Trees and Texts

Omslag

Tekening: 'Winterswijkse wilg' van Agatha Koers
Ontwerp: Cédilles, Amsterdam

Trees and Texts

Proefschrift
ter verkrijging van de graad van Doctor
aan de Rijksuniversiteit te Leiden,
op gezag van de Rector Magnificus Dr. L. Leertouwer,
hoogleraar in de faculteit der Godgeleerdheid,
volgens besluit van het College van Dekanen
te verdedigen op woensdag 17 mei 1995
te klokke 16.15 uur

door

Paulien ten Pas

geboren te Vlissingen in 1966

Promotiecommissie

Promotor:	Prof. dr. G. Rozenberg
Co-promotor:	Dr. H.J. Hoogeboom
Referent:	Prof. dr. A. Salomaa (Universiteit van Turku, Finland)
Overige leden:	Prof. dr. W. Thomas (Christian-Albrechts-Universität Kiel, Duitsland)
	Dr. T. Harju (Universiteit van Turku, Finland)
	Prof. dr. G. Engels
	Prof. dr. H.A.G. Wijshoff

Wie
een boom
tekent
laat
het weten
zien.

Jan Arends, 'Lunchpauzegedichten', 1974.

Preface

This thesis consists of two parts, each consisting of three chapters, and an introduction. Each of the chapters is a paper; however, the references from the papers have been collected at the end of the respective parts. Chapters 1,2, and 4 are written in cooperation with A. Ehrenfeucht and G. Rozenberg, Chapter 3 is written in cooperation with A. Ehrenfeucht, J. Engelfriet, and G. Rozenberg, and Chapters 5 and 6 are written in cooperation with H.J. Hoogeboom.

Chapter 1 (*Properties of Grammatical Codes of Trees*) has appeared in Theoretical Computer Science 125 (1994) pp. 259–293, Chapter 2 (*A Note on Binary Grammatical Codes of Trees*) will appear in Theoretical Computer Science, Chapter 3 (*Grammatical Codes of Trees and Terminally Coded Grammars*) will appear in Fundamenta Informaticae, Chapter 4 (*Context-free Text Grammars*) has appeared in Acta Informatica 31 (1994) pp. 161–206, Chapter 5 (*Text Languages in an Algebraic Framework*) will appear in Fundamenta Informaticae, Chapter 6 (*Monadic Second-Order Definable Text Languages*) is a technical report of Leiden University (submitted for publication) – a short version of this chapter has appeared in the proceedings of the conference Mathematical Foundations of Computer Science 1994.

Since the chapters are written as independent journal contributions, several notions and notations are introduced more than once.

The research presented here has been carried out within the framework of ESPRIT Basic Research Action "ASMICS".

Contents

Introduction

One of the central notions in computer science is that of a tree, in particular that of a (directed) ordered tree. As Knuth puts it ([12]), trees are the most important nonlinear structures arising in computer algorithms. In general, any hierarchical classification scheme leads to a tree structure.

In formal language theory, ordered trees function as syntactic structures of words; such a syntactic structure expresses the syntactic relationships between various parts of a word. Knowing the syntactic structure of a word allows one to give a semantical interpretation of the word.

Classically, the syntactic structure of a word is determined by an (unambiguous) grammar the language of which contains the given word. Thus, a context-free grammar defines a language, and it assigns to every word in the language a syntactic tree, viz. its derivation tree. In the framework given by Chomsky, this tree is the *deep structure* of the word ([3]), from which its meaning can be derived. Since such a syntactic tree is determined by a grammar, we will also call it the grammatical tree structure of the word.

Clearly, a word belongs to many different languages, and so it may have many different grammatical tree structures. In this thesis, we are interested in the situation where each separate word has an "individual", a priori available, syntactic structure, which is independent of any language generating device. Then, for a grammar defining a set of such "structured" words, we require that for every generated word, the grammatical structure assigned to it by the grammar matches its individual syntactic structure. In this context a grammar may be seen as a definition of similarity of individual syntactic structures of the words from its language.

The two parts of this thesis represent two different approaches within this set-up:

A. *implicit:* the individual syntactic tree can be deduced from the word itself, or

B. *explicit:* the word is equipped with an additional structure, which determines its individual syntactic tree.

Let us first discuss the approach A. We consider a language such that each of its words "contains" an encoding of its individual syntactic tree (which is an unlabeled ordered tree). If this language is generated by a context-free grammar G, then for every word its grammatical tree, i.e., its unlabeled derivation tree in G, should be equal to its individual syntactic tree. One may expect that such a grammar has a simple parsing algorithm, and hence that a language with the above property can be efficiently recognized.

An example of such a situation is a parenthesis language (see, e.g., [14]), consisting of words over an alphabet $\Delta \cup \{(,)\}$, where the parentheses inside every word determine an unlabeled ordered tree "on top of" the word. We do not view the parentheses as additional structure on a parenthesis-free word, but as part of the word – the parentheses are just labels occurring at certain positions in the word. A parenthesis grammar G, with productions of the form $A \to (w)$, generating a parenthesis language satisfies our requirement that the unlabeled derivation tree in G of a word equals the tree defined by the parentheses inside the word.

In a generalization of this idea, every unlabeled ordered tree is encoded into a code-word over an arbitrary finite alphabet, i.e., an alphabet without explicit pairs of parentheses. This code-word must be recoverable from the word that is generated by the grammar. Since the encoding of trees we use is applied to derivation trees, it is natural to require the following property: when a production is applied, i.e., when children are added to a leaf of the tree, then the code-word for the new tree can be obtained by locally altering the code-word for the original tree.

Such a property is guaranteed if one encodes trees in a grammatical way, using "grammatical codes of trees". In Part I of the thesis we study these codes. Grammatical codes are interesting in itself as an elegant way of coding trees into strings. But we will also use these codes to define a class of context-free grammars satisfying our basic requirement: the syntactic tree can be deduced from the word itself. More precisely, for every word its syntactic tree is obtained by applying a projection followed by a decoding of the so obtained code-word. The parenthesis grammars form a subclass of this class of grammars.

The idea of the approach B is to generate objects each of which consists of a word equipped with an *additional* structure, that can be used to construct the individual syntactic tree on top of the word.

To start with, the additional structure may be an explicit tree. We discuss two known examples of such a situation.

Tree adjoining grammars (TAGs) are designed by Joshi et al. ([11]) for the purpose of describing natural languages. Starting from a base set of elementary trees, larger trees are obtained by substituting subtrees for inner nodes. A sentence is generated by a TAG if it is the yield of a tree generated by the TAG. In our terminology, this tree can be seen as the "individual syntactic structure" of the sentence. The meaning of a sentence generated by a TAG is supposed to be obtained from its "derivation structure"(see [11]). This derivation structure is the "grammatical tree structure" assigned to the sentence by the TAG. The situation here is more general than our approach, since the derivation structure is not *equal* to the generated tree, but implicitly characterizes it.

Regular tree grammars ([10]) generate trees, where in one generation step a subtree is substituted for a leaf. The word languages generated by regular tree grammars, obtained by taking the yields of generated trees, are precisely the context-free languages. In fact, every context-free grammar G can be interpreted as a regular tree grammar G' such that G' generates precisely the (unlabeled) derivation trees of G. Thus, "grammatical trees", i.e., derivation structures assigned by the grammar, coincide with "individual syntactic trees", i.e., the objects in the generated tree language.

In the cases of TAGs and regular tree grammars, the additional structure for a word is a tree. Suppose now that the additional structure is given by binary relations, or dependencies, between various occurrences of (possibly different) letters. This means that the object to be generated is a graph rather than a tree.

There is an extensive theory of different types of graph grammars, based on different substitution mechanisms (see, e.g., [8]). In general it will not be easy to obtain for every graph generated by a graph grammar a grammatical tree structure that can also be defined independently of the grammar in terms of the graph itself. In Part II of the thesis we consider specific graphs, called "texts", for which this can be done. As is the case for regular tree languages, the underlying word languages generated by the corresponding grammars are the context-free word languages.

Texts are close to words, in the sense that they are obtained from words by a simple generalization. More precisely, a *text* is a labeled domain with two linear orders on it. Note that a word is a labeled domain with a single linear order on it, and so a text is a word with an additional linear order of its (labeled) domain.

What kind of information can be described by a text? One can think of a situation where the first order gives the actual order in which information is given, and the second order reorders the information in such a way that it becomes easier to process. E.g., in the case of programming languages, the first order may give the sequence of statements composing a program, while the second order gives the order in which these statements should be evaluated. Also, for natural languages, the first order may represent the word order from a given sentence and the second order gives another order of the words satisfying certain logical properties; e.g., an adjective modifying a subject is put before that subject in the second order (whereas, e.g., in French, in the first order it may come either before or after its subject).

The additional structure of a word in a text, viz. the second linear order, determines a tree. This tree is the individual syntactic structure associated with the text. It should be stressed here that such a tree is more general than the ordered trees considered until now – the difference is explained in this introduction later on.

The next step of our methodology is to have a notion of "text grammar" that assigns grammatical tree structures to texts. This is achieved by a straightforward generalization to texts of the notion of context-free word grammar. Following our approach, the derivation trees obtained within such a context-free text grammar should be equal to the syntactic trees as defined by the additional linear orders of the generated texts. In Part II of the thesis we consider the languages of texts obtained in this way.

We now continue our introductory explanation for each part of the thesis separately.

Part I: Grammatical codes of trees

Grammatical codes of trees are introduced in [5]. In particular a subclass of these codes, called *strict codes*, is studied there. We recall now briefly the basic concepts from [5]. A *code* (of trees) is an injective function from the set of all chain-free ordered trees to words over a finite alphabet (we consider trees modulo isomorphism – injectivity means that non-isomorphic trees get different images; by chain-free we mean that every inner node has at least two children).

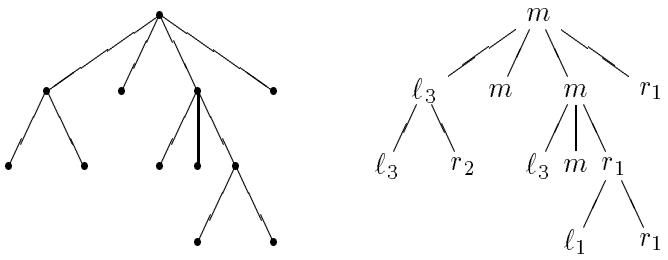


Figure 0.1: A tree t and its labeled version

Furthermore, a code is supposed to have the following properties:

- *length-preserving*: the length of the word coding a tree equals the number of leaves of that tree, hence the code-word can be seen as a leaf-labeling of its frontier,
- *local*: given a tree t coded by the word x, the code-word of the tree obtained by adding n children to the i'th leaf of t is obtained by substituting a word of length n for the i'th letter of x.

A length-preserving and local code is called a *grammatical* code. This is because the local function defining for each letter a and each $n \geq 2$ the word of length n that must be substituted for a determines the production set of a grammar.

The grammars that are used for this purpose are *unlimited OS systems*. Such a system is like a context-free grammar except that there is no distinction between terminals and non-terminals, and it may have infinitely many productions. Moreover, in order to correspond exactly to a code, an unlimited OS system must be unambiguous and such that for each letter a and for each $n \geq 2$ there is precisely one production of length n for a. The *code-word* for a tree t is then the yield of the (unique) derivation tree that has t as its underlying tree. E.g., the unlimited OS system with axiom m and productions

$$\begin{aligned} \ell_1 &\to \ell_2 m^k r_1, & m &\to \ell_3 m^k r_1, \\ \ell_2 &\to \ell_2 m^k r_2, & r_1 &\to \ell_1 m^k r_1, \\ \ell_3 &\to \ell_3 m^k r_2, & r_2 &\to \ell_1 m^k r_2, \end{aligned}$$

for all $k \geq 0$, gives a grammatical code. For the left tree t in Figure 0.1, the corresponding derivation tree is the right tree, and hence the code-word for t is $\ell_3 r_2 m \ell_3 m \ell_1 r_1 r_1$.

Strict codes are defined by certain consistency properties on the occurrences of subwords in a code-word, and by the condition that the code is *rich*, meaning that every word must occur as a subword of some code-word. The consistency properties of a strict code ensure its injectivity, and hence the unambiguity of the corresponding grammar.

The example grammar given above corresponds to a strict code. The grammars corresponding to strict codes can be characterized by certain demands on the alphabet and the production rules. One of these demands is that such a grammar uses exactly six letters. Note that any length-preserving code must use at least six letters, since the number of trees with n leaves exceeds 5^n for large enough n (see, e.g., [12]). Hence in this sense, strict codes are minimal. This minimality follows from the richness condition.

Outline of Part I

Part I consists of three chapters. The first two chapters continue the investigation of basic properties of grammatical codes of trees, and in the third chapter these codes are related to context-free grammars.

The first section of Chapter 1 establishes properties of a code that are weaker than the properties of strict codes, but already ensure the unambiguity of the corresponding grammar. This leads to the notions of marked and nonoverlapping codes.

For a *marked code* it is required that the borders of right-hand sides of productions from the corresponding grammar are "marked", and hence distinguishable within a code-word; in the example given above, the letters ℓ_1, ℓ_2, and ℓ_3 mark left borders, and the letters r_1, r_2 mark right borders. Due to the marking of the borders, bottom-up parsing for such a grammar is easy. It follows that the grammar is indeed unambiguous and moreover bottom-up parsing corresponds to decoding. It is shown that marked codes with a minimal alphabet (of 6 letters) are precisely the strict codes. In a *non-overlapping code*, the right-hand sides of the corresponding grammar do not overlap, which implies that also in this case, right-hand sides can be recognized within a code-word. The class of non-overlapping codes contains the marked codes, as well as minimal codes (with 6 letters) that are not strict.

The third section of Chapter 1 concerns codings of node-labeled trees. The grammatical code used to code the underlying tree defines a "direction function", which determines for each inner node a path to a leaf, where the label of the inner node is stored. Hence a leaf of a node-labeled tree is labeled by a triplet consisting of the label of the leaf itself, a code-letter of the underlying grammatical code, and, if present, a stored label of an inner node. The so obtained coding of node-labeled trees into words over an alphabet of such triplets is then length-preserving, and local in the sense that when adding a subtree at a leaf, the corresponding code-word is obtained by substituting a word for the triplet at that leaf.

The special case of binary trees (which are chain-free, and hence full) is considered in the second section of Chapter 1. An alphabet of 4 letters (rather than 6 letters) suffices here – strict binary codes have two "left" letters marking left borders of right-hand sides, and two "right" letters marking right borders. Again, minimizing marked binary codes leads to strict binary codes.

It is shown in Chapter 2 that every minimal grammatical code for binary trees (i.e., a grammatical code that uses 4 letters) is strict. Also, a way of extending binary codes to codes for arbitrary trees is presented. This yields a class of codes for arbitrary trees incomparable with the classes of strict, marked, and nonoverlapping codes.

According to our approach, we consider context-free grammars such that the grammatical tree structures of generated words are equal to the individual syntactic structures of those words. We use marked codes to code the individual syntactic structures. This leads to the notion of *terminally coded grammars*, which are investigated in Chapter 3. These are context-free grammars for which there exists a projection from the terminal alphabet onto a set of code-letters such that for every generated word, the code-word for any of its unlabeled derivation trees equals the projective image of the word. Then the grammar (unlimited OS system) that defines the marked code is "contained" in the terminally coded context-free

grammar. The simple decoding mechanism for marked codes provides the basis of an efficient parsing algorithm for terminally coded languages. In fact, terminally coded grammars are closely connected to a subclass of the *simple precedence grammars* ([1]). Also, terminally coded languages have a decidable equivalence problem. Although the class of terminally coded languages seems rather restricted, it nevertheless turns out that every context-free language is the projection of a terminally coded language.

Part II: Text languages

As explained already, a text is a word equipped with an additional linear order on its labeled domain. Formally, a *text* is defined as a triple $\tau = (\lambda, \rho_1, \rho_2)$ such that λ is a labeling function, and ρ_1 and ρ_2 are linear orders on the domain of λ. Usually, we abstract from the elements of the domain; one may assume then that the first order equals $(1, 2, \ldots, n)$ and thus view the text τ as the word $\lambda(1)\lambda(2)\ldots\lambda(n)$ together with the additional linear order ρ_2 on the domain $\{1, \ldots, n\}$.

This notion of text was introduced in [7], as a representation of specific "2-structures". A 2-structure is a directed edge-labeled, loop-free graph such that between every pair of distinct nodes there is precisely one (directed) edge. 2-structures are introduced in [6], where also the theory of hierarchical representations (decompositions) for 2-structures is developed. This theory is closely related to the theory of modular decompositions of graphs (see, e.g., [13, 9]). It provides a way of assigning node-labeled trees to a 2-structure. In particular, each 2-structure has a unique tree representation of a specific form, called the "shape" of the 2-structure, from which all other tree representations can be derived.

Through the correspondence between texts and certain 2-structures, the results on decomposition trees may be transferred to texts. The tree-like structure used to represent a text hierarchically is a so-called leaf-labeled *bi-ordered* tree, which generalizes the concept of a leaf-labeled ordered tree often used to give the syntactic structure of a word. A tree is bi-ordered if with each inner node *two* linear orderings of its children are associated. These local orderings determine then two orderings on the leaves of the tree, and hence, together with the leaf-labeling function, a text on the leaves.

A bi-ordered tree representing a text describes a modular decomposition of the text, which coincides with the decomposition as given by the theory of 2-structures. In general, a text may have several such decomposition trees. If a text has no non-trivial decomposition, then it is called *primitive*, and the unique (trivial) tree representing such a text consists of root and leaves only, where the orderings at the root are the orderings of the text itself. For non-primitive texts, it is natural to look for a "maximal" decomposition. Such a maximal decomposition does not have to be unique. However, it follows from results on the decomposition of 2-structures that maximal decompositions of a text differ only by certain binary subtrees. Hence, by not decomposing the parts represented by these binary subtrees one obtains for each text a unique bi-ordered tree, which is the "shape" of the text; by decomposing every such part into a "right-most" binary subtree one obtains for each text a unique maximal decomposition, called the "r-shape" of the text.

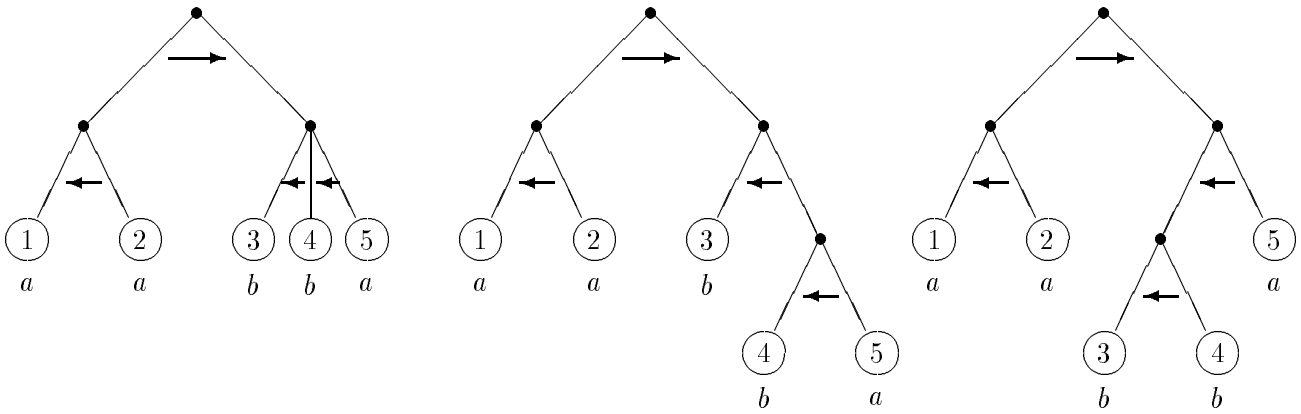


Figure 0.2: Three tree representations for the text $\tau = (\lambda, (1,2,3,4,5), (2,1,5,4,3))$

For the text $\tau = (\lambda, (1,2,3,4,5), (2,1,5,4,3))$, where $\lambda(1) = \lambda(2) = \lambda(5) = a, \lambda(3) = \lambda(4) = b$, three bi-ordered trees are given in Figure 0.2. The first ordering on children of a node is the left-to-right order, the second one is indicated by arrows. The leftmost tree is the shape of τ, the middle tree is the r-shape of τ, and the rightmost tree is a maximal decomposition of τ which is not the r-shape.

Thus, the additional linear order of a text determines an individual syntactic tree for the text, which may be one of the above "natural" tree representations. Following our methodology, we consider context-free grammars for texts such that for every generated text the derivation tree matches its syntactic tree. There are three candidates for this "natural" tree representation: a maximal decomposition, the shape, and the r-shape. These alternatives lead to different classes of text languages. The obtained classes are well-behaved in the sense that they admit characterizations in terms of universal algebra and second-order logic.

Outline of Part II

Part II consists of the last three chapters of this thesis.

In Chapter 4 we begin with an explanation of the relationships between texts and 2-structures, and between tree representations for both of them. Subsequently, context-free text grammars are introduced, which are direct generalizations of context-free word grammars. Productions are of the form $A \to \tau$ with A a nonterminal and τ a text. Application of such a production, say to a sentential form $\tau' = (\lambda', \rho_1', \rho_2')$, amounts to substituting the text $\tau = (\lambda, \rho_1, \rho_2)$ for an element of the domain of τ' with label A, where ρ_1 is substituted into ρ_1', and ρ_2 is substituted into ρ_2'.

While derivation trees in a context-free word grammar are node-labeled *ordered* trees, derivation trees in a context-free text grammar are node-labeled *bi-ordered* trees. These derivation trees (without inner labels) are required to be either maximal decompositions, or shapes, or r-shapes, corresponding to the three natural representations for texts.

It is shown that, given a context-free text grammar, one can decompose each text at the right-hand side of a production, obtaining in this way an equivalent context-free text grammar

such that every right-hand side of a production is a primitive text. Each derivation tree of the so obtained grammar is a maximal decomposition of the generated text. Hence, requiring that derivation trees are maximal decompositions yields the whole class of context-free text languages.

Requiring that derivation trees are shapes of the generated texts leads to *shapely* grammars, and their languages called *shapely languages*. The shapes of such a language have bounded out-degree, which in general does not hold for a context-free text language. Hence not every context-free text language is shapely. It is shown that the shapely languages are characterized by the requirement that the shapes have bounded out-degree. Still every context-free *word* language is the underlying word language of a shapely text language.

The case where derivation trees are required to be r-shapes is dealt with in Chapter 5. First of all, an algebraic framework for text languages is given, leading to the definition of equational and recognizable text languages. The equational text languages are precisely the context-free ones, which supports the naturality of our notion of context-freeness. The main result of Chapter 5 is that the recognizable text languages coincide with the "right-linear" text languages, which are the languages generated by a context-free grammar the derivation trees of which are precisely r-shapes. Recognizable text languages form a proper subclass of the class of context-free text languages, and strictly contain the shapely languages. The properties of the obtained classes of text languages are compared with those of the analogous language classes in the case of trees and words.

Recognizable languages can be characterized in terms of monadic second-order logic for structures such as words, trees, and traces (see [2, 4, 15, 16]). In Chapter 6 we obtain a similar result for text languages, by using the known equivalence for tree languages. The major step in the proof consists of showing that the construction of the r-shape from a given text is expressible in monadic second-order logic. From this it follows that a formula defining a tree language of r-shapes can be translated into a formula for the corresponding text language. With the help of a similar (reverse) translation from text languages to tree languages, we prove that a text language is recognizable if and only if it is definable by a formula in monadic second-order logic.

The proof idea used here will in general not apply for arbitrary graphs instead of texts. However, by looking closely at the properties on which the proof relies, it is possible to extend this result to a specific class of graphs. These matters are discussed at the end of Chapter 6.

References

[1] A.H. Aho and J.D. Ullman, *The Theory of Parsing, Translation, and Compiling*, Prentice Hall, New Jersey, 1972.

[2] J.R. Büchi, Weak second-order arithmetic and finite automata, *Zeitschrift für Mathematik, Logik und Grundlagen der Mathematik* **6** (1960) 66–92.

[3] N. Chomsky, *Aspects of the theory of syntax*, M.I.T. Press, Cambridge, Massachusetts, 1965.

[4] J. Doner, Tree acceptors and some of their applications, *Journal of Computer and System Sciences* **4** (1970) 406–451.

[5] A. Ehrenfeucht and G. Rozenberg, Grammatical codes of trees, *Discrete Applied Mathematics* **32** (1991) 103–129.

[6] A. Ehrenfeucht and G. Rozenberg, Theory of 2-structures, Parts I and II, *Theoretical Computer Science* **70** (1990) 277–342.

[7] A. Ehrenfeucht and G. Rozenberg, T-structures, T-functions, and texts, *Theoretical Computer Science* **116** (1993) 227–290.

[8] H. Ehrig, H.-J. Kreowski, and G. Rozenberg, eds., *Graph Grammars and their Application to Computer Science*, Lecture Notes in Computer Science 532, Springer, Berlin, 1990.

[9] J. Engelfriet, T. Harju, A. Proskurowski, and G. Rozenberg, Characterization and complexity of uniformly non-primitive labeled 2-structures, Technical Report 94-31 (1994), Leiden University.

[10] F. Gecseg and M. Steinby, *Tree Automata*, Akademiai Kiado, Budapest, 1984.

[11] A. Joshi, L. Levy, and M. Takahashi, Tree adjunct grammars, *Journal of Computer and System Sciences* **10** (1975) 136–163.

[12] D. Knuth, *The Art of Computer Programming, vol.1: Fundamental Algorithms*, Addison-Wesley, Reading, Massachusetts, 1973.

[13] R. Möhring and F.J. Radermacher, Substitution decomposition for discrete structures and connections with combinatorial optimization, *Annals of Discrete Mathematics* **19** (1984) 257–356.

[14] A. Salomaa, *Formal Languages*, Academic Press, New York and London, 1973.

[15] J.W. Thatcher, J.B. Wright, Generalized finite automata theory with an application to a decision problem of second-order logic, *Mathematical Systems Theory* **2** (1968) 57–82.

[16] W. Thomas, On logical definability of trace languages, in *Proc. ASMICS Workshop 'Free Partially Commutative Monoids'*, V. Diekert, ed., Rep. TUM I9002 (1990), TU München.

Part I

Grammatical Codes of Trees

Chapter 1

Properties of Grammatical Codes of Trees

Abstract

In this paper grammatical codes of trees are investigated. In particular it is shown how to extend grammatical codes of trees to node-labeled trees, and the case of binary trees (forests) is studied.

Introduction

Trees play an important role in many branches of science, among others in linguistics, mathematics, and computer science. *Linear notations*, or *linear codings*, for trees are useful in many applications and there are many methods for obtaining such codings (see, e.g., [12, 14]). The notion of a *strict code* for trees has been introduced in [5] where it has been shown that strict codes are grammatical (in a well-defined sense). The intrinsic feature of a strict code is that the length of the word coding a tree t (according to a strict code) equals the number of leaves of t, and so it can be considered as a labeling of the leaves of t by letters of the alphabet of the code. It is also shown in [5] that, in order to code (the class of) all trees, a strict code has to use precisely *six* letters; in linguistic terms this means that one needs precisely six letters to code deep structures of words.

In this paper we continue the investigation of grammatical codes of trees, and in particular, the investigation of strict codes.

In Section 1.1 we recall the notion of a strict code and we give an equivalent definition of strict codes (through the notion of a *marked code*). Also, the notion of a *composite category* (basic to parsing of strict codes) is discussed.

In Section 1.2 we consider the case of coding *binary trees* (or forests). In particular we prove that there are 24 different (non-isomorphic) strict codes for coding binary forests, and one needs exactly four letters for such a code.

In Section 1.3 we demonstrate how to extend the notion of a strict code to *node-labeled trees*. We give an axiomatization of such codes, and then we provide a combinatorial characterization of them.

Preliminaries

We assume the reader to be familiar with the basic notions of graph theory, especially concerning trees (see, e.g., [9, 12]) and the basic notions concerning context-free grammars (see, e.g., [14]). In this section we recall some of these notions in order to establish the notation for this paper, and we introduce some notions to be used in this paper.

For a set Z, $\#Z$ denotes its cardinality; $\emptyset$ denotes the empty set. For sets Y and Z, $Y \subseteq Z$ denotes the inclusion of Y in Z, and $Y \subset Z$ denotes the strict inclusion of Y in Z.

$\mathbf{N}_+$ denotes the set of all positive natural numbers.

For a function $\varphi : X \to Y$, $dom(\varphi)$ denotes X and $ran(\varphi) = \{y \in Y \mid y = \varphi(x) \text{ for some } x \in X\}$. We consider only total functions.

For a sequence x, $|x|$ denotes its length, $first(x)$ denotes the first element of x, and $last(x)$ denotes the last element of x. For an $1 \leq i \leq |x|$, $x(i)$ denotes the ith element of x. A *segment* of a sequence x is a sequence $(x(i), x(i+1), \ldots, x(i+k))$, where $1 \leq i \leq i+k \leq |x|$. These notations carry over to words which are sequences of letters.

In this paper, by a tree we mean a nonempty rooted directed ordered tree without chains (i.e., each inner node of t has at least two direct descendants).

Let t be a tree.

$nd(t)$ denotes the set of all nodes of t, $in(t)$ denotes the set of internal nodes of t, $leaf(t)$ denotes the set of leaves of t, $\langle leaf \rangle(t)$ denotes the sequence of all leaves of t ordered according to the order of t, and $root(t)$ denotes the root of t.

For an internal node v of t, $ddes_t(v)$ is the set of all direct descendants of v in t and $\langle ddes \rangle_t(v)$ is the sequence of all direct descendants of v in t (i.e., the elements of $ddes_t(v)$ ordered according to the order of t).

A *path (from v_1 to v_n)* is a sequence of nodes $(v_1, \ldots, v_n)$, $n \geq 1$, such that $v_{i+1} \in ddes_t(v_i)$ for all $i = 1, \ldots, n-1$. The path from v to w, where $v, w \in nd(t)$, is denoted by $\pi(v, w)$, and the set of nodes on this path is denoted by $\Pi(v, w)$. For $v \in nd(t)$, the *level of v*, denoted by $level(v)$, is $|\pi(root(t), v)| - 1$; the *depth* of t, denoted by $depth(t)$, is $\max\{level(v) \mid v \in leaf(t)\}$.

If $x \in leaf(t)$, then

x is a *leftmost child* if there exists $v \in in(t)$ such that $x = first(\langle ddes \rangle_t(v))$, (we then write $x = left_t(v)$),

x is a *rightmost child* if there exists $v \in in(t)$ such that $x = last(\langle ddes \rangle_t(v))$, (we then write $x = right_t(v)$),

x is a *middle child* if there exists $v \in in(t)$ such that x is not $root(t)$ and x is neither a leftmost nor a rightmost child.

For $v \in nd(t)$, $sub_t(v)$ denotes the subtree of t rooted at v; $leaf(sub_t(v))$ is called the *contribution of v*, denoted by $contr_t(v)$, and the *ordered contribution* of v, i.e., $\langle leaf \rangle(sub_t(v))$, is denoted by $\langle contr \rangle_t(v)$.

If w is a segment of $\langle leaf \rangle(t)$, then

w is a *sibling segment (of $\langle leaf \rangle(t)$)* if $|w| = 2$ and there exists $v \in in(t)$ such that w is a segment of $\langle ddes \rangle_t(v)$, and

w is a *complete segment (of $\langle leaf \rangle(t)$)* if there exists $v \in in(\mathrm{t})$ such that $w = \langle ddes \rangle_t(v)$.

Two trees t and t' are *isomorphic* if there is a bijection $\delta : nd(t) \to nd(t')$ such that $\delta(root(t)) = root(t')$ and for each $v \in in(t)$, if $\langle ddes \rangle_t(v) = (v_1, \ldots, v_n)$, then $\langle ddes \rangle_{t'}(\delta(v)) = (\delta(v_1), \ldots, \delta(v_n))$.

For a subtree t' of t, $t \bullet t'$ denotes the tree that results from t by removing t' (if the resulting tree has no chains), and $t \circ t'$ denotes the tree that results by removing t' except its root. If we remove several subtrees t_i ($i \in I$) from t, then we write $t \circ \bigcup_{i \in I} t_i$ (resp. $t \bullet \bigcup_{i \in I} t_i$) for the resulting tree.

For $1 \leq i \leq \#leaf(t)$ and $n \geq 2$, $subs_t(i, n)$ denotes the family of all isomorphic trees resulting from t by adding n new nodes and making them the direct descendants of $\langle leaf \rangle(t)(i)$ (which in the resulting tree becomes an internal node).

A *cut of* t is a set $\rho \subseteq nd(t)$ such that, for each $w \in leaf(t)$, $\#(\Pi(root(t), w) \cap \rho) = 1$. For $v \in in(t)$, a cut of t *below* v is a cut ρ of t such that $\Pi(root(t), v) \cap \rho = \emptyset$.

For a cut ρ of t, $tree(t, \rho)$ denotes the tree $t \circ \bigcup_{v \in \rho} sub_t(v)$.

A *binary tree* is a tree in which each internal node has exactly two direct descendants (recall that trees have no chains). A *node-labeled tree* t is a pair (t', η), where t' is a tree and $\eta : nd(t') \to \Sigma$ is a mapping, with Σ an alphabet. We say that t' is the *underlying tree of* t, denoted by $und(t)$, and η is the *node-labeling function of* t, denoted by lb_t. The notation and terminology concerning $und(t)$ carry over to t. Also, $yield(t) = lb_t(v_1) \cdots lb_t(v_n) \in \Sigma^+$, where $(v_1, \ldots, v_n) = \langle leaf \rangle(t)$. In this paper, by a *forest* we mean a sequence of trees. All notions concerning (node-labeled) trees carry over to (node-labeled) forests in the obvious way.

If we do not want to distinguish between isomorphic trees, then we can consider a *selector set* (*of trees*) which is a set of trees T such that, for each tree t, there exists $t' \in$ T isomorphic with t, and, moreover, for all distinct $t_1, t_2 \in$ T, t_1 is not isomorphic with t_2.

An *OS system* is like a context-free grammar, except that it does not have terminal symbols. Formally, an *OS* system is a triple (Σ, P, σ), where Σ is the (finite) alphabet of G, $P \subseteq \Sigma \times \Sigma\Sigma^+$ is the set of productions, and $\sigma \in \Sigma$ is the initial symbol. For $(a, x) \in P$ we use the notation $a \to x$. Note that we assume here that, for each production $a \to x \in P$, $|x| \geq 2$. An *unlimited OS system* is an *OS* system with infinitely many productions.

Let $G = (\Sigma, P, \sigma)$ be an (unlimited) *OS* system.

For words $x, y \in \Sigma^+$, $y \Rightarrow_G x$ means that y *directly derives* x (*in* G), $y \Rightarrow_G^* x$ means that y *derives* x (*in* G), and $y \Rightarrow_G^+ x$ means that y *derives* x (*in* G) *in at least one step.*

A sequence of words $y = x_0, x_1, \ldots, x_n = x$, $n \geq 0$, such that $x_{i-1} \Rightarrow_G x_i$ for $i = 1, \ldots, n$, is a *derivation of* x *from* y (*in* G).

For a node-labeled tree t, and $v \in in(t)$, the *production for* v in t, denoted by $prod_t(v)$, is the production $lb_t(v) \to lb_t(v_1) \cdots lb_t(v_n)$, where $(v_1, \ldots, v_n) = \langle ddes \rangle_t(v)$. Then *the set of productions for* t, denoted by $Prod(t)$, is the set $\{prod_t(v) \mid v \in in(t)\}$.

For $x \in \Sigma^+$ and $a \in \Sigma$, a *derivation tree of* x *from* a (*in* G) is a node-labeled tree t such that $lb_t(root(t)) = a$, $yield(t) = x$, and $Prod(t) \subseteq P$. If $a = \sigma$, then we say that t is a *derivation tree of* x (*in* G). A node-labeled tree is a *derivation tree* (*in* G) if it is a derivation tree of some $x \in \Sigma^+$. As usual, for $a \in \Sigma$ and $x \in \Sigma^+$, there is a derivation tree of x from a in G iff $a \Rightarrow_G^* x$.

Whenever the *OS* system G is clear from the context, we use $\Rightarrow$ and $\Rightarrow^*$ rather than $\Rightarrow_G$ and $\Rightarrow_G^*$.

An (unlimited) OS system $G = (\Sigma, P, \sigma)$ is *backwards deterministic* if $a \to x \in P$ and $a' \to x \in P$ imply that $a = a'$. It is *deterministic* if, for each $a \in \Sigma$, there exists exactly one production $a \to x \in P$.

In this paper, we assume that each OS system $G = (\Sigma, P, \sigma)$ is *reduced*, i.e., for each $a \in \Sigma$ there are $x, y \in \Sigma^*$ such that $\sigma \Rightarrow^* xay$ (a is *reachable*). It should be clear that for each OS system (Σ, P, σ) an equivalent reduced OS system can be constructed by removing from Σ all letters that are not reachable and from P all productions in which these letters occur.

1.1 Strict codes

In this section we consider grammatical codes for trees. First we present "strict codes", which were introduced in [5]. Then we give an alternative way to obtain strict codes by introducing "marked codes". Finally, looking at marked codes as grammars, we consider the parsing of these codes; this leads to the notion of a "non-overlapping code".

We begin by recalling the notion of a code and of a strict code, and we present some results from [5] concerning strict codes.

Definition 1.1.1 Let T be a selector set of trees, let Σ be a (finite) alphabet, and let $\varphi : \mathrm{T} \to \Sigma^*$.
(i) φ is *length-preserving* if, for all $t \in \mathrm{T}$, $|\varphi(t)| = \#leaf(t)$.
(ii) φ is *local* if there exists a mapping $\psi : \Sigma \times (\mathbf{N}_+ - \{1\}) \to \Sigma^+$ such that, for all $t_1, t_2 \in \mathrm{T}$, where $t_2 \in subs_{t_1}(i, n)$ for some $i \in \mathbf{N}_+$, $n \in \mathbf{N}_+ - \{1\}$, if $\varphi(t_1) = xay$ with $|x| = i - 1$ and $a \in \Sigma$, then $\varphi(t_2) = x\psi(a, n)y$.
(iii) φ is a *code* (*of* T) if φ is injective, length-preserving, and local.

Let $\varphi : \mathrm{T} \to \Sigma^*$ be a code. The value of φ for the one node tree in T is denoted by one_φ. A mapping ψ as above is called a *local function of* φ. Σ is called the *alphabet of* φ, denoted by $alph(\varphi)$. Note that one_φ and a local function ψ of φ determine φ.

From now on, T is a fixed but arbitrary selector set of trees. Also, we assume for each code φ that for each $a \in alph(\varphi)$ there exist $x, y \in alph(\varphi)^*$ such that $xay \in ran(\varphi)$; in other words, we assume that a code uses all letters of its alphabet. Consequently, each code has a unique local function.

Definition 1.1.2 Let $\varphi : \mathrm{T} \to \Sigma^*$ be a code with local function ψ.
(i) φ is *sibling consistent* if for all $x \in \Sigma^+$ and all $y, z \in ran(\varphi)$ such that $|x| = 2$, $y = y_1xy_2$, and $z = z_1xz_2$ for some $y_1, y_2, z_1, z_2 \in \Sigma^*$ with $|y_1| = i$ and $|z_1| = j$,
$(\langle leaf\rangle(\varphi^{-1}(y))(i+1), \langle leaf\rangle(\varphi^{-1}(y))(i+2))$ is a sibling segment of $\langle leaf\rangle(\varphi^{-1}(y))$ iff
$(\langle leaf\rangle(\varphi^{-1}(z))(j+1), \langle leaf\rangle(\varphi^{-1}(z))(j+2))$ is a sibling segment of $\langle leaf\rangle(\varphi^{-1}(z))$.
(ii) φ is *completeness consistent* if
(1) for all $x \in \Sigma^+$ and all $y, z \in ran(\varphi)$ such that $|x| = n$, $y = y_1xy_2$, $z = z_1xz_2$ for some $n \geq 2$, $y_1, y_2, z_1, z_2 \in \Sigma^*$ with $|y_1| = i$ and $|z_1| = j$,
$(\langle leaf\rangle(\varphi^{-1}(y))(i+1), \ldots, \langle leaf\rangle(\varphi^{-1}(y))(i+n))$ is a complete segment of $\langle leaf\rangle(\varphi^{-1}(y))$ iff
$(\langle leaf\rangle(\varphi^{-1}(z))(j+1), \ldots, \langle leaf\rangle(\varphi^{-1}(z))(j+n))$ is a complete segment of $\langle leaf\rangle(\varphi^{-1}(z))$, and

(2) φ is *locally injective*, i.e., for all $a, b \in \Sigma$ and all $n \in \mathbf{N}_+ - \{1\}$, $\psi(a,n) = \psi(b,n)$ implies $a = b$.
(iii) φ is *rich* if for each $x \in \Sigma^+$ there exist $y, z \in \Sigma^+$ such that $yxz \in ran(\varphi)$.
(iv) φ is *strict* if φ is sibling consistent, completeness consistent, and rich.

A strict code induces a special kind of partition of its alphabet.

Definition 1.1.3 Let φ be a code and let $a \in alph(\varphi)$.
(1) a is *left* (*w.r.t* φ) if there exist $x \in ran(\varphi)$ and $1 \leq i \leq |x|$, such that $x(i) = a$ and $\langle leaf \rangle(\varphi^{-1}(x))(i)$ is a leftmost child.
(2) a is *right* (*w.r.t* φ) if there exist $x \in ran(\varphi)$ and $1 \leq i \leq |x|$, such that $x(i) = a$ and $\langle leaf \rangle(\varphi^{-1}(x))(i)$ is a rightmost child.
(3) a is *middle* (*w.r.t* φ) if there exist $x \in ran(\varphi)$ and $1 \leq i \leq |x|$, such that $x(i) = a$ and $\langle leaf \rangle(\varphi^{-1}(x))(i)$ is a middle child.

We use L_φ, R_φ, and M_φ to denote the sets of left, right, and middle letters respectively.

Proposition 1.1.4 *Let φ be a strict code.*
(1) *$\{L_\varphi, M_\varphi, R_\varphi\}$ is a partition of $alph(\varphi)$.*
(2) *Either $\#M_\varphi = 1$, $\#L_\varphi = 3$, and $\#R_\varphi = 2$, or $\#M_\varphi = 1$, $\#L_\varphi = 2$, and $\#R_\varphi = 3$.*

By Proposition 1.1.4, for each strict code φ, $\#alph(\varphi) = 6$. For convenience, in what follows we restrict ourselves to the case that $one_\varphi = m$, where $\{m\} = M_\varphi$ – from now on m will be a reserved symbol used in this way.

Definition 1.1.5 Let $\varphi : \mathrm{T} \to \Sigma^*$ be a code, and let $x \in \Sigma^+$, where $|x| = n$. x is *complete* (*w.r.t.* φ) if, there exist $y \in ran(\varphi)$ and $i \geq 0$, such that $y(i+1) \cdots y(i+n) = x$, and $(\langle leaf \rangle(\varphi^{-1}(y))(i+1), \ldots, \langle leaf \rangle(\varphi^{-1}(y))(i+n))$ is a complete segment of $\langle leaf \rangle(\varphi^{-1}(y))$.

We use C_φ to denote the set of all complete words (w.r.t. φ) of Σ^*. Clearly, by Definition 1.1.3, $C_\varphi \subseteq L_\varphi M_\varphi^* R_\varphi$. If φ is a strict code, then $C_\varphi = L_\varphi M_\varphi^* R_\varphi$ (see [5, Lemma 2.7]). The following lemma was stated for strict codes in [5, Lemma 3.1], where it was used to prove Proposition 1.1.4(2). Here we give it for arbitrary codes that are locally injective.

Lemma 1.1.6 *Let φ be a code, and let ψ be the local function of φ.*
(i) *For each $a \in \Sigma$ and each $n \geq 2$, $|\psi(a,n)| = n$.*
(ii) *If φ is locally injective, then ψ is a bijection from $alph(\varphi) \times (\mathbf{N}_+ - \{1\})$ onto C_φ.*

Proof.
(i) Follows from the fact that φ is length-preserving and uses all letters of $alph(\varphi)$.
(ii) Since φ uses all letters of $alph(\varphi)$ and φ is local, it follows that, for each $a \in \Sigma$ and each $n \geq 2$, $\psi(a,n) \in C_\varphi$. Since φ is local, each complete word is of the form $\psi(a,n)$ for some $a \in \Sigma$ and $n \geq 2$. Hence ψ is onto C_φ.

Suppose that $\psi(a,n) = \psi(b,m)$ for some $a, b \in alph(\varphi)$, and $n, m \in \mathbf{N}_+ - \{1\}$. By (i) $n = |\psi(a,n)| = |\psi(b,m)| = m$. Since ψ is locally injective, it follows that $a = b$.

Hence ψ is a bijection from $alph(\varphi) \times (\mathbf{N}_+ - \{1\})$ onto C_φ. □

Strict codes can be described in terms of unlimited OS systems.

Definition 1.1.7 Let $G = (\Sigma, P, \sigma)$ be an unlimited OS system.
(1) G is *semi-deterministic* if, for each $a \in \Sigma$ and each $n \geq 2$, there exists exactly one production $a \to x$ in P such that $|x| = n$.
(2) G is *strict* if G is semi-deterministic, backwards deterministic, and there exists a partition of Σ into three sets L, M, R such that $M = \{\sigma\}$, either $\#L = 3$ and $\#R = 2$, or $\#L = 2$ and $\#R = 3$, and for each production $a \to x \in P$, $x \in LM^*R$.

In [5] it was shown that there is a one-to-one correspondence between strict codes and strict unlimited OS systems (see Proposition 1.1.12). Since we consider also other kinds of codes than strict codes in this paper, we would like to have such a correspondence between arbitrary codes and unlimited OS systems. Therefore, we state the results of [5] concerning this correspondence in more general terms here.

With each code we can associate a semi-deterministic unlimited OS system whose productions are given by the local function ψ. Conversely, with each semi-deterministic unlimited OS system we can associate a "code", by taking the yield of a derivation tree as the image of its underlying tree. Formally, this is not a code, since it might not be injective. Accordingly, we give definitions not in terms of codes but in terms of local and length-preserving mappings. All the same, for such mappings we use the same terminology as for codes.

Definition 1.1.8 Let $\varphi : \mathrm{T} \to \Sigma^*$ be a length-preserving and local mapping, and let ψ be the local function of φ. The *unlimited* OS *system induced by* φ, denoted $\mathrm{OS}(\varphi)$, is the unlimited OS system $(alph(\varphi), P, one_\varphi)$, where $P = \{a \to x \mid \psi(a, n) = x \text{ for some } n \geq 2\}$.

Definition 1.1.9 Let $G = (\Sigma, P, \sigma)$ be a semi-deterministic unlimited OS system, and let $t \in \mathrm{T}$.
(i) The *node-labeling of* t *induced by* G is the mapping $\eta : nd(t) \to \Sigma$ defined as follows:
$\eta(root(t)) = \sigma$;
if $v \in in(t)$ is such that $\langle ddes \rangle_t(v) = (v_1, \ldots, v_k)$ for $k \geq 2$, and $\eta(v) = a$, then $\eta(v_i) = x(i)$ for each $1 \leq i \leq k$, where x is such that $a \to x \in P$ and $|x| = k$.
(ii) The *mapping induced by* G, denoted COD_G, is the mapping of T into Σ^* defined as follows: for $t \in \mathrm{T}$, $\mathrm{COD}_G(t) = yield((t, \eta))$.

Lemma 1.1.10
(1) *For each length-preserving and local mapping* φ, $\mathrm{OS}(\varphi)$ *is a semi-deterministic unlimited OS system and* $\mathrm{COD}_{\mathrm{OS}(\varphi)} = \varphi$.
(2) *For each semi-deterministic unlimited OS system* G, COD_G *is length-preserving and local, and* $\mathrm{OS}(\mathrm{COD}_G) = G$.
(3) *For each length-preserving and local mapping* φ, φ *is locally injective iff* $\mathrm{OS}(\varphi)$ *is backwards deterministic.*

Proof.
(1) By Definition 1.1.1(ii), $\mathrm{OS}(\varphi)$ is semi-deterministic. By the construction of $\mathrm{OS}(\varphi)$, $\mathrm{COD}_{\mathrm{OS}(\varphi)}(t) = \varphi(t)$ for each $t \in \mathrm{T}$.
(2) Clearly, COD_G is length-preserving. Let $G = (\Sigma, P, \sigma)$, and let ψ be the mapping defined as follows : for each $a \in \Sigma$ and each $n \geq 2$, $\psi(a, n) = x$ such that $a \to x \in P$. Since ψ

obviously satisfies Definition 1.1.1(ii), COD_G is local, and ψ is the local function of COD_G. Clearly, $one_{\mathrm{COD}_G} = \sigma$. It follows that $\mathrm{OS}(\mathrm{COD}_G) = G$.
(3) Follows immediately from the definition of the productions in $\mathrm{OS}(\varphi)$. □

Remark 1.1.11 We use the following notations : for a tree $t \in \mathrm{T}$ and a local and length-preserving mapping φ, $t[\varphi]$ denotes the node-labeled tree (t, η), where η is the node-labeling induced by $\mathrm{OS}(\varphi)$. Hence $lb_{t[\varphi]}$ denotes η. Note that $t[\varphi]$ is a derivation tree in $\mathrm{OS}(\varphi)$. In fact, it is the unique derivation tree with t as its underlying tree. Hence, $yield(t[\varphi]) = \varphi(t)$.

For technical reasons, when we consider OS systems, we will assume that the underlying tree of each derivation tree is in the selector set T. Hence, for each derivation tree t in $\mathrm{OS}(\varphi)$, $t = und(t)[\varphi]$.

Note that it also follows from the above constructions that if φ uses all letters of its alphabet, then $\mathrm{OS}(\varphi)$ is reduced, and if G is reduced, then COD_G uses all letters of its alphabet. □

Lemma 1.1.10 establishes a one-to-one correspondence between codes and semi-deterministic OS systems for which the induced mapping is injective. This correspondence relates strict codes with strict OS systems.

Proposition 1.1.12
(1) *For each strict code φ, $\mathrm{OS}(\varphi)$ is a strict unlimited OS system.*
(2) *For each strict unlimited OS system G, COD_G is a strict code.*

Hence we may specify strict codes in the form of strict unlimited OS systems.

Example 1.1.13 Consider the unlimited OS system $G = (\Sigma, P, m)$, where $\Sigma = \{\ell_1, \ell_2, \ell_3, m, r_1, r_2\}$, and P consists of the productions, for $k \geq 0$,

$$\ell_1 \to \ell_1 m^k r_1, \qquad m \to \ell_2 m^k r_2,$$
$$\ell_2 \to \ell_1 m^k r_2, \qquad r_1 \to \ell_3 m^k r_1,$$
$$\ell_3 \to \ell_2 m^k r_1, \qquad r_2 \to \ell_3 m^k r_2.$$

Clearly, G is semi-deterministic, backwards deterministic, and $L = \{\ell_1, \ell_2, \ell_3\}$, $M = \{m\}$, and $R = \{r_1, r_2\}$ satisfy the conditions in Definition 1.1.7(2). Hence G is a strict OS system, and for convenience, we also say that G is a strict code. □

This concludes our recalling of results from [5].

Now we consider strict codes from another point of view. We distinguish a priori left, middle, and right letters in the alphabet. This leads to the notion of a *marked code*. We will show that strict codes are then precisely marked codes with an alphabet of minimal size.

Definition 1.1.14 A code φ is *marked* if
(i) φ is locally injective, and
(ii) L_φ, R_φ, and M_φ are mutually disjoint.

It follows from Definition 1.1.2(ii.2) and Proposition 1.1.4(1) that each strict code is marked. Because we wish to make further comparisons between marked and strict codes, and since we have restricted ourselves to strict codes φ with $one_\varphi = m$, we assume for each marked code φ that $one_\varphi \in M_\varphi$. Marked codes can also be described in terms of unlimited OS systems.

Definition 1.1.15 An unlimited OS system $G = (\Sigma, P, \sigma)$ is *marked* if it is semi-deterministic, backwards deterministic, and there are three mutually disjoint subsets L, M, R of Σ such that, for each production $a \to x \in P$, $x \in LM^*R$ and $\sigma \in M$.

Clearly, each strict OS system is marked, but there exist marked OS systems that are not strict, as the following example shows.

Example 1.1.16 Consider the unlimited OS system $G = (\Sigma, P, m)$, where $\Sigma = \{\ell_1, \ell_2, \ell_3, m, r_1, r_2, r_3\}$, and P consists of the productions, for $k \geq 0$,

$$\begin{array}{ll} \ell_1 \to \ell_1 m^k r_1, & r_1 \to \ell_3 m^k r_3, \\ \ell_2 \to \ell_2 m^k r_1, & r_2 \to \ell_3 m^k r_1, \\ \ell_3 \to \ell_1 m^k r_2, & r_3 \to \ell_1 m^k r_3. \\ m \to \ell_2 m^k r_3, & \end{array}$$

Clearly, G is semi-deterministic, backwards deterministic, and $L = \{\ell_1, \ell_2, \ell_3\}$, $M = \{m\}$, and $R = \{r_1, r_2, r_3\}$ satisfy the conditions in Definition 1.1.15. Hence G is a marked OS system. However, G is not strict, because $\#\Sigma > 6$. Note that in particular COD_G is not rich : the word $\ell_2 r_2$ does not occur in the yield of any derivation tree in G. □

There is a one-to-one correspondence between marked codes and marked OS systems, analogous to Proposition 1.1.12. The fact that the mapping induced by a marked OS system is injective follows from the fact that this mapping is completeness consistent. We will return to this later in detail, when we consider the parsing of marked OS systems.

First we will establish the precise relationship between marked and strict codes.

A marked code φ is *minimal* if $\#alph(\varphi) \leq \#alph(\varphi')$ for every marked code φ'.

We will show that the class of minimal marked codes is exactly the class of strict codes (Theorem 1.1.19).

Lemma 1.1.17 *For each marked code φ, $\#alph(\varphi) \geq 6$.*

Proof. Let φ be a marked code.

Define $C_\varphi^2 = \{x \in C_\varphi \mid |x| = 2\}$. Note that $C_\varphi^2 \subseteq L_\varphi R_\varphi$. By Lemma 1.1.6(ii), $\#alph(\varphi) = \#C_\varphi^2$. Thus, $\#alph(\varphi) \leq \#L_\varphi \#R_\varphi$, and, since L_φ, M_φ, and R_φ are disjoint, $\#alph(\varphi) > \#L_\varphi + \#R_\varphi$.

It is easily seen that if $x, y \in \mathbf{N}_+$ are such that $xy > x + y$, then either $x \geq 2$ and $y > 2$, or $x > 2$ and $y \geq 2$. Consequently the above implies that either $\#L_\varphi \geq 2$ and $\#R_\varphi > 2$, or $\#L_\varphi > 2$ and $\#R_\varphi \geq 2$. Hence, $\#alph(\varphi) > \#L_\varphi + \#R_\varphi \geq 5$. □

We have already observed that each strict code is a marked code, which has an alphabet of 6 letters. Hence, by Lemma 1.1.17, each strict code is a minimal marked code. Moreover it

follows that minimal marked codes are exactly those marked codes that have an alphabet of 6 letters.

We will show now that the unlimited OS system of a minimal marked code is strict. Hence, the class of minimal marked codes is exactly the class of strict codes.

Lemma 1.1.18 *Let φ be a minimal marked code. Then* $\mathrm{OS}(\varphi)$ *is strict.*

Proof. By Lemma 1.1.10(1) $\mathrm{OS}(\varphi)$ is semi-deterministic, and by Lemma 1.1.10(3) $\mathrm{OS}(\varphi)$ is backwards deterministic. In the proof of Lemma 1.1.17 it is shown that either $\#L_\varphi \geq 2$ and $\#R_\varphi > 2$, or $\#L_\varphi > 2$ and $\#R_\varphi \geq 2$. By Definition 1.1.14(ii), $\#alph(\varphi) \geq \#L_\varphi + \#M_\varphi + \#R_\varphi$. Since $\#alph(\varphi) = 6$, it follows that either $\#L_\varphi = 2$, $\#R_\varphi = 3$, and $\#M_\varphi = 1$, or $\#L_\varphi = 3$, $\#R_\varphi = 2$, and $\#M_\varphi = 1$. Since by definition $one_\varphi \in M_\varphi$, $M_\varphi = \{one_\varphi\}$. Hence L_φ, M_φ, and R_φ satisfy the conditions in Definition 1.1.7(2), because, for each production $a \to x$ of $\mathrm{OS}(\varphi)$, if ψ is the local function of φ, then $x = \psi(a, |x|) \in C_\varphi$, and so $x \in L_\varphi M_\varphi^* R_\varphi$. Consequently, $\mathrm{OS}(\varphi)$ is strict. □

Theorem 1.1.19 *A code φ is strict iff φ is a minimal marked code.*

Proof. As observed before, if φ is strict, then φ is a minimal marked code.

Suppose that φ is a minimal marked code. By Lemma 1.1.18, $\mathrm{OS}(\varphi)$ is strict. By Lemma 1.1.10, $\mathrm{COD}_{\mathrm{OS}(\varphi)} = \varphi$, and then, by Proposition 1.1.12(2), φ is a strict code. □

We now turn to the parsing of (marked) OS systems.

Definition 1.1.20 Let $G = (\Sigma, P, \sigma)$ be an unlimited OS system. A word $x \in \Sigma^+$ is an *origin* (w.r.t. G) if, for each $y \in \Sigma^+$, $y \Rightarrow^* x$ implies that $y = x$.

Hence an origin is a word that is not derivable from any other word, and so it does not contain an occurrence of the right-hand side of any production.

To prove that COD_G is injective for a marked OS system G (see the observation following Example 1.1.16), we will show that G is unambiguous, in the sense that there is a unique derivation tree for every word x generated by G. In fact, we will even show that G has the nice property that *every* $x \in \Sigma^+$ (not only those generated by G) has a unique origin and a unique derivation forest from it. As we will see, this property of a marked OS system is, in essence, caused by the fact that the right-hand sides of its productions are non-overlapping. For this reason, we will consider OS systems (and the corresponding codes) with non-overlapping right-hand sides. It will turn out that the above property is precisely characterized by such non-overlapping OS systems.

Definition 1.1.21
(1) Let Σ be an alphabet, and $x, y \in \Sigma^+$ (not necessarily distinct). Then x and y are *overlapping* if there exists $v \in \Sigma^+$ such that $x = u_1 v w_1$, $y = u_2 v w_2$, $u_1 w_1 u_2 w_2 \neq \Lambda$, and $u_i w_j = \Lambda$ for some $i, j \in \{1, 2\}$.
(2) A code φ is *non-overlapping* if φ is locally injective, and, for all $a, b \in alph(\varphi)$, and all $n, m \geq 2$, $\psi(a, n)$ and $\psi(b, m)$ are not overlapping, where ψ is the local function of φ.
(3) An unlimited OS system is *non-overlapping* if it is semi-deterministic, backwards deterministic, and, for all productions $a \to x$ and $b \to y$, x and y are not overlapping.

Note that according to Definition 1.1.21(1) a word may or may not overlap itself.

Example 1.1.22 Let $\Sigma = \{a, b\}$, and let $x = aabb$, $y = abab$, $z = abbb$. Then x and z are overlapping (take $v = abb$, $x = u_1vw_1$, $z = u_2vw_2$, then $u_2w_1 = \Lambda$), y and z are overlapping, and x and y are not overlapping.

Furthermore, x is not overlapping itself; y is overlapping itself (take $v = ab$, $u_1 = \Lambda$, $w_1 = ab$, $u_2 = ab$, $w_2 = \Lambda$, then $u_1vw_1 = y = u_2vw_2$, and $u_1w_2 = \Lambda$). Also, z is not overlapping itself. □

Clearly, each marked code is a non-overlapping code, and each marked OS system is a non-overlapping OS system. Not all non-overlapping OS systems are marked, as the following example shows.

Example 1.1.23 Consider the unlimited OS system $G = (\Sigma, P, m)$, where $\Sigma = \{\ell_1, \ell_2, \ell_3, m, r_1, r_2, c\}$, and P consists of the productions, for $k \geq 0$,

$$\begin{array}{ll} \ell_1 \to cm^kr_1, & r_1 \to \ell_3m^kr_1, \\ \ell_2 \to \ell_1m^kr_2, & r_2 \to \ell_3cm^kr_2, \\ \ell_3 \to \ell_2m^kr_1, & r_2 \to \ell_3r_2, \\ m \to \ell_2m^kr_2, & c \to \ell_1m^kr_1. \end{array}$$

Clearly, G is semi-deterministic and backwards deterministic, and no right-hand sides are overlapping. Hence G is a non-overlapping code.

However, G is not marked, because if L, M, R are subsets of Σ such that $x \in LM^*R$ for each $a \to x \in P$, then L, M, R are *not* disjoint, since $c \in L \cap M$. □

The following lemma gives a basic technical property of non-overlapping OS systems. It strengthens the notion of completeness consistency and implies the "unique origin property".

Lemma 1.1.24 *Let $G = (\Sigma, P, \sigma)$ be a non-overlapping OS system. For all $y, u \in \Sigma^+$, $w, w' \in \Sigma^*$, and $a \in \Sigma$, if y is an origin such that $y \Rightarrow^* wuw'$ with corresponding derivation forest f, and $a \to u \in P$, then $(z(i+1), \ldots, z(i+n))$ is a complete segment of z, where $z = \langle leaf \rangle(f)$, $i = |w|$, and $n = |u|$, and the direct ancestor of this complete segment is labeled a.*

Proof. Let $y, u \in \Sigma^+$, $w, w' \in \Sigma^*$, and $a \in \Sigma$ be such that y is an origin, $y \Rightarrow^* wuw'$, and $a \to u \in P$. Let f be the corresponding derivation forest of wuw' from y. Let $z = \langle leaf \rangle(f)$, $i = |w|$, and $n = |u|$, and let $v_k = z(i+k)$ for $k = 1, \ldots, n$. Thus $u = lb_f(v_1) \cdots lb_f(v_n)$.

Since y is an origin, y does not contain u. Hence there exists $k \in \{1, \ldots, n\}$ such that $v_k \notin root(f)$. Let $m = \min\{k \in \{1, \ldots, n\} \mid level(v_k) \geq level(v_j) \text{ for all } j \in \{1, \ldots, n\}\}$. Note that either v_m is a leftmost child or $m = 1$, and that $v_m \notin root(f)$. Let v be the direct ancestor of v_m in f, and let $\langle ddes \rangle_f(v) = (w_1, \ldots, w_s)$, $s \geq 2$. Since f is a derivation forest in G, there is a production $b \to u'$ in P such that $b = lb_f(v)$ and $u' = lb_f(w_1) \cdots lb_f(w_s)$.

Let $j \in \{1, \ldots, s\}$ be such that $v_m = w_j$ (thus, $j = 1$ or $m = 1$). We will prove by induction that $v_{m+k} = w_{j+k}$ for all $k = 0, \ldots, \min(n - m, s - j)$. By definition, $v_m = w_j$. Suppose that $v_m = w_j, v_{m+1} = w_{j+1}, \ldots, v_{m+k-1} = w_{j+k-1}$. If $v_{m+k} \neq w_{j+k}$, then there is a path of length ≥ 1 from w_{j+k} to v_{m+k}. But then $level(v_{m+k}) > level(v_m)$, which contradicts the definition of m. Consequently, $v_{m+k} = w_{j+k}$ and the induction is completed.

Hence $(v_m, \ldots, v_{m+r}) = (w_j, \ldots, w_{j+r})$, where $r = \min(n-m, s-j)$. Define $t, u_1, u_2, u_1', u_2' \in \Sigma^*$ as follows:

$$t = lb_f(v_m) \cdots lb_f(v_{m+r}) = lb_f(w_j) \cdots lb_f(w_{j+r})$$
$$u_1 = lb_f(v_1) \cdots lb_f(v_{m-1}),$$
$$u_2 = lb_f(v_{m+r+1}) \cdots lb_f(v_n),$$
$$u_1' = lb_f(w_1) \cdots lb_f(w_{j-1}), \text{ and}$$
$$u_2' = lb_f(w_{j+r+1}) \cdots lb_f(w_s).$$

Hence $u = u_1 t u_2$ and $u' = u_1' t u_2'$.
Note that $m = 1$ or $j = 1$, and $m + r = n$ or $j + r = s$.

Claim 1.1.25 *$m = j = 1$ and $n = m + r = j + r = s$.*
Proof. Assume to the contrary that $m \neq j$ or $n \neq s$. Then $u_1 u_2 u_1' u_2' \neq \Lambda$. Since G is non-overlapping, u and u' are not overlapping. However, since $m = 1$ or $j = 1$ we have that $u_1 = \Lambda$ or $u_1' = \Lambda$, and since $m + r = n$ or $j + r = s$ we have that $u_2 = \Lambda$ or $u_2' = \Lambda$. Consequently one of the words $u_1 u_2, u_1' u_2', u_1 u_2', u_1' u_2$ is the empty word, which contradicts the fact that u and u' are not overlapping. □

Hence, by Claim 1.1.25, $(w_1, \ldots, w_s) = (v_1, \ldots, v_n)$, and $u = u'$. Since G is backwards deterministic it follows that $a = b$.

Consequently, $(v_1, \ldots, v_n) = (z(i+1), \ldots, z(i+n))$ is a complete segment of z, with direct ancestor v, which is labeled a. □

Now we are able to prove that non-overlapping OS systems do have the property that each word has a unique origin and a unique derivation forest from it. Moreover, they are characterized by this property.

Theorem 1.1.26 *Let $G = (\Sigma, P, \sigma)$ be a semi-deterministic OS system. Then G is non-overlapping iff for each $x \in \Sigma^+$ there exist a unique origin y such that $y \Rightarrow^* x$ and a unique derivation forest of x from y.*

Proof. (Only if) Let $G = (\Sigma, P, \sigma)$ be a non-overlapping OS system.

For each $x \in \Sigma^+$ the productions of G can be applied backwards, and each application shortens the word. Hence, for each $x \in \Sigma^+$, there exist an origin y such that $y \Rightarrow^* x$ and a derivation forest f of x from y. It remains to be proved that y and f are unique. We will prove this by induction on $|x|$.

If $|x| = 1$, then x is its own unique origin with a unique corresponding derivation forest, since there are no chain productions.

Suppose now that each word $x \in \Sigma^*$ with $|x| \leq n$, $n \geq 1$, has a unique origin and a unique derivation forest from it. Let $x \in \Sigma^*$ be such that $|x| = n + 1$. If x is an origin, then it has itself as unique origin and there is a unique corresponding derivation forest. If x is not an origin, then x contains a right-hand side of a production of G. Let $x = wuw'$, where u is a right-hand side, and $w, w' \in \Sigma^*$.

Let y, y' be origins such that $y \Rightarrow^* x$ and $y' \Rightarrow^* x$. Let f be a derivation forest of x from y and let f' be a derivation forest of x from y'. Let $z = \langle leaf \rangle(f)$, $z' = \langle leaf \rangle(f')$, $i = |w|$, and $n = |u|$. By Lemma 1.1.24, $(z(i+1), \ldots, z(i+n))$ is a complete segment of z, and $(z'(i+1), \ldots, z'(i+n))$ is a complete segment of z'. Let $v \in in(f)$ and $v' \in in(f')$ be the direct ancestors of these complete segments, respectively. Then, by Lemma 1.1.24, v and v' are both labeled by a, where a is such that $a \to u \in P$. Then $f \triangleq sub_f(v)$ is a derivation forest for waw' from y, and $f' \triangleq sub_{f'}(v')$ is a derivation forest for waw' from y'.

Since G is chain-free, $|waw'| < |wuw'|$, and hence by the inductive assumption waw' has a unique origin and a unique derivation forest from it. Consequently, $y = y'$ and $f \triangleq sub_f(v) = f' \triangleq sub_{f'}(v')$. Since v is the $(i+1)$th leaf of $f \triangleq sub_f(v)$, and v' is the $(i+1)$th leaf of $f' \triangleq sub_{f'}(v')$, it follows that $v = v'$, and because f and f' are both obtained by adding to v a segment of n leaves labeled by u, $f = f'$. Hence x has a unique origin y and a unique derivation forest from y. This completes the induction proof.

(If) Let $G = (\Sigma, P, \sigma)$ be a semi-deterministic OS system such that, for each $x \in \Sigma^+$, there exist a unique origin y such that $y \Rightarrow^* x$ and a unique derivation forest from y. We will prove that G is non-overlapping. Clearly, G is backwards deterministic, since each right-hand side has a unique origin. It remains to be proved that no right-hand sides of G are overlapping.

Assume to the contrary that there exist $a \to x$, $b \to y$ in P such that $x = u_1 v w_1$ and $y = u_2 v w_2$, with $v \neq \Lambda$, $u_1 w_1 u_2 w_2 \neq \Lambda$, and $u_i w_j = \Lambda$ for some $i, j \in \{1, 2\}$. We will obtain a contradiction in all of these four cases. It is sufficient to consider the cases $u_1 w_1 = \Lambda$ and $u_2 w_1 = \Lambda$; the other two cases ($u_2 w_2 = \Lambda$ and $u_1 w_2 = \Lambda$) follow by symmetric arguments. If $u_1 w_1 = \Lambda$, then x is a subword of y. Clearly, the origin of x is a, and the origin of y is b, and the derivation tree t of y from b is such that $ddes_t(root(t)) = leaf(t)$. Also, the origin of $u_2 a w_2$ is b. Let t' be the derivation tree of $u_2 a w_2$ from b. Let t'' be the tree that results from t' by adding $|x|$ labeled nodes as direct descendants of the leaf in t' with label a. Then, since $u_2 w_2 \neq \Lambda$, t'' is a derivation tree of y from b different from t. This contradicts the fact that y has a unique derivation tree from its origin b.

Now suppose that $u_2 w_1 = \Lambda$. Let z be the unique origin of $a w_2$ in G. Then z is also the origin of $u_1 v w_2$. Similarly, the origin of $u_1 b$ is the origin of $u_1 v w_2$. Hence $a w_2$, $u_1 b$, and $u_1 v w_2$ all have the same origin z.

Let f_1 be the unique derivation forest of $a w_2$ from z, and let f_1' be the forest that results from f_1 by adding $|x|$ labeled nodes as direct descendants of the leaf in f_1 with label a, such that the new segment is labeled x. Then f_1' is a derivation forest of $x w_2 = u_1 v w_2$ from z.

Analogously, given the derivation forest f_2 of $u_1 b$ from z, we can construct a derivation forest f_2' of $u_1 y = u_1 v w_2$ from z by adding a segment labeled y to the node that is labeled b. Clearly, since $u_1 w_2 \neq \Lambda$, $f_1' \neq f_2'$. Hence we have constructed two different derivation forests of $u_1 v w_2$ from z, a contradiction.

Consequently, G is non-overlapping. □

Remark 1.1.27 For the only-if part of the proof, we might also consider the semi-Thue system formed by the production rules of a OS system in reverse direction. Then the semi-Thue system corresponding to a non-overlapping OS system has no critical pairs, and hence is confluent; since there are no chain productions, it is also length-terminating. From this it

follows (see, e.g., [11]) that each word has a unique normal form, i.e., a unique origin. Here we have proved more, namely that also the derivation forest from the origin is also unique. □

Now we can prove that the mapping induced by a non-overlapping OS system is injective.

Corollary 1.1.28 *For each non-overlapping OS system G, COD_G is injective.*

Proof. Let $\varphi = \mathrm{COD}_G$, and let $t_1, t_2 \in \mathrm{T}$. Then $t_1[\varphi]$ and $t_2[\varphi]$ are derivation trees in $\mathrm{OS}(\varphi) = G$. Furthermore, $\varphi(t_1) = \mathit{yield}(t_1[\varphi])$ and $\varphi(t_2) = \mathit{yield}(t_2[\varphi])$. Hence, by Theorem 1.1.26, if $\varphi(t_1) = \varphi(t_2)$, then $t_1[\varphi] = t_2[\varphi]$ and hence $t_1 = t_2$. □

From Corollary 1.1.28 we obtain that there is a one-to-one correspondence between non-overlapping codes and non-overlapping OS systems (cf. Lemma 1.1.10 and Proposition 1.1.12), and similarly between marked codes and marked OS systems.

Theorem 1.1.29
(1) *For each non-overlapping code φ, $\mathrm{OS}(\varphi)$ is a non-overlapping unlimited OS system.*
(2) *For each non-overlapping unlimited OS system G, COD_G is a non-overlapping code.*

Proof. (1) By Lemma 1.1.10(1) $\mathrm{OS}(\varphi)$ is semi-deterministic, and by Lemma 1.1.10(3), $\mathrm{OS}(\varphi)$ is backwards deterministic. Note that for each production $a \to x$ of $\mathrm{OS}(\varphi)$, $x = \psi(a, |x|)$, where ψ is the local function of φ. Hence for all productions $a \to x$ and $b \to y$ in $\mathrm{OS}(\varphi)$, x and y are not overlapping, since φ is non-overlapping.

Hence $\mathrm{OS}(\varphi)$ is non-overlapping.
(2) By Lemma 1.1.10(2) COD_G is local and length-preserving. By Corollary 1.1.28 COD_G is injective. Hence COD_G is a code. By Lemma 1.1.10(3), COD_G is locally injective. For the local function ψ of COD_G, $\psi(a, n)$ is a right-hand side of a production in G for each $a \in \Sigma$ and each $n \geq 2$. Hence, for all $a, b \in \Sigma$ and $n, m \geq 2$, $\psi(a, n)$ and $\psi(b, m)$ are not overlapping.

Hence COD_G is non-overlapping. □

Theorem 1.1.30
(1) For each marked code φ, $\mathrm{OS}(\varphi)$ is a marked unlimited OS system.
(2) For each marked unlimited OS system G, COD_G is a marked code.

Proof.
(1) By Theorem 1.1.29(1), $\mathrm{OS}(\varphi)$ is semi-deterministic and backwards deterministic. For each production $a \to x$ of $\mathrm{OS}(\varphi)$, $x = \psi(a, |x|) \in L_\varphi M_\varphi^* R_\varphi$, where ψ is the local function of φ. Hence L_φ, M_φ, and R_φ satisfy the conditions in Definition 1.1.15. Consequently, $\mathrm{OS}(\varphi)$ is marked.
(2) By Theorem 1.1.29(2), COD_G is a non-overlapping code. Hence COD_G is a locally injective code. If L, M, and R are sets as in Definition 1.1.15, then $L_{\mathrm{COD}_G} \subseteq L$, $R_{\mathrm{COD}_G} \subseteq R$, and $M_{\mathrm{COD}_G} \subseteq M$. Hence L_{COD_G}, M_{COD_G}, and R_{COD_G} are disjoint. This shows that COD_G is a marked code. □

It can be shown, using arguments in the spirit of the proof of Lemma 1.1.17, that each non-overlapping code has an alphabet of at least 6 letters.

By minimizing marked codes, we obtained strict codes (Theorem 1.1.19). By Example 1.1.23, there exist non-overlapping codes that are not marked. The following example shows that if we minimize non-overlapping codes, we can still obtain codes that are not marked. Hence not all minimal non-overlapping codes are strict.

Example 1.1.31 Consider the unlimited OS system $G = (\Sigma, P, \sigma)$, where $\Sigma = \{\ell_1, \ell_2, \ell_3, r_1, r_2, r_3\}$, $\sigma = r_3$, and P consists of the productions, for $k \geq 0$,

$$\begin{aligned} \ell_1 &\to \ell_1 r_3^k r_1, & r_1 &\to \ell_2 r_1^k r_3, \\ \ell_2 &\to \ell_1 r_3^k r_2, & r_2 &\to \ell_3 r_2^k r_1, \\ \ell_3 &\to \ell_2 r_1^k r_2, & r_3 &\to \ell_3 r_2^k r_3. \end{aligned}$$

Clearly, G is semi-deterministic and backwards deterministic, and no right-hand sides are overlapping. Hence G is a non-overlapping code.

However, G is not marked. □

We have that non-overlapping (marked, strict) codes can be seen as non-overlapping (marked, strict) OS systems. Hence for a non-overlapping code φ, when we speak of "an origin w.r.t φ", it is an origin w.r.t OS(φ). The set of origins of a non-overlapping code φ is denoted by OR_φ. This is the set of words that do not contain complete subwords. Hence if φ is a strict code, then $OR_\varphi = (R_\varphi \cup M_\varphi)^*(L_\varphi \cup M_\varphi)^* - \{\Lambda\}$, since $C_\varphi = L_\varphi M_\varphi^* R_\varphi$.

If $\varphi : \mathrm{T} \to \Sigma^*$ is a non-overlapping code, then by Theorem 1.1.26, one can assign to each word over Σ its (composite) category.

Definition 1.1.32 Let $\varphi : \mathrm{T} \to \Sigma^*$ be a non-overlapping code, and let $x \in \Sigma^+$. The *composite category of* x, denoted by $cat_\varphi(x)$, is the unique origin of x. If $|cat_\varphi(x)| = 1$, then it is called the *category of* x.

Example 1.1.33 Let φ be the strict code such that for all $k \geq 0$,

$$\begin{aligned} m &\to \ell_1 m^k r_3, & r_1 &\to \ell_2 m^k r_3, \\ \ell_1 &\to \ell_1 m^k r_2, & r_2 &\to \ell_2 m^k r_2, \\ \ell_2 &\to \ell_1 m^k r_1, & r_3 &\to \ell_2 m^k r_1. \end{aligned}$$

If $x = \ell_1 m r_2 \ell_1 \ell_1 m r_3 \ell_2 r_3$, then $cat_\varphi(x) = \ell_1 \ell_2$, and the derivation forest f of x from $\ell_1 \ell_2$ is as shown in Figure 1.1.

If $x' = \ell_2 r_1 r_2 \ell_1 m r_3$, then $cat_\varphi(x') = r_3 r_2 m$, and the derivation forest f' of x' from $r_3 r_2 m$ is as shown in Figure 1.2. □

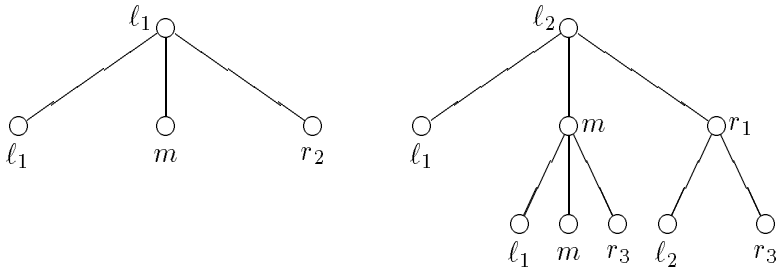


Figure 1.1: derivation forest of $x = \ell_1 m r_2 \ell_1 \ell_1 m r_3 \ell_2 r_3$ from $\ell_1 \ell_2$

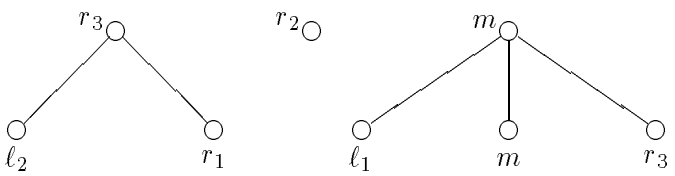


Figure 1.2: derivation forest of $x' = \ell_2 r_1 r_2 \ell_1 m r_3$ from $r_3 r_2 m$

The following theorem says that the composite category of a word can be computed by first computing the composite categories of subwords comprising the word.

Theorem 1.1.34 *Let $\varphi : \mathrm{T} \to \Sigma^*$ be a non-overlapping code, and let $x \in \Sigma^+$. If $x = x_1 \cdots x_n$ with $x_i \in \Sigma^+$, then $cat_\varphi(x) = cat_\varphi(cat_\varphi(x_1) \cdots cat_\varphi(x_n))$.*

Proof. Let $y_i = cat_\varphi(x_i)$ for all $i = 1, \ldots, n$. Let $y = cat_\varphi(x)$ and $y' = cat_\varphi(y_1 \cdots y_n)$. We must prove that $y = y'$.

We have that $y \in OR_\varphi$, $y \Rightarrow^* x$, $y' \in OR_\varphi$, $y' \Rightarrow^* y_1 \cdots y_n$, and $y_i \Rightarrow^* x_i$ for all $i = 1, \ldots, n$. So $y' \Rightarrow^* x_1 \cdots x_n = x$.

By Theorem 1.1.26, x has a unique origin. Hence, $y = y'$, and $cat_\varphi(x) = y = y' = cat_\varphi(y_1 \cdots y_n) = cat_\varphi(cat_\varphi(x_1) \cdots cat_\varphi(x_n))$. □

Example 1.1.35 (*Example 1.1.33 continued.*) Let φ and x be as in Example 1.1.33. Let $x_1 = \ell_1 m r_2 \ell_1$, and $x_2 = \ell_1 m r_3 \ell_2 r_3$. Then $x = x_1 x_2$, $cat_\varphi(x_1) = \ell_1 \ell_1$, and $cat_\varphi(x_2) = m r_1$. Hence $cat_\varphi(cat_\varphi(x_1) cat_\varphi(x_2)) = cat_\varphi(\ell_1 \ell_1 m r_1) = \ell_1 \ell_2 = cat_\varphi(x)$. □

Corollary 1.1.36 *Let $\varphi : \mathrm{T} \to \Sigma^*$ be a non-overlapping code.*

For each $w \in \Sigma^+$, each $k \in \mathbf{N}_+$, all $w_1, \ldots, w_k \in \Sigma^+$ such that $w = w_1 \cdots w_k$, and each tree t with $\#leaf(t) = k$, there is a unique labeling $\kappa : nd(t) \to OR_\varphi$ such that

(i) $\kappa(\langle leaf \rangle(t)(i)) = cat_\varphi(w_i)$ for $i = 1, \ldots, k$,

(ii) for each $v \in in(t)$, $\kappa(v) = cat_\varphi(\kappa(v_1) \cdots \kappa(v_n))$, where $(v_1, \ldots, v_n) = \langle ddes \rangle_t(v)$, and

(iii) $\kappa(root(t)) = cat_\varphi(w)$.

Proof. Let $w \in \Sigma^+$, $k \in \mathbf{N}_+$, and $w_1, \ldots, w_k \in \Sigma^+$ be such that $w = w_1 \cdots w_k$, and let t be a tree with $\#leaf(t) = k$. Let κ be the labeling defined by (i) and (ii). Define, for each $v \in nd(t)$, $w_v := w_i w_{i+1} \cdots w_{i+\ell}$, where $i \geq 1$ and $\ell \geq 0$ are such that $contr_t(v) = \{\langle leaf \rangle(t)(j) \mid j = i, \ldots, i + \ell\}$.

Claim 1.1.37 *For all $v \in nd(t)$, $\kappa(v) = cat_\varphi(w_v)$.*

Proof. By induction on $depth(sub_t(v))$.

If $depth(sub_t(v)) = 0$, then $v = \langle leaf \rangle(t)(i)$ for some $i \in \{1, \ldots, k\}$, and $w_v = w_i$. Hence $\kappa(v) = cat_\varphi(w_i) = cat_\varphi(w_v)$.

Suppose now that the claim holds for all $v \in nd(t)$ with $depth(sub_t(v)) \leq m, m \geq 0$. Let $v \in nd(t)$ with $depth(sub_t(v)) = m + 1$. Then, by condition (ii),

$\kappa(v) = cat_\varphi(\kappa(v_1)\cdots\kappa(v_n))$, where $(v_1,\ldots,v_n) = \langle ddes\rangle_t(v)$. By the induction hypothesis $\kappa(v_i) = cat_\varphi(w_{v_i})$ for all $i = 1,\ldots,n$.

Clearly, $w_v = w_{v_1}\cdots w_{v_n}$. Then, by Theorem 1.1.34,

$$cat_\varphi(w_v) = cat_\varphi(cat_\varphi(w_{v_1})\cdots cat_\varphi(w_{v_n})) = cat_\varphi(\kappa(v_1)\cdots\kappa(v_n)) = \kappa(v).$$

This completes the induction proof of Claim 1.1.37. □

By Claim 1.1.37, $\kappa(root(t)) = cat_\varphi(w_{root(t)}) = cat_\varphi(w_1\cdots w_k) = cat_\varphi(w)$. Hence the unique labeling κ determined by conditions (i) and (ii) also satisfies condition (iii). □

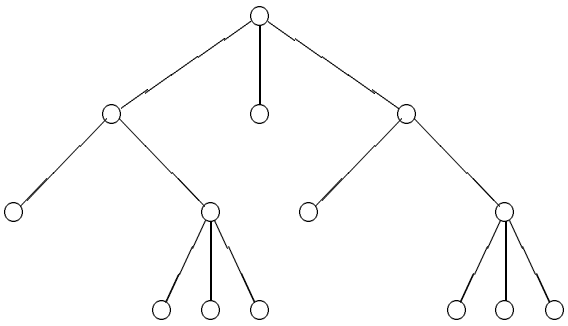


Figure 1.3: t

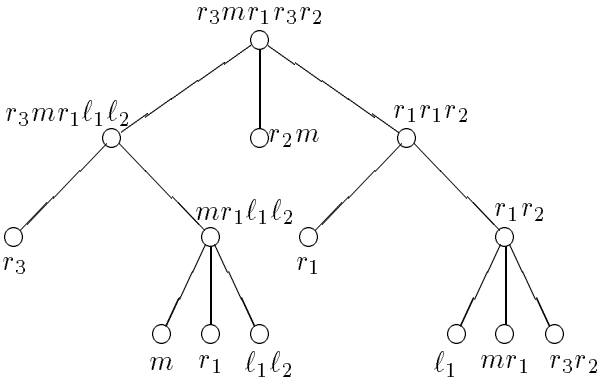


Figure 1.4: (t,κ)

Example 1.1.38 Let φ be the strict code from Example 1.1.33, and let $w = \ell_1 r_1 r_1 m \ell_2 m r_3 \ell_1 \ell_2 r_2 m r_1 \ell_1 m r_1 r_3 r_2$. Then $w = w_1 w_2 \cdots w_9$, where $w_1 = \ell_1 r_1 r_1$, $w_2 = m$, $w_3 = \ell_2 m r_3$, $w_4 = \ell_1 \ell_2$, $w_5 = r_2 m$, $w_6 = r_1$, $w_7 = \ell_1$, $w_8 = m r_1$, $w_9 = r_3 r_2$. Consider the tree t with $\#leaf(t) = 9$ shown in Figure 1.3. Then (t, κ) is as shown in Figure 1.4, where κ is the node-labeling of t as in Corollary 1.1.36.

Also, $w = w'_1 w'_2 \cdots w'_7$, where $w'_1 = \ell_1$, $w'_2 = r_1 r_1 m$, $w'_3 = \ell_2 m$, $w'_4 = r_3$, $w'_5 = \ell_1 \ell_2 r_2 m r_1$, $w'_6 = \ell_1 m r_1$, and $w'_7 = r_3 r_2$. Consider tree t' with $\#leaf(t') = 7$ shown in Figure 1.5. Now (t', κ') is as shown in Figure 1.6, where κ' is the node-labeling of t' as in Corollary 1.1.36. □

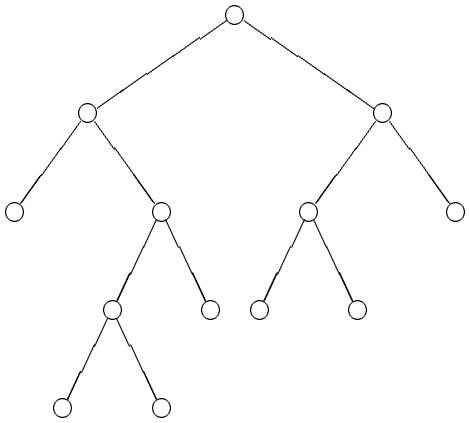


Figure 1.5: t'

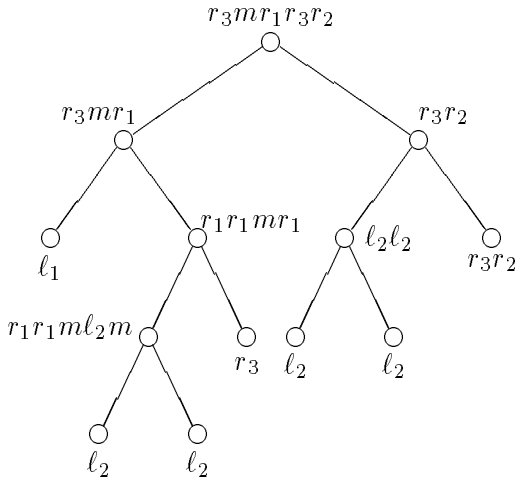


Figure 1.6: (t', κ')

1.2 Binary codes

In this section we consider *binary* OS systems, which can be identified with *binary codes*, i.e., codes on the set of binary trees.

Many notions and notations for arbitrary trees are carried over to binary trees. Since it is normally clear how these notions and notations are formally defined in the binary case, we often do not give those definitions here.

Definition 1.2.1 Let $\mathrm{T_b}$ be a selector set of binary trees and let Σ be an alphabet.
(1) A mapping $\varphi : \mathrm{T_b} \to \Sigma^*$ is *local* if there is a $\psi : \Sigma \to \Sigma^2$, such that for all $t_1, t_2 \in \mathrm{T_b}$, where $t_2 \in subs_{t_1}(i,2)$ for some $i \in \mathbf{N}_+$, if $\varphi(t_1) = xay$ with $|x| = i-1$ and $a \in \Sigma$, then $\varphi(t_2) = x\psi(a)y$.
(2) A mapping $\varphi : \mathrm{T_b} \to \Sigma^*$ is a *binary code* if it is injective, length preserving, and local.
(3) A binary code φ is a *strict binary code* if it is completeness consistent and rich.

Note that sibling consistency is not needed here, because the sibling segments of a binary tree are the same as its complete segments.

Also in the binary case, we assume that a binary code uses all letters of its alphabet. Then again, each binary code has a unique local function. From now on $\mathrm{T_b}$ is a fixed selector set of binary trees. Note that nothing is assumed about one_φ.

For strict binary codes a result analogous to Proposition 1.1.4(1) holds.

Lemma 1.2.2 *Let φ be a strict binary code. Then $\{L_\varphi, R_\varphi\}$ is a partition of $alph(\varphi)$.*

Proof. Since φ is rich, $alph(\varphi) = L_\varphi \cup R_\varphi$. It remains to be proved that $L_\varphi \cap R_\varphi = \emptyset$. Assume to the contrary that there is an $a \in alph(\varphi)$ such that $a \in L_\varphi \cap R_\varphi$. Hence, there exist $b, c, d \in alph(\varphi)$ such that $ab, ca \in C_\varphi$, $d \to ca$, and $x_1 dx_2, x_1 cax_2 \in ran(\varphi)$ for some $x_1, x_2 \in alph(\varphi)^*$. Since φ is rich there exist $y_1, y_2 \in alph(\varphi)^+$ such that $y_1 dby_2 \in ran(\varphi)$. Then also $w = y_1 caby_2 \in ran(\varphi)$. But, clearly ab does not label a complete segment in $\varphi^{-1}(w)$, which contradicts the completeness consistency of φ.

Hence $L_\varphi \cap R_\varphi = \emptyset$, and $\{L_\varphi, R_\varphi\}$ is a partition of $alph(\varphi)$. □

As in Section 1.1, another point of view is taken by requiring that the sets of left and right letters are disjoint in the definition of a binary code, leading to the notion of a *marked binary code*.

Definition 1.2.3 A binary code φ is *marked* if φ is locally injective, and L_φ and R_φ are disjoint.

We now investigate the number of letters of the alphabet of binary marked codes and binary strict codes. Recall that in the case of marked (resp. strict) codes for arbitrary trees this number is at least 6 (resp. exactly 6). Now we will show that in the binary case this number is at least 4 (resp. exactly 4), by using similar arguments as in Lemmas 1.1.6 and 1.1.17.

Lemma 1.2.4 *Let φ be a marked binary code, with local function ψ. Then ψ is a bijection from $alph(\varphi)$ onto C_φ.*

Lemma 1.2.5 *For each marked binary code φ, $\#alph(\varphi) \geq 4$, and, for each strict binary code φ, $\#alph(\varphi) = 4$.*

Proof. Let φ be a marked binary code. Note that $C_\varphi \subseteq L_\varphi R_\varphi$. By Lemma 1.2.4, $\#alph(\varphi) = \#C_\varphi$. Thus, $\#alph(\varphi) \leq \#L_\varphi \#R_\varphi$, and, since L_φ and R_φ are disjoint, $\#alph(\varphi) \geq \#L_\varphi + \#R_\varphi$. It is easily seen that if $x, y \in \mathbf{N}_+$ are such that $xy \geq x+y$, then $x, y \geq 2$. Consequently, the above implies that $\#L_\varphi \geq 2$ and $\#R_\varphi \geq 2$. Hence, $\#alph(\varphi) \geq \#L_\varphi + \#R_\varphi \geq 4$.

If φ is a strict code, then the richness of φ implies that $C_\varphi = L_\varphi R_\varphi$ (see [5] for the general case) and $\#alph(\varphi) = \#L_\varphi + \#R_\varphi$. Hence, in that case, $\#alph(\varphi) = \#L_\varphi + \#R_\varphi = \#L_\varphi \#R_\varphi$, which implies that $\#L_\varphi = 2$, $\#R_\varphi = 2$, and $\#alph(\varphi) = 4$. □

Marked and strict binary codes can also be translated to "binary" OS systems.

Definition 1.2.6 An OS system $G = (\Sigma, P, \sigma)$ is a *binary* OS *system* if for each $a \to x \in P$, $|x| = 2$.

Definition 1.2.7 Let $G = (\Sigma, P, \sigma)$ be a binary OS system.
(1) G is a *strict binary* OS *system* if G is deterministic, backwards deterministic, and there is a partition of Σ into two sets L and R such that $\#L = 2$, $\#R = 2$, and, for each production $a \to x \in P$, $x \in LR$.
(2) G is a *marked binary* OS *system* if G is deterministic, backwards deterministic, and there are two disjoint subsets L and R of Σ such that, for each production $a \to x \in P$, $x \in \mathrm{LR}$.

For a binary OS system $G = (\Sigma, P, \sigma)$, we use L_G and R_G to denote $\{b \in \Sigma \mid b = first(x)$ for some $a \to x$ in $P\}$, and $\{b \in \Sigma \mid b = last(x)$ for some $a \to x$ in $P\}$, respectively.

Remark 1.2.8
(1) Note that if G is a marked (strict) binary OS system, then $L_G = L_{\mathrm{COD}_G}$ and $R_G = R_{\mathrm{COD}_G}$.
(2) In a strict binary OS system G, the two sets L and R mentioned in Definition 1.2.7(1) are exactly the sets L_G and R_G of left resp. right letters of G. In a marked binary OS system, this is not necessarily the case. Let $G = (\Sigma, P, \sigma)$ be a marked binary OS system, and let L and R be sets as in Definition 1.2.7(2). Then $L_G \subseteq L$ and $R_G \subseteq R$, and if $L_G \subset L$, then $L = L_G \cup \{\sigma\}$, if $R_G \subset R$, then $R = R_G \cup \{\sigma\}$. Hence, we get an equivalent definition of a marked binary OS system if we replace the third condition in Definition 1.2.7(2) by $L_G \cap R_G = \emptyset$. □

In the binary case we have the same correspondence between marked (strict) binary codes and marked (strict) binary OS systems as in the general case (cf. Lemma 1.1.10, Proposition 1.1.12 and Theorem 1.1.30).

Theorem 1.2.9
(1) *For each marked (resp. strict) binary code φ, $\mathrm{OS}(\varphi)$ is a marked (resp. strict) binary OS system, and, moreover, $\mathrm{COD}_{\mathrm{OS}(\varphi)} = \varphi$.*
(2) *For each marked (resp. strict) binary OS system G, COD_G is a marked (resp. strict) binary code, and, moreover, $\mathrm{OS}(\mathrm{COD}_G) = G$.*

Proof. Analogous to the proof of Theorem 1.1.30 and the proof of Proposition 1.1.12 (see [5]). What is not trivial is to show that COD_G is rich for a strict binary OS system G (see [5]). □

Again, proving that COD_G is injective can be done through the notion of "non-overlapping binary OS system". It should be clear that a binary OS system G is non-overlapping iff it is marked, since each right-hand side is in $L_G R_G$.

From Lemma 1.2.2, Lemma 1.2.5, Definition 1.2.7, and Theorem 1.2.9, we obtain the following result, which is analogous to Theorem 1.1.19.

Corollary 1.2.10 *A binary code φ is strict iff φ is a minimal marked binary code.*

Such minimal marked (i.e., strict) binary codes do exist, as the following example shows.

Example 1.2.11 Let $G = (\Sigma, P, \sigma)$ be the OS system with $\Sigma = \{\ell_1, \ell_2, r_1, r_2\}$, $P = \{\ell_1 \to \ell_2 r_2, \ell_2 \to \ell_1 r_1, r_1 \to \ell_1 r_2, r_2 \to \ell_2 r_1\}$, and $\sigma \in \Sigma$ arbitrarily chosen. Clearly, G is a deterministic binary OS system that is backwards deterministic. Furthermore, if $L = \{\ell_1, \ell_2\}$ and $R = \{r_1, r_2\}$, then L and R satisfy the conditions in Definition 1.2.7(1). Hence G is a strict binary OS system. □

Of course, we can also use the marked OS systems from Section 1.1 to code binary trees, if we restrict them in such a way that they become binary OS systems. That is, the productions with right-hand sides of length > 2 are removed. Formally, an OS system $G = (\Sigma, P, \sigma)$ is a 2-*restricted* OS system if there is a marked unlimited OS system $G' = (\Sigma, P', \sigma)$ such that $P = \{(a \to x) \in P' \mid |x| = 2\}$. Note that a 2-restricted OS system is reduced. As we will show now, 2-restricted OS systems are an example of marked, but not strict (because not minimal) binary OS systems.

Theorem 1.2.12 *A binary OS system $G = (\Sigma, P, \sigma)$ is 2-restricted iff it is a marked binary OS system such that $\sigma \notin L_G \cup R_G$.*

Proof. Let $G = (\Sigma, P, \sigma)$ be a binary OS system.

Suppose that G is 2-restricted. Since the marked unlimited OS system from which G originates is semi-deterministic and backwards deterministic, it follows that G is deterministic and backwards deterministic. Obviously, if L, M, R are the original subsets of Σ satisfying the conditions of Definition 1.1.15, then L and R satisfy the conditions of Definition 1.2.7(2). Hence G is a marked binary OS system, and, since $\sigma \in M$, $\sigma \notin L_G \cup R_G$.

Suppose now that G is a marked binary OS system such that $\sigma \notin L_G \cup R_G$. Let $G' = (\Sigma, P', \sigma)$ be the unlimited OS system such that $P' = \{a \to \ell\sigma^k r \mid a \to \ell r \in P, k \geq 0\}$. Clearly, G' is semi-deterministic and backwards deterministic because G is deterministic and backwards deterministic. Furthermore, L_G, $\{\sigma\}$, and R_G satisfy the conditions in Definition 1.1.15. Hence G' is a marked OS system. Since $P = \{a \to x \in P' \mid |x| = 2\}$, it follows that G is a 2-restricted OS system. □

Since in the binary case the notions of marked and non-overlapping OS systems coincide, we obtain a result similar to Theorem 1.1.26 for marked binary OS systems.

Theorem 1.2.13 *A deterministic binary OS system $G = (\Sigma, P, \sigma)$ is marked iff, for each $x \in \Sigma^+$, there exist a unique origin y such that $y \Rightarrow^* x$ and a unique derivation forest of x from y.*

Proof. Analogous to the proof of Theorem 1.1.26. □

From Theorem 1.2.13 we obtain that a deterministic binary OS system with the "unique origin property" has an alphabet of at least four letters (by Lemma 1.2.5), and that such a binary OS system is minimal (i.e., has an alphabet of exactly four letters) iff it is a strict binary OS system (by Corollary 1.2.10).

We now show that we obtain the same results if we relax the determinism requirement.

Definition 1.2.14 Let $G = (\Sigma, P, \sigma)$ be a binary OS system.
(1) G is *binary forest complete* if, for each binary forest f, there is a derivation forest in G such that f is its underlying forest.
(2) G is a *binary forest coding scheme*, abbreviated as *bfcs*, if
(i) G is binary forest complete, and
(ii) for each word $x \in \Sigma^+$ there is a unique origin y, and a unique derivation forest of x from y.

By requiring that a bfcs is binary forest complete, we have made sure that still for each binary forest f there is at least one code-word, which is the yield of a derivation forest of which f is the underlying forest. Note that this way of "coding", using a bfcs, is not necessarily unique. However, due to Property 2(ii) of Definition 1.2.14, it is injective.

If we minimize bfcs's, then we obtain deterministic bfcs's.

Theorem 1.2.15 *Every minimal bfcs is deterministic.*

Proof. It is clear from the proof of Theorem 1.1.26 that an arbitrary bfcs has the property that no right-hand sides overlap, i.e., $L_G \cap R_G = \emptyset$.

Let $G = (\Sigma, P, \sigma)$ be a minimal bfcs. We first show that
$(*)$ *for each letter b of Σ there is at least one production in P starting with b.*

Assume to the contrary that there is $ab \in \Sigma$ such that none of the productions in P starts with b. Then, in any derivation forest, b can label leaves only. But then it would be possible to construct a bfcs with the remaining letters of Σ in the following way.

Let $G' = (\Sigma - \{b\}, P', \sigma)$, where $P' = \{a \to \ell r \in P \mid b \neq \ell, r\}$. Note that $\sigma \neq b$. We claim that G' is a bfcs. First we show that G' is binary forest complete.

Let f be a binary forest. Attach to each leaf of f two descendants. Label the so obtained forest f' according to a derivation in G. This can be done since G is binary forest complete. If we cut f' in such a way that we obtain f again, then we have a labeling for f which does not use b. Hence the thus constructed labeled forest is a derivation forest in G' with underlying forest f. Hence G' is binary forest complete.

Let $x \in (\Sigma - \{b\})^+$, and let $y, y' \in (\Sigma - \{b\})^+$ be origins in G' such that $y \Rightarrow^* x$ and $y' \Rightarrow^* x$ in G', with corresponding derivation forests f and f', respectively. Since y and y' do not contain b, y and y' are also origins in G. Obviously, f and f' are derivation forests of x from y and y', respectively, in G. Hence $y = y'$ and $f = f'$, because G is a bfcs. Hence each word $x \in (\Sigma - \{b\})^+$ has a unique origin and a unique derivation forest from it in G'.

Consequently, G is a bfcs, with an alphabet of $|\Sigma| - 1$ letters. But this contradicts the minimality of G. Hence for each letter b of Σ there is at least one production starting with b.

Since each right-hand side has a unique origin, G is backwards deterministic. Hence there are at most $\#L_G\#R_G$ productions. By $(*)$ there are at least $\#L_G + \#R_G$ productions. Hence $\#L_G + \#R_G \leq \#P \leq \#L_G\#R_G$. Consequently, $\#L_G \geq 2$ and $\#R_G \geq 2$. Since there exist bfcs's with 4 letters (e.g., see Example 1.2.11), it follows that $\#\Sigma = 4$, $\#L_G = 2$, $\#R_G = 2$, and $\#P = 4$.

Hence, by $(*)$, for each letter $b \in \Sigma$ there is exactly one production, i.e., G is deterministic. □

Obviously it follows from Theorem 1.2.15 that each bfcs has at least 4 letters, and (using Theorem 1.2.13 and Corollary 1.2.10) that a bfcs is minimal iff it is a strict binary OS system.

We now consider the number of non-isomorphic strict binary codes (cf. [5]). Two binary codes φ and φ' are *isomorphic* if there exists a bijection $f : alph(\varphi) \to alph(\varphi')$ such that for each $t \in \mathrm{T_b}$, $f(\varphi(t)) = \varphi'(t)$ (where the homomorphic extension of f is also denoted simply by f). Hence two strict binary OS systems $G = (\Sigma, P, \sigma)$ and $G' = (\Sigma', P', \sigma')$ are isomorphic (as codes) if there exists a bijection $f : \Sigma \to \Sigma'$ such that $f(P) = P'$ (i.e., $a \to x \in P$ iff $f(a) \to f(x) \in P'$) and $f(\sigma) = \sigma'$.

Theorem 1.2.16 *There are exactly 24 mutually non-isomorphic strict binary OS systems.*

Proof. It is sufficient to consider only the strict binary OS systems $G = (\Sigma, P, \sigma)$ with Σ such that $L_G = \{\ell_1, \ell_2\}$ and $R_G = \{r_1, r_2\}$. The right-hand sides are then $\ell_1 r_1, \ell_1 r_2, \ell_2 r_1$, and $\ell_2 r_2$.

There are $4 \times 4! = 96$ possibilities to construct G. But some of these strict binary OS systems are isomorphic.

> **Claim 1.2.17** *For each strict binary OS system as given above there are exactly 4 isomorphic strict binary OS systems.*
> **Proof.** Two strict binary OS systems $G = (\Sigma, P, \sigma)$ and $G' = (\Sigma, P', \sigma')$ are isomorphic iff there exists a permutation $f : \Sigma \to \Sigma$ such that $f(P) = P'$ and $f(\sigma) = \sigma'$. For such a permutation f it must be that $f(\{\ell_1, \ell_2\}) = \{\ell_1, \ell_2\}$ and $f(\{r_1, r_2\}) = \{r_1, r_2\}$. Hence there are 4 such permutations, say f_1, f_2, f_3, f_4. Let $G = (\Sigma, P, \sigma)$ be a strict binary OS system. We will show that these 4 permutations give 4 distinct isomorphic strict binary OS systems. Let $i, j \in \{1, \ldots, 4\}$ be such that $f_i(P) = f_j(P)$ and $f_i(\sigma) = f_j(\sigma)$. Consider $\sigma \to \ell_\sigma r_\sigma$ in P. Since $f_i(P) = f_j(P)$, $f_i(\sigma) \to f_i(\ell_\sigma) f_i(r_\sigma) \in f_j(P)$. Since $f_i(\sigma) = f_j(\sigma)$, it follows that $f_i(\ell_\sigma) = f_j(\ell_\sigma)$ and $f_i(r_\sigma) = f_j(r_\sigma)$. Consequently $f_i|_{L_G} = f_j|_{L_G}$ and $f_i|_{R_G} = f_j|_{R_G}$. Hence $f_i = f_j$.
>
> Thus, for each strict binary OS system there are exactly 4 isomorphic strict binary OS systems. □

By Claim 1.2.17 it follows that there are $96/4 = 24$ mutually non-isomorphic strict binary OS systems. This proves the theorem. □

1.3 Codes for node-labeled trees

In this section we investigate grammatical codings for node-labeled trees. For this purpose, "storage functions" are introduced and a combinatorial characterization of these functions is given. For convenience, we consider *all concrete trees* instead of the trees in a selector set. Clearly, a code φ for a selector set can be extended to a mapping on all trees (which is injective modulo isomorphism).

From now on, unless clear otherwise, φ is a fixed arbitrary code and Σ is its alphabet.

It is obvious that we can define grammatical codes for leaf-labeled trees: a leaf of a tree will simply be labeled by (x, a), where x is the original label of the leaf and a is the label that is assigned by coding the underlying tree.

Example 1.3.1 Consider the leaf-labeled tree t as shown in Figure 1.7.

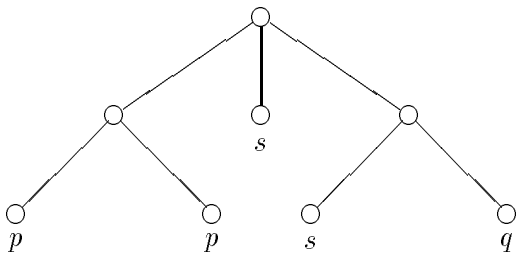


Figure 1.7: leaf-labeled tree t

Let φ be the (strict) code such that, for all $k \geq 0$,

$$\begin{array}{lll} m \to \ell_1 m^k r_2, & \ell_1 \to \ell_1 m^k r_3, & \ell_2 \to \ell_2 m^k r_3, \\ r_1 \to \ell_2 m^k r_1, & r_2 \to \ell_2 m^k r_2, & r_3 \to \ell_1 m^k r_1 \end{array} .$$

Then t is coded by the word $(p, \ell_1)(p, r_3)(s, m)(s, \ell_2)(q, r_2)$. □

As for node-labeled trees, a natural way to solve the problem of coding them is to store the labels of internal nodes into leaves. Fortunately, there are more leaves than internal nodes in each tree (recall that trees are assumed to be chain-free). To formalize the storing of internal nodes, we need the notion of a *storage function*, which for each internal node tells us in which leaf its label is stored.

Definition 1.3.2 A *storage function* is a function γ such that
(1) for each tree t and each $v \in in(t)$, $\gamma(t, v) \in leaf(t)$, and
(2) for each tree t and all $v_1, v_2 \in in(t)$, $v_1 \neq v_2$ implies $\gamma(t, v_1) \neq \gamma(t, v_2)$.

Example 1.3.3 Let t be the tree shown in Figure 1.8.

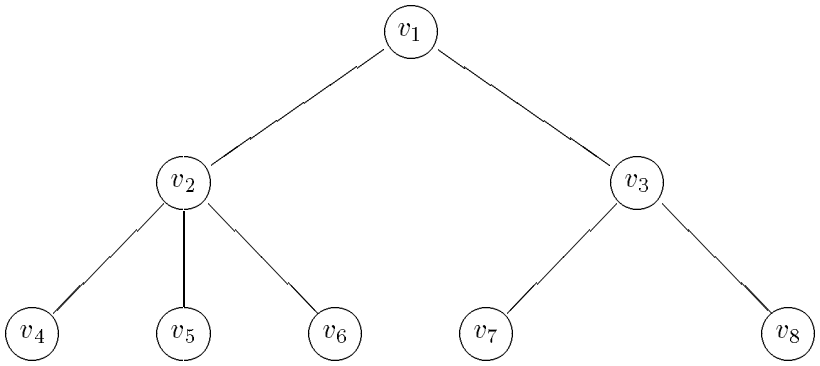


Figure 1.8: t

The values for a storage function γ on t are:

$$\gamma(t, v_1) = v_4, \quad \gamma(t, v_2) = v_7, \quad \gamma(t, v_3) = v_8.$$

Hence indeed, all three values are different leaves. □

Clearly, for a given code φ, not every storage function makes sense. There must be something "consistent" about storage functions, e.g., if we extend a tree t to a tree t', then the storage function of t' must "extend" the storage function of t. The notion of a consistent storage function is axiomatized as follows.

Definition 1.3.4 A storage function γ is *consistent with* φ if it satisfies the following axioms:
(A1) for each tree t and each $v \in in(t)$, $\gamma(t, v) \in contr_t(v)$;
(A2) for all trees t, t', and all $v \in nd(t)$, $v' \in nd(t')$, if there exists an isomorphism δ from $sub_t(v)$ onto $sub_{t'}(v')$ and if $lb_{t[\varphi]}(v) = lb_{t'[\varphi]}(v')$, then for all $w \in in(sub_t(v))$, $\delta(\gamma(t, w)) = \gamma(t', \delta(w))$;
(A3) for each tree t, each $v \in in(t)$, and each cut ρ of t which is below v, $\gamma(tree(t, \rho), v) = \rho \cap \Pi(v, \gamma(t, v))$; and
(A4) for each tree t, each $v \in in(t)$, and each $v' \in nd(t)$ such that $\gamma(t, v) \notin contr_t(v')$, $\gamma(t \bullet sub_t(v'), v) = \gamma(t, v)$.

The above axioms are surely natural. They are illustrated in Figure 1.9.

(A1) says that for each internal node v the label is stored into a leaf that is reachable from v. (A2) says that in isomorphic subtrees that are coded by φ in the same way, the labels of corresponding internal nodes are stored into corresponding leaves. By (A3), if a tree t is cut, then for each remaining internal node v, the node that becomes the new leaf where the label of v is stored lies on the path from v to the leaf of t where the label of v was stored originally. By (A4), if a subtree is removed from a tree, then the label of a remaining node is stored in the same leaf as it was before assuming that this leaf was not removed ($\gamma(t, v) \notin contr_t(v)$).

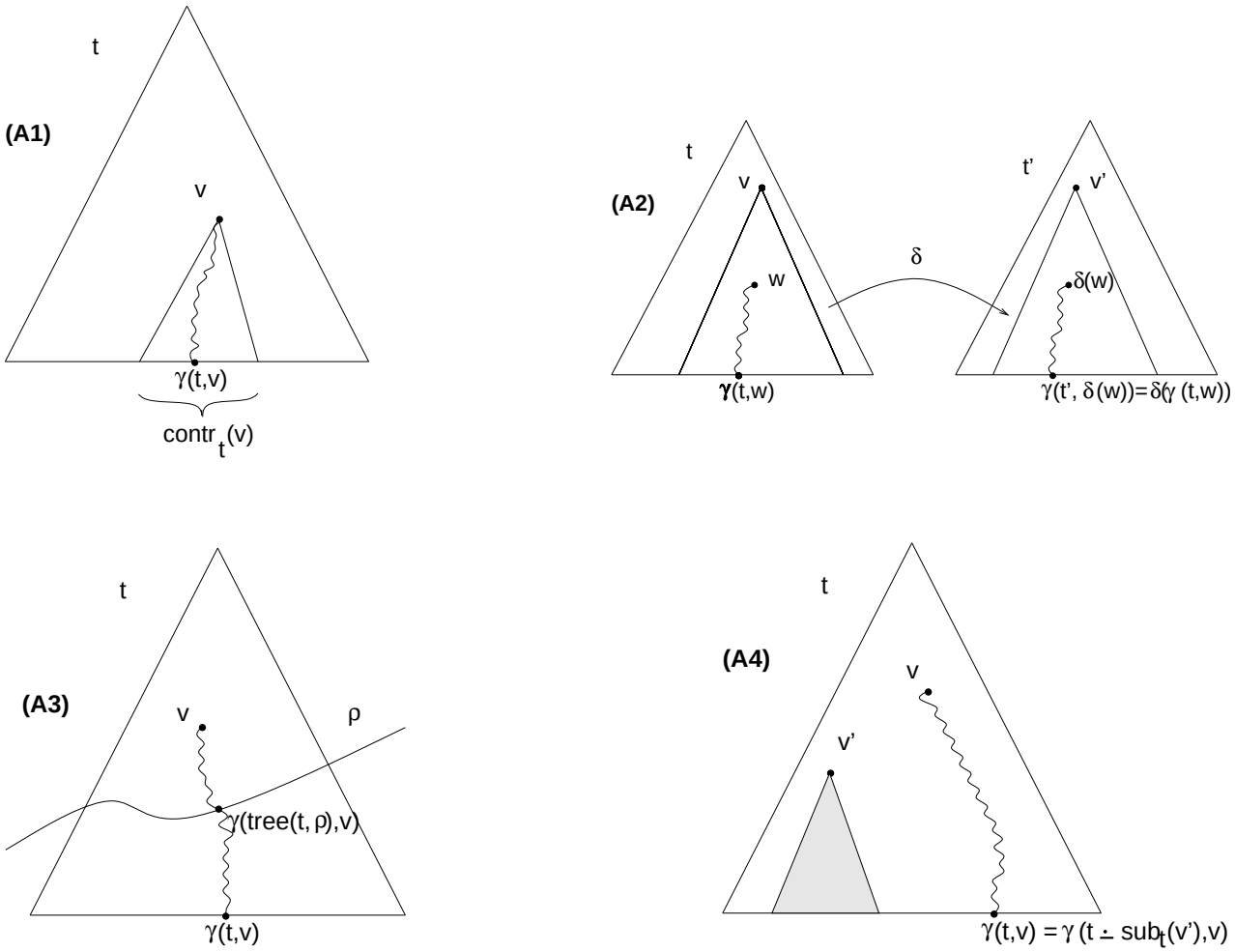


Figure 1.9: axioms (A1) to (A4)

Example 1.3.5

(1) The storage function γ from Example 1.3.3 is not consistent with any code φ because $\gamma(t, v_2) \notin \mathit{contr}_t(v_2)$.

(2) Consider the trees t and t' in Figures 1.10 and 1.11, respectively.

Let φ be such that, for all $k \geq 0$,

$$\begin{array}{llll} m \to_\varphi \ell_1 m^k r_1, & \ell_1 \to_\varphi \ell_1 m^k r_2, & \ell_2 \to_\varphi \ell_2 m^k r_2, \\ \ell_3 \to_\varphi \ell_3 m^k r_1, & r_1 \to_\varphi \ell_2 m^k r_1, & r_2 \to_\varphi \ell_3 m^k r_2 \end{array} .$$

Then $t[\varphi]$ and $t'[\varphi]$ are as shown in Figures 1.12 and 1.13, respectively.

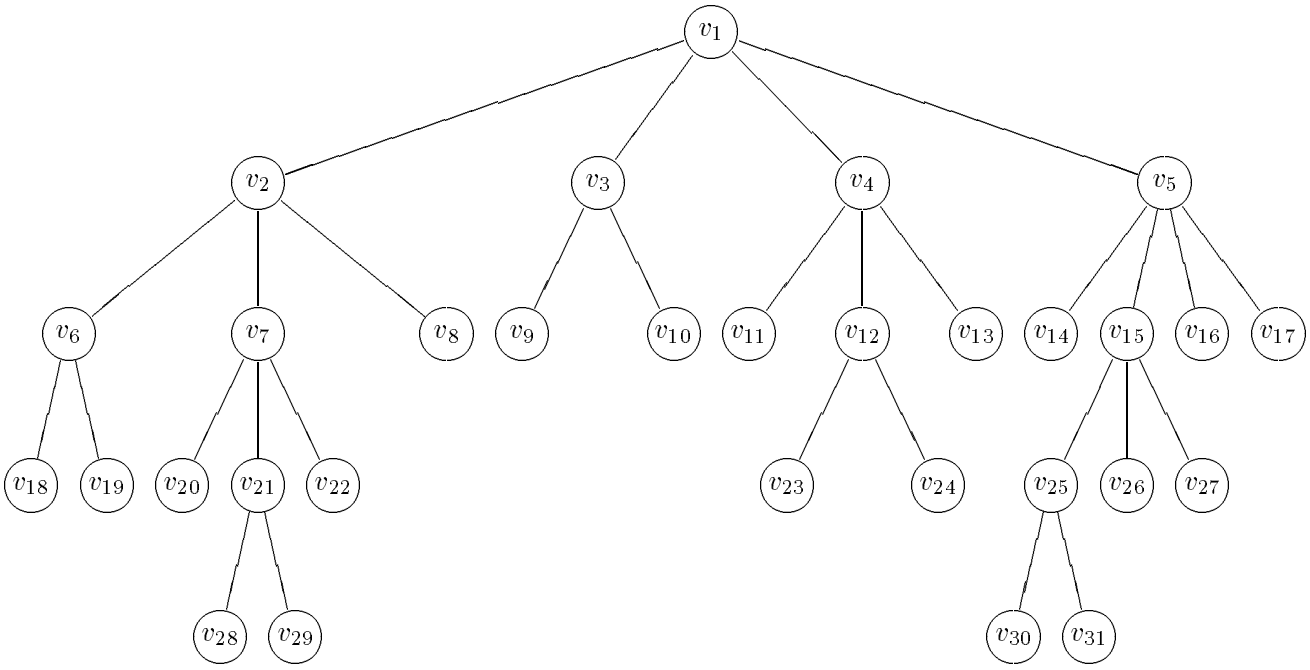


Figure 1.10: t

(2.1) Let γ_1 be a storage function such that

$$\gamma_1(t, v_4) = v_{23}, \quad \gamma_1(t, v_7) = v_{29}, \quad \gamma_1(t, v_{12}) = v_{24}, \quad \gamma_1(t, v_{21}) = v_{28}.$$

Then γ_1 is *not* consistent with φ, because $lb_{t[\varphi]}(v_7) = lb_{t[\varphi]}(v_4)$ and there is an isomorphism δ of $sub_t(v_7)$ onto $sub_t(v_4)$ such that $\delta(\gamma_1(t, v_7)) = \delta(v_{29}) = v_{24} \neq \gamma_1(t, v_4)$; hence (A2) is not satisfied.

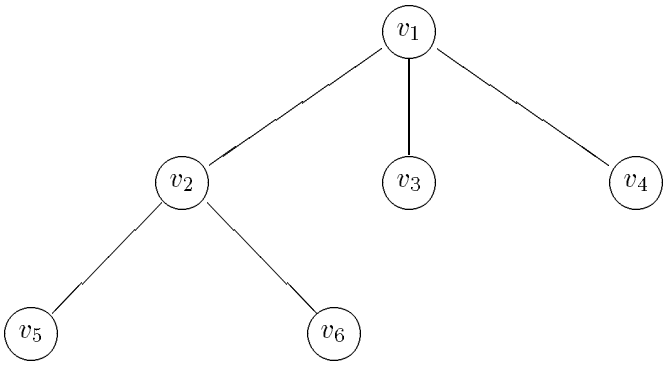


Figure 1.11: t'

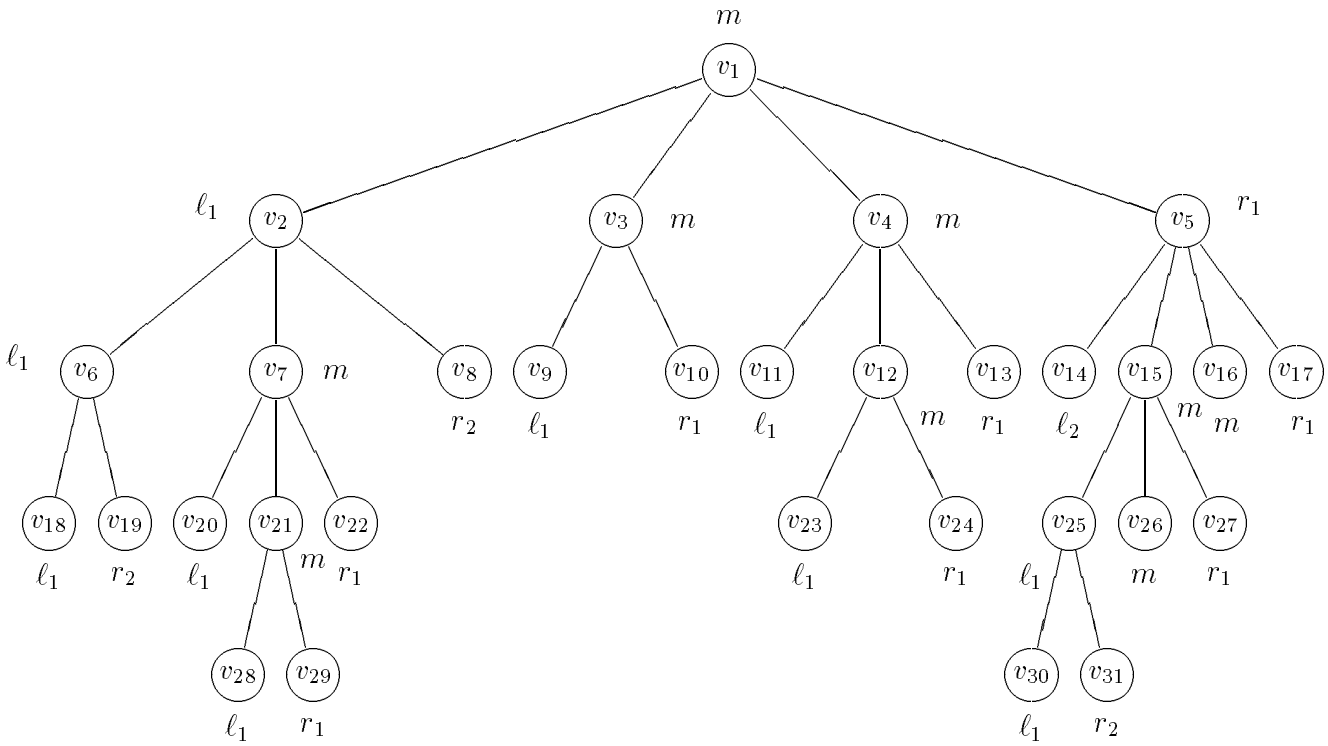


Figure 1.12: $t[\varphi]$

(2.2) Let γ_2 be a storage function such that

$$\gamma_2(t', v_1) = v_5, \quad \gamma_2(t', v_2) = v_6, \quad \gamma_2(t, v_{15}) = v_{31}, \quad \gamma_2(t, v_{25}) = v_{30}.$$

Then γ_2 is *not* consistent with φ, because $lb_{t'[\varphi]}(v_1) = lb_{t[\varphi]}(v_{15})$ and there is an isomorphism δ of t' onto $sub_t(v_{15})$ such that $\delta(\gamma_2(t', v_1)) = \delta(v_5) = v_{30} \neq \gamma_2(t, v_{15})$; hence (A2) is not satisfied.

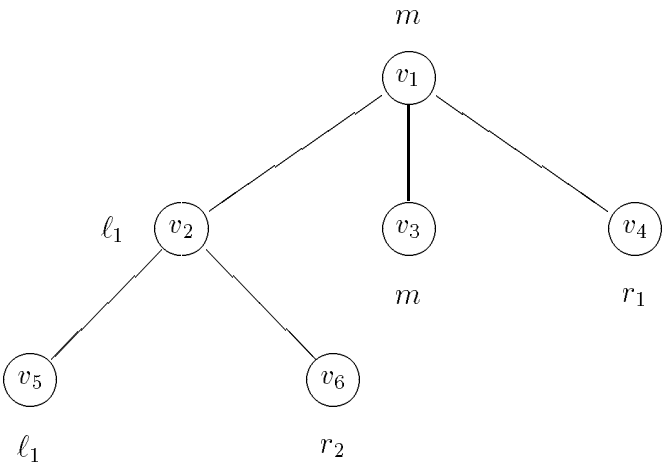


Figure 1.13: $t'[\varphi]$

(2.3) Let γ_3 be a storage function such that

$$\begin{array}{lll} \gamma_3(t,v_1)=v_8, & \gamma_3(t,v_2)=v_{19}, & \gamma_3(t,v_3)=v_9, \\ \gamma_3(t,v_4)=v_{11}, & \gamma_3(t,v_5)=v_{17}, & \gamma_3(t,v_6)=v_{18}, \\ \gamma_3(t,v_7)=v_{20}, & \gamma_3(t,v_{12})=v_{23}, & \gamma_3(t,v_{15})=v_{31}, \\ \gamma_3(t,v_{21})=v_{28}, & \gamma_3(t,v_{25})=v_{30}, & \\ \gamma_3(t',v_1)=v_6, & \gamma_3(t',v_2)=v_5. & \end{array}$$

Note that, indeed,
(i) $\gamma_3(t,v) \in contr_t(v)$ for each $v \in in(t)$, and $\gamma_3(t',v) \in contr_{t'}(v)$ for each $v \in in(t')$; hence (A1) is satisfied.
(ii) $lb_{t[\varphi]}(v_4) = lb_{t[\varphi]}(v_7)$, and for the isomorphism δ_1 of $sub_t(v_4)$ onto $sub_t(v_7)$ we have that $\delta_1(\gamma_3(t,v_4)) = \delta_1(v_{11}) = v_{20} = \gamma_3(t,v_7)$, and $\delta_1(\gamma_3(t,v_{12})) = \delta_1(v_{23}) = v_{28} = \gamma_3(t,v_{21})$;

$lb_{t[\varphi]}(v_6) = lb_{t[\varphi]}(v_{25})$, and for the isomorphism δ_2 of $sub_t(v_6)$ onto $sub_t(v_{25})$ we have that $\delta_2(\gamma_3(t,v_6)) = \delta_2(v_{18}) = v_{30} = \gamma_3(t,v_{25})$;

$lb_{t[\varphi]}(v_3) = lb_{t[\varphi]}(v_{12})$, and for the isomorphism δ_3 of $sub_t(v_3)$ onto $sub_t(v_{12})$ we have that $\delta_3(\gamma_3(t,v_3)) = \delta_3(v_9) = v_{23} = \gamma_3(t,v_{12})$;

$lb_{t'[\varphi]}(v_1) = lb_{t[\varphi]}(v_{15})$, and for the isomorphism δ of t' onto $sub_t(v_{15})$ we have that $\delta(\gamma_3(t',v_1)) = \delta(v_6) = v_{31} = \gamma_3(t,v_{15})$, and $\delta(\gamma_3(t',v_2)) = \delta(v_5) = v_{30} = \gamma_3(t,v_{25})$.

Hence (A2) is satisfied. □

In the rest of the section, we will characterize consistent storage functions, which were defined axiomatically above. This characterization will be given in terms of walks along paths in trees: an internal node v is stored in the leaf which lies at the end of such a path beginning in v.

Definition 1.3.6
(1) A *direction function* is a function $\psi : \Sigma \to \{\text{left}, \text{right}\}$.
(2) Let ψ be a direction function, t a tree, $v \in in(t)$, and $w \in ddes_t(v)$.
(2.1) The ordered pair (v,w) *agrees with* ψ if
(i) $w = left_t(v)$ if $\psi(lb_{t[\varphi]}(v)) = \text{left}$, and
(ii) $w = right_t(v)$ if $\psi(lb_{t[\varphi]}(v)) = \text{right}$.
(2.2) The ordered pair (v,w) *disagrees with* ψ if
(i) $w = right_t(v)$ if $\psi(lb_{t[\varphi]}(v)) = \text{left}$, and
(ii) $w = left_t(v)$ if $\psi(lb_{t[\varphi]}(v)) = \text{right}$.
(3) Let ψ be a direction function.
A storage function γ *follows the* ψ*-strategy* if for each tree t and each $v \in in(t)$, there is a path from v to $\gamma(t,v)$, and this path $\pi(v,\gamma(t,v)) = (v = v_0, v_1, \ldots, v_n = \gamma(t,v))$ is such that
(i) (v_0, v_1) agrees with ψ and
(ii) (v_i, v_{i+1}) disagrees with ψ for $i = 1, \ldots, n-1$.
(4) A storage function γ is an *agree/disagree function*, abbreviated as *A/D function,* if there exists a direction function ψ such that γ follows the ψ-strategy.

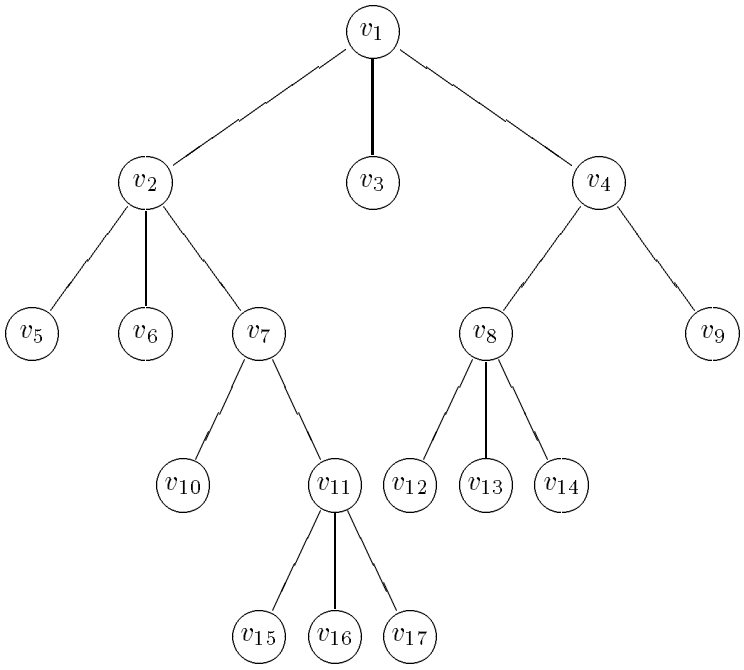


Figure 1.14: t

Example 1.3.7 Let φ be the strict code such that, for all $k \geq 0$,

$$m \to_\varphi \ell_1 m^k r_1, \quad \ell_1 \to_\varphi \ell_1 m^k r_2, \quad \ell_2 \to_\varphi \ell_3 m^k r_1,$$
$$\ell_3 \to_\varphi \ell_3 m^k r_2, \quad r_1 \to_\varphi \ell_2 m^k r_1, \quad r_2 \to_\varphi \ell_2 m^k r_2.$$

Let ψ be the direction function defined by

$$\psi(m) = \text{left}, \quad \psi(\ell_1) = \text{right}, \quad \psi(\ell_2) = \text{left},$$
$$\psi(\ell_3) = \text{left}, \quad \psi(r_1) = \text{left}, \quad \psi(r_2) = \text{left}.$$

Consider the tree t shown in Figure 1.14. Then $t[\varphi]$ is as shown in Figure 1.15. The storage function γ that follows the ψ-strategy is defined as follows:

$$\gamma(t, v_1) = v_5, \quad \gamma(t, v_2) = v_{17}, \quad \gamma(t, v_4) = v_{14},$$
$$\gamma(t, v_7) = v_{10}, \quad \gamma(t, v_8) = v_{12}, \quad \gamma(t, v_{11}) = v_{15}.$$

Figure 1.16 illustrates the directions and the strategy. □

Now we can give a combinatorial characterization of consistent storage functions.

Theorem 1.3.8 *A storage function γ is an A/D function iff γ is consistent with φ.*

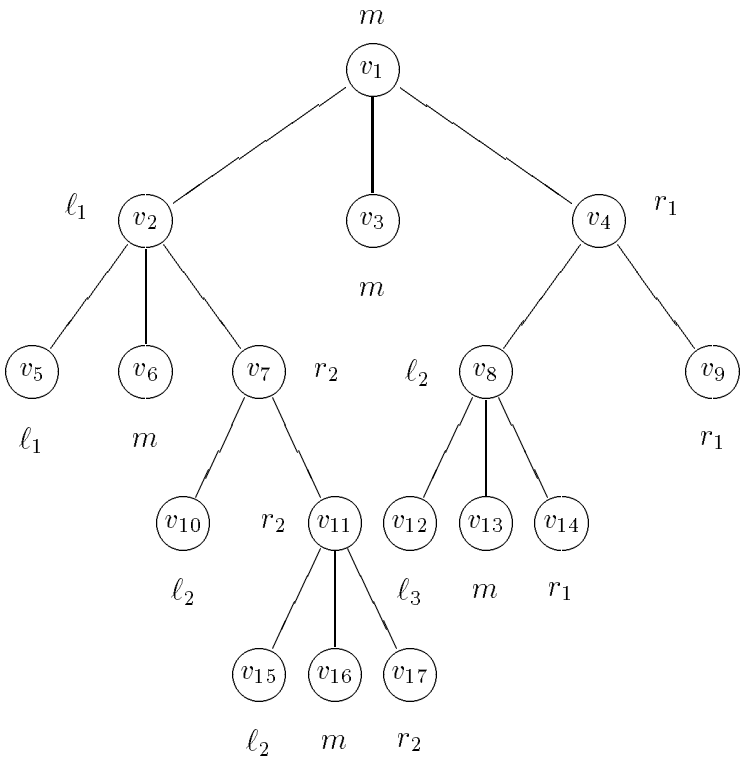


Figure 1.15: $t[\varphi]$

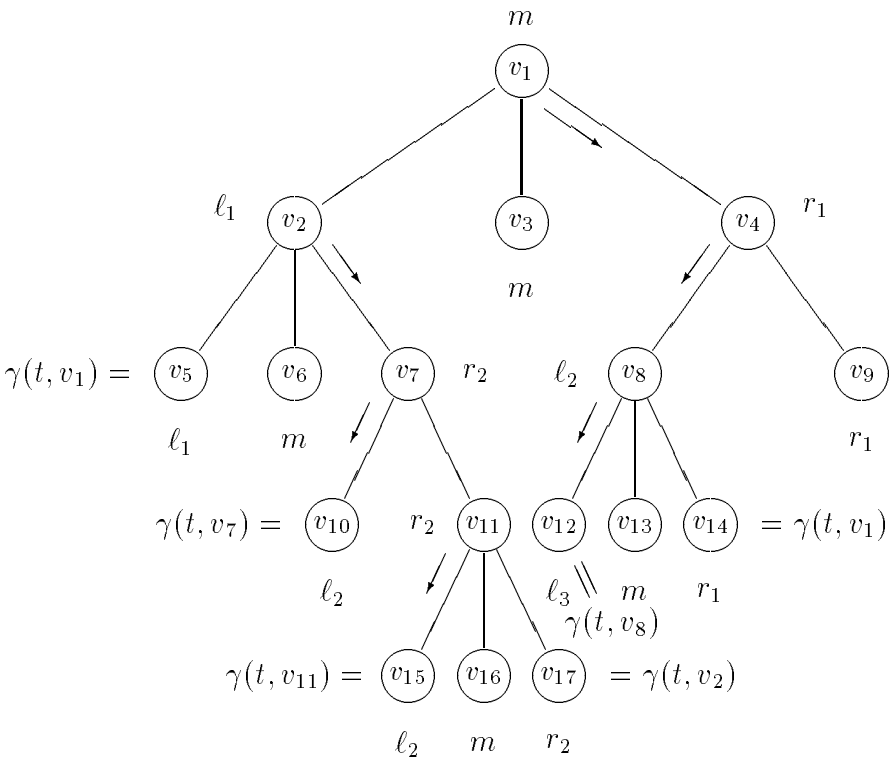


Figure 1.16: γ following the ψ-strategy

Proof. (Only if) Assume that γ is an A/D function, and let ψ be the direction function such that γ follows the ψ-strategy.
(A1) γ satisfies (A1), since for each tree t, and for each $v \in in(t)$ there is a path from v to $\gamma(t,v)$.
(A2) Let t, t' be trees, and let $v \in nd(t)$, $v' \in nd(t')$ be such that there exists an isomorphism δ from $sub_t(v)$ onto $sub_{t'}(v')$ and $lb_{t[\varphi]}(v) = lb_{t'[\varphi]}(v')$.

Note that, for each $w \in in(sub_t(v))$, $lb_{t[\varphi]}(w) = lb_{t'[\varphi]}(\delta(w))$.

Let $w \in in(sub_t(v))$, and let $\pi(w, \gamma(t,w)) = (w = w_0, w_1, \ldots, w_n = \gamma(t,w))$. Since γ follows the ψ-strategy, it follows that $(\delta(w_0), \delta(w_1))$ agrees with ψ, and $(\delta(w_i), \delta(w_{i+1}))$ disagrees with ψ for $i = 1, \ldots, n-1$. Hence, in $sub_{t'}(v')$ the path from $\delta(w)$ to $\delta(\gamma(t,w))$ equals the path from $\delta(w)$ to $\gamma(t', \delta(w)))$. Consequently, for each $w \in in(sub_t(v))$, $\delta(\gamma(t,w)) = \gamma(t', \delta(w))$, and so γ satisfies (A2).
(A3) Let t be a tree, let $v \in in(t)$, and let ρ be a cut of t below v. If $\pi(v, \gamma(t,v)) = (v = v_0, v_1, \ldots, v_n = \gamma(t,v))$, then $\pi(v, \gamma(tree(t,\rho), v)) = (v = v_0, v_1, \ldots, v_s)$, with $1 \leq s \leq n$. Since $v_s \in leaf(tree(t,\rho))$, $v_s \in \rho$, and therefore $\gamma(tree(t,\rho), v) = v_s = \rho \cap \Pi(v, \gamma(t,v))$.

Hence, γ satisfies (A3).
(A4) Let t be a tree, and let $v \in in(t)$, $v' \in nd(t)$ be such that $\gamma(t,v) \notin contr_t(v')$. Then $\Pi(v, \gamma(t,v)) \cap nd(sub_t(v')) = \emptyset$, and therefore γ follows from v the same path in $t \dot{-} sub_t(v')$ as in t. Hence $\gamma(t,v) = \gamma(t \dot{-} sub_t(v'), v)$, and so γ satisfies (A4).

Consequently, γ satisfies (A1), ..., (A4), and so γ is consistent with φ.

(If) Assume that γ is a storage function consistent with φ.

Claim 1.3.9 *For each tree t, each $v \in in(t)$, and each $v' \in nd(t)$, if $v' \in \Pi(v, \gamma(t,v))$ and $v' \neq v$, then v' is not a middle child.*

Proof. Assume to the contrary that there is a tree t, $v \in in(t)$ and $v' \in \Pi(v, \gamma(t,v))$ such that $v' \neq v$ and v' is a middle child.

Let $u \in in(t)$ be the direct ancestor of v', and consider the cut $\rho = (leaf(t) - contr_t(u)) \cup ddes_t(u)$ of t. Since γ satisfies (A3), $\gamma(tree(t,\rho), v) = \rho \cap \Pi(v, \gamma(t,v)) = v'$. Let $t_1 = tree(t,\rho) \dot{-} \bigcup_{w \in V_1} sub_t(w)$, with $V_1 = ddes_t(u) - \{v', left_t(u)\}$, and let $t_2 = tree(t,\rho) \dot{-} \bigcup_{w \in V_2} sub_t(w)$, with $V_2 = ddes_t(u) - \{v', right_t(u)\}$. Since v' is a middle child, u has exactly two direct descendants in t_1, and u has exactly two direct descendants in t_2.

Let δ be the mapping from $nd(t_1)$ to $nd(t_2)$ defined by $\delta(u) = u$, $\delta(v') = right_{t_2}(u)$, $\delta(left_{t_1}(u)) = v'$, and $\delta(x) = x$ for all $x \in nd(t_1 \dot{-} sub_{t_1}(u)) = nd(t_2 \dot{-} sub_{t_2}(u))$.

Clearly, δ is an isomorphism of t_1 onto t_2. By (A4), $\gamma(t_1, v) = \gamma(tree(t,\rho), v) = v'$, and $\gamma(t_2, v) = \gamma(tree(t,\rho), v) = v'$. By (A2), $\delta(v') = \delta(\gamma(t_1, v)) = \gamma(t_2, \delta(v)) = \gamma(t_2, v) = v'$; which contradicts the definition of δ.

Hence for each tree t, each $v \in in$(t), and each $v' \in nd(t)$, if $v' \in \Pi(v, \gamma(t,v))$, then v' is not a middle child. □

Claim 1.3.10 *For all trees t_1, t_2, each $v_1 \in in(t_1)$, each $w_1 \in nd(t_1)$, each $v_2 \in in(t_2)$, and each $w_2 \in nd(t_2)$, if $w_i \in ddes_{t_i}(v_i)$ and $w_i \in \Pi(v_i, \gamma(t_i, v_i))$ for $i = 1, 2$,*

and $lb_{t_1[\varphi]}(v_1) = lb_{t_2[\varphi]}(v_2)$, *then* $w_1 = left_{t_1}(v_1)$ *iff* $w_2 = left_{t_2}(v_2)$, *and* $w_1 = right_{t_1}(v_1)$ *iff* $w_2 = right_{t_2}(v_2)$.

Proof. Let $\rho_i = (leaf(t_i) - contr_{t_i}(v_i)) \cup ddes_{t_i}(v_i))$ for $i = 1, 2$. Then, by (A3), $\gamma(tree(t_i, \rho_i), v_i) = w_i$ for $i = 1, 2$.

Let $t_i' = tree(t_i, \rho_i) \dot{-} \bigcup_{u \in V_i} sub_{t_i}(u)$, with $V_i = ddes_{t_i}(v_i) - \{left_{t_i}(v_i), right_{t_i}(v_i)\}$ for $i = 1, 2$. By Claim 1.3.9, $w_i \notin V_i$, and so $sub_{t_i}(w_i)$ is not removed for $i = 1, 2$. By (A4), $\gamma(t_i', v_i) = \gamma(tree(t_i, \rho_i), v_i) = w_i$ for $i = 1, 2$.

Clearly, there exists an isomorphism from $sub_{t_1'}(v_1)$ onto $sub_{t_2'}(v_2)$. Since $lb_{t_1[\varphi]}(v_1) = lb_{t_2[\varphi]}(v_2)$, it follows, by (A2), that $\delta(w_1) = \delta(\gamma(t_1', v_1)) = \gamma(t_2', v_2) = w_2$, and therefore both w_1 and w_2 are leftmost or both w_1 and w_2 are rightmost. □

Now define $\psi : \Sigma \to \{\text{left}, \text{right}\}$ as follows.

Let $a \in \Sigma$ and let t be a tree such that there exists a $v \in in(t)$ with $lb_{t[\varphi]}(v) = a$, and $\langle ddes \rangle_t(v) = (v_1, v_2)$, for some $v_1, v_2 \in leaf(t)$.
Then

$$\psi(a) = \begin{cases} \text{left} & \text{if } \gamma(t, v) = v_1, \\ \text{right} & \text{if } \gamma(t, v) = v_2. \end{cases}$$

Note that by (A1), $\gamma(t, v) \in \{v_1, v_2\}$, and, by Claim 1.3.10, $\psi(a)$ does not depend on the choice of t. Hence ψ is a well-defined direction function.

Claim 1.3.11 *Let* t *be a tree, and let* $v \in in(t)$. *If* $\pi(v, \gamma(t, v)) = (v = v_0, v_1, \ldots, v_n = \gamma(t, v))$, *then* (v_0, v_1) *agrees with* ψ, *and* (v_i, v_{i+1}) *disagrees with* ψ *for* $i = 1, \ldots, n - 1$.

Proof. Let t_a be the tree we used in defining $\psi(a)$, where $a = lb_{t[\varphi]}(v_0)$. So there is a $w \in in(t_a)$ that has exactly two direct descendants, which are leaves, and $lb_{t_a[\varphi]}(w) = a$. By Claim 1.3.10, both v_1 and $\gamma(t_a, w)$ are leftmost, or both v_1 and $\gamma(t_a, w)$ are rightmost. From the definition of ψ it follows that (v_0, v_1) agrees with ψ.

It remains to be proved that (v_i, v_{i+1}) disagrees with ψ for $i = 1, \ldots, n - 1$.

To this aim, assume to the contrary that there is a j, $1 \leq j \leq n - 1$, such that (v_j, v_{j+1}) agrees with ψ. Let v' be the second node in $\pi(v_j, \gamma(t, v_j))$. We have already proved that then (v_j, v') agrees with ψ. Hence $v' = v_{j+1}$.

Let $\rho = (leaf(t) - contr_t(v_j)) \cup ddes_t(v_j)$. Then by (A3), $\gamma(tree(t, \rho), v) = v_{j+1}$ and $\gamma(tree(t, \rho), v_j) = v'$.

This, however, contradicts the injectivity of γ (condition (2) of Definition 1.3.2), and so (v_i, v_{i+1}) disagrees with ψ for $i = 1, \ldots, n - 1$. □

By Claim 1.3.11, γ is an A/D function. This completes our proof of Theorem 1.3.8. □

From Theorem 1.3.8 the following corollary is immediate.

Corollary 1.3.12 *For each strict code* φ, *there are exactly* $2^6 = 64$ *storage functions consistent with* φ.

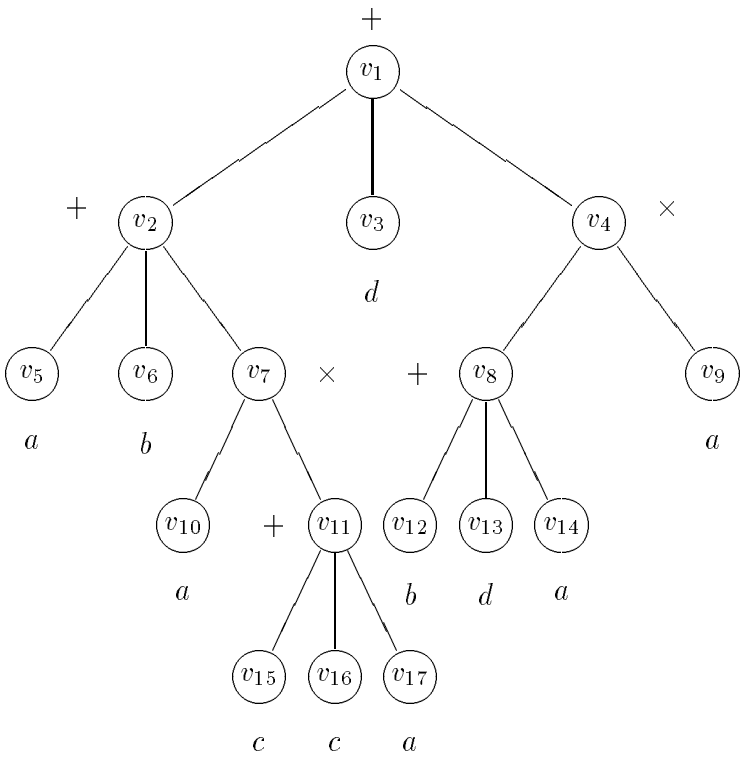


Figure 1.17: (t, η)

Example 1.3.13 Let φ, ψ, t, and γ be the strict code, the direction function, the tree, and the storage function from Example 1.3.7. Consider the node-labeling η of t shown in Figure 1.17.

Then, using γ and φ we obtain the leaf-labeling of t shown in Figure 1.18. Hence we have coded the node-labeled tree (t, η) by the code-word $(+, a, \ell_1)(\emptyset, b, m)(\times, a, \ell_2)(+, c, \ell_2)(\emptyset, c, m)$ $(+, a, r_2)(\emptyset, d, m)(+, b, \ell_3)(\emptyset, d, m)(\times, a, r_1)(\emptyset, a, r_1)$. □

Acknowledgement

The authors are indebted to J. Engelfriet and K. Salomaa for very useful comments concerning the previous versions of this paper.

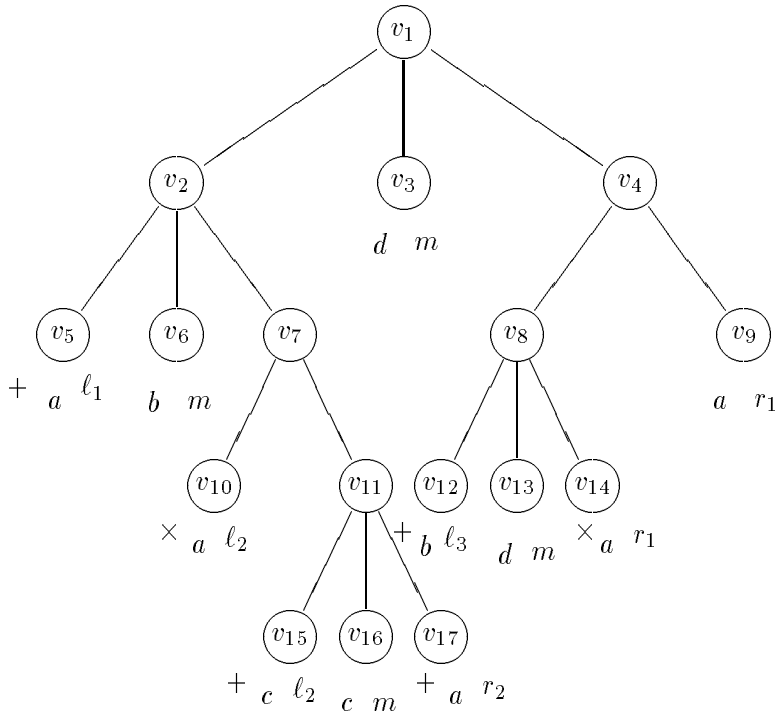


Figure 1.18: leaf-labeled version of (t, η)

Chapter 2

A Note on Binary Grammatical Codes of Trees

Introduction

Grammatical codes of trees provide a way to encode ordered trees into strings over a finite alphabet in such a way that the length of each code-word is precisely the number of leaves of the coded tree. Such codes are grammatical because they result by applying production rules of a grammar G to a tree t which becomes then a derivation tree t' in G and the yield of this derivation tree t' becomes the code-word for t. Grammatical codes were investigated in [5] and [4], see also [3].

In this note we present two topics related to grammatical codes of binary trees.

The first topic (see Section 2.2) is binary grammatical codes with a *minimal* code alphabet. It is shown that the only binary codes that are minimal in this sense are the so-called "strict" binary codes (as considered in [4]).

The second topic (see Section 2.3) concerns the extension of binary grammatical codes to grammatical codes for trees of arbitrary degree. We make comparisons between classes of codes obtained in this way and the classes from [5, 4].

In Section 2.1 we recall (from [5, 4]) some notions and results concerning grammatical codes.

2.1 Grammatical codes of trees

In this note, by a tree we mean a nonempty rooted directed ordered tree without chains (i.e., each inner node of t has at least two direct descendants). Hence a tree t is a pair $(V, \mathcal{O})$, where V is the set of nodes of t, and $\mathcal{O}$ is a function on the inner nodes of t that assigns to each inner node the sequence of its children. We use $nd(t)$ to denote the set of nodes V, $in(t)$ to denote the set of inner nodes of t, and $leaf(t)$ to denote the set of leaves of t. The *frontier of* t is the sequence of all leaves of t ordered according to $\mathcal{O}$.

A *binary tree* is a tree in which each internal node has exactly two direct descendants – hence here binary trees are *full* binary trees. A *node-labeled tree* t is a pair (t', η), where t' is

a tree and $\eta : nd(t') \to \Sigma$ is a mapping, with Σ an alphabet. We say that t' is the *underlying tree of* t, denoted by $und(t)$. The notation and terminology concerning $und(t)$ carries over to t. Also, $yield(t) = \eta(v_1) \cdots \eta(v_n) \in \Sigma^+$, where $v_1 \cdots v_n$ is the frontier of t. An *inner-labeled tree* is a pair (t, η) where t is a tree and η is a mapping defined on the *inner* nodes of t.

We do not distinguish between isomorphic trees. Trees $t = (V, \mathcal{O})$ and $t' = (V', \mathcal{O}')$ are *isomorphic* if there is a bijection $\delta : V \to V'$ such that for each $v \in in(t)$, $\mathcal{O}(v) = (v_1, \ldots, v_n)$ iff $\mathcal{O}'(\delta(v)) = (\delta(v_1), \ldots, \delta(v_n))$. The set of all (chain-free) trees modulo isomorphism is denoted by T; the set of all binary trees modulo isomorphism is denoted by $\mathrm{T_b}$; for $n \geq 1$, $\mathrm{T_b}(n)$ denotes the set of all binary trees with n leaves modulo isomorphism.

By a *code* we mean a mapping $\varphi : \mathrm{T} \to \Sigma^+$ that is injective and *length-preserving*, i.e., for each $t \in \mathrm{T}$, $|\varphi(t)| = \#leaf(t)$. Analogously, a *binary code* is a mapping $\varphi : \mathrm{T_b} \to \Sigma^+$ that is injective and length-preserving. A word $\varphi(t)$ in Σ^+ with $t \in \mathrm{T}$ (or $t \in \mathrm{T_b}$ in the binary case) is called a *code-word*; the set of all code-words of φ is denoted by $ran(\varphi)$.

A *OS system* is like a context-free grammar, except that it does not have terminal symbols, and it may have infinitely many productions. Formally, a OS system G is a triple (Σ, P, σ), where Σ is the (finite) alphabet of G, P is the (possibly infinite) set of productions of the form $a \to x$, with $a \in \Sigma$ and $x \in \Sigma^+$, $|x| \geq 2$, and $\sigma \in \Sigma$ is the axiom. The OS system G generates words in the usual way, as follows. One derivation step amounts to the substitution of a word x for an occurrence of a letter a, where $a \to x$ is a production in P. If a word $v \in \Sigma^+$ is obtained from $w \in \Sigma^+$ by a finite number of consecutive derivation steps, then we say that v is *derived from* w. The words generated by the OS system G are then the words derived from the axiom σ. We use L_G to denote $\{b \in \Sigma \mid a \to by \in P \text{ for some } a \in \Sigma \text{ and } y \in \Sigma^+\}$, M_G to denote $\{b \in \Sigma \mid a \to xby \in P \text{ for some } a \in \Sigma \text{ and } x, y \in \Sigma^+\}$, and R_G to denote $\{b \in \Sigma \mid a \to xb \in P \text{ for some } a \in \Sigma \text{ and } x \in \Sigma^+\}$.

A OS system $G = (\Sigma, P, \sigma)$ is *backwards deterministic* if $a \to x \in P$ and $a' \to x \in P$ imply that $a = a'$. It is *semi-deterministic* if for each $n \geq 2$ and for each $a \in \Sigma$ there is exactly one production $a \to x$ with $|x| = n$. If G is a semi-deterministic OS system, then each tree t is the underlying tree of exactly one derivation tree of G; this derivation tree is denoted by $t[G]$. A OS system G is *unambiguous* if for each w generated by G, there is a unique derivation tree of w.

A code $\varphi : \mathrm{T} \to \Sigma^+$ is *grammatical* if there is a semi-deterministic OS system $G = (\Sigma, P, \sigma)$ such that for each $t \in \mathrm{T}$, $\varphi(t) = yield(t[G])$. Note that if such a OS system exists, then it is unique – we say then that G *determines* φ. In what follows we shall not distinguish a grammatical code $\varphi : \mathrm{T} \to \Sigma^+$ from the OS system $G = (\Sigma, P, \sigma)$ determining φ, and we use φ for the mapping on T as well as for the OS system. Clearly (see [4]), a semi-deterministic OS system φ determines a code in the above-mentioned way iff φ is unambiguous.

In [5] and [4], grammatical codes were investigated that had a property stronger than unambiguity, the so-called "unique origin property". A OS system $\varphi = (\Sigma, P, \sigma)$ has the *unique origin property* if for each $x \in \Sigma^+$ there is a unique $y \in \Sigma^+$ such that if x is derived from some $y' \in \Sigma^+$, then y' is derived from y, and the derivation forest of x from y is unique. OS systems with the unique origin property were characterized in [4] using the notion of non-overlapping right-hand sides (Proposition 2.1.1). In general, we say that words $x, y \in \Sigma^+$ (where possibly $x = y$) are *overlapping* if there exist $v \in \Sigma^+$, $u_1, u_2, w_1, w_2 \in \Sigma^*$ such that $x = u_1 v w_1$, $y = u_2 v w_2$, $u_1 w_1 u_2 w_2 \neq \Lambda$, and $u_i w_j = \Lambda$ for some $i, j \in \{1, 2\}$.

Proposition 2.1.1 *A semi-deterministic OS system φ has the unique origin property iff it is backwards deterministic, and for all right-hand sides x, y of φ, x and y are not overlapping.*

Thus every semi-deterministic and backwards deterministic OS system with non-overlapping right-hand sides is unambiguous, and hence a grammatical code. Such grammatical codes are called "non-overlapping codes". We recall some types of grammatical codes introduced in [5] and [4] which are subclasses of the non-overlapping codes.

Definition 2.1.2
(1) A grammatical code $\varphi = (\Sigma, P, \sigma)$ is *non-overlapping* if it is backwards deterministic, and for all productions $a \to x$ and $b \to y$, x and y are not overlapping.
(2) A grammatical code $\varphi = (\Sigma, P, \sigma)$ is *marked* if it is backwards deterministic and $\{L_\varphi, M_\varphi, R_\varphi\}$ is a partition of Σ.
(3) A grammatical code $\varphi = (\Sigma, P, \sigma)$ is *strict* if it is backwards deterministic and $\{L_\varphi, M_\varphi, R_\varphi\}$ is a partition of Σ such that $\#M_\varphi = 1$, and either $\#L_\varphi = 2$ and $\#R_\varphi = 3$, or $\#L_\varphi = 3$ and $\#R_\varphi = 2$.

Clearly, each strict code is marked, and each marked code is non-overlapping. It was shown in [4] that non-overlapping and marked codes have an alphabet of at least 6 letters, and that each marked code with 6 letters is strict. In [5, 4] it was assumed that the axiom was in M_φ, but here we omit this restriction.

We have similar results in the case of binary codes. A *binary OS system* is a OS system such that for each production $a \to x$, $|x| = 2$; it is *deterministic* if for each $a \in \Sigma$ there is exactly one production $a \to x$ in P. A binary code is *grammatical* if it is determined by a deterministic binary OS system. Binary grammatical codes have *the unique origin property* if they are backwards deterministic, and no right-hand sides are overlapping. In the binary case, the fact that right-hand sides are not overlapping trivially implies that $\{L_\varphi, R_\varphi\}$ is a partition of Σ; hence, in the binary case the notions of marked code and non-overlapping code coincide.

Definition 2.1.3
(1) A binary grammatical code $\varphi = (\Sigma, P, \sigma)$ is *marked* if it is backwards deterministic and $\{L_\varphi, R_\varphi\}$ is a partition of Σ.
(2) A binary grammatical code $\varphi = (\Sigma, P, \sigma)$ is *strict* if it is backwards deterministic and $\{L_\varphi, R_\varphi\}$ is a partition of Σ such that $\#L_\varphi = 2$ and $\#R_\varphi = 2$.

It was shown in [4] that each marked binary code has an alphabet of at least 4 letters, and that a binary code is strict iff it is marked and has 4 letters. As a matter of fact there are 24 distinct non-isomorphic strict binary codes (see [4]). Strict codes can easily be decoded, see [4] and [3].

Example 2.1.4
(1) Consider the OS system $\varphi = (\{\ell_1, \ell_2, \ell_3, m, r_1, r_2\}, P, \ell_1)$, where P consists of the following productions (for each $k \geq 1$):

$$\ell_1 \to \ell_2 m^k r_1, \qquad m \to \ell_3 m^k r_1,$$
$$\ell_2 \to \ell_2 m^k r_2, \qquad r_1 \to \ell_1 m^k r_1,$$
$$\ell_3 \to \ell_3 m^k r_2, \qquad r_2 \to \ell_1 m^k r_2.$$

φ is a strict code.

(2) Consider the binary OS system $\varphi = (\{a, b, c, d\}, P, a)$ where P consists of the productions $a \to bc$, $b \to bd$, $c \to ac$, and $d \to ad$. Then φ is a strict binary code. □

2.2 Binary grammatical codes with 4 letters

To code binary trees (in a length-preserving manner) one needs at least 4 letters, since the number of binary trees with n leaves is greater than 3^n for sufficiently large n (see, e.g., [12]). As shown in [4] minimal binary codes (i.e., binary codes with an alphabet of 4 letters) exist, e.g., strict binary codes are such codes. Here we will show (Theorem 2.2.7) that strict codes are the only binary codes that are minimal and grammatical.

It is well-known (see, e.g., [12]) that the number b_n of binary trees with n leaves is "close to 4^n". More precisely,

$$b_n = \frac{4^n}{\sqrt{\pi} n^{3/2}}(1 + \mathcal{O}(1/n)). \tag{2.1}$$

We will use the following consequence of this fact.

Lemma 2.2.1 *For each α with $0 < \alpha < 4$, there exists a natural number $N_1 > 0$ such that for each $n \geq N_1$, $b_n > \alpha^n$.*

Proof. Since $\alpha < 4$, it follows that there exists a natural number $N_1 > 0$ such that for each $n \geq N_1$,

$$\alpha^n \cdot \frac{n^{3/2}}{4^n} = \frac{n^{3/2}}{\left(\frac{4}{\alpha}\right)^n} < \frac{1}{\sqrt{\pi}}.$$

Combining this with (1) we obtain that for each $n \geq N_1$,

$$b_n > \frac{4^n}{\sqrt{\pi} n^{3/2}} > \alpha^n.$$

□

Lemma 2.2.1 implies that, given a binary code with 4 letters, every word over its alphabet occurs in some code-word – this is shown in the next lemma. In fact we prove something stronger: every word occurs in some code-word at a position which is a multiple of its length plus one.

For $w, x \in \Sigma^+$ and $1 \leq k \leq |w|$, we say that x is a *k-segment of w* if $|x| = k$ and $w = uxv$ with $|u|$ a multiple of k.

Lemma 2.2.2 *Let $\varphi : \mathrm{T_b} \to \Sigma^+$ be a binary code with $\#\Sigma = 4$. For each $x \in \Sigma^+$ there exists a $t \in \mathrm{T_b}$ such that x is a $|x|$-segment of $\varphi(t)$.*

Proof. Let $x \in \Sigma^+$ with $|x| = k$. Assume to the contrary that for all $t \in \mathrm{T_b}$, x is not a k-segment of $\varphi(t)$. Then

$$ran(\varphi) \subseteq (\Sigma^k - \{x\})^* \cdot (\bigcup_{j=0}^{k-1} \Sigma^j).$$

In particular, if n is a multiple of k, then

$$\#(ran(\varphi \mid_{\mathrm{T_b}(n)})) \leq (4^k - 1)^{n/k}.$$

Since φ is injective, $b_n = \#\mathrm{T_b}(n) = \#(ran(\varphi \mid_{\mathrm{T_b}(n)}))$. We conclude that $b_n \leq (4^k - 1)^{n/k}$ for each n that is a multiple of k.

However, by Lemma 2.2.1 with $\alpha = (4^k - 1)^{1/k}$, we have that for sufficiently large n, $b_n > \alpha^n = (4^k - 1)^{n/k}$; a contradiction.

Consequently, there is a $t \in \mathrm{T_b}$ such that x is a k-segment of $\varphi(t)$. □

To prove the main theorem of this section (Theorem 2.2.7), we need a particular implication of Lemma 2.2.2. Given two distinct words x and y of the same length, we wish to construct a "context" for them, i.e., a pair of code-words which differ only in the occurrences of x and y. Lemma 2.2.4, which follows easily from Lemma 2.2.2, states that this is possible.

Let $x, y \in \Sigma^+$ be such that $|x| = |y|$ and $x \neq y$. We define a mapping $h_{(x,y)} : \Sigma^+ \to \Sigma^+$ that replaces every $|x|$-segment x of w by y. Formally, if $w = v_1 v_2 \dots v_n u \in \Sigma^+$, with $n \geq 0$, $|v_i| = |x|$ for $i = 1, \dots, n$, and $|u| < |x|$, then $h_{(x,y)}(w) = v_1' \dots v_n' u$, where for $i = 1, \dots, n$, $v_i' = v_i$ if $v_i \neq x$, and $v_i' = y$ otherwise.

Definition 2.2.3 Let $\varphi : \mathrm{T_b} \to \Sigma^+$ be a binary code with $\#\Sigma = 4$. For $n \geq 1$, binary trees $t_1, t_2 \in \mathrm{T_b}(n)$, and $x, y \in \Sigma^+$ with $x \neq y$ and $|x| = |y|$, t_1 and t_2 are (x,y)-*related* if $h_{(x,y)}(\varphi(t_1)) = h_{(x,y)}(\varphi(t_2))$.

Hence t_1 and t_2 are (x,y)-related if $\varphi(t_1)$ and $\varphi(t_2)$ differ only in some k-segments which equal x in one of these words and y in the other one.

Lemma 2.2.4 *Let $\varphi : \mathrm{T_b} \to \Sigma^+$ be a binary code with $\#\Sigma = 4$.*
For each pair $x, y \in \Sigma^+$ with $|x| = |y|$ and $x \neq y$, there exist $n \geq 1$ and $t_1, t_2 \in \mathrm{T_b}(n)$ with $t_1 \neq t_2$ such that t_1 and t_2 are (x,y)-related.

Proof. Let $x, y \in \Sigma^+$ be such that $|x| = |y|$ and $x \neq y$. Assume to the contrary that for every n, and for all $t_1, t_2 \in \mathrm{T_b}(n)$, if $t_1 \neq t_2$, then t_1 and t_2 are not (x,y)-related. This implies that the mapping $h_{(x,y)} \circ \varphi$ is injective (and hence a binary code). However, by the definition of $h_{(x,y)}$, for every $t \in \mathrm{T_b}$, x is not a $|x|$-segment of $h_{(x,y)}(\varphi(t))$, which contradicts Lemma 2.2.2. □

The next two lemmas yield the proof of the main theorem.

Lemma 2.2.5 *Each binary grammatical code $\varphi = (\Sigma, P, \sigma)$ with $\#\Sigma = 4$ is backwards deterministic.*

Proof. Let $\varphi = (\Sigma, P, \sigma)$ be a binary grammatical code with $\#\Sigma = 4$, and assume to the contrary that φ has productions $a \to pq$ and $b \to pq$, with $a \neq b$. By Lemma 2.2.4 there exist $n \geq 1$ and $t_1, t_2 \in \mathrm{T_b}(n)$ with $t_1 \neq t_2$ such that t_1 and t_2 are (a, b)-related. Now for each j, if the j'th letter of $\varphi(t_1)$ differs from the j'th letter of $\varphi(t_2)$ (which implies that these letters are a and b), then add two direct descendants to the j'th leaf of t_1 and to the j'th leaf of t_2. For the so obtained trees t_1' and t_2' we have $t_1' \neq t_2'$ and, since a and b have the same right-hand side in P, we have $\varphi(t_1') = \varphi(t_2')$, which contradicts the injectivity of φ. □

Lemma 2.2.6 *For each binary grammatical code $\varphi = (\Sigma, P, \sigma)$ with $\#\Sigma = 4$, $\{L_\varphi, R_\varphi\}$ is a partition of Σ.*

Proof. Let $\varphi = (\Sigma, P, \sigma)$ be a binary grammatical code with $\#\Sigma = 4$. It is sufficient to show that $L_\varphi \cap R_\varphi = \emptyset$. Assume to the contrary that there is a $q \in L_\varphi \cap R_\varphi$. Hence φ has productions $a \to pq$ and $b \to qr$. Consider the words ar and pb.

Suppose first that $ar \neq pb$. By Lemma 2.2.4, there exist $n \geq 1$ and $t_1, t_2 \in \mathrm{T_b}(n)$ with $t_1 \neq t_2$ such that t_1 and t_2 are (ar, pb)-related. Let $w_1 = \varphi(t_1)$, and $w_2 = \varphi(t_2)$. For every odd j, $1 \leq j < n$, do the following: if $w_1(j)w_1(j+1) = ar$ and $w_2(j)w_2(j+1) = pb$, then add two direct descendants to the j'th leaf of t_1, and add two direct descendants to the $j+1$'st leaf of t_2; if $w_1(j)w_1(j+1) = pb$ and $w_2(j)w_2(j+1) = ar$, then do the same with the roles of t_1 and t_2 interchanged. Let t_1' be the tree obtained in this way from t_1, and let t_2' be the tree obtained in this way from t_2. Since $w_1 \neq w_2$, there is a leaf of one of these trees, say the j'th leaf, that has been added in the above construction as the right child of a node with label a. But then in the other tree the j'th leaf is a left child, and thus $t_1' \neq t_2'$. Also, since t_1 and t_2 are (ar, pb)-related, it follows that $\varphi(t_1') = \varphi(t_2')$. This contradicts the injectivity of φ.

In the case that $ar = pb$, we start with a tree $t \in \mathrm{T_b}$ such that ar occurs in $\varphi(t)$ (by Lemma 2.2.2 such a tree exists), and construct two trees out of t. Let j be such that $\varphi(t) = warz$, with $|w| = j - 1$. The first tree is obtained by adding two descendants to the j'th leaf of t, and the second tree is obtained by adding two descendants to the $j+1$'st leaf of t. Clearly, these trees are different, but they get the same code-word. Hence also in this case we obtain a contradiction with the injectivity of φ.

Consequently, $L_\varphi \cap R_\varphi = \emptyset$. □

Theorem 2.2.7 *Each binary grammatical code with an alphabet of 4 letters is a strict binary code.*

Proof. By Lemmas 2.2.5 and 2.2.6 each binary grammatical code with 4 letters is marked, and hence it is strict. □

For binary grammatical codes with larger alphabets, Lemma 2.2.6 no longer holds, as will be shown in Example 2.2.8. Note that if a left leaf is followed by a right leaf in the frontier of a tree $t \in \mathrm{T_b}$, then these leaves are the two children of one node; we call such a pair of leaves a *complete pair of t*.

Example 2.2.8 Let φ be a binary OS system with productions $a \to ae$, $b \to bd$, $c \to ac$, $d \to ed$, and $e \to bc$, and some arbitrary axiom. Clearly this OS system is deterministic and

backwards deterministic, but not marked. Since the right-hand sides ae and ed overlap, φ does not have the unique origin property. However, we will show that φ *is* unambiguous. Hence φ is a grammatical code.

Let $n \geq 1$, and let $t_1, t_2 \in \mathrm{T_b}(n)$ be such that $\varphi(t_1) = \varphi(t_2)$. We will show by induction on n that $t_1[\varphi] = t_2[\varphi]$ (and hence $t_1 = t_2$). If $n = 1$, then trivially $t_1 = t_2$. Now suppose that the claim holds for all $n \leq k$, for some $k \geq 1$. Let $n = k + 1$, and let $t_1, t_2 \in \mathrm{T_b}(n)$. We look for a subword in $\varphi(t_1)$ that labels a complete pair in $t_1[\varphi]$ as well as in $t_2[\varphi]$. Then we can apply the induction hypothesis after removing from each tree the complete pair, and conclude that $t_1[\varphi] = t_2[\varphi]$.

Note that in any code-word $\varphi(t)$, the letters a and b label left leaves in $t[\varphi]$, and c and d label right leaves. Hence if $\varphi(t_1)$ contains a subword of the form ac, bc, or bd, then this subword labels a complete pair in both $t_1[\varphi]$ and $t_2[\varphi]$. We claim that if $\varphi(t_1)$ contains a subword ae, then this also labels a complete pair, because the corresponding occurrence of e cannot label a left leaf.

To see this, assume to the contrary that e labels a left leaf in $t[\varphi]$. Then the parent v of this leaf has label d. Let t_1' be the tree obtained from t_1 by removing all nodes below v. Then the subword ad occurs in $\varphi(t_1')$, and it labels a left and a right leaf, i.e., a complete pair in $t_1'[\varphi]$. This contradicts the fact that ad is not a right-hand side of φ.

A symmetric argument applies for every occurrence of ed: if e labels a right leaf, then its parent has label a, contradicting the fact that ad cannot label a complete pair. □

It follows from Theorem 2.2.7 and Proposition 2.1.1 that every binary OS system with 4 letters that is semi-deterministic and unambiguous has the unique origin property. It might be interesting to see whether in coding arbitrary chain-free trees the situation is similar, i.e., whether the fact that a grammatical code is minimal (using 6 letters) implies that it has the unique origin property, meaning that it is a non-overlapping code.

2.3 Extensions of binary codes

One way to obtain grammatical codes for arbitrary trees is to extend a given binary grammatical code. The idea is based on the following translation of arbitrary trees into binary trees. What happens is that each inner node together with its direct descendants is refined into a binary subtree, where additionally a labeling of the inner nodes denotes whether a node comes from the original tree or is constructed in the refinement.

Definition 2.3.1 Let $t = (V, \mathcal{O})$ be a tree. Let $v \in in(t)$ with $\mathcal{O}(v) = (u_1, \ldots, u_n), n \geq 2$. Define a new set of nodes $W_v = \{v^{(2)}, \ldots, v^{(n-1)}\}$, and corresponding ordering functions $\mathcal{O}_v$ on $\{v\} \cup W_v$ as follows.
If $n > 2$, then
$\mathcal{O}_v(v) = (u_1, v^{(2)})$,
$\mathcal{O}_v(v^{(i)}) = (u_i, v^{(i+1)})$ for $i = 2, \ldots, n-2$, and
$\mathcal{O}_v(v^{(n-1)}) = (u_{n-1}, u_n)$.
If $n = 2$, then
$\mathcal{O}_v(v) = \mathcal{O}(v) = (u_1, u_2)$.

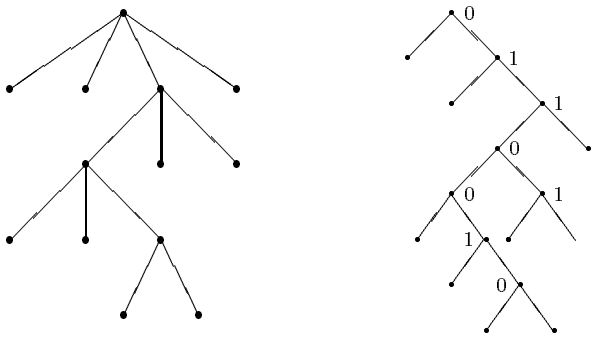


Figure 2.1: a tree t and its binary refinement $bin(t)$

Then the *binary refinement of* t, denoted by $bin(t)$, is the inner-labeled binary tree $(V \cup \bigcup_{v \in in(t)} W_v, \bigcup_{v \in in(t)} \mathcal{O}_v, \eta)$ where η is defined by $\eta(v) = 1$ if $v \notin V$ and $\eta(v) = 0$ if $v \in V$.

Example 2.3.2 Figure 2.1 gives an example of a tree t and its binary refinement $bin(t)$. □

Note that the mapping *bin* that assigns to every $t \in \mathrm{T}$ the binary refinement $bin(t)$ is injective.

For a tree t of arbitrary degree, if $bin(t) = (t', \eta)$, then the binary tree t' can be coded by means of a binary grammatical code φ. Moreover, in order to mark which nodes have label 1 in $bin(t)$, we adapt the node-labeling of $t'[\varphi]$ as follows: if v is a node in $bin(t)$ with label 1, then for each node (other than v) on the path from v to the leaf that is the leftmost descendant of v, its label a in $t'[\varphi]$ is changed into $\hat{a}$. The yield of this tree will be the code-word for t. Since only right children can have label 1 in $bin(t)$, $bin(t)$ can be recovered from this yield as follows: by removing the hats $\varphi(t')$ is obtained, and hence t'; now if the i'th letter of the yield has a hat, then the lowest ancestor of the i'th leaf of t' that is a right child gets label 1.

Since *bin* is injective, this construction indeed gives a code for T, in the sense that different trees get different code-words. Moreover it can be defined by the following OS system, which means that this code is grammatical.

Definition 2.3.3 Let φ be a binary grammatical code. The *extension of* φ is the OS system $\varphi_{ext} = (\Sigma_{ext}, P_{ext}, \sigma)$ where $\Sigma_{ext} = \Sigma \cup \{\hat{\ell} \mid \ell \in L_\varphi\}$, and

$$\begin{aligned} P_{ext} &= \{a \to \ell_1 \hat{\ell}_2 \dots \hat{\ell}_n r_n \mid n \geq 1, a \to \ell_1 r_1, r_1 \to \ell_2 r_2, \dots, r_{n-1} \to \ell_n r_n \in P\} \\ &\cup \{\hat{a} \to \hat{\ell}_1 \hat{\ell}_2 \dots \hat{\ell}_n r_n \mid n \geq 1, a \in L_\varphi, a \to \ell_1 r_1, r_1 \to \ell_2 r_2, \dots, r_{n-1} \to \ell_n r_n \in P\}. \end{aligned}$$

Theorem 2.3.4 *For each binary grammatical code* φ*, the extension* φ_{ext} *is a grammatical code for the set of chain-free trees.*

Proof. Let $\varphi = (\Sigma, P, \sigma)$ be a binary code, and let $\varphi_{ext} = (\Sigma_{ext}, P_{ext}, \sigma)$ be the extension of φ. By the construction of φ_{ext}, and since φ is deterministic, it follows that for each $n \geq 2$, and each $a \in \Sigma_{ext}$, there is exactly one production $a \to w$ in P with $|w| = n$. Hence φ_{ext} is semi-deterministic.

Let $t \in \mathrm{T}$ be a tree, and let $bin(t) = (t', \eta)$. Let $\alpha : nd(t') \to \Sigma_{ext}$ be the adapted node-labeling as described above. It follows from the definitions of $bin(t)$ and of φ_{ext} that $t[\varphi_{ext}] = (t, \alpha|_{nd(t)})$. This implies that $bin(t)$, and hence the original tree t, can be uniquely determined from $yield(t[\varphi_{ext}])$. Hence φ_{ext} is unambiguous.

Consequently, φ_{ext} is a grammatical code for T. □

Example 2.3.5 Consider the strict binary code $\varphi = (\{a, b, c, d\}, P, a)$ from Example 2.1.4(2), with production set $a \to bc$, $b \to bd$, $c \to ac$, $d \to ad$. Let φ_{ext} be the extension of φ. Then the productions of φ_{ext} are (for each $k \geq 1$)

$a \to b\hat{a}^k c$, $\quad d \to a\hat{a}^k d$,
$b \to b\hat{a}^k d$, $\quad \hat{a} \to \hat{b}\hat{a}^k c$,
$c \to a\hat{a}^k c$, $\quad \hat{b} \to \hat{b}\hat{a}^k d$.

Note that φ_{ext} is a strict code; in fact it is (a renaming of) the strict code given in Example 2.1.4(1).

Figure 2.2 gives the adapted node-labeling α for the binary refinement from Figure 2.1, and $t[\varphi_{ext}]$. □

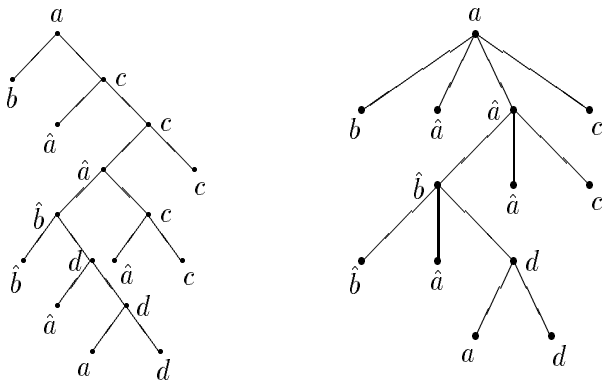


Figure 2.2: $und(bin(t))$ with adapted node-labeling and $t[\varphi_{ext}]$

Remark 2.3.6 In Definition 2.3.3, one could also refine the nodes of degree larger than 2 in a "leftmost" way, as done in Figure 2.3 for the tree t of Figure 2.1. Then an extension code is obtained where copies of right letters are added; symmetric results hold for these extensions. □

Remark 2.3.7 In [4] a way of coding node-labeled trees was discussed where the underlying tree is coded with a marked code, and the node-labels are stored in the leaves by use of a so-called "direction function". Extension codes closely correspond to such a way of coding the binary refinements of T (which are inner-labeled binary trees), provided that we extend *marked* binary codes. More precisely, let $\varphi = (\Sigma, P, \sigma)$ be a marked binary code, let $\psi : \Sigma \to \{left, right\}$ be the direction function defined by $\psi(a) = left$ if $a \in R_\varphi$ and $\psi(a) = right$ if $a \in L_\varphi$, and consider the code-word w of $bin(t)$ given by φ and ψ (as described in [4]). Then each label 1 ends up in a left leaf of $bin(t)$, and for $a \in \Sigma$, $\hat{a}$ and a in $\varphi_{ext}(t)$ correspond to letters $(a, 1)$ and $(a, 0)$ respectively in w. □

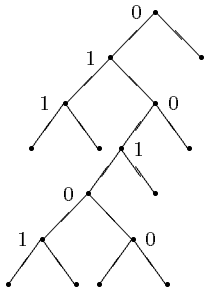


Figure 2.3: a symmetric way of refining t

So far, we have imposed no restrictions on the form of φ. The following theorem states that non-overlapping (i.e., marked) binary codes extend to non-overlapping codes for T. Moreover, a condition is given that characterizes those binary codes that extend to marked codes. We use LL_φ to denote $\{b \in L_\varphi \mid$ there exists $a \in L_\varphi$ such that $a \to bc$ in $\varphi\}$, and LR_φ to denote $\{b \in L_\varphi \mid$ there exists $a \in R_\varphi$ such that $a \to bc$ in $\varphi\}$.

Theorem 2.3.8 *Let φ be a binary grammatical code, and let φ_{ext} be the extension of φ.*
(1) φ is a marked binary code iff φ_{ext} is a non-overlapping code.
(2) φ is backwards deterministic and $LL_\varphi \cap LR_\varphi = \emptyset$ iff φ_{ext} is a marked code.

Proof. Let $\varphi = (\Sigma, P, \sigma)$ be a binary code, and let $\varphi_{ext} = (\Sigma_{ext}, P_{ext}, \sigma)$ be the extension of φ.
(1) Firstly, assume that φ is marked. It follows from the backwards determinism of φ that φ_{ext} is backwards deterministic. We show now that the right-hand sides of φ_{ext} do not overlap. From the fact that L_φ and R_φ are disjoint and that every right-hand side of φ_{ext} is in $(L_\varphi \cup L_{ext}) \cdot L_{ext}^* \cdot R_\varphi$, where $L_{ext} = \{\hat{\ell} \mid \ell \in L_\varphi\}$ we obtain that the only possibility for overlapping right-hand sides is that one right-hand side is a suffix of another right-hand side. More precisely there is a production of the form $a \to \ell_1 \hat{\ell}_2 \ldots \hat{\ell}_n r_n$ or $\hat{a} \to \hat{\ell}_1 \hat{\ell}_2 \ldots \hat{\ell}_n r_n$, with $a, \ell_1, \ldots, \ell_n, r_n \in \Sigma$, and a production of the form $\hat{b} \to \hat{\ell}_j \ldots \hat{\ell}_n r_n$, with $2 \leq j \leq n$, and $b \in L_\varphi$. This implies the existence of productions $a \to \ell_1 r_1, r_1 \to \ell_2 r_2, \ldots, r_{n-2} \to \ell_{n-1} r_{n-1}, r_{n-1} \to \ell_n r_n$ in P as well as productions $b \to \ell_j r'_j, r'_j \to \ell_{j+1} r'_{j+1}, \ldots, r'_{n-2} \to \ell_{n-1} r'_{n-1}, r'_{n-1} \to \ell_n r_n$. Since φ is backwards deterministic, it follows that $r_{n-1} = r'_{n-1}$, and hence also that $r_{n-2} = r'_{n-2}, \ldots, r_j = r'_j$, and $r_{j-1} = b$. However, $r_{j-1} = b$ contradicts the fact that $b \in L_\varphi$. Hence no right-hand sides of P_{ext} overlap. Consequently, φ_{ext} is a non-overlapping code.

Conversely, if φ_{ext} is non-overlapping, then, since all productions of φ are also productions of φ_{ext}, L_φ and R_φ are disjoint and φ is backwards deterministic – hence φ is marked.
(2) By the construction of φ_{ext}, $L_{\varphi_{ext}} = L_\varphi \cup \{\hat{\ell} \mid \ell \in LL_\varphi\}$, $M_{\varphi_{ext}} = \{\hat{\ell} \mid \ell \in LR_\varphi\}$, and $R_{\varphi_{ext}} = R_\varphi$. Note that disjointness of LL_φ and LR_φ implies disjointness of L_φ and R_φ. Consequently, $L_{\varphi_{ext}}$, $M_{\varphi_{ext}}$, and $R_{\varphi_{ext}}$ are mutually disjoint iff LL_φ and LR_φ are disjoint. By (1), if φ_{ext} is marked, then φ is backwards deterministic. Hence φ satisfies the given condition iff φ_{ext} is marked. □

Theorem 2.3.8 implies that φ is a strict binary code iff φ_{ext} is a non-overlapping code with 6 letters. These minimal non-overlapping codes obtained by extending strict codes are not

necessarily strict, see, e.g., the code from Example 2.3.9(1). Note that strict codes obtained as extensions have 3 left letters; using the symmetric extension of Remark 2.3.6 one may obtain strict codes with 3 right letters. It is impossible that both symmetric versions of the extension of a strict code are strict. More precisely, of the 24 strict binary codes 16 codes have extensions that are not strict; of the other 8 codes 4 codes yield so-called 'insertive' (see [5]) strict extensions according to Definition 2.3.3, and 4 codes yield 'insertive' strict extensions following a symmetric definition (see Remark 2.3.6). In fact the 4 binary codes giving strict extensions according to Definition 2.3.3 have the productions of Example 2.3.5 and one of the 4 letters as axiom. The production set of the obtained extensions gives, with the right choice of the axiom, one of the two strongly recursive dependent codes with 3 left letters which were discussed in [5].

Example 2.3.9

(1) Consider the strict binary code $\varphi = (\{a, b, c, d\}, P, a)$ where P consists of the productions $a \to ac$, $b \to bd$, $c \to bc$, and $d \to ad$, and let φ_{ext} be the extension of φ. Then the productions of φ_{ext} are (for each $k \geq 1$):

$$\begin{aligned} a &\to a\hat{b}^k c, & d &\to a\hat{a}^k d,\\ b &\to b\hat{a}^k d, & \hat{a} &\to \hat{a}\hat{b}^k c,\\ c &\to b\hat{b}^k c, & \hat{b} &\to \hat{b}\hat{a}^k d. \end{aligned}$$

Hence φ_{ext} is a non-overlapping code that is not strict.

(2) Consider the non-overlapping code (taken from [4]) $\varphi = (\{\ell_1, \ell_2, \ell_3, r_1, r_2, r_3\}, P, \ell_1)$ where P consists of the productions, for $k \geq 1$,

$$\begin{aligned} \ell_1 &\to \ell_1 r_3^k r_1, & r_1 &\to \ell_2 r_1^k r_3,\\ \ell_2 &\to \ell_1 r_3^k r_2, & r_2 &\to \ell_3 r_2^k r_1,\\ \ell_3 &\to \ell_2 r_1^k r_2, & r_3 &\to \ell_3 r_2^k r_3. \end{aligned}$$

This code is an example of a non-overlapping code that is not an extension code. This is easily seen by the fact that each (left- or right-) extension of a binary strict code introduces at most two letters that can occur as "middle" letters.

(3) Consider the strict code $\varphi = (\{\ell_1, \ell_2, \ell_3, m, r_2, r_3\}, P, \ell_3)$ where P consists of the productions, for $k \geq 1$,

$$\begin{aligned} m &\to \ell_1 m^k r_1, & \ell_3 &\to \ell_1 m^k r_2,\\ \ell_1 &\to \ell_2 m^k r_1, & r_1 &\to \ell_2 m^k r_2,\\ \ell_2 &\to \ell_3 m^k r_1, & r_2 &\to \ell_3 m^k r_2. \end{aligned}$$

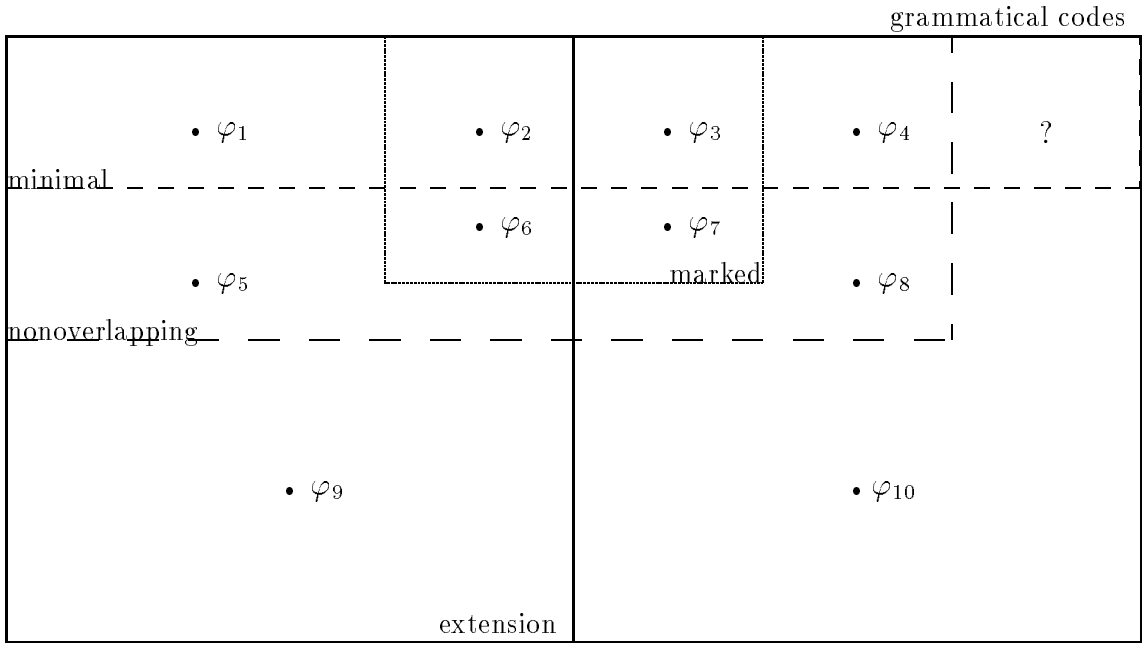
This code is an example of a strict code that is not an extension code. Note that if it would be an extension code, then it would be an extension introducing m and one of the ℓ_j as new letters. By Definition 2.3.3 this new letter ℓ_j would be ℓ_1, since $m \to \ell_1 r_1$ is a production of φ. But then also ℓ_2 would be a new letter, since $\ell_1 \to \ell_2 r_1$ is a production of φ; contradiction. □

There is an easy way to construct codes that are not extensions, by choosing the axiom in such a way that it must be a new letter. However, intuitively, the axiom is not crucial to the nature of a grammatical code. Hence we prefer to give examples of codes that are not extensions for more essential reasons.

Summarizing, we obtain the inclusion diagram of Figure 2.4 for grammatical codes of chain-free trees. The question mark denotes that we do not know whether this area is empty

or not (cf. the discussion at the end of Section 2.2). For the subclass of extension codes we *do* have that every minimal code is non-overlapping: if φ_{ext} is an extension code with 6 letters, then it is obtained from a binary code φ of 4 letters; by Theorem 2.2.7 φ is strict, and hence, by Theorem 2.3.8, φ_{ext} is non-overlapping.

For the sake of completeness we give representative codes corresponding with the diagram:
- φ_1 is the code φ_{ext} from Example 2.3.9(1),
- φ_2 is the code φ_{ext} from Example 2.3.5,
- φ_3 is the code φ from Example 2.3.9(3),
- φ_4 is the code from Example 2.3.9(2),
- φ_5 is the extension of the marked binary code with productions $a \to ad$, $b \to bd$, $c \to ce$, $d \to ae$, and $e \to be$,
- φ_6 is the extension of the marked binary code with productions $a \to ad$, $b \to bd$, $c \to be$, $d \to cd$, and $e \to ce$,
- φ_7 is the marked code with productions, for $k \geq 0$, $a \to ac^k e$, $b \to bc^k e$, $c \to ac^k f$, $d \to bc^k f$, $e \to ac^k g$, $f \to bc^k f$, and $g \to bc^k d$,
- φ_8 is the non-overlapping code with productions, for $k \geq 0$, $a \to ad^k e$, $b \to bd^k e$, $c \to ad^k f$, $d \to ce^k d$, $e \to ce^k f$, $f \to bd^k g$, $g \to ad^k g$,
- φ_9 is the extension of the binary code of Example 2.2.8,
- φ_{10} is the code with productions, for $k \geq 0$, $a \to af^k e$, $b \to bf^k d$, $c \to af^k c$, $d \to ef^k d$, $e \to bf^k c$, $f \to af^k g$, $g \to bf^k g$.



extensions of strict binary codes = minimal codes ∩ extension codes
extensions of marked binary codes = non-overl. codes ∩ extension codes
strict codes = marked codes ∩ minimal codes

Figure 2.4: inclusion diagram

The fact that φ_7, φ_8, and φ_{10} are not extensions can be shown by arguments similar to the ones used in Example 2.3.9(2) and (3). The unambiguity of φ_{10} follows by reasoning as in Example 2.2.8.

Chapter 3

Grammatical Codes of Trees and Terminally Coded Grammars

Abstract

We introduce *terminally coded (TC) grammars*, which generalize parenthesis grammars in the sense that from each word w generated by a TC grammar we can recover the unlabeled tree t underlying its derivation tree(s). More precisely, there is a length-preserving homomorphism that maps w to an encoding of t. Basic properties of TC grammars are established. For backwards deterministic TC grammars we give a shift-reduce precedence parsing method without look-ahead, which implies that TC languages can be recognized in linear time. The class of TC languages contains all parenthesis languages, and is contained in the classes of simple precedence languages and NTS languages.

Introduction

We wish to investigate context-free grammars such that every word in the generated language contains an encoding of the structure of the derivation tree for that word, where by "structure of the derivation tree" we mean the derivation tree without its labels. In particular, we require that the encoding can be obtained from the generated word by a length-preserving homomorphism. The classical example of such grammars are the parenthesis grammars ([13]). In a parenthesis grammar every production is of the form $A \rightarrow (\alpha)$where α does not contain the (terminal) symbols "(" and ")". Clearly, from a given word in the language one can reconstruct the unlabeled derivation tree for that word. In fact, an encoding of the unlabeled tree can be obtained from the generated word by a length-preserving homomorphism that leaves the parentheses as they are, and maps each other terminal letter to some fixed symbol. It should be realized, however, that the derivation trees of a parenthesis grammar have a very special structure: the first and last child of an internal node are always leaves, labeled by parentheses. In this paper we generalize the parenthesis grammars to the so-called terminally coded (TC) grammars, of which the derivation trees may have arbitrary structure. Note that, in order that the generated word contains an encoding of the unlabeled derivation tree, the encoding of the tree should have the same length as its yield. Such "length-preserving" codes were shown to exist for arbitrary (chain-free) trees in [5, 4]; here we also allow chains (of bounded

length). Note that, of course, parenthesis expressions also encode arbitrary trees (because there is a bijective correspondence between arbitrary trees and trees of the above-mentioned special structure); however, that encoding is not length-preserving.

Intuitively, the most obvious way in which a context-free grammar G can generate words that contain an encoding of the unlabeled derivation tree, is to keep track of the encoding of the tree of each sentential form. This means that the application of a production should simulate, in the encoding, the addition of children to a leaf of the tree. This "locality" of the encoding of trees suggests that the encoding itself should also be defined by way of a context-free grammar C, and that G simulates C. Such codes are called "grammatical" codes, introduced and investigated in [5, 4]. In the TC grammars of this paper we allow in particular the so-called marked codes of [4]. Note that for grammatical codes the decoding of the codeword into the tree it represents, corresponds to the parsing of that word. The marked codes are defined by grammars for which this decoding is "simple". The basic idea is to generalize the parenthesis encoding by allowing arbitrary terminals and nonterminals to act as "parentheses" that mark the beginning and end of the right-hand side of a production.

Due to the fact that each TC grammar "simulates" a marked code, for which we have a simple parsing method, the recognition of TC languages is also fairly easy. In fact, TC grammars are closely related to a natural subclass of the simple precedence grammars: the so-called very simple precedence (VSP) grammars introduced in this paper. They are the simple precedence grammars for which the handle can be detected without looking at symbols outside the handle; thus, these grammmars are also of bounded context, with bound 0. More precisely, the relationship between TC grammars and VSP grammars is as follows. Every backwards deterministic TC grammar is a VSP grammar; this implies that every backwards deterministic TC grammar can be parsed by a shift-reduce algorithm without look-ahead. Backwards deterministic TC grammars and VSP grammars generate the same class of languages, which is a proper subclass of the class of languages generated by arbitrary TC grammars. However, allowing a VSP grammar to have several initial nonterminals rather than one, the class of languages generated by TC grammars equals the class of languages generated by such VSP grammars. Thus, TC languages can be recognized in linear time. We also show that the class of TC languages is (effectively) contained in the class of NTS languages; this implies that TC grammars have a decidable equivalence problem (see [15]). The class of parenthesis languages is a proper subclass of the class of TC languages.

The structure of this paper is as follows. In Section 3.1, definitions and properties concerning grammatical codes of trees are given. The notion of terminally coded grammar is introduced in Section 3.2. Basic properties of these grammars are established. Some of them are compared with properties of parenthesis grammars. Section 3.3 concerns VSP (very simple precedence) grammars. In particular, a shift-reduce parsing algorithm for VSP grammars is presented, and it is shown that backwards deterministic TC grammars are VSP. The language classes corresponding with the above-mentioned grammar types are compared in Section 3.4.

Although this paper continues the research from [5] and [4], it is completely self-contained.

Preliminaries

In this section we give some basic notations and terminology to be used in this paper, in particular notations and terminology concerning trees and context-free grammars.

For a set Z, $\#Z$ denotes its cardinality; $\emptyset$ denotes the empty set. For sets Y and Z, $Y \subseteq Z$ denotes the inclusion of Y in Z, and $Y \subset Z$ denotes the strict inclusion of Y in Z. The cartesian product of sets X and Y is denoted by $X \times Y$, and we use $proj_1$ and $proj_2$ to denote the projections onto X and Y respectively.

For a relation $\rho \subseteq X \times Y$, $dom(\rho)$ denotes the domain of ρ, i.e., $dom(\rho) = \{x \in X \mid$ there is a $y \in Y$ such that $(x, y) \in \rho\}$, and $ran(\rho)$ denotes the range of ρ, i.e., $ran(\rho) = \{y \in Y \mid$ there is a $x \in X$ such that $(x, y) \in \rho\}$. In particular, these notations will be used with respect to (partial) functions. For a (partial) function $\varphi : X \to Y$, and for $Z \subseteq X$, $\varphi \mid_Z$ denotes the restriction of φ to Z. For (partial) functions $\varphi : X \to Y$ and $\psi : Y \to Z$, $\psi \circ \varphi$ denotes the composition of ψ and φ, i.e., $\psi \circ \varphi(x) = \psi(\varphi(x))$ for all $x \in X$.

$\mathbf{N}_+$ denotes the set of all positive natural numbers.

For a sequence x, $|x|$ denotes its length, $first(x)$ denotes the first element of x, and $last(x)$ denotes the last element of x. These notations carry over to words which are sequences of letters.

By a *projection* from one alphabet into another we mean a length-preserving homomorphism.

In this paper, by a tree we mean a finite nonempty rooted directed ordered tree.

Let t be a tree.

$nd(t)$ denotes the set of all nodes of t, $in(t)$ denotes the set of internal nodes of t, $leaf(t)$ denotes the set of leaves of t, $\langle leaf \rangle(t)$ denotes the sequence of all leaves of t ordered according to the order of t, and $root(t)$ denotes the root of t.

For $v \in nd(t)$, $sub_t(v)$ denotes the subtree of t rooted at v. For $v \in in(t)$, $t \backslash v$ denotes the tree that results from t by removing $sub_t(v)$, except v, and making v into a leaf.

For $v \in in(t)$, $ddes_t(v)$ is the set of all direct descendants of v in t, and $\langle ddes \rangle_t(v)$ is the sequence of all direct descendants of v in t (i.e., the elements of $ddes_t(v)$ ordered according to the order of t).

A *chain of* t is an edge (v, w) such that $ddes_t(v) = \{w\}$; t is *chain-free* if for each $v \in in(t)$, $\#ddes_t(v) \geq 2$.

If $x \in nd(t)$, then
x is a *leftmost child* if there exists $v \in in(t)$ such that $\#ddes_t(v) \geq 2$ and $x = first(\langle ddes \rangle_t(v))$,
x is a *rightmost child* if there exists $v \in in(t)$ such that $\#ddes_t(v) \geq 2$ and $x = last(\langle ddes \rangle_t(v))$,
x is a *middle child* if there exists $v \in in(t)$ such that $\#ddes_t(v) \geq 3$, $x \in ddes_t(v)$ and x is neither a leftmost nor a rightmost child, and
x is a *single child* if there exists $v \in in(t)$ such that (v, x) is a chain of t.

A *complete segment* (*of* $\langle leaf \rangle(t)$) is a sequence w of leaves of t such that $w = \langle ddes \rangle_t(v)$ for some $v \in in(t)$.

Trees t and t' are *isomorphic* if there is a bijection $\delta : nd(t) \to nd(t')$ such that $\delta(root(t)) = root(t')$ and for each $v \in in(t)$, if $\langle ddes \rangle_t(v) = (v_1, \ldots, v_n)$, $n \geq 1$, then $\langle ddes \rangle_{t'}(\delta(v)) = (\delta(v_1), \ldots, \delta(v_n))$. *In this paper we deal with "abstract" trees, i.e., isomorphic trees are considered to be equal.* Note that, in this way, there is a unique tree consisting of one node only.

As usual, we often use concrete trees to represent abstract trees.

A *node-labeled tree* t is a pair (t', η), where t' is a tree and $\eta : nd(t') \to \Sigma$ is a mapping, where Σ is an alphabet. For a node-labeled tree $t = (t', \eta)$, t' is the *underlying tree of* t, denoted by $und(t)$, and η is the *node-labeling function of* t, denoted by lb_t. The above notations concerning $und(t)$ are carried over to t in the obvious way, e.g., we speak of $in(t)$ and $ddes_t(v)$. In particular, for a node-labeled tree t and $v \in nd(t)$, we mean by $sub_t(v)$ the node-labeled subtree $(sub_{und(t)}(v), lb_t \mid_{nd(sub_{und(t)}(v))})$, and, for $v \in in(t)$, $t\backslash v$ denotes the node-labeled tree that results from t by removing $sub_t(v)$, except v (and its label).

Let t be a node-labeled tree. The *yield of* t, denoted by $yield(t)$, is the word $lb_t(v_1) \dots lb_t(v_n) \in \Sigma^+$, where $(v_1, \dots, v_n) = \langle leaf \rangle(t)$. With each $v \in in(t)$ we associate a production $prod_t(v) = lb_t(v) \to lb_t(v_1) \dots lb_t(v_n)$, where $(v_1, \dots, v_n) = \langle ddes \rangle_t(v)$, and we write $prod(t) = \{prod_t(v) \mid v \in in(t)\}$.

A context-free grammar (from now on abbreviated as *cf grammar*) is denoted by a 4-tuple (Σ, Δ, P, S), where Σ is the set of all (terminal and nonterminal) symbols, $\Delta \subseteq \Sigma$ is the set of terminals, $P \subseteq (\Sigma - \Delta) \times \Sigma^+$ is the set of productions, and $S \in \Sigma - \Delta$ is the initial nonterminal. For $(A, \alpha) \in P$ we use the notation $A \to \alpha$. The cf grammars in this paper are ε-free; thus, we restrict ourselves to languages that do not contain the empty word ε.

Let $G = (\Sigma, \Delta, P, S)$ be a cf grammar.

For words $\alpha, \beta \in \Sigma^*$, $\beta \Rightarrow_G \alpha$ means that β *directly derives* α *(in* G*)*, $\beta \Rightarrow_G^* \alpha$ means that β *derives* α *(in* G*)*, and $\beta \Rightarrow_G^+ \alpha$ means that β *derives* α *(in* G*) in at least one step.* β *directly derives* α *(in* G*) in a rightmost way*, notation $\beta \overset{r}{\Rightarrow}_G \alpha$, if $\beta = \gamma A w$ and $\alpha = \gamma \delta w$, where $A \to \delta \in P$, $\gamma \in \Sigma^*$ and $w \in \Delta^*$. A word $\alpha \in \Sigma^+$ is a *sentential form* if $S \Rightarrow_G^* \alpha$, and a *rightmost sentential form* if $S \overset{r}{\Rightarrow}_G^* \alpha$.

Whenever the cf grammar G is clear from the context, we write $\Rightarrow$, $\Rightarrow^*$, $\overset{r}{\Rightarrow}$ instead of $\Rightarrow_G$, $\Rightarrow_G^*$, $\overset{r}{\Rightarrow}_G$.

For $\alpha \in \Sigma^+$ and $A \in \Sigma - \Delta$, a *derivation subtree of* α *from* A *(in* G*)* is a node-labeled tree t such that $lb_t(root(t)) = A$, $yield(t) = \alpha$, and $prod(t) \subseteq P$. If $A = S$, then we say that t is a *derivation tree of* α *(in* G*)*; t is a *successful* derivation tree if $\alpha \in \Delta^+$. As usual, for $A \in \Sigma$ and $\alpha \in \Sigma^+$, there is a derivation subtree of α from A in G iff $A \Rightarrow^* \alpha$.

A cf grammar is *chain-free* if it has no productions of the form $A \to B$ with B a nonterminal. A cf grammar $G = (\Sigma, \Delta, P, S)$ is *reduced* if each symbol is useful and reachable, i.e., for each $A \in \Sigma$ there is a derivation $S \Rightarrow^* \alpha_1 A \alpha_2 \Rightarrow^* \alpha_1 \beta \alpha_2 \in L(G)$ in G, where $\alpha_1, \alpha_2 \in \Delta^*$, $\beta \in \Delta^+$. Note that each cf grammar has an equivalent reduced chain-free grammar. Note finally that the derivation trees of a chain-free cf grammar are not necessarily chain-free : they may contain chains that end in a leaf (corresponding to productions of the form $A \to a$ with $a \in \Delta$).

3.1 Grammatical codes of trees

Codings of (chain-free) trees into words that have a "grammatical" nature, i.e., that can be represented by cf grammars, were investigated in [5, 4]. The aim of this section is to present some of the basic concepts from [5, 4], generalized to codes that are defined on a subset of trees (which may have chains).

Let T be the set of all abstract trees, as defined in the preliminaries.

Definition 3.1.1 A *tree function* is a partial function $\varphi : \mathrm{T} \to \Sigma^*$, where Σ is a (finite) alphabet, and $dom(\varphi) \neq \emptyset$.

For a tree function $\varphi : \mathrm{T} \to \Sigma^*$, the alphabet Σ is denoted by $alph(\varphi)$. A letter $a \in alph(\varphi)$ is *reachable* if there exists $t \in dom(\varphi)$ such that a occurs in $\varphi(t)$. We now define the main notion of this section, viz. that of a grammatical tree-code.

Definition 3.1.2 Let $\varphi : \mathrm{T} \to \Sigma^*$ be a tree function. φ is
(1) a *tree-code* if it is injective; it is then said to be a tree-code for $dom(\varphi) \subseteq \mathrm{T}$,
(2) *length-preserving* if $|\varphi(t)| = \#leaf(t)$ for every $t \in dom(\varphi)$,
(3) *prefix-closed* if for each $t \in dom(\varphi)$ and for each $v \in in(t)$, $t\backslash v \in dom(\varphi)$,
(4) *local* if there exists a partial function $\psi : \Sigma \times \mathbf{N}_+ \to \Sigma^+$ such that if $\varphi(t) = xay$ with $x, y \in \Sigma^*$, $a \in \Sigma$, $|x| = i - 1$, then
(i) $\varphi(t{<}i, n{>})$ is defined iff $\psi(a, n)$ is defined, and
(ii) if $\varphi(t{<}i, n{>})$ is defined, then $\varphi(t{<}i, n{>}) = x\psi(a, n)y$, where $t{<}i, n{>}$ denotes the tree that results from t by adding n direct descendants to the i'th leaf of t,
(5) *grammatical* if it is length-preserving, prefix-closed, and local,
(6) a *grammatical tree-code* if it is both grammatical and a tree-code.

Note that for each prefix-closed tree function φ the one node tree is in its domain; the value of φ for the one node tree is denoted by one_φ.

Note also that if φ is a grammatical tree function with local function ψ, then for each reachable $a \in alph(\varphi)$, and for each $n \in \mathbf{N}_+$, $\psi(a, n)$ is uniquely determined by φ; if $\psi(a, n)$ is defined, then $|\psi(a, n)| = n$.

Obviously, the length-preserving property and the finiteness of Σ imply that there is no grammatical tree-code φ for the set of all trees, i.e., with $dom(\varphi) = \mathrm{T}$. In [5] the rather surprising fact is shown that grammatical tree-codes do exist for the set of all chain-free trees, even with $\#\Sigma = 6$ (see Example 3.1.14). We first give some well-known examples of tree-codes, through which the reader may appreciate the result of [5].

Example 3.1.3
(1) Consider the function $\varphi_{par1} : \mathrm{T} \to \{(,), m\}^*$ defined as follows : for $t \in \mathrm{T}$, $\varphi_{par1}(t)$ is the word obtained by writing an m for each leaf of t, and putting the symbol "(" before each leftmost child and the symbol ")" after each rightmost child. Figure 3.1 gives an example of a tree t_1 and its code-word $\varphi_{par1}(t_1)$.

φ_{par1} is the usual parenthesis code for trees, with m standing for $()$; it is a tree-code that is prefix-closed (since $dom(\varphi_{par1}) = \mathrm{T}$) but not length-preserving.
(2) Consider the partial function $\varphi_{par2} : \mathrm{T} \to \{(,), m\}^*$ defined as follows : for $t \in \mathrm{T}$ such that t is chain-free and each leftmost and each rightmost child is a leaf, $\varphi_{par2}(t)$ is the word obtained by writing m for each leaf that is a middle child, (for each leaf that is a leftmost child, and) for each leaf that is a rightmost child. In Figure 3.2 a tree t_2 and its code-word $\varphi_{par2}(t_2)$ is given. Note that $\varphi_{par2}(t_2) = \varphi_{par1}(t_1)$.

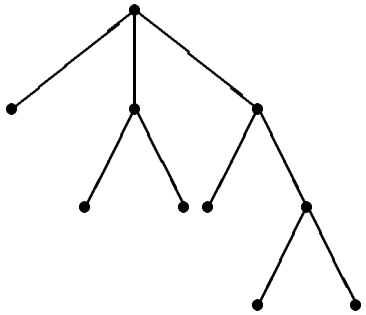


Figure 3.1: $\varphi_{par1}(t_1) = (m(mm)(m(mm)))$

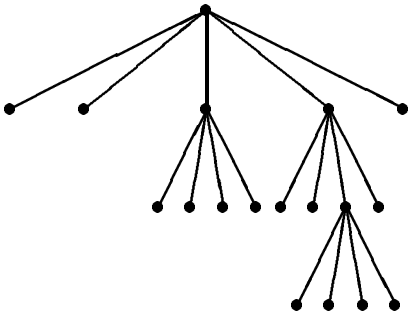


Figure 3.2: $\varphi_{par2}(t_2) = (m(mm)(m(mm)))$

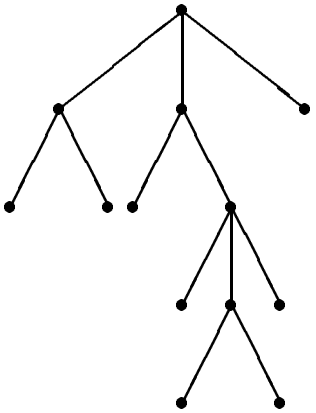


Figure 3.3: $\varphi_{Dew} = (1 \cdot 1)(1 \cdot 2)(2 \cdot 1)(2 \cdot 2 \cdot 1)(2 \cdot 2 \cdot 2 \cdot 1)(2 \cdot 2 \cdot 2 \cdot 2)(2 \cdot 2 \cdot 3)(3)$

φ_{par2} is a tree-code for the set of chain-free trees such that all leftmost and rightmost children are leaves; it is length-preserving, prefix-closed, and local with local function ψ_{par} defined by $\psi_{par}(m,n) =$(m^{n-2})for each $n \in \mathbf{N}_+ - \{1\}$.

Hence φ_{par2} is a grammatical tree-code.

(3) In this example we temporarily drop the restriction that Σ is finite and take $\Sigma = \mathbf{N}^*$. Consider the usual Dewey notation for trees $\varphi_{Dew} : T \to (\mathbf{N}^*)^*$ defined as follows. Let $t \in T$ be a tree. For each $v \in in(t)$, label the edges from v to its direct descendants by $1, 2, .., s$, where $s = \#ddes_t(v)$, according to the order of t. Then $\varphi_{Dew}(t)$ is the word obtained by writing for each leaf u of t the word in Σ that labels the path from the root to u. In particular, $one_\varphi = (\varepsilon)$. As an example, consider the tree t_3 and its Dewey code-word in Figure 3.3.

The Dewey code is injective, length-preserving, prefix-closed, and local with local function ψ_{Dew} defined by $\psi_{Dew}(w,n) = (w1)(w2)\dots(wn)$ for each $w \in \mathbf{N}^*$ and each $n \in \mathbf{N}_+$. Its domain is T, i.e., it codes all trees. However, φ_{Dew} is not a grammatical tree-code, since the alphabet $\mathbf{N}^*$ is not finite. □

In this paper (as in [5, 4]), we will consider grammatical tree-codes that are similar to φ_{par2}. Since φ_{par2} has such a specific domain, we wish to have more general grammatical tree-codes that still are easy to decode.

Grammatical tree functions are called "grammatical" because they can be represented by context-free grammars, as will be shown in what follows. *In this section (only) we assume that context-free grammars are allowed to have infinitely many productions.* We are interested in the sentential forms generated by a cf grammar instead of the terminal words. In fact, terminals will not play an essential role, and one might even assume that there are none (or, equivalently, that we work with OS systems (see, e.g., [5, 4]). Thus, we also do not assume cf grammars to be reduced. Unless specified otherwise, when we speak of a derivation tree we mean a derivation tree of a sentential form from the initial nonterminal (cf. the Preliminaries).

Definition 3.1.4 A cf grammar (Σ, Δ, P, S) is *forwards deterministic* if for every $A \in \Sigma$ and every $n \in \mathbf{N}_+$ there is at most one production $A \to \alpha$ in P with $|\alpha| = n$.

In [5, 4] this notion is used for grammars of which all right-hand sides have length > 1; there it is called "semi-deterministic".

We denote a forwards deterministic cf grammar by fd-cf grammar. Note that for an fd-cf grammar, if t_1 and t_2 are derivation trees and $und(t_1) = und(t_2)$, then $t_1 = t_2$. In fact, a reduced cf grammar is forwards deterministic if and only if the mapping und is injective on derivation trees. Each fd-cf grammar defines a (grammatical) tree function in a natural way as follows.

Definition 3.1.5 Let $G = (\Sigma, \Delta, P, S)$ be an fd-cf grammar. The *tree function of* G is the partial function $\varphi_G : \mathrm{T} \to \Sigma^*$ such that $dom(\varphi_G) = \{und(t) \mid t \text{ is a derivation tree in G}\}$, and for $t \in dom(\varphi_G)$, $\varphi_G(t) = yield(t[G])$, where $t[G]$ is the unique derivation tree in G such that $und(t[G]) = t$.

Lemma 3.1.6 *A tree function φ is grammatical iff there exists an fd-cf grammar G such that $\varphi = \varphi_G$.*

Proof. Suppose that $\varphi : \mathrm{T} \to \Sigma^*$ is a grammatical tree function with local function ψ. Let $G = (\Sigma, \emptyset, P, S)$ be the cf grammar such that

$$P = \{a \to x \mid a \text{ is reachable, } \psi(a,n) = x \text{ for some } n \in \mathbf{N}_+\}$$

and $S = one_\varphi$. Clearly, G is forwards deterministic. Also, by Conditions 3 and 4(i) of Definition 3.1.2, $dom(\varphi) = dom(\varphi_G)$, and by Condition 4(ii) of Definition 3.1.2, for each tree $t \in dom(\varphi_G)$, $\varphi_G(t) = yield(t[G]) = \varphi(t)$.

Suppose that φ is a tree function such that $\varphi = \varphi_G$ for some fd-cf grammar G. Then, for each tree t, $|\varphi(t)| = |yield(t[G])| = \#leaf(t)$. Hence φ is length-preserving.

For each $t \in dom(\varphi)$, $t[G]$ is a derivation tree in G, and hence for each $v \in in(t)$, $t[G]\backslash v$ is a derivation tree in G. Since $und(t[G]\backslash v) = t\backslash v$, $t\backslash v \in dom(\varphi)$. Hence φ is prefix-closed.

Let ψ be defined as follows : for $a \in \Sigma$ and $n \in \mathbf{N}_+$, $\psi(a,n) = x$ such that $|x| = n$ and $a \to x$ is a production in G. The partial function ψ is well-defined, because G is forwards deterministic. Clearly φ and ψ satisfy Conditions 4(i) and 4(ii) from Definition 3.1.2. Hence φ is local, and consequently φ is grammatical. □

A cf grammar will be called *unambiguous* if the mapping *yield* is injective on derivation trees, i.e., for all derivation trees t_1 and t_2, if $yield(t_1) = yield(t_2)$ then $t_1 = t_2$. This definition is non-standard, because it also concerns unsuccessful derivation trees for sentential forms; however, for reduced cf grammars it is equivalent with the usual definition.

Lemma 3.1.7 *For every fd-cf grammar G, φ_G is injective iff G is unambiguous.*

Proof. Let $G = (\Sigma, \Delta, P, S)$ be an fd-cf grammar.

Suppose that φ_G is injective. Let t_1 and t_2 be derivation trees in G such that $yield(t_1) = yield(t_2)$. Since G is forwards deterministic, $t_1 = t_2$ iff $und(t_1) = und(t_2)$. From $\varphi_G(und(t_1)) = yield(t_1) = yield(t_2) = \varphi_G(und(t_2))$ it follows that $und(t_1) = und(t_2)$, since φ_G is injective. Hence $t_1 = t_2$. Consequently, G is unambiguous.

Suppose now that G is unambiguous. If $t_1, t_2 \in dom(\varphi_G)$ are such that $\varphi_G(t_1) = \varphi_G(t_2)$, i.e., $yield(t_1[G]) = yield(t_2[G])$, then it follows by the unambiguity of G that $t_1[G] = t_2[G]$. Hence $t_1 = t_2$, and φ_G is injective. □

By Lemmas 3.1.6 and 3.1.7 we have the following result.

Proposition 3.1.8 *A tree function φ is a grammatical tree-code iff there exists an unambiguous fd-cf grammar G such that φ is the tree function of G.*

Because of Proposition 3.1.8 we will also say that an unambiguous fd-cf grammar G *is* a grammatical tree-code (and we will often use φ for both G and φ_G). Note that G is a tree-code for $\{und(t) \mid t \text{ is a derivation tree in } G\}$.

Consequently, investigating grammatical tree-codes amounts to investigating unambiguity (of fd-cf grammars). One well-known way of getting unambiguity is to require that the cf grammar can be parsed deterministically (but note that we need a parsing method that works for all sentential forms, not only for rightmost or leftmost ones). Such a parsing method is then a way of *decoding* the code $\varphi_G : \mathrm{T} \to \Sigma^*$, i.e, given a word $w \in \Sigma^*$, to find a tree

$t \in \mathrm{T}$ such that $\varphi_G(t) = \mathrm{w}$. We consider (as in [5, 4]) codes that can be parsed in a very easy fashion, i.e., that have a simple decoding mechanism. For a cf grammar $G = (\Sigma, \Delta, P, S)$, we use L_G to denote the set $\{X \in \Sigma \mid X = \mathit{first}(\alpha)$ for some $A \to \alpha$ in $P\}$, and R_G to denote $\{X \in \Sigma \mid X = \mathit{last}(\alpha)$ for some $A \to \alpha$ in $P\}$.

Definition 3.1.9 A cf grammar $G = (\Sigma, \Delta, P, S)$ is *bracketed* if
(i) for every $A \to \alpha$ in P, $\alpha \in L_G(\Sigma - L_G)^* \cap (\Sigma - R_G)^* R_G$, and
(ii) there is no derivation $S \Rightarrow^+ S$ in G.

For readers familiar with [7], we stress that the name "bracketed" is used here for a notion that is related to but more general than the one in [7].

Intuitively, L_G is the set of left-brackets and R_G the set of right-brackets. Thus, tree functions represented by bracketed cf grammars generalize the parenthesis code φ_{par2} (see Example 3.1.13); note that L_G and R_G may contain both nonterminals and terminals.

Clearly, if a cf grammar is bracketed, one can easily detect the right-hand sides of productions in sentential forms. Thus, such grammars can be parsed bottom-up if one additionally requires backwards determinism. Condition (ii) of Definition 3.1.9 is used to determine when the parsing should halt.

A cf grammar is *backwards deterministic* (and is called a bd-cf grammar) if distinct productions have distinct right-hand sides. In the next lemma a property of bracketed bd-cf grammars is given which implies the unambiguity of these grammars (Proposition 3.1.11; cf. Lemma 1.1.24 and Theorem 1.1.26 of [4]). This property embodies the defining property of the so-called cf *grammars of bounded context*(0,0) (abbreviated BC(0,0) grammars), see [6]. The lemma here is in fact a special case of the characterization of BC(0,0) grammars given in [6].

Lemma 3.1.10 *Let $G = (\Sigma, \Delta, P, S)$ be a bracketed bd-cf grammar.*
If t is a derivation subtree of $\alpha\beta\gamma$ from A, and t' is a derivation subtree of β from B, with $\beta \in \Sigma^+$, $\alpha, \gamma \in \Sigma^*$, and $A, B \in \Sigma - \Delta$ such that there is no derivation $B \Rightarrow^+ A$, then there is a $v \in nd(t)$ such that $sub_t(v) = t'$ and $yield(t\backslash v) = \alpha B \gamma$.

Proof. We will prove the following claim using induction on $\#nd(t')$.
If t is a derivation subtree of $\alpha\beta\gamma$ from A, and t' is a derivation subtree of β from B, then one of the following cases holds :
(i) there is a $v \in nd(t)$ such that $sub_t(v) = t'$ and $yield(t\backslash v) = \alpha B \gamma$, or
(ii) there is a $v' \in nd(t')$ such that $sub_{t'}(v') = t$ and $yield(t'\backslash v') = A$.

If $\#nd(t') = 1$, then (i) holds.

Suppose that for some $n \geq 1$, the claim holds for all derivation subtrees t' such that $\#nd(t') \leq n$.

Let t be a derivation subtree of $\alpha\beta\gamma$ from A, and let t' be a derivation subtree of β from B with $n+1$ nodes. Let $(w_1, \ldots, w_k) = \langle ddes \rangle_{t'}(root(t'))$, $k \geq 1$, and let $B_j = lb_{t'}(w_j)$ for $j = 1, \ldots, k$. Hence $B \to B_1 \ldots B_k \in P$. By the inductive hypothesis we have for each of the $sub_{t'}(w_j)$ that either (i) or (ii) holds.

Suppose first that case (ii) holds for one of these subtrees, say $sub_{t'}(w_\ell)$ with $\ell \in \{1, \ldots, k\}$, i.e., there is a node w in $sub_{t'}(w_\ell)$ such that $sub_{t'}(w) = t$ and $yield(sub_{t'}(w_\ell)\backslash w) = A$. Since

$yield(t') = \beta$ and $yield(sub_{t'}(w_\ell)) = yield(t) = \alpha\beta\gamma$, it follows that $k = 1$ (and $\alpha = \gamma = \varepsilon$), and hence t' satisfies (ii).

Suppose now that for all $sub_{t'}(w_j)$ case (i) holds. Hence there exist $v_1, \ldots, v_k \in nd(t)$ such that $sub_t(v_j) = sub_{t'}(w_j)$ for $j = 1, \ldots, k$ and the tree $t'' = (\ldots((t\backslash v_1)\backslash v_2)\ldots\backslash v_k)$ is a derivation subtree from A with yield $\alpha B_1 B_2 \ldots B_k \gamma$. If t'' is the one node tree, then it must be that $k = 1$ and $t = sub_t(v_1) = sub_{t'}(w_1)$. Hence in that case t' satisfies (ii). Assume now that t'' has more than one node.

Since G satisfies Definition 3.1.9(i), for each derivation subtree s and for each $u \in nd(s)$, $u \neq root(s)$,
u is a leftmost child iff $lb_s(u) \in L_G - R_G$,
u is a middle child iff $lb_s(u) \in \Sigma - (L_G \cup R_G)$,
u is a rightmost child iff $lb_s(u) \in R_G - L_G$, and
u is a single child iff $lb_s(u) \in L_G \cap R_G$
(the only-if-directions are obvious; the if-directions then follow from the fact that the four mentioned sets form a partition of Σ).
By the above remark, if $k = 1$, then B_1 labels a single leaf of t'', which is a complete segment. If $k \geq 2$, then $B_1 \in L_G - R_G$, $B_j \in \Sigma - (L_G \cup R_G)$ for $j = 2, \ldots, k-1$, and $B_k \in R_G - L_G$. Hence $B_1 \ldots B_k$ labels a complete segment of $\langle leaf \rangle(t'')$, which means that $(v_1, \ldots, v_k)$ is a complete segment.

Let $v \in in(t'')$ be the direct ancestor of $(v_1, \ldots, v_k)$ in t''. Since G is backwards deterministic, v is labeled by B. By the construction of t'', t is recovered from t'' by adding $sub_t(v_j) = sub_{t'}(w_j)$ to v_j for $j = 1, \ldots, k$. It follows that $t\backslash v$ is a derivation subtree of $\alpha B \gamma$ from A, and that $sub_t(v) = t'$. Thus, case (i) holds, and the induction proof is completed.

Clearly, the lemma follows from the above claim. □

Proposition 3.1.11 *Every bracketed bd-cf grammar is unambiguous.*

Proof. Let $G = (\Sigma, \Delta, P, S)$ be a bracketed bd-cf grammar. Assume that t_1 and t_2 are derivation trees of $\beta \in \Sigma^+$. By Definition 3.1.9(ii), there is no derivation $S \Rightarrow^+ S$. Then, by Lemma 3.1.10, there is a $v_1 \in nd(t_1)$ such that $sub_{t_1}(v_1) = t_2$, and, vice versa, there is a $v_2 \in nd(t_2)$ such that $sub_{t_2}(v_2) = t_1$. It follows that $t_1 = t_2$. Consequently, G is unambiguous. □

Thus, one (natural) way of obtaining a grammatical tree-code is to require that an fd-cf grammar is bracketed and backwards deterministic.

Definition 3.1.12 A *marked tree-code* is a cf grammar that is forwards deterministic, backwards deterministic, and bracketed.

Example 3.1.13 The parenthesis code φ_{par2} from Example 3.1.3(2) is represented by the cf grammar $(\Sigma, \emptyset, P, m)$, with $\Sigma = \{m, (,)\}$ and $P = \{m \rightarrow (m^k) \mid k \geq 0\}$. Clearly, this grammar is backwards deterministic, forwards deterministic, and bracketed. Hence φ_{par2} is a marked tree-code. □

The notion of marked tree-code given in Definition 3.1.12 is closely related to the notion of *marked code* from [4]. As a matter of fact, a marked code in the sense of [4] is a marked tree-code φ such that $dom(\varphi)$ is the set of all chain-free trees and $one_\varphi \notin L_\varphi \cup R_\varphi$ (the latter condition is a stronger version of Condition (ii) of Definition 3.1.9). The *strict codes* from [5] are marked tree-codes for the set of all chain-free trees with an alphabet of precisely 6 letters, where again the initial symbol is outside $L_\varphi \cup R_\varphi$.

Example 3.1.14 It is very easy to verify that the grammar $\varphi_1 = (\Sigma_1, \emptyset, P_1, m)$ is a marked tree-code for the chain-free trees, where $\Sigma_1 = \{\ell_1, \ell_2, \ell_3, r_1, r_2, m\}$ and P_1 consists of the following productions, with $k \geq 0$:

$$\ell_1 \to \ell_1 m^k r_2, \qquad m \to \ell_1 m^k r_1,$$
$$\ell_2 \to \ell_2 m^k r_1, \qquad r_1 \to \ell_3 m^k r_1,$$
$$\ell_3 \to \ell_2 m^k r_2, \qquad r_2 \to \ell_3 m^k r_2.$$

Hence φ_1 is a strict code in the sense of [5].

Strict codes can easily be extended to marked tree-codes for the set of trees such that each chain ends in a leaf. E.g., let $\varphi_1 = (\Sigma_1, \emptyset, P_1, m)$ be the strict code defined above. Define $\varphi_2 = (\Sigma_2, \emptyset, P_2, m)$, where $\Sigma_2 = \Sigma_1 \cup \{\ell_1', \ell_2', \ell_3', r_1', r_2', m'\}$, and $P_2 = P_1 \cup \{c \to c' \mid c \in \Sigma_1\}$.

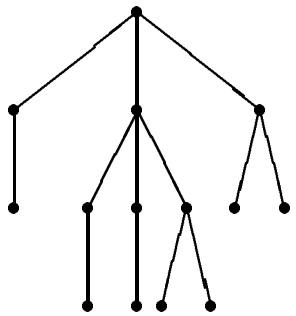


Figure 3.4: the tree t

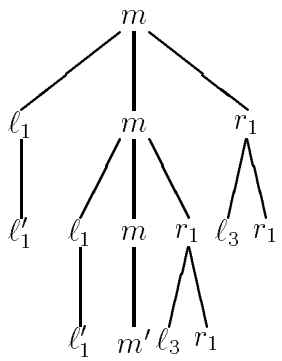


Figure 3.5: the coded version $t[\varphi_2]$

Clearly, φ_2 is backwards deterministic, forwards deterministic, and bracketed. Hence φ_2 is a marked tree-code, and $dom(\varphi_2) = \{t \in \mathrm{T} \mid \text{for each } v \in in(t),\ ddes_t(v) = \{w\} \text{ implies } w \in leaf(t)\}$.

The tree t of Figure 3.4 is in $dom(\varphi_2)$; $t[\varphi_2]$ is given in Figure 3.5 (recall that $t[\varphi_2]$ is a derivation tree in the cf grammar φ_2, as in Definition 3.1.5). Hence $\varphi_2(t) = \ell_1'\ell_1' m'\ell_3 r_1 \ell_3 r_1$. □

3.2 Terminally coded grammars

There are several methods to define classes of cf grammars that can easily be analysed. One of these is to include in some way information about the derivation tree in each word that is generated. Parenthesis grammars are an example of such a class of cf grammars. Recall that in a parenthesis grammar all productions have the form $A \rightarrow(\alpha)$, see, e.g., [13] or [14, Section VIII.3]. The parentheses can be seen as additional symbols that "code" the structure of the derivation trees of the cf grammar with productions $A \rightarrow \alpha$ (cf. the code φ_{par1} from Example 3.1.3(1)). Interpreting the parenthesis code as φ_{par2} (from Example 3.1.3(2)), one might also say that each parenthesis expression generated by a parenthesis grammar codes the structure of its *own* derivation tree (where the terminals that are not parentheses are identified, or, projected onto the codeletter m). However, the derivation trees of parenthesis grammars are of a very specific form.

In this section, a class of grammars is defined in which the structure of derivation trees is coded by marked tree-codes. We assume that given a derivation tree of a word w, the code-word of its structure can be obtained from w by a projection from the terminals of the grammar onto letters of the coding alphabet. This generalizes the notion of parenthesis grammar in the sense that derivation trees may have any form, and still no additional symbols are needed.

Due to the use of parentheses, parenthesis languages can be recognized very easily in linear time. Also, the equivalence problem for parenthesis grammars is decidable. As for the grammars introduced here, in Section 3.3 we will consider the parsing of these grammars, and show that the corresponding languages can be recognized in linear time. In Section 3.4 we obtain that their equivalence problem is decidable.

Definition 3.2.1
(1) Let φ be a marked tree-code. A cf grammar $G = (\Sigma, \Delta, P, S)$ is *terminally coded by* φ if there is a projection $\chi : \Delta \rightarrow alph(\varphi)$ such that for each $w \in L(G)$ and each derivation tree t of w in G, $und(t) \in dom(\varphi)$ and $\varphi(und(t)) = \chi(w)$.
(2) A cf grammar G is *terminally coded* (abbreviated TC) if there is a marked tree-code φ such that G is terminally coded by φ.
(3) A cf language K is *terminally coded* (TC) if there is a terminally coded cf grammar G such that $K = L(G)$.

For a TC grammar G we will use φ to denote the code by which it is terminally coded without always mentioning this explicitly.

Note that a cf grammar G is terminally coded by φ iff the reduced version of G is terminally coded by φ. *In the remainder of the paper we assume all cf grammars to be reduced*, except of course the fd-cf grammars that are used to represent grammatical tree-codes.

Example 3.2.2
(1) Let $G = (\Sigma, \Delta, P, S)$ be the cf grammar with
$\Delta = \{a, b, c, d\}$,
$\Sigma = \{S, A, B\} \cup \Delta$,
$P = \{S \rightarrow \text{ ASB}, S \rightarrow d, A \rightarrow a, B \rightarrow bc\}$.
The grammar G is terminally coded by the code φ_2 from Example 3.1.14. This is easily seen by taking $\chi : \Delta \rightarrow alph(\varphi_2)$ as follows :
$\chi(a) = \ell_1'$,
$\chi(b) = \ell_3$,
$\chi(c) = r_1$,
$\chi(d) = m'$.
E.g., consider the derivation tree of *aadbcbc* in Figure 3.6. The underlying tree is the tree t in Figure 3.4. In Example 3.1.14 we saw that $\varphi_2(t) = \ell_1'\ell_1'm'\ell_3r_1\ell_3r_1$. Hence indeed, $\chi(aadbcbc) = \varphi_2(t)$.

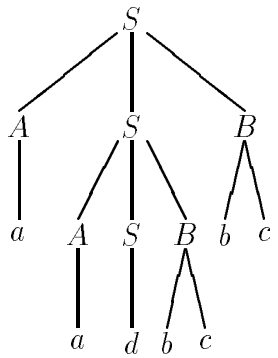


Figure 3.6: a derivation tree of *aadbcbc*

(2) The language $\{a^n d(bc)^n \mid n \geq 1\}$ is terminally coded, since it is generated by the grammar G from (1). □

Example 3.2.3 For each regular language $L \subseteq \Delta^*$ the language $\#L\$$ is terminally coded, where $\{\#, \$\} \cap \Delta = \emptyset$. This is seen as follows. Let φ be the marked tree-code such that $one_\varphi = m$, and $m \rightarrow \ell_1 r, r \rightarrow \ell_2 r$ are the productions of φ. Now let L be a regular language, and let $G = (\Sigma, \Delta, P, S)$ be a right-linear grammar for L. Then G can be transformed into a cf grammar $\overline{G} = (\Sigma, \Delta \cup \{\#, \$\}, \overline{P}, \overline{S})$ for $\#L\$$, where $\overline{S} \notin \Sigma$ and
$\overline{P} = \{A \rightarrow aB \mid A \rightarrow aB \in P\} \cup \{A \rightarrow a\$ \mid A \rightarrow a \in P\} \cup \{\overline{S} \rightarrow \#S\}$.
Define the projection $\chi : \Delta \cup \{\#, \$\} \rightarrow alph(\varphi)$ by $\chi(\#) = \ell_1$, $\chi(\$) = r$, $\chi(a) = \ell_2$ for each $a \in \Delta$. Then $\overline{G}$ is terminally coded by φ. □

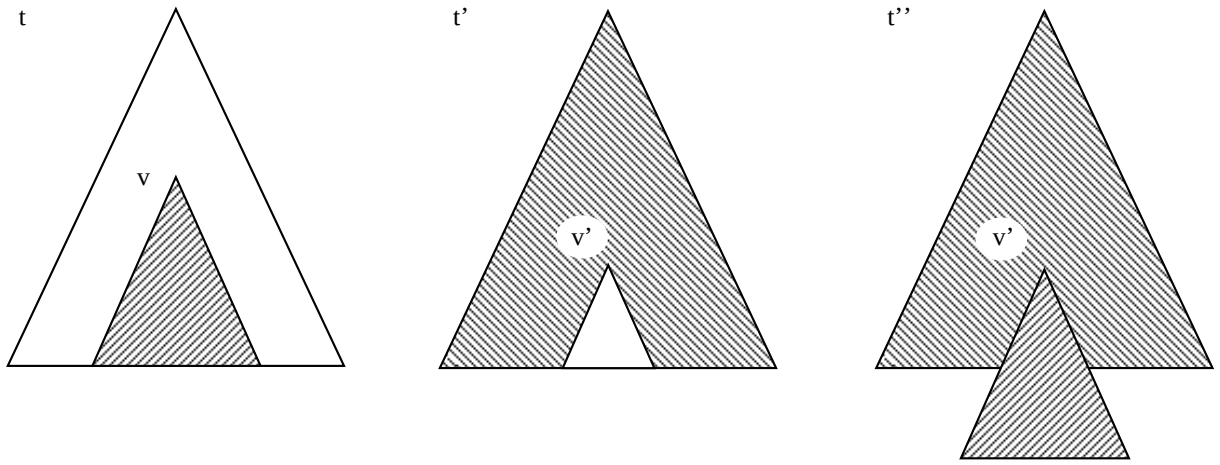


Figure 3.7: the trees t,t', and t''

Examples of cf languages that are *not* TC will be given later. The following theorem says that the productions of a terminally coded grammar implicitly contain productions of the code.

Theorem 3.2.4 *Let φ be a marked tree-code.*
A cf grammar $G = (\Sigma, \Delta, P, S)$ is terminally coded by φ iff there is a projection $\chi : \Sigma \to alph(\varphi)$ such that
(i) $\chi(S) = one_\varphi$,
(ii) if $A \to \alpha \in P$, then $\chi(A) \to \chi(\alpha)$ is a production of φ.

Proof. Let $G = (\Sigma, \Delta, P, S)$ be a cf grammar.

Suppose first that G is terminally coded by φ. Hence for each successful derivation tree (t, η) in G, the underlying tree t is in $dom(\varphi)$, and so $t[\varphi]$ is defined and $yield(t[\varphi]) = \chi(yield(t, \eta))$.

> **Claim 3.2.5** *If (t, η) and (t', η') are successful derivation trees in G, and $v \in in(t)$, $v' \in in(t')$ are such that $\eta(v) = \eta'(v')$, then $lb_{t[\varphi]}(v) = lb_{t'[\varphi]}(v')$.*
>
> **Proof.** Let (t, η), (t', η') be successful derivation trees in G, and let $v \in in(t)$, $v' \in in(t')$ be such that $\eta(v) = \eta'(v')$. Let (t'', η'') be the node-labeled tree constructed from (t', η') by replacing $sub_{(t',\eta')}(v')$ by $sub_{(t,\eta)}(v)$, where v and v' are identified. Since $\eta(v) = \eta'(v')$, (t'', η'') is well-defined; it is a successful derivation tree in G, and hence $t'' \in dom(\varphi)$. The situation is sketched in Figure 3.7.
>
> Since $t' \backslash v' = t'' \backslash v'$, and φ is forwards deterministic, it follows that $lb_{t'[\varphi]}(v') = lb_{t''[\varphi]}(v')$.
>
> Since G is terminally coded by φ,
>
> $$\begin{aligned} yield(sub_{t''[\varphi]}(v')) &= \chi(yield(sub_{(t'',\eta'')}(v')) \\ &= \chi(yield(sub_{(t,\eta)}(v)) = yield(sub_{t[\varphi]}(v)). \end{aligned}$$

Since $sub_{t''[\varphi]}(v')$ and $sub_{t[\varphi]}(v)$ have the same yield and the same underlying tree, backwards determinism of φ implies that they are the same. Hence, in particular, $lb_{t''[\varphi]}(v') = lb_{t[\varphi]}(v)$.

Consequently, $lb_{t[\varphi]}(v) = lb_{t''[\varphi]}(v') = lb_{t'[\varphi]}(v')$. □

Now define $\chi : \Sigma \to alph(\varphi)$ as follows. $\chi \mid_\Delta$ is the mapping χ as in Definition 3.2.1(1). For $A \in \Sigma - \Delta$, $\chi(A) = lb_{t[\varphi]}(v)$, where (t, η) is a successful derivation tree and $v \in in(\mathrm{t})$ is such that $\eta(v) = A$. By Claim 3.2.5 (and because G is reduced), χ is well-defined. Clearly $\chi(S) = one_\varphi$.

Let $A_0 \to A_1 \dots A_n \in P$ with $A_i \in \Sigma$ for $i = 0, \dots, n$. Let (t, η) be the derivation tree of a derivation in which the production $A_0 \to A_1 \dots A_n$ is applied. Let $v_0, \dots, v_n \in nd(t)$ be such that $\eta(v_i) = A_i$ for $i = 0, \dots, n$ and $\langle ddes \rangle_t(v_0) = (v_1, \dots, v_n)$. Then, by the definition of χ, $lb_{t[\varphi]}(v_i) = \chi(A_i)$ for $i = 0, \dots, n$. Hence, since $t[\varphi]$ is a derivation tree in φ, $\chi(A_0) \to \chi(A_1) \dots \chi(A_n)$ is a production of φ.

Hence one direction of Theorem 3.2.4 has been proved.

Assume now that there is a mapping χ as described in the statement of the theorem. Then we must prove that G is terminally coded by φ. We claim that $\chi \mid_\Delta$ satisfies the conditions in Definition 3.2.1(1).

Let $w \in L(G)$, and let (t, η) be a derivation tree of w.

Claim 3.2.6 *The node-labeled tree $(t, \chi \circ \eta)$ is a derivation tree of $\chi(w)$ in φ.*

Proof. By Condition (i), $\chi \circ \eta(root(t)) = \chi(S) = one_\varphi$. Let $v_0 \in in(t)$ with $\langle ddes \rangle_t(v_0) = (v_1, \dots, v_n)$, $n \geq 1$, and let $A_i \in \Sigma$ be such that $\eta(v_i) = A_i$ for $i = 0, \dots, n$. Since (t, η) is a derivation tree in G, $A_0 \to A_1 \dots A_n \in P$. It follows by Condition (ii) that $prod_{(t, \chi \circ \eta)}(v) = \chi \circ \eta(v) \to \chi \circ \eta(v_1) \dots \chi \circ \eta(v_n) = \chi(A_0) \to \chi(A_1) \dots \chi(A_n)$ is a production of φ. Clearly, $yield(t, \chi \circ \eta) = \chi(yield(t, \eta)) = \chi(w)$. Hence $(t, \chi \circ \eta)$ is a derivation tree of $\chi(w)$ in φ. □

By Claim 3.2.6, $t \in dom(\varphi)$, and $(t, \chi \circ \eta) = t[\varphi]$. Moreover, $\varphi(t) = yield(t[\varphi]) = yield(t, \chi \circ \eta) = \chi(w)$.

Consequently, G is terminally coded by φ.

This concludes the proof of the theorem. □

Example 3.2.7

(1) In Example 3.2.2 it was shown that the cf grammar G given there is terminally coded by φ_2 from Example 3.1.14. The following projection χ (which extends the χ of Example 3.2.2) satisfies the conditions of Theorem 3.2.4 w.r.t. G and φ_2 :

$\chi(a) = \ell_1'$, $\chi(S) = m$,
$\chi(b) = \ell_3$, $\chi(A) = \ell_1$,
$\chi(c) = r_1$, $\chi(B) = r_1$.
$\chi(d) = m'$,

(2) For each right-linear grammar $G = (\Sigma, \Delta, P, S)$, the cf grammar $\overline{G}$ constructed from G as in Example 3.2.3 is terminally coded by the marked tree-code φ given in Example 3.2.3. The projection χ defined by $\chi(\#) = \ell_1$, $\chi(\$) = r$, $\chi(a) = \ell_2$ for each $a \in \Delta$, $\chi(\overline{S}) = m$, $\chi(A) = r$ for each $A \in \Sigma - \Delta$ satisfies the conditions of Theorem 3.2.4 w.r.t. $\overline{G}$ and φ. □

This result provides a way to extend the definition of terminally coded grammars to OS systems (i.e., cf grammars that may have infinitely many productions and no terminals). In particular, each marked tree-code is then terminally coded by itself.

For a cf grammar $G = (\Sigma, \Delta, P, S)$ and a projection $\chi : \Sigma \to \Gamma$, where Γ is some alphabet, we define $\chi(G)$ as the cf grammar $(\Gamma, \emptyset, \chi(P), \chi(S))$, where
$\chi(P) = \{\chi(A) \to \chi(\alpha) \mid A \to \alpha \in P\}$. Note that $\chi(G)$ does not have terminals, although G may have terminals.

Theorem 3.2.8 *A cf grammar $G = (\Sigma, \Delta, P, S)$ is terminally coded iff there is a projection $\chi : \Sigma \to \Sigma$ such that $\chi(G)$ is a marked tree-code.*

Proof. Let $G = (\Sigma, \Delta, P, S)$ be a cf grammar, and let χ be a projection from Σ into itself such that $\chi(G)$ is a marked tree-code. Clearly, by the construction of $\chi(G)$, the projection χ satisfies the conditions of Theorem 3.2.4 w.r.t. G and $\chi(G)$. Hence G is terminally coded by $\chi(G)$.

Suppose now that G is terminally coded by some marked tree-code φ. Then there is a projection χ from Σ to $alph(\varphi)$ as in Theorem 3.2.4. Since the productions of $\chi(G)$ are productions of φ and the initial nonterminal of $\chi(G)$ is one_φ, it follows that $\chi(G)$ is backwards deterministic, forwards deterministic, and bracketed, i.e., $\chi(G)$ is a marked tree-code. Now let κ be an injective function from $\chi(\Sigma)$ into Σ. It follows by the injectivity of κ that $\kappa(\chi(G))$ is also a marked tree-code. Hence $\kappa \circ \chi$ is a projection from Σ into Σ such that the image $\kappa \circ \chi(G)$ of G is a marked tree-code. □

If one does not distinguish between nonterminals and terminals, this theorem shows that a TC grammar is a strict interpretation of a marked tree-code, in the sense of grammar form theory (see [16]).

As a consequence of Theorem 3.2.8, the TC property is decidable.

Theorem 3.2.9 *It is decidable whether or not a cf grammar is terminally coded.*

Proof. Let $G = (\Sigma, \Delta, P, S)$ be a cf grammar. Check for each projection χ from Σ to Σ whether $\chi(G)$ is backwards deterministic, forwards deterministic, and bracketed.

By Theorem 3.2.8, G is terminally coded if and only if these three conditions are satisfied for one of the projections. Since there are finitely many such projections χ, and since these conditions can be checked in a finite number of steps, it is decidable whether G is terminally coded. □

By their special form, backwards deterministic parenthesis grammars can be parsed deterministically in a "trivial" way. In [13] (and [14]) it is shown that each parenthesis grammar has an equivalent parenthesis grammar that is backwards deterministic, where it should be noted

that in parenthesis grammars more than one initial nonterminal is allowed. Consequently, parenthesis languages can easily be recognized.

Using the result for parenthesis grammars, we will show that a similar result holds for terminally coded grammars. More precisely, not each TC grammar has an equivalent backwards deterministic TC grammar (as will be shown in Theorem 3.4.6), but it does have an equivalent TC grammar that is backwards deterministic and has more than one initial nonterminal (Theorem 3.2.13).

We use the abbreviation BD-TC grammar for a backwards deterministic TC grammar. A grammar is a *cf grammar in the wider sense*, abbreviated *cf-w grammar*, if it is like a cf grammar $G = (\Sigma, \Delta, P, S)$, but instead of one initial nonterminal it has a finite set S of initial nonterminals. The language generated by G is $L(G) = \{w \in \Delta^* \mid Z \Rightarrow^* w \text{ for some } Z \in S\}$. Most definitions concerning cf grammars carry over to cf-w grammars in a direct way.

Definition 3.2.10
(1) A cf-w grammar $G = (\Sigma, \Delta, P, S)$ is a *parenthesis* (PAR) *grammar* if $\{(,)\} \subseteq \Delta$, and each production is of the form $A \rightarrow (\alpha)$, where $\alpha \in (\Sigma - \{(,)\})^*$.
(2) A cf language K is a *parenthesis* (PAR) *language* if there is a parenthesis grammar G such that $K = L(G)$.
(3) Let $G = (\Sigma, \Delta, P, S)$ be a cf-w grammar such that $\Sigma \cap \{(,)\} = \emptyset$. The *parenthesized version of* G, denoted (G), is the parenthesis grammar $(\Sigma \cup \{(,)\}, \Delta \cup \{(,)\}, P', S)$ where $P' = \{A \rightarrow (\alpha) \mid A \rightarrow \alpha \in P\}$.
(4) Two cf-w grammars are *structurally equivalent* if their parenthesized versions are equivalent.

Of course we might use other symbols than (and)to denote the "parentheses" of a parenthesis grammar, provided they have the same role in the grammar.

A well-known equivalent formulation of structural equivalence is as follows : cf-w grammars G_1 and G_2 are structurally equivalent if for each successful derivation tree in G_1 there is a successful derivation tree in G_2 with the same yield and the same underlying tree, and vice versa. The next lemma is also well-known.

Lemma 3.2.11 *Each cf grammar has a structurally equivalent cf-w grammar that is backwards deterministic.*

Proof. Let $G_1 = (\Sigma, \Delta, P, S)$ be a cf grammar, and let (G_1)be its parenthesized version.

By [13, Theorem 1], there is a parenthesis grammar $\widehat{G}$ that is equivalent with $(G_1)$and backwards deterministic. Let G_2 be the cf-w grammar such that $(G_2) = \widehat{G}$. It is easily seen that G_2 is backwards deterministic. Furthermore, since $(G_1)$and $(G_2)$are equivalent, G_1 and G_2 are structurally equivalent. □

Now we get to the result for TC grammars. The definition of a terminally coded cf-w grammar is as Definition 3.2.1(1,2) with the interpretation that a derivation tree is a derivation subtree from one of the initial symbols. Such a grammar will also be called a TC-W grammar, and if it is backwards deterministic, a BD-TC-W grammar. A language generated by a (BD-)TC(-W) grammar is called a (BD-)TC(-W) language.

Lemma 3.2.12 *Let G_1 and G_2 be structurally equivalent cf-w grammars. Then G_1 is terminally coded iff G_2 is terminally coded.*

Proof. Suppose that one of the grammars is terminally coded, say G_1. Let t be a successful derivation tree of a word w in G_2. There is a derivation tree t' of w in G_1 such that $und(t) = und(t')$, because G_1 and G_2 are structurally equivalent (see the observation following Definition 3.2.10). Hence the projection χ that is used for G_1 in Definition 3.2.1(1) is also appropriate for G_2. Consequently G_2 is TC. □

Theorem 3.2.13 *For each* TC *grammar there is an equivalent* TC-W *grammar that is backwards deterministic.*

Proof. Let G be a TC grammar. By Lemma 3.2.11, there is a backwards deterministic cf-w grammar G' that is structurally equivalent with G. Then, by Lemma 3.2.12, G' is a TC-W grammar. □

Remark 3.2.14 Allowing TC grammars to be in the wider sense does not affect the class of languages that they generate, as we will show now.

Let $G = (\Sigma, \Delta, P, S)$ be a TC-W grammar. Let $G' = (\Sigma \cup \{S'\}, \Delta, P', S'\}$ be the cf grammar such that $S' \notin \Sigma$ and $P' = P \cup \{S' \to \alpha \mid Z \to \alpha \in P \text{ for some } Z \in S\}$. It is easily seen that G' is equivalent with G.

Note that for grammars in the wider sense, a result analogous with Theorem 3.2.4 holds, i.e., a cf-w grammar $G = (\Sigma, \Delta, P, S)$ is terminally coded by a marked tree-code φ iff there is a projection $\chi : \Sigma \to alph(\varphi)$ such that Condition (ii) of Theorem 3.2.4 is satisfied, and $\chi(Z) = one_\varphi$ for each $Z \in S$. Now, if this projection χ for G is extended to $\Sigma \cup \{S'\}$ by defining $\chi(S') = one_\varphi$, then clearly χ satisfies the conditions of Theorem 3.2.4 w.r.t. G'. Hence G' is a TC grammar equivalent with G. Consequently, each TC-W grammar has an equivalent TC grammar, and the class of TC-W languages equals the class of TC languages. □

Remark 3.2.15 Parenthesis grammars are TC-W grammars. Let $G = (\Sigma \cup \{(,)\}, \Delta \cup \{(,)\}, P, S)$ be a parenthesis grammar, and let $\chi : \Sigma \to \{m,(,)\}$ be as follows : $\chi(A) = m$ for each $A \in \Sigma$, $\chi(() = (,$ and $\chi()) =)$. Then χ satisfies the conditions of Theorem 3.2.4 w.r.t. G and φ_{par2} from Example 3.1.3(2), and hence G is terminally coded by φ_{par2}. The bracketed grammars in the sense of [7] are BD-TC grammars. In fact they are also terminally coded by φ_{par2}. Another kind of cf grammars which are terminally coded by φ_{par2} are the cf grammars obtained by interpreting regular tree grammars in normal form (see, e.g., [8]) as string grammars. □

Not all cf languages are TC; it will be shown (in Theorem 3.4.6) that there are even finite languages that are not TC, and that all TC languages are deterministic cf languages. Here we will show that each cf language is a projection of a TC language. It is instructive to compare this with the situation for parenthesis languages. Not each cf language is a projection of a PAR language, e.g., the finite language $\{ab, ac\}$ can not be obtained from a PAR language by a projection, since b and c must come from different symbols of the original language. However, each cf language is the homomorphic image of a parenthesis language; for each cf grammar G,

$L(G)$ is obtained from the parenthesis language $L((G))$ by the homomorphism that simply erases the parentheses. The same is true for bracketed languages in the sense of [7].

We now prove that each cf language is a projection of a TC language, and even of a BD-TC language.

Theorem 3.2.16 *Every cf language is a projection of a backwards deterministic* TC *language.*

Proof. Let K be a cf language, and let $G = (\Sigma, \Delta, P, S)$ be a cf grammar in Greibach normal form such that $L(G) = K$. First we show that $L(G)$ is the projection of a backwards deterministic cf language.

Let $G' = (\Sigma', \Delta', P', S)$ be as follows :
$\Delta' = \{a_p \mid a \in \Delta, p \in P\}$,
$\Sigma' = \Delta' \cup (\Sigma - \Delta)$,
$P' = \{A \to a_p\alpha \mid p = A \to a\alpha \in P\}$.
Then obviously G' is a backwards deterministic cf grammar. Let $\mu : \Delta' \to \Delta$ be the projection such that for each $a_p \in \Delta'$, $\mu(a_p) = a$. Then $\mu(L(G')) = L(G)$.

Next we show that $L(G')$ is the projection of a BD-TC language. Let $\varphi = \varphi_2$ be the marked tree-code from Example 3.1.14. Since G' is chain-free, we have that $und(t) \in dom(\varphi)$ for every derivation tree t in G'.

We index the cf grammar G' with φ, which results in the cf grammar $G'' = (\Sigma'', \Delta'', P'', S'')$ defined by
$\Delta'' = \Delta' \times alph(\varphi)$,
$\Sigma'' = \Sigma' \times alph(\varphi)$,
$S'' = (S, one_\varphi)$,
$P'' = \{(A, c) \to (A_1, c_1) \ldots (A_n, c_n) \mid n \geq 1, A \to A_1 \ldots A_n \in P',$
$c \to c_1 \ldots c_n$ a production of $\varphi\}$.
The cf grammar G'' is still backwards deterministic, since φ itself is also backwards deterministic. From Theorem 3.2.4 it follows by taking $\chi = proj_2 : \Sigma' \times alph(\varphi) \to alph(\varphi)$ that G'' is terminally coded by φ. Hence $L(G'')$ is a backwards deterministic TC language, and we have for the projection $proj_1 : \Delta' \times alph(\varphi) \to \Delta'$ that $proj_1(L(G'')) = L(G')$.

Now the composition of μ and $proj_1$ is a projection of $L(G'')$ onto $L(G)$.

Hence $K = L(G)$ is a projection of a backwards deterministic TC language. □

Instead of Definition 3.2.1(1) one might consider the following alternative, more general, definition of a terminally coded grammar : for a cf grammar $G = (\Sigma, \Delta, P, S)$, and a marked tree-code φ, G is terminally coded by φ if there is a projection $\chi : \Delta \to alph(\varphi)$ such that for each $w \in L(G)$ *there exists* a derivation tree t of w in G such that $und(t) \in dom(\varphi)$ and $\varphi(und(t)) = \chi(w)$. We will call a grammar that is terminally coded in the above sense a $TC_\exists$ grammar. Clearly, each TC grammar is a $TC_\exists$ grammar. Note that for $TC_\exists$ grammars we no longer have the characterization of Theorem 3.2.4. Hence it is questionable whether the $TC_\exists$ property is decidable (cf. Theorem 3.2.9). We conclude this section by showing that $TC_\exists$ grammars generate the same class of languages as TC grammars.

Theorem 3.2.17 *Each* $TC_\exists$ *grammar is equivalent to a* TC *grammar.*

Proof. Let $G = (\Sigma, \Delta, P, S)$ be a TC$_\exists$ grammar coded by the marked tree-code φ, and let χ be the appropriate projection. Define the cf grammar $G' = (\Sigma', \Delta, P', S')$ by

$\Sigma' = \Delta \cup ((\Sigma - \Delta) \times alph(\varphi))$,

$S' = (S, one_\varphi)$, and

$P' = \{A'_0 \to A'_1 \dots A'_n \mid n \geq 1, A_0 \to A_1 \dots A_n \in P,$ $c_0 \to c_1 \dots c_n$ a production of φ, $A'_i = A_i$ and $\chi(A_i) = c_i$ if $A_i \in \Delta$, and $A'_i = (A_i, c_i)$ if $A_i \in \Sigma - \Delta\}$.

Clearly, if $w \in L(G')$, then $w \in L(G)$. Now let $w \in L(G)$, and let (t, η) be a derivation tree of w in G such that $\varphi(t) = \chi(w)$. Hence $t[\varphi]$ is defined. Let (t, η') be the labeled tree such that for each $v \in in(t)$, $\eta'(v) = (\eta(v), lb_{t[\varphi]}(v))$, and for each $v \in leaf(t)$, $\eta'(v) = \eta(v)$. Since $\chi(yield(t, \eta)) = \chi(w) = \varphi(t) = yield(t[\varphi])$, we have for each $v \in leaf(t)$ that $\chi(\eta(v)) = lb_{t[\varphi]}(v)$. It follows that $prod(t, \eta')$ consists of productions of G'. Consequently (t, η') is a derivation tree of w in G', and so $w \in L(G')$. Hence G and G' are equivalent.

Clearly, the mapping χ' defined on Σ' by $\chi' \mid_\Delta = \chi$ and $\chi' \mid_{\Sigma' - \Delta} = proj_2$ satisfies the conditions of Theorem 3.2.4 w.r.t. G' and φ. Thus, G' is a TC grammar. □

3.3 VSP grammars

As one might expect, backwards deterministic terminally coded grammars can easily be parsed. In this section we provide a formal proof of this fact. We will give a shift-reduce parsing algorithm without look-ahead, which works in particular for BD-TC-W grammars. Then, by Theorem 3.2.13, it follows that TC languages can be recognized in linear time.

First we recall some aspects of shift-reduce parsing for precedence grammars (see, e.g., [1, Section 5.3]).

A shift-reduce parsing algorithm uses a left-to-right input scan and a pushdown list. It finds a rightmost parse in the following way : it shifts letters of the input to the pushdown list until the handle is found; then it applies the appropriate reduction. For that purpose the algorithm uses a *shift-reduce function*, which decides whether to shift the first letter of the input list to the pushdown list, or to make a reduction. For precedence grammars, the right and left end of the handle are detected by consulting (the table of) precedence relations that hold between symbols of the grammar.

One of the classes of grammars that can be parsed in this way is the class of simple precedence grammars (see [1, Section 5.3.2]).

Definition 3.3.1

(1) The *Wirth-Weber precedence relations* $\lessdot$, $\doteq$, and $\gtrdot$ for a cf grammar $G = (\Sigma, \Delta, P, S)$ are defined as follows. For $X, Y \in \Sigma$,

(i) $X \lessdot Y$ if there exists $A \to \alpha X B \beta$ in P such that $B \Rightarrow^+ Y\gamma$,

(ii) $X \doteq Y$ if there exists $A \to \alpha X Y \beta$ in P, and

(iii) for $a \in \Delta$, $X \gtrdot a$ if $A \to \alpha B Y \beta$ is in P, $B \Rightarrow \gamma X$, and $Y \Rightarrow a\delta$.

(2) A cf grammar $G = (\Sigma, \Delta, P, S)$ is a *simple precedence* (SP) *grammar* if G is backwards deterministic, there is no derivation $A \Rightarrow^+ A$ with $A \in \Sigma - \Delta$ in G, and the Wirth-Weber relations are disjoint.

(3) A cf language K is a *simple precedence* (SP) *language* if there is a simple precedence grammar G such that $K = L(G)$.

The shift-reduce function for a simple precedence grammar compares the letter on top of the pushdown list to the first letter of the remaining input. If the relation $\gtrdot$ holds, which means that the right end of the handle is found, then a reduction is called for, otherwise a shift is made. In case of a reduction, the letters on the pushdown list are compared until the relation $\lessdot$ holds, which gives the left end of the handle. Hence the algorithm examines the handle and the first letter beyond the handle on the pushdown list. Note that by the backwards determinism of the grammar, there is only one way to reduce the handle.

We will introduce a new class of grammars, called "very simple precedence grammars", containing all backwards deterministic TC grammars. These grammars are called very simple precedence grammars, since they are a special and natural kind of simple precedence grammars. Let us explain this. One way to generalize SP grammars is to define the Wirth-Weber relations on words and to demand that these relations are disjoint. The so obtained grammars are called extended precedence grammars (see also [1, Section 5.3.3]). The shift-reduce function for extended precedence grammars decides on the basis of a prefix of the input, rather than on the basis of only the first letter, whether to shift or to reduce (i.e., whether the right end of the handle is on top of the pushdown list); when a reduction is called for, it considers a (fixed) number of letters, instead of only one, beyond the handle on the pushdown list in order to locate the left end of the handle.

For the grammars that we are going to introduce now, there is a shift-reduce algorithm which detects the right end of the handle without examining any letters of the remaining input, and the left end without looking beyond the handle on the pushdown list (see Definition 3.3.9). The fact that the algorithm does not examine any letters of the input means that it does not use look-ahead.

As usual in precedence parsing, we will use the symbol \$ as a begin and endmarker. In particular, for a cf grammar $G = (\Sigma, \Delta, P, S)$, we enlarge the Wirth-Weber precedence relations by adding that $\$ \lessdot A$ for each $A \in \Sigma$ such that $S \Rightarrow^+ Aw$ for some $w \in \Sigma^*$, and that $A \gtrdot \$$ for each $A \in \Sigma$ such that $S \Rightarrow^+ wA$ for some $w \in \Sigma^*$. From now on in this paper, when we speak of the Wirth-Weber relations of a cf grammar, we mean the Wirth-Weber precedence relations including the above additions.

Definition 3.3.2
(1) a cf grammar $G = (\Sigma, \Delta, P, S)$ is a *very simple precedence* (VSP) *grammar* if G is backwards deterministic, there is no derivation $A \Rightarrow^+ A$ with $A \in \Sigma - \Delta$, and $dom(\gtrdot) \cap dom(\doteq) = \emptyset$, $ran(\lessdot) \cap ran(\doteq) = \emptyset$ for the Wirth-Weber precedence relations of G.
(2) a cf language K is a *very simple precedence* (VSP) *language* if there is a VSP grammar G such that $K = L(G)$.

First note that $dom(\lessdot) \subseteq dom(\doteq)$. One may think of a shift-reduce algorithm for VSP grammars as follows : it shifts while encountering symbols in $dom(\doteq)$; when the first element in $dom(\gtrdot)$ is found, it knows that this is the last letter of the handle because $dom(\gtrdot) \cap dom(\doteq) = \emptyset$; it then searches the pushdown list for the first element in $ran(\lessdot)$, which is the first letter

of the handle because $ran(\lessdot) \cap ran(\doteq) = \emptyset$. Because of the marker \$ it always finds an input letter in $dom(\gtrdot)$ and a pushdown element in $ran(\lessdot)$.

We will show now that indeed VSP grammars form a subclass of SP grammars.

Theorem 3.3.3 *Each* VSP *grammar is an* SP *grammar.*

Proof. Let $G = (\Sigma, \Delta, P, S)$ be a VSP grammar. Since $dom(\gtrdot) \cap dom(\doteq) = \emptyset$, and $ran(\lessdot) \cap ran(\doteq) = \emptyset$, it follows that $\gtrdot$ and $\doteq$ are disjoint, and that $\lessdot$ and $\doteq$ are disjoint. As noticed before, $dom(\lessdot) \subseteq dom(\doteq)$. Hence also $\lessdot$ and $\gtrdot$ are disjoint. Hence G is a simple precedence grammar. □

Imposing the conditions of Definition 3.3.2(1) on the Wirth-Weber relations of G is equivalent to requiring G to be bracketed (Definition 3.1.9), as shown in the next lemma.

Lemma 3.3.4 *A cf grammar G is a* VSP *grammar iff G is bracketed and backwards deterministic.*

Proof. The crucial observation is that $dom(\gtrdot) = R_G$ and $ran(\lessdot) = L_G$ for every grammar G (due to the fact that G is reduced and that $\lessdot$ and $\gtrdot$ are extended by means of the marker \$).

Suppose first that G is VSP. Let $A \to A_1 A_2 \dots A_n$ be a production of G. Then $A_j \in ran(\doteq)$ for $j = 2, \dots, n$, and $A_j \in dom(\doteq)$ for $j = 1, \dots, n-1$. Since $ran(\lessdot) = L_G$, and $ran(\lessdot) \cap ran(\doteq) = \emptyset$, it follows that $A_2 \dots A_n \in (\Sigma - L_G)^*$, and, similarly, since $dom(\gtrdot) = R_G$, and $dom(\gtrdot) \cap dom(\doteq) = \emptyset$, it follows that $A_1 \dots A_{n-1} \in (\Sigma - R_G)^*$. Hence $A_1 \dots A_n \in L_G(\Sigma - L_G)^* \cap (\Sigma - R_G)^* R_G$. By definition, there is no derivation $S \Rightarrow^+ S$, and G is backwards deterministic. Consequently, G is bracketed and backwards deterministic.

Suppose now that G is a bracketed bd-cf grammar. Let X be an element of $dom(\doteq)$. Hence there is a production $A \to \alpha X Y \beta$ in G with $\alpha, \beta \in \Sigma^*$, $Y \in \Sigma$. By Condition (i) of Definition 3.1.9, $X \notin R_G$. Since $R_G = dom(\gtrdot)$, it follows that $dom(\gtrdot) \cap dom(\doteq) = \emptyset$. Similarly, for each $Y \in ran(\doteq)$, we obtain that $Y \notin L_G$, and hence $ran(\lessdot) \cap ran(\doteq) = \emptyset$. Finally, since G is unambiguous by Proposition 3.1.11, there is no derivation $A \Rightarrow^+ A$ with $A \in \Sigma - \Delta$ in G. Consequently, G is VSP. □

In what follows we will use the above characterization of VSP grammars rather than Definition 3.3.2(1). Note that, by Lemma 3.3.4, VSP grammars have the property of Lemma 3.1.10. Hence, by Proposition 3.1.11, VSP grammars are unambiguous. From Lemma 3.1.10 we obtain the following property of VSP languages.

Lemma 3.3.5 *For each* VSP *grammar G, if $xuy \in L(G)$, $u \in L(G)$, and $w \in L(G)$, then $xwy \in L(G)$.*

Proof. Let S be the initial nonterminal of G. Let t be a derivation tree of xuy, let t' be a derivation tree of u, and let t'' be a derivation tree of w. Since there is no derivation $S \Rightarrow^+ S$, t and t' satisfy the conditions of Lemma 3.1.10. Thus, by Lemma 3.1.10, there is a $v \in in(t)$ such that $t \backslash v$ is a derivation tree of xSy. By adding t'' to the leaf v in $t \backslash v$, a derivation tree of xwy is obtained. Hence $xwy \in L(G)$. □

Because of the results in Section 3.2, we will also consider "VSP grammars in the wider sense". A cf-w grammar $G = (\Sigma, \Delta, P, S)$ is a very simple precedence grammar if it is backwards deterministic, there is no derivation $A \Rightarrow^+ A$ with $A \in \Sigma - \Delta$, there is no derivation $Z \Rightarrow^+ Z'$ for $Z, Z' \in S$, and $dom(\gtrdot) \cap dom(\doteq) = \emptyset$, $ran(\lessdot) \cap ran(\doteq) = \emptyset$ for the Wirth-Weber precedence relations of G (which include $\$ \lessdot A$ if $Z \Rightarrow^+ Aw$, and $A \gtrdot \$$ if $Z \Rightarrow^+ wA$, where $Z \in S$). Such a grammar is also called a VSP-W grammar, and a language generated by a VSP-W grammar is called a VSP-W language. Analogously to Lemma 3.3.4, a cf-w grammar $G = (\Sigma, \Delta, P, S)$ is VSP-W iff it is backwards deterministic and bracketed, where bracketed for cf grammars in the wider sense means that Condition (i) of Definition 3.1.9 is satisfied, and that there is no derivation $Z \Rightarrow^+ Z'$ for $Z, Z' \in S$. It is easy to see that Lemma 3.1.10 and Proposition 3.1.11 also hold for VSP-W grammars.

We show now that the class of VSP grammars indeed contains the class of BD-TC grammars.

Theorem 3.3.6 *Each* BD-TC *grammar is a* VSP *grammar. Also, each* BD-TC-W *grammar is a* VSP-W *grammar.*

Proof. Let $G = (\Sigma, \Delta, P, S)$ be a backwards deterministic TC grammar. Let χ be the projection from Theorem 3.2.4.

Since φ is bracketed, for each $A \to \alpha \in P$, $\chi(\alpha) \in L_\varphi(\Sigma - L_\varphi)^* \cap (\Sigma - R_\varphi)^* R_\varphi$. By Theorem 3.2.4, for each $B \in \Sigma$, $\chi(B) \notin L_\varphi$ implies that $B \notin L_G$, and $\chi(B) \notin R_\varphi$ implies that $B \notin R_G$. Thus, for each $A \to \alpha \in P$, $\alpha \in L_G(\Sigma - L_G)^* \cap (\Sigma - R_G)^* R_G$.

By Theorem 3.2.4, $\chi(S) = one_\varphi$. If there is a derivation $S \Rightarrow^+ S$ in G, then Theorem 3.2.4 implies that there is a derivation $\chi(S) \Rightarrow^+ \chi(S)$ in φ, which contradicts the fact that φ is bracketed.

Hence G is bracketed. Consequently, G is a VSP grammar.

Moreover, if G is in the wider sense, then, as observed in Remark 3.2.14, there is a projection $\chi : \Sigma \to alph(\varphi)$ such that Condition (ii) of Theorem 3.2.4 is satified, and $\chi(Z) = one_\varphi$ for each $Z \in S$. It follows that each BD-TC-W grammar is a VSP-W grammar. □

Not each VSP grammar is a BD-TC grammar, as the following example shows.

Example 3.3.7 Let G be the cf grammar with nonterminals S, A, B, where S is the initial nonterminal, terminals a, b, c, d, and productions $S \to ac$, $S \to bc$, $S \to AB$, $A \to ad$, and $B \to bd$. Clearly, G is bracketed and backwards deterministic, and hence VSP. Suppose that G is a BD-TC grammar. By Theorem 3.2.8, there is a projection χ such that $\chi(G)$ is a marked tree-code. Since $\chi(G)$ is forwards deterministic, it follows that $\chi(a) = \chi(b)$. Hence $\chi(ad) = \chi(bd)$, which implies that $\chi(A) = \chi(B)$, because $\chi(G)$ is backwards deterministic. But, since $\chi(G)$ is bracketed, $\chi(A) \in L_{\chi(G)} - R_{\chi(G)}$ and $\chi(B) \in R_{\chi(G)} - L_{\chi(G)}$, a contradiction. Hence G is not a BD-TC grammar. □

From Theorems 3.3.3 and 3.3.6, and Remark 3.2.15 we obtain an inclusion diagram for classes of grammars as shown in Figure 3.8, where a straight line between two classes denotes that the lower class is included in the upper class.

Now we present the description of an actual *shift-reduce algorithm without look-ahead for* VSP-W *grammars*. First we give an elementary property of VSP-W grammars.

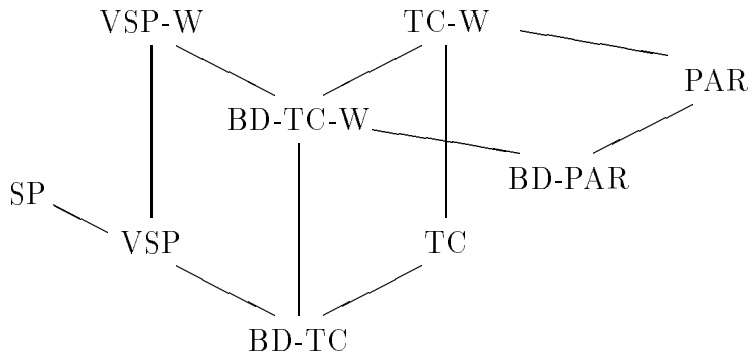


Figure 3.8: a hierarchy of grammar classes

Lemma 3.3.8 *Let $G = (\Sigma, \Delta, P, S)$ be a* VSP-W *grammar. If $Z \stackrel{r}{\Rightarrow}{}^* \alpha A w$ in G, where $Z \in S$, $\alpha \in \Sigma^*$, $A \in \Sigma - \Delta$, and $w \in \Delta^*$, then $\alpha \in (\Sigma - R_G)^*$.*

Proof. By induction on the length ℓ of the rightmost derivation $Z \stackrel{r}{\Rightarrow}{}^* \alpha A w$.

For $\ell = 0$ the claim holds.

Suppose now that for some $\ell \geq 0$ the claim holds for all rightmost derivations of length $\leq \ell$. Let $\alpha A w$ be a rightmost sentential form such that the length of its rightmost derivation is $\ell + 1$. Let $Z \in S$, $\alpha' \in \Sigma^*$, $A' \in \Sigma - \Delta$, and $w' \in \Delta^*$ be such that $Z \stackrel{r}{\Rightarrow}{}^* \alpha' A' w' \stackrel{r}{\Rightarrow} \alpha' \beta' w' = \alpha A w$. Recall that, by Lemma 3.3.4, G is bracketed.

By the inductive assumption $\alpha' \in (\Sigma - R_G)^*$. Hence, since G is bracketed, $last(\beta')$ is the first letter in $\alpha' \beta' w'$ that is in R_G. From $|w'| \leq |w|$ we obtain that $|\alpha| < |\alpha' \beta'|$. Consequently, $\alpha \in (\Sigma - R_G)^*$, which completes the induction proof. □

Definition 3.3.9 Let $G = (\Sigma, \Delta, P, S)$ be a VSP-W grammar (where $\$ \notin \Sigma$).
(1) The *shift-reduce function of* G, $f : \Sigma \cup \{\$\} \rightarrow \{\mathbf{shift}, \mathbf{reduce}\}$ is defined as follows : for $X \in \Sigma \cup \{\$\}$,
$f(X) = \mathbf{shift}$ if $X \notin R_G$,
$f(X) = \mathbf{reduce}$ if $X \in R_G$.
(2) A *parsing configuration of* G is a triple (α, w, π), with $\alpha \in \$\Sigma^*$, $w \in \Delta^*$, and $\pi \in P^*$.
(3) The *step relation of* G, denoted by $\vdash$, is a relation on parsing configurations of G defined as follows. For $X \in \Sigma \cup \{\$\}$, $a \in \Delta$,
(i) $(\alpha X, aw, \pi) \vdash (\alpha X a, w, \pi)$ if $f(X) = \mathbf{shift}$,
(ii) $(\alpha \beta X, w, \pi) \vdash (\alpha B, w, \pi p)$ if $f(X) = \mathbf{reduce}$, and $B \rightarrow \beta X$ is production p.
As usual, $\vdash^+$ is the transitive closure of $\vdash$, and $\vdash^*$ is the reflexive and transitive closure of $\vdash$.
(4) The *parsing translation of* G, denoted by $T(G)$, is $\{(w, \pi) \in \Delta^* \times P^* \mid (\$, w, \varepsilon) \vdash^* (\$Z, \varepsilon, \pi)$ for some $Z \in S\}$.

Intuitively, the algorithm (as described by the step relation $\vdash$) finds a rightmost parse of the input word w, using a pushdown list.

The symbol \$ is used as a bottom-marker of the pushdown list. It is needed to produce the first shift.

For a configuration $(\$\alpha, w, \pi)$, $\$\alpha$ is the word on the pushdown list, w is the remaining input, and π the string of those productions that give a rightmost parse of the original input into αw.

The algorithm does not use look-ahead because it bases its choice of shifting or reducing on the contents of the pushdown list only. The algorithm is deterministic because the relation $\vdash$ is a partial function : f is a function, and in case 3(ii) of Definition 3.3.9, only one right-hand side can be found, since $\beta X \in L_G(\Sigma - L_G)^*$. Because of this and because there is no derivation $Z \Rightarrow^+ Z'$ for $Z, Z' \in S$, $T(G)$ is also a partial function.

The following theorem proves the correctness of the algorithm.

Theorem 3.3.10 *For every $w \in \Delta^*$ and every $\pi \in P^*$, the pair (w, π) is in $T(G)$ iff π is a rightmost parse of w.*

Proof. Let $w \in \Delta^*$. The proof relies on the following two claims.

Claim 3.3.11 *For each parsing configuration $(\$\alpha, w', \pi')$ such that $(\$, w, \varepsilon) \vdash^* (\$\alpha, w', \pi')$, π' is a rightmost parse of w into $\alpha w'$.*

Proof. This is easily seen by induction on the length of the computation $(\$, w, \varepsilon) \vdash^* (\$\alpha, w', \pi')$, as follows.

For the empty computation there is nothing to prove.

If the algorithm has made a shift in the last step of the computation, i.e., $(\$\alpha', aw', \pi') \vdash (\$\alpha' a, w', \pi') = (\$\alpha, w', \pi')$, then, by the inductive assumption, π' is a rightmost parse of w into $\alpha' a w' = \alpha w'$.

If the algorithm has made a reduction in the last step, i.e., $(\$\alpha'\beta, w', \pi'') \vdash (\$\alpha' B, w', \pi'' p) = (\$\alpha, w', \pi')$, where $p = B \to \beta$, then, by the inductive assumption, π'' is a rightmost parse of w into $\alpha'\beta w'$, and hence $\pi' = \pi'' p$ is a rightmost parse of w into $\alpha' B w' = \alpha w'$. □

Claim 3.3.12 *For each rightmost parse π' of w into a rightmost sentential form $\alpha w'$, with $\alpha \in \{\varepsilon\} \cup \Sigma^*(\Sigma - \Delta)$ and $w' \in \Delta^*$, $(\$, w, \varepsilon) \vdash^* (\$\alpha, w', \pi')$.*

Proof. We use induction on the length of π'.

The claim trivially holds for $\pi' = \varepsilon$.

Let $p = B \to \beta$ be the production $last(\pi')$, i.e., $\alpha w' = \alpha' B w' \overset{T}{\Rightarrow} \alpha'\beta w'$ and $\pi' = \pi'' p$, where π'' is a rightmost parse of w into the rightmost sentential form $\alpha'' w'' = \alpha'\beta w'$. By the inductive assumption, $(\$, w, \varepsilon) \vdash^* (\$\alpha'', w'', \pi'')$.

Clearly, α'' is a prefix of $\alpha'\beta$. By Lemma 3.3.8, $\alpha'\beta \in (\Sigma - R_G)^* R_G$, and hence the algorithm first shifts all possibly remaining letters of $\alpha'\beta$, i.e., $(\$, w, \varepsilon) \vdash^* (\$\alpha'', w'', \pi'') \vdash^* (\$\alpha'\beta, w', \pi'')$.

Since $last(\beta) \in R_G$, the algorithm then makes the reduction $(\$\alpha'\beta, w', \pi'') \vdash (\$\alpha' B, w', \pi'' p)$. Hence $(\$, w, \varepsilon) \vdash^* (\$\alpha, w', \pi')$. □

Now suppose that $(w, \pi) \in T(G)$ for $\pi \in P^*$. It follows by applying Claim 3.3.11 to the configuration $(\$Z, \varepsilon, \pi)$, where $Z \in S$, that π is a rightmost parse of w into Z.

On the other hand, if there is a rightmost parse π of w into some $Z \in S$, then by Claim 3.3.12, $(\$, w, \varepsilon) \vdash^* (\$Z, \varepsilon, \pi)$. Hence $(w, \pi) \in T(G)$. □

Remark 3.3.13 By Lemma 3.3.4, every forward deterministic VSP grammar is a marked tree-code. Not each marked tree-code is a VSP grammar, because marked tree-codes may be infinite, may have no terminals, and need not be reduced. However, we can still use the above shift-reduce algorithm to decode marked tree-codes. This can be done as follows.

Let $\varphi = (\Sigma, \emptyset, P, S)$ be a marked tree-code such that each $a \in \Sigma$ is reachable. Define the VSP grammar (possibly with infinitely many productions!) $G = (\Sigma', \Delta, P', S)$ such that $\Sigma' = \Sigma \cup \Delta$, $\Delta = \{c' \mid c \in \Sigma\}$, and $P' = \{c \to \beta \mid c \to \gamma \in P, \beta \in \sigma(\gamma)\}$, where $\sigma : \Sigma^* \to 2^{(\Sigma')^*}$ is the substitution given by $\sigma(c) = \{c, c'\}$ for each $c \in \Sigma$. Note that G is reduced.

Let $\mu : \Sigma \to \Delta$ be the projection such that $\mu(c) = c'$ for each $c \in \Sigma$. It is not difficult to see that for each tree t and for each $x \in \Sigma^*$, except $x = S$, $\varphi^{-1}(x) = t$ iff $(\mu(x), \pi) \in T(G)$ and the derivation tree corresponding with π has underlying tree t. □

Remark 3.3.14 As noticed in Section 3.1, bracketed bd-cf grammars are BC(0,0) grammars. Hence VSP grammars are BC(0,0), and consequently they are BRC(0,0) (i.e., of bounded *right* context (0,0)) as defined in [10] - this follows also directly from Lemma 3.3.8. Thus, according to the definitions in [10], VSP grammars are LR(0). □

It has now become evident that TC languages can easily be recognized.

Theorem 3.3.15 *Every* TC *language can be recognized in linear time.*

Proof. By Theorem 3.2.13 every TC language is generated by a BD-TC-W grammar. By Theorem 3.3.6, such a grammar can be parsed using the shift-reduce algorithm of Definition 3.3.9, and clearly, it works in linear time. □

We end this section by comparing VSP grammars with another class of known shift-reduce parsable grammars, the so-called *nonterminal separated* (*NTS*) grammars (see [2]). These are cf-w grammars for which the generated language remains unchanged if the productions are used in both ways (i.e., either in generative or in reducing fashion). This property implies that a shift-reduce parsing method very similar to the one in Definition 3.3.9 may be used for chain-free NTS grammars. However, NTS grammars need not be unambiguous; in fact, the shift-reduce algorithm finds at most one of the derivation trees of the input word.

Definition 3.3.16
(1) A cf-w grammar $G = (\Sigma, \Delta, P, S)$ is an *NTS grammar* if for all $A, B \in \Sigma - \Delta$ and all $\alpha, \beta, \gamma \in \Sigma^*$, if $A \Rightarrow^* \alpha\beta\gamma$ and $B \Rightarrow^* \beta$, then $A \Rightarrow^* \alpha B\gamma$.
(2) A cf language K is an *NTS language* if there is an NTS grammar G such that $K = L(G)$.

Theorem 3.3.17 *Each chain-free* VSP-W *grammar is an* NTS *grammar.*

Proof. Let $G = (\Sigma, \Delta, P, S)$ be a chain-free VSP-W grammar. Let $A, B \in \Sigma - \Delta$ and $\alpha, \beta, \gamma \in \Sigma^*$ be such that $A \Rightarrow^* \alpha\beta\gamma$ and $B \Rightarrow^* \beta$. Since G is chain-free, there is no derivation $B \Rightarrow^+ A$ in G, and hence the derivation subtrees corresponding with the above derivations satisfy the conditions of Lemma 3.1.10. Since Lemma 3.1.10 also holds for cf-w grammars, it follows that $A \Rightarrow^* \alpha B\gamma$. Consequently, G is an NTS grammar. □

3.4 Classes of languages

In this section we compare the classes of languages corresponding to the grammar types described in Sections 3.2 and 3.3 (see Figure 3.9). We repeat this for the case of chain-free grammars (see Figure 3.10). In the first case, we immediately obtain inclusion relations between the classes of languages from the inclusion diagram for classes of grammars given in Figure 3.8. However, some of these classes collapse.

In Figure 3.9 and in what follows, we denote by $\mathcal{L}(X)$ the class of X languages, where X stands for the name of any type of language, like TC, VSP, etc.

First we will show that $\mathcal{L}(\text{VSP-W}) \subseteq \mathcal{L}(\text{SP}) \cap \mathcal{L}(\text{NTS})$. For this purpose we need the next lemma.

Lemma 3.4.1 *Each* VSP *grammar is equivalent to a chain-free* VSP *grammar. The same holds for* VSP-W *grammars.*

Proof. We will show that each VSP-W grammar has an equivalent chain-free VSP-W grammar with the same set of initial nonterminals.

Let $G = (\Sigma, \Delta, P, S)$ be a VSP-W grammar. Note that (as in any bracketed grammar) for each $A \in L_G \cap R_G$ the only right-hand side in which A occurs is A itself. Moreover, since G is backwards deterministic, there is a unique B such that $B \to A$ is a production of G. Since there is no derivation $X \Rightarrow^+ X$ with $X \in \Sigma - \Delta$, it follows that for each $A \in \Sigma - \Delta$, there is a unique nonterminal $Y_A \notin L_G \cap R_G$ such that $Y_A \Rightarrow^* A$ (if $A \notin L_G \cap R_G$, then $Y_A = A$).

Let ψ be the substitution on Σ defined by $\psi(A) = \{Z \in S \mid Y_Z = A\} \cup \{A\}$, and let $\Sigma' = \Delta \cup S \cup \{A \in \Sigma - \Delta \mid A \notin L_G \cap R_G\}$. Consider the cf grammar $G' = (\Sigma', \Delta, P', S)$, where

$$P' = \{A \to \alpha' \mid \alpha' \in \psi(\alpha), \alpha \notin \Sigma - \Delta, A \in \Sigma' - \Delta, \text{ and for some } A_1, \ldots, A_n \in \Sigma - S, \\ n \geq 0, A = A_0 \Rightarrow A_1 \Rightarrow \ldots \Rightarrow A_n \Rightarrow \alpha \text{ in } G \quad \}.$$

From the above-mentioned fact that the nonterminals in $L_G \cap R_G$ occur only in chain-productions of G, we obtain that each right-hand side α of G such that $\alpha \notin \Sigma - \Delta$ is in $(\Sigma')^*$. Thus also $\psi(\alpha) \subseteq (\Sigma')^*$ for such right-hand sides α, and hence G' is well-defined.

First we will prove that the constructed G' is equivalent with G.

Claim 3.4.2
(1) *For all $A \in \Sigma'$ and all $w \in \Delta^*$, if $A \Rightarrow^* w$ in G, then there exists an $A' \in \psi(A)$ such that $A' \Rightarrow^* w$ in G',*
(2) *For all $A' \in \Sigma'$ and all $w \in \Delta^*$, if $A' \Rightarrow^* w$ in G', then $A' \Rightarrow^* w$ in G.*

Proof.
(1) We use induction on the length ℓ of a derivation $A \Rightarrow^* w$ in G. If $\ell = 0$, i.e., $A = w$, then $A \Rightarrow^* w$ in G'. Suppose that, for some $\ell \geq 0$, the claim holds for all $A \in \Sigma'$, $w \in \Delta^*$ such that $A \Rightarrow^* w$ in at most ℓ steps. Consider a derivation $A = A_0 \Rightarrow A_1 \Rightarrow \ldots \Rightarrow A_n \Rightarrow \alpha \Rightarrow^* w$ in G of length $\ell + 1$, where $n \geq 0$ is such that $A_j \in \Sigma - \Delta$ for $j = 0, \ldots, n$, and $\alpha \notin \Sigma - \Delta$. As observed before, since $\alpha \notin \Sigma - \Delta$, we have that $\alpha \in (\Sigma')^*$. Hence the induction hypothesis can be applied to the letters of α separately. As a result, there is an $\alpha' \in \psi(\alpha)$ such that $\alpha' \Rightarrow^* w$ in G'.

Note that at most one of the nonterminals $A_0, \ldots, A_n$ is in S, because there is no derivation $Z \Rightarrow^+ Z'$ for $Z, Z' \in S$. If $\{A_1, \ldots, A_n\} \cap S = \emptyset$, then $A \rightarrow \alpha'$ is a production of G', and hence $A \Rightarrow \alpha' \Rightarrow^* w$ in G'. If $k \in \{1, \ldots, n\}$ is such that $A_k \in S$, then $A_k \rightarrow \alpha'$ is a production of G', and $A_k \Rightarrow \alpha' \Rightarrow^* w$ in G'; moreover, since $A \notin L_G \cap R_G$, $A = Y_{A_k}$, which implies that $A_k \in \psi(A)$. This completes the inductive proof of (1).
(2) By induction on the length ℓ of a derivation $A' \Rightarrow^* w$ in G'. If $A' = w$, then $A' \Rightarrow^* w$ in G. Suppose that, for some $\ell \geq 0$, the claim holds for all $A' \in \Sigma'$, $w \in \Delta^*$ such that $A' \Rightarrow^* w$ in at most ℓ steps. Consider a derivation $A' \Rightarrow \alpha' \Rightarrow^* w$ in G' of length $\ell + 1$; hence in the first step the production $A' \rightarrow \alpha'$ is applied. By applying the induction hypothesis to each letter of α' we obtain that $\alpha' \Rightarrow^* w$ in G. The production $A' \rightarrow \alpha'$ comes from a derivation $A' \Rightarrow^* \alpha$ in G, where α is such that $\alpha' \in \psi(\alpha)$. Clearly, $\alpha \Rightarrow^* \alpha'$ in G. Hence there is a derivation $A' \Rightarrow^* \alpha \Rightarrow^* \alpha' \Rightarrow^* w$ in G. This completes the inductive proof of (2). □

Since there is no derivation $Z' \Rightarrow^+ Z$ for $Z, Z' \in S$, $\psi(Z) = \{Z\}$ for all $Z \in S$. Hence if $Z \Rightarrow^* w$ in G for some $Z \in S$, then, by Claim 3.4.2(1), $Z \Rightarrow^* w$ in G'. Conversely, if $Z \Rightarrow^* w$ in G' for some $Z \in S$, then by Claim 3.4.2(2), $Z \Rightarrow^* w$ in G. Thus, $w \in L(G)$ iff $w \in L(G')$ for every $w \in \Delta^*$. Consequently, G and G' are equivalent.

Clearly, G' is chain-free. The right-hand sides of G' are like right-hand sides of G, except that nonterminals of the type Y_Z for $Z \in S \cap L_G \cap R_G$ may be altered to Z. Since G' does not have productions of the form $A \rightarrow Z$, it follows that G' is still bracketed. Also, G' is backwards deterministic, which is seen as follows.

Let $A \rightarrow \alpha'$ and $B \rightarrow \alpha'$ be productions of G'. These productions come from derivations $A \Rightarrow^+ \alpha$ and $B \Rightarrow^+ \beta$ in G, where α and β are such that $\alpha' \in \psi(\alpha)$ and $\alpha' \in \psi(\beta)$. Note that if $Z \in S$ occurs in α', then the corresponding letter in α (as well as in β) is Z or Y_Z; if it is Z, then $Z \notin L_G \cap R_G$, which implies that $Z = Y_Z$. Hence for each A occurring in α' the corresponding letter in α (and in β) is A if $A \notin S$, and Y_A if $A \in S$. It follows that $\alpha = \beta$.

We will show now that $A = B$. Since derivations of the form $X \Rightarrow^+ X$ with $X \in \Sigma - \Delta$ are forbidden, there is either no derivation $B \Rightarrow^+ A$ or no derivation $A \Rightarrow^+ B$ in G. Assume without loss of generality that there is no derivation $B \Rightarrow^+$ A. Let $A = A_0 \Rightarrow A_1 \Rightarrow \ldots \Rightarrow A_n \Rightarrow \alpha$ be a derivation in G causing the production $A \rightarrow \alpha'$ of G'. By the definition of G', $\{A_1, \ldots, A_n\} \cap S = \emptyset$. Since $B \Rightarrow^+ \alpha$, and not $B \Rightarrow^+ A$, it follows by Lemma 3.1.10 that $B = A_j$ for some $j \in \{0, \ldots, n\}$. If $B \neq A$, then $B \notin S$ and $B \in L_G \cap R_G$, which contradicts the fact that $B \in \Sigma' - \Delta$. Consequently, $B = A$.

Hence G' is a chain-free VSP-W grammar equivalent with G, with the same set of initial nonterminals. □

By Theorem 3.3.17, each language generated by a chain-free VSP-W grammar is an NTS language. Hence, by Lemma 3.4.1, we have that $\mathcal{L}(\text{VSP-W}) \subseteq \mathcal{L}(\text{NTS})$.

From Theorem 3.3.3 we know that $\mathcal{L}(\text{VSP}) \subseteq \mathcal{L}(\text{SP})$. We show now that even $\mathcal{L}(\text{VSP-W}) \subseteq \mathcal{L}(\text{SP})$.

Lemma 3.4.3 *Each* VSP-W *language is an* SP *language.*

Proof. Let K be a VSP-W language, and let $G = (\Sigma, \Delta, P, S)$ be a VSP-W grammar such that $K = L(G)$. By Lemma 3.4.1 we may assume that G is chain-free. Let $\widehat{G} = (\Sigma \cup \{\widehat{S}\}, \Delta, \widehat{P}, \widehat{S})$ be the cf grammar such that $\widehat{P} = P \cup \{\widehat{S} \to Z \mid Z \in S\}$. Clearly, $\widehat{G}$ is equivalent with G. We will prove that $\widehat{G}$ is SP. The Wirth Weber precedence relations of G are disjoint (as in the proof of Theorem 3.3.3); the Wirth Weber relations of $\widehat{G}$ are obtained from those of G by adding $\$ \lessdot Z$ and $Z \gtrdot \$$ for all $Z \in S$, and hence they are disjoint (but note that $dom(\gtrdot) \cap dom(\doteq)$ and $ran(\lessdot) \cap ran(\doteq)$ may well become nonempty).

There is no production $X \to Z$ in G with $X \in \Sigma$ and $Z \in S$, because G is chain-free. From this and the fact that G is backwards deterministic it follows that $\widehat{G}$ is backwards deterministic. There is no derivation $A \Rightarrow^+ A$ for $A \in \Sigma - \Delta$ in G, and hence neither in $\widehat{G}$. Obviously, there is no derivation $\widehat{S} \Rightarrow^+ \widehat{S}$ in $\widehat{G}$. Hence $\widehat{G}$ is an SP grammar. □

By Theorem 3.3.6, $\mathcal{L}(\text{BD-TC}) \subseteq \mathcal{L}(\text{VSP})$. Although not each VSP grammar is a BD-TC grammar (see Example 3.3.7), the opposite inclusion also holds.

Lemma 3.4.4 *Each* VSP *grammar is equivalent to a* BD-TC *grammar. The same holds for grammars in the wider sense.*

Proof. Let $G = (\Sigma, \Delta, P, S)$ be a VSP grammar. By Lemma 3.4.1 we may assume that G is chain-free. We construct a VSP grammar equivalent with G that is "almost" forwards deterministic by reintroducing chain-productions. For each nonterminal A of G we do the following : if $A \to \alpha_1, \ldots, A \to \alpha_n \in P$ are all productions starting with A, then we add new nonterminals $A_2, \ldots, A_n$ to Σ, and replace the productions $A \to \alpha_1, \ldots, A \to \alpha_n$ by the productions $A \to \alpha_1, A \to A_2, A_2 \to \alpha_2, \ldots, A_{n-1} \to A_n, A_n \to \alpha_n$. In this way we obtain a grammar $G' = (\Sigma', \Delta, P', S)$ that is equivalent to G. It is easy to verify that G' is bracketed and backwards deterministic. For each $A \in \Sigma'$, and for each $n > 1$, there is at most one production $A \to \alpha$ with $|\alpha| = n$. Moreover, if there is more than one production of the form $A \to \alpha$ in P' with $|\alpha| = 1$, then there are precisely two such productions : one is $A \to a$ for some $a \in \Delta$ and the other $A \to B$ with B a 'new' nonterminal.

We claim that G' is a BD-TC grammar. We define a projection χ of Σ' as follows. Let $Y \in \Sigma'$. If $Y \notin \Delta \cap L_G \cap R_G$, then we define $\chi(Y) = Y$. If $Y \in \Delta \cap L_G \cap R_G$, then, since G' is backwards deterministic, there is precisely one $A \in \Sigma'$ such that $A \to Y \in P'$. Now if there is a production $A \to B$ with $B \in \Sigma' - \Delta$, then we define $\chi(Y) = B$, otherwise $\chi(Y) = Y$.

Then $\chi(G')$ is almost equal to G', except that productions of the form $A \to a$ with $a \in \Delta$ may be removed. Clearly, $\chi(G')$ is forwards deterministic, backwards deterministic, and bracketed. Hence $\chi(G')$ is a marked tree-code, and by Theorem 3.2.8, G' is TC. Since G'

is backwards deterministic, G' is a BD-TC grammar equivalent to G. Clearly, this proof can be easily extended to grammars in the wider sense. □

Example 3.4.5 For the grammar G from Example 3.3.7, the equivalent BD-TC grammar G' that one gets by the above construction is itself a marked tree-code, with productions $S \to ac$, $S \to S_2$, $S_2 \to bc$, $S_2 \to S_3$, $S_3 \to AB$, $A \to ad$, and $B \to bd$. □

Thus, by Lemma 3.4.4 and Theorem 3.3.6, $\mathcal{L}$(VSP) = $\mathcal{L}$(BD-TC), and $\mathcal{L}$(VSP-W) = $\mathcal{L}$(BD-TC-W). Furthermore, by Theorem 3.2.13 and Remark 3.2.14, $\mathcal{L}$(BD-TC-W) = $\mathcal{L}$(TC) = $\mathcal{L}$(TC-W). From Lemma 3.4.3, Theorem 3.3.17, and Lemma 3.4.1 it follows that $\mathcal{L}$(VSP-W) $\subseteq$ $\mathcal{L}$(SP) $\cap$ $\mathcal{L}$(NTS). By Remark 3.2.15, $\mathcal{L}$(PAR) $\subseteq$ $\mathcal{L}$(TC-W) and $\mathcal{L}$(BD-PAR) $\subseteq$ $\mathcal{L}$(BD-TC-W). By [13, Theorem 1], $\mathcal{L}$(BD-PAR) = $\mathcal{L}$(PAR).

These observations lead to the following result.

Theorem 3.4.6 *The inclusions given in Figure 3.9 hold (where a straight line between two classes denotes that the lower class is strictly included in the upper class). The classes* $\mathcal{L}$(VSP) *and* $\mathcal{L}$(PAR) *are incomparable.*

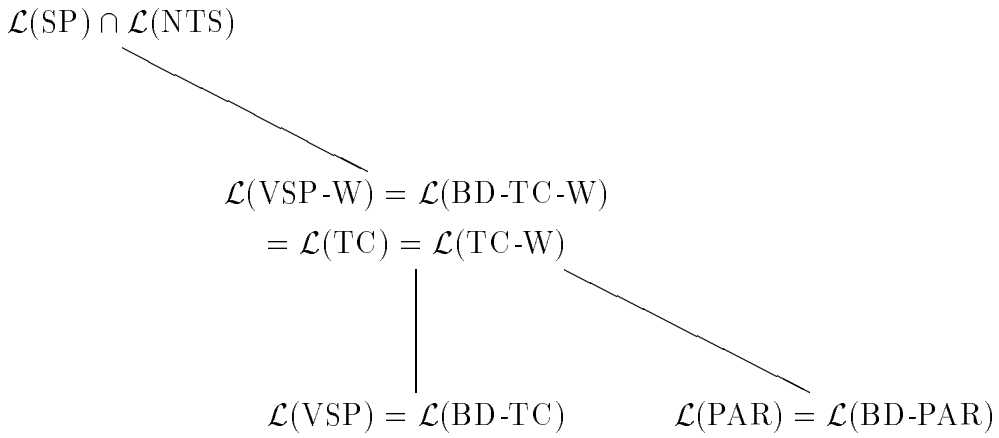


Figure 3.9: a hierarchy of language classes

Proof. It remains to be proved that $\mathcal{L}$(VSP-W) $\subset$ $\mathcal{L}$(SP) $\cap$ $\mathcal{L}$(NTS), and that $\mathcal{L}$(PAR) and $\mathcal{L}$(VSP) are incomparable.

First we give an example of a language that is a parenthesis language, but not a VSP language.

Consider the language $K = \{(a(b)c), (b), (d)\}$. K is generated by the parenthesis grammar with productions $S \to (aBc)$, $S \to (b)$, $S \to (d)$, $B \to (b)$, where S is the initial nonterminal. Hence K is a parenthesis language.

Assume that there is a VSP grammar G' such that $L(G') = K$. Since $(a(b)c) \in L(G')$, $(b) \in L(G')$, and $(d) \in L(G')$, it follows by Lemma 3.3.5 that $(a(d)c) \in L(G')$, but $(a(d)c) \notin K$. Thus there is no VSP grammar that generates K.

Hence $\mathcal{L}(\text{PAR}) \not\subseteq \mathcal{L}(\text{VSP})$. Note that K is an example of a TC language that is not a BD-TC language.

Obviously, $\{ab, ac\}$ is an example of a language that is a VSP language (the cf grammar with productions $S \to ab$, $S \to ac$ is VSP), but not a parenthesis language, since we can not distinguish the right parenthesis.

Hence $\mathcal{L}(\text{VSP}) \not\subseteq \mathcal{L}(\text{PAR})$.

Finally we prove that $\mathcal{L}(\text{VSP-W}) \subset \mathcal{L}(\text{SP}) \cap \mathcal{L}(\text{NTS})$. The language $\{aa\}$ is obviously in $\mathcal{L}(\text{SP}) \cap \mathcal{L}(\text{NTS})$. We will show that it is not generated by a VSP-W grammar, by using the following claim.

> **Claim 3.4.7** *If $G = (\Sigma, \Delta, P, S)$ is a* VSP-W *grammar, and $X\alpha Y$ is a sentential form with $X, Y \in \Sigma$, $\alpha \in \Sigma^*$, then $X \neq Y$.*
>
> **Proof.** Assume to the contrary that G has sentential forms of the form $X\alpha X$. Let d be a derivation with minimal length of a sentential form of this type, say d yields $X\alpha X$. Since $X \in L_G \cap R_G$ and G is bracketed, each production with X in its right-hand side is of the form $A \to X$. Since G is backwards deterministic, there is exactly one such production. Let t be the derivation tree of $X\alpha X$ corresponding with d. If we cut off the leftmost and the rightmost leaf of t, then we obtain a derivation tree t' of $A\alpha A$ in G. But the derivation of $A\alpha A$ corresponding with t' is shorter than d. This contradicts the minimality of d. Consequently the claim holds. □

From Claim 3.4.7 it follows at once that $\{aa\} \notin \mathcal{L}(\text{VSP-W})$. □

Note that both $\mathcal{L}(\text{NTS})$ and $\mathcal{L}(\text{SP})$ are proper subclasses of the class of deterministic cf languages (see [2] for NTS languages).

From Theorem 3.4.6 we obtain the following result on TC languages.

Theorem 3.4.8 *The equivalence problem for* TC *grammars is decidable.*

Proof. Given two TC grammars, we can, by Theorem 3.2.13, construct for each of them an equivalent TC grammar in the wider sense that is backwards deterministic. By Theorem 3.3.6, these grammars are VSP-W. By Lemma 3.4.1, we can construct for each of these grammars an equivalent chain-free VSP-W grammar. By Theorem 3.3.17 these are NTS grammars. By [15] their equivalence is decidable. □

Remark 3.4.9 By Remark 3.3.14, VSP languages, and hence BD-TC languages are LR(0) languages (in the sense of [10]). However TC languages (or, equivalently, VSP-W languages) are in general not LR(0). Intuitively, one would think that they are, since we have presented a parsing method for them without look-ahead (in Definition 3.3.9), but this asks for an extension of the LR(0) notion to grammars in the wider sense. The problem here is that parsing "without look-ahead" is not really a well-defined notion, although intuitively clear.

The fact that not all TC languages are LR(0) can be shown using the following characterization of LR(0) languages given in [10, Theorem 13.3.1]: a language K is LR(0) iff K

is deterministic context-free and for all words u, y, w, if $uy \in K$, $u \in K$, and $w \in K$, then $wy \in K$. It follows that the language $K = \{a, ab\}$ is not LR(0), otherwise the fact that $ab(= uy)$, $a(= u)$, and $ab(= w)$ are in K would imply that $abb(= wy)$ is also in K. The cf grammar with productions $S \to a$, $S \to Tb$, $T \to a$ generates K and is terminally coded by the marked tree-code with productions $S \to a$, $S \to Sb$, and hence K is a chain-free TC language.

Also, by the above characterization, a TC language is LR(0) if it is generated by a cf grammar which is terminally coded by a marked tree-code φ such that $one_\varphi \notin L_\varphi$; it can be shown, using Theorems 3.2.13 and 3.2.4 and Lemma 3.1.10, that such a TC language is prefix-free. Note that all marked tree-codes from the examples in Sections 3.1 and 3.2 satisfy the condition that $one_\varphi \notin L_\varphi$. □

From the viewpoint of coding, it is natural to restrict oneself to trees that have no chains. Hence one might add to the definition of terminally coded grammars the condition that they are chain-free. Note that derivation trees of chain-free grammars still may have chains, all of them ending in a leaf. However, any code for chain-free trees can easily be adapted to trees of this type, as suggested in Example 3.1.14.

It turns out that requiring chain-freeness for TC grammars does make a difference for the class of generated languages. We will compare the classes of languages generated by chain-free grammars of the types introduced in Sections 3.2 and 3.3. We use $\mathcal{L}_c(X)$ to denote the class of languages generated by chain-free X grammars.

It is easy to see that Theorem 3.2.13 and Remark 3.2.14 still hold for languages generated by chain-free grammars. Hence again we obtain that $\mathcal{L}_c(\text{BD-TC-W}) = \mathcal{L}_c(\text{TC}) = \mathcal{L}_c(\text{TC-W})$. Furthermore, all parenthesis grammars are chain-free, and hence $\mathcal{L}_c(\text{PAR}) = \mathcal{L}(\text{PAR})$ and $\mathcal{L}_c(\text{BD-PAR}) = \mathcal{L}(\text{BD-PAR})$.

However, for chain-free grammars there is no result analogous with Lemma 3.4.4. Consequently the inclusion diagram of classes of languages generated by chain-free grammars is different from Figure 3.9.

Theorem 3.4.10 *The inclusions given in Figure 3.10 hold. The classes* $\mathcal{L}_c(\text{VSP})$ *and* $\mathcal{L}_c(\text{TC})$*, the classes* $\mathcal{L}_c(\text{PAR})$ *and* $\mathcal{L}_c(\text{VSP})$*, and the classes* $\mathcal{L}_c(\text{PAR})$ *and* $\mathcal{L}_c(\text{BD-TC})$ *are incomparable.*

Proof. As observed before, the language $\{ab, ac\}$ is not a parenthesis language. For the cf grammar G with productions $S \to ab$ and $S \to ac$, and initial nonterminal S, and the projection χ defined by $\chi(a) = a$, $\chi(b) = \chi(c) = b$, $\chi(S) = S$, the cf grammar $\chi(G)$ is a marked tree-code. Hence, by Theorem 3.2.8, $\{ab, ac\} \in \mathcal{L}_c(\text{BD-TC}) - \mathcal{L}_c(\text{PAR})$.

In the proof of Theorem 3.4.6 it was shown that the language $K = \{(a(b)c), (b), (d)\}$ is a parenthesis language, but not a VSP language. It follows that $K \in \mathcal{L}_c(\text{PAR}) - \mathcal{L}_c(\text{VSP})$. Hence all that is left to prove is the existence of a language $L \in \mathcal{L}_c(\text{VSP}) - \mathcal{L}_c(\text{TC})$.

Let $L = \{ac, bc, adbd\}$. L is generated by the chain-free VSP grammar G from Example 3.3.7. Hence $L \in \mathcal{L}_c(\text{VSP})$.

We will prove that there is no chain-free BD-TC-W grammar that generates L. Assume to the contrary that there exists such a grammar $G = (\Sigma, \Delta, P, S)$, which is terminally coded

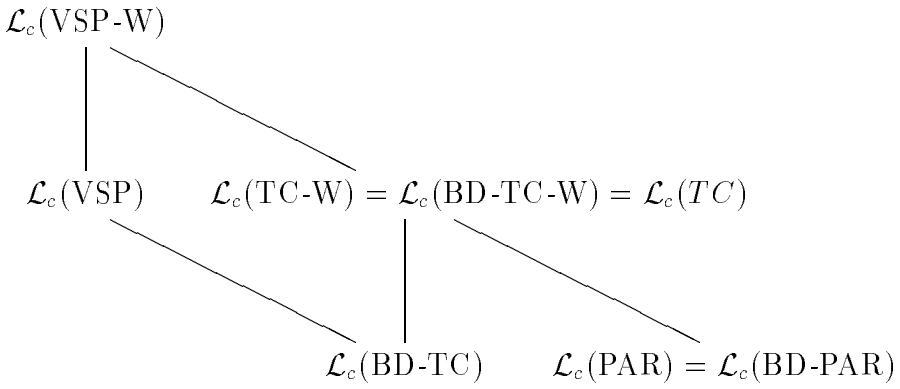


Figure 3.10: a hierarchy of chain-free language classes

by a marked tree-code φ. Note that, by Theorem 3.3.6, G is bracketed. Let t_1, t_2, t_3 be derivation trees of $ac, bc, adbd$, respectively. Define $\overline{a} \in \Sigma$ as follows : if $a \in L_G \cap R_G$, then $\overline{a} = A$ with $A \to a \in P$, otherwise $\overline{a} = a$. Define $\overline{b}, \overline{c}, \overline{d} \in \Sigma$ analogously. Since G is bracketed and backwards deterministic, $\overline{a}, \overline{b}, \overline{c}$, and $\overline{d}$ are well-defined, and, since G is chain-free, $\overline{a}, \overline{b}, \overline{c}, \overline{d} \notin L_G \cap R_G$.

For $i = 1, 2, 3$, let $\overline{t}_i$ be the derivation tree that is obtained from t_i by removing its chains. Then $yield(\overline{t}_1) = \overline{ac}$, $yield(\overline{t}_2) = \overline{bc}$, and $yield(\overline{t}_3) = \overline{adbd}$. Since $\overline{a}, \overline{b} \in L_G - R_G$ and $\overline{c}, \overline{d} \in R_G - L_G$, $\overline{t}_1, \overline{t}_2$, and $\overline{t}_3$ are as in Figure 3.11, where $T, U \in \Sigma$ are such that $T \to \overline{a}\,\overline{d} \in P$ and $U \to \overline{bd} \in P$, and $Z_1, Z_2, Z_3 \in S$.

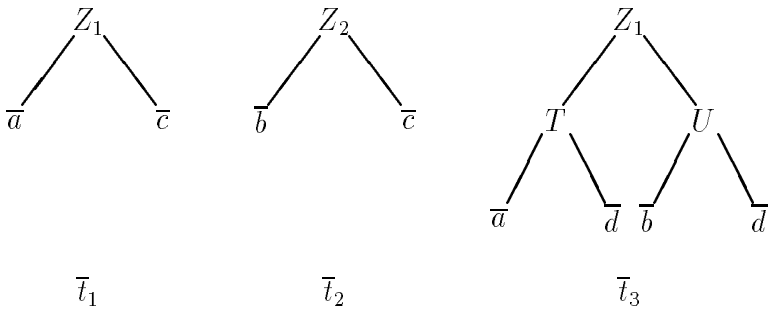


Figure 3.11: the derivation trees $\overline{t}_1$, $\overline{t}_2$, and $\overline{t}_3$

Since $und(\overline{t}_1) = und(\overline{t}_2)$, $\varphi(und(\overline{t}_1)) = \varphi(und(\overline{t}_2))$. Because$G$ is terminally coded by φ, it follows that $\chi(\overline{a}) = \chi(\overline{b})$. By the form of $\overline{t}_3$ it follows that $\chi(T) = \chi(U)$, since φ is backwards deterministic, and, since φ is bracketed, that $\chi(T) \in L_\varphi - R_\varphi$, $\chi(U) \in R_\varphi - L_\varphi$; a contradiction.

Hence there is no chain-free BD-TC-W grammar that generates L. Consequently, since $\mathcal{L}_c(\text{BD-TC-W}) = \mathcal{L}_c(\text{TC})$, $L \in \mathcal{L}_c(\text{VSP}) - \mathcal{L}_c(\text{TC})$. □

Note that Theorem 3.4.10 implies that in general TC grammars cannot be made chain-free. In particular (as shown in the proof of Theorem 3.4.10 there is no chain-free TC grammar equivalent with the BD-TC grammar of Example 3.4.5.

Finally, in Figure 3.12, the relations between language classes of both Figure 3.9 and Figure 3.10 are combined. Note that $\mathcal{L}_c(\text{VSP}) = \mathcal{L}(\text{VSP})$ and $\mathcal{L}_c(\text{VSP-W}) = \mathcal{L}(\text{VSP-W})$ by Lemma 3.4.1.

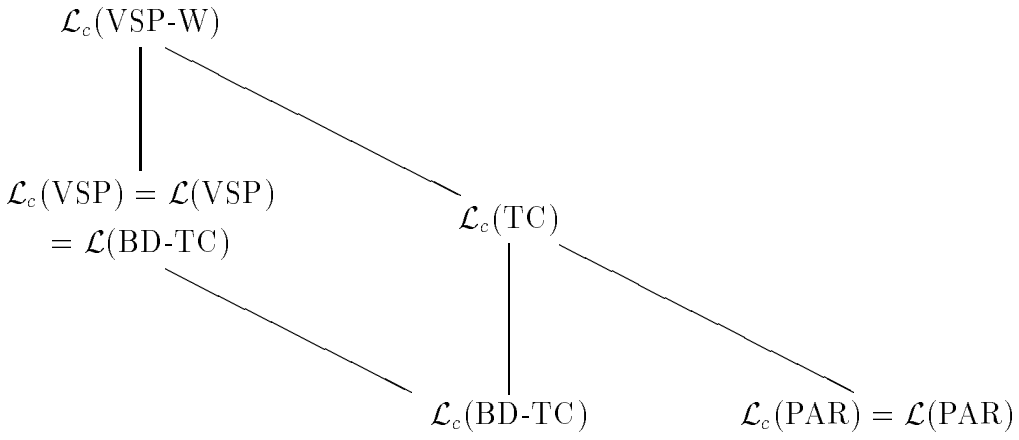


Figure 3.12: the combined hierarchy of language classes

Bibliography of Part I

[1] A.H. Aho and J.D. Ullman, *The Theory of Parsing, Translation, and Compiling*, Prentice Hall, New Jersey, 1972.

[2] L. Boasson and G. Senizergues, NTS languages are deterministic and congruential, *Journal of Computer and System Sciences* **31** (1985) 332–342.

[3] Chapter 3 of this thesis:

A. Ehrenfeucht, J. Engelfriet, P. ten Pas, and G. Rozenberg, Grammatical codes of trees and terminally coded grammars, to appear in *Fundamenta Informaticae*.

[4] Chapter 1 of this thesis, published as:

A. Ehrenfeucht, P. ten Pas, and G. Rozenberg, Properties of grammatical codes of trees, *Theoretical Computer Science* **125** (1994) 259–293.

[5] A. Ehrenfeucht and G. Rozenberg, Grammatical codes of trees, *Discrete Applied Mathematics* **32** (1991) 103–129.

[6] R.W. Floyd, Bounded context syntactic analysis, *Communications of the ACM* **7** (1964) 62–67.

[7] S. Ginsburg and M.A. Harrison, Bracketed context-free languages, *Journal of Computer and System Sciences* **1** (1967) 1–23.

[8] F. Gécseg and M. Steinby, *Tree Automata*, Akadémiai Kiadó, Budapest, 1984.

[9] F. Harary, *Graph Theory*, Addison-Wesley, Reading, Massachusetts, 1969.

[10] M.A. Harrison, *Introduction to Formal Language Theory*, Addison-Wesley, Reading, Massachusetts, 1978.

[11] G. Huet, Confluent reductions: abstract properties and applications to term rewriting systems, *Journal of the ACM* **27** (1980) 797–821.

[12] D. Knuth, *The Art of Computer Programming, vol. 1 : Fundamental Algorithms*, Addison-Wesley, Reading, Massachusetts, 1973.

[13] R. McNaughton, Parenthesis grammars, *Journal of the ACM* **14** (1967) 490–500.

[14] A. Salomaa, *Formal Languages*, Academic Press, New York and London, 1973.

[15] G. Senizergues, The equivalence and inclusion problems for NTS languages, *Journal of Computer and System Sciences* **31** (1985) 303–331.

[16] D. Wood, *Grammar and L Forms : An Introduction*, Lecture Notes in Computer Science 91, Springer, Berlin, 1980.

Part II

Text Languages

Chapter 4

Context-free Text Grammars

Abstract

A text is a triple $\tau = (\lambda, \rho_1, \rho_2)$ such that λ is a labeling function, and ρ_1 and ρ_2 are linear orders on the domain of λ; hence τ may be seen as a word (λ, ρ_1) together with an additional linear order ρ_2 on the domain of λ. The order ρ_2 is used to give to the word (λ, ρ_1) its *individual* hierarchical representation (syntactic structure) which may be a tree but it may be also more general than a tree. In this paper we introduce *context-free grammars for texts* and investigate their basic properties. Since each text has its own individual structure, the role of such a grammar should be that of a definition of a pattern common to all individual texts. This leads to the notion of a *shapely* context-free text grammar also investigated in this paper.

Introduction

This paper continues the investigation of *texts*, initiated in [16] and [11]. A *text* is a triple $(\lambda, \rho_1, \rho_2)$ such that λ is a labeling function, and ρ_1 and ρ_2 are linear orders on the domain of λ. Hence a text may be seen as a generalization of a word, where we specify a word as a pair (λ, ρ) such that λ is a labeling function, and ρ is a linear order on the domain of λ.

The notion of text originates in the theory of 2-structures. The so-called (*labeled*) T-*structures* form an important subclass of (labeled) 2-structures (see [15]). It turns out that each T-structure may be specified through a pair of linear orders. In this way one gets a very exact correspondence between (specifically) labeled T-structures and pairs of linear orders. This leads to the notion of a text which is then a *specification* of a labeled T-structure on the domain of the labeling function of the text. One of the main developments of the theory of 2-structures is the theory of their hierarchical representations (see, e.g., [13] and [14]). In particular, it is known that each 2-structure g has a unique hierarchical representation called the *shape* of g. Since with each text τ we may associate a unique labeled T-structure (viz., the labeled T-structure g specified by τ), we get in this way for each text τ a unique hierarchical representation (viz., the shape of g).

In this way we may see a text $(\lambda, \rho_1, \rho_2)$ as a word (λ, ρ_1) *together* with a hierarchical representation (syntactic structure) given by ρ_2. Such a hierarchical representation may be a tree but it also may be more general than a tree. Reasoning in this way one is lead to

a different way of looking at the formal language theory. Each text has its own individual structure - one does not need grammars in order to assign syntactic structures. Rather, one can see a grammar as a definition of a certain "pattern" common to hierarchical structures of a set of texts. This point of view is exploited in this paper. We introduce here *context-free text grammars* (i.e., context-free grammars generating texts) and investigate their basic properties. The paper is organized as follows.

In the first two sections we recall some basic notions and results concerning 2-structures, T-structures, and texts; in this way the paper is self-contained. Context-free text grammars are introduced in Section 4.3, and Section 4.4 establishes some "traditional" (in the sense of formal language theory) normal forms. In Section 4.5 we consider a normal form specific for texts rather than words; it has to do with the so-called *primitive texts*. Each text τ generated by a context-free grammar G gets a grammatical structure, viz. a derivation tree in G. On the other hand τ has its own individual structure, viz. its shape. In this setup one gets immediately a criterion for a "good" context-free grammar G for a given set K of texts: G is good if its derivation trees match the shapes of texts in K; such a grammar is called *shapely*. Shapely context-free text grammars are investigated in Section 4.6. The traditional important notions of ambiguity and pumping properties of context-free text grammars and languages are investigated in Section 4.7. The discussion in Section 4.8 concludes the paper.

Preliminaries

In this section we establish notations and terminology used in this paper. We assume the reader to be familiar with basic graph theoretical notions, in particular those concerning trees.

For a set Z, $\#Z$ denotes its cardinality; $\emptyset$ denotes the empty set. $S_2(Z) = \{\{x,y\} \mid x,y \in Z \text{ and } x \neq y\}$, and $E_2(Z) = \{(x,y) \mid x,y \in Z \text{ and } x \neq y\}$; each element of $E_2(Z)$ is a *2-edge over* Z. If $e = (x,y)$ is a 2-edge, then the *reverse of* e, denoted $rev(e)$, is the 2-edge (y,x). For a set T of 2-edges, the *reverse of* T, denoted $rev(T)$, is the set $\{rev(e) \mid e \in T\}$. Sets X, Y are *overlapping* if $X - Y \neq \emptyset$, $Y - X \neq \emptyset$, and $X \cap Y \neq \emptyset$. For sets X, Y we write $X \subseteq Y$ if X is included in Y, $X \subset Y$ if X is *strictly* included in Y, and $X \times Y$ denotes the Cartesian product of X, Y.

For sets X and Y, and a relation $\rho \subseteq X \times Y$, $dom(\rho) = \{x \in X \mid (x,y) \in \rho \text{ for some } y \in Y\}$. In a partition of a set we assume that each partition class is nonempty.

For a sequence s, $|s|$ denotes its length, and for $1 \leq i \leq |s|$, $s(i)$ denotes the i'th element of s. By a *function* in this paper we understand a set of ordered pairs φ such that, for all $(x,y),(u,v) \in \varphi$, $x = u$ implies $y =$ v. We say that φ is a *function on* $dom(\varphi)$. If $Z \subseteq dom(\varphi)$, then $\varphi|_Z$ denotes the restriction of φ to Z.

A (directed) *graph* is an ordered pair $h = (D,T)$, where D is a (finite) nonempty set of *nodes*, denoted by $nd(h)$, and $T \subseteq D \times D$ is the set of *edges*. h is *antireflexive* if, for each $x \in D$, $(x,x) \notin T$; h is *transitive* if, for all $x,y,z \in D$, $(x,y) \in T$ and $(y,z) \in T$ implies $(x,z) \in T$. h is a *linear order* if h is antireflexive, transitive, and for each $(x,y) \in E_2(D)$, either $(x,y) \in T$ or $(y,x) \in T$. We carry over to T the terminology and notations concerning h. Hence, T is a *linear order* iff h is a linear order.

If $h = (D, T)$ is a linear order such that $D = \{x_1, \ldots, x_n\}$ for some $n \geq 2$, and $(x_j, x_{j+1}) \in T$ for all $j \in \{1, \ldots, n-1\}$, then we write T in the form $(x_1, \ldots, x_n)$. Hence a linear order (D, T) with $\#D \geq 2$ can be specified as a sequence of the elements of D. The terminology and notations concerning sequences carry over to linear orders.

We extend this specification of T as a sequence to the case where D consists of one element x. Then according to the above definition $T \subseteq E_2(D)$ is the empty set, but for technical convenience we specify T as the sequence (x) of length 1. If $h = (D, T)$ is a linear order and $X \subseteq D$, then X is a *segment of* h (or *of* T) if for all $y, z \in X$, if $u \in D$ is such that $(y, u) \in T$ and $(u, z) \in T$, then $u \in X$. We use $seg(h)$ (or $seg(T)$) to denote all segments of h. For $X \subseteq D$, $h|_X$ denotes the restriction of h to X, i.e., $(X, T \cap E_2(X))$, and $T|_X$ denotes $T \cap E_2(X)$. For $x \in dom(h)$, $h_{<x}$ (or $T_{<x}$) denotes the restriction of h (or T) to the set $\{y \in dom(h) \mid (y, x) \in T\}$; similarly, $h_{>x}$ (or $T_{>x}$) is the restriction of h (or T) to the set $\{y \in dom(h) \mid (x, y) \in T\}$.

Let (h_1, h_2) with $h_1 = (D_1, T_1)$, $h_2 = (D_2, T_2)$ be an ordered pair of disjoint linear orders (i.e., $D_1 \cap D_2 = \emptyset$). The *sum of* h_1 and h_2, denoted $h_1 + h_2$, is the linear order $h = (D_1 \cup D_2, T)$, such that $T = T_1 \cup T_2 \cup \{(x, y) \mid x \in D_1 \text{ and } y \in D_2\}$. Note that the sum operation is not commutative.

Two graphs $h_1 = (D_1, T_1)$, $h_2 = (D_2, T_2)$ are *isomorphic* if there is a bijection $\varphi : D_1 \to D_2$ such that, for all $x, y \in D_1$, $(x, y) \in T_1$ iff $(\varphi(x), \varphi(y)) \in T_2$; φ is an *isomorphism* between h_1 and h_2 .

A graph $t = (D, T)$ is a *tree* if h is acyclic and there exists a node v of t (the *root of* t, denoted $root(t)$) such that each node of t is reachable from v by a unique path. We use $leaf(t)$ to denote the set of *leaves* of t, and $in(t)$ to denote the set of *inner nodes* of t (i.e., $in(t) = nd(t) - leaf(t)$). For a node $v \in in(t)$, $ddes_t(v)$ denotes the set of *direct descendants of* v *(in* t*)*, i.e., nodes x such that $(v, x) \in T$; the *contribution of* v *(in* t*)*, denoted by $contr_t(v)$ is the set $\{w \in leaf(t) \mid \text{ there is a path from } v \text{ to } w\}$. Furthermore, *out-degree*(t) denotes $\max\{\#ddes_t(v) \mid v \in in(t)\}$, and t is *chain-free* if each inner node has at least two direct descendants. If $t = (D, T)$ and $t' = (D', T')$ are trees such that $D' \subseteq D$, and for each $v \in in(t')$, $ddes_{t'}(v) = ddes_t(v)$, then we say that t' is a *subtree of* t; t' is an *elementary subtree of* t if there exists $v \in in(t)$ such that $D' = \{v\} \cup ddes_t(v)$. For $v \in in(t)$, if t' is a subtree of t with $root(t') = v$ and $leaf(t') \subseteq leaf(t)$, then t' is denoted by $sub_t(v)$.

Mostly, the trees in this paper have one or more *labeling functions* on (a subset of) the nodes. A *node-labeled tree* is a triple $t = (D, T, \eta)$, where $b = (D, T)$ is a tree, and η is a function on $nd(b)$. A (*elementary*) *subtree of* t is a node-labeled tree $t' = (D', T', \eta')$ such that (D', T') is a (elementary) subtree of b, and $\eta' = \eta|_{D'}$. An *inner-labeled tree* is a triple $t = (D, T, \eta)$, where $b = (D, T)$ is a tree, and η is a function on $in(b)$. A (*elementary*) *subtree of* t is an inner-labeled tree $t' = (D', T', \eta')$ such that $b' = (D', T')$ is a (elementary) subtree of b, and $\eta' = \eta\ |_{in(b')}$. In both cases, notations for the tree b carry over to the node-labeled (inner-labeled) tree t, e.g., we may write $in(t)$, $ddes_t(v)$ instead of $in(b)$, $ddes_b(v)$.

Obviously, these definitions may be extended to trees with more than one labeling function.

Often for the inner-labeled trees considered in this paper the identity of the inner nodes or even the identity of all nodes is not important. Hence in that case for an inner-labeled tree (D, T, η), the tree (D, T) is considered modulo isomorphism (where in the first case the isomorphism on the leaves is the identity).

4.1 Labeled 2-structures

In this section we recall some of the notions and results concerning labeled 2-structures from [13, 14, 15].

Definition 4.1.1 A *labeled 2-structure* (abbreviated ℓ2s) is a 3-tuple $g = (D, \Delta, \delta)$, where D is a finite nonempty set, Δ is a finite alphabet, and δ is a function from $E_2(D)$ into Δ.

As usual, we assume that (labeled) 2-structures are "reversible", i.e., if $g = (D, \Delta, \delta)$ is a ℓ2s, and $e_1, e_2 \in E_2(D)$, then $\delta(e_1) = \delta(e_2)$ iff $\delta(rev(e_1)) = \delta(rev(e_2))$. It is demonstrated in [13] that reversible 2-structures constitute a normal form for 2-structures in a well-defined sense.

For formal reasons we also admit 2-structures of one element keeping in mind that this is a "degenerate" special case (e.g., in order to avoid too many technicalities we will not be adapting our definitions for the 1-element case).

Let $g = (D, \Delta, \delta)$ be a ℓ2s. The *domain* D of g is denoted by $dom(g)$. The *labeling function* δ induces an equivalence relation on the set of 2-edges of g as follows: for all $e_1, e_2 \in E_2(D)$, e_1 is equivalent with e_2 iff $\delta(e_1) = \delta(e_2)$. This equivalence relation on $E_2(D)$ corresponds to a partition of $E_2(D)$ into equivalence classes, which is denoted by $part(g)$.

The set of labels occurring in g is denoted by $ualph(g)$, i.e., $ualph(g) = \{\delta(e) \mid e \in E_2(D)\} \subseteq \Delta$. For technical reasons we consider ℓ2s's $g = (D, \Delta, \delta)$ and $g' = (D, \Delta', \delta)$ with $ualph(g) = ualph(g')$ as equal.

A 2-edge e of a ℓ2s $g = (D, \Delta, \delta)$ is *symmetric* if $\delta(e) = \delta(rev(e))$; otherwise e is called *asymmetric*. For each class $P \in part(g)$, $rev(P) \in part(g)$, and either $P = rev(P)$ (hence P consists of symmetric 2-edges only and is called *symmetric*), or $P \cap rev(P) = \emptyset$ (hence P consists of asymmetric 2-edges only and is called *antisymmetric*). Consequently, the classes in $part(g)$ can be grouped into sets $\{P, rev(P)\}$ – such a set is called a *feature of* g. If P is symmetric, then the corresponding feature has one class only; if P is antisymmetric the feature consists of two classes. A ℓ2s g is *symmetric* if all its 2-edges are symmetric and g is *antisymmetric* if all its 2-edges are asymmetric.

To represent ℓ2s's pictorially, we use the traditional graphical notation for edge-labeled graphs, where arrows depict antisymmetric 2-edges, and lines depict symmetric 2-edges.

Example 4.1.2 Consider the ℓ2s's g_1 and g_2 in Figure 4.1.

g_1 is symmetric and has two features. g_2 has one symmetric class $\{(1,2),(2,1),(2,4),(4,2),(3,4),(4,3)\}$ in $part(g_2)$ and two antisymmetric classes, $\{(1,3),(1,4),(2,3)\}$ and its reverse $\{(3,1),(4,1),(3,2)\}$. Hence g_2 has two features. □

For a ℓ2s $g = (D, \Delta, \delta)$ and $X \subseteq D$, the *substructure of g determined by X*, denoted $sub_g(X)$, is the ℓ2s $(X, \Delta, \delta|_X)$.

The basic technical notion concerning labeled 2-structures is the notion of a "clan".

Definition 4.1.3 Let $g = (D, \Delta, \delta)$ be a ℓ2s, and let $X \subseteq D$. X is a *clan (of g)* if for all $x, y \in X$ and all $z \in D - X$, $\delta(z, x) = \delta(z, y)$.

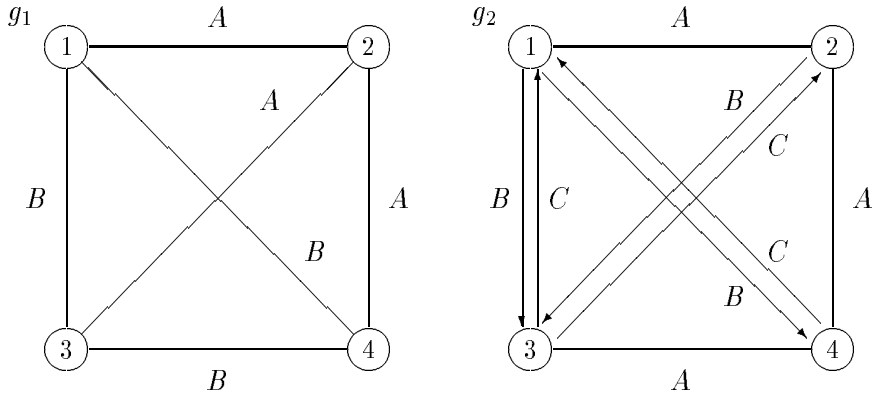


Figure 4.1: ℓ2s's g_1 and g_2

Hence a subset X of $dom(g)$ is a clan if each element outside X "sees" all elements of X "in the same way". We use $C(g)$ to denote the set of all clans of g. Clearly, for each ℓ2s $g = (D, \Delta, \delta)$, $\emptyset \in C(g)$, $D \in C(g)$, and for each $x \in D$, $\{x\} \in C(g)$. These clans are called the *trivial clans of* g, denoted by $TC(g)$. A *prime* clan of g is a clan that is not overlapping any other clan of g. The set of all prime clans of g is denoted by $PC(g)$. Note that every trivial clan is prime.

An important property of clans is the following one (see [13, Lemma 4.11]): if X and Y are disjoint clans of a ℓ2s g, then for all $x_1, x_2 \in X$ and all $y_1, y_2 \in Y$, (x_1, y_1) is equivalent with (x_2, y_2). This property allows one to form *quotients* as follows. If $g = (D, \Delta, \delta)$ is a ℓ2s, and M is a partition of D into clans of g, then g/M is the ℓ2s (M, Δ, δ'), where δ' is such that, for $(X, Y) \in E_2(M)$, $\delta'(X, Y) = \delta(x, y)$, where $x \in X$ and $y \in Y$. Note that we recover g from g/M if we "substitute", for each $X \in M$, $sub_g(X)$ for the node X in g/M. In this way the operation of substitution is the inverse of the operation of forming quotients, where the domains of the substituted ℓ2s's become clans of the resulting ℓ2s.

In this paper we consider the substitution of a single ℓ2s into a ℓ2s. Formally, for ℓ2s's $g = (D, \delta, \Delta)$ and $g' = (D', \delta', \Delta')$ with disjoint domains, and for $x \in dom(g)$, we define the *substitution of* g' *into* g *at* x, denoted by $sub(g, x, g')$, as the ℓ2s $((D - \{x\}) \cup D', \delta'', \Delta \cup \Delta')$, where for each 2-edge (y, z),

$$\delta''(y,z) = \begin{cases} \delta(y,z) & \text{if } y, z \in D - \{x\} \\ \delta'(y,z) & \text{if } y, z \in D' \\ \delta(y,x) & \text{if } y \in D - \{x\},\ z \in D' \\ \delta(x,z) & \text{if } y \in D',\ z \in D - \{x\} \end{cases}$$

This definition of substitution of a single ℓ2s is also given in [14, Definition 7.9].

The following subclasses of the class of labeled 2-structures are both natural and important.

Definition 4.1.4 Let $g = (D, \Delta, \delta)$ be a ℓ2s.
(1) g is *special* if $PC(g) = TC(g)$.
(2) g is *primitive* if $C(g) = TC(g)$.

(3) g is *complete* if either $\#D = 1$ or $\#part(g) = 1$.
(4) g is *linear* if either $\#D = 1$ or g is antisymmetric, $part(g) = \{P, rev(P)\}$ and there exists a linear order $(x_1, \ldots, x_n)$, $n \geq 2$ of D such that, for all different $i, j \in \{1, \ldots, n\}$, $(x_i, x_j) \in P$ iff $i < j$.

Example 4.1.5 Let g and h be as in Figure 4.2.

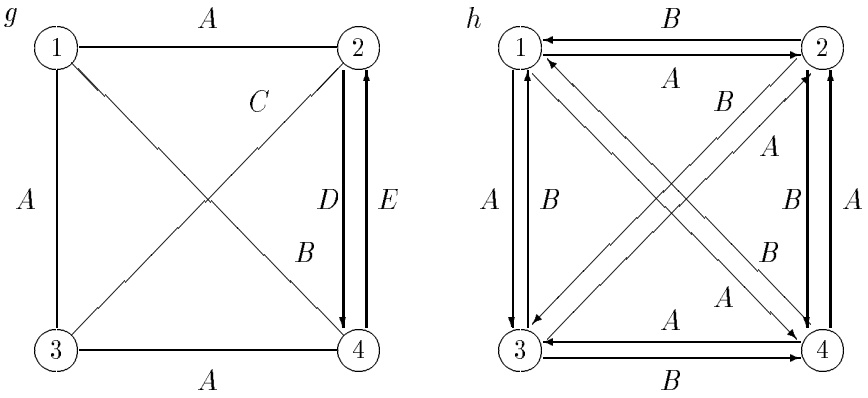


Figure 4.2: ℓ2s's g and h

$C(g) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{4\}, \{1, 2, 3, 4\}\} = TC(g)$, hence g is a primitive ℓ2s. h is a linear ℓ2s (consider the linear order $(1, 4, 3, 2)$ of its domain). □

Let $g = (D, \delta, \Delta)$ and $g' = (D', \delta', \Delta)$ be ℓ2s's. g and g' are *isomorphic* if there is a bijection κ from D onto D' such that for all $(x, y) \in E_2(D)$, $\delta(x, y) = \delta'(\kappa(x), \kappa(y))$. *Note that the notion of isomorphism we use in this paper is label-preserving.*

A basic technical result for labeled 2-structures is that a ℓ2s g is special iff g is complete, linear or primitive. The decomposition theory developed in [14] allows one to define tree-like "hierarchical representations" of labeled 2-structures using these three types of labeled 2-structures only. We recall now the basic notions of this theory.

A ℓ2s-*labeled tree* is an inner-labeled tree $t = (D, T, \varphi)$ where φ is such that for each $v \in in(t)$, $\varphi(v)$ is a ℓ2s g with $dom(g) = ddes_t(v)$. The labeling function φ is denoted by $\ell 2s_t$.

Note that our definition of a ℓ2s-labeled tree differs from the one in [14] in that we admit chains. We do this because later (see, e.g., Definition 2.3 and Theorem 3.1) we establish relationships between derivation trees in "context-free text grammars" (which naturally have chains) and ℓ2s-labeled trees.

A ℓ2s-labeled tree t is called *locally special* (*primitive*, *linear*) if for each $v \in in(t)$, $\ell 2s_t(v)$ is special (primitive, linear). A ℓ2s-labeled tree t defines a ℓ2s $g = (D, \delta, \Delta)$ as follows: $D = leaf(t)$, and for all $v \in in(t)$, $u, w \in ddes_t(v)$, $x \in contr_t(u)$, $y \in contr_t(w)$, $\delta(x, y) = \delta'(u, w)$, where δ' is the labeling function of $\ell 2s_t(v)$. We say that t *represents* g, and g is denoted by $\ell 2s(t)$. By the construction of $\ell 2s(t)$ we obtain the following lemma.

Lemma 4.1.6 *Let g be a $\ell 2s$, and let t be a $\ell 2s$-labeled tree with $dom(g) = leaf(t)$. Then t represents g iff for each $v \in in(t)$, $contr_t(v)$ is a clan of g, and $\ell 2s_t(v)$ is isomorphic to the quotient-structure $sub_g(contr_t(v))/\{contr_t(u) \mid u \in ddes_t(v)\}$.*

Note that for a ℓ2s-labeled tree representing a ℓ2s, the identity of the inner nodes is not important. Hence ℓ2s-labeled trees that differ *only* in the identities of the inner nodes (i.e., there exists an isomorphism between the sets of inner nodes of the trees with corresponding ℓ2s-labels) may be identified. This is implicitly used in the following notion.

Definition 4.1.7 Let g be a ℓ2s. The *shape of* g, denoted $shape(g)$, is the chain-free ℓ2s-labeled tree t representing g such that $\{contr_t(v) \mid v \in nd(t)\} = PC(g) - \{\emptyset\}$.

The requirement of chain-freeness is needed in order to ensure that the shape is unique.

The following proposition is the main theorem of the decomposition theory for labeled 2-structures.

Proposition 4.1.8 (cf. [14, Th. 4.2, Cor 6.18])
(1) *For each $\ell 2s$ g, $shape(g)$ is locally special.*
(2) *If g and h are $\ell 2s$'s, then $shape(g) = shape(h)$ iff $g = h$.*

Hence in a shape each of the ℓ2s's corresponding to an inner node is either complete, linear or primitive.

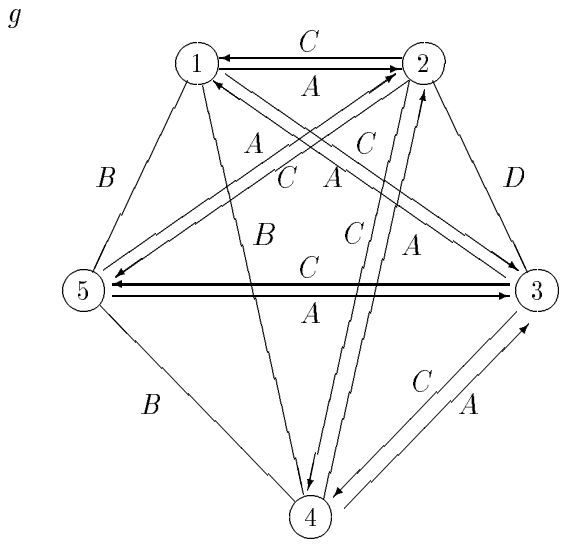


Figure 4.3: ℓ2s g

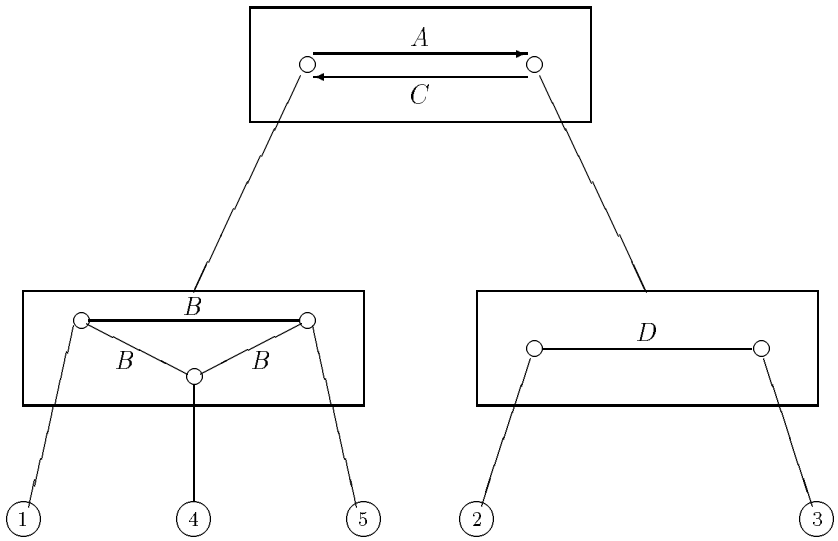


Figure 4.4: $shape(g)$

Example 4.1.9 Let g be the ℓ2s in Figure 4.3.
Then $shape(g)$ is as in Figure 4.4.
Here each rectangle is an inner node, and the ℓ2s labeling the node is drawn inside.
Note that indeed $\ell 2s_{shape(g)}(v)$ is special for each inner node v. □

Remark 4.1.10 Substitution of ℓ2s's can be described in terms of ℓ2s-labeled trees. For ℓ2s's g and g' with disjoint domains, and $x \in dom(g)$, $subst(g, x, g')$ is precisely the ℓ2s represented by the ℓ2s-labeled tree t with two inner nodes: $root(t)$ labeled by g, and the inner node x labeled by g'. □

The primitive substructures of a ℓ2s can be localized in its clans.

Proposition 4.1.11 (cf. [14, Th. 4.4]) *Let g be a ℓ2s, let t be a ℓ2s-labeled tree representing g, and let $X \subseteq dom(g)$ be such that $\#X \geq 3$ and $sub_g(X)$ is primitive. Then there exists $v \in in(t)$ such that $X \subseteq contr_t(v)$ and $sub_g(X)$ is isomorphic with a primitive substructure of $\ell 2s_t(v)$.*

Example 4.1.12 Consider the ℓ2s g in Figure 4.5.
Note that $sub_g(\{1,2,3\})$ is primitive. It is inside the node $\{1,2,3,4\}$ of the ℓ2s-labeled tree t in Figure 4.6 representing g. □

Note that Proposition 4.1.11 does *not* hold for linear and complete substructures.

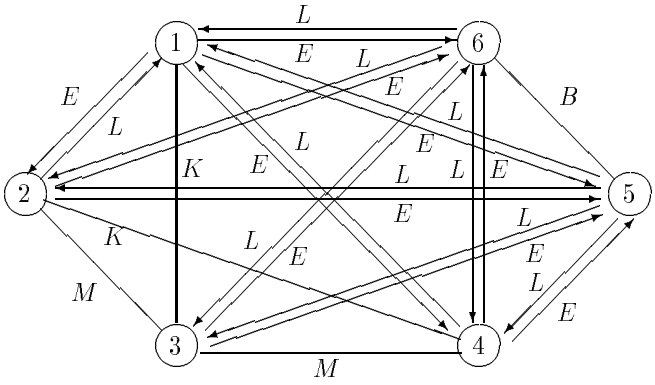


Figure 4.5: ℓ2s g

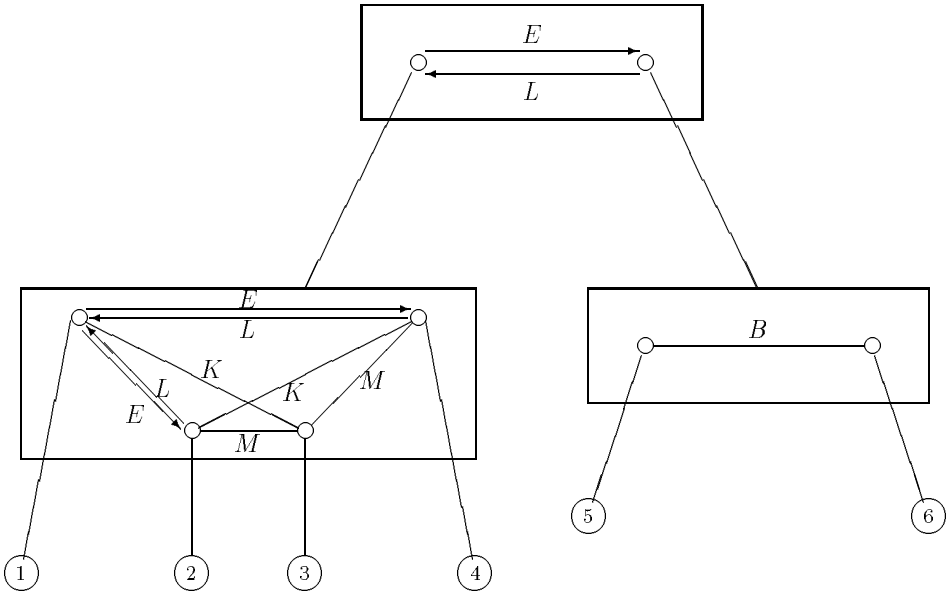


Figure 4.6: ℓ2s-labeled tree t representing g

Example 4.1.13 Let g and t be the ℓ2s and the ℓ2s-labeled tree from Example 4.1.12. The substructure $sub_g(\{1,4,5\})$ is linear, but it does not occur as a substructure of a "local" ℓ2s. □

Those ℓ2s-labeled trees that are shapes can be characterized as follows. Let t be a locally special chain-free ℓ2s-labeled tree. Define the relation $\equiv_t$ on $in(t)$ as follows: for $u, v \in in(t)$, $u \equiv_t v$ iff $u \in ddes_t(v)$, either both $\ell 2s_t(u)$ and $\ell 2s_t(v)$ are linear or both $\ell 2s_t(u)$ and $\ell 2s_t(v)$ are complete, and $\ell 2s_t(u)$ and $\ell 2s_t(v)$ use the same labels, i.e., $ualph(\ell 2s_t(u)) = ualph(\ell 2s_t(v))$.

A locally special chain-free ℓ2s-labeled tree t is *disjoint* if for all $u, v \in in(t)$, $u \not\equiv_t v$.

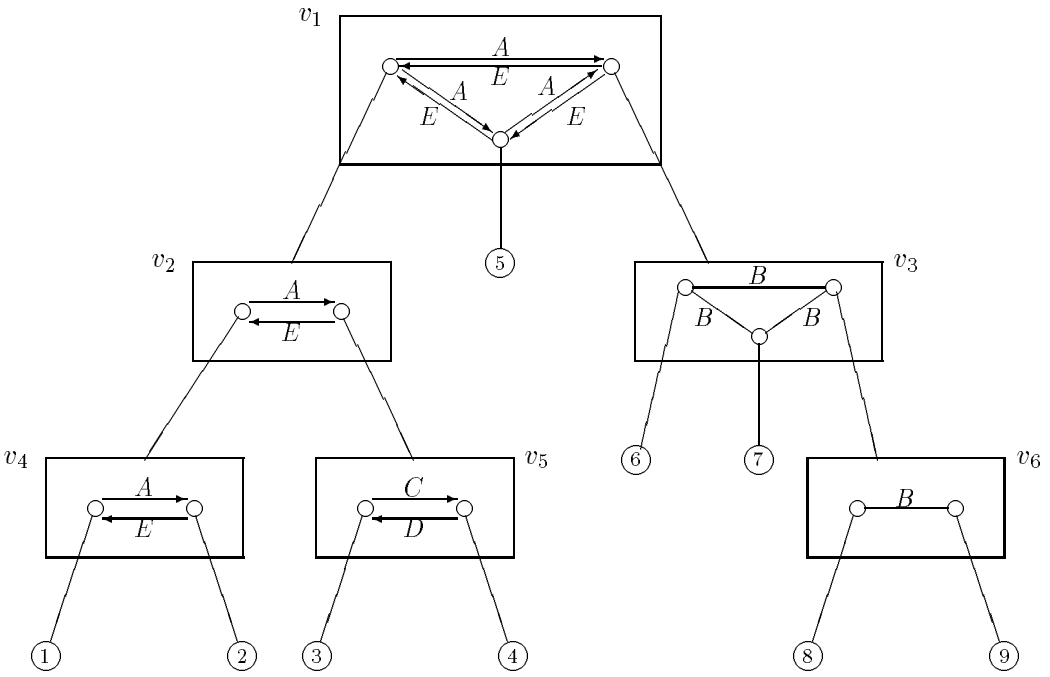


Figure 4.7: ℓ2s-labeled tree t

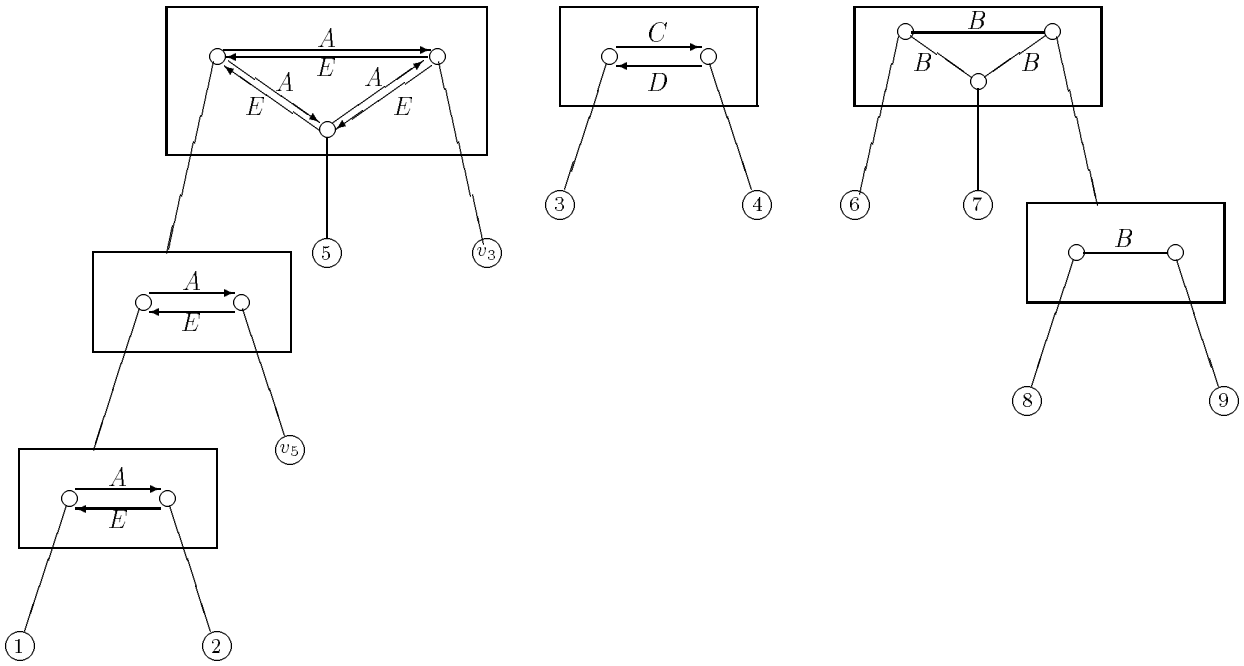


Figure 4.8: subtrees of equivalence classes

Proposition 4.1.14 (cf. [14, Cor. 6.20]) *A $\ell 2s$-labeled tree t representing a $\ell 2s$ g is the shape of g iff t is chain-free, locally special, and disjoint.*

Let $\equiv_t^*$ be the symmetric, reflexive, and transitive closure of $\equiv_t$. The following result can easily be verified.

Lemma 4.1.15 *Let $t = (D, T, \varphi)$ be a locally special chain-free $\ell 2s$-labeled tree, and let $\mathcal{E}$ be an equivalence class of $\equiv_t^*$. Let $D' = (\bigcup_{v \in \mathcal{E}} ddes_t(v)) \cup \mathcal{E}$, let $T' = T \cap E_2(D')$, and let $\varphi' = \varphi|_{\mathcal{E}}$. Then*
(1) *$t' = (D', T', \varphi')$ is a subtree of t, and*
(2) *$\ell 2s(t')$ is linear (complete, primitive) if t' is locally linear (complete, primitive); hence $\ell 2s(t')$ is special.*

The ℓ2s-labeled tree t' formed as above by an equivalence class $\mathcal{E}$ is denoted by $tree_t(\mathcal{E})$. Note that, by Proposition 4.1.14, t is the shape of a ℓ2s iff $tree_t(\mathcal{E})$ is an elementary subtree of t for each equivalence class $\mathcal{E}$ of $\equiv_t^*$.

Example 4.1.16 Consider the ℓ2s-labeled tree t in Figure 4.7.

The equivalence classes of $\equiv_t^*$ are $\{v_1, v_2, v_4\}$, $\{v_5\}$, $\{v_3, v_6\}$. The corresponding subtrees $tree_t(\{v_1, v_2, v_4\})$, $tree_t(\{v_5\})$, and $tree_t(\{v_3, v_6\})$ are given in Figure 4.8.

Hence $tree_t(\{v_1, v_2, v_4\})$ and $tree_t(\{v_5\})$ are locally linear subtrees of t, $tree_t(\{v_3, v_6\})$ is a locally complete subtree of t. The linear ℓ2s represented by $tree_t(\{v_1, v_2, v_4\})$ looks as in Figure 4.9. □

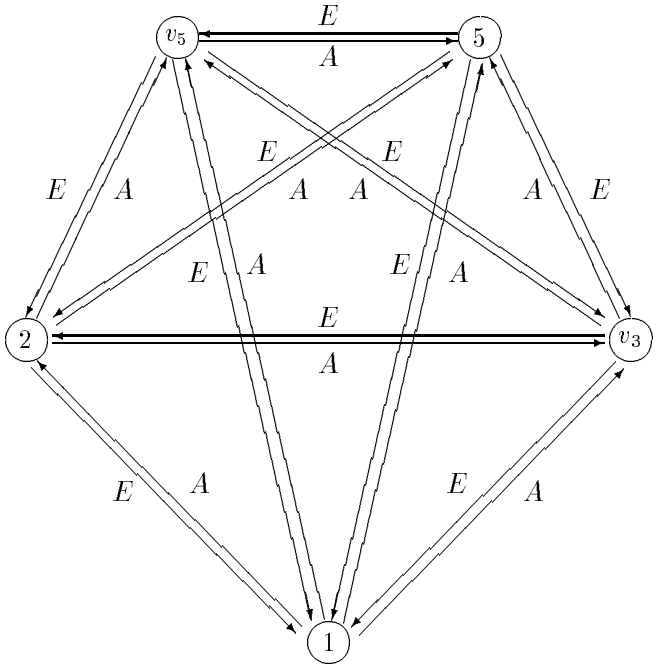


Figure 4.9: ℓ2s represented by $tree_t(\{v_1, v_2, v_4\})$

Each ℓ2s g has a "trivial" hierarchical representation: a tree consisting of a root labeled by g and leaves which are direct descendants of the root. The ℓ2s-labeled trees representing g are "refinements" of this trivial representation. In a (non-trivial) ℓ2s-labeled tree t representing g, each inner node v together with its direct descendants is a "trivial" representation of $\ell 2s_t(v)$. If these local trivial representations are "refined" we again obtain a ℓ2s-labeled tree representing g. We now give a formal definition of this notion of "refinement". In [14, Def. 6.21], the definition of refinement for chain-free ℓ2s-labeled trees is given in a different ("non-operational") way.

Definition 4.1.17 Let t be a ℓ2s-labeled tree. A ℓ2s-labeled tree $t' = (D, T, \varphi)$ is a *refinement of* t if for each $v \in in(t)$ there is a ℓ2s-labeled tree $t_v = (D_v, T_v, \varphi_v)$ with root v and leaves $ddes_t(v)$ representing $\ell 2s_t(v)$ such that $D = \bigcup_{v \in in(t)} D_v$, $T = \bigcup_{v \in in(t)} T_v$, and $\varphi = \bigcup_{v \in in(t)} \varphi_v$.

If t is refined into t' as above, and if v is an inner node of t, then we say that v *is replaced by* t_v. In particular, v may be replaced by the elementary subtree of t rooted at v, which means that v is left unaltered; in this way a refinement of t allows also for "real replacement" of some of the nodes of t only.

Note that a refinement of a ℓ2s-labeled tree t again represents $\ell 2s(t)$.

Example 4.1.18 Consider the ℓ2s-labeled trees t and t' in Figure 4.10.

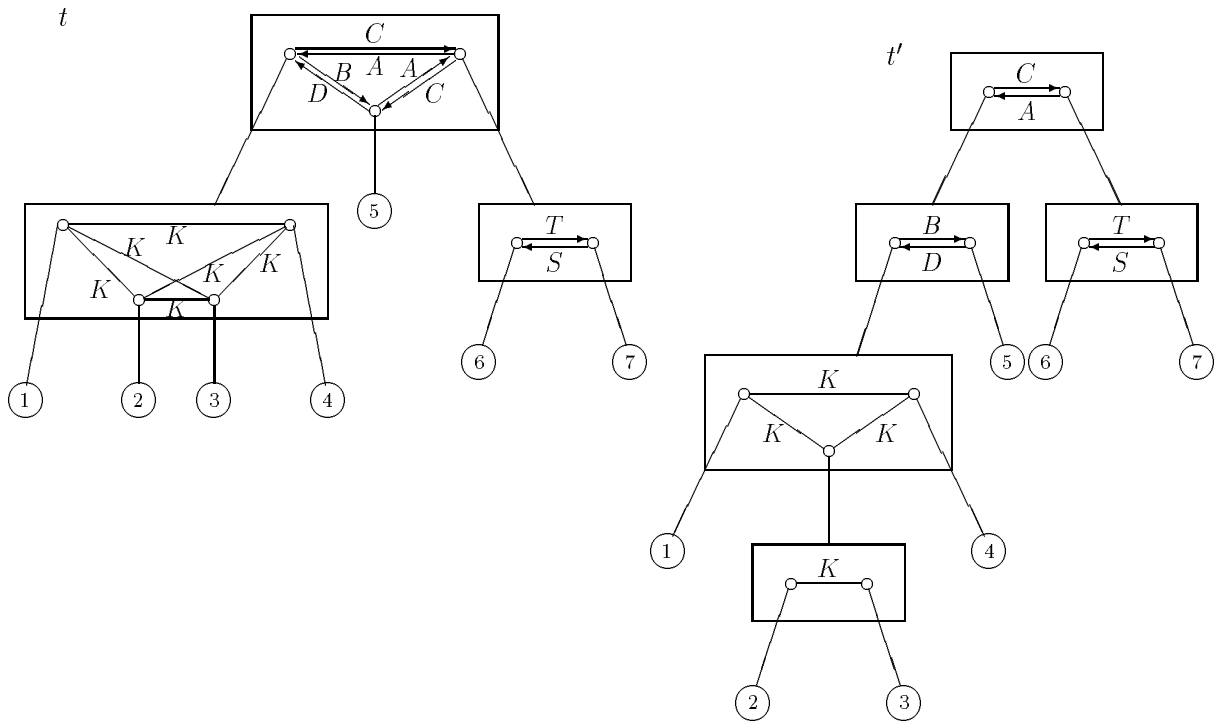


Figure 4.10: ℓ2s-labeled trees t and t'

t' is a refinement of t, e.g., the root of t is replaced by a tree with two inner nodes; the inner node v of t is replaced by an elementary subtree, and is hence unaltered. □

The following lemma has been proved (for chain-free ℓ2s-labeled trees) in [14, Th. 6.22]. We give here its proof following the above definition of refinement.

Lemma 4.1.19 *If t is a locally special $\ell 2s$-labeled tree representing a $\ell 2s$ g, then t is a refinement of $shape(g)$.*

Proof. Let t be a locally special ℓ2s-labeled tree representing a ℓ2s g. We may assume that t is chain-free, since each locally special ℓ2s-labeled tree with chains is a refinement of a chain-free locally special ℓ2s-labeled tree.

Now let t' be the ℓ2s-labeled tree that results from t by replacing for each equivalence class $\mathcal{E}$ of $\equiv_t^*$ the subtree $tree_t(\mathcal{E})$ by an elementary subtree with root $v_\mathcal{E}$, where $v_\mathcal{E}$ is labeled by the ℓ2s represented by $tree_t(\mathcal{E})$. More precisely, $t' = (D, T, \varphi)$, where

$D = \{v_\mathcal{E} \mid v_\mathcal{E} = root(tree_t(\mathcal{E}))$ for an equivalence class $\mathcal{E}$ of $\equiv_t^*\} \cup leaf(t)$,

$T = \{(v_\mathcal{E}, w) \mid w \in leaf(tree_t(\mathcal{E}))$ for an equivalence class $\mathcal{E}$ of $\equiv_t^*\}$,

and φ is such that for $v_\mathcal{E} \in D$, $\varphi(v_\mathcal{E}) = \ell 2s(tree_t(\mathcal{E}))$.

Since equivalence classes of $\equiv_t^*$ are disjoint, t' is well-defined; from the definition of t' it follows directly that t is a refinement of t'.

On the other hand, by the construction of t' and by Lemma 4.1.15, t' is chain-free, locally special, and disjoint – hence, by Proposition 4.1.14, t' is the shape of g. □

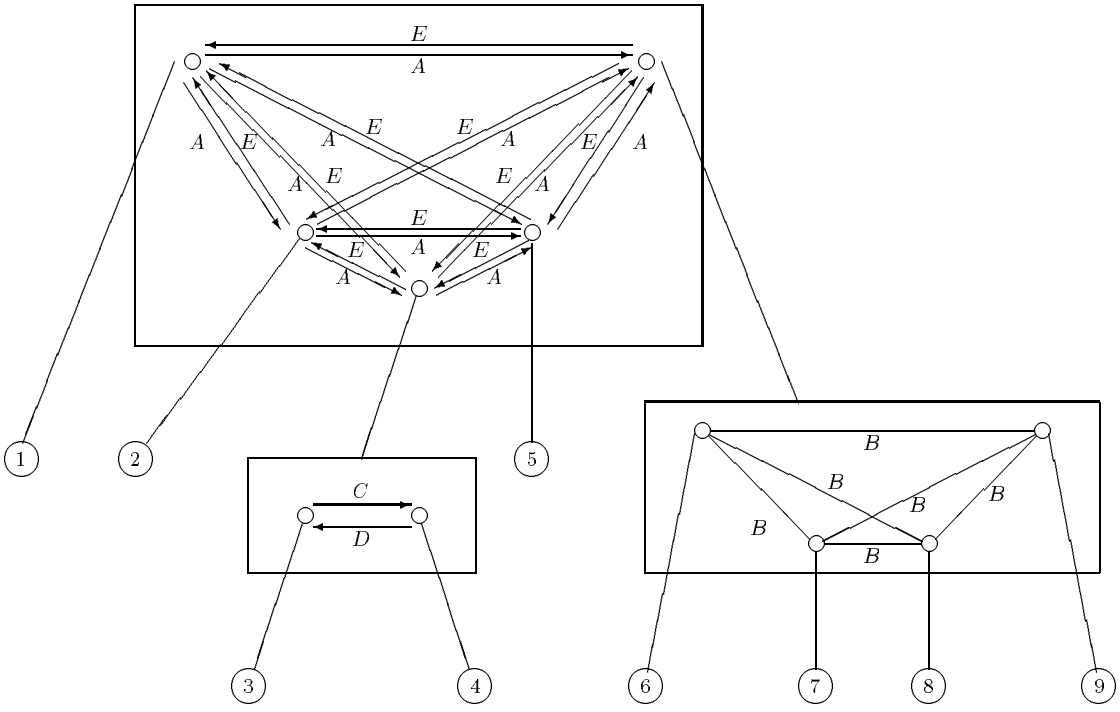


Figure 4.11: $shape(\ell 2s(t))$

Example 4.1.20 The $\ell 2$s-labeled tree t from Example 4.1.16 is a refinement of the shape of $\ell 2s(t)$, given in Figure 4.11. □

4.2 Bi-orders and texts

In this section we consider specifications of labeled T-structures and of T-functions (which are labeled T-structures with an additional valuation, or labeling, of the elements of its domain). In the first part of the section (subsection 4.2.1) we consider bi-orders, which are specifications of labeled T-structures, and in the second part (subsection 4.2.2) we consider texts which are specifications of T-functions.

4.2.1 Labeled T-structures and bi-orders

A natural subclass of (labeled) 2-structures is the class of (labeled) 2-structures that are antisymmetric and "angular". These so-called *T-structures* have been introduced in [15], and are further investigated in [16].

Since in this paper we consider labeled 2-structures, we will deal with T-structures that are labeled.

Definition 4.2.1 A *labeled T-structure* (abbreviated ℓTs) is a $\ell 2$s $g = (D, \delta, \Delta)$ such that g is antisymmetric, and for all $X \subseteq D$ with $\#X = 3$, there exists $x \in X$ such that $\delta(x, y) = \delta(x, z)$, where $X = \{x, y, z\}$.

Note that, by definition, a ℓTs has only antisymmetric features.

Proposition 4.2.2 (cf. [16, Th. 2.11]) *For each ℓTs g, if g has exactly n features $F_1, \ldots, F_n$, for $n \geq 1$, then for all $P_1, \ldots, P_n \subseteq \mathrm{part}(g)$ such that $P_j \in F_j$ for $j = 1, \ldots, n$, $\bigcup_{j=1}^{n} P_j$ is a linear order.*

In this way, a ℓTs can be seen as a generalization of a linear order. One may see a feature $F = \{P, P'\}$ as two "opposite orientations" for the involved 2-edges: one orientation given by P and the other one by P'. Then Proposition 4.2.2 says that *whichever* choice of orientations (arrows) is made for the 2-edges in an arbitrary ℓTs, the resulting directed graph will be a linear order.

It is shown in [16] that each ℓTs is equivalent to a ℓTs with two features in a well-defined sense. This allows one to consider ℓTs's with at most two features. It turns out that such a ℓTs has a representation by a pair of linear orders (we will use the term *bi-order* for an ordered pair of linear orders with a common domain).

Proposition 4.2.3 (cf. [16, Th. 3.6]) *Let g be a ℓTs with at most two features. Then there exists a bi-order $\sigma = (\rho_1, \rho_2)$ with domain $dom(g)$ such that $part(g) = \{\rho_1 \cap \rho_2, \rho_1 \cap rev(\rho_2), rev(\rho_1) \cap \rho_2, rev(\rho_1) \cap rev(\rho_2)\} - \{\emptyset\}$.*

By Proposition 4.2.3 a bi-order determines a ℓTs (with one or two features) modulo the labels of its partition classes. In order to obtain a more precise correspondence between bi-orders and ℓTs's we restrict our attention to ℓTs's with a specific labeling alphabet, namely $\{VH, V\overline{H}, \overline{V}H, \overline{VH}\}$, such that each feature is labeled either by $\{VH, \overline{VH}\}$ or by $\{V\overline{H}, \overline{V}H\}$. Such a ℓTs is called a *normal* T-structure (abbreviated *nTs*) and the alphabet $\{VH, V\overline{H}, \overline{V}H, \overline{VH}\}$ is denoted by Δ_{VH}. The intuition behind the choice of the letters V and H, corresponding with the first and the second order of a bi-order respectively, will be explained in Subsection 4.2.2 where the notion of a text is introduced.

From now on in this paper all ℓ2s's we consider will be normal T-structures.

A *nTs-labeled tree* is a ℓ2s-labeled tree such that each inner node is labeled by a normal T-structure. It is easily seen that the ℓ2s represented by a nTs-labeled tree is a nTs, and, since a substructure of a nTs is a nTs, a tree that represents a nTs is a nTs-labeled tree. Note that, by Proposition 4.1.8(1), the nodes of the shape of a nTs are either linear or primitive.

A bi-order determines a normal T-structure as follows.

Definition 4.2.4 Let $\sigma = (\rho_1, \rho_2)$ be a bi-order with domain D. The *normal T-structure determined by* σ, denoted by $nTs(\sigma)$, is the nTs $g = (D, \delta, \Delta_{VH})$ such that

$part(g) = \{\rho_1 \cap \rho_2, \rho_1 \cap rev(\rho_2), rev(\rho_1) \cap \rho_2, rev(\rho_1) \cap rev(\rho_2)\} - \{\emptyset\}$

and δ is such that

$\delta(\rho_1 \cap \rho_2) = VH$

$\delta(\rho_1 \cap rev(\rho_2)) = V\overline{H}$,

$\delta(rev(\rho_1) \cap \rho_2) = \overline{V}H$,

$\delta(rev(\rho_1) \cap rev(\rho_2)) = \overline{VH}$.

Example 4.2.5 For the bi-order $\sigma = (1,4,3,5,2,6),(2,5,4,3,1,6))$ $nTs(\sigma)$ is given in Figure 4.12. □

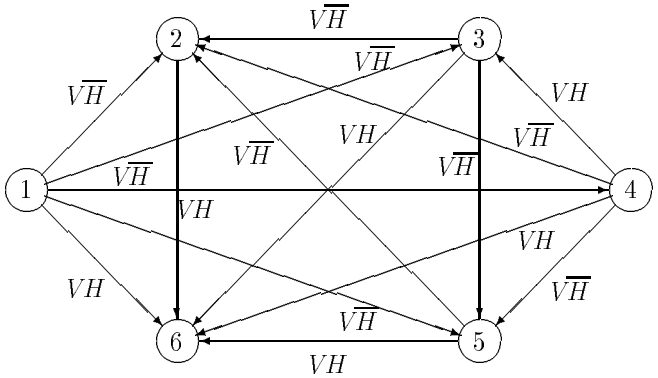


Figure 4.12: $nTs(\sigma)$

Here and in what follows we adopt the notational convention to omit in pictures of nTs's the partition classes labeled by $\overline{V}H$ and $\overline{VH}$.

By Proposition 4.2.2, for a nTs each union of two partition classes that are not in the same feature is a linear order. Moreover, it is easy to see that if $g = (D, \delta, \Delta_{VH})$ is a nTs, and $\sigma = (\rho_1, \rho_2)$ is the bi-order representing g, then $\bigcup\{P \in part(\mathrm{g}) \mid \delta(P) \in \{VH, \overline{VH}\}\}$ is exactly the linear order ρ_1, and $\bigcup\{P \in part(\mathrm{g}) \mid \delta(P) \in \{VH, \overline{VH}\}\}$ is the linear order ρ_2.

In this way, a nTs determines the bi-order by which it is represented.

Example 4.2.6 (Example 4.2.5 continued.) For the nTs of Figure 4.12 we have indeed that $\delta^{-1}(VH) \cup \delta^{-1}(\overline{VH}) = \{(1,2),(1,3),(1,4),(1,5),(1,6),(4,2),(4,3),(4,5),(4,6),(3,2),(3,5),(3,6),(5,2),(5,6),(2,6)\} = (1,4,3,5,2,6) = \rho_1$, and $\delta^{-1}(VH) \cup \delta^{-1}(\overline{VH}) = \{(2,1),(2,3),(2,4),(2,5),(2,6),(5,1),(5,3),(5,4),(5,6),(4,1),(4,3),(4,6),(3,1),(3,6),(1,6)\} = (2,5,4,3,1,6) = \rho_2$, where δ is the labeling function of $nTs(\sigma)$, and $\sigma = (\rho_1, \rho_2)$. □

Hence there is a one-to-one correspondence between bi-orders and normal T-structures. This correspondence is given by the bijective mapping nTs which assigns to each bi-order σ the nTs it determines (see Definition 4.2.4); for each nTs g, $nTs^{-1}(g)$ is the bi-order determining g.

The clans of a nTs can be found in a natural way in its specification by a bi-order.

Proposition 4.2.7 (cf. [16, Th. 3.10]) *Let $\sigma = (\rho_1, \rho_2)$ be a bi-order with domain D. A subset $X \subseteq D$ is a clan of $nTs(\sigma)$ iff $X \in seg(\rho_1) \cap seg(\rho_2)$.*

Following Proposition 4.2.7, the subsets of the domain of a bi-order that are segments in both linear orders are called *clans* of the bi-order. Also, other notions for nTs's can be transferred to bi-orders: a bi-order σ is *primitive* or *linear* iff $nTs(\sigma)$ is primitive or linear, respectively.

The most obvious comparison of the two linear orders of a bi-order leads to the following straightforward notions. A bi-order $\sigma = (\rho_1, \rho_2)$ is *sequential* if either $\rho_1 = \rho_2$ or $\rho_1 = rev(\rho_2)$. In the former case σ is called *forward* sequential, in the latter case σ is called *backward* sequential. The notion of sequentiality of a bi-order is directly related to the notion of linearity of the nTs denoted by a given bi-order.

Lemma 4.2.8 *A bi-order is linear iff it is sequential.*

Proof. Let $\sigma = (\rho_1, \rho_2)$ be a bi-order. If $nTs(\sigma)$ is linear, then it contains precisely one feature labeled either by $\{VH, \overline{VH}\}$ or by $\{\overline{VH}, \overline{VH}\}$. It follows that either $\rho_1 = \rho_2$ or $\rho_1 = rev(\rho_2)$, respectively. Conversely, if $\rho_1 = \rho_2$ or $\rho_1 = rev(\rho_2)$, then $part(nTs(\sigma)) = \{\rho_1, rev(\rho_1)\}$, and hence $nTs(\sigma)$ is linear. □

In the same way as nTs-labeled trees are hierarchical representations of nTs's, one can use trees with inner nodes labeled by bi-orders as hierarchical representations of bi-orders.

Definition 4.2.9 A *bi-ordered tree* is a 4-tuple (D, T, ν, ϑ), where (D, T) is a tree, and ν and ϑ are functions assigning linear orders to inner nodes of (D, T) such that for each inner node v, both $\nu(v)$ and $\vartheta(v)$ are linear orders on the direct descendants of v.

For a bi-ordered tree $t = (D, T, \nu, \vartheta)$, the labeling functions ν and ϑ are denoted by VO_t and HO_t, respectively. The linear orders on the nodes of t defined by VO_t and HO_t induce in the usual way linear orders on $leaf(t)$, denoted by $VO(t)$ and $HO(t)$ respectively. In other words, the local bi-orders of t induce a bi-order $(VO(t), HO(t))$ with domain $leaf(t)$, which will be denoted by $bo(t)$. This is similar to the way a ℓ2s-labeled tree defines a labeled 2-structure: a "local" ℓ2s which labels an inner node defines a label for each pair (u, w) of direct descendants of this node. This label is then inherited by each pair of leaves "below" the pair (u, w).

Accordingly, as for ℓ2s's, we say that a bi-ordered tree t *represents* a bi-order σ if $\sigma = bo(t)$.

Example 4.2.10 Consider the bi-ordered tree t in Figure 4.13. Here the direct descendants of a node are drawn from left to right according to the first order of the node, and the second order is given explicitly by arrows.

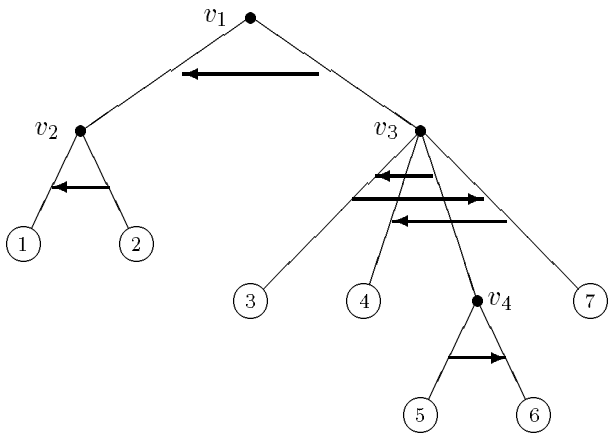


Figure 4.13: bi-ordered tree t

Hence $VO_t(v_1) = (v_2, v_3)$, $HO_t(v_1) = (v_3, v_2)$ $VO_t(v_2) = (1, 2)$, $HO_t(v_2) = (2, 1)$, $VO_t(v_3) = (3, 4, v_4, 7)$, $HO_t(v_3) = (v_4, 3, 7, 4)$, $VO_t(v_4) = (5, 6)$, $HO_t(v_4) = (5, 6)$.

This bi-ordered tree t represents the bi-order $\sigma = ((1, 2, 3, 4, 5, 6, 7), (5, 6, 3, 7, 4, 2, 1))$, i.e., $bo(t) = \sigma$.

To illustrate that the local second orders induce the second order of σ, we redraw the bi-ordered tree in Figure 4.14 in such a way that the left-to-right order of the direct descendants of a node represents the second order of the node. □

From now on we will draw all bi-ordered trees in such a way that for each inner node v the first order is the left-to-right order of direct descendants of v, and the second order is given explicitly by arrows. Using the correspondence between nTs's and bi-orders we obtain a correspondence between nTs-labeled trees and bi-ordered trees by "locally" replacing the nTs's labeling the inner nodes by equivalent bi-orders or the other way around.

More precisely, the bijective mapping nTs on bi-orders is extended to a mapping $\widehat{nTs}$ on bi-ordered trees, defined as follows. For each bi-ordered tree $t = (D, T, \nu, \vartheta)$, $\widehat{nTs}(t)$ is the

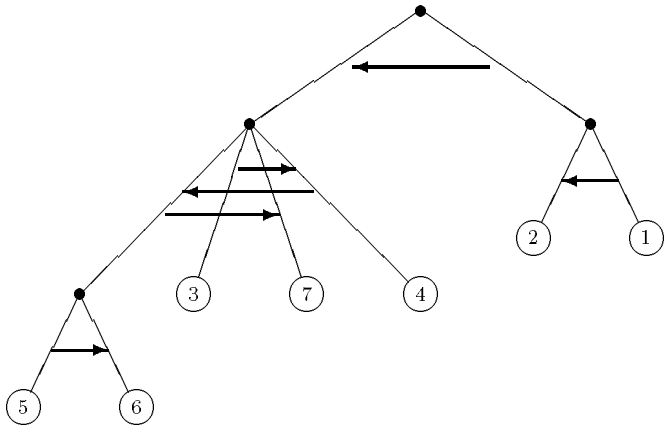


Figure 4.14: bi-ordered tree t redrawn

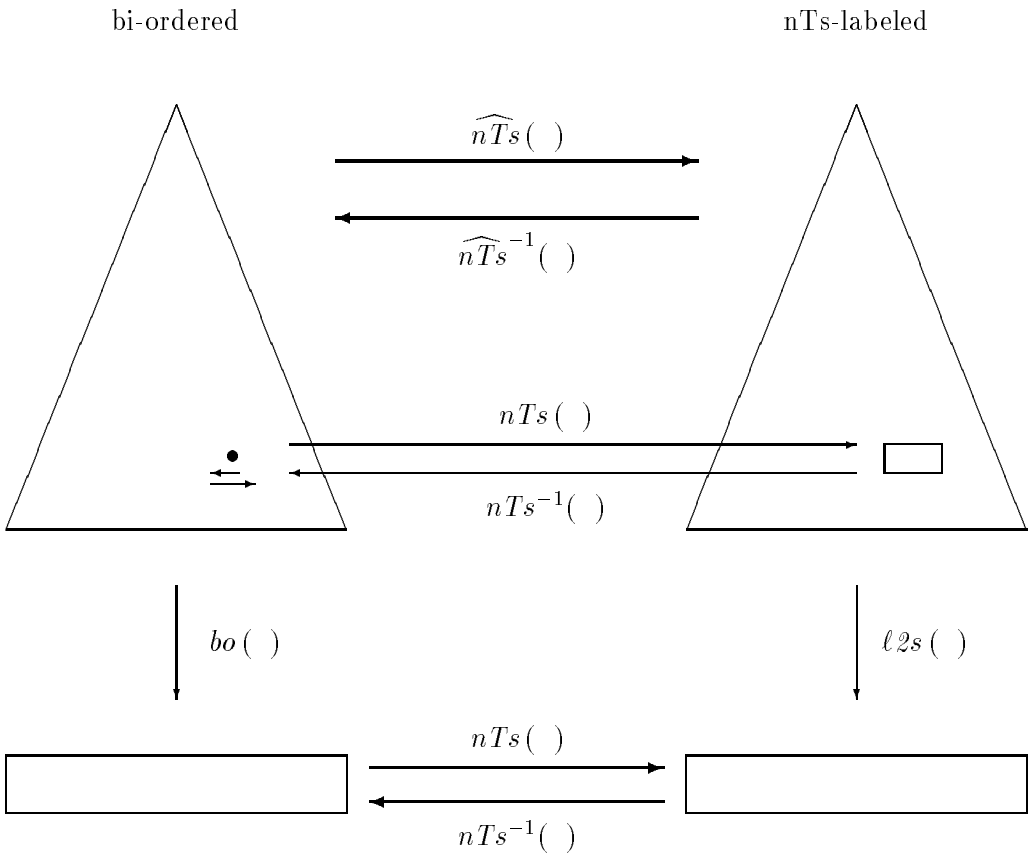


Figure 4.15: $\ell 2s(\widehat{nTs}(t)) = nTs(bo(t))$ and $nTs^{-1}(\ell 2s(t')) = bo(\widehat{nTs}^{-1}(t))$

nTs-labeled tree (D, T, η), where for each $v \in in(t)$, $\eta(v) = nTs(\nu(v), \vartheta(v))$. Clearly, this makes $\widehat{nTs}$ a bijection between bi-ordered trees and nTs-labeled trees.

A bi-ordered tree t represents a bi-order $bo(t)$, and a nTs-labeled tree t' represents a normal T-structure $\ell 2s(t')$. It is an immediate consequence of the constructions of $bo(t)$ and $\ell 2s(t')$ that these representations are compatible.

Lemma 4.2.11 *For each bi-ordered tree t, $\ell 2s(\widehat{nTs}(t)) = nTs(bo(t))$, and for each nTs-labeled tree t', $nTs^{-1}(\ell 2s(t')) = bo(\widehat{nTs}^{-1}(t))$.*

In other words the diagram given in Figure 4.15 commutes.

Example 4.2.12 The bi-ordered tree t from Example 4.2.10 corresponds to the nTs-labeled tree $\widehat{nTs}(t)$ in Figure 4.16.

Indeed, this nTs-labeled tree represents $nTs(\sigma)$, which is given in Figure 4.17. □

We define substitution for bi-orders in the obvious way: for bi-orders $\sigma = (\rho, \delta)$ and $\sigma' = (\rho', \delta')$ with disjoint domains, and for x in the domain of σ, the *substitution of σ' into σ at x*, denoted by $subst(\sigma, x, \sigma')$, is the bi-order $(\rho_{<x} + \rho' + \rho_{>x}, \delta_{<x} + \delta' + \delta_{>x})$. Note that

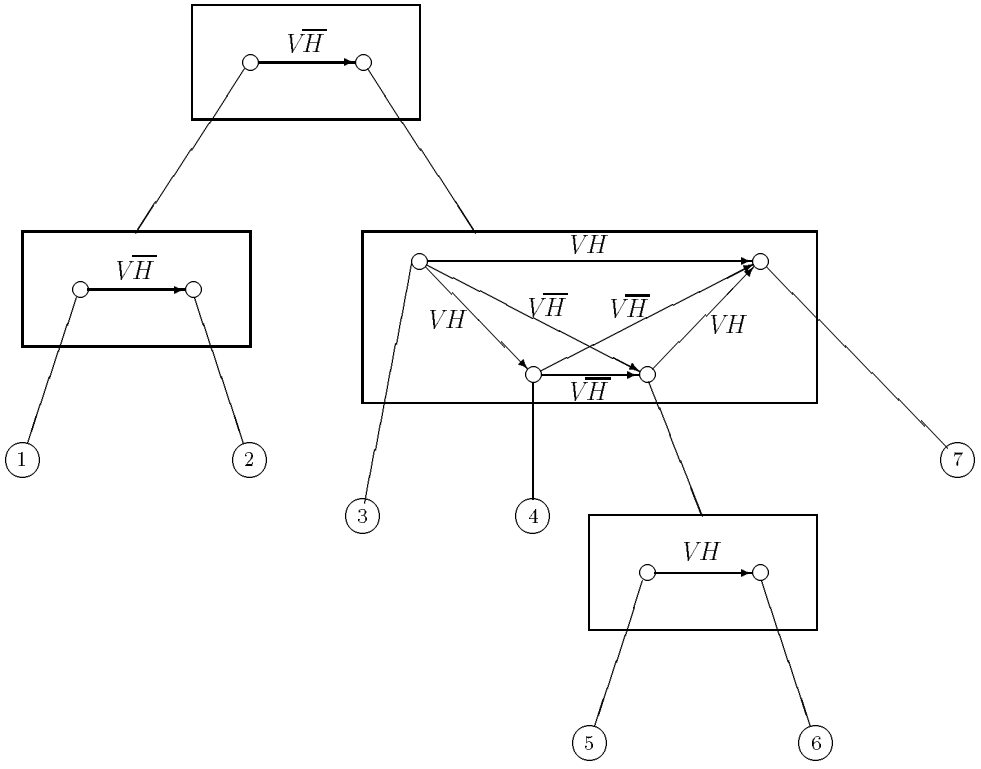


Figure 4.16: $\widehat{nTs}(t)$

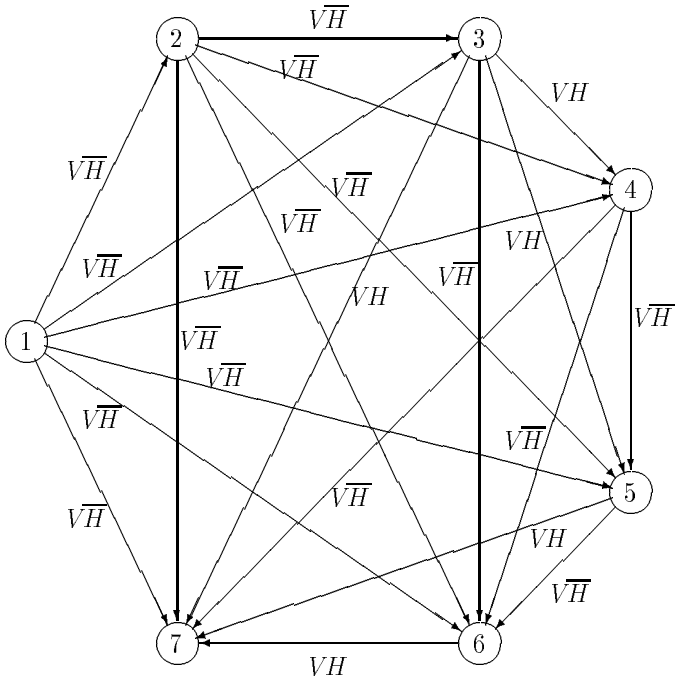


Figure 4.17: $nTs(\sigma)$

$subst(\sigma, x, \sigma')$ is the bi-order represented by the bi-ordered tree with two inner nodes such that the root is labeled by σ and the inner node x is labeled by σ'. This implies that substitution of bi-orders is compatible with that of nTs's (by Remark 4.1.10, this compatibility amounts to a special case of Lemma 4.2.11). However we still give a proof of Lemma 4.2.13 just to show explicitly the relationship between the nTs induced by the substitution and the substitution of nTs's.

Lemma 4.2.13 *Let σ and σ' be bi-orders with disjoint domains, and let x an element of the domain of σ. Then $nTs(subst(\sigma, x, \sigma')) = subst(nTs(\sigma), x, nTs(\sigma'))$.*

Proof. Let $nTs(subst(\sigma, x, \sigma')) = (D_1, \delta_1, \Delta_{VH})$, let $nTs(\sigma) = (D, \delta, \Delta_{VH})$, let $nTs(\sigma') = (D', \delta', \Delta_{VH})$, and let $subst(nTs(\sigma), x, nTs(\sigma')) = (D_2, \delta_2, \Delta_{VH})$. Clearly, $D_1 = D_2$. By Definition 4.2.4, and by the above definition of substitution for bi-orders, for $(y, z) \in E_2(D_1)$,

$$\delta_1(y,z) = \begin{cases} \delta(y,z) & \text{if } y,z \in D - \{x\} \\ \delta'(y,z) & \text{if } y,z \in D' \\ \delta(y,x) & \text{if } y \in D - \{x\},\ z \in D' \\ \delta(x,z) & \text{if } y \in D',\ z \in D - \{x\} \end{cases}$$

It follows immediately from the definition of substitution for ℓ2s's that $\delta_2 = \delta_1$. □

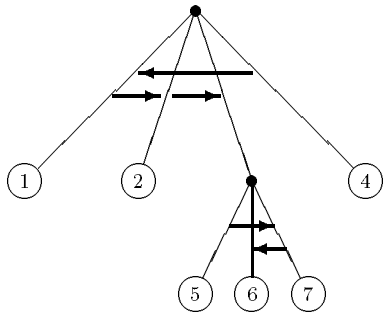


Figure 4.18: substitution of σ' into σ at 3

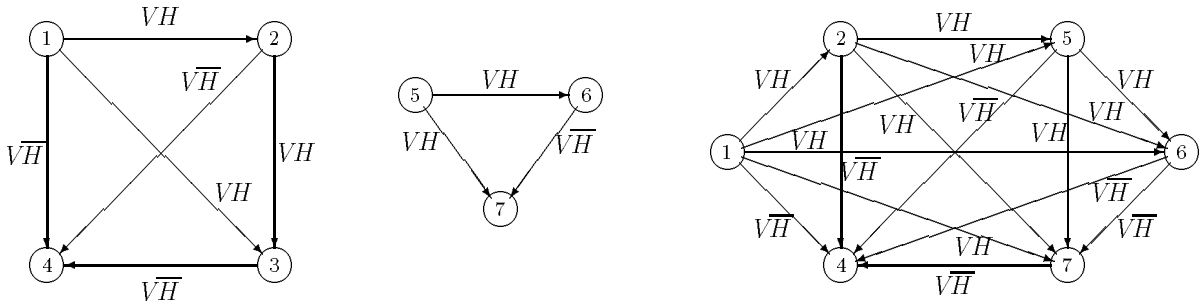


Figure 4.19: $nTs(\sigma)$, $nTs(\sigma')$, $nTs(subst(\sigma, 3, \sigma'))$

Example 4.2.14 Let σ be the bi-order $((1,2,3,4),(4,1,2,3))$, and let σ' be the bi-order $(5,6,7),(5,7,6))$. Then $subst(\sigma, 3, \sigma') = ((1,2,5,6,7,4),(4,1,2,5,7,6))$; it is represented by the bi-ordered tree with two inner nodes given in Figure 4.18.

$nTs(\sigma)$, $nTs(\sigma')$, and $nTs(subst(\sigma, 3, \sigma'))$ are given in Figure 4.19.

Hence indeed $nTs(subst(\sigma, 3, \sigma')) = subst(nTs(\sigma), 3, nTs(\sigma'))$. □

By Lemma 4.2.11, notions and results concerning the ℓ2s-labeled tree representation of nTs's carry over to the representation of bi-orders by bi-ordered trees. We sum up now some of the so transferred notions.

A bi-ordered tree t is *locally special* if for each $v \in in(t)$, $(VO_t(v), HO_t(v))$ is primitive or sequential; t is *locally sequential* if for each $v \in in(t)$, $(VO_t(v), HO_t(v))$ is sequential; t is *locally primitive* if for each $v \in in(t)$, $(VO_t(v), HO_t(v))$ is primitive. For each chain-free locally special bi-ordered tree t, one can define a relation $\equiv_t$ on $in(t)$ such that for all $u, v \in in(t)$, $u \equiv_t v$ if $u \in ddes_t(v)$ and either $VO_t(u) = HO_t(u)$ and $VO_t(v) = HO_t(v)$ or $VO_t(u) = rev(HO_t(u))$ and $VO_t(v) = rev(HO_t(v))$.

The shape of a bi-order σ is a bi-ordered tree representing σ that is chain-free, locally special, and "disjoint" (no nodes of t are in the relation $\equiv_t$).

A refinement of a bi-ordered tree t is a bi-ordered tree obtained from t by replacing each inner node v of t by a bi-ordered tree representing $(VO_t(v), HO_t(v))$. Each locally special

bi-ordered tree t representing a bi-order σ is a refinement of $shape(\sigma)$, where each inner node of the shape is replaced by a subtree corresponding to an equivalence class of $\equiv_t^*$.

4.2.2 T-functions and texts

A word over an alphabet Σ is usually defined as a sequence over Σ, i.e., a function on the ordered domain $(1, 2, \ldots, n)$, where n is the length of the word. Formally speaking, in this way a word is an ordered pair (λ, ρ), where ρ is a linear order, and λ is a function from $dom(\rho)$ into Σ.

We introduce now the notion of a *text* which in this spirit can be seen as a common generalization of a word and of a bi-order.

Definition 4.2.15 A *text* is a 3-tuple $\tau = (\lambda, \rho_1, \rho_2)$, where λ is a finite function, and ρ_1 and ρ_2 are linear orders on $dom(\lambda)$.

Hence a text is a function on a bi-ordered domain, or alternatively, a word with an additional order on its domain.

For a text $\tau = (\lambda, \rho_1, \rho_2)$, we use fun_τ, $VO(\tau)$, $HO(\tau)$ and $dom(\tau)$ to denote λ , ρ_1, ρ_2, and $dom(\lambda)$ respectively. In those abbreviations VO stands for "the visible order" and HO stands for "the hidden order". This corresponds to the basic intuition behind the notion of a text $(\lambda, \rho_1, \rho_2)$ which consists of the word (λ, ρ_1) – which is the visible (or surface) structure – and the syntactic structure (or deep structure) of this word given by ρ_2; ρ_2 is simply coding the shape of the text.

Let τ be a text. The *length of* τ, denoted by $|\tau|$, equals $\#dom(\tau)$. The *text-word of* τ, denoted $word(\tau)$, is the word $fun_\tau(VO(\tau))$, where fun_τ is extended to linear orders in the natural way. We say that τ is a *text over* Δ, where Δ is an alphabet, if $word(\tau) \in \Delta^+$.

If τ' is a text such that $dom(\tau') \subseteq dom(\tau)$, and $fun_{\tau'}$, $VO(\tau')$, and $HO(\tau')$ are the restrictions of fun_τ, $VO(\tau)$, and $HO(\tau)$ to $dom(\tau')$, then τ' is a *subtext of* τ.

In what follows, notions and results concerning texts may be obtained by viewing texts as generalizations of words.

Of course, notions and results concerning bi-orders can (and will) be translated to texts. E.g., the shape of a text $\tau = (\lambda, \rho_1, \rho_2)$, denoted by $shape(\tau)$, is the shape of the bi-order (ρ_1, ρ_2) where additionally the leaves are labeled by the function λ.

Through the correspondence between bi-orders and nTs's, a text corresponds to a nTs the nodes of which are additionally given values by λ, hence they are labeled by λ. This leads us to the notion of a T-function.

Definition 4.2.16
(1) A *T-function* is a pair (λ, g), where λ is a function, and g is a ℓTs such that $dom(g) = dom(\lambda)$.
(2) For a text τ, the *T-function of* τ is the T-function (λ, g) where $\lambda = fun_\tau$ and $g = nTs((VO(\tau), HO(\tau)))$.

Often in considering texts, we are dealing with the underlying nTs of the T-function of a text rather than with the T-function itself; we call this nTs the structure of the text, i.e., for

a text τ, the *structure of* τ is the nTs $nTs((VO(\tau), HO(\tau))$. Through this connection we can transfer back and forth the notions and notations concerning texts and nTs's. In particular, also in this way we can define the shape of τ (viz. the shape of the structure of τ together with a labeling of its leaves).

Example 4.2.17 Let $\tau = (\lambda, (1,6,3,4,2,5), (2,4,6,3,5,1))$, where $\lambda(1,6,3,4,2,5) = aabaab$. Then $shape(\tau)$ is as in Figure 4.20.
The T-function of τ is given in Figure 4.21. □

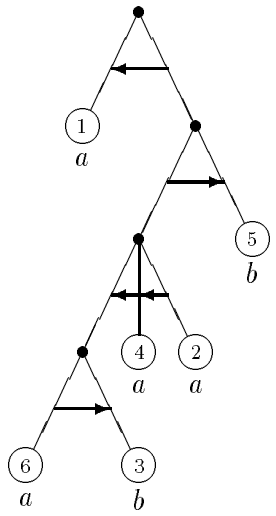


Figure 4.20: $shape(\tau)$

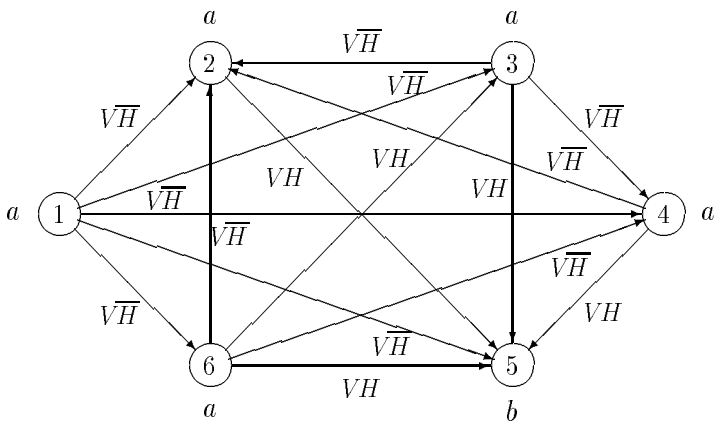


Figure 4.21: the T-function of $\tau = (\lambda, (1,6,3,4,2,5), (2,4,6,3,5,1))$

Recall that the nodes of the shape of a text are sequential or primitive. If the shape contains only sequential nodes (i.e., it is locally sequential), then, since the shape is disjoint, on each

path in the shape the nodes are alternately forward and backward sequential. Accordingly, such texts, i.e., texts of which the shape is locally sequential, are called *alternating*.

A *singleton text* is a text τ with $|\tau| = 1$. Hence (see Preliminaries) for a singleton text τ, $VO(\tau) = HO(\tau) = (x)$, where $dom(\tau) = \{x\}$.

Note that a text is primitive *and* sequential iff it is either a singleton text or a text of length 2. To distinguish primitive texts that are not sequential from texts of length 1 or 2, we refer to them as *strictly primitive* texts.

Two texts τ and τ' are *isomorphic* if there is a bijection $\kappa : dom(\tau) \to dom(\tau')$ such that $\tau' = \kappa(\tau)$, i.e., $fun_{\tau'} = fun_\tau \circ \kappa^{-1}$, $VO(\tau') = \kappa(VO(\tau))$, and $HO(\tau') = \kappa(HO(\tau))$, where κ is extended to linear orders. Hence, two texts are isomorphic iff they have the same text-word and their structures are isomorphic.

A *standard text* is a text τ such that $VO(\tau) = (1, \ldots, |\tau|)$. Clearly, each text is isomorphic with a standard text.

In this paper we consider *abstract* texts, that is, isomorphism classes of texts. Hence, texts are considered to be equal if they are isomorphic. In most cases we will use standard texts as concrete representatives of abstract texts. Accordingly, we sometimes specify an abstract text as an ordered pair (w, μ), where w is a word of length n and μ is a linear order on $\{1, \ldots, n\}$; (w, μ) denotes the isomorphism class of the standard text τ with $word(\tau) = w$, $VO(\tau) = (1, \ldots, n)$, and $HO(\tau) = \mu$.

An abstract singleton text τ is uniquely determined by its single letter; the singleton text determined by the letter A is denoted by $st(A)$.

Recall that bi-ordered trees that differ only in the identity of the inner nodes are identified. Hence, for a bi-ordered tree representing a text τ, the inner nodes are defined up to isomorphism while the leaves are the elements of $dom(\tau)$. For a bi-ordered tree representing an *abstract* text also the leaves are considered modulo isomorphism. Substitution of bi-orders carries over to substitution of texts in the obvious way. Formally, for texts $\tau = (\lambda, \rho, \delta)$ and $\tau' = (\lambda', \rho', \delta')$ with disjoint domains, and for $x \in dom(\tau)$, the *substitution of τ' into τ at x*, denoted by $subst(\tau, x, \tau')$, is the text $(\lambda|_{dom(\tau)-\{x\}} \cup \lambda', \rho_{<x} + \rho' + \rho_{>x}, \delta_{<x} + \delta' + \delta_{>x})$. Note that this is also generalizing the "substitution" of a word w' into a word w at occurrence x.

Since we consider texts modulo isomorphism, this definition can be extended to texts that are not disjoint, by just taking disjoint isomorphic copies.

Example 4.2.18 Let $\tau_1 = (\lambda_1, (1,2,3,4,5,6), (1,3,4,2,5,6))$, where $\lambda_1(1) = \lambda_1(4) = \lambda_1(5) = a$, $\lambda_1(2) = \lambda_1(6) = b$, and $\lambda_1(3) = c$, and let $\tau_2 = (\lambda_2, (1,2,3,4), (4,1,3,2))$, where $\lambda_2(1) = \lambda_2(2) = c$, and $\lambda_2(3) = \lambda_2(4) = b$. We construct $subst(\tau_1, 4, \tau_2)$ by taking an isomorphic copy of τ_1 with domain $\{1,2,3,x,8,9\}$, and an isomorphic copy of τ_2 with domain $\{4,5,6,7\}$. Then for $\tau = subst(\tau_1', x, \tau_2')$, we have that $VO(\tau) = (1,2,3,\underline{4},\underline{5},\underline{6},\underline{7},8,9)$, $HO(\tau) = (1,3,\underline{7},\underline{4},\underline{6},\underline{5},2,8,9)$, and fun_τ is such that $word(\tau) = abcccbbab$.

If we construct $subst(\tau_2, 2, \tau_1)$ in a similar way, then we obtain the standard text τ' such that $VO(\tau') = (1,\underline{2},\underline{3},\underline{4},\underline{5},\underline{6},\underline{7},8,9)$, $HO(\tau') = (9,1,8,\underline{2},\underline{4},\underline{5},\underline{3},\underline{6},\underline{7})$, and $fun_{\tau'}$ is such that $word(\tau') = cabcaabbb$. □

By transforming texts into their T-functions, one might consider substitution of texts as substitution of T-functions (which comes from the substitution of nTs's in the obvious way). The compatibility of these two ways of substituting is guaranteed by Lemma 4.2.13.

We would like to conclude this section by pointing out that the theory of texts can be seen as a theory of permutations on labeled domains. Since each text is isomorphic with a standard text, we may assume that for a text $\tau = (\lambda, \rho_1, \rho_2)$ of length n we have $\rho_1 = (1, \ldots, n)$ and consequently ρ_2 may be seen as a permutation on $\{1, \ldots, n\}$; this point of view is taken in [11].

4.3 Context-free text grammars

In this section, we will introduce grammars that generate texts in a context-free way: this is done by generalizing context-free string grammars.

Definition 4.3.1 A *context-free text grammar*, abbreviated *cft grammar*, is a 4-tuple (Θ, Δ, Π, S), where Θ and Δ are alphabets such that $\Delta \subseteq \Theta$, Π is a finite set of ordered pairs of the form (A, τ), where $A \in \Theta - \Delta$ and τ is a text over Θ, and S is a singleton text over $\Theta - \Delta$.

Let $G = (\Theta, \Delta, \Pi, S)$ be a cft grammar.

An element (A, τ) of Π is called a *text-production*; in the sequel we write $A \to \tau$ instead of (A, τ). The letters of Δ are called *terminal* letters, the letters of $\Theta - \Delta$ are called *nonterminal*, S is called the *axiom of* G, and Π is denoted by $prod(G)$; $maxrh(G)$ denotes $\max\{\, |\tau| \mid A \to \tau \in \Pi \text{ for some } A \in \Theta - \Delta\}$.

Let τ and τ' be texts over Θ. τ *directly derives* τ' *(in* G*)*, denoted $\tau \Rightarrow_G \tau'$, if there is a production $A \to \sigma \in \Pi$ and $x \in dom(\tau)$ with $fun_\tau(x) = A$ such that $\tau' = subst(\tau, x, \sigma)$. The reflexive and transitive closure of $\Rightarrow_G$ is denoted by $\Rightarrow_G^*$; if $\tau \Rightarrow_G^* \tau'$, then we say that τ *derives* τ' *(in* G*)*. We write $\Rightarrow$ and $\Rightarrow^*$ rather than $\Rightarrow_G$ and $\Rightarrow_G^*$ whenever the grammar G is clear from the context. For a text τ over Θ, a *derivation of* τ is a sequence of texts $(\tau_0, \tau_1, \ldots, \tau_n)$ such that $\tau_0 = S$, $\tau_n = \tau$, and $\tau_i \Rightarrow \tau_{i+1}$ for $i = 0, \ldots, n-1$. A derivation of a text τ is *successful* if τ is a text over Δ.

For a cft grammar $G = (\Theta, \Delta, \Pi, S)$, $\mathrm{TxL}(G)$ denotes the *text language generated by* G, i.e., $\mathrm{TxL}(G) = \{\tau \text{ over } \Delta \mid S \Rightarrow_G^* \tau\}$. In general, by a *text language (over* Δ*)* we mean a family of abstract texts (over Δ), where Δ is some alphabet. A text language K is a *context-free text language* (abbreviated *cft language*) if there exists a cft grammar G such that $K = \mathrm{TxL}(G)$.

If we "project" texts onto their text-words, we obtain the corresponding notions for string languages. For a text-production $\pi = A \to \tau$, the corresponding string-production $A \to word(\tau)$ is denoted by $word(\pi)$. The cf (string) grammar $(\Theta, \Delta, word(\Pi), word(S))$, where $word(\Pi) = \{word(\pi) \mid \pi \in \Pi\}$, is denoted by $word(G)$. The *string language generated by* G, denoted by $\mathrm{WL}(G)$, is the language $L(word(G))$, which is the same as $word(\mathrm{TxL}(G)) = \{word(\tau) \mid \tau \in \mathrm{TxL}(G)\}$.

Example 4.3.2
(1) Let $G = (\Theta, \Delta, \Pi, st(S))$ be the cft grammar such that $\Theta = \{S, A, B\}$, $\Delta = \{a, b\}$, and Π consists of the productions $S \to (AB, (1,2))$, $A \to (aA, (2,1))$, $A \to (a, (1))$, $B \to (bBb, (3,2,1))$, $B \to (b, (1))$.

Then $\tau = (aaabbb, (3,2,1,6,5,4)) \in \mathrm{TxL}(G)$, and a derivation of τ in G is $(st(S), (AB,(1,2)), (aAB,(2,1,3)), (aaAB,(3,2,1,4)), (aaaB,(3,2,1,4)), (aaabBb,(3,2,1,6,5,4)), (aaabbb,(3,2,1,6,5,4)))$. E.g., $(aAB,(2,1,3)) \Rightarrow_G (aaAB,(3,2,1,4))$ by application of the production $A \to (aA,(2,1))$, since $(aaAB,(3,2,1,4)) = subst((aAB,(2,1,3)),2,(aA,(2,1)))$.

It is easy to verify that

$$\mathrm{TxL}(G) = \{(a^m b^{2n+1}, (m, m-1, \ldots, 1, m+2n+1, m+2n, \ldots, m+1)) \mid m \geq 1, n \geq 0\}.$$

(2) Let $G = (\{S, A, a, b, c\}, \{a, b, c\}, \Pi, st(S))$ be the cft grammar such that Π consists of the following productions: $S \to (ASc, (1,2,3))$, $S \to (ab, (2,1))$, $A \to (ab, (2,1))$.

Then $\mathrm{TxL}(G) = \{((ab)^{n+1}c^n, \mu_n) \mid n \geq 0\}$, where $\mu_0 = (2,1)$, and for $n \geq 1$, $\mu_n = (2,1,4,3,\ldots,2n+2,2n+1,2n+3,\ldots,3n+2)$. □

In the case of a cf *string* grammar, we associate with a derivation of a word its derivation tree, which is a node-labeled ordered tree yielding the derived word. For texts, we will use a node-labeled *bi-ordered* tree as the derivation tree of a context-free derivation, where this bi-ordered tree "yields" (i.e., represents) the generated text. We give now a formal description of such a derivation tree.

Let t be a node-labeled bi-ordered tree, i.e., $t = (D, T, \nu, \vartheta, \eta)$, where (D, T, ν, ϑ) is a bi-ordered tree, and η is a function on $nd(t)$, which is denoted by lb_t. For each node $v \in in(t)$, the *text-production associated to* v, denoted by $prod_t(v)$, is the text production $lb_t(v) \to (\lambda_t, VO_t(v), HO_t(v))$, where λ_t is the labeling function lb_t of t restricted to the direct descendants of v. We use $prod(t)$ to denote the set $\{prod_t(v) \mid v \in in(t)\}$.

Let $G = (\Theta, \Delta, \Pi, S)$ be a cft grammar.

For a text τ over Θ, a node-labeled bi-ordered tree t is a *derivation tree of* τ *(in* G*)* if $lb_t \mid_{leaf(t)} = fun_\tau$, $VO(t) = VO(\tau)$, $HO(t) = HO(\tau)$, $prod(t) \subseteq \Pi$, and $lb_t(root(t)) = word(S)$. A derivation tree of τ *(in* G*)* is *successful* if τ is a text over Δ.

Note that a derivation tree is defined here as a concrete tree for a concrete text. However, isomorphic (as to the identity of nodes) derivation trees yield isomorphic texts. Consequently, from now on we will not distinguish between derivation trees that only differ in the identity of their nodes.

Now with each derivation D in G of a text τ we can associate a derivation tree t of τ in G by the usual inductive construction.

If $D = (S)$, then t is the node-labeled tree with one node v, and $lb_t(v) = word(S)$.

Let $D = (S = \tau_0, \tau_1, \ldots, \tau_n = \tau)$, for $n \geq 1$, and let t' be the derivation tree of τ_{n-1} associated with the derivation $(S = \tau_0, \tau_1, \ldots, \tau_{n-1})$. Let $\pi = A \to \tau'$ be the production that is applied in the last step of D, i.e., $\tau = subst(\tau_{n-1}, x, \tau')$, where $x \in dom(\tau_{n-1})$ is such that $fun_{\tau_{n-1}}(x) = A$. Now t is constructed from t' as follows: add the nodes in $dom(\tau')$ as direct descendants of x, and define $lb_t = lb_{t'} \cup fun_{\tau'}$, $VO_t|_{in(t')} = VO_{t'}$, $VO_t(x) = VO(\tau')$, $HO_t|_{in(t')} = HO_{t'}$, $HO_t(x) = HO(\tau')$. Then t is a derivation tree of τ (note that $prod_t(x) = \pi$).

Example 4.3.3 Consider the cft grammar $G = (\{S, A, B, a, b, c\}, \{a, b, c\}, \Pi, st(S))$, where Π consists of the productions $S \to (AaBb, (2,4,1,3))$, $A \to (Aa, (1,2))$, $A \to (ac, (2,1))$, $B \to (bBc, (3,1,2))$, $B \to (b, (1))$.

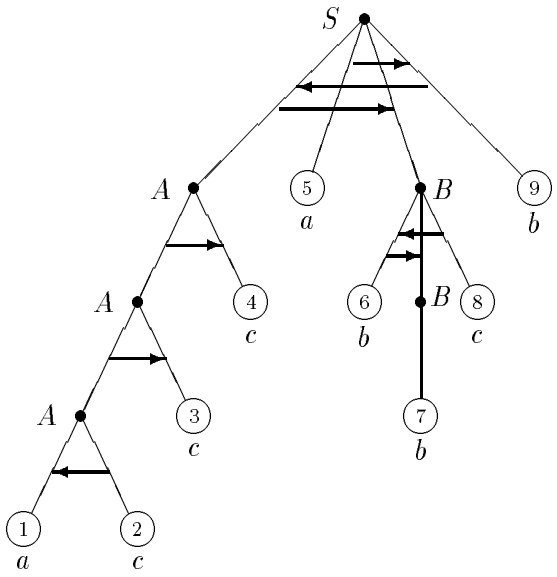


Figure 4.22: derivation tree of τ

Consider the following successful derivation in G: $(st(S) = (S,(1)),(AaBb,(2,4,1,3))$, $(AaaBb,(3,5,1,2,4))$, $(AaabBcb,(3,7,1,2,6,4,5))$, $(AaaabBcb,(4,8,1,2,3,7,5,6))$, $(Aaaabbcb,(4,8,1,2,3,7,5,6))$, $(acaaabbcb,(5,9,2,1,3,4,8,6,7)) = \tau$).

The derivation tree of τ that corresponds with this derivation is given in Figure 4.22. □

Given a derivation tree t of τ, we can associate with t a derivation D of τ as follows: if $(v_1,\ldots,v_n)$ is the order of $in(t)$ induced by a preorder traversal of t, then $D = (\tau_0,\tau_1,\ldots,\tau_n)$ where $\tau_0 = S$, and $\tau_i \Rightarrow \tau_{i+1}$ by application of $prod_t(v_{i+1})$. This way of associating a derivation with a derivation tree is based on a particular (viz. the preorder) traversal of nodes of a tree. Clearly, taking any other fixed traversal of nodes will give a different way of associating a derivation with a derivation tree.

Example 4.3.4 Let G be the cft grammar from Example 4.3.3. Consider the derivation tree t in G in Figure 4.23.

The derivation associated with t as above is $(st(S),(AaBb,(2,4,1,3)),(AaaBb,(3,5,1,2,4))$, $(acaaBb,(4,6,2,1,3,5)),(acaabBcb,(4,8,2,1,3,7,5,6)),(acaabbBccb,(4,10,2,1,3,9,5,8,6,7))$, $(acaabbbccb,(4,10,2,1,3,9,5,8,6,7)))$.

Also the following derivation corresponds to the same derivation tree t:
$(st(S),(AaBb,(2,4,1,3)),(AabBcb,(2,6,1,5,3,4)),(AaabBcb,(3,7,1,2,6,4,5)),(AaabbBccb,$ $(3,9,1,2,8,4,7,5,6)),(Aaabbbccb,(3,9,1,2,8,4,7,5,6)),(acaabbbccb,(4,10,2,1,3,9,5,8,6,7)))$. □

Let t be a derivation tree of a text τ in some cft grammar G. The leaf-labeled bi-ordered tree that results from t by deleting the nonterminal labels (i.e. labels assigned by lb_t) from the inner nodes is denoted by $di(t)$. The following theorem is an immediate consequence of the definition of a derivation tree.

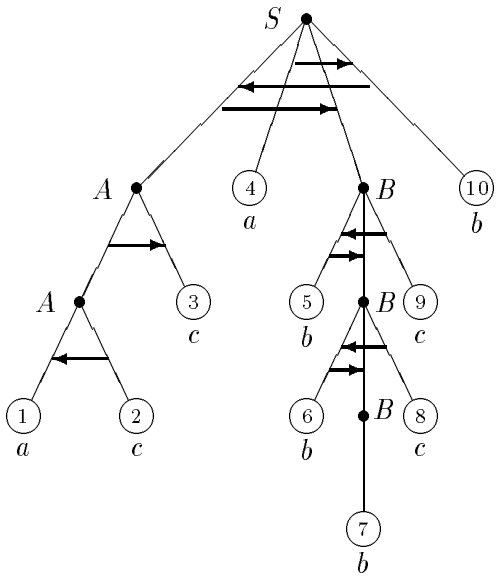


Figure 4.23: derivation tree t

Theorem 4.3.5 *For each derivation tree of a text τ, $di(t)$ represents τ.*

By Theorem 4.3.5, the diagram in Figure 4.24 commutes.

The following corollary of Theorem 4.3.5 is a consequence of the fact that the contributions of a bi-ordered tree are clans of the bi-order it represents (cf. Lemma 4.1.6).

Corollary 4.3.6 *Let t be a derivation tree of a text τ. For each $v \in in(t)$, $contr_t(v)$ is a clan of τ.*

Example 4.3.7 Consider the grammar G from Example 4.3.3 and the derivation tree of τ in Figure 4.23. $\{1,2,3\}$ and $\{5,6,7,8,9\}$ are clans of τ. □

Example 4.3.8 Let $G = (\{S,a,b\},\{a,b\},\Pi,st(S))$ be the cft grammar such that Π consists of the productions $S \to (SS,(1,2))$, $S \to (SS,(2,1))$, $S \to (a,(1))$, $S \to (b,(1))$. Then $\mathrm{TxL}(G) = \{\tau \mid word(\tau) \in \{a,b\}^*, \tau \text{ alternating }\}$. This is seen as follows. By Theorem 4.3.5, G generates texts with a locally sequential hierarchical representation. It follows that $\mathrm{TxL}(G) \subseteq \{\tau \mid word(\tau) \in \{a,b\}^*, \tau \text{ alternating }\}$. Now consider an alternating text τ with $word(\tau) \in \{a,b\}^*$. It is possible (see also Section 4.5, Example 4.5.1) to refine the shape of τ into a locally sequential bi-ordered tree representing τ such that each inner node has at most two direct descendants. By labeling each inner node with S, we obtain a derivation tree of τ in G. This proves that $\mathrm{TxL}(G) \supseteq \{\tau \mid word(\tau) \in \{a,b\}^*, \tau \text{ alternating }\}$. □

Remark 4.3.9 Turning from texts to T-functions, we obtain from a cft grammar a kind of graph grammar (see, e.g., [17]), which generates T-functions of texts. One step in the derivation of a T-function in such a grammar is given by the substitution of T-functions. Clearly, by the compatibility of the substitution operations, see Lemma 4.2.13, a text is generated by a cft grammar iff its T-function is generated by the corresponding graph grammar. Thus from the

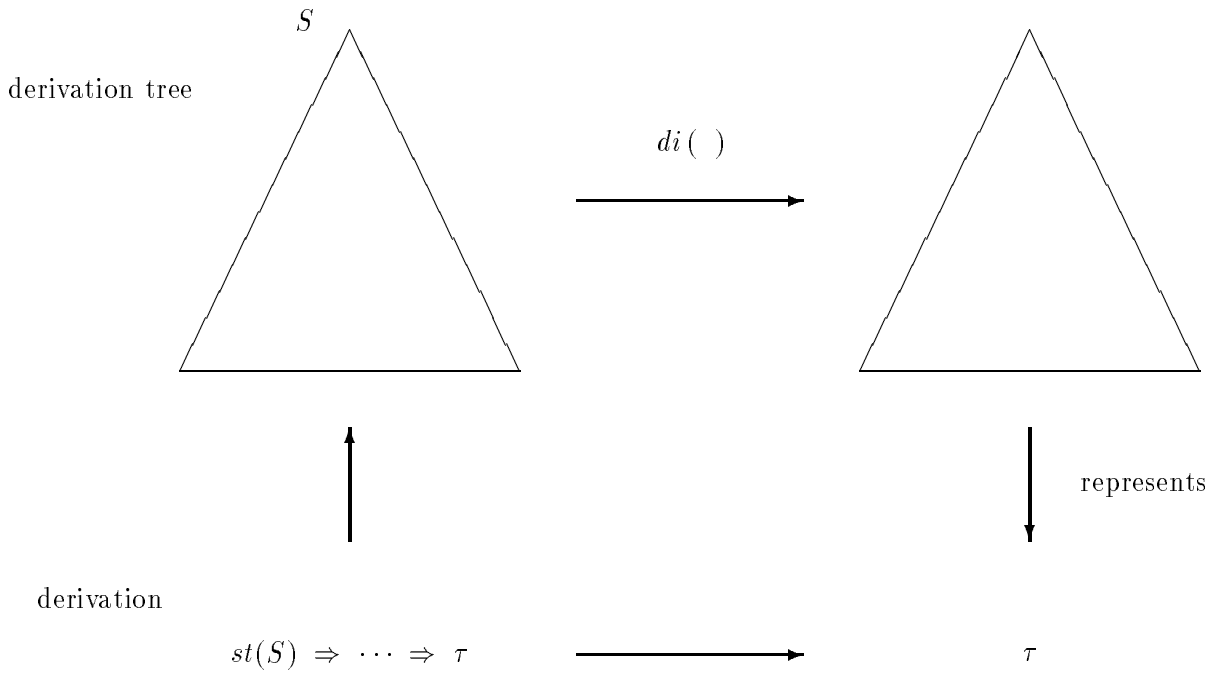


Figure 4.24: $di(t)$ represents τ

point of view of graph grammars we get here a rather interesting situation. Since T-functions (which are node-labeled oriented graphs of a certain kind) have a "linear notation" in the form of a text one can reduce the problem of the generation of T-functions to the problem of the generation of texts. More specifically, the rewriting of texts by context-free productions is equivalent to the rewriting of T-functions by NLC-like grammars where the right-hand sides of productions are also T-functions and the connection relation is total. Clearly, the above comments only indicate the connection between context-free text grammars and graph grammars; this relationship certainly deserves an in-depth investigation. □

Let G_1 and G_2 be cft grammars. G_1 and G_2 are *equivalent* if they generate the same text language; they are 1-*equivalent* if they generate the same text language modulo singletons, i.e., if $\{\tau \in \mathrm{TxL}(G_1) \mid |\tau| > 1\} = \{\tau \in \mathrm{TxL}(G_2) \mid |\tau| > 1\}$.

4.4 Traditional normal forms

In this section we investigate the existence of a number of "traditional" (i.e., traditionally considered in formal language theory, cf. [28]) normal forms for cft grammars, such as chain-

free, Chomsky, and Greibach normal form. We prove that chain-freeness is a normal form for cft grammars, while Chomsky and Greibach form of grammars is not a normal form for cft grammars. We would like to stress here that our proofs of the non-existence of the Chomsky and Greibach normal form are based on the properties of shapes of texts.

Let $G = (\Theta, \Delta, \Pi, S)$ be a cft grammar. A production $A \to \tau \in \Pi$ with τ a singleton text is called a *chain-production*. A cft grammar that has no chain-productions is called *chain-free*.

Lemma 4.4.1 *Each cft grammar has a 1-equivalent chain-free grammar.*

Proof. The proof is a standard construction. Let $G = (\Theta, \Delta, \Pi, S)$ be a cft grammar. Define for each $A \in \Theta - \Delta$, $Z_A = \{C \in \Theta - \Delta \mid st(A) \Rightarrow^* st(C)\}$, and $\Pi_A = \{A \to (w, \mu) \mid C \to (w, \mu) \in \Pi, C \in Z_A, |w| \geq 2\}$. Let G' be the grammar $(\Theta, \Delta, \Pi', S)$, where $\Pi' = \bigcup_{A \in \Theta - \Delta} \Pi_A$. Clearly, G' is equivalent to G, and G' has no chain-productions of the form $A \to st(B)$ with $A, B \in \Theta - \Delta$.

Now define for each $A \in \Theta$, $Y_A = \{a \in \Delta \mid A \to st(a) \in \Pi'\}$. Let φ be the substitution on Θ^* such that for each $A \in \Theta$, $\varphi(A) = Y_A \cup \{A\}$.

Let G'' be the grammar $(\Theta, \Delta, \Pi'', S)$, where $\Pi'' = \{C \to (v, \mu) \mid C \to (w, \mu) \in \Pi', |w| > 2, v \in \varphi(w)\}$. Then G'' is 1-equivalent to G' (and hence to G), and G'' is chain-free. □

Let K be a text language. A text τ is a *subtext of* K if it is a subtext of an element of K. The set of primitive subtexts of K is denoted by $prims(K)$. The set of sequential subtexts of K is denoted by $seqs(K)$; by Lemma 4.2.8, $seqs(K)$ is precisely the subclass of those subtexts of K that are linear.

Lemma 4.4.2 *Let $G = (\Theta, \Delta, \Pi, S)$ be a cft grammar. Then for each $\tau \in prims(\mathrm{TxL}(G))$, $|\tau| \leq maxrh(G)$.*

Proof. Let $\tau' \in \mathrm{TxL}(G)$ and let τ be a primitive subtext of τ'. Let t be a derivation tree of τ' in G. By Theorem 4.3.5, $di(t)$ represents τ'. Hence, by Proposition 4.1.11, and Lemma 4.2.11, there exists $v \in in(t)$ such that the structure of τ, i.e., $nTs(VO(\tau), HO(\tau))$, is isomorphic to a substructure of $nTs(VO_t(v), HO_t(v))$. It follows that $|\tau| \leq \#ddes_t(v)$.

Hence for each $\tau \in prims(\mathrm{TxL}(G))$, $|\tau| \leq maxrh(G)$. □

As an immediate consequence of Lemma 4.4.2 we have the following result.

Theorem 4.4.3 *For each cft language K, $prims(K)$ is finite.*

Note that Theorem 4.4.3 is sharp: for any finite set F of primitive texts there exists a finite cft language K such that $F \subseteq prims(K)$.

For sequential subtexts we do not have a result similar to Theorem 4.4.3. This is related to the fact that Proposition 4.1.11 does not hold for linear substructures (see Example 4.1.13). We prove now that *any* text language with finitely many sequential subtexts is finite. For this we use the Theorem of Ramsey (see, e.g., [22]) in the following form:

Proposition 4.4.4 *(Ramsey) Let $k, n_0 \in \mathbf{N}_+$. There exists a constant $r(k, n_0) \in \mathbf{N}_+$ such that if D is a set with $\#D \geq r(k, n_0)$ and $\{P_1, \ldots, P_k\}$ is a partition of $S_2(D)$, then there exists $X \subseteq D$ such that $\#X \geq n_0$ and $S_2(X) \subseteq P_i$ for some $1 \leq i \leq k$.*

Theorem 4.4.5 *Let K be an arbitrary text language. If $seqs(K)$ is finite, then K is finite.*

Proof. Assume to the contrary that there exists a text language K such that $seqs(K)$ is finite and K is infinite. Let $M_0 = \max\{|\tau| \mid \tau \in seqs(K)\}$. Let $\tau \in K$ be such that $|\tau| \geq r(2, M_0)$.

We divide $S_2(dom(\tau))$ in P_1 and P_2 as follows. $P_1 = \{\{x, y\} \in S_2(dom(\tau)) \mid (x, y) \in VO(\tau)$ iff $(x, y) \in HO(\tau)\}$, $P_2 = \{\{x, y\} \in S_2(dom(\tau)) \mid (x, y) \in VO(\tau)$ iff $(x, y) \in rev(HO(\tau))\}$.

Hence, $\{x, y\} \in P_1$ iff (x, y) is labeled VH or $\overline{VH}$ in the structure of τ, and $\{x, y\} \in P_2$ iff (x, y) is labeled $V\overline{H}$ or $\overline{V}H$ in the structure of τ.

By the Theorem of Ramsey, there exists $X \subseteq dom(\tau)$ such that $\#X \geq M_0$, and either $S_2(X) \subseteq P_1$ or $S_2(X) \subseteq P_2$.

Consider now $\tau|_X$. Then either $VO(\tau|_X) = HO(\tau|_X)$, if $S_2(X) \subseteq P_1$, or $VO(\tau|_X) = rev(HO(\tau|_X))$, if $S_2(X) \subseteq P_2$. Hence $\tau|_X$ is sequential, while $\#X \geq M_0$; a contradiction. □

A cft grammar G is *in Chomsky (Greibach) form* if the corresponding cf string grammar $word(G)$ is in Chomsky (Greibach) normal form. Neither Chomsky form nor Greibach form are normal forms for cft grammars, as we will show now.

Theorem 4.4.6 *There exists a cft language K such that no cft grammar generating K is in Chomsky form.*

Proof. Let G be a cft grammar in Chomsky form. By Lemma 4.4.2, for each $\tau \in prims(\mathrm{TxL}(G))$, $|\tau| \leq maxrh(G) \leq 2$. But obviously there are cft languages that contain strictly primitive texts. Hence these languages can not be generated by a cft grammar in Chomsky form. □

Example 4.4.7 Let G be the following cft grammar: $G = (\{A, a\}, \{a\}, \{A \to (A^5, (2, 5, 3, 1, 4)), A \to (a, (1))\}, st(A)\}$. Then, for every context-free text grammar G' equivalent with G, $maxrh(G') \geq 5$. Hence there is no cft grammar in Chomsky form that generates $\mathrm{TxL}(G)$. □

Lemma 4.4.8 *If G is a cft grammar in Chomsky form, then $\mathrm{TxL}(G)$ consists of alternating texts.*

Proof. Let G be a cft grammar in Chomsky form. Let $\tau \in \mathrm{TxL}(G)$, and let t be a derivation tree of τ in G. Then t is a locally sequential bi-ordered tree representing τ. By Lemma 4.1.19, t is a refinement of $shape(\tau)$. Hence $shape(\tau)$ is locally sequential, and consequently, τ is alternating. □

Example 4.4.9
(1) The cft grammar G from Example 4.3.7(2), which generates the language of all alternating texts over $\{a, b\}$, is in Chomsky form.
(2) Let $G = (\{S, A, a, b\}, \{a, b\}, \Pi, st(S))$, where $\Pi = \{S \to (AS, (1, 2)), S \to (AS, (2, 1)), S \to (b, (1)), A \to (a, (1))\}$. Then, e.g., $\tau_0 = (aaaab, (1, 2, 5, 4, 3)) \in \mathrm{TxL}(G)$. The node-labeled bi-ordered tree of Figure 4.25 is a derivation tree of τ_0 in G.

The shape of τ_0 is given in Figure 4.26.

As a matter of fact, for $\tau \in \mathrm{TxL}(G)$, $shape(\tau)$ is as in Figure 4.27. □

Lemma 4.4.10 *Let G be a cft grammar in Greibach form. There exists a constant C_0 such that for each $\tau \in \mathrm{TxL}(G)$ there is a partition P of $dom(\tau)$ into clans of τ with $\{VO(\tau)(1)\} \in P$ and $\#P \leq C_0$.*

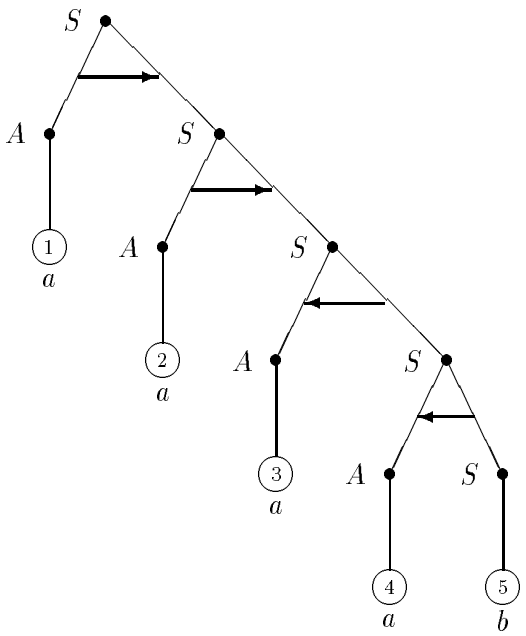


Figure 4.25: derivation tree of τ_0

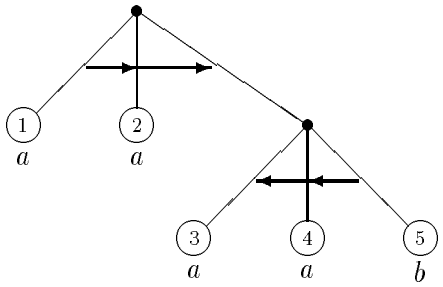


Figure 4.26: $shape(\tau_0)$

Proof. Take $C_0 = maxrh(G)$. Let $\tau \in K$, and let t be a derivation tree of τ. Let $v_1, \ldots, v_s$ be the direct descendants of the root of t, and let $X_i = contr_t(v_i)$ for $i = 1, \ldots, s$. By Corollary 4.3.6, $X_1, \ldots, X_s$ is a partition of $dom(\tau)$ into clans of τ, and, clearly, $s \leq C_0$. Furthermore, since G is in Greibach form, $X_j = \{VO(\tau)(1)\}$ where v_j is the first node in $VO_t(root(t))$. □

Theorem 4.4.11 *There exists a cft language K such that no cft grammar generating K is in Greibach form.*

Proof. Consider the cft grammar G, with $G = (\{A, B, a, b\}, \{a, b\}, \Pi, st(A))$, with $\Pi = \{A \to (Ba, (1,2)), B \to (Aa, (2,1)), A \to (bb, (1,2)), B \to (bb, (2,1))\}$. We will show that K is not generated by a grammar in Greibach form by proving that there is not a C_0 as in the statement of Lemma 4.4.10.

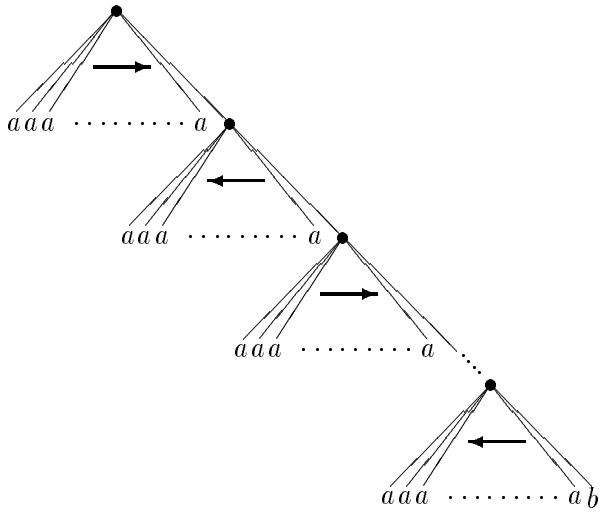


Figure 4.27: $shape(\tau)$

Assume to the contrary that there exists C_0 meeting the conditions of Lemma 4.4.10. Let $\tau \in \mathrm{TxL}(G)$ be such that $|\tau| > C_0$ (note that such a τ exists), and let t be a derivation tree of τ in G. As usual, we assume that τ is standard. It should be clear that $di(t)$ is chain-free, locally special, and disjoint, which implies that $di(t) = shape(\tau)$.

E.g., the derivation tree of $(bbaaa, (4,2,1,3,5)) \in \mathrm{TxL}(G)$ is given in Figure 4.28.

By the form of t, it follows that the non-trivial prime clans of τ are $\{\{1,\ldots,i\} \mid i = 2,\ldots,|\tau|\}$.

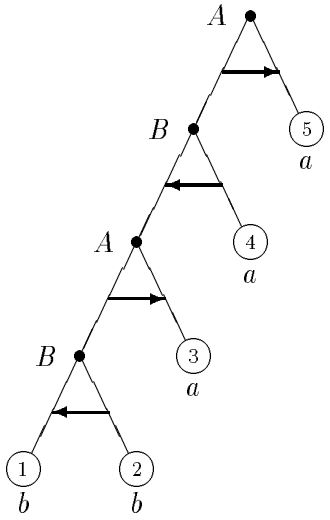


Figure 4.28: derivation tree of $(bbaaa, (4,2,1,3,5))$

Now let P be a partition of $dom(\tau)$ into clans of τ such that $\{1\} \in P$. Let Z be a clan in P, $Z \neq \{1\}$. Let $j = \min(Z)$. Then Z and the prime clan $\{1, \ldots, j\}$ are overlapping unless $Z = \{j\}$. Hence $P = \{\{x\} \mid x \in dom(\tau)\}$. But then $\#P > C_0$, which contradicts the assumption.

Hence there is no cft grammar in Greibach form that generates $\mathrm{TxL}(G)$. □

4.5 The primitive normal form

It is quite natural to consider primitive bi-orders as the basic building blocks for bi-orders. Given a shape of a bi-order σ, all its nodes are either primitive or sequential – if we refine the sequential nodes by using only sequential nodes with two direct descendants, then we obtain a representation (hence a "construction tree") for σ with primitive nodes only. In this section we prove that such representation trees (consisting of primitive nodes only) are grammatical in the sense that each cft language can be generated by a cft grammar where each production $A \to \tau$ is such that τ is primitive. This allows us to prove the following result (see Theorem 4.5.6) which is the converse of Lemma 4.4.8: each context-free text language that consists of alternating texts is generated by a grammar in Chomsky form.

To start with, we consider a restricted kind of refinement of bi-ordered trees. Let t be a bi-ordered tree, and consider a refinement t' of t. Then t' is a *primitive* refinement of t if t' is a locally primitive bi-ordered tree. Thus, by Lemma 4.1.19, each locally primitive bi-ordered tree representing a bi-order σ is a primitive refinement of $shape(\sigma)$.

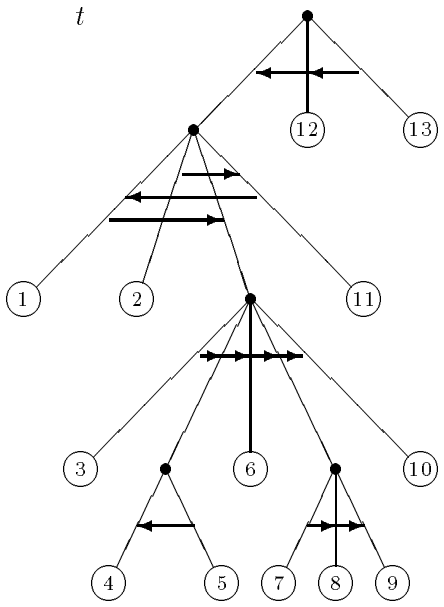


Figure 4.29: locally special bi-ordered tree t

For each locally special bi-ordered tree t it is possible to construct a primitive refinement of t by replacing each sequential node with more than two children, i.e., each sequential node that is not primitive, by a tree consisting of primitive nodes.

Example 4.5.1 Consider the locally special bi-ordered tree t in Figure 4.29.
Then t' and t'' in Figure 4.30 are both primitive refinements of t. □

Now we are ready to prove our normal form result.

Theorem 4.5.2 *For each cft language K there exists a cft grammar G such that $K = \mathrm{TxL}(G)$, and for each $A \to \tau \in prod(G)$, τ is primitive.*

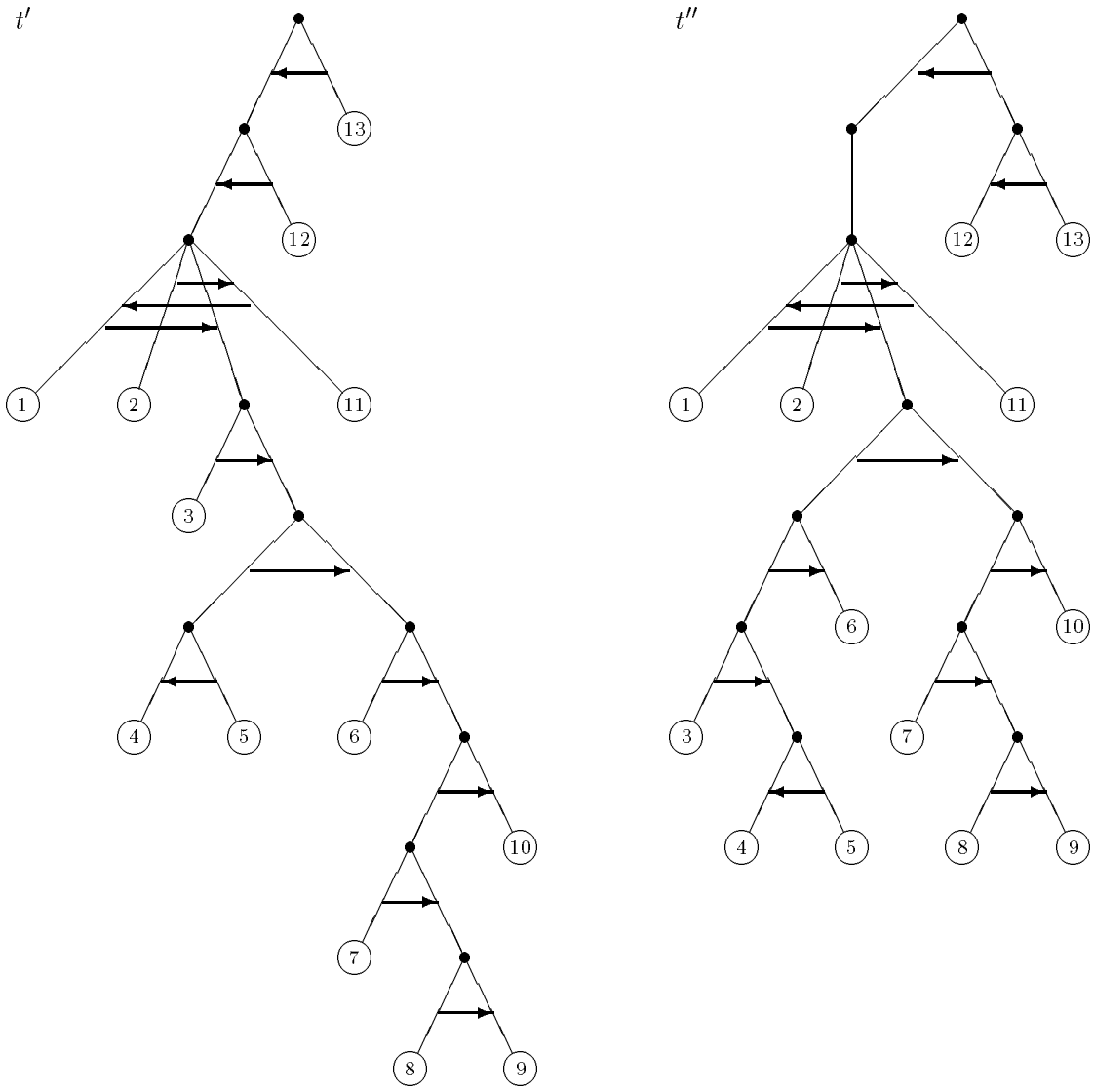


Figure 4.30: primitive refinements t' and t''

Proof. Let K be a cft language, and let $H = (\Theta, \Delta, \Pi, S)$ be a cft grammar generating K. Let Π_0 be the set of chain-productions in Π. Note that these productions already have the desired form.

Let $\pi = A \to \tau$ be a production in $\Pi - \Pi_0$. Let d_π be a primitive refinement of $shape(\tau)$. It can be made into a node-labeled bi-ordered tree, denoted by t_π, by labeling the inner nodes of d_π as follows: for each $v \in in(d_\pi)$, $v \neq root(d_\pi)$, $lb_{t_\pi}(v) = (\pi, v)$, and $lb_{t_\pi}(root(d_\pi)) = A$.

Consider the cft grammar $G = (\Theta', \Delta, \Pi', S)$ with $\Pi' = \{prod(t_\pi) \mid \pi \in \Pi - \Pi_0\} \cup \Pi_0$, and $\Theta' = \Theta \cup \{(\pi, v) \mid \pi \in \Pi - \Pi_0, v \in in(t_\pi)\}$.

Note that if $\pi \neq \pi'$, then the sets of "new" nonterminals in t_π and in $t_{\pi'}$ are disjoint. It follows directly from our construction that for each $A \to \tau$ in Π', τ is primitive.

Clearly, for each successful derivation of a text τ in H there exists a successful derivation of τ in G, and vice versa.

Hence $\mathrm{TxL}(G) = K$, and G has the desired form. □

Hence the grammars satisfying the property from Theorem 4.5.2 are a normal form for cft languages, denoted by *primitive normal form* (*PNF*).

Example 4.5.3 Consider the cft grammar $G = (\Theta, \Delta, \Pi, st(S))$, where $\Delta = \{a, b, c\}$, $\Theta = \Delta \cup \{S, A\}$, and Π consists of the productions $S \to (ASSbc, (2,4,1,3,5))$, $S \to (A, (1))$, $A \to (aAb, (1,3,2))$, $A \to (a, (1))$.

G is not in primitive normal form, since e.g., $\tau = (aAb, (1,3,2))$ is not primitive ($\{2,3\} \in seg(VO(\tau) \cap seg(HO(\tau))$, and $A \to \tau \in \Pi$.

As in Theorem 4.5.2, we construct $G' = (\Theta', \Delta, \Pi', st(S))$. Let $\pi_1 = (S \to (ASSbc, (2,4,1,3,5)))$, and let $\pi_2 = (A \to (aAb, (1,3,2)))$. Then t_{π_1} and t_{π_2} are as in Figure 4.31.

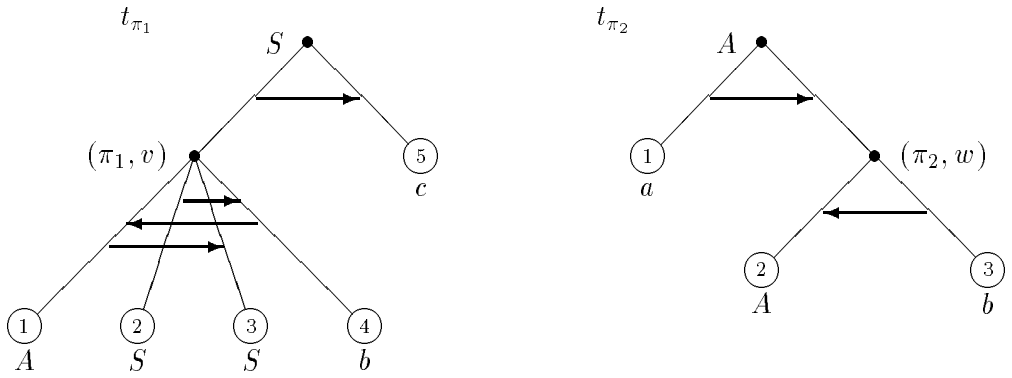


Figure 4.31: t_{π_1} and t_{π_2}

Now $\Theta' = \Theta \cup \{(\pi_1, v), (\pi_2, w)\}$, and Π' contains the productions $S \to ((\pi_1, v)c, (1,2))$, $(\pi_1, v) \to (ASSb, (2,4,1,3))$, $S \to (A, (1))$, $A \to (a(\pi_2, w), (1,2))$, $(\pi_2, w) \to (Ab, (2,1))$, $A \to (a, (1))$.

Clearly, G' is equivalent with G and in primitive normal form. □

Note that for a cft grammar in PNF, if a 1-equivalent chain-free cft grammar is constructed as in Lemma 4.4.1, then this chain-free cft grammar again is in PNF. Also, if we carry out only the first part of this construction, we obtain an equivalent grammar in PNF in which the only chain-productions are of the form $A \to st(a)$ where a is a terminal. Using this fact, it is easily verified that the primitive normal form can be strengthened by additionally requiring the following "Chomsky-like" restriction on the form of the productions.

Corollary 4.5.4 *For each cft language K over Δ there exists a cft grammar $G = (\Theta, \Delta, \Pi, S)$ in PNF such that $K = \mathrm{TxL}(G)$ and for each production $A \to \tau \in \Pi$, either $|\tau| > 1$ and $word(\tau) \in (\Theta - \Delta)^*$, or $|\tau| = 1$ and $word(\tau) \in \Delta$.*

Example 4.5.5 (Example 5.2 continued) Let G' be the cft grammar in PNF from Example 4.5.3. We can change G' into an equivalent "Chomsky-like" cft grammar $G'' = (\Theta \cup \{(\pi_1, v), (\pi_2, w), B, C\}, \Delta, \Pi'', st(S))$, where Π'' contains the productions $S \to ((\pi_1, v)C, (1,2))$, $(\pi_1, v) \to (ASSB, (2,4,1,3))$, $S \to (A(\pi_2, w), (1,2))$, $S \to (a, (1))$, $A \to (A(\pi_2, w), (1,2))$, $(\pi_2, w) \to (AB, (2,1))$, $A \to (a, (1))$, $B \to (b, (1))$, $C \to (c, (1))$. □

Using Corollary 4.5.4, we can characterize those cft languages that are generated by cft grammars in Chomsky form.

Theorem 4.5.6 *Let K be a cft language. K is generated by a cft grammar in Chomsky form iff K consists of alternating texts.*

Proof. By Lemma 4.4.8, the only-if part of the claim holds.

Suppose that K is a cft language consisting of alternating texts. Let $G = (\Theta, \Delta, \Pi, S)$ be a cft grammar in the form as in the statement of Corollary 4.5.4 generating K. We may assume that for each production π in Π, there is a successful derivation in G using π. We claim that G is in Chomsky form.

Assume to the contrary that G has a production of the form $A \to \tau$ with $|\tau| > 2$. Let $\tau' \in K$ be such that this production is applied in a derivation of τ' in G. Hence the derivation tree corresponding with this derivation contains a primitive node which is not sequential. But then, by Theorem 4.3.5 and Lemma 4.1.19, the shape of τ' is not locally sequential, which contradicts the fact that τ' is alternating.

Hence the productions in Π are of the form $A \to \tau$ with either $|\tau| = 1$ and $word(\tau) \in \Delta$ or $|\tau| = 2$ and $word(\tau) \in \Theta^+$; in other words G is in Chomsky form. □

4.6 Shapely cft grammars

A cft grammar G generates a set of texts and it provides each text τ that it generates with a derivation tree t, and hence with a representation $di(t)$. However, independently of G, τ has its own unique representation, viz. $shape(\tau)$. Hence naturally one considers G "faithful" (in its generation of $\mathrm{TxL}(G)$) if whenever G generates a text τ with a derivation tree t, then $di(t)$ is the shape of τ; such a faithful grammar G is called *shapely*. In this section we investigate the existence of shapely grammars.

Definition 4.6.1 Let G be a cft grammar. G is *shapely* if for each derivation tree t of a text τ in G, $di(t)$ is the shape of τ.

Example 4.6.2
(1) Consider the cft grammar $G_1 = (\{S, a, b\}, \{a, b\}, \Pi_1, st(S))$, where Π_1 consists of the productions $S \to (aSb, (1,2,3))$ and $S \to (ab, (1,2))$. Then $\mathrm{TxL}(G_1) = \{\tau \mid word(\tau) = a^n b^n, n \geq 1$, and $VO(\tau) = HO(\tau) = \{(a^n b^n, (1,2,\ldots,2n)) \mid n \geq 1\}$.

Hence the shape of a text τ in $\mathrm{TxL}(G_1)$ is as in Figure 4.32.

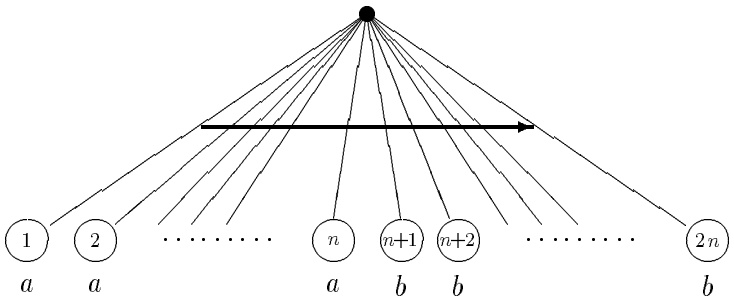


Figure 4.32: $shape(\tau)$

Clearly, G_1 is not shapely.
(2) Consider the cft grammar $G_2 = (\{S, a, b\}, \{a, b\}, \Pi_2, st(S))$, where Π_2 consists of the productions $S \to (aSb, (1,3,2))$ and $S \to (ab, (1,2))$. Note that $\mathrm{WL}(G_2) = \{a^n b^n \mid n \geq 1\} = \mathrm{WL}(G_1)$.

The text $\tau = (a^3 b^3, (1,6,2,5,3,4))$ is in $\mathrm{TxL}(G_2)$, and the tree t of Figure 4.33 is a derivation tree of τ in G_2.

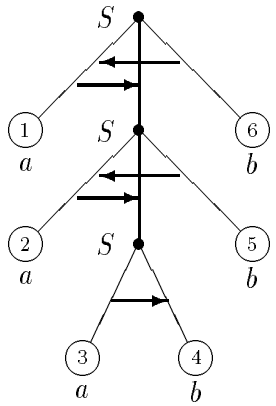


Figure 4.33: derivation tree t

The shape of τ is given in Figure 4.34.

Hence G_2 is not shapely. In fact, $(aSb, (1,3,2))$ is neither sequential nor primitive, and hence for each derivation tree t of a text $\tau \in \mathrm{TxL}(G)$ with $|\tau| > 2$, $di(t) \neq shape(\tau)$.

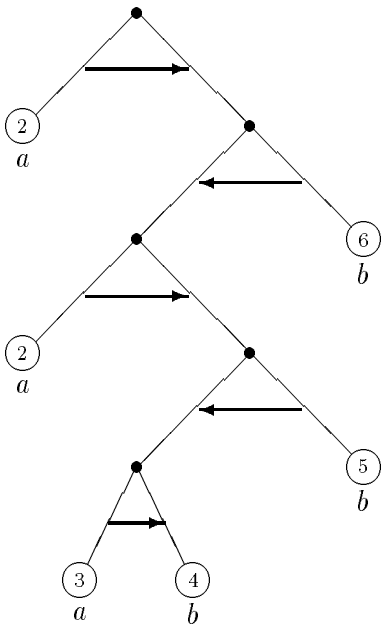


Figure 4.34: $shape(\tau)$

(3) Consider the cft grammar $G_3 = (\{S, T, a, b\}, \{a, b\}, \Pi_3, st(S))$, where Π_3 consists of the productions $S \to (aT, (1,2))$, $T \to (Sb, (2,1))$, and $S \to (ab, (1,2))$. Note that, since $S \Rightarrow^*_{G_3} (aSb, (1,3,2))$, G_3 is equivalent to G_2. Each derivation tree of a text $\tau \in \mathrm{TxL}(G_3)$ is chain-free, locally special, and disjoint. E.g., the derivation tree of $(a^3b^3, (1,6,2,5,3,4))$ is given in Figure 4.35.

Hence G_3 is shapely. □

Definition 4.6.3 Let K be a text language. K is *limited* if there exists $C_0 \in \mathbf{N}$ such that, for each $\tau \in K$, *out-degree*$(shape(\tau)) < C_0$.

The requirement that a text language is limited is a restriction: by Proposition 4.1.11, if K is a limited text language, then $prims(K)$ is finite. Also, there are context-free text languages that are not limited. E.g., the language $\mathrm{TxL}(G_1)$ from Example 4.6.2 is not limited. It was argued in Example 4.6.2 that the grammar G_1 is not shapely. As a matter of fact, the following result holds.

Theorem 4.6.4 *For each cft language K, K is limited iff there is a shapely cft grammar H such that $K = \mathrm{TxL}(H)$ modulo singletons.*

Proof. Suppose that there is a shapely cft grammar H that generates K modulo singletons. Let $\tau \in K$, with $|\tau| > 1$. Then $shape(\tau) = di(t)$, where t is a derivation tree of τ. Hence *out-degree*$(shape(\tau)) \leq maxrh(H)$. Consequently, K is limited.

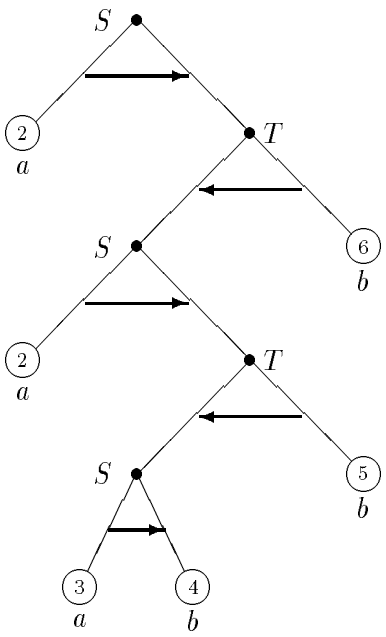


Figure 4.35: derivation tree of $(a^3b^3, (1,6,2,5,3,4))$

Suppose that K is limited. Let $G = (\Theta, \Delta, \Pi, st(S))$ be a chain-free cft grammar in PNF generating K modulo singletons. By Lemma 4.1.19, for each derivation tree t of a text τ in K, $di(t)$ is a primitive refinement of $shape(\tau)$. More precisely, $di(t)$ is obtained from $shape(\tau)$ by replacing each inner node by a subtree of the form $tree_t(\mathcal{E})$, where $\mathcal{E}$ is an equivalence class of $\equiv_t^*$ (as in the proof of Lemma 4.1.19).

Let $\mathcal{D} = \{tree_t(\mathcal{E}) \mid t \text{ a derivation tree in } G, \mathcal{E} \text{ an equivalence class of } \equiv_t^*\}$.

Here the nodes of each $tree_t(\mathcal{E})$ keep their nonterminal labels from t. Hence $\mathcal{D}$ consists of node-labeled bi-ordered trees.

Claim 4.6.5 *$\mathcal{D}$ is finite.*

Proof. Let t be a derivation tree of a text τ, and let $\mathcal{E}$ be an equivalence class of $\equiv_t^*$. By the above mentioned correspondence between $tree_t(\mathcal{E})$ and an inner node of $shape(\tau)$, it follows that $\#leaf(tree_t(\mathcal{E})) \leq out\text{-}degree(shape(\tau))$. Since K is limited and G is chain-free, the number of nodes of the trees in $\mathcal{D}$ is bounded. Consequently, $\mathcal{D}$ is finite. □

We will construct a new cft grammar for K by introducing single productions for each $t \in \mathcal{D}$. In a derivation tree of the new grammar one of these productions will replace the subtree t. In order to make this grammar shapely, we must additionally ensure that its derivation trees are disjoint. We will do this by marking the nonterminals according to the three types of texts we consider: forward and backward sequential, and strictly primitive texts.

Let $\hat{\Theta}$ be the alphabet $\{S\} \cup \Delta \cup \{A^x \mid A \in \Theta - \Delta, x \in \{f, b, p\}\}$. For $x \in \{f, b, p\}$, let ζ^x be the homomorphism from Θ to $\hat{\Theta}$ defined by $\zeta^x(A) = A$, for $A \in \Delta$, and $\zeta^x(A) = A^b$,

for $A \in \Theta - \Delta$. Let $t \in \mathcal{D}$, with $A = lb_t(root(t))$, representing the text $\tau = (w, \mu)$. Define Π_t such that if τ is forward sequential, then $\Pi_t = \{A^x \to (\zeta^f(w), \mu) \mid x \in \{b, p\}\}$; if τ is backward sequential, then $\Pi_t = \{A^x \to (\zeta^b(w), \mu) \mid x \in \{f, p\}\}$; if τ is strictly primitive, then $\Pi_t = \{A^x \to (\zeta^p(w), \mu) \mid x \in \{f, b, p\}\}$; Let $\Pi_{\mathcal{D}} = \bigcup_{t \in \mathcal{D}} \Pi_t$, and let $\Pi_S = \{S \to \tau \mid S^x \to \tau \in \Pi_{\mathcal{D}}, x \in \{f, b, p\}\}$.

Let $\hat{\Pi} = \Pi_{\mathcal{D}} \cup \Pi_S$, and let H be the grammar $(\hat{\Theta}, \Delta, \hat{\Pi}, \tau_S)$. Then H is a shapely grammar equivalent with G, as we will show now.

Let $\tau \in \mathrm{TxL}(H)$, and let t be a derivation tree of τ. By the construction of H, a nonterminal that was introduced by a forward sequential production cannot be rewritten using such a production, and similarly for the backward sequential productions. This means that t (and so $di(t)$) is disjoint. Hence, $di(t)$, being chain-free, locally special, and disjoint, equals $shape(\tau)$. Consequently, H is a shapely grammar.

For each derivation tree t of a text τ in G, a derivation tree of τ in H can be found according to the above construction, i.e., by replacing each subtree $tree_t(\mathcal{E})$ by a single node and adding suitable markings to the nonterminals, according to the type of production that was used to introduce them.

Conversely, each derivation tree t of a text τ in H can be made into a derivation tree of τ in G, by replacing each node v in t by a bi-ordered tree t' from $\mathcal{D}$ such that $\Pi_{t'}$ contains $prod_t(v)$, and removing the markings of the nonterminals.

Hence G and H are equivalent. □

Example 4.6.6 Let G_1, G_2, and G_3 be the cft grammars from Example 4.6.2. The language $\mathrm{TxL}(G_1)$ is not limited, since for each $n \geq 1$, we have $\tau = (a^n b^n, (1, \ldots, 2n)) \in \mathrm{TxL}(G_1)$ with $out\text{-}degree(shape(\tau)) = 2n$. Hence $\mathrm{TxL}(G_1)$ is not limited, and consequently there exists no shapely grammar for $\mathrm{TxL}(G_1)$.

The language $\mathrm{TxL}(G_2)$ is limited; in fact, for each $\tau \in \mathrm{TxL}(G_2)$, $out\text{-}degree(shape(\tau)) \leq 2$. Hence there exists a shapely grammar generating $\mathrm{TxL}(G_2)$, and in fact G_3 is such a grammar. □

It is not difficult to see that it is decidable for a context-free text grammar G whether or not $\mathrm{TxL}(G)$ is limited. This follows from the simple observation that, if G is reduced and in PNF, then $\mathrm{TxL}(G)$ is not limited iff there exists a nonterminal A such that $A \Rightarrow^+ (uAv, \mu)$ where μ is sequential.

We have seen that not each context-free *text* language has a shapely grammar. The following lemma says that it is possible to construct for each context-free *string* language a shapely text grammar generating the string language (modulo words of length 1).

Theorem 4.6.7 *For each cf language L there is a shapely cft grammar G such that* $\mathrm{WL}(G) = \{w \in L \mid |w| > 1\}$.

Proof. Let L be a cf language, and let $H = (\Theta, \Delta, P, S)$ be a cf grammar in Chomsky normal form such that $L = L(H)$. For each A in $\Theta - \Delta$, let $W_A \subseteq \Theta^+$ be such that $\beta \in W_A$ iff either $\beta \in \Delta^+$ or $|\beta| = 4$, and β is derived from A in a leftmost way in a minimal number of steps. Let $H' = (\Theta, \Delta, P', S)$ be the cf grammar such that $P' = \{A \to \beta \mid A \in \Theta - \Delta,$ and $\beta \in W_A\}$. Clearly, H and H' are equivalent.

For each $p = A \to \beta$ in P', define a linear order μ_p as follows:

$$\mu_p = \begin{cases} (1, \ldots, |\beta|) & \text{if } |\beta| < 4 \\ (2, 4, 1, 3) & \text{if } |\beta| = 4 \end{cases} .$$

Now let G' be the cft grammar $(\Theta, \Delta, \Pi, st(S))$, with $\Pi = \{A \to (\beta, \mu_p) \mid p = A \to \beta \in P'\}$. Hence $H' = word(G')$, and so $L = \mathrm{WL}(G')$. Let G be the 1-equivalent chain-free cft grammar obtained from G' as in Lemma 4.4.1. Let t be a derivation tree of a text τ in G. By the form of the productions of G, it follows that each $v \in in(t)$ is primitive or sequential, and v is sequential implies that $ddes_t(v) \subseteq leaf(t)$. Hence $di(t)$ is a chain-free, locally special, and disjoint bi-ordered tree. Consequently, $di(t)$ is the shape of τ.

Hence G is shapely, and, since G and G' are 1-equivalent, $\mathrm{WL}(G) = \{w \in L \mid |w| > 1\}$. □

4.7 Ambiguity and pumping

In this section we investigate two traditional language theoretic topics –ambiguity and pumping– in the setting of cft grammars. The importance of these two topics stems from the facts that the (word-)ambiguity for text languages, in the way we define it, is a notion intrinsic for text languages, and having a pumping lemma for cft languages gives us a much needed tool to prove that some text languages cannot be generated by cft grammars.

In cf string languages, unambiguity means that we can assign to each word of the language a unique structure, namely its derivation tree in an unambiguous cf grammar. In (not necessarily context-free) text languages, we have the situation that each word already *has* a structure, given by (the shape of) a text of which it is the text-word. The requirement that this structure is unique leads to the notion of (word-)unambiguity intrinsic to text languages.

Definition 4.7.1
(1) A text language K is *word-unambiguous* if for all $\tau_1, \tau_2 \in K$, $word(\tau_1) = word(\tau_2)$ implies that $\tau_1 = \tau_2$.
(2) A cft grammar G is *word-unambiguous* if $\mathrm{TxL}(G)$ is word-unambiguous.

For a cft grammar G, G being word-unambiguous is not the same as $word(G)$ being unambiguous. The following theorem relates these notions. By a *weakly unambiguous* cf string grammar we mean a cf grammar such that each word has a unique *naked* derivation tree (i.e., the tree resulting from a derivation tree by removing nonterminal labels).

Theorem 4.7.2 *Let G be a cft grammar.*
(1) *If G is word-unambiguous and shapely, then $word(G)$ is weakly unambiguous.*
(2) *If $word(G)$ is weakly unambiguous and if word is a bijection on $prod(G)$, then G is word-unambiguous.*

Proof.
(1) Each word w generated by $word(G)$ is the text-word of a unique text τ in $\mathrm{TxL}(G)$, since G is word-unambiguous. The derivation tree of τ in G is $shape(\tau)$, since G is shapely. Hence each word w in $L(word(G))$ has a unique naked derivation tree, viz. the underlying ordered tree of $shape(\tau)$.

(2) Since $word$ is a bijection on $prod(G)$, each derivation tree of w in $word(G)$ has a unique extension to a derivation tree of a text τ in G with $word(\tau) = w$. Since $word(G)$ is weakly unambiguous, it follows that all texts with text-word w have the same naked derivation tree, say t, in G. Hence for each text τ with $word(\tau) = w$, the bi-order $(VO(\tau), HO(\tau))$ is represented by this bi-ordered tree t. Consequently, there is exactly one text τ with $word(\tau) = w$, and thus G is word-unambiguous. □

Theorem 4.7.3 *It is not decidable whether a given context-free text grammar is word-unambiguous.*

Proof. We will show that the problem whether a given cft grammar is word-unambiguous is undecidable by reducing the Post Correspondence Problem (PCP) to this problem.

Let $(x_1, \ldots, x_n), (y_1, \ldots, y_n)$ be an instance of the PCP.

Consider the cft grammar $G = (\Theta, \Delta, \Pi, st(S))$, with $\Theta = \{S, S_1, S_2, a, b, c, d\}$, $\Delta = \{a, b, c, d\}$, and $\Pi = \{S \to st(S_1), S \to st(S_2)\} \cup \Pi_1 \cup \Pi_2$, where Π_1 consists of the productions, for $i = 1, \ldots, n$,

$S_1 \to (x_i dc^i, (1, 2, \ldots, n_i))$,

$S_1 \to (x_i S_1 c^i, (1, 2, \ldots, n_i))$, with $n_i = |x_i| + i + 1$,

and Π_2 consists of the productions, for $i = 1, \ldots, n$,

$S_2 \to (y_i dc^i, (m_i, m_i - 1, \ldots, 1))$,

$S_2 \to (y_i S_2 c^i, (m_i, m_i - 1, \ldots, 1))$, with $m_i = |y_i| + i + 1$.

Let G_1 be the cft grammar $(\Theta, \Delta, \Pi_1, st(S_1))$, and let G_2 be the cft grammar $(\Theta, \Delta, \Pi_2, st(S_2))$. Clearly, $\mathrm{TxL}(G) = \mathrm{TxL}(G_1) \cup \mathrm{TxL}(G_2)$, and $\mathrm{WL}(G_1) = \{x_{i_1} \ldots x_{i_k} dc^{i_1} \ldots dc^{i_k} \mid \{i_1, \ldots, i_k\} \subseteq \{1, \ldots, n\}, 1 \leq k \leq n\}$, and the texts in $\mathrm{TxL}(G_1)$ are all forward sequential; similarly, $\mathrm{WL}(G_2) = \{y_{i_1} \ldots y_{i_k} dc^{i_1} \ldots dc^{i_k} \mid \{i_1, \ldots, i_k\} \subseteq \{1, \ldots, n\}, 1 \leq k \leq n\}$, and the texts in $\mathrm{TxL}(G_2)$ are all backward sequential.

Moreover, G is word-unambiguous iff $\mathrm{WL}(G_1) \cap \mathrm{WL}(G_2) \neq \emptyset$, and $\mathrm{WL}(G_1) \cap \mathrm{WL}(G_2) \neq \emptyset$ iff the PCP has a solution.

Hence word-unambiguity for cft grammars is not decidable. □

We move now to a pumping lemma for cft languages. For reasons of readability we introduce the notation $subst^k(\tau, x, \tau')$ for $k \geq 0$ the meaning of which is inductively defined by $subst^0(\tau, x, \tau') = \tau'$, and $subst^k(\tau, x, \tau') = subst(\tau, x, subst^{k-1}(\tau, x, \tau'))$ for $k > 0$.

Theorem 4.7.4 *Let K be a cft language. There exist constants p and q such that for each $\tau \in K$ with $|\tau| > p$, there exist texts $\tau_1, \tau_2, \tau_3, x \in dom(\tau_1), y \in dom(\tau_2)$ such that*

- $\tau = subst(\tau_1, x, subst(\tau_2, y, \tau_3))$,
- $|\tau_2| > 1$,
- $|subst(\tau_2, y, \tau_3)| \leq q$,
- *for each $k \geq 0$, $subst(\tau_1, x, subst^k(\tau_2, y, \tau_3)) \in K$.*

Proof. Our proof of the theorem is rather standard, however we would like to point out that the substitution on texts is a more subtle notion than the substitution on words.

Let $G = (\Theta, \Delta, \Pi, S)$ be a cft grammar in PNF as in Corollary 4.5.4 that generates K. Note that for each successful derivation tree t in G, if each path contains at most n edges, then $\#leaf(t) \leq maxrh(G)^{n-1}$.

Now let $n = \#(\Theta - \Delta)$, $p = maxrh(G)^{n-1}$, and $q = maxrh(G)^n$.

Let $\tau \in K$ be such that $|\tau| > p$, and let t be a derivation tree of τ in G. Since $|\tau| = \#leaf(t) > p$, there is a path consisting of at least $n+1$ edges in t. Consider the lowermost $n+1$ internal nodes on the longest path of t. By the choice of n, there are nodes x and y amongst these nodes such that $lb_t(x) = lb_t(y)$. Let τ_1 be the text represented by the tree that results from t when $sub_t(x)$ is removed with the exception of its root x, let τ_2 be the text represented by the tree that results from $sub_t(x)$ when $sub_t(y)$ is removed with the exception of its root y, and let τ_3 be the text represented by $sub_t(y)$. Then t represents $subst(\tau_1, x, subst(\tau_2, y, \tau_3))$, and hence $\tau = subst(\tau_1, x, subst(\tau_2, y, \tau_3))$. By the form of G, it follows that $|\tau_2| > 1$. In $sub_t(x)$ all paths consist of at most $n+1$ edges, because x and y are amongst the lowermost $n+1$ internal nodes. Since $sub_t(x)$ represents $subst(\tau_2, y, \tau_3)$, and by the form of $sub_t(x)$, it follows that $|subst(\tau_2, y, \tau_3)| \leq q$.

For each $k \geq 1$, a derivation tree for $subst(\tau_1, x, subst^k(\tau_2, y, \tau_3))$ is obtained by substituting $k-1$ copies of the subtree of t that represents τ_2 into t. For $k = 0$, a derivation tree of $subst(\tau_1, x, \tau_3)$ is obtained by removing the subtree representing τ_2 from t, and by making x the root of $sub_t(y)$.

Hence for each $k \geq 0$, $subst(\tau_1, x, subst^k(\tau_2, y, \tau_3) \in K$. □

As in the theory of context-free string grammars, one can use the above pumping lemma to provide examples of text languages that are not context-free.

Example 4.7.5 Consider the text language $K = \{(a^n, \mu_n) \mid n \geq 1\}$, where

$$\mu_n = \begin{cases} (1, \ldots, n) & \text{if } n = 2^j \text{ for some } j \geq 1 \\ (n, \ldots, 1) & \text{otherwise} \end{cases}.$$

Assume that K is context-free. Let p and q be constants as in Theorem 4.7.4. Let $j \geq 1$ be such that $p < 2^j$. Consider $(a^{2^j}, \mu_{2^j}) \in K$. By Theorem 4.7.4 there exist τ_1, τ_2, τ_3, and $x \in dom(\tau_1), y \in dom(\tau_2)$ such that $(a^{2^j}, \mu_{2^j}) = subst(\tau_1, x, subst(\tau_2, y, \tau_3))$ and for each $i \geq 0$, $subst(\tau_1, x, subst^i(\tau_2, y, \tau_3)) \in K$.

Hence $\tau_1 = (a^k x a^\ell, (1, \ldots, k+\ell+1))$ for some $k, \ell \geq 0$; $\tau_2 = (a^r y a^s, (1, \ldots, r+s+1))$, with $r+s \geq 1$; $\tau_3 = (a^m, (1, \ldots, m))$ with $m = 2^j - (k+\ell+r+s)$, and it follows that for each $i \geq 0$,

$(a^{n(i)}, (1, \ldots, n(i))) \in K$, where $n(i) = 2^j + (i-1)(r+s)$. By the definition of K, for each $i \geq 0$, $n(i)$ is a power of 2. Let $n(0) = 2^{j(0)}$, $n(2) = 2^{j(2)}$. Note that $n(0) + n(2) = 2n(1) = 2^{j+1}$. Then it follows that $j(0) = j = j(2)$, which contradicts the fact that $r + s \geq 1$.

Hence K is not a context-free text language. Note that the corresponding string language $\{a^n \mid n \geq 1\}$ is context-free. □

4.8 Discussion

In this paper we have introduced the notion of a context-free text grammar and investigated some of its basic properties.

Although the notion of a context-free grammar is well-known, it gets a rather non-standard interpretation in the case of texts. Since a text has its own individual syntactic structure (viz. its shape), the basic role of a grammar to assign syntactic structures to the objects it generates disappears. Rather one would like to see a context-free text grammar as a definition of a pattern common to all individual syntactic structures of the texts it generates.

The special set-up for context-free grammars which generate texts is reflected e.g. in the basic definitions of a context-free text grammar, some normal forms and in specific notions such as that of a shapely grammar.

As indicated in the paper, the notion of a context-free text grammar can be viewed as a special sort of graph grammar. It also has some similarities with the notion of a context-free tree grammar. Clearly, both relationships should be thoroughly investigated.

Another topic certainly worth investigation is the nature of the context-free sets of texts. One can think here of either a characterization of a combinatorial type (using e.g., notions from [11]), or of a characterization of algebraic style (such as Nerode-type theorems).

Acknowledgements

We would like to thank T. Harju for useful comments on the first draft of this paper, and we are very indebted to H.J. Hoogeboom for being such a useful and engaged partner in discussions concerning this paper; his comments have certainly influenced the current form of the paper.

Chapter 5

Text Languages in an Algebraic Framework

Abstract

A text can be defined as a word w together with a (second) linear order on its domain $\{1, ..., |w|\}$. This second order may be used to define a hierarchical, tree-like, structure representing the text. The family of *context-free* sets of texts is investigated, i.e., sets of texts defined by context-free text grammars. In particular, those sets of texts are studied in the framework of *universal algebra*. This allows to compare the classical notions of *equational* and *recognizable* families in an algebra with context-free sets in the "algebra of texts". Within this algebra the notion of equational sets coincides with the context-free sets. A grammatical characterization of the family of recognizable sets is given as a subfamily of the context-free sets of texts.

Introduction

This paper further investigates the class of context-free texts, that was introduced in [12] generalizing context-free word grammars to the setting of texts.

The notion of a text itself generalizes the notion of a word. A *text* is a triple $\tau = (\lambda, \rho_1, \rho_2)$ such that λ is a labeling function from a domain D to some alphabet, and ρ_1 and ρ_2 are linear orders on the domain D of λ. Usually, $D = \{1, 2, \ldots, n\}$ and ρ_1 is of the standard form $(1, 2, \ldots, n)$. Hence τ may be seen as a word $\lambda(1)\lambda(2)\cdots\lambda(n)$ (referred to as the *word of* τ) together with an additional linear order ρ_2 on the domain $\{1, \ldots, n\}$ of λ.

The traditional role of a context-free word grammar is to define a language as a set of words (generated by the grammar) and to provide each word in the language with its syntactic structure (given by a derivation tree of the grammar). In the case of texts, each text already has an *intrinsic* tree-like structure, called its *shape*, a notion which originates from the decomposition theory of 2-structures (see [13, 14]). Hence, rather than providing each text with a syntactic structure, the role of a context-free text grammar is limited to that of defining a set of texts.

The tree-like structure used to represent a text hierarchically is a so-called leaf-labeled *bi-ordered* tree, which generalizes the concept of a leaf-labeled ordered tree giving a structure

to a word. A tree is bi-ordered if with each inner node *two* linear orderings of its children are associated. These local orderings then determine two orderings on the leaves of the tree.

If the second order of a text equals the first order or the reverse of the first order, then such a *forward* or *backward sequential* text is very much like a word: the text does not impose any structure on the bi-ordered tree representation. On the other hand, very unlike the case of words, there is an important class of texts that have only one (rather trivial) representation, where the leaves of the tree are children of the root, and the associated orderings of the root are the orderings of the text itself. These text are called *primitive*.

A bi-ordered tree representation for a text describes a modular decomposition of the text. This way of decomposing texts is in complete accordance with the decomposition theory for 2-structures (see [16]), due to a close correspondence between texts and a certain subclass of 2-structures. Primitive texts, being undecomposable, play the role of primes in this theory.

It is natural to look for a "maximal" decomposition of a text, i.e., a tree representation where all nodes have a primitive structure. Such a maximal decomposition may not be unique, as in the case of sequential texts where many binary bi-ordered tree representations for each text exist.

However, it turns out that maximal decompositions of a text differ only in their binary subtrees representing sequential parts of the text. Now the *shape* of a text is the unique bi-ordered tree representing the text such that each inner node is either primitive or sequential, and for each forward (backward) sequential node, none of its children is forward (backward, respectively) sequential. Thus, the shape indicates how the text is built up with words (the sequential nodes) and primitive building blocks. Every maximal decomposition of a text can be obtained from its shape by decomposing each sequential node into a binary (bi-ordered) subtree.

Context-free text grammars are direct generalizations of context-free word grammars. In [12] this class of grammars and their text languages were studied.

A text on the one hand can be seen as a *word* with an additional ordering, on the other hand it has an intrinsic *tree*-like structure (as, e.g., given by its shape). The main motivation of this paper is to explore this apparent duality. In particular we want to compare and relate the families of context-free word languages, context-free text languages, and tree languages generated by "regular" tree grammars.

The natural framework to relate these three different structures is that of universal algebra. An algebra is a set A together with a collection Σ of operations on A. If A is the set of words over a given alphabet, then Σ contains only concatenation. Choosing A to be a family of (ranked, ordered) trees, then an operation of rank n in Σ builds a tree out of n given subtrees. For texts one may take operations corresponding to the primitive (de-)compositions of texts.

In this algebra of texts we study the well-established notion of *equational* languages, that formalizes (in an algebraic setting) the notion of a language specified by a set of recursive equations (a context-free grammar can be interpreted as such). Using elementary techniques we show that the equational text languages coincide with the context-free text languages. Additionally we consider the *recognizable* sets, which extends the idea of languages accepted by a finite state device.

For text languages, as for word languages, the recognizable sets are strictly included in the equational sets. For tree languages the two notions coincide.

We isolate a class of context-free grammars that precisely generate the recognizable text languages. We call a context-free text grammar *right-linear* if its (bi-ordered) derivation trees are "rightmost" maximal decompositions of the derived texts. (Such a rightmost maximal decomposition is obtained from the shape by decomposing each sequential node into a "rightmost" binary subtree; thus in this way each text has a unique rightmost maximal decomposition.) Hence, derivation trees are allowed to be "regular tree-like" where the shape has primitive nodes, but they are restricted to "right-linear" sequential parts. This class of right-linear grammars is more powerful than the *shapely* grammars from [12], which allow only shapes as derivation trees.

The paper is organized as follows. We start by giving some preliminaries. In Section 5.1 we present the framework of universal algebra with the classical definitions of recognizable and equational subsets of an algebra. Additionally, in Section 5.2 we recall the basic notions and results on texts and their hierarchical representations.

In Section 5.3 we provide an algebraic framework for texts, and start considering the recognizable and equational text languages. The equational text languages are then the context-free text languages from [12].

In Section 5.4, we give characterizations of recognizable text languages. In particular we prove that the recognizable text languages are precisely those text languages generated by the so-called right-linear text grammars (Theorem 5.4.9). In Section 5.5 we consider how the notions of recognizability and equationality for text languages are related to those for word languages and tree languages. Finally, in Section 5.6 we consider some closure properties of families of text languages.

Preliminaries

For a (finite) sequence $s = (x_1, \ldots, x_n)$, $|s|$ denotes its length n, and for $1 \leq i \leq |s|$, $s(i)$ denotes the i'th element x_i of s. In particular, we view a word w over an alphabet Δ as a sequence of letters of Δ, but as usual we write $w = a_1 \cdots a_n$ if $w(i) = a_i \in \Delta$ for $i = 1, \ldots, n$.

For a nonempty finite set D, a *linear order (on D)* is a relation ρ on D such that ρ is antireflexive, transitive, and total, i.e., for all $x, y \in D$ with $x \neq y$, either $(x, y) \in \rho$ or $(y, x) \in \rho$. For each linear order ρ on D there is a unique ordering $x_1, \ldots, x_n$ of the elements in D such that $(x_i, x_j) \in \rho$ iff $i < j$. Hence a linear order ρ on D can be specified as a sequence of the elements $(x_1, \ldots, x_n)$ of D. The terminology and notations concerning sequences carry over to linear orders.

For a linear order $\rho = (x_1, \ldots, x_n)$, we use $dom(\rho)$ to denote the set $\{x_1, \ldots, x_n\}$. A subset $X \subseteq dom(\rho)$ is a *segment of* ρ if there exist $i, j \in \{1, \ldots, n\}$ such that $X = \{x_\ell \mid i \leq \ell \leq j\}$ (this includes $X = \emptyset$).

For linear orders $\rho_1 = (x_1, \ldots, x_n)$ and $\rho_2 = (y_1, \ldots, y_m)$ with disjoint domains, the *sum of* ρ_1 and ρ_2, denoted $\rho_1 + \rho_2$, is the linear order specified by the sequence $(x_1, \ldots, x_n, y_1, \ldots, y_m)$. Note that this sum operation is not commutative.

If a function f on a set D is given, then we shall extend f in the usual way to a subset X of D, yielding the set $f(X)$, or to a sequence ρ on D, yielding the sequence $f(\rho)$.

By a *tree* t we mean a directed graph with one designated node, the *root of* t, such that each node is connected with the root by a unique directed path from the root. The nodes without outgoing edges are the *leaves* of t, the other nodes are the *inner* nodes of t. A tree is *chain-free* if it has no nodes with precisely one outgoing edge. The *out-degree* of t is the maximum number of outgoing edges per node. A *node-labeled* (or *inner-*, *leaf-labeled*) tree is a tree where in addition each node (or each inner node or each leaf) has a label.

An *ordered tree* is a tree together with a function *ord* that associates to each inner node v a linear order $ord(v)$ on the children of v. These local linear orders induce a linear order ρ on the leaves of the tree. The *yield* of a leaf-labeled ordered tree is the word formed by the labels of the leaves according to the induced ordering of the leaves.

For our purposes, the identity of the nodes of trees and ordered trees is not important. Hence we will consider (ordered) trees modulo the identity of their nodes.

A context-free grammar G is denoted by a 4-tuple (N, Δ, P, S), where N is the alphabet of nonterminals, Δ is the alphabet of terminals, P is the set of productions of the form $A \to w$ with $A \in N$ and $w \in (N \cup \Delta)^*$, and $S \in N$ is the axiom. To emphasize the fact that G is used to generate words we call it a context-free *word* grammar.

5.1 Sigma-algebras

We recall here some notions concerning universal algebra – see, e.g., [2]. A *ranked alphabet* Σ is a finite alphabet of operator symbols, where each operator symbol $\sigma \in \Sigma$ has a rank $r(\sigma) \in \mathbf{N}$; for $m \in \mathbf{N}$, Σ_m denotes $\{\sigma \in \Sigma \mid r(\sigma) = m\}$. A Σ-*algebra* $\mathcal{A}$ is a pair (A, Σ), where A is a set and Σ a ranked alphabet, and each operator $\sigma \in \Sigma_m$, $m \geq 0$, defines a mapping $\sigma^{\mathcal{A}} : A^m \to A$.

Let Σ be a fixed ranked alphabet.

Let $\mathcal{A} = (A, \Sigma)$ and $\mathcal{B} = (B, \Sigma)$ be Σ-algebras. A *homomorphism* h from $\mathcal{A}$ to $\mathcal{B}$ is a mapping $h : A \to B$ such that for each $\sigma \in \Sigma_m$, $h(\sigma^{\mathcal{A}}(a_1, \ldots, a_m)) = \sigma^{\mathcal{B}}(h(a_1), \ldots, h(a_m))$ for all $a_1, \ldots, a_m \in A$. A *congruence of* $\mathcal{A}$ is a relation on A that is invariant under every operator, i.e., $\cong$ is a congruence if, for all $\sigma \in \Sigma_m$, and all $a_1, \ldots, a_m, a'_1, \ldots, a'_m \in A$, $a_i \cong a'_i$ for $1 \leq i \leq m$ implies $\sigma^{\mathcal{A}}(a_1, \ldots, a_m) \cong \sigma^{\mathcal{A}}(a'_1, \ldots, a'_m)$. An *elementary translation of* $\mathcal{A}$ is a mapping $\varphi : A \to A$ defined by $\varphi(v) = \sigma^{\mathcal{A}}(a_1, \ldots, a_{j-1}, v, a_{j+1}, \ldots, a_m)$, where $\sigma \in \Sigma_m$, $1 \leq j \leq m$, and $a_1, \ldots, a_{j-1}, a_{j+1}, \ldots, a_m \in A$. A *translation of* $\mathcal{A}$ is the composition of elementary translations of $\mathcal{A}$. A relation on A is a congruence of $\mathcal{A}$ iff it is invariant under the (elementary) translations of $\mathcal{A}$.

Given a congruence $\cong$ of $\mathcal{A}$, the corresponding *quotient algebra*, denoted by $\mathcal{A}/\cong$, is the Σ-algebra (C, Σ), where C consists of the congruence classes of $\cong$, and for $\sigma \in \Sigma_m$, $\sigma^{\mathcal{A}/\cong}(c_1, ..., c_m)$ is the class of $\sigma^{\mathcal{A}}(a_1, ..., a_m)$, where a_i is a representative of c_i for $i = 1, \ldots, m$.

Homomorphisms and congruences are related as follows : the kernel of a homomorphism h from $\mathcal{A}$ to $\mathcal{B}$, denoted by $\ker(h)$, is the congruence such that $a, a' \in A$ are congruent iff

$h(a) = h(a') \in B$, and each congruence $\cong$ of $\mathcal{A}$ is the kernel of the homomorphism from $\mathcal{A}$ to $\mathcal{A}/\cong$ which assigns to each element of A its congruence class.

Let V be a set of variables. The set of ΣV-*terms*, denoted by $F_\Sigma(V)$, is the smallest set of words over $\Sigma \cup V \cup \{\langle, \rangle\}$ that contains $V \cup \Sigma_0$, and such that if $m \geq 1$, $t_1, \ldots, t_m \in F_\Sigma(V)$, $\sigma \in \Sigma_m$, then $\sigma\langle t_1 \cdots t_m \rangle \in F_\Sigma(V)$. For $V = \emptyset$, we denote $F_\Sigma(\emptyset)$ by F_Σ. Note that $F_\Sigma = \emptyset$ iff $\Sigma_0 = \emptyset$. By considering the variables in V as nullary symbols in Σ we identify $F_\Sigma(V)$ with $F_{\Sigma \cup V}$.

The Σ-algebra of (ground) *terms* $\mathcal{F}_\Sigma = (F_\Sigma, \Sigma)$ is the Σ-algebra such that $\sigma^{\mathcal{F}_\Sigma}(t_1, \ldots, t_m) = \sigma\langle t_1 \cdots t_m \rangle$, for $\sigma \in \Sigma_m$ with $m \geq 1$, $t_1, \ldots, t_m \in F_\Sigma$, and $\sigma^{\mathcal{F}_\Sigma} = \sigma$ for $\sigma \in \Sigma_0$. $\mathcal{F}_\Sigma$ is initial in the class of all Σ-algebras, i.e., for each Σ-algebra $\mathcal{A}$ there is a unique homomorphism from $\mathcal{F}_\Sigma$ to $\mathcal{A}$; if this homomorphism is surjective we say that $\mathcal{A}$ is *generated by* Σ.

Let $\mathcal{A} = (A, \Sigma)$ be a Σ-algebra, and let $\{v_1, \ldots, v_n\}$ be an (ordered) set of variables. With each term $t \in F_\Sigma(\{v_1, \ldots, v_n\})$ we associate a mapping $t^{\mathcal{A}} : A^n \to A$, which is defined by $v_i^{\mathcal{A}}(a_1, \ldots, a_n) = a_i$, and $\sigma\langle t_1 \cdots t_m \rangle^{\mathcal{A}}(a_1, \ldots, a_n) = \sigma^{\mathcal{A}}(t_1^{\mathcal{A}}(a_1, \ldots, a_n)), \ldots\ldots, t_m^{\mathcal{A}}(a_1, \ldots, a_n))$; $t^{\mathcal{A}}$ is a so-called *derived operator*.

A *theory* T is a pair (Σ, E) where Σ is a ranked alphabet, and E is a set of equations of the form $t = t'$ where $t, t' \in F_\Sigma(V)$ for a set of variables V. The Σ-algebra $\mathcal{A} = (A, \Sigma)$ is called a T-*algebra* if it satisfies the equations of T: $t^{\mathcal{A}} = t'^{\mathcal{A}}$ for each $(t = t') \in E$. In particular, the *quotient term algebra* $\mathcal{F}_\Sigma/\cong$, where $\cong$ is the congruence on $\mathcal{A}$ generated by the equations in E, is a T-algebra; moreover it is initial in the class of T-algebras.

5.1.1 Recognizable and equational sets

We recall the basic notions of recognizable and equational sets, in the setting of Σ-algebras (as introduced in [25], see also, e.g., [3], [5], [29]). Let Σ be an arbitrary, but fixed ranked alphabet, and let $\mathcal{A} = (A, \Sigma)$ and $\mathcal{B} = (B, \Sigma)$ be Σ-algebras.

The notion of word languages recognizable by finite state automata can be generalized to subsets of an algebra. Finite algebras take the role of (deterministic) finite state acceptors, and the mapping that assigns to each word the state reached is in this setting a homomorphism of algebras.

Definition 5.1.1 A subset $K \subseteq A$ is *recognizable* if there is a finite Σ-algebra $\mathcal{Q} = (Q, \Sigma)$, a homomorphism $h : \mathcal{A} \to \mathcal{Q}$, and a subset $F \subseteq Q$ such that $h^{-1}(F) = K$.

In view of the natural correspondence of homomorphisms and congruences, one might alternatively define that $K \subseteq A$ is recognizable if there is a finite congruence of $\mathcal{A}$ that saturates K (i.e., there are finitely many congruence classes and K is the union of some of them). It is well-known that the greatest congruence saturating a set K, called the Nérode congruence of K and denoted by $\cong_K$, can be characterized as follows: $a \cong_K a'$ iff for every translation φ of $\mathcal{A}$, $\varphi(a) \in K$ iff $\varphi(a') \in K$.

Proposition 5.1.2 *$K \subseteq A$ is recognizable iff $\cong_K$ is finite.*

Context-free word grammars may be seen as a recursive mechanism for specifying languages. In the framework of universal algebra this generalizes to systems of equations and equational sets.

A *polynomial system* S is a set of equations $\{v_i = t_{i1} + \cdots t_{ik_i} \mid i = 1, \ldots, n\}$, where $\{v_1, \ldots, v_n\}$ is a fixed set of variables, and each t_{ij} is a term in $F_\Sigma(\{v_1, \ldots, v_n\})$. With such a polynomial system S one associates a system function $S^{\mathcal{A}} : (2^A)^n \to (2^A)^n$ satisfying $S^{\mathcal{A}}(W_1, \ldots, W_n) = (W_1', \ldots, W_n')$, where $W_i' = \bigcup_{j=1}^{k_i} t_{ij}^{\mathcal{A}}(W_1, \ldots, W_n)$, for $W_1, \ldots, W_n \subseteq A$. Being a continuous mapping, the system function S has a least fixed point, denoted by $[S^{\mathcal{A}}]$.

Definition 5.1.3 A subset $K \subseteq A$ is *equational* if there is a polynomial system S such that K is a component of the least solution $[S^{\mathcal{A}}]$.

The next theorem collects some facts concerning the behaviour of homomorphisms with respect to recognizability and equationality.

Theorem 5.1.4 *Let $h : \mathcal{A} \to \mathcal{B}$ be a homomorphism.*
(1) *If $L \subseteq A$ is equational, then $h(L) \subseteq B$ is equational.*
(2) *If $K \subseteq B$ is recognizable, then $h^{-1}(K) \subseteq A$ is recognizable.*
(3) *If h is surjective, and if $h^{-1}(K) \subseteq A$ is recognizable (equational), then $K \subseteq B$ is recognizable (equational).*
(4) *If h is injective, and if $h(L) \subseteq B$ is recognizable (equational), then $L \subseteq A$ is recognizable (equational).*
(5) *If $K \subseteq B$ is equational, then there exists an equational set $L \subseteq A$ such that $K = h(L)$.*

Proof. The results for equational sets rely on the general fact (see [25]) that each homomorphism preserves the least fixed point of a polynomial system S, i.e., $h[S^{\mathcal{A}}] = [S^{\mathcal{B}}]$, where h is extended to sequences of subsets of A. E.g., claim (5) follows from the fact that if K is the i'th component of $[S^{\mathcal{B}}]$ for some i, S being a polynomial system, then the i'th component of $[S^{\mathcal{A}}]$ is an equational subset of A and its image under h is K.

For recognizable sets, (2) and (4) follow immediately from the fact that if $K = j^{-1}(F)$ for some homomorphism j, then $h^{-1}(K) = (j \circ h)^{-1}(F)$.

The proof of (3) is as follows. Due to the surjectivity of h, for each (elementary) translation ψ of $\mathcal{B}$ there exists an (elementary) translation φ of $\mathcal{A}$ such that $h(\varphi(a)) = \psi(h(a))$ for each $a \in A$. Assuming that $a \cong_{h^{-1}(K)} a'$ for some $a, a' \in A$, then also $h(a) \cong_K h(a')$ by the characterization of the Nérode congruence in terms of translations. Using again the surjectivity of h we may infer that the index of $\cong_K$ is not larger than the index of $\cong_{h^{-1}(K)}$. Consequently, K is recognizable whenever $h^{-1}(K)$ is. □

Note that in particular it follows from this theorem that isomorphisms preserve recognizability and equationality.

In the case of the term Σ-algebra F_Σ, we have the following result from [25].

Proposition 5.1.5 *For each $T \subseteq F_\Sigma$, T is equational iff T is recognizable.*

For arbitrary Σ-algebras that are generated by Σ this result holds in only one direction.

Corollary 5.1.6 *Let $\mathcal{A} = (A, \Sigma)$ be a Σ-algebra generated by Σ. If $K \subseteq A$ is recognizable, then K is equational.*

Proof. Let h be the unique homomorphism from $\mathcal{F}_\Sigma$ to $\mathcal{A}$. Since $\mathcal{A}$ is generated by Σ, h is surjective. By Theorem 5.1.4(2), $h^{-1}(K) \subseteq F_\Sigma$ is recognizable. By Proposition 5.1.5, $h^{-1}(K)$ is equational. By Theorem 5.1.4(1), $h(h^{-1}(K))$ is equational. Since h is surjective, $K = h(h^{-1}(K))$. □

The general concepts given above can again be specialized to word languages and tree languages.

The word languages in this paper are all ε-free, and hence we work in the semi-group Δ^+ rather than in the monoid Δ^*. Any semi-group can be viewed as a Σ-algebra, where Σ consists of one operation of rank 2. In the case of the free semi-group Δ^+, we may extend Σ by adding the elements of Δ as nullary operators, which makes Δ^+ an algebra generated by its ranked alphabet. More precisely, Δ^+ is a Σ-algebra, with $\Sigma = \Delta \cup \{\sigma\}$ such that σ is an operator of rank 2 that is interpreted as word concatenation, and every nullary operator $a \in \Delta$ is interpreted as the word a of length 1. Moreover, the associativity of concatenation can be expressed by stating that Δ^+ satisfies the equation $e : \sigma\langle u\sigma\langle vw\rangle\rangle = \sigma\langle\sigma\langle uv\rangle w\rangle$, i.e., Δ^+ is a T-algebra for the theory $T = (\Sigma, \{e\})$; the freeness of Δ^+ is expressed by stating that it is an *initial* T-algebra. As is well-known (see,e.g., [25]) the recognizable subsets of Δ^+ are then precisely the (ε-free) word languages recognized by finite state automata, and the equational subsets of Δ^+ are precisely the (ε-free) context-free word languages.

For any ranked alphabet Σ, the terms in F_Σ describe node-labeled ordered trees (modulo the identity of their nodes). Accordingly, the subsets of F_Σ are known as *tree languages* (for an overview on tree languages, see, e.g., the book [21]). Recognizable tree languages are usually defined as tree languages accepted by so-called deterministic bottom-up tree recognizers. This definition is equivalent with Definition 5.1.1.

By Proposition 5.1.5 a tree language is recognizable iff it is equational. A polynomial system for an equational tree language corresponds closely with the notion of *regular tree grammar*, which is a 4-tuple (N, Δ, P, S), where $\Delta = \Sigma_0$, N is the set of nonterminals disjoint from Δ, $S \in N$, and P consists of productions of the form $A \to t$, where $A \in N$ and $t \in F_\Sigma(N)$. A tree $t' \in F_\Sigma(N)$ is derived from a tree $t \in F_\Sigma(N)$, denoted $t \Rightarrow_G t'$, if there is a production $A \to u$ in P such that t' is obtained from t by substituting the tree u for an occurrence of A. As usual $\Rightarrow_G^*$ denotes the transitive and reflexive closure of $\Rightarrow_G$. The *tree language generated by* the regular tree grammar $G = (N, \Delta, P, S)$, denoted by $\mathrm{TrL}(G)$, is the set of trees $\{t \in F_\Sigma \mid S \Rightarrow_G^* t\}$. A regular tree grammar is *in normal form* if each production is of the form $A \to \sigma\langle A_1 \cdots A_m\rangle$ with $\sigma \in \Sigma_m$, $m \geq 0$, and $A_1, \ldots, A_m \in N$.

A regular tree grammar $H = (N, \Delta, P, A_k)$, where $N = \{A_1, \ldots, A_n\}$, corresponds with a polynomial system S with equations $A_i = t_{i1} + \cdots + t_{ik_i}$, for $i = 1, \ldots, n$, where $A_i \to t_{i1}, \ldots, A_i \to t_{ik_i}$ are the productions in P with A_i as left-hand side. The k'th component of $[S^{\mathcal{F}}]$ equals $\mathrm{TrL}(H)$.

Let us note here that also the notion of "context-free tree grammar" exists (see [27]). In such a grammar nonterminals may have nonzero rank. The class of tree languages generated by these context-free tree grammars strictly contains the class of equational tree languages occurring in this paper.

5.2 Texts and text languages

All notions and results given in this section are from [13, 14, 16, 12]. We have tried to keep this overview as brief as possible. For additional technical details we refer to the above-mentioned papers.

5.2.1 Texts and bi-orders

Let Δ be an alphabet.

Definition 5.2.1 A *text* τ *(over* Δ*)* is a triple $(\lambda, \rho_1, \rho_2)$, where ρ_1 and ρ_2 are linear orders such that $dom(\rho_1) = dom(\rho_2)$, and λ is a function from $dom(\rho_1)$ to Δ.

For a text $\tau = (\lambda, \rho_1, \rho_2)$, the *domain of* τ, denoted by $dom(\tau)$, is $dom(\rho_1)$; the *word of* τ, denoted by $word(\tau)$, is the word $\lambda(\rho_1) \in \Delta^+$.

The pair (ρ_1, ρ_2) determines the structural properties of the text τ. A pair of linear orders $\sigma = (\rho_1, \rho_2)$ such that $dom(\rho_1) = dom(\rho_2)$ is called a *bi-order*; the common domain of ρ_1 and ρ_2 is denoted by $dom(\sigma)$.

Bi-orders (and hence texts) correspond with a specific kind of labeled 2-structures ([16, 12]). As a consequence the decomposition theory of 2-structures has a translation to bi-orders. We give here only the result of this translation, and not the details concerning 2-structures.

For a bi-order $\sigma = (\rho_1, \rho_2)$, a subset $X \subseteq dom(\sigma)$ is a *clan of* σ if X is a segment of both ρ_1 and ρ_2. Note that for each bi-order σ, the subsets $\emptyset$, $dom(\sigma)$, and the singletons in $dom(\sigma)$ are all clans of σ, the so-called *trivial* clans. A bi-order is *primitive* if it has no non-trivial clans; it is *sequential* if all segments of both of its linear orders are clans. There are two possible forms for a sequential bi-order $\sigma = (\rho_1, \rho_2)$: either ρ_1 equals ρ_2 (then σ is called *forward sequential*) or ρ_1 and ρ_2 are reverses of each other (then σ is called *backward sequential*). These notions carry over to texts.

In this paper, we will work with abstract bi-orders and texts, i.e., isomorphism classes of bi-orders and texts. Formally, bi-orders $\sigma = (\rho_1, \rho_2)$ and $\sigma' = (\rho_1', \rho_2')$ are *isomorphic* if there is a bijection $\psi : dom(\sigma) \to dom(\sigma')$ such that $\psi(\rho_1) = \rho_1'$ and $\psi(\rho_2) = \rho_2'$. The *length of* an (abstract) bi-order σ, denoted by $|\sigma|$, is $\#dom(\sigma')$ for some representative σ' of σ.

Texts $\tau = (\lambda, \rho_1, \rho_2)$ and $\tau' = (\lambda', \rho_1', \rho_2')$ are *isomorphic* if (ρ_1, ρ_2) and (ρ_1', ρ_2') are isomorphic and for the corresponding bijection $\psi : dom(\tau) \to dom(\tau')$, $\lambda' = \lambda \circ \psi^{-1}$. Hence isomorphic texts have the same word; this allows us to say that an (abstract) text is a pair (w, σ) where σ is an (abstract) bi-order, and w is a word of length $|\sigma|$. The *length of* $\tau = (w, \sigma)$, denoted by $|\tau|$, is $|\sigma|$.

Note that isomorphism of texts or bi-orders respects clans and hence also the above defined properties based on clans.

Sometimes, e.g., in examples, we have to give an abstract bi-order a concrete representation. We then write for a bi-order σ of length n simply the order $(i_1, \ldots, i_n)$ which comes from the representative $((1, 2, \ldots, n), (i_1, \ldots, i_n))$ with domain $\{1, 2, \ldots, n\}$. Accordingly, $\tau = (w, \sigma)$ is written as $(w, (i_1, \ldots, i_n))$. This notation is called the *standard form* of a bi-order (or of a text).

Note that the only bi-orders that are both primitive and sequential are the forward sequential bi-order of length 2, which will be denoted by σ_f in the sequel, the backward sequential bi-order of length 2, denoted by σ_b, and the bi-order of length 1. The set of primitive bi-orders of length > 1 is denoted by PRIM. The bi-orders in PRIM $- \{\sigma_f, \sigma_b\}$ are called *strictly primitive*. There are infinitely many strictly primitive texts.

A text of length 1 is called a *singleton text*. A singleton text τ represented by $(\lambda, (x), (x))$ with $\lambda(x) = a \in \Delta$ is shortly denoted by $\underline{a}$.

5.2.2 Hierarchical representation of texts

In the theory of 2-structures, the notion of a clan underlies the decomposition of 2-structures by forming quotients. By repeatedly applying this quotient decomposition one obtains a decomposition tree which is a hierarchical representation of a 2-structure.

In the case of bi-orders, *bi-ordered trees* serve as hierarchical representations. This notion generalizes an ordered tree in that it is a tree t together with *two* orderings $ord_1(v)$ and $ord_2(v)$ associated to its inner nodes such that for each inner node v, $(ord_1(v), ord_2(v))$ is a bi-order on the children of v.

Remark 5.2.2 Note that equivalently, one can imagine a bi-ordered tree as an ordered tree where each inner node is labeled by an (abstract) bi-order. Then for an inner node v, its bi-order should be matched with its children in such a way that the first order is precisely the order $ord(v)$ from the tree. □

Given a bi-ordered tree t, t represents a bi-order as follows. Similar to the situation for ordered trees, the local linear orders $ord_1(v)$ and $ord_2(v)$ each induce a linear order on the leaves of t. The bi-order represented by t is (ρ_1, ρ_2), where ρ_1 and ρ_2 are the respective induced leave orderings.

Just as an ordered tree which is leaf-labeled hierarchically represents a word, viz. its yield, a leaf-labeled bi-ordered tree t represents the text (w, σ), where w is the yield of the leaf-labeled ordered tree obtained from t by forgetting the second ordering function ord_2, and σ is the bi-order represented by the underlying bi-ordered tree. The text represented by a leaf-labeled bi-ordered tree t is denoted by $txt(t)$.

Thus, a leaf-labeled bi-ordered tree is a hierarchical representation of a text. It corresponds with a decomposition of the text by repeatedly forming quotients into clans (analogous to the decomposition theory of 2-structures). Conversely, given a text τ, each decomposition tree of τ, obtained by repeatedly dividing into clans, is a leaf-labeled bi-ordered tree which represents τ as described above. All this is best illustrated by an example.

Example 5.2.3 Consider the leaf-labeled bi-ordered tree t from Figure 5.1.

For each inner node, the first associated order on the children is the left-to-right order, and the given label determines the second order (cf. Remark 5.2.2 – this label is the standard form of the bi-order associated with the node). E.g., the second order on the children of the root is first the middle child, then the rightmost child, and then the leftmost child. By naming the leaves 1 to 8 from left to right, and reading from the tree the order on the leaves induced by the second orders, we obtain the standard form of $txt(t) : \tau = (abaccbaa, (5, 6, 7, 8, 3, 1, 4, 2))$.

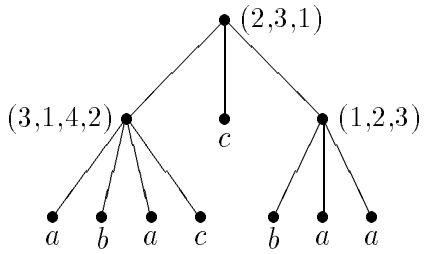


Figure 5.1: bi-ordered tree representing $\tau = (abaccbaa, (5,6,7,8,3,1,4,2))$

The tree t corresponds with the decomposition of τ into $\{1,2,3,4\}$, $\{5\}$, $\{6,7,8\}$ at the root level, followed by decomposing into singletons. Note that these subsets are indeed clans of τ, i.e., segments in both $(\overbrace{1,2,3,4}, \overbrace{5}, \overbrace{6,7,8})$ and $(\overbrace{5}, \overbrace{6,7,8}, \overbrace{3,1,4,2})$.

The node corresponding with $\{6,7,8\}$ can be further refined into $\{6\}$ and $\{7,8\}$, and thus we obtain the decomposition tree t_1, depicted as the leftmost tree in Figure 5.2. By additionally refining the root of the tree we obtain t_2, the middle tree in the figure. Note that the node with associated bi-order $(3,1,4,2)$ allows no further refinement. The rightmost tree t_3 gives another decomposition of τ, but t_3 is not obtained by refining t. Note that t_1, t_2, t_3 are indeed representing the text τ. □

Obviously, adding nodes with a single outgoing edge (i.e., chains) to a leaf-labeled bi-ordered tree does not change the represented text. Throughout this paper, bi-ordered trees are assumed not to have chains, unless they serve as "derivation trees" (see subsection 5.2.3).

For a (leaf-labeled) bi-ordered tree we write simply that an inner node *is* primitive (or sequential) if the node is labeled by a primitive (or sequential) bi-order.

A *primitive representation* of a text τ is a leaf-labeled bi-ordered tree representing τ such that each inner node is primitive. A primitive representation of a text τ corresponds with a "maximal" decomposition of τ in the sense that further decomposing is impossible. In general a text may have more than one primitive representation.

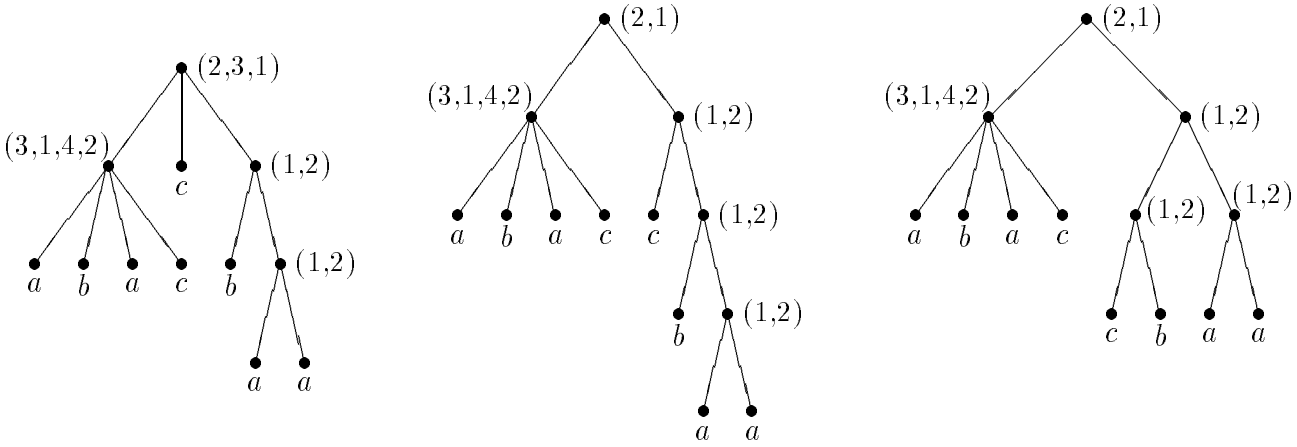


Figure 5.2: three more representations for $\tau = (abaccbaa, (5,6,7,8,3,1,4,2))$

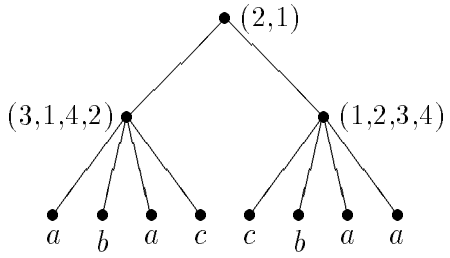


Figure 5.3: the shape of $\tau = (abaccbaa, (5,6,7,8,3,1,4,2))$

The *shape* of a text τ, denoted by $shape(\tau)$, is the unique leaf-labeled bi-ordered tree representing τ such that each inner node is primitive or sequential and, for each forward (backward) sequential node, none of its children is forward (backward, respectively) sequential. The notion of a shape comes from the theory of 2-structures. The shape is obtained by repeatedly decomposing into clans of maximal size that do not overlap other clans. This way of partitioning forces the quotients to be primitive or sequential; the uniqueness of the shape follows from the fact that such partitions are uniquely determined for each bi-order.

Example 5.2.4 In Example 5.2.3 neither t nor t_1 is a primitive representation or the shape of τ, since the root is neither primitive nor sequential. t_2 and t_3 are both primitive representations of the text τ. Neither of them is the shape of τ, since the second child of the root is forward sequential, and has a child which is also forward sequential. The shape of τ is given in Figure 5.3; its root is backward sequential, the second child of the root is forward sequential. □

Given a primitive representation of τ, the shape of τ can be obtained by contracting subsequent nodes with label σ_f into one forward sequential node, and contracting similarly nodes with label σ_b. This entails the following result, shown in [12] (see also [11]).

Proposition 5.2.5 *Each primitive representation of a text τ can be obtained from $shape(\tau)$ by refining the sequential nodes into subtrees the nodes of which have associated bi-orders σ_f or σ_b.*

We denote by $op(\tau)$ the set of bi-orders occurring in a primitive representation of τ. By the above proposition this set is well-defined.

A leaf-labeled bi-ordered tree t representing a text τ can be obtained by step-wise decomposing τ. Conversely, one can view the recovering of τ from t as a step-wise composition of τ. Then one step amounts to applying the operation of *simultaneous substitution*. For a bi-order σ of length $m \geq 1$, and texts $\tau_1 = (w_1, \sigma_1), \ldots, \tau_m = (w_m, \sigma_m)$, the text $[\sigma \leftarrow (\tau_1, \ldots, \tau_m)]$ is defined as follows: let $(\rho_1, \rho_2), (\rho_1^{(1)}, \rho_2^{(1)}), \ldots, (\rho_1^{(m)}, \rho_2^{(m)})$ be representatives of $\sigma, \sigma_1, \ldots \sigma_m$ with mutually disjoint domains, then $[\sigma \leftarrow (\tau_1, \ldots, \tau_m)]$ is the text $(w_1 \cdots w_m, \sigma_0)$ where σ_0 is the bi-order with representative $(\rho_1^{(\rho_1(1))} + \rho_1^{(\rho_1(2))} + \cdots + \rho_1^{(\rho_1(m))},\ \rho_2^{(\rho_2(1))} + \rho_2^{(\rho_2(2))} + \cdots + \rho_2^{(\rho_2(m))})$.

For a leaf-labeled bi-ordered tree t, $txt(t)$ can be obtained from t by repeatedly substituting texts corresponding with subtrees into the bi-order associated with the parent of these subtrees.

More precisely, if t is a leaf-labeled bi-ordered tree where the root has associated bi-order σ, and the direct subtrees of the root are $t_1, \ldots, t_m$, then $txt(t) = [\sigma \leftarrow (txt(t_1), \ldots, txt(t_m))]$.

The following proposition gives a reformulation of the fact that for each text one can construct a primitive representation (see also [11]), and a consequence of the fact that such a primitive representation is a refinement of the shape (Proposition 5.2.5).

Proposition 5.2.6
(1) *Each text can be obtained from singleton texts by repeated substitution into primitive bi-orders.*
(2) *If* $\tau = [\sigma \leftarrow (\tau_1, \ldots, \tau_m)]$ *with* $\sigma \in$ PRIM, *then for each primitive representation of* τ *the root has bi-order* σ*; if* σ *is strictly primitive, then the direct subtrees of its root are primitive representations of* $\tau_1, \ldots, \tau_m$, *respectively.*

We will also use the notion of singular substitution of texts, which is a special case of simultaneous substitution. For a text τ of length m with $word(\tau) = a_1 \cdots a_m$, a text τ', and $1 \leq i \leq m$, the *substitution of* τ' *into* τ *at* i, denoted by $subst(\tau, i, \tau')$, is the text $[\sigma \leftarrow (\underline{a_1}, \ldots, \underline{a_{i-1}}, \tau', \underline{a_{i+1}}, \ldots, \underline{a_m})]$. (Recall that here $\underline{a_j}$ denotes the singleton text with word a_j). Singular substitution underlies the notion of a derivation step in a text grammar.

5.2.3 Text grammars

A set of texts K is called a *text language*. For an alphabet Δ, TXT(Δ) denotes the set of all texts over Δ. Also, for a finite subset Π of PRIM, we use $\mathrm{TXT}_\Pi(\Delta)$ to denote the set of texts over Δ that have a primitive representation using only bi-orders from Π, i.e., $\mathrm{TXT}_\Pi(\Delta)$ is the set of texts τ over Δ for which $op(\tau) \subseteq \Pi$.

A *context-free text grammar* is a 4-tuple $G = (N, \Delta, P, \tau_0)$, where N and Δ are disjoint alphabets, P is a finite set of productions $A \to \tau$, where $A \in N$ and $\tau \in \mathrm{TXT}(N \cup \Delta)$, and τ_0 is a singleton text over N. As usual, we say that the letters of Δ are the terminals, and that the letters of N are the nonterminals of the grammar.

Let $G = (N, \Delta, P, \tau_0)$ be a text grammar.

Let $\tau = (w, \sigma)$ and τ' be texts in $\mathrm{TXT}(N \cup \Delta)$. τ (directly) *derives* τ' (*in* G), denoted $\tau \Rightarrow_G \tau'$, if there is a production $A \to \nu \in P$ and $1 \leq i \leq |\tau|$ with $w(i) = A$ such that $\tau' = subst(\tau, i, \nu)$.

The transitive closure of $\Rightarrow_G$ is denoted by $\Rightarrow_G^+$, and the reflexive and transitive closure by $\Rightarrow_G^*$. We omit the subscript G whenever the grammar G is clear from the context.

$\mathrm{TxL}(G)$ denotes the *text language generated by* G, i.e., $\mathrm{TxL}(G) = \{\tau \in \mathrm{TXT}(\Delta) \mid \tau_0 \Rightarrow_G^* \tau\}$.

Next we define derivation trees in the text grammar G. First recall that in a bi-ordered tree, with each inner node v a bi-order $(ord_1(v), ord_2(v))$ on its children is associated. Now in a *node-labeled* bi-ordered tree, we can associate with each inner node v a *text* on its children, where the word of the text is formed by the labels of the children according to the first ordering $ord_1(v)$. For a node-labeled bi-ordered tree t, we denote by $di(t)$ the leaf-labeled bi-ordered tree that is obtained from t by removing the labels of the inner nodes, and, if occurring, its chains.

Now for a text $\tau \in \mathrm{TXT}(N \cup \Delta)$ and $A \in N$, a *derivation tree of τ from A* in G is a node-labeled bi-ordered tree t such that the root has label A, $di(t)$ represents τ, and for each inner node v of t, the production $B \to \nu$ is a production of P, where B is the label of v and ν is the text associated to v as described above. As usual, for $A \in N$, and $\tau \in \mathrm{TXT}(N \cup \Delta)$, $\underline{A} \Rightarrow^* \tau$ iff there is a derivation tree of τ from A. By a *derivation tree* we mean a derivation tree of some $\tau \in \mathrm{TXT}(\Delta)$ from S, where S is the nonterminal specifying τ_0. We denote by $\mathrm{Di}(G)$ the set $\{di(t) \mid t \text{ is a derivation tree in } G\}$.

A text language K is a *context-free text language* if there exists a context-free text grammar G such that $K = \mathrm{TxL}(G)$. In [12] it was shown that every context-free text language has finitely many primitive subtexts. Consequently, we have the following result, where $op(K) = \{op(\tau) \mid \tau \in K\}$.

Proposition 5.2.7 *For each context-free text language K, $op(K)$ is finite.*

Hence for every context-free text language K, there exists a finite subset $\Pi \subset \mathrm{PRIM}$ and an alphabet Δ such that $K \subseteq \mathrm{TXT}_\Pi(\Delta)$.

By a standard construction productions of the form $A \to \underline{B}$ with A, B nonterminals can be eliminated, i.e., each context-free text grammar has an equivalent chain-free grammar. Therefore, in what follows we assume that each context-free text grammar is chain-free. Hence the only possible chains in derivation trees are ending in a leaf. Also, obviously, we may assume that text grammars are reduced, i.e., for each nonterminal A there is a derivation of a text of the generated language that uses A.

By decomposing the right-hand sides one obtains for each context-free grammar an equivalent text grammar in so-called primitive normal form. A context-free text grammar $G = (N, \Delta, P, \tau_0)$ is in *primitive normal form*, abbreviated PNF, if for each production $A \to \tau$ in P, τ is a primitive text. G is in *Chomsky-like primitive normal form*, abbreviated CPNF, if G is in PNF and for each production $A \to \tau$, either $word(\tau) \in \Delta$, or $word(\tau) \in N^+$.

Note that for each derivation tree t of a context-free text grammar in PNF $di(t)$ is a primitive representation of the generated text. For the following class of text grammars the derivation trees are the *shapes* of the generated texts.

Definition 5.2.8
(1) A context-free text grammar G is *shapely* if $\mathrm{Di}(G) = \{shape(\tau) \mid \tau \in \mathrm{TxL}(G)\}$.
(2) A text language K is *shapely* if there is a shapely grammar generating K.

A text language K is *limited* iff there exists a constant C such that, for each $\tau \in K$, the outdegree of the nodes in the shape of τ is bounded by C. By Proposition 5.2.7, for a context-free text language the outdegree of primitive nodes in the shapes is bounded. However, the sequential nodes in the shapes may in general be of unbounded outdegree. The requirement that these too are bounded forms a necessary and also sufficient condition for the language to be shapely.

Proposition 5.2.9 ([12, Theorem 4.6.4]) *A text language K is shapely iff K is context-free and limited.*

To show the non-context-freeness of text languages, we may use the following pumping lemma, where the meaning of the notation $subst^k(\tau, i, \tau')$ is inductively defined as τ' if $k = 0$, and as $subst(\tau, i, subst^{k-1}(\tau, i, \tau'))$ for $k > 0$.

Proposition 5.2.10 ([12, Theorem 4.7.4]) *Let K be a context-free text language. There exist constants p and q such that for each $\tau \in K$ with $|\tau| > p$, there exist texts τ_1, τ_2, τ_3, $1 \leq i \leq |\tau_1|$, $1 \leq j \leq |\tau_2|$ such that*
(1) $\tau = subst(\tau_1, i, subst(\tau_2, j, \tau_3))$,
(2) $|\tau_2| > 1$,
(3) $|subst(\tau_2, j, \tau_3)| \leq q$,
(4) *for each $k \geq 0$, $subst(\tau_1, i, subst^k(\tau_2, j, \tau_3)) \in K$.*

5.3 An algebra of texts

We give an algebraic structure to the set of texts $\mathrm{TXT}_\Pi(\Delta)$, where Π is a finite subset of PRIM, and Δ is a (finite) alphabet. Each of the primitive bi-orders $\pi \in \Pi$ will act as an operator on $\mathrm{TXT}_\Pi(\Delta)$; its associated mapping is the simultaneous substitution in π.

Let $\Sigma = \Pi \cup \Delta$ be the ranked alphabet such that the rank of each $\sigma \in \Pi$ is $|\sigma|$ and the rank of each $a \in \Delta$ is 0. Then $\mathcal{T}_\Sigma = (\mathrm{TXT}_\Pi(\Delta), \Sigma)$ is the Σ-algebra defined by $\sigma^{\mathcal{T}_\Sigma}(\tau_1, \ldots, \tau_m) = [\sigma \leftarrow (\tau_1, \ldots, \tau_m)]$ for $\sigma \in \Sigma_m$, $m \geq 2$, $\tau_1, \ldots, \tau_m \in \mathrm{TXT}_\Pi(\Delta)$, and $a^{\mathcal{T}_\Sigma} = \underline{a}$ for $a \in \Sigma_0$.

Let $\mathcal{F}_\Sigma = (F_\Sigma, \Sigma)$ be the term Σ-algebra. As mentioned in subsection 5.1.1 we think of the elements of F_Σ as trees; for this specific choice of Σ as $\Pi \cup \Delta$, the trees in F_Σ are, by Remark 5.2.2, primitive bi-ordered trees which are hierarchical representations of texts over Δ. Hence the notion of a primitive representation of a text is in this setting an algebraic expression of a text.

The mapping $txt : F_\Sigma \to \mathrm{TXT}_\Pi(\Delta)$ which assigns to each $t \in F_\Sigma$ the text $txt(t)$ represented by t is a homomorphism of Σ-algebras. First of all, it should be noted that $txt(F_\Sigma) \subseteq \mathrm{TXT}_\Pi(\Delta)$, because, for each tree $t \in F_\Sigma$, $op(txt(t))$ consists of bi-orders labeling inner nodes of t, and so $op(txt(t)) \subseteq \Pi$. Also, txt is indeed a homomorphism, since for each $a \in \Delta$, $txt(a) = \underline{a} = a^{\mathcal{T}_\Sigma}$, and for each $\sigma \in \Sigma_m$, $m \geq 2$, and all $t_1, \ldots, t_m \in F_\Sigma$, $txt(\sigma\langle t_1 \cdots t_m \rangle) = [\sigma \leftarrow (txt(t_1), \ldots, txt(t_m))]$, as explained in subsection 5.2.2.

Hence txt is the unique homomorphism from the initial term Σ-algebra $\mathcal{F}_\Sigma$ to $\mathcal{T}_\Sigma$. By Proposition 5.2.6(1), txt is surjective. We conclude that $\mathcal{T}_\Sigma$ is generated by Σ and that $\mathcal{T}_\Sigma$ is isomorphic with the quotient algebra $\mathcal{F}_\Sigma / \ker(txt)$.

Consider the congruence given by the kernel of txt. Two primitive bi-ordered trees are in the same congruence class iff they represent the same text. By Proposition 5.2.5, such trees differ only in the way the sequential nodes of the shape are refined. It follows that $\ker(txt)$ is precisely the congruence generated by the equations $\sigma_f\langle u \sigma_f \langle vw \rangle\rangle = \sigma_f\langle \sigma_f \langle uv \rangle w\rangle$, $\sigma_b\langle u \sigma_b \langle vw \rangle\rangle = \sigma_b\langle \sigma_b \langle uv \rangle w\rangle$, where u, v, w are variables. Hence $\mathcal{T}_\Sigma$ is a T-algebra, where T is the theory (Σ, E) such that E is the set consisting of the above two equations expressing the associativity of σ_f and σ_b. Moreover, $\mathcal{T}_\Sigma$, being isomorphic with the quotient term algebra $\mathcal{F}_\Sigma / \ker(txt)$, is initial in the class of T-algebras. Of course, if some of the operations σ_f, σ_b are not in Σ, then we restrict E to a subset of these equations. In particular, if $\Sigma \cap \{\sigma_f, \sigma_b\} = \emptyset$, then $E = \emptyset$, and $\mathcal{T}_\Sigma$ is isomorphic with the term Σ-algebra $\mathcal{F}_\Sigma$.

Note also that if $\Pi = \{\sigma_x\}$, with $x \in \{f, b\}$, then the corresponding Σ-algebra of sequential texts $\mathcal{T}_\Sigma = (\mathrm{TXT}_{\{\sigma_x\}}(\Delta), \Sigma)$, with $\Sigma = \{\sigma_x\} \cup \Delta$, is isomorphic with the semi-group Δ^+ seen as a Σ-algebra (see subsection 5.1.1).

Remark 5.3.1
One could add the (primitive) bi-order of length 1 as a unary operation to the ranked alphabet Σ. Its interpretation in $\mathcal{T}_\Sigma$ is the identity, and the terms in F_Σ describe then also trees with chains. If the equation '$\sigma_1(v) = v$', where σ_1 stands for the bi-order of length 1 is added to E, then again $\mathcal{T}_\Sigma$ is a T-algebra, with $T = (\Sigma, E)$. Including σ_1 in this way would not affect any of the results in this paper, but for technical simplicity we have chosen to leave it out. □

We consider recognizability and equationality of text languages, interpreting Definitions 5.1.1 and 5.1.3 in a Σ-algebra $\mathcal{T}_\Sigma$ of texts as described above. Note that if K is a recognizable (or equational) text language in this sense, then for every choice of Π and Δ such that $K \subseteq \mathrm{TXT}_\Pi(\Delta)$, K is recognizable (or equational) w.r.t. the corresponding ranked alphabet $\Sigma = \Pi \cup \Delta$.

In particular, for a forward sequential text language K, K is recognizable or equational iff K is a recognizable or equational subset of $\mathrm{TXT}_{\{\sigma_f\}}(\Delta)$ iff the underlying word language is recognizable or equational w.r.t. the isomorphic $\{\sigma_f\} \cup \Delta$-algebra Δ^+. This immediately provides easy examples of text languages that are equational but not recognizable, e.g., the text language $\{(a^n b^n, (1, \ldots, 2n)) \mid n \geq 1\}$.

We will show that the equational languages are precisely the context-free text languages. Recall that every context-free language, as every equational language, is a subset of $\mathrm{TXT}_\Pi(\Delta)$ for some finite $\Pi \subset \mathrm{PRIM}$ and Δ.

Remark 5.3.2 There is a close correspondence between context-free text grammars for text languages in $\mathrm{TXT}_\Pi(\Delta)$, and regular tree grammars generating tree languages in F_Σ.

For a regular tree grammar $H = (N, \Delta, P, S)$, we denote by $txt(H)$ the text grammar $(N, \Delta, P', \underline{S})$, where $P' = \{A \to txt(t) \mid A \to t \in P\}$. Then $\mathrm{TxL}(txt(H)) = txt(\mathrm{TrL}(H))$. If H is in normal form, then $txt(H)$ is in CPNF, and $\mathrm{Di}(txt(H)) = \mathrm{TrL}(H)$.

Conversely, let $G = (N, \Delta, P, \underline{S})$ be a context-free text grammar. Let $H = (N, \Delta, P', S)$ be a regular tree grammar such that P' contains for each $A \to \tau \in P$ one production $A \to t$ where $t \in F_\Sigma$ is such that $txt(t) = \tau$. Then $txt(\mathrm{TrL}(H)) = \mathrm{TxL}(G)$. If G is in CPNF, then H is in normal form, and $\mathrm{Di}(G) = \mathrm{TrL}(H)$. □

Example 5.3.3 Let $G = (N, \Delta, P, \underline{S})$ be the context-free text grammar such that $N = \{S, A, C\}, \Delta = \{a, c\}$, and P consists of the productions
$S \to (AS, (1, 2)), A \to (aC, (2, 1)), C \to (cCac, (3, 1, 4, 2)), S \to \underline{a}, C \to \underline{c}, A \to \underline{a}$.

Then the regular tree grammar $H = (N, \Delta, P', S)$ generates $\mathrm{Di}(G)$, where P' consists of the productions $S \to \sigma_f\langle AS\rangle$, $A \to \sigma_b\langle aC\rangle$, $C \to \pi\langle cCac\rangle$, $S \to a$, $C \to c$, $A \to a$, where π is the abstract bi-order with standard form $(3, 1, 4, 2)$. □

Lemma 5.3.4 *A text language is equational iff it is context-free.*

Proof. Let K be a text language. It follows from Remark 5.3.2 that K is context-free iff there exists a tree language T, generated by a regular tree grammar, such that $txt(T) = K$. By Theorem 5.1.4(1) and (5) and the fact that a tree language is generated by a regular tree grammar iff it is equational (see subsection 5.1.1), it follows that K is context-free iff K is equational. □

Through this connection between polynomial systems and text grammars, the construction in [25, Lemma 3.1] which yields a "normal form" for polynomial systems is related to the result (in [12]) that each context-free text language has a context-free text grammar in CPNF. Also, this connection is a special case of the situation described in [5], see also [4], where it is shown that given a Σ-algebra $\mathcal{A} = (A, \Sigma)$, one can define a well-behaved substitution device in A such that the equational subsets of A given by a polynomial system are precisely the sets generated (using this substitution) by an "abstract" context-free grammar.

We end this section by formulating the consequences for text languages following from subsection 5.1.1 and this section.

Theorem 5.3.5 *Let $K \subseteq \mathrm{TXT}_\Pi(\Delta)$ be a text language.*
(1) *K is context-free iff it equals $txt(T)$ for some recognizable tree language T.*
(2) *K is recognizable iff $txt^{-1}(K)$ is a recognizable tree language.*
(3) *If K is recognizable, then K is context-free.* □

5.4 Recognizable text languages

We have seen that equational text languages coincide with the text languages generated by context-free text grammars, which were investigated in [12].

We now consider the family of recognizable text languages. Like for generated algebras in general, in the case of texts the class of recognizable sets is included in the class of equational sets. Hence, each recognizable text language is generated by a context-free text grammar (Theorem 5.3.5(3)). Like for words, but unlike trees, this inclusion of recognizable sets in equational sets is strict. As the main result of this section we give a grammatical characterization of the recognizable text languages by restricting the context-free text grammars to a natural subclass. This generalizes to texts the well-known characterization of regular word languages by right-linear grammars.

Previously we have defined recognizability of text languages using finite algebras. Reformulating Proposition 5.1.2, which characterizes the recognizable subsets of an algebra in terms of their Nérode congruences, we obtain the following result.

Lemma 5.4.1 *A text language $K \subseteq \mathrm{TXT}_\Pi(\Delta)$ is recognizable iff the congruence $\cong_K$ is finite, where for $\tau_1, \tau_2 \in \mathrm{TXT}_\Pi(\Delta)$, $\tau_1 \cong_K \tau_2$ iff for all $\tau \in \mathrm{TXT}_\Pi(\Delta)$, and for all i with $1 \leq i \leq |\tau|$, $subst(\tau, i, \tau_1) \in K$ iff $subst(\tau, i, \tau_2) \in K$.* □

Each text has a natural structure, its shape. As we have discussed, all primitive hierarchical representations of a text differ from the shape only by the refinement of sequential nodes into binary subtrees. This implies that if the root of the shape is strictly primitive, then this is the

root of every primitive representation of the text (see Proposition 5.2.6(2)). As a derivation tree for a context-free text grammar gives a representation for the derived text we can translate this observation to derivations in text grammars.

Lemma 5.4.2 *Let $G = (N, \Delta, P, \tau_0)$ be a context-free text grammar in PNF. Let $\sigma \in \mathrm{PRIM}$ with $|\sigma| = m > 2$, and let $\tau = [\sigma \leftarrow (\tau_1, \ldots, \tau_m)]$, where $\tau_1, \ldots, \tau_m \in \mathrm{TXT}(N \cup \Delta)$. Then $\underline{A} \Rightarrow^* \tau$ iff there exist $A_1, \ldots, A_m \in N \cup \Delta$ such that $A \to (A_1 \cdots A_m, \sigma)$ and $\underline{A_j} \Rightarrow^* \tau_j$ for $j = 1, \ldots, m$.* □

More generally, in a derivation tree of a text the subtrees consisting of strictly primitive nodes are determined by the text itself (i.e., by its shape). The only structural freedom in deriving a text lies in the possible decompositions for the sequential nodes of the shape. A natural restriction to context-free text grammars is to force the grammar to choose right-linear derivations for these sequential (word-like) substructures. In other words, we forbid left-recursion in derivation trees: subtrees of the form $\sigma_x\langle\sigma_x\langle t_1 t_2\rangle t_3\rangle$ where $x \in \{f, b\}$. Note that each text τ has a unique primitive representation that has no left recursion; we denote this tree by $nlr(\tau)$.

We formulate this requirement in terms of productions, rather than in terms of derivation trees.

Definition 5.4.3 A context-free text grammar $G = (N, \Delta, P, \tau_0)$ is *right-linear* if G is in PNF and for each production $A \to (BC, \sigma_x) \in P$, with $A, B \in N, C \in N \cup \Delta$, and $x \in \{b, f\}$, if $B \to (w, \sigma) \in P$, then $\sigma \neq \sigma_x$.

Example 5.4.4 Let $G = (N, \Delta, P, \underline{S})$ be the context-free text grammar such that $N = \{S, A, B, C\}, \Delta = \{a, b, c\}$, and P consists of the productions $S \to (AS, (1,2))$, $S \to \underline{b}$, $A \to (aB, (2,1))$, $B \to (AB, (1,2))$, $B \to \underline{b}$, $A \to (cB, (1,2))$. Then G is not right-linear, since $S \to (AS, (1,2)) \in P$, and $A \to (cB, (1,2)) \in P$. □

Lemma 5.4.5 *A context-free text grammar G is right-linear iff* $\mathrm{Di}(G) = nlr(\mathrm{TxL}(G))$. □

The following observation turns out to be crucial in our considerations on right-linear grammars. It is a reformulation of the intuition that sequential substructures of a text are generated by the grammar in a right-linear way. We say that a text τ is *of type* $x \in \{f, b\}$ if the root of a primitive representation of τ has label σ_x (cf. Proposition 5.2.6(2)).

Lemma 5.4.6 *Let $G = (N, \Delta, P, \tau_0)$ be a right-linear text grammar.*
$\underline{A} \Rightarrow^* [\sigma_x \leftarrow (\tau_1, \tau_2)]$ for $x \in \{f, b\}$, $A \in N$, and texts τ_1 and τ_2 over $N \cup \Delta$, iff there exists a $B \in N$ such that $\underline{A} \Rightarrow^* [\sigma_x \leftarrow (\tau_1, \underline{B})]$ and $\underline{B} \Rightarrow^* \tau_2$. Additionally, if τ_1 is not of type x, then $\underline{A} \Rightarrow^* [\sigma_x \leftarrow (\tau_1, \tau_2)]$ iff there exists a production $A \to (CB, \sigma_x)$, where $C \in N$ is such that $\underline{C} \Rightarrow^* \tau_1$. □

It is perhaps instructive to notice that in the case of words this lemma says that in a right-linear grammar $A \Rightarrow^* w_1 w_2$ iff there is a B such that $A \Rightarrow^* w_1 B$ and $B \Rightarrow^* w_2$.

In the definition of right-linearity we have forced the context-free grammar to generate texts according to a specific structure on the derivation trees. We will now define a dual class of grammars. Rather than choosing one normal form for the derivation trees we will impose on the grammar that if it generates a text in any way, then it can also do so according to all primitive representations of the text. This notion generalizes the property of Lemma 5.4.2 to the case where $|\sigma| = 2$.

Definition 5.4.7 A context-free text grammar $G = (N, \Delta, P, \tau_0)$ is *complete* if G is in PNF and for each $A \in N$, and for each $\tau = [\sigma_x \leftarrow (\tau_1, \tau_2)]$, where $x \in \{f, b\}$ and $\tau_1, \tau_2 \in \mathrm{TXT}(\Delta)$, if $\underline{A} \Rightarrow^* \tau$, then there exist $A_1, A_2 \in N \cup \Delta$ such that $A \to (A_1 A_2, \sigma_x) \in P$ and $\underline{A_j} \Rightarrow^* \tau_j$ for $j = 1, 2$.

Note that every complete text grammar has an equivalent complete text grammar in CPNF.

The completeness property is perhaps more intuitive when stated in terms of derivation trees of the grammar.

Lemma 5.4.8 *A context-free text grammar G is complete iff* $\mathrm{Di}(G) = txt^{-1}(\mathrm{TxL}(G))$. □

It turns out that complete grammars and right-linear grammars characterize recognizable text languages.

Theorem 5.4.9 *Let K be a text language. The following statements are equivalent.*
(1) *K is recognizable.*
(2) *There is a complete context-free text grammar G such that $K = \mathrm{TxL}(G)$.*
(3) *There is a right-linear context-free text grammar G such that $K = \mathrm{TxL}(G)$.*

Proof. Note that (in each of the three cases) we can assume that Π and Δ are given such that $K \subseteq \mathrm{TXT}_\Pi(\Delta)$; let $\Sigma = \Pi \cup \Delta$ be the corresponding ranked alphabet.

(1) $\Rightarrow$ (2). If K is recognizable, then by Theorem 2.4(2), $txt^{-1}(K) \subseteq F_\Sigma$ is recognizable, and hence there is a regular tree grammar H for $txt^{-1}(K)$. We may assume that H is in normal form. Since $K = txt(txt^{-1}(K))$, the context-free text grammar $G = txt(H)$ generates K, and $\mathrm{Di}(G) = txt^{-1}(K)$ (see Remark 2.1). Hence, by Lemma 5.4.8, G is complete.

(2) $\Rightarrow$ (3). Let $G = (N, \Delta, P, \tau_0)$ be a complete context-free text grammar in CPNF such that $\mathrm{TxL}(G) = K$. We transform G into a right-linear text grammar by forcing it to choose right-linear derivation trees.

Formally, let $G' = (N', \Delta, P', \tau_0)$ be the context-free text grammar with

$$\begin{aligned} N' &= N \cup \{A^f, A^b \mid A \in N\}, \text{ and} \\ P' &= \{A' \to (w, \sigma) \mid A \to (w, \sigma) \in P, A' \in \{A, A^f, A^b\}, \sigma \notin \{\sigma_b, \sigma_f\}\} \\ &\cup \{A' \to (B^x C, \sigma_x) \mid A \to (BC, \sigma_x) \in P, A' \in \{A, A^y\} \text{ with } y \neq x, x \in \{f, b\}\}. \end{aligned}$$

It is not difficult to see that G' is a right-linear context-free text grammar, and that $\mathrm{TxL}(G') \subseteq \mathrm{TxL}(G)$. Since G is complete, by Lemma 5.4.8, for each text $\tau \in \mathrm{TxL}(G)$ there is a derivation tree in G without left recursion. This derivation tree can be made into a derivation tree of

τ in G' by adding superscripts f and b to some of the nonterminal labels. Hence $\mathrm{TxL}(G) = \mathrm{TxL}(G')$.

(3) $\Rightarrow$ (1). Let $G = (N, \Delta, P, \underline{S})$ be a right-linear grammar in CPNF such that $\mathrm{TxL}(G) = K$. Based on G, we will define a finite Σ-algebra $\mathcal{Q} = (Q, \Sigma)$, and a homomorphism $h : \mathcal{T}_\Sigma \to \mathcal{Q}$ such that $K = h^{-1}(F)$ for some $F \subseteq Q$.

Let W be the set $N \times \{f, b\} \times N$. Let $Q = 2^{N \cup W}$, and let $\mathcal{Q} = (Q, \Sigma)$ be the Σ-algebra defined as follows :

(i) For $a \in \Sigma_0$, let $V = \{A \in N \mid A \to \underline{a} \in P\}$. Then

$$a^{\mathcal{Q}} = V \cup \{(A, x, C) \in W \mid A \to (BC, \sigma_x) \in P, B \in V\}.$$

(ii) For $\sigma \in \Sigma_m$, $m > 2$, $V_1, \ldots, V_m \in Q$, let

$$V = \{A \in N \mid A \to (A_1 \cdots A_m, \sigma) \in P, A_i \in V_i \text{ for } i = 1, \ldots, m\}.$$

Then

$$\sigma^{\mathcal{Q}}(V_1, \ldots, V_m) = V \cup \{(A, x, C) \in W \mid A \to (BC, \sigma_x) \in P, B \in V\}.$$

(iii) For $x \in \{f, b\}$, $V_1, V_2 \in Q$, let

$$V = \{A \in N \mid (A, x, C) \in V_1, C \in V_2\}.$$

Then

$$\begin{aligned} \sigma_x^{\mathcal{Q}}(V_1, V_2) &= V \\ &\cup \{(A, x, C) \in W \mid (A, x, B) \in V_1, (B, x, C) \in V_2\} \\ &\cup \{(A, y, C) \in W \mid A \to (BC, \sigma_y) \in P, B \in V\}, \text{ where } y \neq x. \end{aligned}$$

For notational convenience, we will use $\tau \oplus_x \underline{B}$ as shorthand for $[\sigma_x \leftarrow (\tau, \underline{B})]$, where $x \in \{f, b\}$, $B \in N$, and $\tau \in \mathrm{TXT}_\Pi(\Delta)$.

Let $h : \mathrm{TXT}_\Pi(\Delta) \to Q$ be the mapping such that

$$h(\tau) = \{A \in N \mid \underline{A} \Rightarrow^* \tau\} \cup \{(A, x, C) \in W \mid \underline{A} \Rightarrow^* \tau \oplus_x \underline{C}\}.$$

Claim 5.4.10 *h is a homomorphism from $\mathcal{T}_\Sigma$ to $\mathcal{Q}$.*

Proof.

(i) Let $a \in \Delta$. Since G is in CPNF, $\underline{A} \Rightarrow^* \underline{a}$ iff $A \to \underline{a} \in P$, and $\underline{A} \Rightarrow^* \underline{a} \oplus_x \underline{C}$ iff there is a $B \in N$ such that $A \to (BC, \sigma_x) \in P$ and $B \to \underline{a} \in P$. It follows that $h(a^{\mathcal{T}_\Sigma}) = h(\underline{a}) = a^{\mathcal{Q}}$.

(ii) Let $\sigma \in \Sigma_m$, $m > 2$, let $\tau_1, \ldots, \tau_m \in \mathrm{TXT}_\Pi(\Delta)$, and let $\tau = \sigma^{\mathcal{T}_\Sigma}(\tau_1, \ldots, \tau_m) = [\sigma \leftarrow (\tau_1, \ldots, \tau_m)]$. By Lemma 5.4.2, and since G is in CPNF, $\underline{A} \Rightarrow^* \tau$ iff there exist $A_1, \ldots, A_m \in N$ such that $A \to (A_1 \cdots A_m, \sigma)$ and $\underline{A_j} \Rightarrow^* \tau_j$ for $j = 1, \ldots m$.

Using Lemma 5.4.6, we see that $\underline{A} \Rightarrow^* \tau \oplus_x \underline{C}$ iff there is a $B \in N$ such that $A \to (BC, \sigma_x) \in P$ and $\underline{B} \Rightarrow^* \tau$. Hence, $A \in h(\tau)$ iff there exists $A \to$

$(A_1 \cdots A_m, \sigma) \in P$ such that $A_j \in h(\tau_j)$ for $j = 1, \ldots, m$, and $(A, x, C) \in h(\tau)$ iff there exists $A \to (BC, \sigma_x) \in P$ such that $B \in h(\tau)$.

Consequently $h(\tau) = h(\sigma^{\mathcal{T}_\Sigma}(\tau_1, \ldots, \tau_m)) = \sigma^{\mathcal{Q}}(h(\tau_1), \ldots, h(\tau_m))$.

(iii) Let $x \in \{f, b\}$, let $\tau_1, \tau_2 \in \mathrm{TXT}_\Pi(\Delta)$, and let $\tau = \sigma_x^{\mathcal{T}_\Sigma}(\tau_1, \tau_2) = [\sigma_x \leftarrow (\tau_1, \tau_2)]$. By Lemma 5.4.6, $\underline{A} \Rightarrow^* \tau$ iff there exists a $C \in N$ such that $\underline{A} \Rightarrow^* \tau_1 \oplus_x \underline{C}$ and $\underline{C} \Rightarrow^* \tau_2$. Hence $A \in h(\tau)$ iff $(A, x, C) \in h(\tau_1)$ and $C \in h(\tau_2)$ for some $C \in N$.

Since $\tau = \sigma_x^{\mathcal{T}_\Sigma}(\tau_1, \tau_2)$, we may write $\tau \oplus_x \underline{C} = [\sigma_x \leftarrow (\tau_1, \tau_2 \oplus_x \underline{C})]$ due to the associativity of σ_x. As before, $\underline{A} \Rightarrow^* \tau \oplus_x \underline{C}$ iff there exists a $B \in N$ such that $\underline{A} \Rightarrow^* \tau_1 \oplus_x \underline{B}$ and $\underline{B} \Rightarrow^* \tau_2 \oplus_x \underline{C}$. Hence $(A, x, C) \in h(\tau)$ iff $(A, x, B) \in h(\tau_1)$ and $(B, x, C) \in h(\tau_2)$ for some $B \in N$.

Using once more Lemma 5.4.6, observe that $\underline{A} \Rightarrow^* \tau \oplus_y \underline{C}$ iff there is a $B \in N$ such that $A \to (BC, \sigma_y) \in P$ and $\underline{B} \Rightarrow^* \tau$. Hence for $y \neq x$, $(A, y, C) \in h(\tau)$ iff $A \to (BC, \sigma_y) \in P$ and $B \in h(\tau)$ for some $B \in N$.

By combining the above three cases it follows that $h(\tau) = \sigma_x^{\mathcal{Q}}(h(\tau_1), h(\tau_2))$. □

Note that the above homomorphism h from $\mathcal{T}_\Sigma$ to $\mathcal{Q}$ is unique, by the fact that $\mathcal{T}_\Sigma$ is an initial T-algebra over the theory T given in the previous section. Also $\mathcal{Q}$ itself is a T-algebra, since σ_f and σ_b are associative in $\mathcal{Q}$.

Now let $F \subseteq Q$ be the set $\{V \in Q \mid S \in V\}$. Then $h^{-1}(F) = \{\tau \in \mathrm{TXT}_\Pi(\Delta) \mid h(\tau) \in F\} = \{\tau \in \mathrm{TXT}_\Pi(\Delta) \mid S \in h(\tau)\} = \{\tau \in \mathrm{TXT}_\Pi(\Delta) \mid \underline{S} \Rightarrow^* \tau\} = K$. Hence K is recognizable. □

Theorem 5.4.9 can be understood as follows. Each context-free text language $\mathrm{TxL}(G)$ is of the form $txt(\mathrm{Di}(G))$, where $\mathrm{Di}(G)$ is a recognizable tree language. By requiring that G is complete, it is ensured that $\mathrm{Di}(G)$ is a so-called "saturated" subset of trees; right-linearity of G ensures that $\mathrm{Di}(G)$ is a subset of "well-formed representatives". For both types of recognizable tree languages we have that the corresponding text languages are recognizable.

The next theorem says that shapely text languages form a proper subclass of the class of recognizable text languages.

Theorem 5.4.11 *Let K be a text language. The following statements are equivalent.*
(1) *K is shapely.*
(2) *K is recognizable and limited.*
(3) *K is context-free and limited.*

Proof.
(1) ⇔ (3) This is Proposition 5.2.9.
(1) ⇒ (2) Let G be a reduced shapely grammar for K. By Proposition 5.2.9, K is limited. We continue by showing that K is recognizable. G can be transformed into an equivalent text grammar by replacing each sequential production $A \to (B_1 \cdots B_m, \sigma)$ by the set of productions $A \to (B_1 A_1, \sigma_x)$, $A_1 \to (B_2 A_2, \sigma_x), \ldots, A_{m-2} \to (B_{m-1} B_m, \sigma_x)$, where $A_1, \ldots, A_{m-2}$ are new nonterminals, and $x = f$ or $x = b$ when σ is forward or backward sequential, respectively.
Note that from the shapeliness of G it follows that there are no sequential productions (of the same type as σ) applicable to any of the nonterminals $B_1, \ldots, B_m$. Hence the resulting grammar is right-linear, and, by Theorem 5.4.9, K is recognizable.
(2) ⇒ (3) Follows immediately from Theorem 5.3.5(3). □

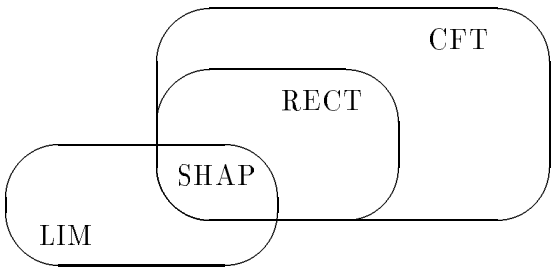


Figure 5.4: families of text languages

The diagram in Figure 5.4 represents the situation. Here CFT, RECT, SHAP, and LIM denote the families of all context-free, recognizable, shapely, and limited text languages, respectively.

5.5 Comparing text, word, tree languages

In comparing texts with words and trees, we take two approaches. First, more or less on the surface, one may view them as three types of objects, ordered by decreasing structure: (bi-ordered) trees can be projected onto texts, and texts can be projected onto words. The other point of view is that words and trees are "inside" a text: they compose the internal structure of a text, in the form of sequential nodes and strictly primitive subtrees of its shape.

We start by taking the first point of view. For given Π and Δ with corresponding ranked alphabet $\Sigma = \Pi \cup \Delta$, the projections involved are the mappings $txt : F_\Sigma \to \mathrm{TXT}_\Pi(\Delta)$, and $word : \mathrm{TXT}_\Pi(\Delta) \to \Delta^+$.

The question addressed here is then: how do these mappings behave with respect to the notions of recognizability and equationality?

For *txt* we obtained some results by applying Theorem 5.1.4 (see Theorem 5.3.5). For *word* we can do the same: one may view it as a homomorphism of Σ-algebras, where every operation of Σ of rank $m \geq 2$ is interpreted in Δ^+ as the concatenation of m words.

If $\Sigma_2 \neq \emptyset$, which is the case iff *word* is surjective, then the notions of recognizability and equationality are stable under this extension of the semi-group Δ^+ to a Σ-algebra. Hence in that case Theorem 5.1.4 can be applied directly. Table 5.1 presents the results, where we have added the projection from trees to words, $yield : F_\Sigma \to \Delta^+$ (which is defined for an arbitrary ranked alphabet Σ with $\Sigma_0 = \Delta$, see also [21]). Here we say that a mapping *reflects* a property of languages if, given that the image of a language has the property, it follows that the original language has the property. Recall that for tree languages, equationality and recognizability coincide.

In the case that $\Sigma_2 = \emptyset$, then at the places where "(if)" is added the claim is not immediate for the mappings *word* and *yield*; a sufficient condition is that $word(word^{-1}(L)) = L$ and that $yield(yield^{-1}(L)) = L$, respectively.

Table 5.1: behaviour with respect to equationality and recognizability

		preserves	reflects
word, *txt*, *yield*	*equat.*	yes	no
	recogn.	no	no
$word^{-1}$, txt^{-1}, $yield^{-1}$	*equat.*	no	yes (if)
	recogn.	yes	yes (if)

We now illustrate the no's in the table by giving some examples. First note that the fact that recognizability is not preserved is a consequence of Theorem 5.1.4(5) and Proposition 5.1.5, and that by claim (3) of Theorem 5.1.4 it can be shown that txt^{-1} and $yield^{-1}$ do not preserve equationality.

The first example in Example 5.5.1 confirms that *txt* and *yield* do not reflect recognizability (nor equationality). The second example shows that also *word* does not reflect recognizability or equationality: a non-context-free text language with a recognizable underlying word language is given. Even if we restrict ourselves a priori to context-free text languages, then still *word* does not reflect recognizability, as is shown in the third example. This example also illustrates that, as opposed to the case of word languages, not all context-free text languages over a one-letter alphabet are recognizable.

Then the only claim left in the above table is that $word^{-1}$ does not preserve equationality, which is shown by the fourth example.

Example 5.5.1
(1) Let T be the tree language $\{\sigma_f\langle t_\ell^{(n)} t_r^{(n)}\rangle \mid n \geq 1\}$, where $t_\ell^{(n)}$ and $t_r^{(n)}$ are trees inductively defined by $t_\ell^{(1)} = t_r^{(1)} = a$ and for $n > 1$, $t_\ell^{(n)} = \sigma_f\langle t_\ell^{(n-1)} a\rangle$ and $t_r^{(n)} = \sigma_f\langle a t_r^{(n-1)}\rangle$. The tree language T is not recognizable, whereas $txt(T)$ and $yield(T)$ are.
(2) Let π be a primitive bi-order such that $|\pi| = 4$. Let K be the text language that consists of all texts with a shape as sketched in Figure 5.5.

Then $word(K) = \{a^{6n+4} \mid n \geq 0\}$ is a recognizable word language. Using Proposition 5.2.10 it can be shown that K is not context-free.
(3) Let $G = (N, \Delta, P, \underline{S})$ be a context-free text grammar such that $N = \{S, A, B\}$, $\Delta = \{a\}$, and P consists of the productions $S \to (ASB, (1,2,3))$, $S \to (AB, (1,2))$, $A \to (a^4, (2,4,1,3))$, $B \to (a^5, (2,5,3,1,4))$.

Let $K = \mathrm{TxL}(G)$. Then $word(K) = \{a^{9n} \mid n \geq 1\}$. Clearly, $word(K)$ is a recognizable word language. However, K is not a recognizable text language. We will show this using Lemma 5.4.1. For $j \geq 1$, we define α_j and β_j as follows. Let $(\alpha_1, \alpha_2, \alpha_3, \alpha_4) = (2,4,1,3)$, and $(\beta_1, \beta_2, \beta_3, \beta_4, \beta_5) = (2,5,3,1,4)$. If $j = 4k + m$ with $m \in \{1,2,3,4\}$, then $\alpha_j = 4k + \alpha_m$; if $j = 5k + m$ with $m \in \{1,2,3,4,5\}$, then $\beta_j = 5k + \beta_m$. Then we can write K as

$$\{(a^{9n}, (\alpha_1, \ldots, \alpha_{4n}, 4n + \beta_1, \ldots, 4n + \beta_{5n}) \mid n \geq 1\}.$$

For $i \geq 1$, let $\tau^{(i)}$ be the text represented by $(a^{5i}, (\beta_1, \ldots, \beta_{5i}))$. Then for all $i, j \geq 1$ with $i \neq j$, there exists a text τ such that $|\tau| = 4i{+}1$, $subst(\tau, 4i{+}1, \tau^{(i)}) \in K$ and $subst(\tau, 4i{+}1, \tau^{(j)}) \notin K$.

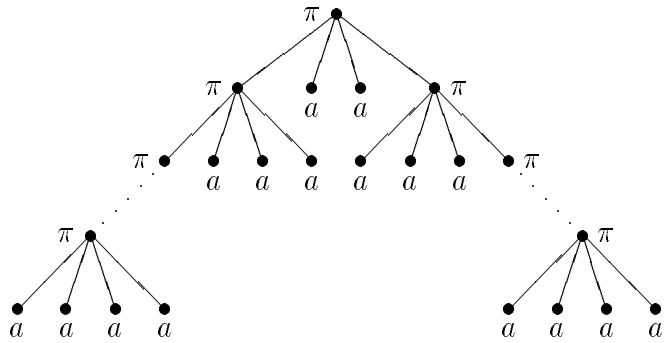


Figure 5.5: the shapes of a non-context-free text language

Hence for all i, j with $i \neq j$, $\tau^{(i)} \not\cong_K \tau^{(j)}$, which implies that $\not\cong_K$ is not finite. By Lemma 5.4.1, K is not recognizable.
(4) Let L be the context-free word language $\{a^{3n+2}b^{3n+2} \mid n \geq 0\}$. Let Σ be a ranked alphabet containing the bi-order π from (2). Then the texts with a shape as in Figure 5.5 but with underlying word $a^{3n+2}b^{3n+2}$ are in $word^{-1}(L)$. It follows that $word^{-1}(L)$ is not a context-free text language, otherwise we could infer from Proposition 5.2.10 that $word^{-1}(L)$ contains a text τ with $word(\tau) = a^i b^j$ with $i \neq j$. □

In the case of *word*, claim (5) of Theorem 5.1.4 says that each context-free word language is the projection of a context-free text language. This was noticed in [12], where moreover it was shown that each context-free word language is the projection of a shapely language.

We now take the second point of view, i.e., seeing words and trees as blocks comprising a text. Intuitively, since recognizability and equationality coincide for tree languages, recognizability of context-free text languages depends only on the word parts. To make this observation explicit, we in some way "extract" word languages from a context-free text grammar. Then a text language is context-free, recognizable, shapely iff these "extracted word languages" are context-free, recognizable, finite (Theorem 5.5.2). This characterization is in particular helpful in the next section, where we consider closure properties.

Formally, we proceed as follows. Let $G = (N, \Delta, P, \tau_0)$ be a context-free text grammar in PNF. Define $V_{p,G}, V_{f,G}, V_{b,G} \subseteq N$ as follows:

$V_{p,G} = \{A \in N \mid \text{there exists } A \to \tau \in P \text{ with } \tau \text{ strictly primitive}\}$,

$V_{f,G} = \{A \in N \mid \text{there exists } A \to (w, \sigma_f) \in P\}$,

$V_{b,G} = \{A \in N \mid \text{there exists } A \to (w, \sigma_b) \in P\}$.

Let i_f and i_b denote the two embeddings from Δ^+ to $\mathrm{TXT}(\Delta)$ defined by, for $w \in \Delta^+$ with $|w| = n$, $i_f(w) = (w, (1, 2, \ldots, n))$, and $i_b(w) = (w, (n, \ldots, 2, 1))$.

For $x \in \{f, b\}$, define $\Delta_{x,G} = V_{p,G} \cup V_{y,G} \cup \Delta$, where $y \in \{f, b\}$ is such that $y \neq x$, and for $A \in N$, let

$$L_{x,G}(A) = \{w \in \Delta_{x,G}^+ \mid \underline{A} \Rightarrow^+ i_x(w)\}.$$

Note that if $A \notin V_{x,G}$, then $L_{x,G}(A)$ is empty or a set of singletons. Finally, let

$$\mathcal{L}_G = \{L_{x,G}(A) \mid A \in N, x \in \{f, b\}\}.$$

Theorem 5.5.2 *Let K be a text language.*
(1) K is context-free iff there exists a context-free text grammar G in PNF generating K such that each $L \in \mathcal{L}_G$ is a context-free word language.
(2) K is recognizable iff there exists a context-free text grammar G in PNF generating K such that each $L \in \mathcal{L}_G$ is a recognizable word language.
(3) K is shapely iff there exists a context-free text grammar G in PNF generating K such that each $L \in \mathcal{L}_G$ is a finite word language.

Proof. We will use the notations given above, where we omit the subscript G if the context-free text grammar G is clear from the context. Note that for the proof it suffices to consider the languages $L \in \mathcal{L}_G$ of the form $L_x(A)$ with $A \in V_x$.

(1) The if-part is, of course, trivial. Let K be a context-free text language, and let $G = (N, \Delta, P, \tau_0)$ be a context-free text grammar in PNF for K. Consider $L \in \mathcal{L}_G$. Suppose that $L = L_f(A)$ for some $A \in V_f$. We show that L is a context-free word language by giving a context-free word grammar for L. The set of nonterminals is $N_f = \{A^f \mid A \in V_f\}$ and the set of productions is

$$P_{f,A} = \{A^f \to w \mid A \to i_f(w') \in P, w \in \varphi(w')\},$$

where φ is the substitution such that for $B \in N \cup \Delta$,

$$\varphi(B) = \begin{cases} \{B^f\} & B \in V_f - (V_p \cup V_b) \\ \{B^f, B\} & \text{if } B \in V_f \cap (V_p \cup V_b) \\ \{B\} & \text{otherwise} \end{cases}.$$

Then for the context-free word grammar $G' = (N_f, \Delta_f, P_{f,A}, A^f)$, $L(G') = L_f(A) = L$.
Similarly, if $L = L_b(A)$ for some $A \in V_b$, then L is a context-free word language.

(2) Let K be a recognizable text language, and let G be a right-linear context-free text grammar for K. Then the context-free grammar G' constructed for $L \in \mathcal{L}_G$ as in (1) is a right-linear word grammar. Hence $L = L(G')$ is a recognizable word language.

Suppose that G is a context-free text grammar for K such that each $L \in \mathcal{L}_G$ is recognizable. For each $L \in \mathcal{L}_G$, let G_L be a right-linear context-free word grammar in Chomsky normal form with production-set P_L. These grammars G_L can be chosen in such a way that the following conditions are satisfied:
- the axiom of G_L is A if $L = L_x(A)$, $x \in \{b, f\}$,
- A does not occur in any right-hand side of P_L, and
- the remaining nonterminals of G_L are disjoint from those of $G_{L'}$ for all $L' \in \mathcal{L}_G$, and disjoint from N.

We remove all sequential productions from G, and add the productions
$\{X \to i_f(w) \mid X \to w \in P_L, L = L_f(A) \text{ for some } A\}$ and
$\{X \to i_b(w) \mid X \to w \in P_L, L = L_b(A) \text{ for some } A\}$.

The thus obtained context-free text grammar is right-linear and equivalent to G. Hence, by Theorem 5.4.9, K is recognizable.

(3) Let K be a text language, and let G be a context-free text grammar in PNF for K. Let $\tau \in K$, and let t be a derivation tree of τ in G. By Proposition 5.2.5, $di(t)$ is a refinement of the shape of τ. It follows that for each sequential node of the shape of τ, say with n children, there is a corresponding derivation $\underline{A} \Rightarrow^+ i_x(w)$ in G such that $|w| = n$ and $w \in L_x(A)$.

Conversely, if w is a word of length n in some $L_x(A) \in \mathcal{L}_G$, then there is a derivation tree t in G of a text $\tau \in K$ with the following property: there is a subtree of t which is a derivation tree of $i_x(w)$ from A and moreover this subtree corresponds with a sequential node with n children in the shape of τ.

Hence the outdegrees of the sequential nodes in the shapes of K are precisely the lengths of the words occurring in the languages $L \in \mathcal{L}_G$. It follows that K is limited iff every $L \in \mathcal{L}_G$ is finite. By Proposition 5.2.9, this proves (3). □

In (1) and (3) of the proof it is in fact shown that K is context-free (shapely) iff for each context-free text grammar G in PNF generating K each $L \in \mathcal{L}_G$ is a context-free (finite) word language. Concerning (2), it is shown that K is recognizable iff for each *right-linear* grammar G generating K each $L \in \mathcal{L}_G$ is a recognizable word language; here we can not replace "right-linear" by "in PNF" as Example 5.5.3 will show.

Example 5.5.3 Let $G = (N, \Delta, P, (S, \sigma_1))$ be a context-free text grammar such that P consists of the productions $S \to (ASB, (1,2,3))$, $S \to (AB, (1,2))$, $A \to (a^4, (2,4,1,3))$, and $B \to (a^4, (2,4,1,3))$.

Then $\mathrm{TxL}(G)$ is recognizable. However, $L_f(S) = \{A^n B^n \mid n \geq 1\}$ is a non-recognizable word language. □

5.6 Closure properties

Most operations on text languages given in this section are defined for arbitrary text languages, i.e., with possibly infinite set of operations $op(K)$, except for the "algebraic closure", which must be defined w.r.t. a fixed text algebra $\mathcal{T}_\Sigma$.

First we introduce the operations which will provide an operational characterization of the context-free text languages (Theorem 5.6.4). These operations are the natural extensions of substitution and substitution closure on word languages.

Definition 5.6.1 A mapping $j : \mathrm{TXT}(\Delta) \to 2^{\mathrm{TXT}(\Delta)}$ is an *alphabetic (text) substitution (on* Δ) if there is a mapping $j_0 : \Delta \to 2^{\mathrm{TXT}(\Delta)}$ such that for $\tau = (a_1 \cdots a_m, \sigma)$,

$$j(\tau) = \{[\sigma \leftarrow (\tau_1, \ldots, \tau_m)] \mid \tau_i \in j_0(a_i) \text{ for } i = 1, \ldots, m\}.$$

j is a *unary* alphabetic substitution *at* a if $j_0(x) = \{x\}$ for every $x \in \Delta$ with $x \neq a$.

We will use j to denote both the mapping j and the mapping j_0. The adjective *alphabetic* is used to distinguish this notion of substitution from the singular and simultaneous substitutions on texts as defined in Section 5.2.

Definition 5.6.2 Let $a \in \Delta$ and let $K \subseteq \mathrm{TXT}(\Delta)$. The *alphabetic substitution closure of* K *at* a is defined to be $\bigcup_{n=0}^{\infty} j^n(\{a\})$, where j is the unary alphabetic substitution at a such that $j(a) = K \cup \{a\}$.

Alphabetic text substitution and alphabetic text substitution closure are the counterparts of the regular operations on tree languages: tree concatenation and tree concatenation closure (called "forest products" in [21]). We now give the counterparts of the regular operations on word languages.

The operations given by the bi-orders of PRIM generalize concatenation of word languages (and correspond with so-called "top-concatenation" of tree languages). Let $\sigma \in \mathrm{PRIM}$ with rank $m \geq 2$. For text languages $K_1, \ldots, K_m \subseteq \mathrm{TXT}(\Delta)$, $[\sigma \leftarrow (K_1, \ldots, K_m)]$ is the text language $\{[\sigma \leftarrow (\tau_1, \ldots, \tau_m)] \mid \tau_i \in K_i \text{ for } i = 1, \ldots, m\}$.

By the *algebraic closure* (*w.r.t.* $\Sigma = \Pi \cup \Delta$) *of* a text language $K \subseteq \mathrm{TXT}_\Pi(\Delta)$ we mean the language $\{[\sigma \leftarrow (\tau_1, \ldots, \tau_m)] \mid (w, \sigma) \in \mathrm{TXT}_\Pi(\Delta) \text{ for some } w \in \Delta^+ \text{ with } |w| = m,\ \tau_1, \ldots, \tau_m \in K\}$, i.e., the sub-algebra of $\mathcal{T}_\Sigma$ generated by K. Algebraic closure generalizes Kleene closure of word languages.

Theorem 5.6.3
(1) CFT *is closed under union, the operations of* PRIM*, algebraic closure, alphabetic substitution, alphabetic substitution closure, and intersection with recognizable text languages.*
(2) CFT *is not closed under intersection and complement.*

Proof. (1) By standard constructions as in [21, Ch. II-4], and [28, Ch. I-3]. See also [5], where in particular it is shown that the intersection of an equational and a recognizable set is again equational (w.r.t. to an arbitrary Σ-algebra).

As an example we will give the proof for the operations of PRIM and for the alphabetic substitution operation.

Let $\sigma \in \mathrm{PRIM}$ with $|\sigma| = m$, and let $K_1, \ldots, K_m \in \mathrm{CFT}$ be such that $K_i = \mathrm{TxL}(G_i)$ with $G_i = (N_i, \Delta_i, P_i, \underline{S_i})$ for $i = 1, \ldots, m$. Let $K = [\sigma \leftarrow (K_1, \ldots, K_m)]$. K is generated by the context-free text grammar $G = (N, \Delta, P, \underline{S})$, where $S \notin \bigcup_{i=1}^m N_i$, $N = (\bigcup_{i=1}^m N_i) \cup \{S\}$, $\Delta = \bigcup_{i=1}^m \Delta_i$, and $P = (\bigcup_{i=1}^m P_i) \cup \{S \to (S_1 \ldots S_m, \sigma)\}$. Hence $K = \mathrm{TxL}(G) \in \mathrm{CFT}$.

Let $K \in \mathrm{CFT}$ be a text language over Δ, and let j be an alphabetic substitution on Δ. For $a \in \Delta$, let $G_a = (N_a, \Delta, P_a, \underline{S_a})$ be a context-free text grammar in PNF for $j(a)$ such that if $a \neq b$, then $N_a \cap N_b = \emptyset$. Let $G = (N, \Delta, P, \underline{S})$ be a context-free grammar in CPNF for K such that N is disjoint from each N_a. Now $G' = (\bigcup_{a \in \Delta} N_a \cup N, \Delta, P', \underline{S})$ is a context-free text grammar generating $j(K)$, where $P' = (\bigcup_{a \in \Delta} P_a) \cup \{A \to \tau \in P \mid |\tau| \geq 2\} \cup \{A \to \tau \mid A \to \underline{a} \in P, S_a \to \tau \in P_a\}$.

Hence $\mathrm{TxL}(G') = j(K)$ is a context-free text language, i.e. $j(K) \in \mathrm{CFT}$.

(2) Let L_1, L_2 be context-free word languages such that $L_1 \cap L_2$ is not context-free. Consider the corresponding forward sequential text languages, i.e., $K_1 = i_f(L_1)$ and $K_2 = i_f(L_2)$. Then K_1 and K_2 are context-free text languages, and $word(K_1 \cap K_2) = word(K_1) \cap word(K_2)$ (note that this is not generally true for arbitrary context-free text languages K_1,K_2). Then $K_1 \cap K_2$ is a text language which is *not* context-free, otherwise $word(K_1 \cap K_2) = L_1 \cap L_2$ would be a context-free word language. For the complement a similar argument applies. □

The following theorem gives an operational characterization of context-free text languages. It is a consequence of Theorem 5.6.3 and the fact that each context-free text language can be obtained from finite text languages by union, alphabetic substitution, and alphabetic substitution closure, which can be shown by Theorem 5.3.5(1) and the analogous result for tree languages (Theorem 5.8 in [21], where tree languages obtained from finite tree languages using the analogous operations are called "regular"), or by directly performing a similar construction as in the proof given there in terms of texts.

Theorem 5.6.4 CFT *is the smallest family of text languages containing the finite text languages that is closed under union, alphabetic substitution, and alphabetic substitution closure.* □

It is well-known (and easy to prove) that recognizable subsets (w.r.t. to any Σ-algebra) are closed under union, intersection, and complement.

In [5] closure properties of recognizable subsets of algebras over a theory are investigated, depending on the form of the equations in the theory. From this we obtain that for each so-called "relabeling" $r : \Delta \to \Gamma$, if $K \subseteq \mathrm{TXT}(\Delta)$ is recognizable, then $r(K) \subseteq \mathrm{TXT}(\Gamma)$ is recognizable. Note that a relabeling is a special case of alphabetic substitution. We will show that in our specific case of texts, RECT is closed under the operations of PRIM, algebraic closure, and alphabetic substitution. This is a consequence of the fact that recognizable *word* languages are closed under (word) concatenation, Kleene plus, and (word) substitution, respectively.

Theorem 5.6.5
(1) RECT *is closed under union, intersection, complement, the operations of* PRIM*, algebraic closure, and alphabetic substitution.*
(2) RECT *is not closed under alphabetic substitution closure.*

Proof. (1) Let $\sigma \in \mathrm{PRIM}$ with $|\sigma| = m \geq 2$. Let $K_1, \ldots, K_m$ be recognizable text languages, and let $K = [\sigma \leftarrow (K_1, \ldots, K_m)]$. By Theorem 5.5.2, for each $i \in \{1, \ldots, m\}$, there is a context-free text grammar $G_i = (N_i, \Delta_i, P_i, \underline{S_i})$ in PNF generating K_i such that each $L \in \mathcal{L}_{G_i}$ is a recognizable word language.

Let G be the text grammar from the proof of Theorem 5.6.3 that generates K. We will use the notations from Section 5.5 used in Theorem 5.5.2. Let $A \in V_{f,G}$. If $A \in V_{f,G_i}$ for some $i \in \{1, \ldots, m\}$, then $L_{f,G}(A) = L_{f,G_i}(A)$, which is a recognizable word language by Theorem 5.5.2. The case that $A \notin V_{f,G_i}$ for each $i \in \{1, \ldots, m\}$ occurs iff $\sigma = \sigma_f$ and $A = S$; then $L_{f,G}(S) = L_{f,G_1}(S_1) \cdot L_{f,G_2}(S_2)$. Since $L_{f,G_1}(S_1)$ and $L_{f,G_2}(S_2)$ are recognizable word languages by Theorem 5.5.2, and recognizable word languages are closed under concatenation, it follows that $L_{f,G}(S)$ is recognizable.

Similarly, we show that the word languages, $L_{b,G}(A)$, $A \in V_{b,G}$, are recognizable.

Consequently, each $L \in \mathcal{L}_G$ is recognizable, and it follows by Theorem 5.5.2 that $\mathrm{TxL}(G) = K$ is a recognizable text language.

This proves that RECT is closed under the operations of PRIM.

Let K be a recognizable text language, generated by the grammar $G = (N, \Delta, P, \underline{S})$. Let $\Sigma = \Pi \cup \Delta$ be a ranked alphabet such that $K \subseteq \mathrm{TXT}_\Pi(\Delta)$. Let $G' = (N, \Delta, P', \underline{S})$, where

$P' = P \cup \{S \to (S^m, \sigma) \mid \sigma \in \Pi, |\sigma| = m\}$. Then $\mathrm{TxL}(G')$ is the algebraic closure of K w.r.t. Σ. For $x \in \{f, b\}$, $L_{x,G'}(A) = L_{x,G}(A)$ for all $A \neq S$, and $L_{x,G'}(S) = L_{x,G}(S)$ if $\sigma_x \notin \Sigma$, $L_{x,G'}(S) = (L_{x,G}(S))^+$ otherwise. Hence $\mathcal{L}_{G'}$ consists of recognizable word languages, which implies that $\mathrm{TxL}(G')$ is a recognizable text language. Hence RECT is closed under algebraic closure.

Now let j be an alphabetic substitution. Let G' be the text grammar in PNF from the proof of Theorem 5.6.3 that generates $j(K)$. Let $x \in \{f, b\}$, and let $A \in V_{x,G'}$. If $A \in N_a$ for some $a \in \Delta$, then $A \in V_{x,G_a}$, and $L_{x,G'}(A) = L_{x,G_a}(A)$ is a recognizable word language. If $A \in N$, then $L_{x,G'}(A) = j_x(L_{x,G}(A))$, where j_x is the word substitution on $N \cup \Delta$ defined by $j_x(A) = A$ for each $A \in N$ and $j_x(a) = L_{x,G_a}(S_a)$ for each $a \in \Delta$. Since, by Theorem 5.5.2, the word languages of the form $L_{x,G_a}(S_a)$ are recognizable and recognizable word languages are closed under substitution, it follows that $L_{x,G'}(A)$ is recognizable.

Hence each $L \in \mathcal{L}_{G'}$ is recognizable, and $K = \mathrm{TxL}(G')$ is a recognizable text language. Consequently, RECT is closed under alphabetic substitution.

(2) This follows immediately from the fact that RECT $\subset$ CFT combined with (1) and Theorem 5.6.4. □

One could also prove (2) using the fact that recognizable word languages are not closed under substitution closure.

For recognizable text language we do not have a characterization as in Theorem 5.6.4. The operations derived from regular word operations do not characterize the recognizable text languages, but the *rational* text languages. In [29], rational subsets of arbitrary Σ-algebras are defined as those subsets built from finite languages by union, the operations of Σ and algebraic closure w.r.t. Σ. A general property of rational sets in an arbitrary algebra (cf. Theorem 5.1.4) is that the homomorphic image of a rational set is rational. It may happen that the class of recognizable sets is strictly contained in the class of rational sets (e.g., in arbitrary monoids), or that the class of rational sets is strictly contained in the class of recognizable sets (as shown for tree languages in [29]). By Kleene's Theorem, the rational word languages are precisely the recognizable word languages.

For text languages we have by Theorem 5.6.5 that RAT $\subseteq$ RECT, where RAT is the class of rational text languages. Considering the homomorphism *word* as in Section 5.5, we obtain, by Kleene's Theorem, that the underlying word languages of rational text languages

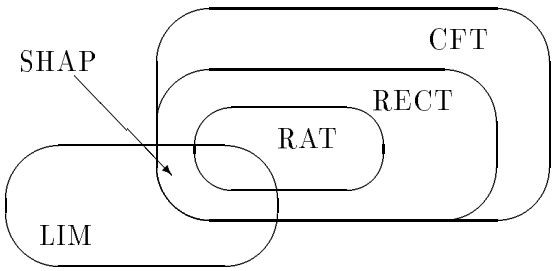


Figure 5.6: families of text languages

are recognizable (cf. the case of recognizable text languages, where the underlying word languages are context-free). This shows that RAT $\subset$ RECT.

Summarizing, we can extend Figure 5.4 yielding the inclusion diagram in Figure 5.6.

Acknowledgements

The authors are indebted to A. Ehrenfeucht and G. Rozenberg for encouraging the research that led to this paper, and to an anonymous referee for making helpful comments.

Chapter 6

Monadic Second-Order Definable Text Languages

Abstract

A *text* is a word together with an (additional) linear ordering. Each text has a generic tree representation, called its *shape*. We consider texts in a logical and an algebraic framework, and we prove that the classes of monadic second-order definable and of recognizable text languages coincide. In particular we demonstrate that the construction of the shape of a text can be formalized in terms of our monadic second-order logic. We briefly consider right-linear grammars for texts.

Introduction

The theory of *2-structures*, introduced in [13, 14], studies modular decompositions of graph-like structures. In this framework *texts* appear as representations for a certain subclass of 2-structures, see [16].

A text is a triple $\tau = (\lambda, \rho_1, \rho_2)$ where ρ_1 and ρ_2 are linear orderings on a common domain D, and λ is a labeling function from this domain D into an alphabet. Usually one takes for D the set $\{1, 2, \ldots, n\}$ and for ρ_1 the standard ordering $(1, 2, \ldots, n)$. In this way a text is in essence a word $\lambda(1)\lambda(2)\cdots\lambda(n)$ extended with an (additional) linear ordering ρ_2 on the set of positions (the domain).

Words can be represented by ordered trees, e.g., to give a syntactic structure to the word. The labels of the leaves spell out the word, where the ordering of the leaves is inherited from the relative orderings of the children of inner nodes. In a similar way one can give hierarchical representations for texts. As a text has two orderings on its domain, the tree representation for a text is *bi-ordered*, i.e., each inner node of the tree has two orderings on its children. Both these orderings are then inherited by the leaves. A bi-ordered tree gives a hierarchical way to build the text from trivial texts (consisting of a single letter) as represented by the leaves, using as operations the bi-orders associated with the inner nodes of the tree, see [12].

In the case of words, the tree representing the word may be refined into a binary tree in an arbitrary fashion; this is the basic step in constructing Chomsky normal form for context-free

grammars. Quite unlike words however, the tree representations for texts are much restricted by the two orderings of the text. There are certain texts (or bi-orders) that can not be decomposed, these are called *primitive*. On the other hand, texts for which the second order equals the first order (or its reverse) are very much like words: there are no restrictions on their (binary) representations, and in each node of the representation the second order equals the first order (or its reverse). Such texts are called *sequential*.

Based on this hierarchical representation of texts, in [24] an algebraic framework for texts is proposed. The operations of a text algebra are given by (a finite set of) primitive bi-orders, that are used as basic building blocks to form texts. It is straightforward to verify that the usual notion of equational sets in this algebra coincides with the notion of sets of texts generated by context-free text grammars as introduced in [12].

It was shown in [24] that *recognizable* subsets of the algebra can be characterized by a subclass of the context-free text grammars called *right-linear* text grammars. These right-linear grammars are much like regular tree grammars when looking at primitive bi-orders and similar to right-linear string grammars when looking at sequential bi-orders.

In this paper we give a characterization of the recognizable text languages in terms of a monadic second-order logic for texts. Let K be a language of texts that are built using a given finite set of bi-orders. We prove that K is definable in our logic if and only if K is recognizable if and only if K is generated by a right-linear text grammar (Theorem 6.3.7).

For recognizable (or regular) word languages and tree languages there exist analogous results, see [1, 9, 25, 30]. Since every word is a specific (sequential) text, our result generalizes the analogous result for word languages. The generalization from words to texts is non-trivial: the underlying words of a recognizable text language do not necessarily form a regular word language (in fact they form a context-free language). Our proof, however, is based on the corresponding result for tree languages. In fact, we show that a text language is mso definable, recognizable or right-linear iff an associated tree language is mso definable, recognizable or regular, respectively. We then use the equivalence of these notions for tree languages to deduce the result for text languages.

As explained above, texts are close to both words and trees. On the other hand, one may view texts as specific graphs, obtained by combining two linear orders. For graph languages in general there does not seem to be such a stable notion of recognizability; e.g., stable in the sense that it has an equivalent grammatical or logical characterization. These matters are discussed extensively by Courcelle in [6], where he conjectures that mso definability and recognizability are equivalent for graph languages of bounded tree-width. In support of this conjecture it is shown that recognizability and mso definability are equivalent within certain "parsable" sets of graphs ([6, Theorem 4.8]), i.e., graph languages where for each graph a tree representing the graph can be constructed – a construction that should be formulated within the logical framework. Examples are given of sets of graphs for which this property holds, in particular the set of graphs of tree-width at most two. Using this terminology, we show that a set of texts built from a finite set of primitive bi-orders is still another example of a parsable set (Theorem 6.2.5). We cannot apply the cited result to obtain equivalence of mso definability and recognizability, as our respective algebraic frameworks differ.

The paper is organized as follows. In Section 6.1 we define texts, and trees representing texts. The notions and results given there rely heavily on the theory of 2-structures. We confine

ourselves to presenting the results on texts needed for this paper and refer to [13, 14, 16, 11] for more background.

Crucial for this paper will be the decomposition of texts into bi-ordered trees representing them. In general this representation is not unique. There exists however a generic tree representation, called the *shape*, that can be constructed from the text. By associating to each text language K the tree language $sh(K)$ of corresponding shapes, and the tree language $txt^{-1}(K)$ of all trees representing texts from K, we obtain a strong connection between text languages and tree languages.

Next, in Section 6.2, we view both texts and their shapes as labeled graphs, and consider a basic monadic second-order logic for graphs (see [18]) to obtain the notion of mso definable text languages. We show that a text language K is mso definable iff the corresponding tree languages $sh(K)$ and $txt^{-1}(K)$ are mso definable. To this aim we demonstrate that the construction of the shape of a given text can be formulated in terms of the logic.

In Section 6.3 we define recognizability for text languages (see also [24]) and prove the equivalence of mso definability and this notion, using the results of the previous section. Additionally, we reconsider the notions of context-free and right-linear text grammar [12, 24]. Observing the close similarity of text grammars and regular tree grammars, the equivalence of right-linear text languages and those that are mso definable is then an easy consequence.

We close our paper with a discussion, in which we sketch how our methods can be used to extend the result to a larger class of graphs than those that represent texts.

An extended abstract of this paper was presented at MFCS'94, see [23].

6.1 Texts and trees

In this preliminary section we present results on texts, and trees representing them, that are needed for this paper. We refer to [16, 11, 12] for more details.

We view a linear order as a nonempty sequence of distinct elements, which form the *domain* of the linear order. For a linear order $\rho = (x_1, \ldots, x_n)$, $n \geq 1$, a *segment of* ρ is a set $\{x_j, x_{j+1}, \ldots, x_k\}$ with $j \leq k$. A *suffix of* ρ is a segment containing x_n. If $i < j$ then we say that x_i *precedes* x_j in ρ.

Definition 6.1.1 Let Δ be an alphabet. A *text* τ (*over* Δ) is a triple $(\lambda, \rho_1, \rho_2)$, where ρ_1 and ρ_2 are linear orders such that the domain of ρ_1 equals the domain of ρ_2, and λ is a labeling function from this common domain to Δ.

For a text $\tau = (\lambda, \rho_1, \rho_2)$, the pair (ρ_1, ρ_2) determines its structural properties; such a pair (ρ_1, ρ_2) of linear orders with a common domain is called a *bi-order*. The common domain of ρ_1 and ρ_2 is the *domain* of the text τ, denoted by $dom(\tau)$. If $\rho_1 = (x_1, \ldots, x_n)$, then the *word* of τ is the word $\lambda(x_1) \cdots \lambda(x_n)$.

By the *length* of a text (or bi-order) we mean the number of elements in its domain. For our purposes the identities of these elements are not important. Therefore, we usually assume that the domain of a text (or bi-order) of length n equals $\{1, \ldots, n\}$, and that the first order is $(1, \ldots, n)$; we refer to this as the *standard form* of a text or bi-order. We may then represent

a bi-order by its second order $(i_1, \ldots, i_n)$ and a text by the pair $(w, (i_1, \ldots, i_n))$, where w is the word of the text.

For a bi-order π of length m and texts $\tau_1, \ldots, \tau_m$, the *substitution* of $\tau_1, \ldots, \tau_m$ into π, denoted by $[\pi \leftarrow (\tau_1, \ldots, \tau_m)]$, is the text constructed as follows: let π be in standard form, take copies of $\tau_1, \ldots, \tau_m$ with mutually disjoint domains $X_1, \ldots, X_m$, and define the substitution text on the domain $\bigcup_{i=1}^{m} X_i$ such that the label of x from X_i is the label of x in τ_i, and for $k \in \{1, 2\}$, $x \in X_i$ precedes $y \in X_j$ in the k-th order if either i precedes j in the k-th order of π, or $i = j$ and x precedes y in the k-th order of τ_i.

Note that the above X_i's become segments of both the first and the second order of the constructed text. For an arbitrary text τ, a nonempty set $X \subseteq dom(\tau)$ that is a segment in both the first and the second order of τ is called a *clan of* τ. Clearly, for each text τ, $dom(\tau)$ is a clan of τ and $\{x\}$ is a clan of τ for each $x \in dom(\tau)$. These clans are called the *trivial clans*. If the only clans of a text are the trivial clans, then the text is called *primitive*. We also speak of primitive bi-orders. Note that the two bi-orders of length 2, in standard form given by $(1, 2)$ and $(2, 1)$, respectively, are both primitive; they will be denoted by σ_f (f for "forward") and σ_b (b for "backward").

One might say that a text τ is "decomposable" if it can be obtained as the substitution of some (proper) subtexts into a bi-order π, or, equivalently, if the domains of these subtexts are clans forming a partition of the domain (π giving their relative first and second ordering). Clearly, primitive texts only allow such decomposition into singletons. By exhaustively decomposing subtexts, we obtain a tree-structure for a given text, called a *primitive representation* of the text, which indicates how it is built up from singletons and primitive (i.e., indecomposable) bi-orders.

Example 6.1.2 Let τ be the text $(acabaacbc, (5, 2, 4, 1, 3, 6, 7, 8, 9))$. In Fig. 6.1 two primitive representations of τ are given, where π is the primitive bi-order given by $(2, 4, 1, 3)$ in standard form. Consider the left tree t. At the root τ is decomposed into the subtexts corresponding with the clans $\{1, 2, 3, 4, 5\}$ and $\{6, 7, 8, 9\}$, in standard form $(acaba, (5, 2, 4, 1, 3))$ and $(acbc, (1, 2, 3, 4))$, respectively; note that τ is reobtained by substituting these subtexts into the root label σ_f which gives the relative orderings of the two clans. At the left child of the root the subtext $(acaba, (5, 2, 4, 1, 3))$ is decomposed into the subtexts corresponding with clans $\{1, 2, 3, 4\}$ and $\{5\}$ relatively ordered by σ_b; the subtext corresponding with $\{1, 2, 3, 4\}$ is primitive with underlying bi-order π.

For each inner node of a primitive representation t its label provides two orderings on its children, where the first order is assumed to be the left-to-right order. In this way t is a "doubly" ordered tree, and we can recover the standard form of τ from t as follows: the word of τ is the yield of t, and the (second) order on the positions of this word is the ordering on the leaves induced by the local second orders in the obvious way, assuming that the leaves are named $1, \ldots, 9$ from left to right. □

The trees used in this paper are directed ordered trees with node-labels. Therefore, they can be represented as terms over a ranked alphabet (cf. the definition of trees in [21]). A *ranked alphabet* Σ is a finite alphabet of operator symbols, where each operator symbol $\sigma \in \Sigma$ has a rank, which is a natural number. The set of Σ-*terms*, denoted by F_Σ, is the smallest

set of words over Σ and the auxiliary symbols $\langle$ and $\rangle$ such that $\sigma \in F_\Sigma$ for every $\sigma \in \Sigma$ of rank 0, and $\sigma\langle t_1 \cdots t_m\rangle \in F_\Sigma$ for all $\sigma \in \Sigma$ of rank $m > 0$ and $t_1, \ldots, t_m \in F_\Sigma$. For primitive representations we use ranked alphabets of the following specific form. For a finite set Π of primitive bi-orders of length ≥ 2, and an alphabet Δ, let Σ be the ranked alphabet $\Pi \cup \Delta$ such that the rank of each $\sigma \in \Pi$ is its length and the rank of each $a \in \Delta$ is 0. Each tree t in F_Σ is then a primitive representation; the text represented by t is denoted by $txt(t)$ and is recursively defined by $txt(a) = (a, (1))$ for $a \in \Delta$, and $txt(\pi\langle t_1 \cdots t_n\rangle) = [\pi \leftarrow (txt(t_1), \ldots, txt(t_n))]$ for $\pi \in \Pi$ of length $n \geq 2$, and $t_1, \ldots, t_n \in F_\Sigma$.

Note that trees of the form $\sigma_x\langle\sigma_x\langle t_1 t_2\rangle t_3\rangle$ and $\sigma_x\langle t_1\sigma_x\langle t_2 t_3\rangle\rangle$, with $x \in \{f, b\}$, $t_1, t_2, t_3 \in F_\Sigma$ represent the same texts. The difference between primitive representations of one text is limited to this kind of associativity in binary subtrees. This is a consequence of general results from the decomposition theory for 2-structures, see [14, Theorem 6.22], [12, Lemma 4.1.19].

Proposition 6.1.3 *Two primitive representations of the same text differ only in subtrees over σ_f and subtrees over σ_b.*

We fix one specific primitive representation for each text τ, called the *r-shape* of τ, denoted by $sh(\tau)$, by demanding that each binary subtree is in a rightmost form, i.e., the r-shape of a text is the primitive representation such that it has no subtree of the form $\sigma_f\langle\sigma_f\langle t_1 t_2\rangle t_3\rangle$ or $\sigma_b\langle\sigma_b\langle t_1 t_2\rangle t_3\rangle$. E.g., the r-shape of the text τ from Example 6.1.2 is the right tree in Figure 6.1. (Note that the r-shape slightly differs from the usual shape in [14].)

The r-shape of a text can be characterized in terms of clans. First observe that the nodes of a given primitive representation t correspond with clans of $txt(t)$, viz., the clans used in the decomposition reflected by t. A clan of a text τ is a *prime clan* if it is not overlapping with any other clan of τ, where sets X and Y are *overlapping* if $X \cap Y \neq \emptyset$, $X - Y \neq \emptyset$, and $Y - X \neq \emptyset$.

Proposition 6.1.4 *A primitive representation t is the r-shape of a text τ iff the clans of τ corresponding with the nodes of t are precisely all clans of τ that are suffixes (w.r.t. the first order) of prime clans of τ.*

Example 6.1.5 For the text τ of Example 6.1.2, the non-trivial prime clans are $\{1, 2, 3, 4, 5\}$ and $\{1, 2, 3, 4\}$, and, e.g., $\{6, 7, 8, 9\}$ is a clan that is a suffix of the trivial prime clan $\{1, 2, \ldots, 9\}$.

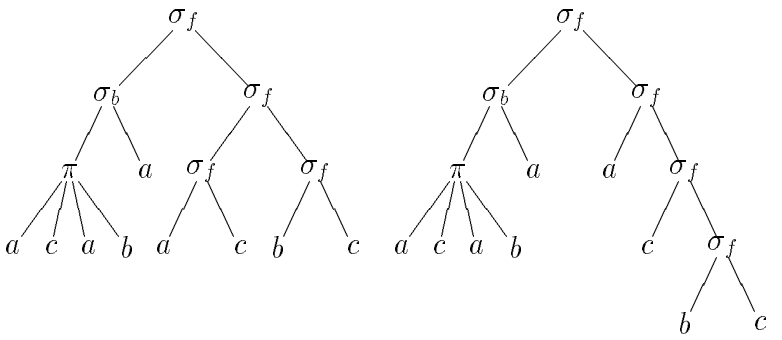


Figure 6.1: two primitive representations of τ

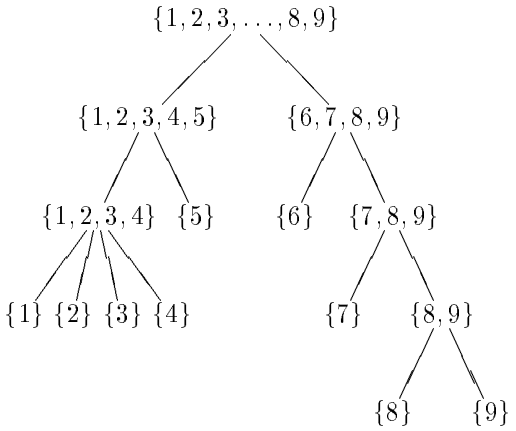


Figure 6.2: the suffixes of prime clans corresponding with the r-shape

In Figure 6.2 the suffixes of prime clans corresponding with the nodes of the r-shape of τ (the right tree in Figure 6.1) are indicated. □

For a ranked alphabet $\Sigma = \Pi \cup \Delta$ as described above, we use $\mathrm{TXT}_\Pi(\Delta)$ to denote the set of texts over Δ that have a primitive representation in F_Σ; the subset of F_Σ consisting of the r-shapes of these texts is denoted by $\mathrm{SH}_\Pi(\Delta)$; $\mathrm{TXT}(\Delta)$ denotes the set of all texts over Δ. It is instructive to note that, for a subset $K \subseteq \mathrm{TXT}_\Pi(\Delta)$, $K = txt(sh(K))$, and $sh(K) = txt^{-1}(K) \cap \mathrm{SH}_\Pi(\Delta)$.

6.2 Mso definable text languages

In this section we view texts, and trees representing them, as graphs. We use monadic second-order logic on graphs to define the class of mso definable text languages, rather than introducing a separate logic for texts. We have chosen this (perhaps indirect) approach since it allows us to perform our constructions within one fixed logical framework, rather than switching between a logic for texts and one for trees representing them.

Let Ω and Γ be alphabets. A *graph* over Ω and Γ is a triple $g = (V, E, \lambda)$, where V is the set of nodes, $E \subseteq V \times \Gamma \times V$ the set of edges, and $\lambda : V \to \Omega$ the node-labeling. The set of all graphs over Ω and Γ is denoted by $\mathrm{GR}(\Omega, \Gamma)$.

The monadic second-order logic $\mathrm{MSO}(\Omega, \Gamma)$ expresses properties of graphs over Ω and Γ. The logic allows both first order variables $x, y, \ldots$ ranging over nodes, and (monadic) second-order variables $X, Y, \ldots$ ranging over sets of nodes. There are four types of atomic formulas: $x = y$, expressing that nodes x and y are equal; $x \in X$, expressing x is an element of X; $\mathrm{lab}_a(x)$, expressing node x has label a (with $a \in \Omega$); and $\mathrm{edge}_\gamma(x, y)$; expressing there is an edge from x to y with label γ (with $\gamma \in \Gamma$).

Formulas are built from atomic formulas with the propositional connectives $\neg, \wedge, \vee, \to$, using the quantifiers $\forall$ and $\exists$ both for node-variables and node set-variables. For better readability we use abbreviations like $X \subseteq Y$ and $X \cap Y = \emptyset$.

Example 6.2.1
(1) The binary predicate $x \prec y$ claiming the existence of a (directed) path from x to y is expressible in $\mathrm{MSO}(\Omega, \Gamma)$:

$$x \prec y \equiv \forall X[x \in X \wedge \forall u \forall v(u \in X \wedge \mathrm{edge}(u,v) \rightarrow v \in X) \rightarrow y \in X]$$

where $\mathrm{edge}(u,v) \equiv \bigvee_{\gamma \in \Gamma} \mathrm{edge}_\gamma(u,v)$. This example is from [30].
(2) The structural graph property of being a (directed) tree is expressed by requiring (a) the existence of a node (the root) that has no incoming edges, and from which all other nodes are reachable, and (b) that all nodes have at most one incoming edge. □

Given a closed formula φ of $\mathrm{MSO}(\Omega, \Gamma)$, and a graph g from $\mathrm{GR}(\Omega, \Gamma)$ we write $g \models \varphi$ if g satisfies φ, i.e., if φ is true when interpreted over g. Now φ defines the graph language of all graphs satisfying φ: $L(\varphi) = \{g \in \mathrm{GR}(\Omega, \Gamma) \mid g \models \varphi\}$. Such a graph language is said to be *mso definable*.

Examples of mso definable graph languages are $L(\,\forall x \forall y[x \prec y]\,)$: the set of all strongly connected graphs, and the set of all (unordered) trees.

Trees have the usual representation as graphs; however, as we are dealing with ordered trees, we need edge-labels to explicitly determine the relative ordering of the children of each node in the tree. We use the natural numbers $1, \ldots, m$ to label the outgoing edges of a node of arity m in the tree. Thus more specifically, for the ranked alphabet Σ, trees in F_Σ are identified with specific graphs in $\mathrm{GR}(\Sigma, \{1, \ldots, \mathrm{M}\})$, where M is the maximal rank of symbols in Σ.

Note that the set F_Σ is mso definable as a subset of $\mathrm{GR}(\Sigma, \{1, \ldots, \mathrm{M}\})$. Apart from imposing the graph to have the structural graph property of Example 6.2.1(2), we additionally require that the outgoing edges of each node are properly labeled by labels $1, \ldots, m$, where m is the arity associated with the label of the node. Let us remark here that it is customary, when defining subsets of F_Σ by mso formulas, to interprete these formulas directly in F_Σ rather than in $\mathrm{GR}(\Sigma, \{1, \ldots, \mathrm{M}\})$. As the property of belonging to F_Σ itself is mso definable, this makes no difference in the expressive power.

Also texts have natural representations as graphs. We use edges to represent the two linear orders of the text, with edge-labels to identify the ordering. Formally, a text $\tau = (\lambda, \rho_1, \rho_2)$ over Δ is identified with the graph $g = (V, E, \lambda)$ over Δ and $\{1,2\}$, where $V = dom(\tau)$, and $E = \{(x,i,y) \mid x, y \in V, x \text{ precedes } y \text{ in } \rho_i, i \in \{1,2\}\}$. Thus, a text language is *mso definable* if it is mso definable as set of graphs over $\mathrm{GR}(\Delta, \{1,2\})$.

The set of texts $\mathrm{TXT}(\Delta)$ in $\mathrm{GR}(\Delta, \{1,2\})$ is easily seen to be mso definable, as we only have to specify the fact that the edges with each label form a linear ordering. Unfortunately, the definability of $\mathrm{TXT}_\Pi(\Delta)$ within $\mathrm{GR}(\Delta, \{1,2\})$ is less transparent. We solve this problem in the proof of Theorem 6.2.5. On the other hand, both F_Σ and the set $\mathrm{SH}_\Pi(\Delta)$ of r-shapes in F_Σ, where $\Sigma = \Pi \cup \Delta$ is a ranked alphabet for primitive representations, are mso definable.

An *mso definable function* f, see [8, 18], specifies the construction of a new graph g' from a graph g using mso formulas specifying nodes, edges and labels of the image g' in terms of the original graph g. The construction starts by taking k copies of the nodes of g (k is fixed

by f). Then for each copy, and each label a of g', a formula $\varphi_a^i(x)$ determines whether the i-th copy x^i of node x of g is present in g' and has label a. Similarly, for each pair of copies and each edge-label γ of g', a formula $\varphi_\gamma^{i,j}(x,y)$ determines whether an edge with label γ is leading from the i-th copy x^i of x to the j-th copy y^j of y.

Definition 6.2.2 An *mso definable function* $f : \mathrm{GR}(\Omega,\Gamma) \to \mathrm{GR}(\Omega',\Gamma')$ is specified by
- a *domain formula* φ_{dom},
- a *constant* $k \geq 1$, *node formulas* $\varphi_a^i(x)$,
- for every $a \in \Omega'$ and every $i \in \{1,\ldots,k\}$, and *edge formulas* $\varphi_\gamma^{i,j}(x,y)$
- for every $\gamma \in \Gamma'$ and all $i,j \in \{1,\ldots,k\}$;

all formulas are in $\mathrm{MSO}(\Omega,\Gamma)$.
For $g \in L(\varphi_{dom})$ with node set V_g, the image $f(g)$ is $(\bigcup_{i\in\{1,\ldots,k\}} V^i, E, \lambda)$,
where for $i \in \{1,\ldots,k\}$,
- $V^i = \{x^i \mid x \in V_g,$ there is exactly one $a \in \Omega'$ such that $g \models \varphi_a^i(x)\}$,
- $E = \{(x^i,\gamma,y^j) \mid x^i \in V^i, y^j \in V^j, i,j \in \{1,\ldots,k\}, g \models \varphi_\gamma^{i,j}(x,y)\}$, and
- $\lambda(x^i) = a$ if $g \models \varphi_a^i(x)$, for $x^i \in V^i$, $i \in \{1,\ldots,k\}$.

Proposition 6.2.3 ([8]) *Let $f : \mathrm{GR}(\Omega,\Gamma) \to \mathrm{GR}(\Omega',\Gamma')$ be an mso definable function. If $L \subseteq \mathrm{GR}(\Omega',\Gamma')$ is mso definable, then $f^{-1}(L) = \{g \in \mathrm{GR}(\Omega,\Gamma) \mid f(g) \in L\}$ is mso definable.*

Proof. Given a formula φ of $\mathrm{MSO}(\Omega',\Gamma')$ we show how to construct a formula φ^{-1} of $\mathrm{MSO}(\Omega,\Gamma)$ such that a graph $g \in \mathrm{GR}(\Omega,\Gamma)$ satisfies φ^{-1} if and only if $f(g) \in \mathrm{GR}(\Omega',\Gamma')$ satisfies φ. From this the proposition follows easily.

For our own convenience we split the construction of φ^{-1} into two steps. We assume (without loss of generality) that φ does not contain universal quantifiers.

first step. In this step we construct a "hybrid" formula in which quantifications range over nodes in g rather than $f(g)$. The properties, however, are still expressed in terms of the images of the nodes in $f(g)$. A subformula of the form $\exists x\psi(x)$ in the original formula φ asserts the existence of a node x in $f(g)$ satisfying ψ. In terms of g this means that it contains a node for which some i-th copy is present in $f(g)$, and satisfies ψ. Similarly a set X of nodes in $f(g)$ can be translated back to k sets of nodes $X_1,\ldots,X_k$ in g, such that $X = \bigcup_{i\in\{1,\ldots,k\}}\{x^i \mid x \in X_i\}$. Now inductively replace in φ any subformula $\exists x\psi(x)$ by the formula $\bigvee_{i\in\{1,\ldots,k\}} \exists x(\mathrm{copy}^i(x) \wedge \psi[x^i/x])$, and any subformula $\exists X\psi(X)$ by the formula $\exists X_1 \cdots \exists X_k \psi(X)$. Here "$\mathrm{copy}^i(x)$" is the formula expressing that the i-th copy of x is defined (i.e., exactly one label predicate $\varphi_a^i(x)$ is true), and $\psi[y/x]$ is the formula obtained from ψ by replacing every occurrence of x by y.

second step. The properties of variables are still expressed in terms of the labels and the edge relations of the image $f(g)$. We obtain the required formula φ^{-1} by changing the atomic formulas as follows. Replace the formula $x^i = y^j$ by $x = y$ if $i = j$, and by false otherwise; replace $x^i \in X$ by $x \in X_i$; replace $\mathrm{lab}_a(x^i)$ by $\varphi_a^i(x)$; and replace $\mathrm{edge}_\gamma(x^i,y^j)$ by $\varphi_\gamma^{i,j}(x,y)$. □

The function *txt* is mso definable, like the translation from shape to 2-structure, as discussed in [20, Section 4.3].

Lemma 6.2.4 *The mapping txt* : $\mathrm{GR}(\Sigma, \{1, \ldots, \mathrm{M}\}) \to \mathrm{GR}(\Delta, \{1, 2\})$*, that assigns to each tree in* F_Σ *the text it represents, is an mso definable function.*

Proof. The text is built starting with a single copy of each node of the tree. Inner nodes are removed (selecting only nodes with labels in Δ) and the edges between a pair of nodes in the text are determined using the label associated to the least common ancestor of these nodes in the tree.

As an mso definable function *txt* is fixed by the domain formula defining F_Σ, constant $k = 1$, node formulas $\varphi_a(x) \equiv \mathrm{lab}_a(x)$, for $a \in \Delta$, and, for $m \in \{1, 2\}$, edge formulas

$$\varphi_m(x, y) \equiv \bigvee_{\pi \in \Pi} \bigvee_{i <^\pi_m j} [\, \exists z \exists x_1 \exists y_1 (\mathrm{lab}_\pi(z) \wedge \mathrm{edge}_i(z, x_1) \wedge \mathrm{edge}_j(z, y_1) \wedge x_1 \prec x \wedge y_1 \prec y)\,]$$

where $x \prec y$ expresses that there is a path from x to y (cf. Example 6.2.1), and $i <^\pi_m j$ abbreviates "i precedes j in the m-th order of π". By our choice of ordering children when viewing the r-shape as a tree, the ordering $<^\pi_1$ equals the usual ordering $<$ on the integers. Note that $i, j \in \{1, \ldots, \mathrm{M}\}$, hence the second disjunction is finite. Additionally we need the finiteness of Π for the first disjunction. □

We now deal with the reverse direction: the construction of a tree representation from a given text. We show that the mapping *sh* that assigns the r-shape to a text is an mso definable function. This basically means, in the terminology of Courcelle [6], that the set of texts is "monadic second-order parsable". Thus we are in a situation like Theorem 4.8 of [6] (although in a somewhat different framework), and we show that mso definability and recognizability are equivalent notions for texts.

To prove our result we have to solve two technical problems. First we have to formulate the structure of the r-shape in terms of mso formulas; this can be done by identifying the nodes of the r-shape with clans of the text. By Proposition 6.1.4, the nodes then are exactly the suffixes of prime clans. Second, we have to translate this into an mso definable function. In particular we need to know which positions in the text have to be duplicated to become the inner nodes of the r-shape.

We start by giving formulas that define the structure of the r-shape of a text.

nodes. A set X is a clan of a text iff it is a segment in both orderings, which can easily be expressed as an mso formula $\mathrm{clan}(X)$. Prime clans are by definition those clans that do not overlap other clans. We obtain the formulas:

$$\mathrm{prim}(X) \equiv \mathrm{clan}(X) \wedge \forall Y [\, \mathrm{clan}(Y) \to (X \cap Y = \emptyset \vee X \subseteq Y \vee Y \subseteq X)\,]$$

$$\mathrm{node}(X) \equiv \mathrm{clan}(X) \wedge \exists Z (\mathrm{prim}(Z) \wedge \mathrm{suff}_1(X, Z))$$

where $\mathrm{suff}_1(X, Z)$ expresses that X is a suffix of Z as segments in the first order.
The usual child-parent relation can be expressed using set inclusion:

$$\mathrm{child}(X, Y) \equiv \mathrm{node}(X) \wedge \mathrm{node}(Y) \wedge X \subseteq Y \wedge \neg \exists Z [\, \mathrm{node}(Z) \wedge X \subset Z \subset Y\,]$$

node-labels. The label of the (inner) node X in the r-shape is determined by the quotient bi-order of X with respect to its children, i.e. by the relative ordering of the children of X in the first and second ordering of the text. Thus, X has the associated bi-order $\pi = ((1,\ldots,n),(i_1,\ldots,i_n))$ if the following predicate, denoted $\mathrm{quot}_\pi(X)$, is satisfied:

$$\exists X_1 \ldots \exists X_n [\, (X = \bigcup_{j=1}^{n} X_j) \wedge \bigwedge_{j=1}^{n} \mathrm{child}(X_j, X) \wedge (X_1 <_1 \ldots <_1 X_n) \wedge (X_{i_1} <_2 \ldots <_2 X_{i_n}) \,]$$

where $X <_i Y$ abbreviates $\forall x \forall y (x \in X \wedge y \in Y \rightarrow \mathrm{edge}_i(x,y))$, meaning that X precedes Y as a segment in the i-th order.

edges. The r-shape has edges connecting nodes to their children as usual in a tree. To order the children of a node, the predicate $\mathrm{child}(X,Y)$ has to be extended to a predicate $\mathrm{child}_k(X,Y)$ meaning that X is the k-th child of Y, which can be done quite similar to the previous predicate quot_π:

$$\begin{aligned} \mathrm{child}_k(X,Y) \;\equiv\; & \exists X_1 \ldots \exists X_{k+1} [\, (Y = \bigcup_{j=1}^{k+1} X_j) \wedge \bigwedge_{j=1}^{k} \mathrm{child}(X_j, X) \\ & \wedge (X_1 <_1 \ldots <_1 X_{k+1}) \wedge (X = X_k) \,] \end{aligned}$$

Here X_{k+1} represents all children after the k-th; this makes this predicate independent of the arity of the node X.

We are now ready to complete the proof.

Theorem 6.2.5 *The mapping* $sh : \mathrm{GR}(\Delta, \{1,2\}) \rightarrow \mathrm{GR}(\Sigma, \{1,\ldots,\mathrm{M}\})$*, that assigns to each text in* $\mathrm{TXT}_\Pi(\Delta)$ *its r-shape, is an mso definable function.*

Proof. We avoid an unelegant duplication in our constructions by first approximating the mapping *sh* by an extension of the domain to the set of all texts over the alphabet Δ. In a second step we show that a proper domain formula, defining $\mathrm{TXT}_\Pi(\Delta)$, exists.

defining the mapping. In the above discussion we have translated the structure of the r-shape into formulas of $\mathrm{MSO}(\Delta, \{1,2\})$. Using these formulas we formalize the construction of the r-shape as an mso definable function. To this end we show how to obtain the nodes of the r-shape (the output graph) by duplicating the positions of the text (i.e., the nodes of the input graph). Each position in the text will be used (at most) twice: one copy will serve as a leaf, the other copy will serve as an inner node of the r-shape.

We associate with each inner node of the r-shape a position in the text or, equivalently, we associate with each inner node a leaf (see also [26]). (Leaves are the singleton clans of the text.) If position x in the text is associated to the inner node X of the r-shape, then the copy x^2 represents X, the first copy x^1 represents the leaf x.

The association we use is known (for binary trees) as the inorder successor of the node: it is the leaf that is found by first moving to the last child, and then repeatedly moving to the

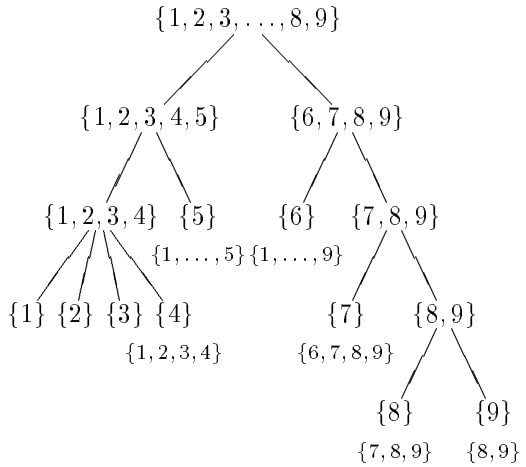


Figure 6.3: associating leaves with inner nodes

first child of the node visited. This gives an injective mapping of inner nodes to leaves of the tree. Moreover, this is important for our result, it can be expressed in an mso formula:

$$\mathrm{assoc}(x,X) \equiv \exists Y[\,\mathrm{frst}_1(x,Y) \wedge \mathrm{child}(Y,X) \wedge \mathrm{suff}_1(Y,X)\,]$$

where $\mathrm{frst}_1(x,Y)$ expresses that x is the first element of Y, and $\mathrm{suff}_1(Y,X)$ expresses that Y is a suffix of X (w.r.t. the first ordering of the text).

As an example, consider the r-shape represented in Figure 6.2 (without node-labels). In Figure 6.3 we have given every inner node below the leaf with which it is associated.

The mapping sh' is specified as an mso definable function as follows:
the domain function φ_{dom} defining $\mathrm{TXT}(\Delta)$, the constant $k = 2$,
for $a \in \Delta$, and $\pi \in \Pi$, the node formulas $\varphi^1_a(x) \equiv \mathrm{lab}_a(x)$, $\varphi^1_\pi(x) \equiv \varphi^2_a(x) \equiv$ false, and $\varphi^2_\pi(x) \equiv \exists X[\,\mathrm{assoc}(x,X) \wedge \mathrm{quot}_\pi(X)\,]$,
for $k \in \{1,\ldots,\mathrm{M}\}$, the edge formulas $\varphi^{1,1}_k(x,y) \equiv \varphi^{1,2}_k(x,y) \equiv$ false,
$\varphi^{2,1}_k(x,y) \equiv \exists X \exists Y[\,\mathrm{assoc}(x,X) \wedge (Y = \{y\}) \wedge \mathrm{child}_k(Y,X)\,]$, and
$\varphi^{2,2}_k(x,y) \equiv \exists X \exists Y[\,\mathrm{assoc}(x,X) \wedge \mathrm{assoc}(y,Y) \wedge \mathrm{child}_k(Y,X)\,]$

defining the domain. The mso definable function sh' constructed above does not have the proper domain. For texts in $\mathrm{TXT}_\Pi(\Delta)$ it satisfies our needs. For other texts in $\mathrm{TXT}(\Delta)$, i.e. those that have quotients not in Π, the labeling formulas are undefined for inner nodes of the r-shape that are labeled by such "illegal" operations. According to the definition of mso definable functions, these nodes are then removed from the constructed graph, leaving an unconnected graph. Hence, sh' constructs a tree if and only if the input graph is an element of $\mathrm{TXT}_\Pi(\Delta)$. Consequently, $\mathrm{TXT}_\Pi(\Delta) = sh'^{-1}(F_\Sigma)$, which, by Proposition 6.2.3, is an mso definable subset of $\mathrm{GR}(\Delta,\{1,2\})$. □

The final conclusion made in the above proof is interesting in its own right.

Corollary 6.2.6 $\mathrm{TXT}_\Pi(\Delta)$ *is an mso definable subset of* $\mathrm{GR}(\Delta,\{1,2\})$.

We are ready to prove the main result of this section. It connects the mso definability of a text language with the mso definability of two tree languages representing these texts.

Theorem 6.2.7 *Let $K \subseteq \mathrm{TXT}_\Pi(\Delta)$ be a text language. The following statements are equivalent:*

(i) *K is mso definable.*
(ii) *$txt^{-1}(K)$ is mso definable.*
(iii) *$sh(K)$ is mso definable.*

Proof.
(i) $\Rightarrow$ (ii). By Lemma 6.2.4, the function txt is mso definable, and thus, by Proposition 6.2.3, if K is mso definable, then $txt^{-1}(K)$ is mso definable.
(ii) $\Rightarrow$ (iii). The mso definable sets are closed under the Boolean operations. This implication follows from the equality $sh(K) = txt^{-1}(K) \cap \mathrm{SH}_\Pi(\Delta)$, and the observation that $\mathrm{SH}_\Pi(\Delta)$ is mso definable.
(iii) $\Rightarrow$ (i). As sh is injective, $K = sh^{-1}(sh(K))$. The implication follows from Theorem 6.2.5 and Proposition 6.2.3. □

The structure of a text interpreted as graph does only depend on properties of the edges, and is independent of node-labels. Using [19, Theorem 6.1] one obtains an operational characterization of the family of mso definable subsets of $\mathrm{TXT}_\Pi(\Delta)$: it is the smallest family containing certain elementary text languages that is closed under intersection, difference, and (node) relabelings. The elementary text languages referred to are $\mathrm{TXT}_\Pi(\Delta)$ itself, as well as each of the languages of texts satisfying a formula of the form

$$\varphi_{a,\gamma,b} \equiv \exists x \exists y [\, \mathrm{lab}_a(x) \wedge \mathrm{lab}_b(y) \wedge \mathrm{edge}_\gamma(x,y) \,]$$

Such a text language consists of all texts in which an a labeled position precedes a b labeled position in the first or second ordering (depending on whether $\gamma = 1$ or $\gamma = 2$).

6.3 Recognizable and right-linear text languages

From the result in Theorem 6.2.7 from the previous section, connecting mso definable text languages with mso definable tree languages, it is only a few steps to obtain a characterization of mso definable text languages in terms of universal algebra and in terms of grammars, which is presented in this section. We first define the algebraic and grammatical notions involved.

We make use of the general definition of recognizability of subsets in a Σ-algebra. For a ranked alphabet Σ, a Σ-*algebra* $\mathcal{A}$ is a pair (A, Σ), where A is a set and each operator $\sigma \in \Sigma$ of rank $m \geq 0$ defines a mapping $\sigma^{\mathcal{A}} : A^m \to A$ (see, e.g., [2]). The term algebra formed by the set of Σ-terms is denoted by $\mathcal{F}_\Sigma = (F_\Sigma, \Sigma)$.

For a Σ-algebra $\mathcal{A} = (A, \Sigma)$, a subset $K \subseteq A$ is *recognizable* if there is a finite Σ-algebra $\mathcal{Q} = (Q, \Sigma)$, a homomorphism $h : \mathcal{A} \to \mathcal{Q}$, and a subset $F \subseteq Q$ such that $h^{-1}(F) = K$.

This definition applies to tree languages in F_Σ, being subsets of the Σ-algebra of terms $\mathcal{F}_\Sigma$; the finite algebra $\mathcal{Q}$ corresponds with a so-called (deterministic) bottom-up tree recognizer (see [21]).

To obtain an algebraic structure on texts we let $\Sigma = \Pi \cup \Delta$ be a ranked alphabet as before: the Σ-algebra $\mathcal{T}_\Sigma = (\mathrm{TXT}_\Pi(\Delta), \Sigma)$ is defined by $a^{\mathcal{T}_\Sigma} = (a, (1))$ for $a \in \Delta$, and for $\pi \in \Pi$ of rank m, $\tau_1, \ldots, \tau_m \in \mathrm{TXT}_\Pi(\Delta)$, $\pi^{\mathcal{T}_\Sigma}(\tau_1, \ldots, \tau_m) = [\pi \leftarrow (\tau_1, \ldots, \tau_m)]$. This leads to the following definition of recognizability for text languages.

Definition 6.3.1 A text language $K \subseteq \mathrm{TXT}_\Pi(\Delta)$ is *recognizable* if there is a finite Σ-algebra $\mathcal{Q} = (Q, \Sigma)$ (where $\Sigma = \Pi \cup \Delta$), a homomorphism $h : \mathcal{T}_\Sigma \to \mathcal{Q}$, and a subset $F \subseteq Q$ such that $h^{-1}(F) = K$.

Note that the mapping $txt : F_\Sigma \to \mathrm{TXT}_\Pi(\Delta)$ which assigns to each $t \in F_\Sigma$ the text $txt(t)$ is precisely the homomorphism evaluating terms of $\mathcal{F}_\Sigma$ in $\mathcal{T}_\Sigma$. As txt is surjective we may apply the more general result [2, Theorem 3.4] to the algebra $\mathcal{T}_\Sigma$.

Proposition 6.3.2 *Let $K \subseteq \mathrm{TXT}_\Pi(\Delta)$ be a text language. K is recognizable in $\mathcal{T}_\Sigma$ iff $txt^{-1}(K)$ is recognizable in $\mathcal{F}_\Sigma$.*

Now we turn to grammars defining tree and text languages. The notion of (singular) substitution of trees and texts forms the basis for the derivation in tree grammars and text grammars.

Let Σ be an arbitrary ranked alphabet, and let $t, u \in F_\Sigma$ be trees. By substituting u for the k-th leaf of t, we again obtain a tree t' in F_Σ. Writing trees as words over $\Sigma \cup \{\langle, \rangle\}$, t' is the word obtained from t by replacing the label of the k-th leaf in t by the word u. We denote the tree t' by $t[k \leftarrow u]$.

Similarly we consider text substitution. Let $\tau, \nu \in \mathrm{TXT}_\Pi(\Delta)$, and let k be a natural number not greater than the length of τ. Then $\tau[k \leftarrow \nu]$ is the text where the k-th element of the domain of τ is replaced by the text ν, and the orderings and labels are inherited in the obvious way (cf. simultaneous substitution as defined before: if τ has underlying bi-order π and word $a_1 \cdots a_m$, then $\tau[k \leftarrow \nu]$ may be defined as $[\pi \leftarrow (\tau_1, \ldots, \tau_m)]$, where $\tau_k = \nu$ and for $j \neq k$, τ_j is the singleton text with word a_j).

In particular, if $\Sigma = \Pi \cup \Delta$ is a ranked alphabet for primitive representations, then the substitutions in F_Σ and $\mathrm{TXT}_\Pi(\Delta)$ are compatible in the sense that $txt(t[k \leftarrow u]) = txt(t)[k \leftarrow txt(u)]$ (this is a consequence of the fact that txt is the homomorphism evaluating terms in $\mathcal{T}_\Sigma$).

A *regular tree grammar* is a 4-tuple $G = (N, \Sigma, P, S)$, where N is a set of nonterminal symbols, Σ is a ranked alphabet, P consists of productions of the form $A \to t$, where $A \in N$ and $t \in F_{\Sigma \cup N}$ (interpreting the nonterminals as operators of rank 0), and $S \in N$. A tree $t' \in F_{\Sigma \cup N}$ is derived from a tree $t \in F_{\Sigma \cup N}$, denoted $t \Rightarrow_G t'$, if there is a production $A \to u$ in P, and a natural number k such that the k-th leaf of t has label A, and $t' = t[k \leftarrow u]$. The *tree language generated by* G is the set of trees $\{t \in F_\Sigma \mid S \Rightarrow_G^+ t\}$; such a tree language is called *regular*. Each regular tree language is generated by a regular tree grammar *in normal form*, i.e., a grammar such that every production is of the form $A \to \sigma\langle B_1 \cdots B_m \rangle$ with $\sigma \in \Sigma$ of rank $m \geq 0$, and $B_1, \ldots, B_m$ nonterminals.

For texts we have the notion of *context-free text grammar* (see [12]), which is a 4-tuple $G = (N, \Delta, P, \tau_0)$, where N is the alphabet of nonterminals, Δ is the alphabet of terminals,

P is a finite set of productions $A \to \tau$, where $A \in N$ and τ is a text over $N \cup \Delta$, and τ_0 is a text of length 1 over N.

For τ, $\tau' \in \mathrm{TXT}(N \cup \Delta)$, τ derives τ' (in G), denoted $\tau \Rightarrow_G \tau'$, if there is a production $A \to \nu \in P$ such that the k-th letter of the word of τ is A, and $\tau' = \tau[k \leftarrow \nu]$.

Example 6.3.3 Consider the text grammar $G = (\{S, A\}, \{a, b, c\}, P, (S, (1)))$ such that P consists of the productions $S \to (Ab, (1, 2))$, $A \to (AA, (2, 1))$, and $A \to (abc, (1, 3, 2))$. Here is an example of a derivation in G : $(S, (1)) \Rightarrow_G (Ab, (1, 2)) \Rightarrow_G (AAb, (2, 1, 3)) \Rightarrow_G (Aabcb, (2, 4, 3, 1, 5)) \Rightarrow_G (abcabcb, (4, 6, 5, 1, 3, 2, 7))$. □

For a context-free text grammar $G = (N, \Delta, P, \tau_0)$, the text language generated by G is $\{\tau \in \mathrm{TXT}(\Delta) \mid \tau_0 \Rightarrow_G^* \tau\}$; such a text language is called *context-free*. Since G has finitely many productions, there is a finite set Π of primitive bi-orders such that all right-hand sides are in $\mathrm{TXT}_\Pi(\Delta \cup N)$; then also the generated language is included in $\mathrm{TXT}_\Pi(\Delta)$.

The notion of a context-free text grammar is too powerful for our purpose of characterizing recognizable text languages. E.g., the text language $K = \{(a^n b^n, (1, 2, \ldots, 2n)) \mid n \geq 1\}$ is a context-free text language (just as the corresponding word language), but $sh(K)$ is not regular.

Decomposing right-hand sides of productions into primitive texts, one obtains for each context-free text grammar an equivalent grammar in *primitive normal form*, i.e., for every production $A \to \tau$, τ is primitive, and the word of τ either is a terminal or it is of length ≥ 2 and consists of nonterminals. Such a text grammar can be translated to a regular tree grammar in normal form, which generates primitive representations. We now define text grammars such that the corresponding tree grammar generates precisely r-shapes.

Definition 6.3.4
(1) A context-free text grammar $G = (N, \Delta, P, \tau_0)$ is *right-linear* if G is in primitive normal form and for each production $A \to (BC, (i_1, i_2)) \in P$, with $A, B, C \in N$, and $\{i_1, i_2\} = \{1, 2\}$, if $B \to (w, (j_1, j_2)) \in P$, then $(j_1, j_2) \neq (i_1, i_2)$.
(2) A text language is *right-linear* if it is generated by a right-linear grammar.

From the compatibility of substitution for texts and for trees, and from the observation that a right-linear text grammar corresponds with a regular tree grammar in normal form that generates r-shapes, we obtain the desired result.

Theorem 6.3.5 *Let $K \subseteq \mathrm{TXT}_\Pi(\Delta)$ be a text language. K is right-linear iff $sh(K)$ is regular.*

Note that if both $\sigma_f, \sigma_b \notin \Pi$, then every context-free text grammar in primitive normal form is right-linear. Hence in that case the families of context-free text languages and of right-linear text languages over $\mathrm{TXT}_\Pi(\Delta)$ coincide.

The well-known equivalences for tree languages given in the next proposition yield similar equivalences for the case of texts.

Proposition 6.3.6 ([9, 25, 30]) *Let $T \subseteq F_\Sigma$ be a tree language. T is mso definable iff T is recognizable iff T is regular.*

Theorem 6.3.7 *Let $K \subseteq \mathrm{TXT}_\Pi(\Delta)$ be a text language. The following statements are equivalent:*

(i) *K is mso definable.*
(ii) *K is recognizable.*
(iii) *K is right-linear.*

Proof. By Proposition 6.3.6, $txt^{-1}(K)$ is mso-definable iff it is recognizable, and $sh(K)$ is mso-definable iff it is regular. Hence, the equivalence of (i) and (ii) follows from Theorem 6.2.7 and Proposition 6.3.2, and the equivalence of (ii) and (iii) follows from Theorem 6.2.7 and Theorem 6.3.5. □

An independent proof of the latter equivalence is given in [24], where for a right-linear text grammar a finite Σ-algebra recognizing its language is directly constructed.

6.4 Discussion

Finally, we discuss how the main result (more precisely the equivalence of (i) and (ii) in Theorem 6.2.7) can be extended to a larger class of graphs than our specific texts. (A similar approach has been taken in [7]). In particular we will sketch how the main equivalences from Section 6.2 can be generalized.

For our constructions we first assume that we are dealing with graphs having the property that for every pair x, y of nodes there is precisely one edge from x to y. These graphs are also known as *2-structures*, see [13]. Every loop-free graph in $\mathrm{GR}(\Delta, \Gamma)$ can be translated to a 2-structure, by replacing all edges from x to y, say with labels $\gamma_1, \ldots, \gamma_k$, by one edge with label $\{\gamma_1, \ldots, \gamma_k\}$ which yields a graph in $\mathrm{GR}(\Delta, 2^\Gamma)$ which is a 2-structure. Since this translation is mso definable (in both directions) we can then extend the result obtained for 2-structures to loop-free graphs with multiple edges.

Like for texts, for 2-structures we have modular decomposition based on clans. In fact the decomposition results for texts that we have used in this paper stem from the translation of a text (in its interpretation as a graph) into a 2-structure. (For readers familiar with [16], in that paper the labels VH, $V\overline{H}$, $\overline{V}H$, $\overline{VH}$ of a particular 2-structure represent the set labels $\{1,2\}$, $\{1\}$, $\{2\}$, and $\emptyset$, respectively, in our 2-structure interpretation of a text.)

A *clan* of a 2-structure is a subset of nodes X such that for every node z outside X, all edges from z into X have the same label, and all edges from X to z have the same label.

Example 6.4.1 Consider the graph g in Figure 6.4. The undirected edges with label A stand for directed edges in two directions, both with label A. Hence g is a 2-structure. The non-trivial clans of g are : $\{1,4,5\}$, $\{1,5\}$, $\{4,5\}$, $\{1,2,4,5\}$, $\{1,3,4,5\}$, $\{2,3\}$. E.g., $\{1,4\}$ is *not* a clan, since the edge from 5 to 1 has label C, and the edge from 5 to 4 has label B. □

Each 2-structure can be decomposed into primitive substructures (i.e., substructures without non-trivial clans) yielding a primitive representation. The 2-structure represented by a primitive representation is obtained by bottom up substitution into the primitive 2-structures at its inner nodes.

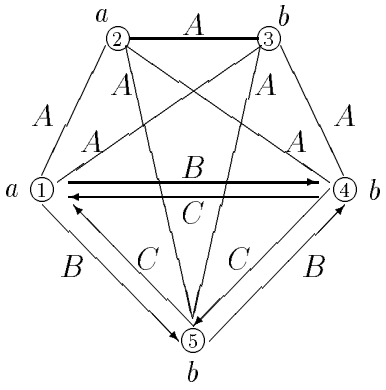


Figure 6.4: a 2-structure g

Example 6.4.2 In Figure 6.5 two primitive representations for the 2-structure g from Example 6.4.1 are given. □

Primitive representations of texts are *ordered* (node-labeled) trees in a natural way – the ordering is induced by the first order of the text. Although for 2-structures this does not come so naturally, we also wish to view primitive representations as ordered trees, to let them correspond with terms over a ranked alphabet.

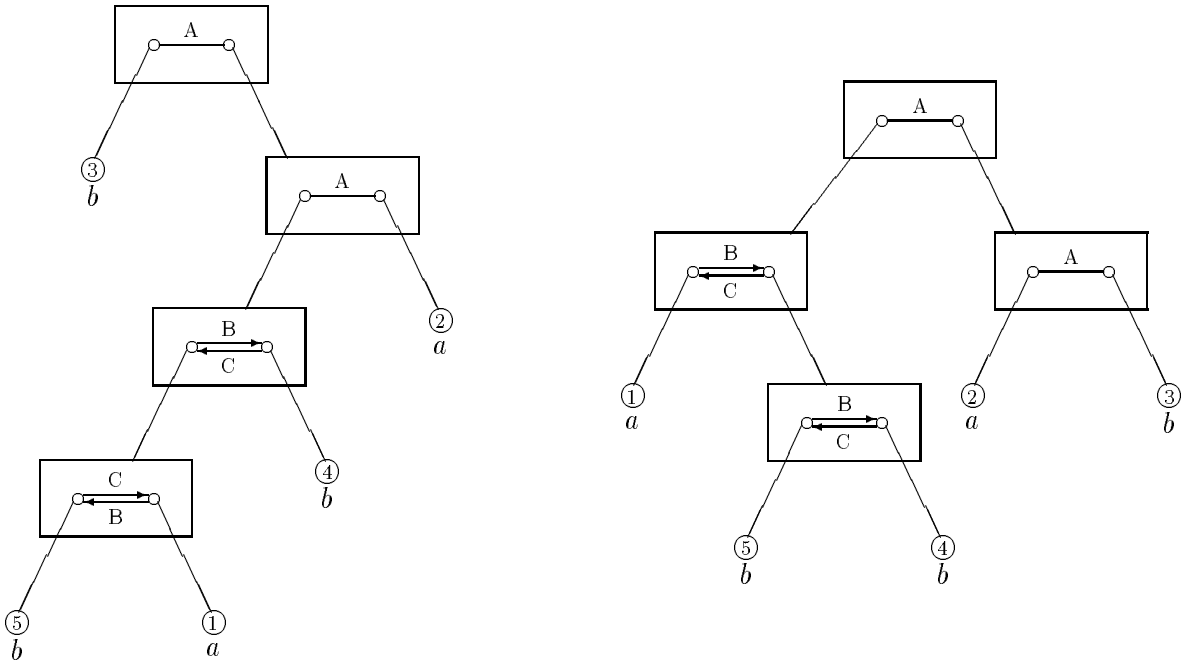


Figure 6.5: two primitive representations for g

The operator symbols in such a ranked alphabet Σ are node-labels defining constants, and primitive 2-structures the rank of which equals their number of nodes. A Σ-algebra of 2-structures is defined by associating with an operator symbol σ of rank $m > 1$ the operation which substitutes an m-tuple of 2-structures into the m nodes of σ. Therefore, we assume that the nodes of the operator symbol σ are ordered by numbering them $\{1, \ldots, m\}$. We use $\sigma_{A,B}$ to denote the primitive 2-structure of rank 2 such that the edge from node 1 to node 2 has label A and the reverse edge has label B.

Example 6.4.3 Consider the primitive representations of Figure 6.5. Let the ordering in these trees be the left-to-right ordering. Consequently, one should give to the left node of each 2-structure in the tree number 1, and to the right node number 2. Hence the left tree corresponds with the term $\sigma_{A,A}\langle b\sigma_{A,A}\langle\sigma_{B,C}\langle\sigma_{C,B}\langle ba\rangle b\rangle a\rangle\rangle$, and the right tree corresponds with the term $\sigma_{A,A}\langle\sigma_{B,C}\langle a\sigma_{B,C}\langle bb\rangle\rangle\sigma_{A,A}\langle ab\rangle\rangle$. □

6.4.1 The r-shape of a 2-structure

Essential in the considerations of Section 6.2 is the existence of a natural tree representation of a text, the r-shape, that can be constructed from the text itself. For a generalization to 2-structures we have to choose an r-shape among the possible primitive representations of a 2-structure.

For 2-structures there are more variations in the form of their primitive representations than in the case of texts (Proposition 6.1.3). However, following from decomposition results for 2-structures (cf. [12, Lemma 4.1.19]), this form is still restricted. The first freedom in choosing a primitive representation is caused by the associativity of operations of rank 2 in the Σ-algebra of 2-structures. Additionally, we may permute the children of a given node in the tree representation if we permute the nodes of the associated primitive 2-structure accordingly.

We discuss these two cases.

associativity. Like in the case of texts any binary operator is associative. This means that a 2-structure built by using the operator $\sigma_{A,B}$ several consecutive times, has many different binary decompositions. This can be solved as before: we choose a rightmost tree representation.

permutations and symmetries. The terms $\sigma_{A,B}\langle ab\rangle$ and $\sigma_{B,A}\langle ba\rangle$ represent the same 2-structure. More generally, if σ is an operator of rank $m > 1$, and π is a permutation of $1, \ldots, m$, then

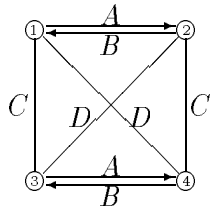


Figure 6.6: Primitive 2-structure with symmetries

$\sigma\langle t_1, \ldots, t_m\rangle$ and $\pi(\sigma)\langle t_{\pi^{-1}(1)}, \ldots, t_{\pi^{-1}(m)}\rangle$ represent the same 2-structure, where $\pi(\sigma)$ is obtained from σ by renumbering the nodes of the graph associated with σ according to the permutation π.

It is important to note that it is not possible to fix the r-shape by allowing for each primitive 2-structure precisely one numbering of its nodes. This is due to possible internal symmetries of the primitive 2-structures, i.e., situations when $\sigma = \pi(\sigma)$. For an operator $\sigma_{A,B}$ of rank 2 this is the case when $A = B$: $\sigma_{A,A}$ is commutative. Similar (but less obvious) symmetries may occur in primitive 2-structures of higher rank, such as in Figure 6.6. For this operator σ of rank 4, terms $\sigma\langle t_1 t_2 t_3 t_4\rangle$ and $\sigma\langle t_3 t_4 t_1 t_2\rangle$ represent the same 2-structure for any choice of t_1, t_2, t_3, t_4.

In [14] the *shape* of a 2-structure is an unordered tree. Choosing one of the permutations of the nodes of an operator in the r-shape corresponds to fixing the order of the children of a node in the original shape.

To choose an ordering we will assume that we deal with 2-structures the nodes of which can be linearly ordered. This "external" linear ordering on the nodes will then be used to determine the "internal" linear ordering on the children of nodes in the shape.

More precisely, we let $\varphi_{lin}(x, y)$ be some fixed mso formula with two free variables x and y, and consider 2-structures for which φ_{lin} determines a linear order on the nodes. (This generalizes the case of texts, where φ_{lin} is the formula edge_1 which defines the first ordering of the text.) There is however one important exception to this: for operators $\sigma_{A,B}$, with $A \neq B$, we choose the ordering beforehand, by preferring one of $\sigma_{A,B}$ and $\sigma_{B,A}$ above the other. (The reason is that we will then be able to give a neat characterization of the nodes in the r-shape.)

These matters necessitate the following two requirements of the ranked alphabet Σ:

- for every pair A, B (with $A \neq B$) at most one of the operators $\sigma_{A,B}$ and $\sigma_{B,A}$ is an element of Σ,
- if σ is a primitive 2-structure of rank > 2 in Σ, then every operator obtained by permuting the nodes of σ is also in Σ.

Then, formally, the *r-shape* of a 2-structure is the primitive representation such that

- it has no subtrees of the form $\sigma_{A,B}\langle\sigma_{A,B}\langle t_1 t_2\rangle t_3\rangle$ for any edge-labels A, B (where possibly $A = B$),
- for every inner node corresponding to an operator that is not of the form $\sigma_{A,B}$ with $A \neq B$, if a x is the i-th child and y is the j-th child, with $i < j$, then there is a leaf x' descending from x such that for every leaf y' descending from y, $\varphi_{lin}(x', y')$ holds.

Example 6.4.4 Consider once more the 2-structure g from Example 6.4.1. Suppose that the ranked alphabet contains $\sigma_{B,C}$ (hence not $\sigma_{C,B}$), and that φ_{lin} determines the linear order $(1, 2, 3, 4, 5)$ on the nodes of g (e.g., $\varphi_{lin}(x, y)$ may specify that nodes with label a precede nodes with label b, and that nodes with label b with adjacent C-edges are ordered according to these edges; finally the two nodes with only adjacent A-edges are placed in the ordering).

The right tree in Figure 6.5 is the r-shape of g. The substructure of g on the nodes $\{1, 4, 5\}$ is built using the operator $\sigma_{B,C}$. Note that its nodes are ordered $(1, 5, 4)$, following

the decomposition using this operator, whereas with respect to the "external" ordering φ_{lin} these nodes are ordered $(1, 4, 5)$. □

We first indicate how to generalize the proof of Theorem 6.2.5.

With our definition above, the nodes of the r-shape of a 2-structure correspond with clans X of the 2-structure that are characterized as follows:

- either X is a prime clan,
- or X is a clan such that there exists a prime clan $Z \supset X$, and an operator $\sigma_{A,B}$ in the ranked alphabet Σ such that for all y in $Z - X$ and all x in X, $\mathrm{edge}_A(y, x)$ and $\mathrm{edge}_B(x, y)$ hold, and if $A = B$, then there exists $y \in Z - X$ such that $\varphi_{lin}(y, x)$ for every $x \in X$,

Example 6.4.5 For the 2-structure g from Example 6.4.1, $\{1, 4, 5\}$ is the only non-trivial prime clan, and $\{4, 5\}$ and $\{2, 3\}$ satisfy the second condition above, where the role of Z is played by $\{1, 4, 5\}$ and $\{1, 2, 3, 4, 5\}$, respectively. □

By this characterization there is an mso formula $\mathrm{node}(X)$ expressing that the clan X is a node in the r-shape. Also, there are mso formulas giving the 2-structure label of each node X in the r-shape, and expressing whether X is the k-th child of Y (where in both formulas φ_{lin} occurs). Hence, as for texts, the r-shape of a 2-structure can be specified in terms of the 2-structure itself (and φ_{lin}). Using this, and the association of leaves and inner nodes as in the proof of Theorem 6.2.5 it can be shown that the function that assigns to each 2-structure its r-shape is mso definable, analogously to the case of texts.

In the other direction, cf. Lemma 6.2.4, the function $2s$ that assigns to each primitive representation the 2-structure it represents is mso definable just as txt was, see [20].

Finally, in order to complete the proof of Theorem 6.2.7 for 2-structures, it remains to be shown that the property of being a r-shape is expressible in $\mathrm{MSO}(\Sigma, \{1, \ldots, \mathrm{M}\})$. The exclusion of certain binary, non-rightmost, subtrees is expressible in an mso formula (as it was for texts). Moreover, we have to check for certain leaves x, y whether $\varphi_{lin}(x, y)$ holds. For this we need to express this property in terms of the tree, i.e., in $\mathrm{MSO}(\Sigma, \{1, \ldots, \mathrm{M}\})$ (where M is the maximal number of nodes of a 2-structure in Σ). Since the function $2s$ is mso definable, it follows analogously to the proof of Theorem 6.2.3 that there exists a formula φ_{lin}^{-1} in $\mathrm{MSO}(\Sigma, \{1, \ldots, \mathrm{M}\})$ such that $\varphi_{lin}^{-1}(x, y)$ holds for leaves of t iff $\varphi_{lin}(x, y)$ holds for nodes of $2s(t)$. (Note that in order to obtain $\varphi_{lin}^{-1}(x, y)$ from $\varphi_{lin}(x, y)$ one has to replace occurrences of the atomic formula $\mathrm{edge}_A(u, v)$ by a formula that refers to the least common ancestor of u and v).

Summarizing, we have the following proposition, generalizing Theorem 6.2.7.

For a set K of 2-structures, all linearly ordered by φ_{lin} and built from Σ, K is mso definable iff the set of all primitive representations $2s^{-1}(K)$ is mso definable iff the set of r-shapes of K is mso definable.

We turn from the logical viewpoint to the algebraic one. With respect to the Σ-algebra of 2-structures, the mapping $2s$ is the homomorphism evaluating terms into 2-structures. Hence K is recognizable iff $2s^{-1}(K)$ is a recognizable tree language (cf. Proposition 6.3.2). This implies that K is mso definable if and only if it is recognizable.

We can extend this result to graphs with multiple edges in an obvious way. We start with a ranked alphabet Σ consisting of primitive 2-structures with edge-labels in 2^Γ. Then the terms over this ranked alphabet represent 2-structures with edge-labels in 2^Γ which stand for graphs in $\mathrm{GR}(\Delta, \Gamma)$ with multiple edges. In this way we can define a Σ-algebra on such graphs. The mapping *graph* which gives for each primitive representation the graph it represents is the homomorphism evaluating terms into graphs. Also, *graph* can be obtained as the composition of mso definable functions, and hence is mso definable. Analogously, the function that that assigns to each graph its r-shape is mso definable.

Consequently, a set of (loop-free) graphs in $\mathrm{GR}(\Delta, \Gamma)$ linearly ordered by some φ_{lin} and built from a fixed ranked alphabet Σ of primitive 2-structures is mso definable iff it is recognizable.

We end by stressing the two limitations of this generalization to graphs. First we restrict ourselves to graphs that can be linearly ordered. Second, we have a finite alphabet of graph operators, and consequently we look only at graph languages that have such a finite number of primitive building blocks.

Acknowledgement

We thank Joost Engelfriet for many stimulating discussions.

Bibliography of Part II

[1] J.R. Büchi, Weak second-order arithmetic and finite automata, *Zeitschrift für Mathematik, Logik und Grundlagen der Mathematik* **6** (1960) 66–92.

[2] P.M. Cohn, *Universal Algebra*, Harper & Row, New York, 1965.

[3] B. Courcelle, Equivalences and transformations of regular systems; applications to recursive program schemes and grammars, *Theoretical Computer Science* **42** (1986) 1–122.

[4] B. Courcelle, An axiomatic definition of context-free rewriting and its application to NLC graph grammars, *Theoretical Computer Science* **55** (1988) 141–181.

[5] B. Courcelle, On recognizable sets and tree automata, in *Resolution of Equations in Algebraic Structures, Vol 1*, H. Ait-Kaci and M. Nivat, eds., Academic Press, New York, 1989.

[6] B. Courcelle, The monadic second-order logic of graphs V: on closing the gap between definability and recognizability, *Theoretical Computer Science* **80** (1991) 153–202.

[7] B. Courcelle, The monadic second-order logic of graphs X: linear orderings, Technical Report, Bordeaux, 1994.

[8] B. Courcelle, Monadic second-order definable graph transductions, *Lecture Notes in Computer Science* **581** (1992) 124–144.

[9] J. Doner, Tree acceptors and some of their applications, *Journal of Computer and System Sciences* **4** (1970) 406–451.

[10] A. Ehrenfeucht, H.J. Hoogeboom, P. ten Pas, and G. Rozenberg, An introduction to context-free text grammars, in *Developments in Language Theory*, G. Rozenberg and A. Salomaa, eds., World Scientific Publishing, 1994.

[11] A. Ehrenfeucht, P. ten Pas, and G. Rozenberg, Combinatorial properties of texts, *RAIRO, Theoretical Informatics and Applications* **27** (1993) 433–464.

[12] Chapter 4 of this thesis, published as:

A. Ehrenfeucht, P. ten Pas, and G. Rozenberg, Context-free text grammars, *Acta Informatica* **31** (1994) 161–206.

[13] A. Ehrenfeucht and G. Rozenberg, Theory of 2-structures, Part I: clans, basic subclasses, and morphisms, *Theoretical Computer Science* **70** (1990) 277–303.

[14] A. Ehrenfeucht and G. Rozenberg, Theory of 2-structures, Part II: representation through labeled tree families, *Theoretical Computer Science* **70** (1990) 305–342.

[15] A. Ehrenfeucht and G. Rozenberg, Angular 2-structures, *Theoretical Computer Science* **92** (1992) 227–248.

[16] A. Ehrenfeucht and G. Rozenberg, T-structures, T-functions, and texts, *Theoretical Computer Science* **116** (1993) 227–290.

[17] H. Ehrig, H.-J. Kreowski, and G. Rozenberg, eds., *Graph Grammars and their Application to Computer Science*, Lecture Notes in Computer Science 532, Springer, Berlin, 1990.

[18] J. Engelfriet, A characterization of context-free NCE graph languages by monadic second-order logic on trees, *Lecture Notes in Computer Science* **532** (1991) 311–327.

[19] J. Engelfriet, A regular characterization of graph languages definable in monadic second-order, *Theoretical Computer Science* **88** (1991) 139–150.

[20] J. Engelfriet, T. Harju, A. Proskurowski, and G. Rozenberg, Characterization and complexity of uniformly non-primitive labeled 2-structures, Technical Report 94-31 (1994), Leiden University.

[21] F. Gécseg and M. Steinby, *Tree Automata*, Akadémiai Kiadó, Budapest, 1984.

[22] R.L. Graham, B.L. Rothschild, and J.H. Spencer, *Ramsey Theory*, Wiley, New York, 1980.

[23] H.J. Hoogeboom and P. ten Pas, MSO definable text languages, *Lecture Notes in Computer Science* **841** (1994) 413–422.

[24] Chapter 5 of this thesis:

H.J. Hoogeboom and P. ten Pas, Text languages in an algebraic framework, to appear in *Fundamenta Informaticae*.

[25] J. Mezei and J.B. Wright, Algebraic automata and context-free sets, *Information and Control* **11** (1967) 3–29

[26] A. Potthoff and W. Thomas, Regular tree languages without unary symbols are star-free, *Lecture Notes in Computer Science* **710** (1993) 396–405.

[27] W.C. Rounds, Context free grammars on trees, *Proceedings ACM STOC 1969* (1969) 143–148.

[28] A. Salomaa, *Formal Languages*, Academic Press, New York and London, 1973.

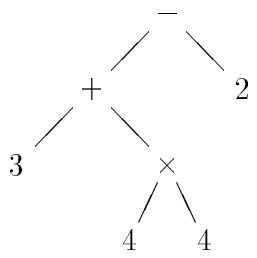
[29] M. Steinby, Some algebraic aspects of recognizability and rationality, *Lecture Notes in Computer Science* **117** (1981) 360–372.

[30] J.W. Thatcher, J.B. Wright, Generalized finite automata theory with an application to a decision problem of second-order logic, *Mathematical Systems Theory* **2** (1968) 57–82.

Samenvatting

Deze samenvatting is met name bedoeld voor lezers zonder achtergrond in de informatica. Eerst worden enkele elementaire begrippen die een belangrijke rol spelen in dit proefschrift verduidelijkt. Vervolgens worden de resultaten uit het proefschrift kort weergegeven.

De twee onderwerpen die in dit proefschrift behandeld worden hebben beide betrekking op boomstructuren. Een boomstructuur, of kortweg '*boom*', geeft een hiërarchische ordening van informatie. Een bekend alledaags voorbeeld is een familie-stamboom. Het belang van bomen binnen de informatica is heel algemeen gesteld dat een boom kan weergeven hoe een probleem is opgebouwd uit deelproblemen. Het oplossen van het probleem gebeurt dan door stapsgewijs oplossingen van deelproblemen te combineren. Een heel eenvoudig voorbeeld hiervan is de berekening die in de volgende boom is weergegeven (volgens gebruik is de boom "ondersteboven" getekend).



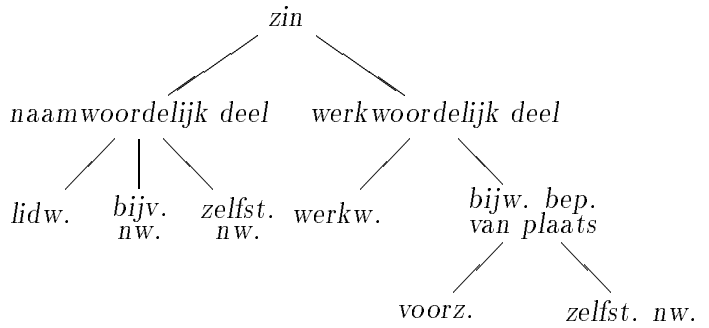
Deze berekening kan ook in een meer "platte" vorm weergegeven worden: $(3 + (4 \times 4)) - 2$. Zo'n expressie is een voorbeeld van de meest gebruikte manier om informatie te noteren: informatie-eenheden worden door symbolen weergegeven (bijvoorbeeld haakjes, cijfers, letters, etc.) en deze symbolen worden in een rij gerangschikt. Zo'n rijtje symbolen wordt een '*woord*' genoemd. Dus ook 'bytes' (rijtjes van nullen en énen), of instructies van een programmeertaal zijn 'woorden'.

Ook in het gebruik van een natuurlijke taal zoals het Nederlands is de informatie op deze manier gerepresenteerd: een Nederlands woord is een rijtje van letters uit het alfabet. Bovendien kunnen ook Nederlandse woorden zelf als symbolen opgevat worden en dan heeft een zin ook de vorm van een 'woord'.

In het algemeen heeft niet iedere rij van symbolen een zinvolle betekenis. Bijvoorbeeld niet ieder rijtje van haakjes, cijfers, en $+$, $\times$, $-$ stelt een rekenkundige expressie voor en slechts bij hoge uitzondering vormt een willekeurige rij Nederlandse woorden een correcte Nederlandse zin. Kortom, alleen woorden met een bepaalde onderliggende structuur representeren informatie.

Een eerste stap in het "begrijpen" van informatie die gegeven is door een woord is daarom het aanbrengen van structuur in het woord. In het bovenstaande voorbeeld van een rekenkundige expressie geeft de getekende boom de structuur van de expressie. Onder het "begrijpen" van zo'n expressie kan men het uitrekenen van de eindwaarde verstaan.

In een zin uit een natuurlijke taal vindt men structuur door te ontleden. Bijvoorbeeld de zin "De fraaie wilg staat in Winterswijk" heeft de volgende grammaticale structuur:



Deze structuur is dus weer een boomstructuur.

Bij het vinden van de (boom)structuur bij deze voorbeelden wordt gebruik gemaakt van bekende algemene regels over de vorm van een rekenkundige expressie danwel een Nederlandse zin, bijvoorbeeld de regel dat een zin bestaat uit een naamwoordelijk en een werkwoordelijk deel. Dit idee is geformaliseerd in één van de gangbare modellen om verzamelingen van woorden te definiëren: de zogeheten '*context-vrije grammatica*'. Een context-vrije grammatica bestaat uit een (eindige) verzameling van symbolen en een stelsel van regels. Met behulp van die regels kunnen woorden, d.w.z. rijtjes van de gegeven symbolen, worden afgeleid. Omgekeerd kunnen ook woorden met behulp van de regels worden herkend, net zoals de zin "De fraaie wilg staat in Winterswijk" herkend wordt als Nederlandse zin met behulp van grammaticale regels. Bij het herkennen van een woord met behulp van de regels van een context-vrije grammatica ontstaat een boom. Zo'n boom heet een 'ontledingsboom'.

In dit proefschrift ligt de nadruk op situaties waarbij een ontledingsboom bij een gegeven woord geconstrueerd kan worden met behulp van andere middelen dan de regels van de grammatica. In Deel I wordt hierbij gebruik gemaakt van een systeem om bomen te coderen. De benadering in Deel II is anders: hier worden objecten bekeken die van zichzelf al meer structuur hebben dan woorden.

Coderen van bomen houdt in dat aan iedere boom precies één woord toegekend wordt. Bijvoorbeeld het woord $(3+(4\times4))-2$ is op te vatten als de eerstgetekende boom in gecodeerde vorm.

Een goede codering moet aan de volgende eisen voldoen: uit een woord dat aan een gegeven boom is toegekend kan de oorspronkelijke boom eenvoudig teruggevonden worden (m.a.w., er is een eenvoudige manier om te 'decoderen') en aan twee verschillende bomen mag niet eenzelfde woord toegekend worden (m.a.w., de codering is 'één-op-één'). Bovendien hebben de coderingen in dit proefschrift de volgende bijzondere eigenschap: het aantal symbolen van het woord waarin een boom gecodeerd wordt is gelijk aan het aantal bladeren van die boom.

Bladeren zijn de plaatsen in de boom zonder vertakking naar beneden (want bomen worden ondersteboven getekend). Merk op dat het eerder gegeven voorbeeld van de rekenkundige expressie hier niet aan voldoet: het aantal symbolen in het woord $(3 + (4 \times 4)) - 2$ is 11 (beide voorkomens van het cijfer 4 tellen mee), en het aantal bladeren in de boom is 4.

In het eerste hoofdstuk worden voorwaarden aan coderingen opgesteld die voldoende zijn om te bewerkstelligen dat er een eenvoudige decoderingsmethode bestaat én dat de codering 'één-op-één' is. In eerste instantie worden coderingen gezocht voor "kale" bomen, dat wil zeggen, bomen zonder extra informatie op de vertakkingspunten. Het is een tamelijk verrassend feit dat er goede coderingen bestaan waarvoor slechts zes verschillende symbolen nodig zijn. Dus bijvoorbeeld de letters a, b, c, d, e, f volstaan: iedere (kale) boom kan gecodeerd worden als een rijtje van deze letters.

Soortgelijke coderingen bestaan voor bomen waarvan iedere vertakking een vertakking in precies twee takken is (zoals ook in de eerste hierboven getekende boom het geval is). Deze worden in een deel van het eerste en in het tweede hoofdstuk onderzocht. In dit geval zijn vier symbolen genoeg.

In het derde hoofdstuk worden de gevonden coderingen gebruikt om een speciale klasse van context-vrije grammatica's te definiëren. Uit een woord dat door zo'n grammatica gegenereerd wordt kan heel makkelijk een bijbehorende ontledingsboom geconstrueerd worden met behulp van de decoderingsmethode. Daardoor zijn de context-vrije grammatica's van deze klasse eenvoudig en snel ontleedbaar.

De objecten die in Deel II onderzocht worden hebben de naam 'tekst' gekregen, om ze te onderscheiden van woorden (het is niet de opzet om een definitie van het algemene begrip tekst voor te stellen).

Een 'tekst' bestaat uit een woord, dus een rij symbolen, uitgebreid met een extra ordening op de posities van die symbolen. Bijvoorbeeld het woord $abac$ heeft 4 posities met op de eerste en derde positie een a, op de tweede positie een b en op de vierde positie een c. Een andere ordening van deze posities is 1243, waarmee aangegeven wordt dat de laatste twee posities verwisseld zijn. Nu is $(abac, 1243)$ een voorbeeld van een tekst. De toegevoegde ordening van een tekst kan gebruikt worden om de in het woord gegeven informatie zodanig te herschikken dat deze informatie doorzichtiger wordt.

Wat vooral van belang is in het kader van dit proefschrift is het feit dat uit de toegevoegde ordening een boom afgeleid kan worden. Elke tekst heeft dus een individueel bepaalde boom. Deze boom geeft de hiërarchische structuur van de tekst weer. Er zijn enkele variaties mogelijk in de gekozen vorm van die individuele boom.

In hoofdstuk 4 worden context-vrije grammatica's gegeneraliseerd van woorden naar teksten. Deze grammatica's worden zodanig beperkt dat de ontledingsbomen overeenkomen met de bovengenoemde individueel bepaalde bomen. Afhankelijk van het gekozen type individuele boom ontstaan zo verschillende klassen van verzamelingen van teksten. Deze klassen worden met bekende klassen van woordverzamelingen vergeleken.

In hoofdstuk 5 wordt deze vergelijking toegespitst op de klasse van zogeheten 'reguliere' verzamelingen. Voor deze klasse wordt een grammaticale en een algebraïsche karakterisatie gegeven. Zo'n karakterisatie is bekend in het geval van woorden. Bovendien bestaat er in dat geval ook een karakterisatie in termen van logica, om precies te zijn in monadische tweede orde

logica. In hoofdstuk 6 wordt een analoog resultaat voor teksten bewezen. Tenslotte wordt de mogelijkheid besproken om dit resultaat uit te breiden naar objecten met een minder eenvoudige structuur dan teksten.

Curriculum Vitae

Paulien ten Pas heeft dit proefschrift geschreven in de periode van 1990 tot 1994 toen zij als assistent-in-opleiding werkzaam was bij de afdeling wiskunde en informatica aan de Rijksuniversiteit Leiden. De begeleiding van haar promotie-onderzoek was in handen van prof. dr. G. Rozenberg en dr. H.J. Hoogeboom.

Van 1985 tot 1990 heeft zij wiskunde gestudeerd, eveneens aan de Rijksuniversiteit Leiden, waarbij zij de laatste drie jaar student-assistent was. Zij is afgestudeerd in de specialistische richting discrete wiskunde op een getaltheorie-project bij prof. dr. R. Tijdeman.

Paulien is geboren op 1 november 1966 te Vlissingen en heeft haar V.W.O.-diploma in 1985 behaald op de Stedelijke Scholengemeenschap Middelburg.