a set of basic symbols. Nevertheless, there are several advantages connected with it. One is that it emphasizes the parallel between the semiotic theory of strings and arithmetics. Another advantage is that, in contrast to concatenation, the adjoining operations admit a unique syntactic analysis of strings, which is crucial for the procedure of semantic interpretation outlined in Art. 3 § 2. The empty word \( \Lambda \) plays a role similar to the number nought in arithmetic. But in contrast to arithmetic, where we have just one successor operation (which yields the successor \( m + 1 \) from its predecessor \( m \)), there are several such operations in the semiotic theory of syntagmatic order – namely the operations which adjoin the basic symbols. Thus Hermes’ procedure reveals that this theory is a kind of generalized arithmetic (cf. Hermes 1938: 9). Ordinary arithmetic is that special case of its semiotic counterpart in which there is just one successor operation (cf. also Quine 1946).

Now let us go a step further in generalizing. There are just one minimal element and one unary successor operation in ordinary arithmetic and there are several minimal elements and several unary successor operations in the theory of syntagmatic order. What about having several starting elements and several \( n \)-ary \((n \geq 1)\) operations? This generalization provides us with the notions of a ranked alphabet and a term algebra (Mahn 1965: § 1, Engelberg 1980: 16–20). A ranked alphabet assigns to each of its items a non-negative number \((0, 1, \ldots)\) as its rank. Simple expressions of rank 0 are the terminal elements or basic terms. They are the “noughts” of our generalized arithmetic or – using tree terminology – the trees which only consist of
a root (without any arcs). A non-terminal $A$ of rank $r_A$ gives rise to an $r_A$-ary successor operation $N_{r_A}$ which assigns to $r_A$ terms $y_1$, ..., $y_{r_A}$ the complex term $(A(y_1, ..., y_{r_A}))$ as their $A$-successor. This may be depicted as in Fig. 2.34.

A tree being a term and a terminal element being a basic term, a tree behaves just like
an ordinary terminal element in view of the successor functions. We therefore have the possibility of constructing trees of any complexity (height and branching structure) from primitive trees of height one (which only consist of a root).

In order to illustrate these considerations with an example, we assign the following ranks to the symbols we used above in $\Theta^\rho_H$ for the analysis of our arithmetical code: (1) The terminal elements $0, 1, \ldots, 9, \sqrt{,}, \sqrt{,}, +, -, \times, =,$ and $EN$ are of rank 0. (2) $M, N, F, R,$ and $P$ are of rank 1. (3) Instead of the single symbol $S,$ we now have two symbols $S_2$ and $S_3$ of ranks 2 and 3 respectively. Similarly instead of the single $Z$ we have now $Z_1, Z_3$ and $Z_5$ of the ranks indicated by their subscripts. This classification suffices to construct the following seven trees of height 2.

$\begin{array}{cccccc}
N & N & F & R & P & P \\
1 & 2 & + & = & ( & ) \\
\end{array}$

Taking into account the classification of $Z_1, Z_3$ and $S_3,$ we may use these trees to successively construct the tree of Fig. 2.36, which is of height 5.

Obviously, assigning ranks to terminal and non-terminal elements in the above way is not sufficient for the characterization of the well-formed trees of our arithmetical code. Since, for instance, $N$ is simply said to be of rank 1, a tree consisting of a root colored with $N$ which dominates just one leaf with the color $\sqrt{,}$ is not yet excluded as ill-formed so far. In order to achieve that, we have to define special functions mapping trees onto yet other trees. Let, e.g., $\varphi_N$ be a function which maps an argument tree consisting only of a root colored with either $1, 2, \ldots,$ or 9 to the tree consisting of a root colored with $N$ and having the single colored node of the argument tree as its sole daughter. (For definiteness, we might postulate $\varphi_N$ to assign the empty tree $\emptyset$ to other arguments.) Then the well-formed trees with roots colored with $N$ are just the values of the function $\varphi_N$. The other non-terminal elements may also be treated in this way. At first glance, the need for functions such as $\varphi_N$ may look like a further unwelcome complication, but note that it makes possible a very direct approach to the question of syntactic complexity.

We have seen that the unrestricted construction of trees involves a generalization of arithmetic. Before discussing syntactic complexity, let us therefore have a look at the complexity of computational tasks in arithmetic. There is one basic distinction within the realm of such tasks: some tasks are easy in the sense that they may be executed by means of a (sometimes awkward but nevertheless mechanical) procedure which yields a definite solution. A simple example of such a task is the addition of two numbers, another more complicated example is the task of computing all the prime factors of a given number (cf. Art. 26). The procedures which are used in solving such (easy) tasks are known as algorithms (cf. Rogers 1967: 1–5, Hermes 1961: ch. 1 and Davis 1958: § 1). On the other hand, there are tasks for which not only no algorithm is known but which can be proved to be unsolvable in an algorithmic manner. Examples of such problems are naturally rather complicated and are not given here for this reason; the interested reader should consult the books cited above as well as Art. 78. Now, algorithms have been described above only as mechanical procedures for the answering of questions within a finite timespan. Admittedly, this is not a strict defi-
nition but only a rough description. Although the notion of an algorithm has been known for a very long time, formal explanations for it were not formulated until the 1930s and 1940s. One such explanation is provided by recursive function theory, which grew out of the metamathematical work of Herbrand (1931) and Gödel (1931). Roughly speaking (and thus grossly oversimplifying the matter), a recursive function is a function whose value for a given argument can be computed by means of an algorithm. This explains recursiveness in terms of algorithms; but Herbrand and Gödel demonstrated how to characterize recursive functions in a purely formal manner without recourse to any notion of algorithm. Thus the special arithmetic concept of a recursive function provides us also with a notion of an algorithm for other areas if we can translate the problems of these areas into arithmetic questions via coding. Such codings are known as “Gödel numberings” or “Gödelizations” (cf. Hermes 1961: 4 and Davis 1958: 56–59).

The expressions of our arithmetic code may, for instance, be gõdelized by recording them in the following way. First, we assign numbers to the basic symbols as shown in Tab. 2.8:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>√</td>
<td>+</td>
</tr>
<tr>
<td>−</td>
<td>×</td>
</tr>
<tr>
<td>EN</td>
<td>=</td>
</tr>
</tbody>
</table>

Then a string \(s = z_1z_2 \ldots z_m\) consisting of \(m\) basic expressions \(z_1, z_2, \ldots, z_m\) with the respective code numbers \(C(z_1), C(z_2), \ldots, C(z_m)\) is coded as the product \(C(z) = p_1^{C(z_1)} p_2^{C(z_2)} \ldots p_m^{C(z_m)}\), where \(p_1, p_2, \ldots, p_m\) are the first \(m\) prime numbers. For instance, the expression \(2 + 7\) is now coded as \(2^{17} \cdot 3^2 \cdot 5^{11} \cdot 7^7 \cdot 11^{18}\). On this basis semiotic tasks relating to our arithmetic code can be treated as special arithmetic tasks. The task of constructing the term \(\left(e_1 + e_2\right)\) is a very simple example of this. We might say that there is an algorithm for solving this task, which can be described with the following list of instructions: (1) Write down a left bracket. (2) Then write down the first input expression \(e_1\). (3) Then add the sign +. (4) Write down the second input expression \(e_2\). (5) Finish by adding a right bracket. Given Herbrand and Gödel’s explication of the concept of an algorithm, we may now ask whether the binary arithmetic function \(f\) which corresponds to this procedure is recursive.

Now the primitive recursive functions are those functions which can be defined in terms of the successor function and certain very simple functions of a combinatorial character (such as the functions assigning constant values to their values), as well as by functional composition and a certain definitional scheme which is called “primitive recursion”; (cf. Rose (1984): 10). Since tree formation is a generalization of arithmetic, we may also generalize the notions of primitive recursiveness and of degrees of sub recursiveness to take them from number arithmetic to tree arithmetic. This enables us to deal very directly with matters of complexity of algorithms for manipulating trees without the detour of Gödelization (cf. the work by Engelsberg 1980). That the notions reached by generalization from numbers to trees indeed coincide with those reached by the indirect method of Gödelization is demonstrated by Mahn (1965).

Given the ease and velocity with which sign users manipulate complex expressions, it seems to be a reasonable hypothesis that the syntactic operations employed are at most recursive. In particular, it is almost generally assumed that the class of well-formed expressions of a code (whether strings or trees or other configurations) is a decidable subclass of the set of all possible expressions of that code (i.e., all expressions that can be formed with the basic syntactic operations without taking care of the syntactic rules). A set is
decidable if there is an algorithm for testing membership in that set. Given the concept of a recursive function, we can define a decidable set to be one with a recursive characteristic function. (The characteristic function of a set assigns the value 1 to its members and 0 to its non-members.) The most noteworthy exception to the general agreement that the well-formedness of expressions is a decidable question is Hintikka (1975), who claims in his famous any-thesis that well-formedness is undecidable for natural languages such as English (but cf. also Hüt 1968).

There is another approach to study questions of complexity in formal syntactics which uses the concepts of automata theory instead of those from recursive function theory. This approach is based on Turing’s (1936) explication of the notion of an algorithm by means of Turing machines. The basic results of this approach have already been presented in § 4.3. above. Turing-computable functions (i.e., functions which may be computed by a Turing machine) are recursive and vice versa; thus Turing’s explication of the notion of an algorithm coincides with that of Herbrand and Gödel (cf. the proofs in Rogers 1967, Hermes 1961, Davis 1958).—Let us conclude this section with the remark that in automata theory a generalization from string processing to tree processing machines has been developed which parallels the above described generalization from number to tree arithmetic (cf. Levy 1972, Thatcher 1973).

5.4. Multi-dimensional sign systems and graph grammars

String and tree structures are useful tools in the linguistic analysis of natural language expressions but they certainly do not suffice if we study the signs of other codes. Non-linguistic signs often have structures which obviously do not belong to the linear types of strings or the branching types of trees. Even in linguistics the structure trees of certain expression are sometimes adorned with additional arcs in order to point out special relationships between the nodes they link (cf. Karttunen 1981 and Joshi 1985: § 2). Strictly speaking, trees with additional links are not trees anymore but graphs of a more general kind.

Non-linear and non-tree-like syntactic structures occur, e.g., in drawings, pictograms, pictures, maps and scores, in the spatial and tactile patterns of perception as well as in the morphological patterns of biology and in the fixed action patterns (FAP) of ethology. Various disciplines have developed attempts to deal with these phenomena and to give a satisfactory description and explanation of their syntactic aspects, and such attempts have often been generalized beyond the border of the respective discipline. Thus there have been projects for a general morphology and theory of morphogenesis (which can be traced back to classical biological works as that of Haeckel 1866 and Thompson 1917 = 1961), or for a general gestalt theory (which originates with gestalt psychology; cf. Köhler 1920). Comparable approaches in more recent times are catastrophetheory as developed by Thom and Peti-tot-Cocorda (cf. Thom 1972 and 1980, Peti-tot-Cocorda 1985 a, b, and Wildgen 1982), chaos theory and fractal geometry as developed by Mandelbrot (cf. Mandelbrot 1982, Edgar 1990, Falconer 1990 and Zeitler and Neidhardt 1993), and the theory of synergetics of Haken (cf. Haken and Haken-Krell 1989 for a general introduction and a biological application). Some of these approaches are dealt with in Chapter XIII of this Handbook (cf. especially Art. 127 – 131). An ambitious and mathematically sophisticated pattern theory has been developed by Ulf Grenander and his group at Brown University (Providence RI; cf. Grenander 1976 – 1981 and Grenander 1994). Furthermore, the need to construct automata which are able to react to sensory input patterns and to process pictorial information has led to a special branch of computer science which deals with patterns and pattern recognition (for the cases of vision and spatial patterns cf. the well-known 1982 book by Marr and his survey article from the same year as well as Alexandrov and Gorsky 1993; for a more general account cf. Niemann 1983).

Since we have a rather good understanding of string structure, there are two obvious strategies in dealing with more complex syntactic patterns: reduction and generalization (cf. § 4.1. above). An instance of the reductive strategy is recoding. In analyzing a pattern which is not string-like, we may try to recode it by means of strings. This approach is exemplified by the analysis of line drawings given in Suppes and Rottmayer (1974). The data analyzed by Suppes and Rottmayer (1974: 350) consist of simple drawings like the one in Fig. 2.37.

A line segment in a drawing is recoded as a sequence of the names of points lying on it.
At least the first and last points of a line segment must be indicated. Thus the long horizontal segment in the drawing of Fig. 2.37 is coded by the name string \( ABCDE \). The complete drawing is represented by a string of such name strings where neighboring name strings are separated by a comma. The drawing in Fig. 2.37 can be represented by \( ABCDE, AG, BF, GCH, HD \) (as well as by, e.g., \( AG, GCH, HD, ABCDE, BF \)). Of course, not every string of capital letters and commas is a recoding of a possible drawing; some are only combinatorial artifacts which represent geometrically impossible constellations (or: only pretend to recode a drawing). Suppes and Rottmayer (1974: 351) therefore provide three additional criteria for a name string to be a proper recoding of a line drawing.

On the basis of this approach, we may now use geometric properties to characterize a class of the line drawings in which we are specially interested. These we consider as the class of well-formed elements of our pictorial code. For instance, we may be interested in those drawings in which it is always possible to reach a certain segment from a given one by traveling along segments of the drawings. Such drawings are called “connected”. The drawing in Fig. 2.37 is an example for connectedness. The simplest example of a non-connected drawing consists of two segments without a shared point (encoded, e.g., by \( AB, CD \)). A representation is called “connected” if the drawing represented by it is connected. We may now ask for an automaton which accepts the connected representations as well-formed and rejects the non-connected ones as ill-formed. Suppes and Rottmayer (1974: 351) state that the connected representations are Turing-computable and this implies that there is a string production grammar for them. Thus the recoding system of connected line drawings is describable by means of a string production grammar. The same is true, e.g., for the system of polygons (cf. Suppes and Rottmayer 1974: 352).

Nevertheless this method of analysis is severely restricted. One of its limitations is that it cannot distinguish convex from concave line drawings (cf. Fig. 2.38). This is simply due to the fact that the relevant information is lost in coding; thus, e.g., the two drawings in Fig. 2.38 have the same encoding \( AB, BC, CD, DE, EA \) (cf. Suppes and Rottmayer 1974: 352).

This is an example of a crucial disadvantage of all recoding-approaches: There may always be properties lost in recoding. To make up for this loss in our example, we might insert numerals into the name strings in order to indicate angles of different degrees. But if we really want to apply an approach like the one sketched, we should clarify in advance which structural properties of the patterns to be studied are so essential that they should be preserved in the recoding. This presupposes that we have intuitions about the syntactic properties of these patterns. It therefore seems appropriate to look for a direct account of these intuitions.

In the strategy of generalizing the syntactic concepts of string theory to the non-linear case, one obvious option is to assume more than one type of syntagmatic relation within a complex sign. Line drawings, for example, have not only a horizontal dimension but also a vertical one. This kind of approach is used in early work on pattern recognition, such as in Eden’s approach to handwriting and in Narasimhan’s syntactic theory of pictures (cf. Eden 1961 and 1962 as well as Narasimhan 1964 and 1966). A syntactic analysis of the print letters of standard German which follows this idea is given by Althaus (1973 = 1980). The syntactic items are called “GDMs” (“graphisch distinktive Merkmale”, “graphically distinctive features”) by Althaus; they are given in the following sequence: \(|, \), \(\sim\), \(-\), \(\O\), \(\), \(|, \), \(\), \(\), \(\). This sequence of GDMs is numbered from 1 (for \(\) ) to 12 (for \(\)). Each GDM may be placed within one of seven numbered cells of different size, which are called “Schreibräume” (“writing spaces”) by Althaus and which subdivide the line on which letters are placed along the vertical axis. These cells are shown in Fig. 2.39.

Let us call a GDM located in one of the seven cells “a figure”. Obviously, figures invite feature descriptions (cf. §4.2.). Each figure can be described by means of a feature symbol specifying its GDM and another one stating the cell. As feature descriptions Althaus uses numerals with exponents \(n^m\) where
2. Syntactics

$n$ specifies the GDM and $m$ the cell. The letter “F”, for example, is composed of the three figures given in Fig. 2.40:

We cannot simply concatenate the GDMs, however, since this would result in the following configuration:

So besides the syntagmatic relation of linear precedence, which Althaus denotes by an arrow “−”, he introduces as a second syntagmatic relation that of vertical precedence, which is denoted by “[”. Now the letter “F” is described by the formula “[1 |= 4 1 ↑ 4”.

As is evident from the work of Shaw (1969), this rather structuralist analysis may easily be extended to a string production grammar approach (cf. also Suppes and Rottmayer 1974: 354–356). Let us first redefine the relation “⇒G” of immediate derivability in a string production grammar G (we suppress the subscript “G” in the following; cf. § 4.4.)! The old definition explains the derivability relation ⇒ in terms of substitution into linear structures built up by concatenation. Now we admit structures which may be ordered with respect to several dimensions. Of course, rewrite rules must also be generalized in this way. If we have a rule $X → YZ$, which admits replacing an occurrence of $X$ by the concatenation of $Y$ and $Z$, we now have to state what type of combination of $Y$ and $Z$ (concatenation or another mode of combination) is allowed to replace occurrences of $X$. Shaw (1969) describes letter shapes and uses graphical items which are similar to the GDMs of Althaus (1973 = 1980), but unlike Althaus he assigns a direction to his graphical items, e.g., the vertical stroke is directed upwards, its bottom point is its tail and its top point its head; the tail of the horizontal stroke is its leftmost point and the head its rightmost one, as in

| ! and ⇒ |
The use of directions enables Shaw to do away with cells. Instead of simple concatenation, we now have several operations which are defined in terms of direction (tail, head) and the operation of identifying points of graphic elements: we shall designate the latter operation by “⊗”. Then we may define the operations + by \( g_1 + g_2 = \text{head}(g_1) \otimes \text{tail}(g_2) \) and × by \( g_1 \times g_2 = \text{tail}(g_1) \otimes \text{head}(g_2) \). Combinations of atomic elements inherit their heads and tails from their components. For instance, we have \( \Gamma = | + − \), where the tail of \(| \), which is the bottom point of \( \Gamma \), is the tail of the complex figure, and the head of \( − \), which is the rightmost point of \( − \), is its head. The letter “F” may now be generated by means of the context-free rule system (L1–3).

(L1) \( S \rightarrow (S + S) \)
(L2) \( S \rightarrow (S \times S) \)
(L3) \( |, − \rightarrow S \)

The following is an example of a derivation yielding “F”:

\[
\begin{align*}
S & \rightarrow (S + S) \\
(S + S) & \rightarrow (S \times S) \\
(S \times S) & \rightarrow ((S + S) \times (S \times S)) \\
((S + S) \times (S \times S)) & \rightarrow ((S + S) \times -) \\
((S + S) \times -) & \rightarrow ((S + S) \times -) \\
((S + S) \times -) & \rightarrow ((S + S) \times -) \\
((S + S) \times -) & \rightarrow (S + S) \\
(S + S) & \rightarrow F
\end{align*}
\]


In graph grammars (cf. Ehrig 1979, 1987 and Nagl 1979) the basic concepts of string theory are generalized in another way. Instead of admitting several syntagmatic operations to form non-linear expressions, we assume only a simple one but allow it to be more complex than in the case of strings. Let us begin our explanation of this approach again with a closer look at strings and the relation of linear precedence between their atomic components. This precedence relation in strings is a projection from the temporal order of the physical production of the string tokens concerned. In representing a temporally extended sign event by a string, one abstracts from the duration of the minimal segments of the event. Consider the case of a sound event (an utterance, for instance) as an example. When we represent this event by means of a string of phonemes, these phonemes are treated as durationless atoms, which are ordered by syntagmatic precedence. If we want to remove this idealization, we must say that a syntactic item such as a phoneme consists of a left border, a characteristic content, and a right border. Pairs of adjoined items share one border. Instead of a string \( UVWXYZ \) we thus have a structure like \([V] | [X] | [Y]Z\) where the strokes indicate borders and the capital letters contents which are enclosed between borders. Such a structure may be viewed as a very simple colored graph. The borders are its nodes, which are not colored here, and the capital letters are the colors of the arcs. The diagram \( \gamma_1 \) makes evident the graph character of the structure \([U] | [V] | [X] | [Y]Z\).

Given a graph such as \( \gamma_1 \), how do we have to modify the rewrite rules in order to account for it? The first change we make is merely notational. Since we want to save the arrow (→) for a later purpose, we shall write “(P, Q)” instead of “P → Q”. Let us assume now that we have a rewrite rule \((W, ST)\) and the corresponding instance \(UVWXYZ \Rightarrow UVSTXYZ\) of a direct derivation; cf. § 4.4. ST corresponds to a graph \( \beta_2 \) which is as simple as \( \gamma_1 \) above (cf. Fig. 2.43).

<Insert Diagram>
Given graphs instead of strings, we now have to substitute arcs for string elements. The subgraph $\beta_1$ in $\gamma_1$, consisting of the arc colored with $W$ which leaves $k_2$ and enters $k_3$, must be replaced by $\beta_2$, thus transforming $\gamma_1$ into $\gamma_2$. How can this be achieved? The first step is to cut out in $\gamma_1$ the arc leading from $k_2$ to $k_3$; this results in a disconnected graph $\chi$ which consists of two separated subgraphs. Of course, $\chi$ is the graph counterpart of the string context $UVXYZ$ and we therefore call $\chi$ “a context”. Analogously, $\beta_1$ and $\beta_2$ correspond to the left and right side of a rewrite rule and thus these graphs will be called “left side” and “right side”, respectively. Now, the second step on our way from $\gamma_1$ to $\gamma_2$ (after the excision of the left side $\beta_1$ from $\gamma_1$ which resulted in the context $\chi$) is to add the right side $\beta_2$ to the context. The resulting graph is $\chi + \beta_2$. The operation $+$, which is known as “disjoint union”, resembles the union $(\cup)$ in set theory except that it dissimilates elements shared by the summands. The reason for using disjoint union instead of the simple union of set theory becomes clear in our last step towards the construction of $\gamma_2$. Note that the graph $\chi + \beta_2$ contains the nodes $k_2$ and $k_3$ from the context $\chi$ as well as the nodes $k_{20}$, $k_{21}$, and $k_{22}$ from the right side. But, clearly, for $\gamma_2$ we want an arc colored by $S$ which leaves $k_2$ and enters $k_{21}$ as well as an arc colored by $T$ which leaves $k_{21}$ and enters $k_3$. Thus we must identify the node $k_{20}$ with the node $k_2$ and the node $k_{22}$ with the node $k_3$ in $\chi + \beta_2$. In this way, the nodes $k_2$ and $k_3$ become shared elements of the context $\chi$ and of the right side $\beta_2$. Note that they constitute a graph which consists of two nodes but has no arc. This two-node graph, which occurs as a shared part of the context and the right side, is called “the interface”.

It is the need to have explicit control of shared nodes that made us choose disjoint union plus a subsequent identification operation as a way of constructing $\gamma_2$ out of the context $\chi$ and the right side $\beta_2$. We said, for example, that the interface is a shared part of the context and the right side. Strictly speaking, however, this is often impossible because context and right side are quite separate graphs. (After all, the right side is to be inserted into the context.) This is why we treat context, interface, and right as well as left side as four disjoint graphs and use identifications to connect them. The complex operation which takes the right side $\beta_2$, the context $\chi$ and the interface $\kappa$ to the graph $\gamma_2$ is called a “gluing operation” and the graph $\gamma_2$ is called a “gluing of $\beta_2$ and $\chi$ along $\kappa$” (cf. Ehrig 1979: 9–14). Obviously, the graph $\gamma_1$, from which we started, is a gluing of the left
side $\beta_1$ and the context $\chi$ along the interface $\kappa$. Thus each instance of the relation of direct derivability involves two gluings.

A similar but more sophisticated technique of gluing together local objects has been elaborated for the description of musical syntagms within mathematical music theory (Mazzola 1990; see above, § 3.). Global compositions are understood as manifolds, consisting of local compositions (comparable to the representation of our globe in an atlas of geographic maps), together with a coherent list of gluing prescriptions (for the gluing of these maps by means of identifying isomorphisms; cf. Fig. 2.44). As a simple example consider the diatonic scale \{c, d, e, f, g, a, b\} interpreted as a global structure, consisting of seven overlapping local compositions $I = \{c, e, g\}$, $II = \{d, f, a\}$, $III = \{e, g, b\}$, $IV = \{f, a, c\}$, $V = \{g, b, d\}$, $VI = \{a, c, e\}$, $VII = \{b, d, f\}$. The nerve of this syntagm is given in Fig. 2.45: Each of the 7 points represents a map, each of the 14 line segments represents a pair of two overlapping maps, each of the 7 triangles represents a triple of overlapping maps. The result is a Möbius strip which geometrically realizes Schönberg’s (1922) idea of harmonic strip. The non-orientability of the Möbius strip is related to a classical problem of the theory of functional harmony (Mazzola 1990: 176 ff.).

After this brief look into mathematical music theory let us now discuss the construction of visual forms by means of graph grammars. Consider constructing the shape “A” from the two connected graphs given in Fig. 2.46. (This again is a simplified case because it neglects the possibility of coloring.)

The construction of “A” can be achieved by a rewrite rule which allows the replacement of the arc $\alpha_1$ in $\eta_1$ by the entire graph
2. Syntactics

\( \beta_2 \). The resulting graph \( \eta_2 \) is given in Fig. 2.47.

![Fig. 2.47](image)

The one-arc graph \( \beta_1 \), which solely consists of an arc and its two nodes, is the left side of the rule involved. We depict it by the diagram in Fig. 2.48.

![Fig. 2.48](image)

The context \( \chi \) here takes the form of Fig. 2.49.

![Fig. 2.49](image)

In this case, the interface \( \kappa \) is again a two-node graph without any arc. When we say that the interface is a shared part of the context and the left side \( \beta_1 \) (as well as of the right side \( \beta_2 \)), then we imply that there are structure preserving functions \( m_1 \) and \( m_2 \) which map the graph \( \kappa \) into the graphs \( \chi \) and \( \beta_1 \) respectively (and this is true for \( \beta_2 \) too). Such structure preserving functions are called “graph morphisms”. Preservance of structure means that such a function respects source-target-relationships in the following sense: Neglecting coloring, we may (according to § 4.2. above) identify a graph \( \gamma \) as a 4-tuple \( \langle \Gamma, \Gamma_N, s, t \rangle \) consisting of a set \( \Gamma \) of arcs, a set \( \Gamma_N \) of nodes, the source function \( s \), and the target function \( t \). Given two graphs \( \gamma = \langle \Gamma, \Gamma_N, s, t \rangle \) and \( \gamma' = \langle \Gamma', \Gamma'_N, s', t' \rangle \) a graph morphism \( f \) is a pair of two functions \( f = \langle f_A, f_N \rangle \) where \( f_A \) maps arcs from \( \Gamma \) to those of \( \Gamma' \) and nodes from \( \Gamma_N \) to those of \( \Gamma'_N \) and for each arc \( \alpha \) of \( \Gamma \), both \( f_A(s(\alpha)) = s'(f_A(\alpha)) \) and \( f_A(t(\alpha)) = t'(f_A(\alpha)) \). This property of graph morphisms is represented by the diagram in Fig. 2.50.

![Fig. 2.50](image)

Interpreting this diagram, we claim that mappings which correspond to different tracks of arrows with the same source and target (but different routes, of course) are identical. In the language of category theory this is expressed by saying that the diagram commutes (cf., e.g., McLarty 1992).

In our case, the interface \( \kappa \) is the graph \( \kappa = \langle \emptyset, \{ k_9, k_{10} \}, \emptyset, \emptyset \rangle \); there are no arcs, and source \( s \) and target \( t \) are the empty function here. The graph morphism \( m_1 \) maps this interface into the left side \( \beta_1 \); this is expressed by “\( m_1 : \kappa \rightarrow \beta_1 \)”, which is read in category theory as “\( m_1 \) is an arrow with source \( \kappa \) and target \( \beta_1 \)”. Since our interface has no arcs, the first component of the morphism \( m_1 \) which maps into \( \beta_1 \) is again the empty function. Its second component maps \( k_9 \) to \( k_3 \) and \( k_{10} \) to \( k_6 \). Thus the relevant copy \( \kappa' \) of \( \kappa \) in the left side \( \beta_1 \) is \( \kappa' = \langle \emptyset, \{ k_9, k_{10} \}, \emptyset, \emptyset \rangle \).

The mapping \( m_2 \) of \( \kappa' \) into the context \( \chi \) maps \( k_9 \) to \( k_0 \) and \( k_{10} \) to \( k_1 \), and thus the relevant copy \( \chi' \) of \( \chi \) in the interface is \( \chi' = \langle \emptyset, \{ k_0, k_1 \}, \emptyset, \emptyset \rangle \).

Now employing graph morphisms \( g_1 : \beta_1 \rightarrow \eta_1 \) and \( g_2 : \chi \rightarrow \eta_1 \), the initial graph \( \eta_1 \) can now be interpreted as a gluing of the left side \( \beta_1 \) with the context \( \chi \) along the interface \( \kappa \). In our example, \( g_1 \) maps the only arc \( \alpha_1 \) of \( \beta_1 \) to the arc \( \alpha_1 \) of \( \eta_1 \); furthermore the nodes \( k_7 \) and \( k_8 \) are mapped to \( k_0 \) and \( k_1 \) respectively. (The reader should verify for himself that \( g_2 \) is indeed a graph morphism!) Since \( \chi \) is a subgraph of \( \eta_1 \), \( g_2 \) is simply the identical embedding of into \( \eta_1 \) which maps every com-
ponent of $\chi$ to itself. Now, $\eta_1$ does not only contain $\beta_1$ and $\chi$ as its part, it is also distinguished by what is called a “universal property” in category theory. If there is another graph $\eta_1'$ and two graph morphisms $g_1'$: $\beta_1 \rightarrow \eta_1'$ and $g_2'$: $\chi \rightarrow \eta_1'$, then there is a unique graph-morphism $h$: $\eta_1 \rightarrow \eta_1'$ for which the diagram in Fig. 2.51 commutes.

![Fig. 2.51](image)

Given objects $\kappa$, $\beta_1$, and $\chi$ with morphisms $m_1$: $\kappa \rightarrow \beta_1$ and $m_2$: $\kappa \rightarrow \chi$, an object such as $\eta_1$ with morphisms like $g_1$ and $g_2$ for which the universal property of the diagram in Fig. 2.51 holds is called a “pushout” in category theory. Thus the gluing of $\beta_1$ and $\chi$ along $\kappa (= 2)$ is a special case of a pushout, and category theory may be used in analyzing graph gluings as pushouts.

Relations analogous to those between the left side $\beta_1$, the context $\chi$, the interface $\kappa$, and the gluing $\eta_1$ also hold between the right side $\beta_2$, the same context $\chi$, the interface $\kappa$, and the gluing $\eta_2$. Again there are graph morphisms $n_1$ and $n_2$ which locate copies of $\kappa (= 2)$ within $\beta_2$ and $\chi$. Furthermore, there are morphisms $h_1$ and $h_2$ identifying the right side and the context in the gluing $\eta_2$. Generalizing the notion of a rewrite rule for the case of graphs, we can therefore define a “graph production rule” as a pair $p = \langle b_1, b_2 \rangle$ of graph morphisms $b_1$: $\kappa \rightarrow \beta_1$ and $b_2$: $\kappa \rightarrow \beta_2$, where $\beta_1$, $\beta_2$ and $\kappa$ are called “the left side”, “the right side”, and “the interface” of the production rule respectively. Given such a production rule $p = \langle b_1, b_2 \rangle$, a graph $\chi$ (called “the context”), and a graph morphism $d$: $\kappa \rightarrow \chi$, then a direct derivation consists of two pushouts $\eta_1$ and $\eta_2$, as shown in the diagram of Fig. 2.52.

![Fig. 2.52](image)

A derivation is, of course, a finite sequence of direct derivations as in the case of strings (cf. § 4.4).

This generalization of the basic concepts of string grammar to that of graph grammar looks rather complicated. Much of this impression may be due to the unfamiliar language of category theory and will thus disappear as soon as one has mastered this language. Nevertheless, certain technicalities remain which might motivate the search for a simpler approach to non-linear syntactic patterns. One strategy for such a search is to look for the mechanisms employed by the sign users themselves when they process such patterns (cf. Art. 6–12 of this Handbook). A relatively well known theoretical device which is said to have similarities to known physiological structures is the perceptron (cf. Minsky and Papert 1969, the review by Block 1970 with its extensive bibliography, and Suppes and Rottmayer 1974: 345–350). The perceptron may be described as a device which solves a problem by carrying out a number of small experiments and then using the outcomes to arrive at the solution.

Both the experiments and the problem to be solved are represented by characteristic functions, which are called “predicates” by Minsky and Papert. (In fact, the matter is a little more complicated; for details cf. Minsky and Papert 1969: 25 and § 1.6, Suppes and Rottmayer 1974: 347.) Let $\Phi = \{\varphi_1, \ldots, \varphi_n\}$ be the experimental predicates and $\psi$ the problem-predicate. For illustration, suppose we are interested in regions of the Euclidean plane. Then $\psi$ might be the predicate $\psi_{\text{CIRCLE}}$, which assigns the value TRUE to a region if an only if it is a circle (and the value FALSE otherwise). Now the idea is to reduce
such a complex predicate $\psi$ to a family $\Phi$ of simpler predicates. Suppose $\phi_1, \ldots, \phi_k$ are such simpler predicates. Then in order to decide whether $\psi_{\text{CIRCLE}}(X) = \text{TRUE}$ for the data $X$, we carry out the experiments — i.e., we determine the values $\phi_1(X), \ldots, \phi_k(X)$ — and then combine them by means of a procedure $\Omega$ in order to obtain $\psi_{\text{CIRCLE}}(X)$. (Of course, nothing guarantees that for each complex predicate one can find a family of simpler ones.) What is characteristic of the perceptron approach is its parallel character, the values of the simpler predicates being simultaneously given as inputs of the procedure $\Omega$. Thus we may assume that these values are determined by parallel but independent computations. The functioning of a perceptron is illustrated by the following diagram from Minsky and Papert (1969: 5).

![Fig. 2.53](image)

Two questions arise here: (1) What exactly constitutes the simplicity of the predicates $\phi_m$ of the set $\Phi$? (2) What exactly is the procedure $\Omega$? We shall answer the second question first. Minsky and Papert (1969: 10) assume that $\Omega$ is a procedure of weighted voting. This means that for each $\phi_m \in \Phi$ there is a number $w_{\phi_m}$ which measures the importance of the outcome of the experiment $\phi_m$. If $\phi_m$ is of minor importance, its weight $w_{\phi_m}$ will be small; crucial experiments, on the other hand, will have large numbers as their weights. Furthermore, there is a number $t$ such that $\psi$ takes the value $\text{TRUE}$ for data $X$ whenever the weighted average of the experimental outcomes exceeds $t$. The number $t$ is called “the threshold”. Thus we have the following reduction of $\psi$ to $\Phi$:

(P) $\psi(X) = \text{TRUE}$ if and only if

$$t < \sum_{\phi \in \Phi} w_{\phi} \cdot \phi(X).$$

If this holds, the predicate $\psi$ is said to be “linear in $\Phi$”. (There are some alternative but equivalent ways to define linearity; cf. Minsky and Papert 1969: 29.)

Let us now turn to the question of what constitutes the simplicity of the predicates $\phi_m$ in the family $\Phi$. So far there exists no generally accepted theory of the simplicity of predicates, though there are some approaches towards such a theory (cf. Goodman 1972: ch. VII, von Kutscher 1972: §4.2, Sober 1975, which is of semiotic importance because Sober deals with the question of the simplicity of grammatical descriptions). Therefore, Minsky and Papert provide explications of the notion of simplicity for the special case of geometric predicates (where “geometric” should be understood in a somewhat abstract sense which will become clearer in the following). The data to which their predicates apply are subsets $X$ of a set $R$ of points; $R$ is called “the retina”. We may think of the data $X$ as that part of the retina which is stimulated, or we may view the set $R$ as a screen and $X$ as the black points on the screen, or $R$ as a plane and $X$ as a figure on it; cf. Fig. 2.53. The simplicity of a predicate $\phi \in \Phi$ may now consist in the fact that the values $\phi(X)$ do not depend on the whole data $X$ but only on a subset $Y \subseteq X$ so that $\phi(X) = \phi(Y)$. If, furthermore, there is a smallest part $S_\phi$ of the retina (i.e., $S_\phi \subseteq R$) such that that part $Y$ of the data $X$ on which $\phi$ really depends may be characterized as $Y = X \cap S_\phi$, then $S_\phi$ is called “the support of $\phi$". Clearly, it makes sense to call a predicate “simple” if it has a finite support — even if this notion of simplicity is not a general one but is restricted to our abstract geometrical settings.

If a predicate $\psi$ can be characterized by a biconditional of the kind (P) where the predicates from $\Phi$ all have finite support, then only finite sets of points from $X$ need to be considered in order to decide whether $\psi$ applies to $X$. This gives $\psi$ a local character; $\psi$ does not depend on the whole of $X$ but only on small — even finite — parts of it. An example for a local predicate of this kind is convexity ($\psi_{\text{CONVEX}}$). Roughly, a figure $X$ is convex if every line segment which links two points of the figure completely lies within the figure. Thus a circle is convex but a half-moon is not (cf. Fig. 2.54).

Intuitively, $\psi_{\text{CONVEX}}$ is local because we only have to consider triples of points in order to decide whether a figure is convex or not (cf. the above drawings).
For any predicate \( \psi \), the smallest number \( k \) such that there is a family of predicates \( \Phi \) characterizing \( \psi \) whose members all have a finite support of at most \( k \) members is called “the order of \( \psi \)”. The above example shows that \( \psi_{\text{convex}} \) is of order three, as are for example the predicates of being a solid rectangle and that of being a hollow one (cf. Minsky and Papert 1969: § 6.3.2). (A hollow rectangle is a rectangle with a hole in it.) There are, however, some predicates which would be considered fairly simple from an intuitive point of view but which turn out to be of no finite order. The most basic example is connectedness; a figure is connected if any two points can be linked by a (not necessarily straight) line which wholly lies within the figure. Figures with isolated parts, as in Fig. 2.55, are not connected.

\[
\begin{align*}
\psi_{\text{convex}} &= \text{TRUE} & \psi_{\text{convex}} &= \text{FALSE} \\
\text{Fig. 2.54}
\end{align*}
\]

Such cases (as well as others, cf. Suppes and Rottmayer 1974: 349) show the serious limitations of the perceptron approach as developed by Minsky and Papert. However, it may be that these limitations are only due to the particular explication provided by these authors (cf. Block 1970).

In any case, this fact as well as the example of graph grammars shows that an adequate framework for the analysis of multi-dimensional syntactic patterns will unavoidably employ notions considerably more complex than those needed for string codes.

\[
\begin{align*}
\psi_{\text{convex}} &= \text{TRUE} & \psi_{\text{convex}} &= \text{FALSE} \\
\text{Fig. 2.55}
\end{align*}
\]

6. Selected references


2. Syntactics


Hindemith, Paul (1940), Unterweisung im Tonsatz. Mainz: Schott.

Uppsala, Sweden, Nov. 8.—9., 1974 (= Filosofiska Studier 26, 1: 1—92.
Holenstein, Elmar (1975), Roman Jakobsones phe

König, Esther (1989), How to Fit the Lambek Calculus into the Chomsky Hierarchy. Stuttgart: IBM Germany, IWBS Report 80.


Shieber, Stuart M. (1986), An Introduction to Unification-Based Approaches to Grammar. Stanford CA.: Center for the Study of Language and Information. CSLI Lecture Notes 43.


Roland Posner and Klaus Robering, Berlin (Germany)