the decidability of the boundedness of functions on hypergraph generating regular tree grammars

masters thesis
wilbert schelvis
august 1994
rijksuniversiteit leiden

Preface

This thesis was written as part of the final assignment for my study at the university of Leiden. I started this study in October 1988, as a part-time student. During the course I switched to the full-time study, since I wanted to graduate in Theoretical Computer Science, which was not possible as part-time student. Luckily, this switch did not cause any significant interference with my personal life and with my professional life, which I spent at the research institute of the PTT in Leidschendam. Up to the final stage of the course. For the final assignment, one has to spent a total of five months at doing research and writing a thesis, which put quite a heavy load on me. Furthermore, most part of the thesis was written during the hottest month of July, since they started scoring the daily temperatures in 1706.

I would like to thank my employer, the Royal PTT Netherlands, for providing me with the facilities, money for the tuition fees and books, and time to attend some lectures. Next I would like to thank all teachers at the Computer Science department of the University of Leiden, for providing me with the basic knowledge and theories needed to write this thesis. Finally, my special thanks are for my supervisor, Joost Engelfriet. Without his great expertise, helpfulness and patience, this thesis would not be anything near to what it is now.

Wilbert Schelvis

Abstract

One of the many formalisms to generate sets of labeled hypergraphs is the theory of hyperedge replacement grammars. In such a grammar, hypergraphs are substituted for hyperedges. Another method for the generation of hypergraph languages is by defining a set of operations on hypergraphs, generating expressions over these operations and finally evaluating these expressions, yielding hypergraphs. These two methods turn out to be equivalent.

For the purpose of the analysis of hypergraph languages it is important to consider certain predicates and numerical functions on the hypergraphs in the languages. A number of relevant predicates and functions can be determined from the derivations of the hypergraphs in the hyperedge replacement grammars or from the expressions that evaluate to the hypergraphs, instead of from the hypergraphs themselves.

For some important properties of the predicates and functions on the hypergraph languages, it turns out that they can be determined from the hyperedge replacement grammars and the regular tree grammars that generate the expressions. For example, whether or not a predicate is true for all hypergraphs in the language or whether or not a function is bounded on the hypergraph language.

Samenvatting

Een van de vele formalismen om verzamelingen van gelabelde hypergrafen te genereren, is de theorie van context-vrije hypergraafgrammatica's. Hierbij worden hypertakken in de hypergrafen vervangen door andere hypergrafen. Een andere methode om hypergrafen te genereren is het definiëren van een aantal operaties op hypergrafen, vervolgens het genereren van expressies over deze operaties met behulp van reguliere boomgrammatica's, welke dan vervolgens geëvalueerd worden. Deze twee methoden blijken equivalent te zijn.

Voor de analyse van hypergraaftalen is het van belang om bepaalde predikaten of numerieke functies op de hypergrafen uit deze talen te beschouwen. Een aantal relevante predikaten en functies kunnen bepaald worden door te kijken naar de afleidingen van de hypergrafen in de hypergraaf grammatica's of naar de expressies die de hypergrafen als betekenis hebben. Dit in plaats van het bepalen van de predikaten en functies op de hypergrafen zelf.

Nu blijkt het dat sommige belangrijke eigenschappen van de predikaten en de functies op de hypergraaftalen aan de hand van de hypergraaf grammatica's en de reguliere boomgrammatica's bepaald kunnen worden. Bijvoorbeeld kan het bepaald worden of een predikaat waar is voor alle hypergrafen uit een hypergraaftaal en of een functie op een hypergraaftaal begrend is.

Contents

Chapter one

Introduction

Graphs and their generalized version, hypergraphs, are widely used in Computer Science, but also in many other scientific disciplines. Various methods for defining and generating hypergraphs have been proposed. Among the more well-known methods are the different types of hypergraph grammars, such as node replacement grammars and hyperedge replacement grammars. Especially hyperedge replacement grammars have nice context-free properties.

When analyzing hypergraphs it is important to determine the values of some numerical functions on these hypergraphs. In [Hab92], a large number of numerical functions on hypergraphs is presented. It is shown that the boundedness of the values of these functions on hypergraph languages, which are generated by hyperedge replacement grammars, is decidable if the functions are compatible with the replacement of hyperedges by hypergraphs.

We will investigate another approach to the generation of hypergraphs. We will not generate hypergraphs using hyperedge replacement grammars, but we will generate expressions over a special set of hypergraph operators, that evaluate to hypergraphs. The numerical functions on the hypergraphs can then be computed by special transformations, that transform the hypergraph expressions into numerical expressions. Evaluating these numerical expressions yields the function values. We will show that for a special class of sets of hypergraph expressions, called regular (hypergraph) expression languages, generated by regular expression grammars, we can obtain similar decidability results as in [Hab92].

In Chapter 2, all the mathematical notions and notations used in this thesis are presented. In Chapter 3, tree grammars and automata are treated. Since expressions closely correspond to trees, these devices are relevant for the above approach. The regular tree grammars are presented, as are tree automata with output, that realize tree transformations. Furthermore, some decidability results are presented, that will be used to show the decidability of the boundedness of functions on hypergraph languages.

In Chapter 4 the two methods for the generation of hypergraphs and hypergraph languages are presented. It is shown that regular expression grammars generate the same hypergraph languages as hyperedge replacement grammars do.

Chapter 5 presents some numerical functions on hypergraphs, realized by tree automata with output, that transform hypergraph expressions in numerical expressions. These functions are defined in [Hab92] for hyperedge replacement grammars. In Chapter 6 compatible and realizable functions are presented. A function on hypergraphs is compatible ([Hab92]) if its value for a hypergraph can be determined on the derivation of the hypergraph in a hyperedge replacement grammar. A function is realizable if it can be computed for a hypergraph, by transforming an expression that evaluates to the hypergraph, into a numerical expression that evaluates to the function value for the hypergraph, using a tree automaton. It is shown that every compatible function is also a realizable function.

Finally, in Chapter 7 some decidability and boundedness results are presented. It is shown that it is decidable for a realizable predicate, whether or not a given regular expression grammar generates expressions that evaluate to hypergraphs for which the predicate holds. It is also shown that it is decidable for a realizable numerical function, whether or not the function value for a hypergraph language, obtained by the evaluation of a regular expression language, grows beyond any bound. The decidability results in [Hab92] follow from the results in Chapter 4 and Chapter 7, and from the fact that every compatible function is also a realizable function. The proofs of the decidability results in Chapter 7 lean heavily on the decidability results in Chapter 3.

Chapter two

Preliminaries

Reading this thesis requires no knowledge in advance. However, it is assumed that the reader is familiar with mathematical notation. This chapter presents all notions that are used throughout the thesis. The first section describes all preliminaries, the second section describes trees, whereas the last section describes some basic universal algebra theory.

2.1 Mathematical preliminaries

This section summarizes the elementary mathematical notions that are used throughout this thesis. The main notions are sets, relations, functions, strings, languages and logic.

Sets

By a *set*, we mean a collection of objects. When an object s is in a set S, we write $s \in S$, and s is called an *element* of S. When s is not in S, we write $s \notin S$. If S contains no elements at all, we call S the *empty* set, denoted by $S = \varnothing$. A set is called *finite* if it contains finitely many elements, otherwise it is called *infinite*. If a set is finite and its k elements are enumerable as $s_1, s_2, \ldots, s_k$, we write $S = \{s_1, s_2, \ldots, s_k\}$. Here the number k is called the *cardinality* of the set S, denoted by $|S|$. A set $\{s\}$ containing only one element is called a *singleton*. If a set is infinite and its elements are enumerable as $s_1, s_2, \ldots$, we write $S = \{s_1, s_2, \ldots\}$. The set of all elements $s \in S$ such that property $p(s)$ holds, is denoted by $\{s \in S \mid p(s)\}$ or $\{s \mid p(s)\}$ if S is understood.

Let S and T be sets. S is called a *subset* of T, denoted by $S \subseteq T$, if all elements of S are also in T. A set U is called the *union* of S and T, denoted by $U = S \cup T$ if U contains all elements that are in S or in T, thus $S \cup T = \{s \mid s \in S \text{ or } s \in T\}$. A set U is called the *intersection* of S and T, denoted by $U = S \cap T$ if U contains all elements that are in S as well as in T, thus $S \cap T = \{s \mid s \in S \text{ and } s \in T\}$.

If S and T are sets, then the *set difference* of S and T, denoted by $S \setminus T$, is the set of all elements of S that are not elements of T, thus $S \setminus T = \{s \in S \mid s \notin T\}$. The set difference of S and T is sometimes called the *complement of T with respect to S*, or simply the complement of T, if S is understood.

If S is a set, then the *powerset* of S, denoted by $\mathbb{P}(S)$, is the set of all subsets of S, thus $\mathbb{P}(S) = \{T \mid T \subseteq S\}$. If S and T are sets, then the *cartesian product* of S and T, denoted by $S \times T$, is the set of all ordered pairs (s, t) such that $s \in S$ and $t \in T$. This can be extended to finitely many sets by

$$S_1 \times \cdots \times S_k = \{(s_1, \ldots, s_k) \mid s_i \in S_i \text{ for } 1 \leq i \leq k\}.$$

Two special sets are the set of all *natural numbers*, $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$ and the set of *booleans*, $\mathbb{B} = \{\mathit{false}, \mathit{true}\}$.

Relations and functions

A subset R of a cartesian product $S \times T$ of two arbitrary sets S and T, is called a *relation* from S to T. Let $R \subseteq S \times T$ be a relation. For every subset $A \subseteq S$, the *image* of A under R is defined as $R(A) = \{t \in T \mid (s, t) \in R$ for at least one $s \in A\}$. For every subset $B \subseteq T$, the *pre-image* of B under R is defined as $R^{-1}(B) = \{s \in S \mid (s,t) \in R$ for at least one $t \in B\}$.

A relation $f \subseteq S \times T$ is called a *partial function* from S to T, denoted by $f : S \to T$, if for every $s \in S$ there is a most one $t \in T$ such that $(s,t) \in f$. Such a t is called the *value* of s under R. This is usually denoted by $f(s) = t$. A function $f : S \to T$ is called *total* (or a *mapping*) if for every $s \in S$, there is precisely one $t \in T$ such that $f(s) = t$. If $f : S_1 \times \cdots \times S_k \to T$ is a function, then k is called the *arity* of the function. The function f is called a k*-ary* function. Let S and T be two arbitrary sets. Then the set of all total functions from S to T is denoted by $(S \to T)$. Let $g : S \to T$ and $f : T \to U$ be two total functions. Then the *composition* of f and g, denoted by $f \circ g$ is a function $f \circ g : S \to U$, defined by $f \circ g(s) = f(g(s))$.

Let S be a set and $R \subseteq S \times S$ a relation on S. Then the *transitive closure* of R, denoted by R^+, is defined as follows.

i. If $(s_1, s_2) \in R$, then $(s_1, s_2) \in R^+$.

ii. If $(s_1, s_2) \in R^+$ and $(s_2, s_3) \in R^+$, then $(s_1, s_3) \in R^+$.

iii. Nothing is in R^+ unless it follows from i and ii.

The *reflexive transitive closure* of R, denoted by R^*, is defined as $R^* = R^+ \cup \{(s,s) \mid s \in S\}$.

Strings and languages

An *alphabet* is a set of symbols. An alphabet is finite, unless it is explicitly defined infinite. Let Σ be an alphabet. A *string* over Σ is a sequence of symbols from Σ. A string is usually written as $\sigma_1 \cdots \sigma_k$, with $\sigma_i \in \Sigma$ for all $1 \leq i \leq k$, where k is the *length* of the string. A string of length 0 is called the empty string, and is denoted by λ.

Let $s = \sigma_1 \cdots \sigma_k$ and $t = \tau_1 \cdots \tau_\ell$ be two strings. The *concatenation* of s and t, denoted by $s \cdot t$, is the string $\sigma_1 \cdots \sigma_k \tau_1 \cdots \tau_\ell$, of length $k + \ell$.

Let Σ be an alphabet and $\sigma \in \Sigma$ a symbol. The concatenation of k times the symbol σ is denoted by σ^k. The set of all strings containing only the symbol σ, including the empty string, is denoted by σ^*. The set $\sigma^* \setminus \{\lambda\}$ is denoted by σ^+. The set of all strings over the alphabet Σ is denoted by Σ^*. Again $\Sigma^+ = \Sigma^* \setminus \{\lambda\}$.

Logic and computability

We will use the standard logical notation. The boolean constants are denoted by *false* and *true*. Logical *and* is denoted by $\wedge$, logical *or* by $\vee$ and *negation* by $\neg$. We will abbreviate "if and only if" by iff. The existential quantifier is denoted by $\exists$ and the universal quantifier by $\forall$.

Furthermore, we will use two notions from computation theory. We say that a function is *computable* if there exists an algorithm that computes its value from a given input. A predicate is called *decidable* if there is an algorithm that decides whether or not the predicate holds for a given input.

2.2 Trees

Trees can be represented in a large number of different, but usually equivalent, ways. The most common representation is probably the graphical representation, as in Figure 2.1. In this section we present our representation of trees as a special kind of strings. Trees correspond closely to expressions, as explained in Section 2.3.

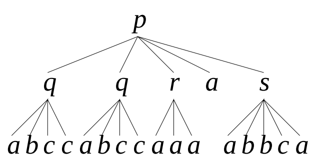


Figure 2.1: Example of a tree

Alphabets and trees

In order to represent trees as strings of symbols, we need to define the number of subtrees that each symbol can have. Of course, this number need not be restricted to one single value. We therefore use the notion of ranked alphabets, defined as follows.

Definition 2.1 (ranked alphabet) An alphabet Σ is a *ranked alphabet* if for each non-negative integer k, a subset Σ_k of Σ is specified, such that Σ_k is nonempty for a finite number of k's only, and such that $\Sigma = \cup_{k \geq 0} \Sigma_k$. If $a \in \Sigma_k$, then we say that a has *rank* k. □

For a ranked alphabet Σ and $a \in \Sigma$, we sometimes use $rank(a) = k$ if it is clear that $a \in \Sigma_k$. Union and equality for ranked alphabets are defined as follows.

Definition 2.2 (union and equality) Let Σ and Δ be ranked alphabets. The *union* of Σ and Δ, denoted by $\Sigma \cup \Delta$, is defined by $(\Sigma \cup \Delta)_k = \Sigma_k \cup \Delta_k$, for all $k \geq 0$. We say that Σ and Δ are *equal*, denoted by $\Sigma = \Delta$, if, for all $k \geq 0$, $\Sigma_k = \Delta_k$. □

Now trees are defined as string over a ranked alphabet and the special symbols [and]. Note that the symbols [and] are not strictly necessary when the rank of each symbol is unique, but even in that case, this syntactic sugar makes them far more readable.

Definition 2.3 (trees over a ranked alphabet) Given a ranked alphabet Σ, the set of *trees over* Σ, denoted by T_Σ, is the language over the alphabet $\Sigma \cup \{[,]\}$ defined inductively as follows.

i. If $a \in \Sigma_0$, then $a \in T_\Sigma$.

ii. For $k \geq 1$, if $a \in \Sigma_k$ and $t_1, \ldots, t_k \in T_\Sigma$, then $a[t_1 t_2 \cdots t_k] \in T_\Sigma$.

The first symbol in a tree is called the *root* of the tree. The symbols of rank 0 are called the *leafs* of the tree. □

Using this definition, the tree in Figure 2.1 can be written as a string over the ranked alphabet Σ, where Σ_0 contains at least the symbols a, b and c, Σ_3 contains at least the symbol r, Σ_4 the symbol q and Σ_5 the symbols p and s. The tree can then be written as

$$p[q[abcc]q[abcc]r[aaa]as[abbca]].$$

A tree language is defined as a subset of the set of all possible trees. This is equivalent to the usual definition of string languages, considered that trees are strings.

Definition 2.4 (tree language) Let Σ be a ranked alphabet. A *tree language* over Σ is any subset of T_Σ. □

Inductive (or recursive) definitions

When working with trees, proofs and definitions are usually very elegant if respectively inductive proofs and inductive definitions are used.

Principle 2.5 (inductive proof) Let P be a property of trees over a ranked alphabet Σ. If

i. all elements of Σ_0 have property P, and

ii. for each $k \geq 1$ and each $a \in \Sigma_k$, if $t_1, \ldots, t_k$ have property P, then $a[t_1 \cdots t_k]$ has property P,

then all trees in T_Σ have property P. □

Principle 2.6 (inductive definition) Suppose we want to associate a value $h(t)$ with each tree t in T_Σ. Then it suffices to define $h(a)$ for all $a \in \Sigma_0$, and to show how to compute the value $h(a[t_1 \cdots t_k])$ from the values $h(t_1), \ldots, h(t_k)$. More formally expressed, given a set $\mathcal{O}$ of objects, and

i. for each $a \in \Sigma_0$, an object $o_a \in \mathcal{O}$, and

ii. for each $k \geq 1$ and each $a \in \Sigma_k$, a mapping $f_a : \mathcal{O}^k \to \mathcal{O}$,

there is exactly one mapping $h : T_\Sigma \to \mathcal{O}$, such that

i. $h(a) = o_a$ for all $a \in \Sigma_0$, and

ii. $h(a[t_1 \cdots t_k]) = f_a(h(t_1), \ldots, h(t_k))$ for all $k \geq 1$, $a \in \Sigma_k$ and $t_1, \ldots, t_k \in T_\Sigma$. $\square$

Two functions on trees that are of special interest are *yield*, which gives the string of leafs in a tree, and *height*, which gives us the longest "distance" from the root of a tree to a leaf of the tree. These function are defined inductively as follows.

Definition 2.7 (yield) The mapping *yield* from T_Σ into Σ_0^+ is defined inductively as follows.

i. For $a \in \Sigma_0$, $yield(a) = a$,

ii. For $a \in \Sigma_k$ and $t_1, \ldots, t_k \in T_\Sigma$,

$$yield(a[t_1 \cdots t_k]) = yield(t_1) \cdot yield(t_2) \cdots yield(t_k).$$

For a tree language $L \subseteq T_\Sigma$, we define $yield(L) = \{yield(t) \mid t \in L\}$. $\square$

Definition 2.8 (height) The mapping *height* from T_Σ into $\mathbb{N}$ is defined recursively as follows.

i. For $a \in \Sigma_0$, $height(a) = 0$.

ii. For $a \in \Sigma_k$ and $t_1, \ldots, t_k \in T_\Sigma$, $height(a[t_1 \cdots t_k]) = max_{1 \leq i \leq k} height(t_i) + 1$. $\square$

Sometimes we will use the set of trees over a ranked alphabet, that is extended with a set of symbols of rank 0, for example a set of variables. This set of trees can be easily defined as follows.

Definition 2.9 (indexed trees) Let Σ be a ranked alphabet and let S be a set of symbols or a tree language. Then the set of *trees indexed by* S, denoted by $T_\Sigma(S)$, is defined inductively as follows.

i. $S \cup \Sigma_0 \subseteq T_\Sigma(S)$.

ii. If $k \geq 1$, $a \in \Sigma_k$ and $t_1, \ldots, t_k \in T_\Sigma(S)$, then $a[t_1 \cdots t_k] \in T_\Sigma(S)$.

Note that $T_\Sigma(\varnothing) = T_\Sigma$. $\square$

2.3 Universal algebra

The theory of universal algebra is often used in computer science as a device for the formal definition of semantics. Detailed information about universal algebra can be obtained from [Wec92] and [GecSte84].

Algebras, expressions and derived functions

The basic notion in the theory of universal algebra is an algebra, which consists of a set of symbols, represented as a ranked alphabet, and a family of mappings that assign meaning to the symbols. The syntactic relation between the symbols stems from the theory of trees, as defined in the previous section.

Definition 2.10 (algebra) Let Σ be a, possibly infinite, ranked alphabet.

i. A pair $\mathfrak{A} = (\mathbf{V}, \alpha)$, where $\mathbf{V}$ is a set of values and α is a family of mappings

$$\alpha = \{\alpha_k : \Sigma_k \to (\mathbf{V}^k \to \mathbf{V})\}_{k \geq 0}$$

that assigns to every $\sigma \in \Sigma_k$, $k \geq 0$, a k-ary total function $\alpha_k(\sigma) : \mathbf{V}^k \to \mathbf{V}$, is a Σ-*algebra*, or just *algebra* if Σ is clear. $\alpha_k(\sigma)$ is also written as $\sigma^{\mathfrak{A}}$. The *domain* of $\mathfrak{A}$, sometimes denoted by $dom(\mathfrak{A})$, is the set $\mathbf{V}$. Σ is called the *signature* of $\mathfrak{A}$.

ii. A symbol $\sigma \in \Sigma$ is called an *operator* and the operation $\sigma^{\mathfrak{A}}$ is called the *realization of* σ *in* $\mathfrak{A}$ or the *operation* corresponding to σ in $\mathfrak{A}$.

iii. An element of T_Σ is called an expression over Σ.

iv. Let $\mathfrak{A} = (\mathbf{V}, \alpha)$ be a Σ-algebra. Then $\mathfrak{A}$ determines a mapping $\mathfrak{m}^{\mathfrak{A}} : T_\Sigma \to \mathbf{V}$, which is defined inductively as follows. For $\sigma \in \Sigma_0$, $\mathfrak{m}^{\mathfrak{A}}(\sigma) = \sigma^{\mathfrak{A}}$, where $\sigma^{\mathfrak{A}}$ is a nullary function or constant. For $\sigma \in \Sigma_k$, $k \geq 1$ and $t_1, \ldots, t_k \in T_\Sigma$,

$$\mathfrak{m}^{\mathfrak{A}}(\sigma[t_1 \cdots t_k]) = \sigma^{\mathfrak{A}}(\mathfrak{m}^{\mathfrak{A}}(t_1), \ldots, \mathfrak{m}^{\mathfrak{A}}(t_k)).$$

The mapping $\mathfrak{m}^{\mathfrak{A}}$ is called the *initial homomorphism*. □

Expressions can consist entirely of operators, representing functions. In this case, the meaning of the expression is a value from the domain of the algebra, as defined in Definition 2.10iv. However, expressions can also contain variables, in which case the meaning of the expression is a function. This is made clear in the following two definitions.

Definition 2.11 ($\Sigma\mathbf{Z}$-expression) Let $\mathfrak{A}$ be a Σ-algebra and $\mathbf{Z}$ a set of variables, such that $\Sigma \cap \mathbf{Z} = \emptyset$. A $\Sigma\mathbf{Z}$-*expression* is any element of $T_\Sigma(\mathbf{Z})$. □

Definition 2.12 (derived function) Let $\mathfrak{A} = (\mathbf{V}, \alpha)$ be a Σ-algebra, $\mathbf{Z}$ a set of variables, disjoint with Σ, and $t \in T_\Sigma(\mathbf{Z})$ a $\Sigma\mathbf{Z}$-expression. Then the *derived function* of t, denoted by $t^{\mathfrak{A}}$, is the mapping $t^{\mathfrak{A}} : (\mathbf{Z} \to \mathbf{V}) \to \mathbf{V}$, defined inductively as follows. For any mapping $\gamma : \mathbf{Z} \to \mathbf{V}$, assigning a value to each variable in $\mathbf{Z}$, let

i. $z^{\mathfrak{A}}(\gamma) = \gamma(z)$ for $z \in \mathbf{Z}$ and

ii. $t^{\mathfrak{A}}(\gamma) = \sigma^{\mathfrak{A}}(t_1^{\mathfrak{A}}(\gamma), \ldots, t_k^{\mathfrak{A}}(\gamma))$ for $t = \sigma[t_1 \cdots t_k]$, $k \geq 0$, $\sigma \in \Sigma_k$ and $t_1, \ldots, t_k \in T_\Sigma(\mathbf{Z})$. □

Every signature, and thus every ranked alphabet, generates an algebra, by using the set of all expressions over the signature as the domain, and the syntactic rules for the construction of the expressions as the operations. This algebra is called the free algebra.

Definition 2.13 (free algebra) Let Σ be a signature and $\mathbf{Z}$ a set of variables, disjoint with Σ. The *free Σ-algebra generated by* $\mathbf{Z}$, denoted by $\mathfrak{F}_\Sigma(\mathbf{Z})$, is the Σ-algebra $(T_\Sigma(\mathbf{Z}), \alpha)$, where α is a mapping that assigns to every $\sigma \in \Sigma_k$, $k \geq 0$, a mapping $\sigma^{\mathfrak{F}_\Sigma(\mathbf{Z})} : T_\Sigma(\mathbf{Z})^k \to T_\Sigma(\mathbf{Z})$, defined by

$$\sigma^{\mathfrak{F}_\Sigma(\mathbf{Z})}(t_1, \ldots, t_k) = \sigma[t_1 \cdots t_k].$$

If $\mathbf{Z} = \varnothing$, then $\mathfrak{F}_\Sigma(\mathbf{Z})$ is written as $\mathfrak{F}_\Sigma = (T_\Sigma, \alpha)$, and it is called the *free Σ-algebra*. In this case, α is a mapping that assigns to every $\sigma \in \Sigma_k$, $k \geq 0$, a mapping $\sigma^{\mathfrak{F}_\Sigma} : T_\Sigma^k \to T_\Sigma$. □

Substitution

For a signature Σ, the free Σ-algebra can be used to give a formal definition of substitution of variables by expressions. Let $\mathbf{Z}$ be a set of variables and $\mathfrak{F}_\Sigma(\mathbf{Z}) = (T_\Sigma(\mathbf{Z}), \alpha)$ be the free Σ-algebra generated by $\mathbf{Z}$. Furthermore, for $k \geq 1$, let $\overline{\mathbf{Z}} = \{z_1, \ldots, z_k\}$ be a set of variables, disjoint with Σ, $t \in T_\Sigma(\overline{\mathbf{Z}})$ a $\Sigma\overline{\mathbf{Z}}$-expression and $\gamma : \overline{\mathbf{Z}} \to T_\Sigma(\mathbf{Z})$ an assignment function. Then the *substitution of $\gamma(z_i)$ for z_i in t*, for $1 \leq i \leq k$, is

$$t\langle z_1 \leftarrow \gamma(z_1), \ldots, z_k \leftarrow \gamma(z_k)\rangle = t^{\mathfrak{F}_\Sigma(\mathbf{Z})}(\gamma),$$

where $t^{\mathfrak{F}_\Sigma(\mathbf{Z})} : (\overline{\mathbf{Z}} \to T_\Sigma(\mathbf{Z})) \to T_\Sigma(\mathbf{Z})$ is the derived function of t in $\mathfrak{F}_\Sigma(\mathbf{Z})$. In the case where $\mathbf{Z} = \varnothing$, it can be shown by induction on t, that for any Σ-algebra $\mathfrak{A}$ and for each $\gamma : \overline{\mathbf{Z}} \to T_\Sigma$,

$$\mathfrak{m}^{\mathfrak{A}}(t^{\mathfrak{F}_\Sigma}(\gamma)) = t^{\mathfrak{A}}(\mathfrak{m}^{\mathfrak{A}} \circ \gamma).$$

Chapter three

Tree automata and tree grammars

As string languages can be generated and recognized by string automata and string grammars, tree languages can be generated and recognized by tree automata and tree grammars. Certain types of automata and grammars generate certain types of tree languages, much similar as in the string case.

Tree automata can also be used to define tree transformations. A tree is then transformed while the automaton traverses the tree in parallel, either in a top-down or in a bottom-up direction. Each symbol or subtree of the tree, can then be replaced by another symbol or subtree, thus realizing the transformation in an effective, inductive, way.

3.1 Recognizable tree languages

The definitions of tree automata and tree grammars used in this thesis are from [Eng74]. Nearly the same definitions are used in [Eng75] and [Eng77]. Another well known source for the theory of tree automata and tree grammars is [GecSte84].

3.1.1 Finite tree automata and regular tree grammars

A finite tree automaton is very much like a finite state automaton for the string case. When the automaton is in a certain state, it can recognize one of several symbols and jump to a new state, depending on the recognized symbol. The major difference between a finite state automaton and a finite tree automaton is, that a finite tree automaton can traverse all the subtrees of a node in parallel. There are two types of finite tree automata, bottom-up and top-down, each of which can be deterministic or non-deterministic. A bottom-up finite tree automaton starts at the leafs of a tree and traverses up to the root, while a top-down finite tree automaton starts at the root, and traverses down to the leafs. We will first define bottom-up finite tree automata.

Definition 3.1 (deterministic bottom-up finite tree automaton) A *deterministic bottom-up finite tree automaton* (fta) is a structure $M = (Q, \Sigma, \delta, s, F)$, where Q is a finite set of *states*, Σ is a ranked alphabet of *input symbols*, δ is a family $\{\delta_a^k\}_{k \geq 1, a \in \Sigma_k}$ of mappings $\delta_a^k : Q^k \to Q$ (the *transition function* for $a \in \Sigma_k$), s is a family $\{s_a\}_{a \in \Sigma_0}$ of *initial states* states $s_a \in Q$ and F is a subset of Q of *final* states.

The mapping $\tilde{\delta} : T_\Sigma \to Q$ is defined recursively as follows:

i. for $a \in \Sigma_0$, $\tilde{\delta}(a) = s_a$ and

ii. for $k \geq 1$, $a \in \Sigma_k$ and $t_1, \ldots, t_k \in T_\Sigma$,

$$\tilde{\delta}(a[t_1 \cdots t_k]) = \delta_a^k(\tilde{\delta}(t_1), \ldots, \tilde{\delta}(t_k)).$$

The *tree language recognized by M*, denoted by $\mathcal{L}(M)$, is defined to be the set $\{t \in T_\Sigma \mid \tilde{\delta}(t) \in F\}$. □

Definition 3.2 (non-deterministic bottom-up finite tree automaton) A *non-deterministic bottom-up finite tree automaton* is a 5-tuple $M = (Q, \Sigma, \delta, S, F)$, where Q, Σ and F are as in the deterministic case, S is a family $\{S_a\}_{a \in \Sigma_0}$, such that $S_a \subseteq Q$ for each $a \in \Sigma_0$, and δ is a family $\{\delta_a^k\}_{k \geq 1, a \in \Sigma_k}$ of mappings $\delta_a^k : Q^k \to \mathbb{P}(Q)$.

The mapping $\tilde{\delta} : T_\Sigma \to \mathbb{P}(Q)$ is defined recursively by

i. for $a \in \Sigma_0$, $\tilde{\delta}(a) = S_a$ and

ii. for $k \geq 1$, $a \in \Sigma_k$ and $t_1, \ldots, t_k \in T_\Sigma$,

$$\tilde{\delta}(a[t_1 \cdots t_k]) = \cup\{\delta_a^k(q_1, \ldots, q_k) \mid q_i \in \tilde{\delta}(t_i) \text{ for } 1 \leq i \leq k\}.$$

The tree language recognized by M, denoted by $\mathcal{L}(M)$, is defined to be $\{t \in T_\Sigma \mid \tilde{\delta}(t) \cap F \neq \varnothing\}$. □

As is the case with deterministic and non-deterministic finite state automata, deterministic and non-deterministic bottom-up finite tree automata recognize the same languages.

Theorem 3.3 For each non-deterministic bottom-up fta, we can find a deterministic bottom-up fta that recognizes the same language.

Proof In [Eng74, pages 25–26]. □

The class of tree languages that can be recognized by bottom-up finite tree automata, turns out to be a very important class. This class is defined as follows.

Definition 3.4 (recognizable tree language) A tree language L is called *recognizable* or *regular* if $L = \mathcal{L}(M)$ for some deterministic bottom-up fta M. The class of all recognizable tree languages will be denoted by **RECOG**. □

As bottom-up finite tree automata start at the leafs and traverse the trees to the root, top-down finite tree automata traverse them in the other direction, from the root to the leafs.

Definition 3.5 (deterministic top-down finite tree automaton) A *deterministic top-down finite tree automaton* is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of states, Σ is a ranked alphabet, δ is a family $\{\delta_a^k\}_{k \geq 1, a \in \Sigma_k}$ of mappings $\delta_a^k : Q \to Q^k$, $q_0 \in Q$ an initial state, and F is a family $\{F_a\}_{a \in \Sigma_0}$ of sets $F_a \subseteq Q$ (the set of final states for $a \in \Sigma_0$).

The mapping $\tilde{\delta} : T_\Sigma \to \mathbb{P}(Q)$ is defined recursively by

i. for $a \in \Sigma_0$, $\tilde{\delta}(a) = F_a$ and

ii. for $k \geq 1$, $a \in \Sigma_k$ and $t_1, \ldots, t_k \in T_\Sigma$,

$$\tilde{\delta}(a[t_1 \cdots t_k]) = \{q \mid \delta_a(q) \in \tilde{\delta}(t_1) \times \cdots \times \tilde{\delta}(t_k)\}.$$

The tree language recognized by M, denoted by $\mathcal{L}(M)$, is defined to be $\{t \in T_\Sigma \mid q_0 \in \tilde{\delta}(t)\}$. □

Definition 3.6 (non-deterministic top-down finite tree automaton) A *non-deterministic top-down finite tree automaton* is a 5-tuple $M = (Q, \Sigma, \delta, S, F)$, where Q, Σ and F are as in the deterministic case, S is a subset of Q and δ is a family $\{\delta_a^k\}_{k \geq 1, a \in \Sigma_k}$ of mappings $\delta_a^k : Q \to \mathbb{P}(Q^k)$.

The mapping $\tilde{\delta} : T_\Sigma \to \mathbb{P}(Q)$ is defined recursively as follows

i. for $a \in \Sigma_0$, $\tilde{\delta}(a) = F_a$ and

ii. for $k \geq 1$, $a \in \Sigma_k$ and $t_1, \ldots, t_k \in T_\Sigma$,

$$\tilde{\delta}(a[t_1 \cdots t_k]) = \{q \mid \exists (q_1, \ldots, q_k) \in \delta_a(q) : q_i \in \tilde{\delta}(t_i) \text{ for all } 1 \leq i \leq k\}.$$

The tree language recognized by M, denoted by $\mathcal{L}(M)$, is defined to be $\{t \in T_\Sigma \mid \tilde{\delta}(t) \cap S \neq \varnothing\}$. □

The class of deterministic top-down finite tree automata is less powerful than the class of bottom-up finite tree automata.

Theorem 3.7 There are recognizable tree languages which cannot be recognized by a deterministic top-down fta.

Proof In [Eng74, pages 28–29]. □

However, the class of non-deterministic top-down finite tree automata is equivalent to the class of bottom-up finite tree automata.

Theorem 3.8 A tree language is recognizable by a non-deterministic bottom-up fta iff it is recognizable by a non-deterministic top-down fta.

Proof In [Eng74, page 30]. □

Another method of generating or recognizing tree languages is via regular tree grammars. This method generates the same class of tree languages as do bottom-up finite tree automata.

Definition 3.9 (regular tree grammar) A *regular tree grammar* is a tuple $G = (N, \Sigma, R, S)$, where N is a finite set of non-terminals, Σ is a ranked alphabet of terminals, such that $\Sigma \cap N = \varnothing$, $S \in N$ is the initial non-terminal, and R is a finite set of *rules* of the form $A \to t$, with $A \in N$ and $t \in T_\Sigma(N)$.

The tree language generated by G, denoted by $\mathcal{L}(G)$, is defined to be $\mathcal{L}(H)$, where H is the context free grammar $(N, \Sigma \cup \{[,]\}, R, S)$. We shall use $\Rightarrow_G$ and $\Rightarrow_G^*$ (or $\Rightarrow$ and $\Rightarrow^*$ when G is understood) to denote the restrictions of $\Rightarrow_H$ and $\Rightarrow_H^*$ to $T_\Sigma(N)$.

The set of all regular tree grammars (N, Σ, R, S), with $\Sigma \subseteq \mathbf{C}$ for some alphabet $\mathbf{C}$, is denoted by $\mathbf{RTG_C}$. □

Theorem 3.10 A tree language can be generated by a regular tree grammar iff it is an element of $\mathbf{RECOG}$.

For proofs it is convenient to have a normal form for regular tree grammars, such that each regular tree grammar has an equivalent grammar in normal form.

Definition 3.11 (normal form) A regular tree grammar $G = (N, \Sigma, R, S)$ is in *normal form*, if each of its rules is either of the form $A \to a[B_1 \cdots B_k]$ or of the form $A \to b$, where $k \geq 1$, $a \in \Sigma_k$, $A, B_1, \ldots, B_k \in N$ and $b \in \Sigma_0$. □

Theorem 3.12 Each regular tree grammar has an equivalent regular tree grammar in normal form.

Proof In [Eng74, pages 32–33]. □

There is a strong relation between tree languages and context-free string languages, as can be seen from the following theorem.

Theorem 3.13 $\mathit{yield}(\mathbf{RECOG}) = \mathbf{CFL}$, or the *yield* of each recognizable tree language is context free, and each context free language is the *yield* of some recognizable tree language.

Proof In [Eng74, pages 34–35]. □

3.1.2 Properties of recognizable tree languages

We will now give some properties of recognizable tree languages, that will be used in later parts of this thesis. More properties can be found in [Eng74]. First, the union and intersection of two regular tree languages is also a regular tree language, as is the complement of a regular tree language.

Theorem 3.14 **RECOG** is closed under union, intersection and complementation (with respect to the set of all trees over the ranked alphabet).

Proof In [Eng74, page 36]. □

In Chapter 7, some decidability results are presented on hypergraph languages, generated by evaluating hypergraph expressions, which are generated by regular tree languages. The proofs of these results are based on decidability results on regular tree languages. For example, it is possible to decide whether or not a regular tree language contains one or more trees.

Theorem 3.15 The emptiness problem for recognizable tree languages is decidable.

Proof In [Eng74, page 59]. □

It is also decidable whether a regular tree language is finite or infinite.

Theorem 3.16 The finiteness problem for recognizable tree languages is decidable.

Proof In [Eng74, page 59]. □

Finally, it is decidable whether or not a certain regular tree language is a subset of another regular tree language.

Theorem 3.17 It is decidable, for arbitrary recognizable tree languages U and V, whether $U \subseteq V$.

Proof In [Eng74, page 60]. □

Note that this implies that it is decidable whether or not two regular tree languages are equal, since $U = V$ iff $U \subseteq V$ and $V \subseteq U$.

3.2 Finite state tree transformations

Tree transformations are, in general, just relations between trees. A tree transformation can be realized by defining a mapping on the symbols in the tree or by adding output to the finite tree automata.

3.2.1 Tree transformations

A tree transformation in its most general form is just a relation from some T_Σ to some T_Δ.

Definition 3.18 (tree transformation) Let Σ and Δ be ranked alphabets. A *tree transformation* from T_Σ into T_Δ is any subset of $T_\Sigma \times T_\Delta$. □

The composition of two tree transformations, the inverse of a tree transformation and the image of a tree language under a tree transformation are defined in a similar way as the composition, inverse and image for functions. Note that the tree transformations need not be functions.

Definition 3.19 (composition) Let Σ, Δ and Ω be ranked alphabets. If $M_1 \subseteq T_\Sigma \times T_\Delta$ and $M_2 \subseteq T_\Delta \times T_\Omega$, then the *composition* of M_2 and M_1, denoted by $M_2 \circ M_1$, is the tree transformation

$$\{(s,t) \in T_\Sigma \times T_\Omega \mid (s,u) \in M_1 \text{ and } (u,t) \in M_2 \text{ for some } u \in T_\Delta\}.$$

If F and G are classes of tree transformations, then $G \circ F$ denotes the class $\{M_2 \circ M_1 \mid M_1 \in F \text{ and } M_2 \in G\}$. Furthermore, $\mathbf{F}^*$ denotes the class

$$\mathbf{F}^* = \{M_n \circ \cdots \circ M_1 \mid n \geq 0, M_i \in \mathbf{F} \text{ for } 1 \leq i \leq n\}$$

of arbitrary compositions of elements of F. □

Definition 3.20 (inverse tree transformation) Let M be a tree transformation from T_Σ to T_Δ. The *inverse* of M, denoted by M^{-1}, is the tree transformation $\{(t,s) \in T_\Delta \times T_\Sigma \mid (s,t) \in M\}$. □

Definition 3.21 (image) Let M be a tree transformation and L a tree language. The *image* of L under M, denoted by $M(L)$, is the tree language $M(L) = \{t \mid (s,t) \in M \text{ for some } s \in L\}$. If M is a tree transformation from T_Σ into T_Δ, then the *domain* of M, denoted by *dom*(M), is $M^{-1}(T_\Delta)$, and the *range* of M, denoted by *range*(M), is $M(T_\Sigma)$. □

The first and simplest method of defining a tree transformation is via a relabeling. In a relabeling, only the symbols in a tree are changed. The structure of the tree remains unchanged.

Definition 3.22 (relabeling) Let Σ and Δ be ranked alphabets. A *relabeling* r is a family $\{r_k\}_{k\geq 0}$ of mappings $r_k : \Sigma_k \to \mathbb{P}(\Delta_k)$. A relabeling determines a mapping $r : T_\Sigma \to \mathbb{P}(T_\Delta)$ by the requirements

i. for $a \in \Sigma_0$, $r(a) = r_0(a)$,

ii. for $k \geq 1$, $a \in \Sigma_k$ and $t_1, \ldots, t_k \in T_\Sigma$, $r(a[t_1 \cdots t_k]) = \{b[s_1 \cdots s_k] \mid b \in r_k(a) \text{ and } s_i \in r(t_i)\}$. □

The image of a regular tree language under a relabeling, is also a regular tree language.

Theorem 3.23 **RECOG** is closed under relabelings.

Proof In [Eng74, page 42]. □

A more complicated method of defining tree transformations is with the aid of tree homomorphisms. With a tree homomorphism, each symbol in a tree is replaced by a tree, where the subtrees of the symbol can be attached at arbitrary positions in the tree that replaces the symbol. The subtrees can also be deleted and/or copied, meaning that the same subtree can be attached at more than one position.

Definition 3.24 (variables) Let $x_1, x_2, x_3, \ldots$ be an infinite sequence of different symbols, called *variables*. Let $\mathbf{X} = \{x_1, x_2, x_3, \ldots\}$, for $k \geq 1$, $\mathbf{X}_k = \{x_1, x_2, \ldots, x_k\}$ and $\mathbf{X}_0 = \varnothing$. □

Definition 3.25 (tree homomorphism) Let Σ and Δ be ranked alphabets. A *tree homomorphism* h is a family $\{h_k\}_{k\geq 0}$ of mappings $h_k : \Sigma_k \to T_\Delta(\mathbf{X}_k)$. A tree homomorphism determines a mapping $h : T_\Sigma \to T_\Delta$ as follows.

i. For $a \in \Sigma_0$, $h(a) = h_0(a)$.

ii. For $k \geq 1$, $a \in \Sigma_k$ and $t_1, \ldots, t_k \in T_\Sigma$,

$$h(a[t_1 \cdots t_k]) = h_k(a)\langle x_1 \leftarrow h(t_1), \ldots, x_k \leftarrow h(t_k)\rangle.$$

In the particular case that, for each $a \in \Sigma_k$, $h_k(a)$ does not contain two occurrences of the same x_i, $i = 1, 2, 3, \ldots$, h is called a *linear* tree homomorphism. □

Unlike relabelings, tree homomorphisms can lead out of the class of regular tree languages, due to the fact that a tree homomorphism can copy subtrees.

Theorem 3.26 **RECOG** is not closed under arbitrary tree homomorphisms.

Proof In [Eng74, page 51]. □

In a linear tree homomorphism, copying is prohibited, and therefore the image of a regular tree language under a linear tree homomorphism is a regular tree language itself.

Theorem 3.27 **RECOG** is closed under linear tree homomorphisms.

Proof In [Eng74, page 51–54]. □

Notation 3.28 We shall use **REL** to denote the class of all relabelings, **HOM** to denote the class of all tree homomorphisms and **LHOM** to denote the class of all linear tree homomorphisms. □

3.2.2 Bottom-up and top-down finite tree transducers

More complicated tree transformations can be defined by using finite tree transducers. These are much like finite tree automata, but have not only input, but also output. Since they are like finite tree automata, they also come in two major flavors, namely bottom-up finite tree transducers and top-down finite tree transducers. The semantics of the finite tree transducers are defined with the aid of tree rewriting systems, which are defined as follows.

Definition 3.29 (rewriting system with variables) A *rewriting system with variables* is a pair $G = (\Delta, R)$ where Δ is an alphabet and R is a finite set of "rule schemes". A *rule scheme* is a triple (v, w, D) such that, for some $k \geq 0$, v and w are strings over $\Delta \cup \mathbf{X}_k$ and D is a mapping from $\mathbf{X}_k$ into $\mathbb{P}(\Delta^*)$. Whenever D is understood, (v, w, D) is denoted by $v \to w$. For $1 \leq i \leq k$, the language $D(x_i)$ is called the *range* or *domain* of the variable x_i.

A relation $\Rightarrow_G$ on Δ^* is defined as follows. For strings $s, t \in \Delta^*$, $s \Rightarrow_G t$ iff there exists a rule scheme $(v, w, D) \in R$, strings $\varphi_1, \ldots, \varphi_k \in D(x_1), \ldots, D(x_k)$ respectively (where $\mathbf{X}_k$ is the domain of D), and strings α and β in Δ^* such that

$$s = \alpha \cdot v\langle x_1 \leftarrow \varphi_1, \ldots, x_k \leftarrow \varphi_k\rangle \cdot \beta \text{ and}$$

$$t = \alpha \cdot w\langle x_1 \leftarrow \varphi_1, \ldots, x_k \leftarrow \varphi_k\rangle \cdot \beta.$$

As usual $\Rightarrow_G^*$ denotes the transitive-reflexive closure of $\Rightarrow_G$. □

Definition 3.30 (tree rewriting system) A rewriting system with variables $G = (\Delta, R)$ is called a *tree rewriting system* if

i. $\Delta = \Sigma \cup \{[,]\}$ for some ranked alphabet Σ and

ii. for each rule $(v, w, D) \in R$, v and w are trees in $T_\Sigma(\mathbf{X}_k)$ and, for $1 \leq i \leq k$, $D(x_i) \subseteq T_\Sigma$, where $\mathbf{X}_k$ is the domain of D. □

The bottom-up version of the finite tree transducer is introduced first.

Definition 3.31 (bottom-up finite tree transducer) A *bottom-up (finite) tree transducer* is a structure $M = (Q, \Sigma, \Delta, R, Q_d)$, where Q is a ranked alphabet of *states*, such that all elements of Q have rank 1 and no other ranks, Σ is a ranked alphabet of *input* symbols, Δ is a (possibly infinite) ranked alphabet of *output* symbols, $Q \cap (\Sigma \cup \Delta) = \varnothing$, Q_d is a subset of Q (the set of *final* states) and R is a finite set of *rules* of one of the forms

i. $a \to q[t]$, where $a \in \Sigma_0$, $q \in Q$ and $t \in T_\Sigma$ or

ii. $a[q_1[x_1] \cdots q_k[x_k]] \to q[t]$, where $k \geq 1$, $a \in \Sigma_k$, $q_1, \ldots, q_k, q \in Q$ and $t \in T_\Delta(\mathbf{X}_k)$.

M is viewed as a tree rewriting system over the ranked alphabet $Q \cup \Sigma \cup \Delta$ with R as the set of rules, such that the range of each variable occurring in R is T_Δ. Therefore the relations $\Rightarrow_M$ and $\Rightarrow^*_M$ are well defined according to Definition 3.29. The *tree transformation realized by* M, also denoted by M, is

$$M = \{(s,t) \in T_\Sigma \times T_\Delta \mid s \Rightarrow^*_M q[t] \text{ for some } q \in Q_d\}.$$

We shall abbreviate “finite tree transducer” by ftt. □

Remark 3.32 The definition of the bottom-up finite tree transducer allows for an infinite output alphabet. Since the number of rules is finite, only a finite subset of the output alphabet will be actually used. □

Remark 3.33 Note that the finite tree transducer and the tree transformation it realizes are both denoted by M. In general, we shall make no distinction between a tree transducer and the tree transformation it realizes. □

Note that the bottom-up finite tree transducer defined above is non-deterministic, since for each state and each input symbol, more than one rule can be applicable.

Definition 3.34 (bottom-up tree transformation) The class of tree transformations realized by bottom-up ftts will be denoted by $\mathbf{B}$. An element of $\mathbf{B}$ will be called a *bottom-up tree transformation.* □

We will now define some special classes of bottom-up finite tree transformations. Each class of bottom-up tree tranducers is able to define a certain class of tree transformations.

Linear bottom-up finite tree transducers are like linear tree homomorphisms in the sense that copying of subtrees is prohibited. Linear bottom-up finite tree transducers are defined with the aid of linear trees.

Definition 3.35 (linear tree) Let Σ be a ranked alphabet and $k \geq 0$. A tree $t \in T_\Sigma(\mathbf{X}_k)$ is *linear* if each element of $\mathbf{X}_k$ occurs at most once in t. The tree t is called *non-deleting with respect to* $\mathbf{X}_k$ if each element of $\mathbf{X}_k$ occurs at least once in t. □

Definition 3.36 (linear) Let $M = (Q, \Sigma, \Delta, R, Q_d)$ be a bottom-up ftt. M is *linear* if the right hand side of each rule in R is linear. $\square$

In a non-deleting bottom-up finite tree transducer, the deletion of subtrees is prohibited.

Definition 3.37 (non-deleting) Let $M = (Q, \Sigma, \Delta, R, Q_d)$ be a bottom-up ftt. M is *non-deleting* if the right hand side of each rule in R is non-deleting with respect to $\mathbf{X}_k$, where k is the rank of the input symbol in the left hand side of the rule.

Definition 3.38 (one-state) Let $M = (Q, \Sigma, \Delta, R, Q_d)$ be a bottom-up ftt. M is *one-state* (or *pure*) if Q is a singleton.

With some restrictions on the set of rules, the bottom-up finite tree transducer, which is essentially non-deterministic, can be made deterministic.

Definition 3.39 (partial deterministic) Let $M = (Q, \Sigma, \Delta, R, Q_d)$ be a bottom-up ftt. M is (*partial*) *deterministic* if

i. for each $a \in \Sigma_0$ there is at most one rule in R with left hand side a and

ii. for each $k \geq 1$, $a \in \Sigma_k$ and $q_1, \ldots, q_k \in Q$ there is at most one rule in R with left hand side $a[q_1[x_1] \cdots q_k[x_k]]$. $\square$

Definition 3.40 (total deterministic) Let $M = (Q, \Sigma, \Delta, R, Q_d)$ be a bottom-up ftt. M is *total deterministic* if

i. for each $a \in \Sigma_0$ there is exactly one rule in R with left hand side a,

ii. for each $k \geq 1$, $a \in \Sigma_k$ and $q_1, \ldots, q_k \in Q$ there is exactly one rule in R with left hand side $a[q_1[x_1] \cdots q_k[x_k]]$ and

iii. $Q_d = Q$. $\square$

Notation 3.41 The same terminology will be applied to the transformations realized by such transducers. The classes of tree transformations obtained by putting one or more of the above restrictions on the bottom-up tree transducer, will be denoted by adding the symbols $\mathbf{L}$, $\mathbf{N}$, $\mathbf{P}$, $\mathbf{D}$ and $\mathbf{D_t}$ (standing for linear, non-deleting, pure, deterministic and total deterministic respectively) to the letter $\mathbf{B}$. Thus the class of linear deterministic bottom-up tree transformations is denoted by $\mathbf{LDB}$. $\square$

Remark 3.42 The total deterministic bottom-up ftts realize tree transformations which are total functions. $\square$

Now we will define the top-down version of the finite tree transducers.

Definition 3.43 (top-down finite tree transducer) A *top-down finite tree transducer* is a structure $M = (Q, \Sigma, \Delta, R, Q_d)$, where Q, Σ and Δ are as for the bottom-up ftt (definition 3.31), $Q_d \subseteq Q$ is a set of *initial* states and R is a finite set of *rules* of one of the forms

i. $q[a[x_1 \cdots x_k]] \rightarrow t$, where $k \geq 1$, $a \in \Sigma_k$, $q \in Q$ and $t \in T_\Delta(Q[\mathbf{X}_k])$ or

ii. $q[a] \rightarrow t$, where $q \in Q$, $a \in \Sigma_0$ and $t \in T_\Delta$.

M is viewed as a tree rewriting system over the ranked alphabet $Q \cup \Sigma \cup \Delta$ with R as the set of rules, such that the range of each variable in $\mathbf{X}$ is T_Σ. The tree transformation realized by M, also denoted by M, is defined as

$$M = \{(s, t) \in T_\Sigma \times T_\Delta \mid q[s] \Rightarrow^*_M t \text{ for some } q \in Q_d\}.$$

Again, both the finite tree transducer and the tree transformation are denoted by M. □

Definition 3.44 (top-down tree transformation) The class of tree transformations realized by top-down ftts will be denoted by $\mathbf{T}$. An element of $\mathbf{T}$ will be called a *top-down tree transformation* . □

The classes of tree transformations for the bottom-up finite tree transducers are also defined for the top-down finite tree transducers. Their definitions are as follows.

Definition 3.45 (linear) Let $M = (Q, \Sigma, \Delta, R, Q_d)$ be a top-down ftt. M is *linear* if the right hand side of each rule in R is linear. □

Definition 3.46 (non-deleting) Let $M = (Q, \Sigma, \Delta, R, Q_d)$ be a top-down ftt. M is *non-deleting* if the right hand side of each rule in R is non-deleting with respect to $\mathbf{X}_k$, where k is the rank of the input symbol in the left hand side of the rule.□

Definition 3.47 (one-state) Let $M = (Q, \Sigma, \Delta, R, Q_d)$ be a top-down ftt. M is *one-state* (or *pure*) if Q is a singleton. □

Definition 3.48 (partial deterministic) Let $M = (Q, \Sigma, \Delta, R, Q_d)$ be a top-down ftt. M is (*partial*) *deterministic* if

i. Q_d is a singleton,

ii. for each $q \in Q$, $k \geq 1$ and $a \in \Sigma_k$, there is at most one rule in R with left hand side $q[a[x_1 \cdots x_k]]$,

iii. for each $q \in Q$ and $a \in \Sigma_0$ there is at most one rule in R with left hand side $q[a]$. □

Definition 3.49 (total deterministic) Let $M = (Q, \Sigma, \Delta, R, Q_d)$ be a top-down ftt. M is *total deterministic* if

i. Q_d is a singleton,

ii. for each $q \in Q$, $k \geq 1$ and $a \in \Sigma_k$, there is exactly one rule in R with left hand side $q[a[x_1 \cdots x_k]]$,

iii. for each $q \in Q$ and $a \in \Sigma_0$ there is exactly one rule in R with left hand side $q[a]$. □

Remark 3.50 The total deterministic top-down ftts realize tree transformations which are total functions. □

We use the same notation to denote the classes of top-down finite tree transformations as we did for the bottom-up finite tree transformations (Notation 3.41).

Remark 3.51 Tree homomorphisms can be realized by total deterministic one-state top-down or bottom-up finite tree transducers: $\mathbf{PD_tB} = \mathbf{PD_tT} = \mathbf{HOM}$.□

Since a total-deterministic top-down tree transducer realizes a total function for each state, this function can be defined in an explicit way, as follows.

Definition 3.52 Let $M = (Q, \Sigma, \Delta, R, \{q_d\})$ be a total deterministic top-down tree transducer. For each $q \in Q$ and $a \in \Sigma_k$, let $rhs_{q,a}$ be the right-hand side of the unique rule with left-hand side $q[a[x_1 \cdots x_k]]$. Now the *function defined by* M, also denoted by M, is the total function $M : Q \times T_\Sigma \to T_\Delta$, defined recursively as follows. If $s = a[s_1 \cdots s_k]$, $k \geq 0$, $a \in \Sigma_k$ and $s_1, \ldots, s_k \in T_\Sigma$, then for all $q \in Q$,

$$M(q, a[s_1 \cdots s_k]) = rhs_{q,a}\langle \bar{q}[x_i] \leftarrow M(\bar{q}, s_i)\rangle_{\bar{q} \in Q, 1 \leq i \leq k}.$$

Thus each subtree of the form $\bar{q}[x_i]$ in $rhs_{q,a}$ is replaced by $M(\bar{q}, s_i)$. □

Without proof we note that M can also be defined by $M(q, s) = t$, such that $q[s] \Rightarrow_M^* t$. Hence, the tree transformation realized by M is $M = \{(s, M(q_d, s)) \mid s \in T_\Sigma\}$. This corresponds to a kind of "context-freeness lemma" for total deterministic top-down finite tree transducers, as follows.

Remark 3.53 Let $M = (Q, \Sigma, \Delta, R, \{q_d\})$ be a total deterministic top-down finite tree transducer and let $q \in Q$ and $s = a[s_1 \cdots s_k]$, $k \geq 0$, $a \in \Sigma_k$ and $s_1, \ldots, s_k \in T_\Sigma$, such that

$$q[a[s_1 \cdots s_k]] \Rightarrow_M rhs_{q,a}\langle x_i \leftarrow s_i\rangle_{1 \leq i \leq k} \Rightarrow_M^* t.$$

Then there exist $t_{\bar{q},j} \in T_\Delta$, with $\bar{q} \in Q$ and $1 \leq j \leq k$, such that

$$t = (rhs_{q,a}\langle x_i \leftarrow s_i\rangle_{1 \leq i \leq k})\,\langle \bar{q}[s_j] \leftarrow t_{\bar{q},j}\rangle_{\bar{q} \in Q, 1 \leq j \leq k}$$

and, for all $\bar{q} \in Q$ and $1 \leq j \leq k$, $\bar{q}[s_j] \Rightarrow_M^* t_{\bar{q},j}$ (or $M(\bar{q}, s_j) = t_{\bar{q},j}$). □

3.2.3 Comparison of B and T

The class of bottom-up finite tree transformations and the class of top-down finite tree transformations are quite different. Two properties that are specific to bottom-up finite tree transducers are the following.

Property 3.54 (B) *Non-determinism followed by copying.* A bottom-up ftt has the ability of first processing an input subtree non-deterministically and then copying the resulting output tree. □

Property 3.55 (B′) *Checking followed by deletion.* A bottom-up ftt has the ability of first processing an input subtree and then deleting the resulting output subtree. In other words, depending on a (recognizable) property of the input subtree, it can decide whether to delete the output subtree or to do something else with it.

Top-down finite tree transducers have the following specific property.

Property 3.56 (T) *Copying followed by different processing.* A top-down ftt has the ability of first copying an input subtree and then treating the resulting copies differently.

The following theorem states that both bottom-up finite tree transducers and top-down finite tree transducers can define tree transformations that can not be defined by the other class.

Theorem 3.57 The classes of bottom-up and top-down finite tree transformations are incomparable. In particular, there are tree transformations in $\mathbf{PNB} - \mathbf{T}$ and $\mathbf{PNT} - \mathbf{B}$.

Proof In [Eng74, pages 95–97]. □

The composition of finite tree transformations can lead out of the classes to which they belong.

Theorem 3.58 $\mathbf{T}$ and $\mathbf{B}$ are not closed under composition. In particular, there are tree transformations in $(\mathbf{HOM} \circ \mathbf{REL}) - \mathbf{T}$ and $(\mathbf{REL} \circ \mathbf{HOM}) - \mathbf{B}$.

Proof In [Eng74, pages 98–99]. □

3.2.4 Some results on tree transformations

This section contains some closure and decidability results about bottom-up finite tree transformations and top-down finite tree transformations. Only the results that are needed in the remainder of this thesis are presented here. Many more, and the proofs of the ones presented here, can be found in [Eng74], [Eng75] and [Eng77].

Theorem 3.59 **RECOG** is closed under inverse bottom-up and top-down tree transformations (in particular under inverse homomorphisms).

Proof In [Eng74, page 112]. $\square$

Definition 3.60 Let $\mathbf{K}$ be a class of tree transformations. A $\mathbf{K}$-*surface tree language* is a language $M(L)$ with $M \in \mathbf{K}$ and $L \in \mathbf{RECOG}$. The class of $\mathbf{K}$-surface languages will be denoted by $\mathbf{K}-\mathbf{SUR}$. As a special case, $(\mathbf{B} \cup \mathbf{T})^{*}-\mathbf{SUR}$ will be denoted by $\mathbf{SUR}$. For the definition of $(\mathbf{B} \cup \mathbf{T})^{*}$, see Definition 3.19. $\square$

Theorem 3.61 The emptiness and membership problems are solvable for **SUR**.

Proof In [Eng74, page 137]. $\square$

Theorem 3.62 The finiteness problem is solvable for **SUR**.

Proof In [Eng74, pages 138 and 139]. $\square$

Chapter four

Hypergraphs, hypergraph languages and hypergraph expressions

The concepts of graph grammars and graph languages can be seen as a generalization of the concepts of string grammars and string languages. They have applications in a wide variety of areas, among which are formal semantics and specifications, pattern recognition and even some branches of biology. Graph grammars come in three major flavors, edge rewriting grammars, node rewriting grammars and graph rewriting grammars. We will consider hypergraph grammars of the edge rewriting type, for it turns out that they have nice context-free properties. Hypergraph grammars and languages are an even more generalized concept than graph languages. The latter are a special case of the first.

In this chapter, we describe two formalisms for generating hypergraph languages. The first formalism is a straightforward rewriting grammar, where non-terminal hyperedges are replaced by hypergraphs, which may contain non-terminal hyperedges themselves. The second formalism generates hypergraphs in an indirect manner. A set of operators on hypergraphs is defined, together with a tree grammar over this set, which generates hypergraph expressions. The hypergraph language is generated by evaluating these expressions. It turns out that the two formalisms generate the same class of hypergraph languages.

4.1 Hypergraphs

A variety of different definitions of hypergraphs and hypergraph grammars is in use. Examples can be found in [EngHey91], [HabKre87], [Hab92] and [BauCou87]. We chose for the definitions in [Hab92] since this book was also the source for the predicates and numerical functions on hypergraphs and hypergraph languages. All definitions and theorems in this section and the sections 4.2 and 4.3 are from [Hab92], sometimes with a minor alteration.

A hypergraph consists of two sets, the set of nodes V and the set of hyperedges E. Each hyperedge has a number of incoming tentacles that are attached to nodes, and a number of outgoing tentacles, also attached to nodes. For each node in E, the sequence of nodes attached to the incoming tentacles is determined by the source function s and the ordered set of nodes attached to the outgoing tentacles is determined by the target function t. Furthermore, each hyperedge will be labelled by the labeling function l.

The labels are elements of a fixed infinite set, denoted by $\mathbf{\Omega}$. Every hyperedge will be labelled with an element from $\mathbf{\Omega}$.

Definition 4.1 (hypergraph) Let $\mathbf{C} \subseteq \mathbf{\Omega}$. A *hypergraph* over $\mathbf{C}$ is a 5-tuple $H = (V, E, s, t, l)$, where V is a finite set of *nodes*, E is a finite set of *hyperedges*, $s : E \to V^*$ is a mapping, assigning a sequence of *sources* $s(e)$ to each hyperedge $e \in E$, $t : E \to V^*$ is a mapping assigning a sequence of *targets* $t(e)$ to each hyperedge and $l : E \to \mathbf{C}$ is a mapping which assigns a *label* to each hyperedge. The sequence $att(e) = s(e) \cdot t(e)$ is called the *attachment* of e. The set of all nodes occurring in $att(e)$ is denoted by $\mathrm{ATT}(e)$. A hyperedge $e \in E$ is called a (m, n)-*edge* (for $m, n \in \mathbb{N}$) if $|s(e)| = m$ and $|t(e)| = n$. The pair (m, n) is the *type* of e, denoted by $type(e)$. The set

$$\{(i,j) \mid 1 \leq i,j \leq m+n, type(e) = (m,n), att(e)_i = att(e)_j\}.$$

is called the relation of e, and is denoted by $rel(e)$. □

As we shall see in later sections, hyperedges will be replaced by hypergraphs. To accomplish this it is necessary to have some distinguished nodes of a hypergraph, that can be attached to the source and target nodes of the hyperedge that must be replaced. These distinguished nodes are called the external nodes of the hypergraph.

Definition 4.2 (multi-pointed hypergraph) Let $\mathbf{C} \subseteq \mathbf{\Omega}$. A *multi-pointed hypergraph* over $\mathbf{C}$ is a 7-tuple $H = (V, E, s, t, l, begin, end)$, where (V, E, s, t, l) is a hypergraph over $\mathbf{C}$ and $begin, end \in V^*$. The set of nodes occurring in the sequence $ext_H = begin_H \cdot end_H$ is called the set of *external nodes* of H, and is denoted by EXT_H. The set $V_H - \mathrm{EXT}_H$ is called the set of *internal* nodes of H, and is denoted by INT_H. H is called a (m, n)-*hypergraph* (for $m, n \in \mathbb{N}$) if $|begin_H| = m$ and $|end_H| = n$. The *type* of H, denoted by $type(H)$ is the pair

(m, n). The set

$$\{(i,j) \mid 1 \le i,j \le m+n, \mathit{type}(H) = (m,n), \mathit{ext}_{H,i} = \mathit{ext}_{H,j}\}.$$

is called the relation of H, and is denoted by $\mathit{rel}(H)$. □

In the definitions of hypergraphs and multi-pointed hypergraphs, one hyperedge can be attached to the same node at a number of tentacles. This turns out to be undesirable. We will therefore restrict ourselves to hypergraphs where each hyperedge can be attached to each node at only one tentacle. Hypergraphs with this property are called repetition-free hypergraphs.

Definition 4.3 (repetition-freeness)

i. A multi-pointed hypergraph H is said to be *repetition-free* if $\mathit{ext}_{H,i} \neq \mathit{ext}_{H,j}$ for all $1 \le i,j \le |\mathit{ext}_H|$ with $i \neq j$. Note that for H a repetition-free hypergraph, $\mathit{rel}(H) = \{(i,i) \mid 1 \le i \le |\mathit{ext}_H|\}$.

ii. Let H be a hypergraph. Then a hyperedge $e \in E_H$ is said to be repetition free if $\mathit{att}_H(e)_i \neq \mathit{att}_H(e)_j$ for all $1 \le i,j \le |\mathit{att}_H(e)|$ with $i \neq j$. If a hyperedge e is repetition-free, then $\mathit{rel}(e) = \{(i,i) \mid 1 \le i \le |\mathit{att}_H(e)|\}$.

iii. A multi-pointed hypergraph H is said to be *completely repetition-free* if it is repetition-free and all hyperedges in E_H are repetition-free.

iv. The set of all completely repetition-free hypergraphs over $\mathbf{C}$ is denoted by $\mathbf{HG}_{\mathbf{C}}$ ($\mathbf{HG}_{\mathbf{\Omega}}$ is denoted by $\mathbf{HG}$). The set of all completely repetition-free hypergraphs over $\mathbf{C}$ of type (m, n) is denoted by $\mathbf{HG}_{\mathbf{C}}^{m,n}$ (or $\mathbf{HG}^{m,n}$, if $\mathbf{C} = \mathbf{\Omega}$). □

Assumption 4.4 From now on, we will consider completely repetition-free hypergraphs only.

An ordinary graph is a special form of a hypergraph.

Definition 4.5 (graph) Let $\mathbf{C} \subseteq \mathbf{\Omega}$. A completely repetition-free (m, n)-hypergraph H over $\mathbf{C}$ is a (m, n)*-graph* if all hyperedges of H are $(1, 1)$-hyperedges. A hyperedge in a graph is called an *edge*. The set of all graphs is denoted by $\mathbf{GR}$.□

We are usually not interested in the identity of the nodes and hyperedges of a hypergraph, and therefore do not want to distinguish between hypergraphs that are "structurally equivalent" and differ only in their nodes and hyperedges. Hypergraphs that are structurally equivalent, are called isomorphic.

Definition 4.6 (subhypergraphs, morphisms, isomorphisms)

i. Let $H, H' \in \mathbf{HG}$. Then H is called a *subhypergraph* of H', denoted by $H \subseteq H'$, if $V_H \subseteq V_{H'}$, $E_H \subseteq E_{H'}$, and $s_H(e) = s_{H'}(e)$, $t_H(e) = t_{H'}(e)$, $l_H(e) = l_{H'}(e)$ for all $e \in E_H$. Note that nothing is assumed about the relation of the external nodes.

ii. Let $H, H' \in \mathbf{HG}$. A *hypergraph morphism* h from H to H', denoted by $h : H \to H'$, consists of a pair of mappings $h = (h_V : V_H \to V_{H'}, h_E : E_H \to E_{H'})$ satisfying the conditions $h_V^*(s_H(e)) = s_{H'}(h_E(e))$, $h_V^*(t_H(e)) = t_{H'}(h_E(e))$, and $l_H(e)) = l_{H'}(h_E(e))$ for all $e \in E_H$.

iii. A hypergraph morphism is said to be an *isomorphism* from H to H' if $h_V : V_H \to V_{H'}$ and $h_E : E_H \to E_{H'}$ are bijective mappings, $h_V^*(begin_H) = begin_{H'}$, and $h_V^*(end_H) = end_{H'}$. If there is an isomorphism from H to H', then H and H' are said to be isomorphic, denoted by $H \equiv H'$. The class of all hypergraphs isomorphic to a hypergraph H is denoted by $[H]$.

A set of hypergraphs that is closed under "structural equivalence", is called a hypergraph language. If a hypergraph is in the language, then all hypergraphs that are structurally equivalent to it, are also in the language.

Definition 4.7 (hypergraph language) Let $\mathbf{C} \subseteq \mathbf{\Omega}$.

i. A set $L \subseteq \mathbf{HG_C}$ of multi-pointed hypergraphs is called a hypergraph language over $\mathbf{C}$ if it is closed under isomorphisms, i.e. if $H \in L$ and $H \equiv H'$, then $H' \in L$. In particular, $L \subseteq \mathbf{HG_C}$ is said to be finite if the number of non-isomorphic hypergraphs in L is finite.

ii. $L \subseteq \mathbf{HG_C}$ is said to be homogeneous if $type(H) = type(H')$ for all $H, H' \in L$. In this case, $type(L)$ denotes the type of the hypergraphs in L. Note that $type(H) = type(H')$ implies $rel(H) = rel(H')$, since H and H' are both repetition-free. □

Finally, a number of special types of hypergraphs will be defined. Ordinary hypergraphs can contain any number of hyperedges. There are two special types of hypergraphs, both with only one hyperedge.

Definition 4.8 (singletons and handles)

- $H \in \mathbf{HG}$ is said to be a *singleton* if $V_H = \mathrm{EXT}_H$ and $|E_H| = 1$. The single hyperedge is denoted as $e(H)$ and its label as $l(H)$.

- A singleton H with $E_H = \{e\}$, $s_H(e) = begin_H$ and $t_H(e) = end_H$ is said to be a *handle*. If $l_H(e) = A$, $type(e) = (m, n)$ then H is said to be the (m, n)-handle induced by A and is denoted by $(A, (m, n))^\bullet$ or $A(m, n)^\bullet$. In this case, H is unique up to isomorphism.

- Let $H \in \mathbf{HG}$. Then each hyperedge $e \in E_H$ induces a handle $e^\bullet$ by restricting the mappings s_H, t_H and l_H to the set $\{e\}$, restricting the set of nodes to those occurring in $s_H(e)$ and $t_H(e)$, and choosing $begin_{e^\bullet} = s_H(e)$ and $end_{e^\bullet} = t_H(e)$. Since we assume complete repetition-freeness, $e^\bullet \equiv A(m, n)^\bullet$, where $A = l(e)$ and $(m, n) = type(e)$. □

4.2 Hyperedge replacement

The replacement of a hyperedge by a hypergraph is the key construction in hyperedge replacement grammars. The replacement of a hyperedge by a hypergraph consists of the removal of the hyperedge, and the attachment of the hypergraph at the nodes to which the hyperedge was attached. In order to be replaced, the hypergraph must "fit" at the position that was previously occupied by the hyperedge. The so called "base for replacement" is used to assure that the hypergraph fits. It also provides a means for the simultaneous replacement of more than one hyperedge. Hyperedge replacement is defined formally, as follows.

Definition 4.9 (hyperedge replacement) Let $H \in \mathbf{HG}$ be a hypergraph (completely repetition-free) and $B \subseteq E_H$ be a set of hyperedges to be replaced. A mapping $rpl : B \to \mathbf{HG}$ is said to be a *base for replacement* if for all $b \in B$, $type(rpl(b)) = type(b)$. Let $H \in \mathbf{HG}$, $B \subseteq E_H$ and $rpl : B \to \mathbf{HG}$ be a base for replacement. Then the *replacement* of B in H by rpl yields a completely repetition-free multi-pointed hypergraph $X \in \mathbf{HG}$, given by

i. $V_X = V_H \cup \bigcup_{b \in B} \mathrm{INT}_{rpl(b)}$,

ii. $E_X = (E_H - B) \cup \bigcup_{b \in B} E_{rpl(b)}$,

iii. each hyperedge keeps it label,

iv. each hyperedge of $E_H - B$ keeps its sources and targets,

v. each hyperedge of $E_{rpl(b)}$, for all $b \in B$, keeps its internal sources and targets and the external ones are handed over to the corresponding sources and targets of b in H, thus for all $b \in B$ and $e \in E_{rpl(b)}$, $s_X(e) = h^*(s_{rpl(b)}(e))$ and $t_X(e) = h^*(t_{rpl(b)}(e))$, where $h : V_{rpl(b)} \to V_X$ is defined by $h(v) = v$ for $v \in \mathrm{INT}_{rpl(b)}$, $h(x_i) = s_i$ for $1 \le i \le m$, where $begin_{rpl(b)} = x_1 \cdots x_m$ and $s_H(b) = s_1 \cdots s_m$, and $h(y_j) = t_j$ for $1 \le j \le n$, where $end_{rpl(b)} = y_1 \cdots y_n$ and $t_H(b) = t_1 \cdots t_n$ (note that, since both b and $rpl(b)$ are repetition-free, $h(x_i) = h(x_j)$ only if $x_i = x_j$ and $h(y_i) = h(y_j)$ only if $y_i = y_j$),

vi. $begin_X = begin_H$ and $end_X = end_H$.

The multi-pointed hypergraph X is denoted by $\mathrm{RPL}(H, rpl)$. If $B = \{e_1, \ldots, e_n\}$ and $rpl(e_i) = R_i$ for $1 \le i \le n$ we also write $H\langle e_1 \leftarrow R_1, \ldots, e_n \leftarrow R_n\rangle$ instead of $\mathrm{RPL}(H, rpl)$. □

Although it is slightly different, this definition coincides with Definition I.2.1 in [Hab92], since, for repetition-free hypergraphs, $type(rpl(b)) = type(b)$ implies $rel(rpl(b)) = rel(b)$ if both b and $rpl(b)$ are repetition-free.

Remark 4.10

i. The construction above determines a unique hypergraph X. More precisely, X is unique up to isomorphism because the construction of the disjoint union is unique up to isomorphism.

ii. Let $h : H \to H'$ be a hypergraph morphism. If $rpl : B \to \mathbf{HG}$ is a base for replacement in H, such that if $h_E(e) = h_E(e')$ then $rpl(e) = rpl(e')$, and $h_E(B)$ is the image of B under h, then $rpl' : h_E(B) \to \mathbf{HG}$ with $rpl'(h_E(e)) = rpl(e)$ for $e \in B$ is a base for replacement in H'. Vice versa, if $rpl' : B' \to \mathbf{HG}$ is a base for replacement in H', then $rpl : h_E^{-1}(B') \to \mathbf{HG}$ with $rpl(e) = rpl(h_E(e))$ for $e \in h_E^{-1}(B')$ is a base for replacement in H.

4.3 Hypergraph grammars

A hypergraph grammar is similar to a string grammar in the sense that non-terminals are rewritten by the right-hand sides of the rules or productions. However, for string grammars, these right-hand sides consist of strings, whereas for hypergraph grammars, they consist of hypergraphs.

Let $\mathbf{Y} = \{y_1, y_2, y_3, \ldots\} \subseteq \mathbf{\Omega}$ be a set of variables. These variables will not be used until Section 4.4, but they must be defined prior to the definition of the productions in hypergraph grammars, since we do not allow them to be used as labels in hypergraph grammars.

Definition 4.11 (productions and derivations)

i. Let $N \subseteq \mathbf{\Omega} \backslash \mathbf{Y}$ be a set of non-terminals. A *production* over N is an ordered pair $p = (A, R)$ with $A \in N$ and $R \in \mathbf{HG}$. A production (A, R) will usually be written as $A \to R$. The non-terminal A is called the *left-hand side* of p, denoted by $lhs(p)$, and the completely repetition-free hypergraph R is called the right-hand side of p, denoted by $rhs(p)$.

ii. Let $H, H' \in \mathbf{HG}$, $p = A \to R$ be a production, and $e \in E_H$ be a hyperedge such that $l_H(e) = A$ and $rpl : \{e\} \to \mathbf{HG}$, given by $rpl(e) = R$ is a base for replacement. Then H *directly derives* H' by p, applied to e, if H' is isomorphic to $H\langle e \leftarrow R\rangle$. We write $H \Longrightarrow_{p,e} H'$, $H \Longrightarrow_p H'$ or $H \Longrightarrow_P H'$ provided that $p \in P$.

iii. A sequence of direct derivations $H_0 \Longrightarrow_{p_1,e_1} \cdots \Longrightarrow_{p_k,e_k} H_k$ is called a *derivation* of length k from H_0 to H_k. The derivation is shortly denoted by $H_0 \Longrightarrow_P^* H_k$, provided that $p_1, \ldots, p_k \in P$. If the length of the derivation is of interest, we write $H_0 \Longrightarrow_P^k H_k$. Additionally, in the case $H_0 \equiv H_0'$, we speak of a derivation from H_0 to H_0' of length 0. □

A hypergraph replacement grammar is a construction, where non-terminal hyperedges will be replaced by hypergraphs, by applying the previously defined productions.

Definition 4.12 (hyperedge replacement grammar)

i. A *hyperedge-replacement grammar* is a system $G = (N, T, P, Z)$, where $N \subseteq \mathbf{\Omega} \setminus \mathbf{Y}$ is a set of non-terminals, $T \subseteq \mathbf{\Omega} \setminus \mathbf{Y}$ is a set of terminals, P is a finite set of productions over N and $Z \in \mathbf{HG}$ is the axiom.

ii. The *hypergraph language* $\mathcal{L}(G)$, generated by G consists of all terminal labeled hypergraphs which can be derived from Z by applying productions of P, or

$$\mathcal{L}(G) = \{H \in \mathbf{HG}_T \mid Z \Longrightarrow_P^* H\}.$$

iii. Two hyperedge replacement grammars G and G' are said to be *equivalent* if $\mathcal{L}(G) = \mathcal{L}(G')$.

iv. Let $G = (N, T, P, Z)$ be a hyperedge replacement grammar. The type of G, denoted by $\mathit{type}(G)$, is the type of the axiom, thus $\mathit{type}(G) = \mathit{type}(Z)$.

v. The set of all hyperedge replacement grammars is denoted by $\mathbf{HRG}$. For $m, n \in \mathbb{N}$, the set of all hyperedge replacement grammars of type (m, n) is denoted by $\mathbf{HRG}^{m,n}$.

vi. Let $\mathbf{C} \subseteq \mathbf{\Omega}$. The set of all hyperedge replacement grammars over $\mathbf{C}$, denoted by $\mathbf{HRG_C}$, is defined by $\mathbf{HRG_C} = \{(N, T, P, Z) \in \mathbf{HRG} \mid T \subseteq \mathbf{C}\}$. For $m, n \in \mathbb{N}$, the set of all hyperedge replacement grammars over $\mathbf{C}$ of type (m, n), denoted by $\mathbf{HRG}_\mathbf{C}^{m,n}$, is defined by $\mathbf{HRG}_\mathbf{C}^{m,n} = \{(N, T, P, Z) \in \mathbf{HRG}^{m,n} \mid T \subseteq \mathbf{C}\}$. □

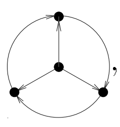
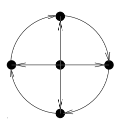
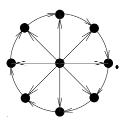
Remark 4.13 For a hyperedge replacement grammar G, the set $\mathcal{L}(G)$ is closed under isomorphisms. Hence, $\mathcal{L}(G)$ is a hypergraph language. Moreover $\mathcal{L}(G)$ is homogeneous because $\mathit{type}(H) = \mathit{type}(G)$ for all $H \in \mathcal{L}(G)$. Therefore, non-homogeneous languages cannot be generated by the grammars introduced above. □

In [Hab92], besides hyperedge replacement grammars, edge replacement grammars are considered, in order to simplify the examples of numerical functions on hypergraphs. Edge replacement grammars generate ordinary graphs.

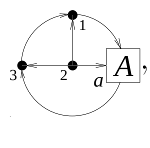
Definition 4.14 (edge replacement grammar) A hyperedge replacement grammar $G = (N, T, P, Z)$ is an *edge replacement grammar* if Z is a $(1, 1)$-graph and all right-hand sides of productions in P are $(1, 1)$-graphs. The set of all edge replacement grammars is denoted by $\mathbf{ERG}$. □

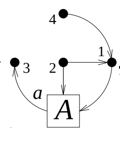
In [Hab92] the axiom Z should be a graph, not necessarily a $(1, 1)$-graph. However, [Hab92] is not always very precise about this distinction.

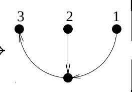
Example 4.15 This example describes a hyperedge replacement grammar that generates directed "wheel" graphs. Directed wheel graphs are directed graphs, consisting of a circular graph, the "rim", and as many spokes as there are edges on the rim. Each spoke connects the hub with one of the edges. The wheel graph with n edges on the rim is denoted by W_n. For example

$W_3 =$ , $W_4 =$  and $W_8 =$ .

In this example, $(1,1)$-hyperedges are represented by arrows, while hyperedges with type not equal to $(1,1)$, are represented by a square with the label of the hyperedge in it. The terminal labels are omitted. Now let $G_h = (N, T, P, Z)$ be a hyperedge replacement grammar where $N = \{S, A\}$, $T = \{\ell\}$, Z is a $(0,0)$-handle with label S and

$P = \{ P_0 = S \rightarrow$ 

$, P_1 = A \rightarrow$ 

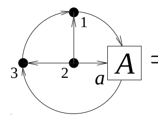
$, P_2 = A \rightarrow$ 

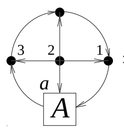
$\}$,

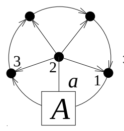
where every $(1,1)$-hyperedge is labelled with ℓ (not shown). The relevant external nodes and source and target nodes are as follows (the other nodes and hyperedges do not have a name).

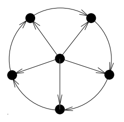
	begin	*end*	$s(a)$	$t(a)$
$rhs(P_0)$	–	–	12	3
$rhs(P_1)$	42	3	12	3
$rhs(P_2)$	12	3	–	–

Note that, although the nodes and hyperedges in the various hypergraphs have the same names, they are not the same nodes and hyperedges. Now, for example, the derivation of W_5 in G_h is

$Z = \boxed{S} \Rightarrow_{P_0}$ 

$\Rightarrow_{P_1}$ 

$\Rightarrow_{P_1}$ 

$\Rightarrow_{P_2}$ 
$= W_5$.

Clearly, $\mathcal{L}(G_h) = \{W_n \mid n \in \mathbb{N}, n \geq 3\}$, which is an infinite subset of **GR**. Note that, although G_h generates only $(0,0)$-graphs, it is not an edge replacement grammar. □

With the hypergraph grammars, a special class of hypergraph languages can be defined, namely the class of context-free hypergraph languages. The name context-free stems from the fact that the replacement process only depends on the hyperedge that is replaced, and not on its context. This will be made more precise in Theorem 4.27.

Definition 4.16 (context-free hypergraph language) Let $\mathbf{C} \subseteq \mathbf{\Omega}$, $m, n \in \mathbb{N}$ and $L \subseteq \mathbf{HG_C}$ be a hypergraph language over $\mathbf{C}$.

i. L is called a *context-free hypergraph language of type* (m,n) if $L = \mathcal{L}(G)$ for some $G \in \mathbf{HRG}_{\mathbf{C}}^{m,n}$. Note that this implies that L must be homogeneous. The set of all context-free hypergraph languages over $\mathbf{C}$ of type (m,n) is denoted by $\mathbf{CFHG}_{\mathbf{C}}^{m,n}$.

ii. L is called a *context-free hypergraph language* if $L = \bigcup_{1 \leq i \leq k} L_i$, with $L_i \in \mathbf{CFHG}_{C}^{m_i,n_i}$, for some $k \geq 1$ and $m_i, n_i \in \mathbb{N}$ for $1 \leq i \leq k$. The set of all context-free hypergraph languages over $\mathbf{C}$ is denoted by $\mathbf{CFHG_C}$.

If $\mathbf{C} = \mathbf{\Omega}$, the subscript $_\mathbf{C}$ is usually omitted. □

The definition of a context-free hypergraph language as the union of a finite number of context-free hypergraph languages of different types, seems not very natural. The reason for this definition becomes apparent in Section 4.5.

As with string grammars, it is often convenient if the constituents of a grammar comply to certain standards. If this is the case, the grammar is said to be in normal form.

Theorem 4.17 (normal form theorem) For each hyperedge replacement grammar $G = (N, T, P, Z)$, an equivalent grammar $G' = (N', T', P', Z')$ can be constructed such that N' and T' are finite, N' and T' are disjoint, $rhs(p) \in \mathbf{HG}_{N' \cup T'}$ for every $p \in P'$, and Z' is a handle in $\mathbf{HG}_{N'}$. A hyperedge replacement grammar satisfying these conditions is said to be a *usual* hyperedge replacement grammar.

Proof In [Hab92], page 25. □

Example 4.18 The hyperedge replacement grammar in Example 4.15 is in normal form. □

An even stronger standard form of grammars are typed grammars. In a typed grammar, each non-terminal has a type associated with it. If in a typed grammar, two hyperedges are labelled with the same non-terminal, they can only be replaced by hypergraphs of the same type.

Definition 4.19 (typed grammars) A usual hyperedge replacement grammar $G = (N, T, P, Z)$ is said to be *typed* if there is a mapping $ltype : N \to \mathbb{N} \times \mathbb{N}$ such that for all hypergraphs H in the grammar (i.e. the right-hand sides of productions as well as the axiom) and all hyperedges $e \in E_H$ with label in N, $ltype(l_H(e)) = type(e)$, and for all productions $A \to R \in P$, $ltype(A) = type(R)$. For $A \in N$, $ltype(A)$ is said to be the type of A. □

As with the normal form, for every grammar it is possible to find an equivalent grammar (that is a grammar that generates the same hypergraph language), which is typed.

Theorem 4.20 (typification theorem) For each hyperedge replacement grammar, an equivalent typed grammar can be constructed.

Proof In [Hab92], pages 26 and 27. □

Remark 4.21

i. Let $G = (N, T, P, Z)$ be a typed hyperedge replacement grammar and $ltype : N \to \mathbb{N} \times \mathbb{N}$ the corresponding type function. Then, for $A \in N$, $A^\bullet$ denotes the handle $(A, ltype(A))^\bullet$ induced by A and the type of A. Note that $Z = l(Z)^\bullet$.

ii. A hyperedge replacement grammar $G = (N, T, P, Z)$ is said to be *completely typed* if there is a mapping $ltype : N \cup T \to \mathbb{N} \times \mathbb{N}$ such that for all hypergraphs H in the grammar and all hyperedges $e \in E_H$, $ltype(l_H(e)) = type(e)$, and for all productions $A \to R \in P$, $ltype(A) = type(R)$. It can be shown that for each hyperedge replacement grammar G, there is a completely typed grammar G' such that $\mathcal{L}(G)$ is equal to $\mathcal{L}(G')$ up to the added type information. □

Example 4.22 The hyperedge replacement grammar in Example 4.15 is typed. The typing function $ltype : N \to \mathbb{N} \times \mathbb{N}$ is defined by $ltype(S) = (0, 0)$ and $ltype(A) = (2, 1)$. □

In [Hab92], yet another special class of hypergraph replacement grammars is introduced. This is the class of well-formed hyperedge replacement grammars. However, since we consider only completely repetition-free hypergraphs, all hyperedge replacement grammars are completely well-formed in the sense of the remark after Theorem I.4.6 in [Hab92]. From this remark we can see that restricting ourselves to completely repetition-free hypergraphs does not essentially limit the generative power of the hyperedge replacement grammars.

By the context-free nature of hyperedge replacement, it is possible to simultaneously replace an arbitrary collection of hyperedges instead of one single hyperedge.

Definition 4.23 (parallel derivations)

i. Let $H \in \mathbf{HG}$, $B \subseteq E_H$ and P be a set of productions. A mapping $prod : B \to P$ is called a *production base* in H if $l_H(b) = lhs(prod(b))$ for all $b \in B$ and $rpl : B \to \mathbf{HG}$ given by $rpl(b) = rhs(prod(b))$ is a base for replacement in H.

ii. Let $H, H' \in \mathbf{HG}$ and $prod : B \to P$ be a production base in H. Then H directly derives H' in parallel (by $prod$) if H' is isomorphic to $\mathrm{RPL}(H, rpl)$ where $rpl : B \to \mathbf{HG}$ is given by $rpl(b) = rhs(prod(b))$ for all $b \in B$. In this case we write $H \Rightarrow H'$ by $prod$ or $H \Rightarrow H'$.

iii. A sequence of direct parallel derivations $H_0 \Rightarrow \cdots \Rightarrow H_k$ by $prod_1, \ldots, prod_k$ is called a parallel derivation of length k from H_0 to H_k and is denoted by $H_0 \Rightarrow^* H_k$. If the length of the derivation shall be stressed, we write $H \Rightarrow^k H_k$. Additionally, in the case $H_0 \equiv H_0'$, we speak of a parallel derivation from H_0 to H_0' of length 0.

iv. A direct parallel derivation $H \Rightarrow H'$ by the empty base $prod : \emptyset \to P$ is called a *dummy*. A parallel derivation is said to be valid if at least one of its steps is not a dummy. □

Remark 4.24 Parallel derivations generalize the usual sequential derivations in the following sense: For each direct sequential derivation $H \Longrightarrow H'$ by $p \in P$ applied to e, there is a direct parallel derivation $H \Rightarrow H'$ by $prod : \{e\} \to P$ with $prod(e) = p$.

Each parallel derivation can be performed by a number of sequential derivations. The order in which the sequential derivations are performed is irrelevant.

Theorem 4.25 (sequentialization theorem) Let $H \Rightarrow H'$ by $prod : B \to P$ be a direct parallel derivation. Then, for each enumeration $e_1, \ldots, e_n$ of the elements in B, there is a derivation $H = H_0 \Longrightarrow \cdots \Longrightarrow H_n = H'$ by $prod(e_1), \ldots, prod(e_n)$.

Proof In [Hab92], page 45. □

Corollary 4.26 For each parallel derivation $H \Rightarrow^* H'$, there is a sequential derivation $H \Longrightarrow^* H'$. □

From now on we will consider parallel derivations only, and we will use short double arrows ($\Rightarrow$) to denote them.

As with string languages, we have a context-freeness lemma (Lemma II.2.4 in [Hab92]). This lemma states that derivations in hyperedge replacement grammars can be decomposed in "thin fibres" (where one starts from the subhypergraph induced by a hyperedge) without losing information. This context-freeness lemma can be stated in an alternative way, which is more convenient in proofs. The decomposition theorem states that each derivation can be decomposed into a number of smaller derivations. This is a generalization of a property that is well-known from ordinary context-free grammars.

Theorem 4.27 (decomposition of derivations) Let $F, H \in \mathbf{HG}$ and F be a handle. Let $k \geq 0$. Then there is a derivation $F \Rightarrow^{k+1} H$ iff there is a direct derivation $F \Rightarrow G$ and, for each $e \in E_G$ labelled with a non-terminal, there is a derivation $e^\bullet \Rightarrow^k H(e)$ such that $H = \mathrm{RPL}(G, rpl)$ with $rpl(e) = H(e)$ for $e \in E_G$.

Proof In [Hab92], page 50. □

Remark 4.28 Given a hypergraph $H \in \mathbf{HG}$, each derivation of H induces a decomposition of H into smaller hypergraphs. Let H be a hypergraph and $F \Rightarrow^{k+1} H$ be a derivation of H from the handle F. Then the derivation decomposes into a direct derivation $F \Rightarrow G$ and derivations $e^\bullet \Rightarrow^k H(e)$ with $H(e) \subseteq H$ ($e \in E_G$). For $e \in E_G$, the derivation $e^\bullet \Rightarrow^k H(e)$ may be valid or not. In the first case, it has the same form as the original derivation, but it is shorter than the original one. In the latter case, $H(e)$ is isomorphic to $e^\bullet$ and, hence, a handle. □

4.4 Hypergraph expressions

In this section, we will define an alternative method for generating hypergraph languages. The hypergraph languages are not generated by hyperedge replacement grammars, but by evaluating a set of hypergraph expressions that is generated by a regular tree grammar. For the hypergraph expressions, we define a set of hypergraph operators, called the substitution operators. Then we use regular tree grammars over these substitution operators, to generate expressions over the operators. This method for the generation of hypergraphs was first presented in [BauCou87].

For the definition of the substitution operators, recall the definition of $\mathbf{Y}$ as a set of variables $\{y_1, y_2, y_3, \ldots\} \subseteq \mathbf{\Omega}$.

Definition 4.29 (substitution operator) Let $k \geq 0$. A completely repetition-free hypergraph $H \in \mathbf{HG}^{m,n}$ is called a *substitution operator* of rank k, if there are distinct hyperedges $e_1, \ldots, e_k \in E_H$ such that

i. $l_H(e_i) = y_i$ for $1 \leq i \leq k$ and

ii. $l_H(e) \notin \mathbf{Y}$ if $e \neq e_i$ for $1 \leq i \leq k$.

These hyperedges $e_1, \ldots, e_k$ are denoted by $var_H(1), \ldots, var_H(k)$ respectively, thus $l_H(var_H(i)) = y_i$ for $1 \leq i \leq k$. A substitution operator of rank k has precisely k edges which are labeled with a variable from $\mathbf{Y}$. A substitution operator of rank 0 is a graph constant.

The (infinite) set of all substitution operators will be denoted by $\mathbf{SUB}$. The set of all substitution operators of rank k will be denoted by $\mathbf{SUB}_k$. Thus $\mathbf{SUB}_0 = \mathbf{HG}_{\mathbf{\Omega}\setminus\mathbf{Y}}$ is the set of constant multi-pointed hypergraphs. The set of all substitution operators of type (m, n) for $m, n \in \mathbb{N}$ is denoted by $\mathbf{SUB}^{m,n}$. Clearly for all $k \geq 0$, $\mathbf{SUB}_k \subseteq \mathbf{SUB}$ and for all $m, n \in \mathbb{N}$, $\mathbf{SUB}^{m,n} \subseteq \mathbf{SUB}$.

For $k, m, n \in \mathbb{N}$, a substitution operator $H \in \mathbf{SUB}_k^{m,n}$ induces a k-ary mapping

$$sub_H : \mathbf{HG}^{m_1,n_1} \times \cdots \times \mathbf{HG}^{m_k,n_k} \to \mathbf{HG}^{m,n},$$

defined, if $var_H(i) = e_i$ and $type(e_i) = type(H_i) = (m_i, n_i)$ for $1 \leq i \leq k$, by

$$sub_H(H_1, \ldots, H_k) = H\langle e_1 \leftarrow H_1, \ldots, e_k \leftarrow H_k\rangle,$$

or $sub_H(H_1, \ldots, H_k) = \mathrm{RPL}(H, rpl)$, where $rpl(e_i) = H_i$ for $1 \leq i \leq k$. Note that sub_H is a partial function $\mathbf{HG}^k \to \mathbf{HG}$. □

The name substitution operator stems from their meaning as the substitution of hypergraphs for the variables.

An expression over $\mathbf{SUB}$ is represented by a tree in the usual way, where $\mathbf{SUB}$ is considered to be a ranked alphabet. Expressions over $\mathbf{SUB}$ are trees over this alphabet.

Definition 4.30 (expressions over substitution operators) $T_{\mathbf{SUB}}$ is the set of all expressions that can be formed with the operators from $\mathbf{SUB}$. If $\Sigma \subseteq \mathbf{SUB}$, then T_Σ is the set of expressions that can be formed with the operators from Σ. If Σ is finite, then T_Σ is a tree language over Σ. For $t \in T_{\mathbf{SUB}}$, $type(t) = type(root(t))$ by definition. □

Each substitution operator of rank k in an expression, has k "arguments" (which are expressions over substitution operators), of which the values are substituted for the variables in the substitution operator. But for the value of an expression to be substituted for a variable, the type of this value and the type of the hyperedge labelled with the variable, must be equal. Expressions that have this property for all the substitution operators that make up the expression, are called well-typed expressions.

Definition 4.31 (well-typed expressions) Let $\Sigma \subseteq \mathbf{SUB}$ be a finite set of substitution operators. The set of *well-typed* expressions over Σ, denoted by WT_Σ, is the subset of T_Σ, defined as follows.

i. For each $H \in \Sigma_0$, $H \in WT_\Sigma$.

ii. For $k \geq 1$, $H \in \Sigma_k$ and $t_1, \ldots, t_k \in WT_\Sigma$, if $\mathit{type}(\mathit{var}_H(i)) = \mathit{type}(t_i)$ for $1 \leq i \leq k$, then $H[t_1 \cdots t_k] \in WT_\Sigma$.

An expression $t \in T_\Sigma$ is well-typed iff $t \in WT_\Sigma$. The set of all well-typed expressions is denoted by $WT_{\mathbf{SUB}}$. □

The set of all well-typed expressions over a finite subset Σ of $\mathbf{SUB}$, defined above, is a regular tree language.

Theorem 4.32 For every finite $\Sigma \subseteq \mathbf{SUB}$, $WT_\Sigma \in \mathbf{RECOG}$.

Proof Let Σ be a finite subset of $\mathbf{SUB}$. Now construct a regular tree grammar $G = (N, \Sigma, R, S)$, where

$$\begin{aligned} N &= \{\mathit{type}(H) \mid H \in \Sigma\} \\ &\quad \cup \{\mathit{type}(\mathit{var}_H(i)) \mid k \geq 1, H \in \Sigma_k, 1 \leq i \leq k\} \\ &\quad \cup \{S\}, \end{aligned}$$

and

$$\begin{aligned} R &= \{\mathit{type}(H) \to H[\mathit{type}(\mathit{var}_H(1)) \cdots \mathit{type}(\mathit{var}_H(k))] \mid k \geq 0, H \in \Sigma_k\} \\ &\quad \cup \{S \to \mathit{type}(H) \mid H \in \Sigma\}. \end{aligned}$$

Clearly, $\mathcal{L}(G) = WT_\Sigma$. □

For a well-typed expression over the substitution operators, the evaluation of the expression, which is performed by the bottom-up substitution of hypergraphs for variables in the substitution operators, results in a hypergraph, where the hyperedges are no longer labelled with variables.

Definition 4.33 Let $t \in T_{\mathbf{SUB}}$ be an expression over $\mathbf{SUB}$. The hypergraph that is represented by t is $\mathit{val}(t)$, where the mapping $\mathit{val} : T_{\mathbf{SUB}} \to \mathbf{HG}$ is defined recursively as follows.

i. For $H \in \Sigma_0$, $\mathit{val}(H) = H$.

ii. For $k \geq 1$, $H \in \mathbf{SUB}_k$, $t_1, \ldots, t_k \in T_{\mathbf{SUB}}$ and $type(var_H(i)) = type(t_i)$ for $1 \leq i \leq k$, $val(H[t_1 \cdots t_k]) = sub_H(val(t_1), \ldots, val(t_k))$.

It is easy to see that $type(val(t)) = type(t)$, provided that $val(t)$ is defined. Note that $val(t)$ is defined iff $t \in WT_{\mathbf{SUB}}$. □

The above definitions can also be made using the theory of universal algebra, by defining $\mathfrak{H} = (\mathbf{HG}, \alpha)$ as the $\mathbf{SUB}$-algebra where for each $H \in \mathbf{SUB}_k$, $k \geq 0$, $\alpha_k(H) : \mathbf{HG}^k \to \mathbf{HG}$ is defined by $\alpha_k(H) = sub_H$. The mapping val can then be defined as the initial homomorphism $\mathfrak{m}^{\mathfrak{H}}$ of $\mathfrak{H}$. Note that the universal algebra theory as described in Section 2.3 can not be used, since the types of the hypergraphs can not be taken into account. This problem can be solved by using so-called many-sorted algebras (see [Wec92]).

Now the alternative method for generating hypergraph languages is as follows. Let $G \in \mathbf{RTG}_{\mathbf{SUB}}$ be a regular tree grammar over $\mathbf{SUB}$. The expression language $\mathcal{L}(G)$, generated by G, evaluates to the hypergraph language $val(\mathcal{L}(G))$. In this way, the class of hypergraph languages $\{val(L) \mid L \subseteq T_{\mathbf{SUB}}, L \in \mathbf{RECOG}\}$ can be generated, according to Theorem 3.10.

4.5 Hypergraph replacement grammars versus regular tree grammars

In this section it will be shown that the class of hypergraph languages $\{val(L) \mid L \subseteq T_{\mathbf{SUB}}, L \in \mathbf{RECOG}\}$ is equal to the class of context-free hypergraph languages, $\mathbf{CFHG}$. In order to do so, some difficulties must be overcome. For example, hyperedge replacement grammars generate only homogeneous hypergraph languages, while expressions over substitution operators may generate mixed typed hypergraph languages. In order to solve this discrepancy, we will first restrict ourselves to homogeneous hypergraph languages. At the end of this section, we will generalize the results from the homogeneous case to the case of mixed typed hypergraph languages. This is why the definition of context-free hypergraph languages (Definition 4.16) was somewhat unnatural.

The tree languages over substitution operators that we used so far, have mixed types. We now introduce recognizable tree languages, where all trees have the same type.

Definition 4.34 (homogeneous regular tree language) A regular tree language $L \subseteq T_{\mathbf{SUB}}$ is said to be *homogeneous* if there is a pair $(m, n) \in \mathbb{N} \times \mathbb{N}$, such that for every $t \in L$, $type(t) = (m, n)$. From the definition of val, it can be seen that if L is homogeneous, then $val(L)$ is homogeneous. Conversely, if $L \subseteq WT_{\mathbf{SUB}}$ and $val(L)$ is homogeneous, then L is homogeneous. □

Ordinary regular tree grammars over $\mathbf{SUB}$ generate mixed typed tree languages. Therefore, single typed regular tree grammars are defined.

Definition 4.35 (single typed regular tree grammar) Let $G = (N, \Sigma, R, S)$ be a regular tree grammar in normal form, with $\Sigma \subseteq \mathbf{SUB}$. Then G is said to be *single typed*, if there is a pair $(m, n) \in \mathbb{N} \times \mathbb{N}$, such that for all rules $r \in R$, if $lhs(r) = S$, then $type(rhs(r)) = (m, n)$. The pair (m, n) is called the type of G, denoted by $type(G)$. It is clear that if $t \in \mathcal{L}(G)$, then $type(t) = type(G)$. $\square$

From these definitions it can easily be seen, that for every regular tree grammar $G \in \mathbf{RTG_{SUB}}$ in normal form, G is single typed iff $\mathcal{L}(G)$ is homogeneous. Hence, $L \subseteq T_{\mathbf{SUB}}$ is a homogeneous regular tree language iff $L = \mathcal{L}(G)$ for some single typed regular tree grammar.

Single typed regular tree grammars generate only homogeneous tree languages, but the trees from these languages are not necessarily well-typed. We now introduce a type of regular tree grammars, that generate homogeneous tree languages, that are also well-typed.

Definition 4.36 (typed regular tree grammar) A regular tree grammar in normal form $G = (N, \Sigma, R, S)$, with $\Sigma \subseteq \mathbf{SUB}$, is said to be *typed* if there is a mapping $ltype : N \to \mathbb{N} \times \mathbb{N}$, such that for all rules $A \to H[A_1 \cdots A_k] \in R$, with $k \geq 0$, $ltype(A) = type(H)$ and $ltype(A_i) = type(var_H(i))$ for $1 \leq i \leq k$. $\square$

Note that a typed regular tree grammar is also single typed, and therefore only generates homogeneous tree languages. Furthermore, for each typed regular tree grammar G, $\mathcal{L}(G) \subseteq WT_{\mathbf{SUB}}$.

Theorem 4.37 For each single typed regular tree grammar $G \in \mathbf{RTG_{SUB}}$, there is a typed regular tree grammar $G' \in \mathbf{RTG_{SUB}}$, such that $\mathcal{L}(G') = \mathcal{L}(G) \cap WT_{\mathbf{SUB}}$.

Proof Let $G = (N, \Sigma, R, S) \in \mathbf{RTG_{SUB}}$ be a single typed regular tree grammar. Now construct $G' = (N', \Sigma, R', S')$, where $N' = N \times \mathrm{TYP}$, with $\mathrm{TYP} \subseteq \mathbb{N} \times \mathbb{N}$ the set of all types occurring in G, i.e. the types of all the hypergraphs as well as the types of all the hyperedges in G. Furthermore

$$\begin{aligned} R' &= \{(A, type(H)) \to H[(A_1, type(var_H(1))) \cdots (A_k, type(var_H(k)))] \\ &\qquad \mid A \to H[A_1 \cdots A_k] \in R\} \end{aligned}$$

and $S' = (S, type(G))$. Now G' is a typed regular tree grammar, where the typing function $ltype : N' \to \mathbb{N} \times \mathbb{N}$ is defined by $ltype((A, (m, n))) = (m, n)$ for each $(A, (m, n)) \in N'$. Furthermore, it can be shown with induction to the length n of the derivations, that for all $A \in N$ and all $t \in T_\Sigma$, $A \Rightarrow^n_G t$ and t is well-typed iff $(A, type(t)) \Rightarrow^n_{G'} t$. From this it follows that $\mathcal{L}(G') = \mathcal{L}(G) \cap WT_\Sigma$. Note that G' is the "product" of G and the regular tree grammar in the proof of Theorem 4.32. $\square$

Now we have that for every homogeneous tree language L, we can find a regular tree grammar, such that all well-typed trees from L are generated by that grammar.

We will now define a relation between regular tree grammars, generating expressions over substitution operators, and hyperedge replacement grammars, such that if a regular tree grammar and a hyperedge replacement grammar are related, then the evaluation of the set of expressions generated by the regular tree grammar equals the hypergraph language generated by the hyperedge replacement grammar.

Definition 4.38 (lead and rank of a production) Let N be a set of non-terminals and $A \to R$ a production over N. Then the *lead* of R, denoted by $\mathit{lead}(R)$, is the set of hyperedges in R which are labeled with elements of N. Thus $\mathit{lead}(R) = \{e \in E_R \mid l_R(e) \in N\}$. The number of elements in $\mathit{lead}(R)$ is called the rank of R, denoted by $\mathit{rank}(R)$. □

For each production, we define a related substitution operator such that all hyperedges in the right-hand side of the production, labelled with non-terminals, are labelled with variables.

Definition 4.39 (associated grammars) Let $G_t = (N_t, \Sigma, R, S) \in \mathbf{RTG_{SUB}}$ be a regular tree grammar in normal form with $\Sigma \subseteq \mathbf{SUB}$ and let $G_h = (N_h, T, P, Z) \in \mathbf{HRG}$ be a hyperedge replacement grammar.

i. A rule $r = A \to \sigma[A_1 \cdots A_k] \in R$ and a production $p = B \to H$ over N_h are *associated*, denoted by $r \bowtie p$, if $A = B$, $\mathit{rank}(H) = k$ and $H = (V_\sigma, E_\sigma, s_\sigma, t_\sigma, l, \mathit{begin}_\sigma, \mathit{end}_\sigma)$, where $l : E_\sigma \to \mathbf{\Omega}$ is defined by

$$l(e) = \begin{cases} A_i & \text{if } e = \mathit{var}_\sigma(i) \text{ for } 1 \leq i \leq k, \\ l_\sigma(e) & \text{otherwise,} \end{cases}$$

Note that $\mathit{type}(\sigma) = \mathit{type}(H)$.

ii. G_t and G_h are associated, denoted by $G_t \bowtie G_h$, if they are both typed (and thus in their respective normal form), $N_h = N_t$, $T = \{l_H(e) \mid e \in E_H, H \in \Sigma\} \setminus \mathbf{Y}$, $P = \{p \mid r \bowtie p, r \in R\}$ and Z is a handle of type $\mathit{type}(G_t)$ with $l(Z) = S$.

iii. When $r \bowtie p$, $G_t \bowtie G_h$ or $C_t \bowtie C_h$, we will also write $p \bowtie r$, $G_h \bowtie G_t$ or $C_h \bowtie C_t$ respectively. □

The following theorem states, that for every typed hyperedge replacement grammar, we can find an associated typed regular tree grammar, and vice cersa.

Theorem 4.40 (associated grammars)

i. For every typed hyperedge replacement grammar $G_h \in \mathbf{HRG}$, there is a typed regular tree grammar $G_t \in \mathbf{RTG_{SUB}}$ such that $G_t \bowtie G_h$.

ii. For every typed regular tree grammar $G_t \in \mathbf{RTG_{SUB}}$, there is a typed hyperedge replacement grammar $G_h \in \mathbf{HRG}$, such that $G_h \bowtie G_t$.

Proof For the first part of the theorem, let $G_h = (N, T, P, Z)$ be a typed hyperedge replacement grammar. Construct a regular tree grammar $G_t = (N, \Sigma, R, S)$, where $\Sigma = \{\sigma_H \mid A \to H \in P\}$,

$$R = \{A \to \sigma_H[l_H(\mathit{var}_{\sigma_H}(1)) \cdots l_H(\mathit{var}_{\sigma_H}(\mathit{rank}(H)))] \mid A \to H \in P\}$$

and $S = l(Z)$. In this construction $\sigma_H = (V_H, E_H, s_H, t_H, l, \mathit{begin}_H, \mathit{end}_H)$, with the label function $l : E_H \to \mathbf{\Omega}$ defined by

$$l(e) = \begin{cases} y_{\pi_H(e)} & \text{if } e \in \mathit{lead}(H) \\ l_H(e) & \text{otherwise,} \end{cases}$$

where $\pi_H : \mathit{lead}(H) \to \{1, \ldots, \mathit{rank}(H)\}$ is an arbitrary but fixed bijection. The mapping *ltype* for G_h can be used to show that G_t is typed. Thus $G_t \bowtie G_h$.

For the second part, let $G_t = (N, \Sigma, R, S)$ be a typed regular tree grammar in normal form, with Σ a finite subset of **SUB**. Now construct a hyperedge replacement grammar $G_h = (N, T, P, Z)$, where T, P and Z are as in Definition 4.39ii. The typing function for G_t can be used to show that G_h is typed. Clearly $G_h \bowtie G_t$.□

Now it turns out that if a hyperedge replacement grammar and a regular tree grammar are associated, then the hypergraph language generated by the hyperedge replacement grammar equals the evaluation of the tree language over **SUB**, generated by the regular tree grammar.

Theorem 4.41 Let $G_h \in \mathbf{HRG}$ and $G_t \in \mathbf{RTG_{SUB}}$ such that $G_h \bowtie G_t$. Then $\mathcal{L}(G_h) = \mathit{val}(\mathcal{L}(G_t))$.

Proof Let $G_h = (N, T, P, Z) \in \mathbf{HRG}$ be a typed hyperedge replacement grammar and $G_t = (N, \Sigma, R, S) \in \mathbf{RTG_{SUB}}$ a typed regular tree grammar, such that $G_h \bowtie G_t$.

We will show first by induction to the length m of a derivation in G_h, that for all $A \in N$, if $A^\bullet \Rightarrow^m_{G_h} H$, with $H \in \mathbf{HG}_T$, then there is a $t \in T_\Sigma$ with $\mathit{val}(t) = H$, such that $A \Rightarrow^*_{G_t} t$.

If $A^\bullet \Rightarrow_{G_h} H$ is a direct derivation by a production $p = A \to H$, with $\mathit{rank}(H) = 0$, then we have a rule $r = A \to H \in R$, such that $p \bowtie r$, with $H \in \Sigma_0$, so clearly $A \Rightarrow_{G_t} H$ with $H = \mathit{val}(H)$.

If $A^\bullet \Rightarrow^{m+1}_{G_h} H$, then, according to Theorem 4.27, there is a direct derivation $A^\bullet \Rightarrow_{G_h} F$ and, for each $e \in \mathit{lead}(F)$, there is a derivation $e^\bullet \Rightarrow^m_{G_h} H(e)$ such that $H = \mathrm{RPL}(F, \mathit{rpl})$ with $\mathit{rpl}(e) = H(e)$ for $e \in E_F$. For the production $p = A \to F$, applied in the direct derivation, there is an associated rule $r = A \to a[l_F(\mathit{var}_a(1)) \cdots l_F(\mathit{var}_a(k))] \in R$, with $k = \mathit{rank}(F)$. By induction hypothesis, for all $e \in \mathit{lead}(F)$, $l_F(e) \Rightarrow^*_{G_t} t'$, with $\mathit{val}(t') = H(e)$ for some $t' \in T_\Sigma$. Now, since $\mathit{var}_a(i) \in E_F$, we find that $l_F(\mathit{var}_a(i)) \Rightarrow^*_{G_t} t_i$, with $t_i \in T_\Sigma$ and $\mathit{val}(t_i) = H(\mathit{var}_a(i))$ for all $1 \leq i \leq k$. Then we can apply the substitution operator a to

$t_1, \ldots, t_k$, yielding $a[t_1 \cdots t_k]$. Using the context-freeness of regular tree grammars and the rule r, we then find that $A \Rightarrow^*_{G_t} a[t_1 \cdots t_k]$. Furthermore

$$\begin{aligned} val(a[t_1 \cdots t_k]) &= sub_a(val(t_1), \ldots, val(t_k)) \\ &= sub_a(H(var_a(1)), \ldots, H(var_a(k))) \\ &= \mathrm{RPL}(a, rpl), \\ &\qquad \text{with } rpl(var_a(i)) = H(var_a(i)) \text{ for } 1 \leq i \leq k \\ &= \mathrm{RPL}(F, rpl), \text{ with } rpl(e) = H(e) \text{ for } e \in lead(F) \\ &= H. \end{aligned}$$

So finally we find that there is a $t \in T_\Sigma$ with $val(t) = H$ and $A \Rightarrow^*_{G_t} t$, namely $t = a[t_1 \cdots t_k]$.

Now for all $A \in N$, if $A^\bullet \Rightarrow^*_{G_h} H$ then $A \Rightarrow^*_{G_t} t$, with $val(t) = H$, for some $t \in T_\Sigma$, so this holds especially for $l(Z) = S$, and therefore $\mathcal{L}(G_h) \subseteq val(\mathcal{L}(G_t))$.

Now it remains to show that $val(\mathcal{L}(G_t)) \subseteq \mathcal{L}(G_h))$. The proof of this inclusion is the reverse of the first part of the proof.

We will show by induction to the length m of a derivation in G_t, that for all $A \in N$, if $A \Rightarrow^m_{G_t} t$, with $t \in T_\Sigma$ and $val(t) = H$, $H \in \mathbf{HG}$, then $A^\bullet \Rightarrow^*_{G_h} H$.

If $A \Rightarrow_{G_t} t$ with $t \in \Sigma_0$ (thus $t = val(t) \in \mathbf{HG}$) is a direct derivation by a rule $r = A \rightarrow t$, then there is a production $p = A \rightarrow H \in P$, such that $r \bowtie p$, with $rank(H) = 0$ and $t = H$. So clearly $A^\bullet \Rightarrow_{G_h} H$.

Now let $A \Rightarrow^{m+1}_{G_t} t$, with $val(t) = H$. Since the language generated by G_t is given by $\mathcal{L}(G'_t)$, where $G'_t = (N, \Sigma \cup \{[,]\}, R, Z)$ is a context free grammar, this derivation can be decomposed to $A \Rightarrow_{G_t} a[A_1 \cdots A_k] \Rightarrow^m_{G_t} t$, where $k \geq 1$, $a \in \Sigma_k$, $A_i \in N$, $A_i \Rightarrow^{m_i}_{G_t} s_i$, $m_i \leq m$ and $s_i \in T_\Sigma$ for $1 \leq i \leq k$, and $t = a[s_1 \cdots s_k]$. The applied rule for the direct derivation is $r = A \rightarrow a[A_1 \cdots A_k]$. Let $p = A \rightarrow F \in P$ be such that $r \bowtie p$. Thus, for all $1 \leq i \leq k$, by induction hypothesis $A_i^\bullet \Rightarrow^*_{G_h} val(s_i)$. Now $p = A \rightarrow F$ gives us a direct derivation $A^\bullet \Rightarrow_{G_h} F$. Furthermore, for each $e \in E_F$, if $l(e) = A_i$ there is a derivation $e^\bullet \Rightarrow^*_{G_h} val(s_i)$. Now let $rpl : lead(F) \rightarrow \mathbf{HG}$ be such that, for all $1 \leq i \leq k$, $rpl(var_a(i)) = val(s_i)$. Then

$$\begin{aligned} \mathrm{RPL}(F, rpl) &= \mathrm{RPL}(a, rpl) \\ &= sub_a(val(s_1), \ldots, val(s_k)) \\ &= val(a[s_1 \cdots s_k]) \\ &= val(t) \\ &= H. \end{aligned}$$

Thus we have a direct derivation $A^\bullet \Rightarrow_{G_h} F$ and for each $e \in lead(F)$ there is a derivation $e^\bullet \Rightarrow^*_{G_h} H(e)$ such that $H = \mathrm{RPL}(F, rpl)$, with $rpl(e) = H(e)$, where, for each $e \in E_F$ labelled with a non-terminal, $H(e) = val(s_i)$ if $e = var_s(i)$ for some $1 \leq i \leq k$. Now we can apply the decomposition theorem for hyperedge replacement grammars (Theorem 4.27), to find $A^\bullet \Rightarrow_{G_h} F \Rightarrow^*_{G_h} H$ or $A^\bullet \Rightarrow^*_{G_h} H$.

Again, if we apply the result to S and $l(Z)$, we find that $val(\mathcal{L}(G_t)) \subseteq \mathcal{L}(G_h))$, so we finally find that $\mathcal{L}(G_h) = val(\mathcal{L}(G_t))$. $\square$

The converse of Theorem 4.41 is clearly not true.

We are now in the position to prove the fact, that the two hypergraph language generating formalisms, generate the same hypergraph languages. First we restrict ourselves to the homogeneous case.

Theorem 4.42 For every hypergraph language L, L is homogeneous and $L \in$ **CFHG** iff $L = val(L')$ for some homogeneous regular tree language $L' \subseteq T_{\mathbf{SUB}}$.

Proof Let $L \in \mathbf{CFHG}$ be a homogeneous hypergraph language and G_h a typed hyperedge replacement grammar such that $L = \mathcal{L}(G_h)$. Such a G_h exists, since L is the union of a finite number of homogeneous context-free hypergraph languages of the same type, and for fixed $m, n \in \mathbb{N}$, $\mathbf{CFHG}^{m,n}$ is closed under union. Let G_t be a typed regular tree grammar such that $G_h \bowtie G_t$. Such a G_t exists according to Theorem 4.40. From Theorem 4.41 it follows that $\mathcal{L}(G_h) = val(\mathcal{L}(G_t))$, thus $L = val(\mathcal{L}(G_t))$, where $\mathcal{L}(G_t) \subseteq T_{\mathbf{SUB}}$ is a homogeneous regular tree language.

Conversely, let L be such that $L = val(L')$ for some homogeneous regular tree language $L' \subseteq T_{\mathbf{SUB}}$. Let G_t be a single typed regular tree grammar such that $L' = \mathcal{L}(G_t)$ and let G'_t be a typed regular tree grammar such that $\mathcal{L}(G'_t) = \mathcal{L}(G_t) \cap WT_{\mathbf{SUB}}$. Such a G'_t exists by Theorem 4.37. By the definition of *val*, we find that $val(\mathcal{L}(G'_t)) = val(\mathcal{L}(G_t))$. Now let G_h be a typed hyperedge replacement grammar such that $G_h \bowtie G'_t$. Such a G_h exists according to Theorem 4.40. From Theorem 4.41 we find that $\mathcal{L}(G_h) = val(\mathcal{L}(G'_t)) = val(\mathcal{L}(G_t)) = val(L') = L$. Thus G_h generates L and therefore $L \in \mathbf{CFHG}$. Furthermore, L is homogeneous since L' is homogeneous. □

In order to show a generalized version of Theorem 4.42, we must be able to dissect a non-homogeneous regular tree language into a number of homogeneous regular tree languages. This can be accomplished by dissecting the regular tree grammar that generates the language into a number of regular tree grammars, each generating a homogeneous tree language. In order to dissect the regular tree grammar, it must have a special property, defined as follows.

Definition 4.43 (proper regular tree grammar) A regular tree grammar $G = (N, \Sigma, R, S)$ is called *proper*, if the initial non-terminal S does not occur in the right-hand side of any rule in R.

This special property does not restrict us, since for each regular tree grammar, we can find an equivalent regular tree grammar, that has this property.

Theorem 4.44 For every regular tree grammar G, there is an equivalent regular tree grammar G', such that G' is proper and in normal form.

Proof Let $G = (N, \Sigma, R, S)$ be a regular tree grammar. From Theorem 3.12 we find that we may assume that G is in normal form. Now construct a regular tree grammar $G' = (N \cup \{S'\}, \Sigma, R', S')$, where S' is any symbol such that $S' \notin N \cup \Sigma$ and $R' = R \cup \{S' \to rhs(r) \mid r \in R, lhs(r) = S\}$. Clearly, G' is proper and in normal form, and $\mathcal{L}(G) = \mathcal{L}(G')$. □

Now we finally come to the main result of this chapter. For each hypergraph language, it turns out that the language is a context-free hypergraph language if and only if it is the evaluation of a regular tree language over **SUB**.

Theorem 4.45 For every hypergraph language L, $L \in \mathbf{CFHG}$ iff $L = val(L')$ for some regular tree language $L' \subseteq T_{\mathbf{SUB}}$.

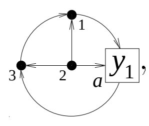
Proof Let $L \in \mathbf{CFHG}$. Then by definition, $L = \cup_{1 \le i \le k} L_i$, with $L_i \in \mathbf{CFHG}^{m_i, n_i}$ for some $k \ge 1$ and $m_i, n_i \in \mathbb{N}$ for $1 \le i \le k$. Now each L_i, $1 \le i \le k$, is homogeneous, so we can apply Theorem 4.42 to find that $L_i = val(L'_i)$ with $L'_i \subseteq T_{\mathbf{SUB}}$ a regular tree language, for $1 \le i \le k$. Since **RECOG** is closed under union, the tree language L', such that $L' = \cup_{1 \le i \le k} L'_i$, is regular. Now $val(L') = val(\cup_{1 \le i \le k} L'_i) = \cup_{1 \le i \le k} val(L'_i) = \cup_{1 \le i \le k} L_i = L$, so $L = val(L')$ for a regular tree language $L' \subseteq T_{\mathbf{SUB}}$.

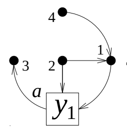
Conversely, let L be such that $L = val(L')$ for some regular tree language $L' \subseteq T_{\mathbf{SUB}}$. Let $G = (N, \Sigma, R, S)$ be a regular tree grammar, such that $L' = \mathcal{L}(G)$. From Theorem 4.44 we find that we may assume that G is proper and in normal form. Let $\Upsilon = \{type(rhs(r)) \mid r \in R, lhs(r) = S\}$ be the set of types that occur in L'. Note that Υ is finite, since R is finite. Now for each $\upsilon \in \Upsilon$, construct a $G_\upsilon = (N, \Sigma, R_\upsilon, S)$, where $R_\upsilon = \{r \in R \mid \text{if } lhs(r) = S \text{ then } type(rhs(r)) = \upsilon\}$. Clearly, $L' = \cup_{\upsilon \in \Upsilon} \mathcal{L}(G_\upsilon)$. Now let $L_\upsilon = val(\mathcal{L}(G_\upsilon))$ for $\upsilon \in \Upsilon$. Since for each $\upsilon \in \Upsilon$, G_υ is single typed, we find that $\mathcal{L}(G_\upsilon)$ is homogeneous for each $\upsilon \in \Upsilon$, so we can apply Theorem 4.42 to find that $L_\upsilon \in \mathbf{CFHG}$ and L_υ is homogeneous for each $\upsilon \in \Upsilon$. Now $L = val(L') = val(\cup_{\upsilon \in \Upsilon} \mathcal{L}(G_\upsilon)) = \cup_{\upsilon \in \Upsilon} val(\mathcal{L}(G_\upsilon)) = \cup_{\upsilon \in \Upsilon} L_\upsilon$. Thus L is the union of a finite number of homogeneous context-free hypergraph languages, and thus $L \in \mathbf{CFHG}$. □

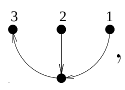
This theorem will be illustrated by an example.

Example 4.46 Let $\mathbf{W} = \{W_n \mid n \in \mathbb{N}, n \ge 3\}$ be the set of directed wheel graphs from Example 4.15. Since this graph language was generated by the hyperedge replacement grammar G_h from this example, we find that $\mathcal{L}(G_h) = \mathbf{W} \in \mathbf{CFHG}$. Now from Theorem 4.45 we find that there is a regular tree language $L \subseteq T_{\mathbf{SUB}}$, such that $\mathbf{W} = val(L)$.

Construct $G_t = (N, \Sigma, R, \{S\})$, where $N = \{S, A\}$, $\Sigma_0 = \{\sigma_0\}$ and $\Sigma_1 = \{\sigma_1, \sigma_2\}$, with

$\sigma_0 =$ 

$, \sigma_1 =$ 

and $\sigma_2 =$ 

,

with the relevant external nodes and source and target nodes as follows.

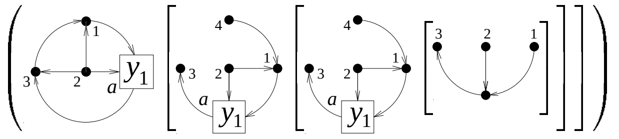
	begin	*end*	$s(a)$	$t(a)$
σ_0	–	–	12	3
σ_1	42	3	12	3
σ_2	12	3	–	–

Furthermore, let $R = \{R_0 = S \to \sigma_0[A], R_1 = A \to \sigma_1[A], R_2 = A \to \sigma_2\}$. Clearly $G_h \bowtie G_t$. The regular tree language generated by G_t is $\mathcal{L}(G_t) = \{\sigma_0([\sigma_1)^m[\sigma_2]()]^m \mid m \in \mathbb{N}\}$. For example, the derivation of $t = \sigma_0[\sigma_1[\sigma_1[\sigma_2]]]$ in G_t is

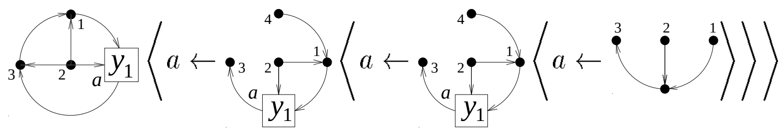
$$S \Rightarrow_{R_0} \sigma_0[A] \Rightarrow_{R_1} \sigma_0[\sigma_1[A]] \Rightarrow_{R_1} \sigma_0[\sigma_1[\sigma_1[A]]] \Rightarrow_{R_2} \sigma_0[\sigma_1[\sigma_1[\sigma_2]]] = t.$$

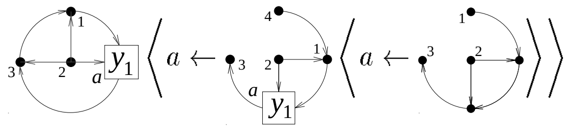
Now t evaluates to a hypergraph as follows.

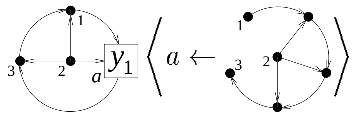
$$val(t) \quad = \quad val$$

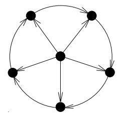


$$\begin{aligned}
&= val(\sigma_0[\sigma_1[\sigma_1[\sigma_2]]])\\
&= sub_{\sigma_0}\,(sub_{\sigma_1}\,(sub_{\sigma_1}\,(\sigma_2)))\\
&= \sigma_0\,\langle a \leftarrow \sigma_1\,\langle a \leftarrow \sigma_1\,\langle a \leftarrow \sigma_2\rangle\rangle\rangle
\end{aligned}$$

$=$ 

$=$ 

$=$ 

$=$ 

$= W_5$.

From this example we can see that $val(\mathcal{L}(G_t)) = \mathbf{W} = \mathcal{L}(G_h)$. Note the strong similarity between the hypergraph expression $\sigma_0[\sigma_1[\sigma_1[\sigma_2]]]$ and the derivation of W_5 in G_h, in Example 4.15. $\square$

fourtysix

Chapter five

Numerical functions on hypergraphs

In this chapter, a number of numerical functions on hypergraphs is defined, such as a function for the number of hyperedges in a hypergraph, a function for the number of nodes in a hypergraph and a function for the number of paths in a hypergraph. It is shown that these functions can be computed by a top-down tree transducer that receives a tree, evaluating to the given hypergraph, as input, and produces as output an expression in an algebra, that represents the arithmetic calculus with constants, addition, multiplication, maximum, minimum and raising to a power. The numerical evaluation of these expressions results in the numerical value of the functions.

The numerical functions that are defined in this chapter determine the number of hyperedges in a hypergraph, the number of nodes in a hypergraph and the number of undirected and directed paths in a $(1,1)$-graph.

Before defining the numerical function on the hypergraph expressions, we first have to define the arithmetic calculus. The calculus is defined for addition, multiplication, maximum, minimum and raising to a power, but it can easily be extended with more operations.

Definition 5.1 (arithmetic calculus) The set of *arithmetic symbols*, denoted by $\mathbf{\Gamma}^{\mathbb{N}}$, is the infinite ranked alphabet defined by $\mathbf{\Gamma}_0^{\mathbb{N}} = \{C_n \mid n \in \mathbb{N}\} \cup \{\oplus, \odot\}$, $\mathbf{\Gamma}_2^{\mathbb{N}} = \{\oplus, \odot, \mathbf{max}, \mathbf{pow}\}$ and $\mathbf{\Gamma}_k^{\mathbb{N}} = \{\oplus, \odot, \mathbf{max}\}$ for $k = 1$ and for all $k \geq 3$. The *arithmetic calculus* is the $\mathbf{\Gamma}^{\mathbb{N}}$-algebra $\mathfrak{N} = (\mathbb{N}, \alpha)$, where the operations are defined as follows.

i. For all $C_n \in \mathbf{\Gamma}_0^{\mathbb{N}}$, $\alpha_0(C_n) = n$.

ii. For $\oplus \in \mathbf{\Gamma}_0^{\mathbb{N}}$, $\alpha_0(\oplus) = 0$ and for $\odot \in \mathbf{\Gamma}_0^{\mathbb{N}}$, $\alpha_0(\odot) = 1$.

iii. For $\oplus \in \mathbf{\Gamma}_k^{\mathbb{N}}$, $k \geq 1$, $\alpha_k(\oplus) : \mathbb{N}^k \to \mathbb{N}$ is defined by

$$\alpha_k(\oplus)(v_1, \ldots, v_k) = \textstyle\sum_{1 \leq i \leq k} v_i.$$

iv. For $\odot \in \mathbf{\Gamma}_k^{\mathbb{N}}$, $k \geq 1$, $\alpha_k(\odot) : \mathbb{N}^k \to \mathbb{N}$ is defined by

$$\alpha_k(\odot)(v_1, \ldots, v_k) = \textstyle\prod_{1 \leq i \leq k} v_i.$$

v. For $\mathbf{max} \in \mathbf{\Gamma}_k^{\mathbb{N}}$, $k \geq 1$, $\alpha_k(\mathbf{max}) : \mathbb{N}^k \to \mathbb{N}$ is defined by

$$\alpha_k(\mathbf{max})(v_1, \ldots, v_k) = max\{v_1, \ldots, v_k\}.$$

vi. For $\mathbf{pow} \in \mathbf{\Gamma}_2^{\mathbb{N}}$, $\alpha_2(\mathbf{pow}) : \mathbb{N}^2 \to \mathbb{N}$ is defined by $\alpha_2(\mathbf{pow})(v, w) = v^w$. □

Since the rules in the top-down finite tree transducers that define the functions have a tendency to get complicated, and since substitution operators of various rank occur in the expressions, we will use a short-hand notation like the notation that is often used for the addition of a large number of values ($\sum_{1 \leq i \leq k} v_i$ as a short-hand for $v_1 + v_2 + \cdots + v_k$).

Notation 5.2 Let $k \geq 1$, $A \in \mathbf{\Gamma}_k^{\mathbb{N}} \backslash \{\mathbf{pow}\}$, S a finite set such that $|S| = k$, $\varphi : S \to \mathcal{O}$ a mapping from S to some set $\mathcal{O}$ of syntactical objects and $\pi : \{1, \ldots, k\} \to S$ a bijection. Then we use $A^{\pi}_{s \in S}[\![\varphi(s)]\!]$ as a short-hand for $A[\varphi(\pi(1)) \cdots \varphi(\pi(k))]$. Since all operators in $\mathbf{\Gamma}_k^{\mathbb{N}} \setminus \{\mathbf{pow}\}$ are commutative, the order imposed by π is irrelevant. Therefore it will be omitted, yielding $A_{s \in S}[\![\varphi(s)]\!]$. If $k = 0$, $A \in \mathbf{\Gamma}_0$ and $S = \varnothing$, then by $A^{\pi}_{s \in S}[\![\varphi(s)]\!]$ we mean the symbol A. □

5.1 Number of hyperedges

In this section we will present a top-down finite tree transducer that realizes a function, to determine the number of hyperedges in a hypergraph. The finite tree transducer is defined for an arbitrary subset of **SUB**. Why this is done will be made clear in Chapter 6.3.

Let Σ be a finite subset of **SUB**. Now construct a top-down tree transducer $\epsilon_\Sigma = (Q, \Sigma, \mathbf{\Gamma}^{\mathbb{N}}, R, Q_d)$, where $Q = \{q\}$ and $Q_d = \{q\}$. Furthermore, R is defined as follows.

i. For each $H \in \Sigma_0$, the rule $q[H] \to C_{|E_H|}$ is in R.

ii. For each $H \in \Sigma_k$, $k \geq 1$, the rule

$$q[H[x_1 \cdots x_k]] \to \oplus[C_{|E_H|-k} q[x_1] \cdots q[x_k]]$$

is in R.

The following theorem states that the above defined top-down finite tree transducer realizes a function for the number of hyperedges in the evaluated hypergraph for a hypergraph expression. The proof of this theorem is given in detail, as is the proof for the theorem in the next section. Since the proofs of these theorems are all basically the same, the proofs for the other theorems are not given in detail.

Theorem 5.3 For all $s \in T_\Sigma$ and $t \in T_{\mathbf{\Gamma}^{\mathbb{N}}}$, if $q[s] \Rightarrow^*_{\epsilon_\Sigma} t$, then $|E_{val(s)}| = \mathfrak{m}^{\mathfrak{N}}(t)$.

Proof The proof is by induction on s. Assume $q[s] \Rightarrow^*_{\epsilon_\Sigma} t$. If $s \in \Sigma_0$, then there is a direct derivation and the applied rule is $q[s] \to C_{|E_s|}$. Now, since $val(s) = s$ and $\mathfrak{m}^{\mathfrak{N}}(C_{|E_s|}) = |E_s|$, clearly $|E_{val(s)}| = \mathfrak{m}^{\mathfrak{N}}(t)$.

Now for $s = H[s_1 \cdots s_k]$, $k \geq 1$, $H \in \Sigma_k$ and $s_1, \ldots, s_k \in T_\Sigma$, let the first rule applied in the derivation $q[H[s_1 \cdots s_k]] \Rightarrow^*_{\epsilon_\Sigma} t$ be

$$q[H[x_1 \cdots x_k]] \to \oplus[C_{|E_H|-k} q[x_1] \cdots q[x_k]],$$

thus

$$q[H[s_1 \cdots s_k]] \Rightarrow_{\epsilon_\Sigma} \oplus[C_{|E_H|-k} q[s_1] \cdots q[s_k]] \Rightarrow^*_{\epsilon_\Sigma} t.$$

Now from Remark 3.53 it can be seen that there exist $t_1, \ldots, t_k \in T_{\mathbf{\Gamma}^{\mathbb{N}}}$ such that $t = \oplus[C_{|E_H|-k} t_1 \cdots t_k]$ and, for all $1 \leq i \leq k$, $q[s_i] \Rightarrow^*_{\epsilon_\Sigma} t_i$. By induction $|E_{val(s_i)}| = \mathfrak{m}^{\mathfrak{N}}(t_i)$ for $1 \leq i \leq k$. So we find that

$$\begin{aligned} |E_{val(s)}| &= |E_{val(H[s_1 \cdots s_k])}| \\ &= |E_{H\langle var_H(1) \leftarrow val(s_1), \ldots, var_H(k) \leftarrow val(s_k)\rangle}| \\ &= |E_H| - k + \textstyle\sum_{1 \leq i \leq k} |E_{val(s_i)}| \qquad (5.4) \end{aligned}$$

$$\begin{aligned} &= \mathfrak{m}^{\mathfrak{N}}(C_{|E_H|-k}) + \textstyle\sum_{1\leq i\leq k} \mathfrak{m}^{\mathfrak{N}}(t_i) \\ &= \mathfrak{m}^{\mathfrak{N}}(\oplus[C_{|E_H|-k}t_1\cdots t_k]) \\ &= \mathfrak{m}^{\mathfrak{N}}(t), \end{aligned}$$

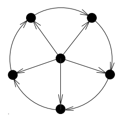
which proves the theorem. Equality 5.4 follows from the hypergraph theoretical fact that if X is the hypergraph $\mathrm{RPL}(H, \mathit{rpl})$, that results from the replacement of B in H by rpl, where $\mathit{rpl} : B \to \mathbf{HG}$ is a base for replacement, then $|E_X| = |E_H| - |B| + \sum_{e\in B} |E_{\mathit{rpl}(e)}|$. □

The top-down tree transducer ϵ is linear, non-deleting, pure and total deterministic, or $\epsilon_\Sigma \in \mathbf{PNLD_tT}$. Note that also $\epsilon_\Sigma \in \mathbf{HOM}$, since $\mathbf{PNLD_tT} \subseteq \mathbf{PD_tT} = \mathbf{HOM}$ (Remark 3.51).

Example 5.5 Let Σ be a finite subset of $\mathbf{SUB}$ such that $\sigma_0, \sigma_1, \sigma_2 \in \Sigma$, where σ_0, σ_1 and σ_2 are defined in Example 4.46. Then

$$q[s] = q[\sigma_0[\sigma_1[\sigma_1[\sigma_2]]]] \Rightarrow^*_{\epsilon_\Sigma} \oplus[C_3\oplus[C_2\oplus[C_2\oplus[C_3]]]] = t.$$

Furthermore

$\mathit{val}(s) = \mathit{val}(\sigma_0[\sigma_1[\sigma_1[\sigma_2]]]) =$  $= W_5$ and

$$\mathfrak{m}^{\mathfrak{N}}(t) = \mathfrak{m}^{\mathfrak{N}}(\oplus[C_3\oplus[C_2\oplus[C_2\oplus[C_3]]]]) = 3+2+2+3 = 10 = |E_{W_5}|,$$

so clearly, $|E_{\mathit{val}(s)}| = \mathfrak{m}^{\mathfrak{N}}(t)$. □

5.2 Number of nodes

In this section a top-down finite tree transducer that realizes a function for the number of nodes in a hypergraph is defined. Let Σ be a finite subset of $\mathbf{SUB}$. Now construct a top-down tree transducer $\nu_\Sigma = (Q, \Sigma, \mathbf{\Gamma}^{\mathbb{N}}, R, Q_d)$, where $Q = \{q, i\}$ and $Q_d = \{q\}$. Furthermore, R is defined as follows.

i. For each $H \in \Sigma_0$, the rules $q[H] \to C_{|V_H|}$ and $i[H] \to C_{|\mathrm{INT}_H|}$ are in R.

ii. For each $H \in \Sigma_k$, $k \geq 1$, the rules

$$q[H[x_1\cdots x_k]] \to \oplus[C_{|V_H|}i[x_1]\cdots i[x_k]] \text{ and}$$

$$i[H[x_1\cdots x_k]] \to \oplus[C_{|\mathrm{INT}_H|}i[x_1]\cdots i[x_k]]$$

are in R.

Informally, state q computes an expression for the number of nodes in H, while state i computes an expression for the number of internal nodes in H.

The proof of the theorem for the function that determines the number of nodes in a hypergraph is also given in detail. The proof is essentially similar to the proof of Theorem 5.3, but the finite tree transducer has two states instead of one. Therefore, the two statements in the proof (one for each state) are proven by simultaneous induction. This is not really necessary, since the two statements are not mutually dependent on each other, but it demonstrates the proof method in general. Also the tree transformation that is realized by the tree transducer is used as a function, as in Definition 3.52, instead of the more operational method used for the proof of Theorem 5.3. This is possible since the finite tree transducer is total deterministic, and therefore the tree transformation is a total function.

Theorem 5.6 For every $s \in T_\Sigma$, $\mathfrak{m}^{\mathfrak{N}}(\nu_\Sigma(q,s)) = |V_{val(s)}|$ and $\mathfrak{m}^{\mathfrak{N}}(\nu_\Sigma(i,s)) = |\mathrm{INT}_{val(s)}|$.

Proof Both statements are proven simultaneously, by induction on s. If $s \in \Sigma_0$, then there are direct derivations with applied rules $q[s] \rightarrow C_{|V_s|}$ and $i[s] \rightarrow C_{|\mathrm{INT}_s|}$. Furthermore, $val(s) = s$, thus

$$\mathfrak{m}^{\mathfrak{N}}(\nu_\Sigma(q,s)) = \mathfrak{m}^{\mathfrak{N}}(C_{|V_s|}) = |V_s| = |V_{val(s)}| \text{ and}$$

$$\mathfrak{m}^{\mathfrak{N}}(\nu_\Sigma(i,s)) = \mathfrak{m}^{\mathfrak{N}}(C_{|\mathrm{INT}_s|}) = |\mathrm{INT}_s| = |\mathrm{INT}_{val(s)}|.$$

If $s = H[s_1 \cdots s_k]$, $k \geq 1$, $H \in \Sigma_k$, $s_1, \ldots, s_k \in T_\Sigma$, then, using the fact that ν_Σ is total deterministic and the induction hypothesis $\mathfrak{m}^{\mathfrak{N}}(\nu_\Sigma(i,s_j)) = |\mathrm{INT}_{val(s_j)}|$ for $1 \leq j \leq k$, we find

$$\begin{aligned}
\mathfrak{m}^{\mathfrak{N}}(\nu_\Sigma(q,s)) &= \mathfrak{m}^{\mathfrak{N}}(\bigoplus[C_{|V_H|}\nu_\Sigma(i,s_1)\cdots\nu_\Sigma(i,s_k)]) \\
&= \mathfrak{m}^{\mathfrak{N}}(C_{|V_H|}) + \textstyle\sum_{1\leq j\leq k} \mathfrak{m}^{\mathfrak{N}}(\nu_\Sigma(i,s_j)) \\
&= |V_H| + \textstyle\sum_{1\leq j\leq k} |\mathrm{INT}_{val(s_j)}| \\
&= |V_{H\langle var_H(1)\leftarrow val(s_1),\ldots,var_H(k)\leftarrow val(s_k)\rangle}| \\
&= |V_{val(H[s_1\cdots s_k])}| \\
&= |V_{val(s)}|
\end{aligned}$$

and

$$\begin{aligned}
\mathfrak{m}^{\mathfrak{N}}(\nu_\Sigma(i,s)) &= \mathfrak{m}^{\mathfrak{N}}(\bigoplus[C_{|\mathrm{INT}_H|}\nu_\Sigma(i,s_1)\cdots\nu_\Sigma(i,s_k)]) \\
&= \mathfrak{m}^{\mathfrak{N}}(C_{|\mathrm{INT}_H|}) + \textstyle\sum_{1\leq j\leq k} \mathfrak{m}^{\mathfrak{N}}(\nu_\Sigma(i,s_j)) \\
&= |\mathrm{INT}_H| + \textstyle\sum_{1\leq j\leq k} |\mathrm{INT}_{val(s_j)}| \\
&= |\mathrm{INT}_{H\langle var_H(1)\leftarrow val(s_1),\ldots,var_H(k)\leftarrow val(s_k)\rangle}| \\
&= |\mathrm{INT}_{val(H[s_1\cdots s_k])}| \\
&= |\mathrm{INT}_{val(s)}|.
\end{aligned}$$

Here we use the hypergraph theoretical fact that, if $X = \mathrm{RPL}(H, rpl)$, resulting from the replacement of B in H by rpl, where $rpl : B \rightarrow \mathbf{HG}$ is a base for replacement, then $|V_X| = |V_H| + \sum_{e\in B} |\mathrm{INT}_{rpl(e)}|$ and $|\mathrm{INT}_X| = |\mathrm{INT}_H| + \sum_{e\in B} |\mathrm{INT}_{rpl(e)}|$. $\square$

The top-down tree transducer ν_Σ is linear, non-deleting and total deterministic, or $\nu_\Sigma \in \mathbf{NLD_tT}$.

5.3 Number of paths

In order to define a tree transducer to compute the number of paths through a hypergraph, we first have to define the notion of a path.

Definition 5.7 (paths) Let $H = (V, E, s, t, l, \mathit{begin}, \mathit{end})$ be a hypergraph and let $P = v_0 e_1 v_1 e_2 \cdots e_{n-1} v_{n-1} e_n v_n \in V(EV)^+$ be a sequence of alternating nodes and hyperedges. The length of P, denoted by $\mathit{len}(P)$, is the number of hyperedges in the sequence, thus $\mathit{len}(P) = n$.

i. If $v_{i-1} \in \mathit{att}_H(e_i)$ and $v_i \in \mathit{att}_H(e_i)$ for $1 \leq i \leq n$, then P is called an *undirected path* through H.

ii. If $v_{i-1} \in s(e_i)$ and $v_i \in t(e_i)$ for $1 \leq i \leq n$, then P is called a *directed path* through H. Note that every directed path is also an undirected path.

iii. P is called *simple*, if $v_i = v_j$ only if $i = j$ for $0 \leq i, j \leq n$.

iv. The set of all hyperedges in P is denoted by E_P and the set of all nodes in P is denoted by V_P.

v. If $P = v_0 e_1 \cdots e_n v_n$ is a path, then hyperedge e_i is denoted by $e_{P,i}$ for $1 \leq i \leq n$ and node v_i is denoted by $v_{P,i}$ for $0 \leq i \leq n$.

vi. For $1 \leq i, j \leq |\mathit{ext}_H|$ the set of all simple undirected paths through H, from $\mathit{ext}_{H,i}$ to $\mathit{ext}_{H,j}$, is denoted by $\mathrm{UP}_H^{i,j}$. Since only simple paths are allowed, $\mathrm{UP}_H^{i,j} = \varnothing$ if $i = j$. The set of all simple undirected paths through H, between two nodes from ext_H, is denoted by UP_H. Note that $\mathrm{UP}_H = \cup_{1 \leq i,j \leq |\mathit{ext}_H|} \mathrm{UP}_H^{i,j}$ and that UP_H is finite.

vii. For $1 \leq i, j \leq |\mathit{ext}_H|$ the set of all simple directed paths through H, from $\mathit{ext}_{H,i}$ to $\mathit{ext}_{H,j}$, is denoted by $\mathrm{DP}_H^{i,j}$. Again, $\mathrm{DP}_H^{i,j} = \varnothing$ if $i = j$. The finite set of all simple directed paths through H, between two nodes from ext_H, is denoted by DP_H. □

Example 5.8 Let H be the multi-pointed hypergraph in Figure 5.1, with $V_H = \{1, \ldots, 7\}$, $E_H = \{a, b, c, d\}$, $\mathit{begin}_H = 12$ and $\mathit{end}_H = 6$. The labels are omitted. Then

$$\begin{aligned}\mathrm{DP}_H &= \{1a3c6, 1a4c6, 1a4d6, 2a3c6, 2a4c6, 2a4d6, 2b4c6, 2b4d6, 2b5d6,\\ &\quad\; 2b4d7b5d6, 2b5d7b4c6\}\end{aligned}$$

and

$$\mathrm{DP}_H^{1,3} = \{1a3c6, 1a4c6, 1a4d6\}.$$

Simple undirected paths are, for example, $2b7d4a3c6$ and $2a1$. □

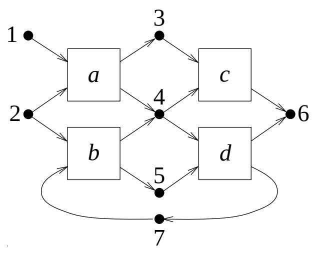


Figure 5.1: Hypergraph H

We will restrict ourselves to paths in $(1,1)$-graphs. The general case for paths through arbitrary hypergraphs is very complicated. This is due to the fact that a non-terminal hyperedge should in general occur more than once on a single path, in order to generate parts of a path in the terminal graph. If this is the case and the hyperedge is replaced by a hypergraph, the paths through this hypergraph may violate the conditions for the resulting paths being simple. This is illustrated in Figure 5.2. The hyperedge labelled A in Figure 5.2i is twice on the path $1A2e3A4$. If this hyperedge is replaced by the hypergraph in Figure 5.2ii, the path $1a5b2e3d6c4$ will still be simple. However, if the hyperedge is replaced by the hypergraph in Figure 5.2iii, the conditions for the paths being simple are violated, since the paths $1a5b2$ and $3d5c4$ are simple, but the path $1a5b2e3d5c4$ is not.

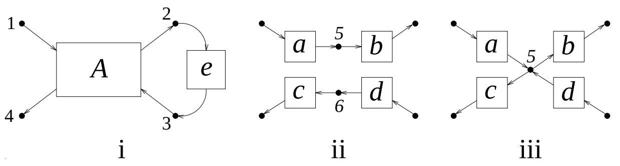


Figure 5.2: Problem with arbitrary paths

5.3.1 Simple undirected paths in (1, 1)-graphs

Let Σ be a finite subset of $\mathbf{SUB}^{1,1} \cap \mathbf{GR}$. Now construct a top-down tree transducer $\rho_\Sigma = (Q, \Sigma, \mathbf{\Gamma}^{\mathbb{N}}, R, Q_d)$ where $Q = Q_d = \{q\}$. The set of rules R is defined as follows.

i. For each $H \in \Sigma_0$, the rule $q[H] \to C_{|\mathrm{UP}_H|}$ is in R.

ii. For each $H \in \Sigma_k$, $k \geq 1$, the rule

$$\begin{aligned} q[H[x_1 \cdots x_k]] \quad \to \quad & \oplus[\odot[\varphi_H(e_{P_1,1}) \cdots \varphi_H(e_{P_1,len(P_1)})] \cdots \\ & \odot[\varphi_H(e_{P_{|\mathrm{UP}_H|},1}) \cdots \varphi_H(e_{P_{|\mathrm{UP}_H|},len(P_{|\mathrm{UP}_H|})})]] \end{aligned}$$

is in R, where $\mathrm{UP}_H = \{P_1, \ldots, P_{|\mathrm{UP}_H|}\}$[1]. Furthermore, φ_H is defined as

$$\varphi_H(e) = \begin{cases} q[x_i] & \text{if } l_H(e) = y_i \text{ for some } 1 \leq i \leq k \\ C_1 & \text{otherwise.} \end{cases}$$

Theorem 5.9 For all $s \in T_\Sigma$ and $t \in T_{\Gamma^{\mathbb{N}}}$, if $q[s] \Rightarrow^*_{\rho_\Sigma} t$, then $|\mathrm{UP}_{val(s)}| = \mathfrak{m}^{\mathfrak{N}}(t)$.

Proof The proof is similar to the proof of Theorem 5.3, but now we use the fact that if X is the graph that results from the replacement of B in H by rpl, where $rpl : B \to \mathbf{HG}$ is a base for replacement, then

$$|\mathrm{UP}_X| = \textstyle\sum_{P \in \mathrm{UP}_H} \prod_{e \in E_P} \Phi(e),$$

where $\Phi(e) = |\mathrm{UP}_{rpl(e)}|$ if $e \in B$ and $\Phi(e) = 1$ otherwise. This formula for the number of paths can be explained as follows. The product term comes from the fact that if an edge on a path is replaced by a graph, the path is "replaced" by as many paths as there are through the graph. Clearly the total number of paths through the graph is the total, over all the paths through H, of all the "replaced" paths. □

The rules can be denoted in a more compact way, using Notation 5.2. Doing so, for $H \in \Sigma_k$, $k \geq 1$, the corresponding rule is denoted by

$$q[H[x_1 \cdots x_k]] \to \oplus_{P \in \mathrm{UP}_H} [\![\odot_{e \in E_P} [\![\varphi_H(e)]\!]]\!],$$

where φ_H is as defined above.

The top-down tree transducer ρ is pure and total deterministic, or $\rho \in \mathbf{PD_tT} = \mathbf{HOM}$. ρ need not be linear since one hyperedge may occur on more than one path. It need not be non-deleting, since not every hyperedge necessarily occurs on a path.

5.3.2 Simple directed paths in (1, 1)-graphs

The case for directed paths is slightly more complicated, because an edge can be replaced by a graph that "points in the other direction". We must therefore use a weaker form of directed paths, where edges that may be replaced are not used to determine whether or not the path is directed. At the time of the substitution, the paths should not be counted if the substitution would yield an undirected path.

Definition 5.10 Let $H \in \mathbf{SUB}_k$ be a substitution operator of rank k and let P be a path through H. Then P is called a *partial directed* path if, for every $1 \leq u \leq len(P)$ such that $l_H(e_{P,u}) \notin \mathbf{Y}$, $v_{P,u-1} \in s_H(e_{P,u})$ and $v_{P,u} \in t_H(e_{P,u})$. For $1 \leq i, j \leq |ext_H|$, the set of all simple partial directed paths through H from $ext_{H,i}$ to $ext_{H,j}$, is denoted by $\mathrm{PDP}^{i,j}_H$. □

[1]If $\mathrm{UP}_H = \varnothing$, the rule reduces to $q[H[x_1 \cdots x_k]] \to \oplus$.

Let Σ be a finite subset of $\mathbf{SUB}^{1,1} \cap \mathbf{GR}$. Now construct a top-down tree transducer $\delta_\Sigma = (Q, \Sigma, \mathbf{\Gamma}^{\mathbb{N}}, R, Q_d)$ where $Q = \{q_{12}, q_{21}\}$ and $Q_d = \{q_{12}\}$. The set of rules R is defined as follows.

i. For each $H \in \Sigma_0$, the rules $q_{12}[H] \to C_{|\mathrm{DP}_H^{1,2}|}$ and $q_{21}[H] \to C_{|\mathrm{DP}_H^{2,1}|}$ are in R.

ii. For each $H \in \Sigma_k$, $k \geq 1$, the rules

$$q_{12}[H[x_1 \cdots x_k]] \to \oplus_{P \in \mathrm{PDP}_H^{1,2}} [\![\odot_{1 \leq u \leq len(P)} [\![\varphi_H(P,u)]\!]]\!] \text{ and}$$

$$q_{21}[H[x_1 \cdots x_k]] \to \oplus_{P \in \mathrm{PDP}_H^{2,1}} [\![\odot_{1 \leq u \leq len(P)} [\![\varphi_H(P,u)]\!]]\!]$$

are in R, where φ_H is defined as

$$\varphi_H(P,u) = \begin{cases} q_{12}[x_i] & \text{if } l_H(e) = y_i, 1 \leq i \leq k \wedge v_{P,u-1} \in s_H(e_{P,u}) \\ q_{21}[x_i] & \text{if } l_H(e) = y_i, 1 \leq i \leq k \wedge v_{P,u-1} \notin s_H(e_{P,u}) \\ C_1 & \text{otherwise.} \end{cases}$$

Theorem 5.11 For all $s \in T_\Sigma$ and $t \in T_{\mathbf{\Gamma}^{\mathbb{N}}}$, if $q_{12}[s] \Rightarrow^*_{\delta_\Sigma} t$, then $|\mathrm{DP}^{1,2}_{val(s)}| = \mathfrak{m}^{\mathfrak{N}}(t)$.

Proof The proof is almost equal to the proof of Theorem 5.9, but now we must consider partial directed paths, since hyperedges on a partial directed path labelled with a variable from $\mathbf{Y}$ will be replaced by a hypergraph, from which only paths in the proper direction should be counted. Since there are two states, it must be proven simultaneously that for all $s \in T_\Sigma$ and $t \in T_{\mathbf{\Gamma}^{\mathbb{N}}}$, if $q_{21}[s] \Rightarrow^*_{\delta_\Sigma} t$, then $|\mathrm{DP}^{2,1}_{val(s)}| = \mathfrak{m}^{\mathfrak{N}}(t)$. □

The top-down tree transducer δ_Σ is total deterministic, or $\delta_\Sigma \in \mathbf{D_t T}$. δ_Σ need not be linear since one hyperedge may occur on more than one path. It need not be non-deleting, since not every hyperedge necessarily occurs on a path.

fiftysix

Chapter six

Compatible and realizable functions

The numerical functions on hypergraphs of Chapter 5 are all realized by top-down tree transducers, which translate the expressions that represent the hypergraphs into numerical expressions, which are then evaluated to obtain the function values. In [Hab92], numerical functions (and predicates) are defined on the derivations of hypergraphs in hyperedge replacement grammars. To accomplish this, the functions had to be "compatible" with the derivation process, which means that the function value for a hypergraph depends on the hypergraph itself and on the function values for the hypergraphs that are replaced for the non-terminal hyperedges of the hypergraph.

In Chapter 4 we have seen that the two formalisms for the generation of hypergraphs which were presented, are equivalent. In this chapter we will present the realizability property for functions on hypergraphs, which is similar to the compatibility property but is based on top-down tree transducers. It will be shown that each compatible function is also a realizable function.

Remark 6.1 All functions in the remainder of this thesis are meant to be total.

6.1 Compatible functions, Habel

The definitions of compatible functions and special compatible functions were introduced in [HabKreVog91] and later recalled in [Hab92]. The definitions, respectively Definition VII.2.1 and Definition VII.2.3 in [Hab92], are as follows.

Definition 6.2 (compatible functions, Habel) Let $C \subseteq \mathbf{HRG}$, I a finite index set and $\mathbf{V}$ a set of values.

i. Let $f : \mathbf{HG} \times I \to \mathbf{V}$ be a function on pairs $(H, i) \in \mathbf{HG} \times I$ and f' a function on triples (R, ass, i) with $R \in \mathbf{HG}$, $ass : E_R \times I \to \mathbf{V}$ and $i \in I$, with values in $\mathbf{V}$. Then f is called *(C, f')-compatible* if, for all $G = (N, T, P, Z) \in C$ and all derivations of the form $A^\bullet \Rightarrow R \Rightarrow^* H$, with $A \in N \cup T$, $H \in \mathbf{HG}_T$ and for all $i \in I$, $f(H, i) = f'(R, ass, i)$, where $ass : E_R \times I \to \mathbf{V}$ is given by $ass(e, j) = f(H(e), j)$, $e \in E_R$ and $j \in I$.

ii. A function $f_0 : \mathbf{HG} \to \mathbf{V}$ is called *C-compatible* if functions f and f' and an index i_0 exist, such that f is (C, f')-compatible and $f_0 = f(-, i_0)$[1].

iii. A function $f : \mathbf{HG} \times I \to \mathbb{N}^\perp$ is said to be *$(C, min, max, +, \cdot)$-compatible* if there exists a function f' such that f is (C, f')-compatible and for each right-hand side R of some production in C and each $i \in I$, $f'(R, -, i)$ corresponds to an expression formed with variables $ass(e, j)$, $e \in E_R$ and $j \in I$, and constants from $\mathbb{N}$ by addition, multiplication, minimum and maximum. The function f is *$(C, max, +, \cdot)$-compatible* if the operation min does not occur.

iv. A function $f_0 : \mathbf{HG} \to \mathbb{N}^\perp$ is *$(C, min, max, +, \cdot)$-compatible* if a function f and an index i_0 exist such that f is $(C, min, max, +, \cdot)$-compatible and $f_0 = f(-, i_0)$. Accordingly, the function f_0 is *$(C, max, +, \cdot)$-compatible* if f is $(C, max, +, \cdot)$-compatible. □

In this definition, $\mathbb{N}^\perp = \mathbb{N} \cup \{\perp\}$, where $\perp$ is a special value, meaning that the value is undefined. We will not use this special value.

The compatible functions are defined on a class of hyperedge replacement grammars. The compatibility property must hold for every step of every derivation in every hyperedge replacement grammar in the class. The parts i and ii of the definition do not have a restriction on the function f'. In the parts iii and iv, for the special compatible functions, this function must be representable by an expression formed with the operators $+$, $\cdot$, max and min.

[1] $f(-, i_0)$ denotes the function given by $f(-, i_0)(H) = f(H, i_0)$ for all $H \in \mathbf{HG}$.

6.2 Compatible functions

It turns out that Definition 6.2 is not suitable for our purposes, without some changes. We first note that derivations in hyperedge replacement grammars starting in terminal labeled edges, as occuring in part i of the definition, consist of dummy steps only, and thus are of no interest.

A more serious problem is that the definition of the special compatible functions (part iii) contains the informal phrase

> "for each right-hand side R of some production in C and each $i \in I$, $f'(R, -, i)$ corresponds to an expression formed with variables $ass(e, j)$, $e \in E_R$ and $j \in I$, and constants from $\mathbb{N}$ by addition, multiplication, minimum and maximum"

which is not precise enough for our purposes. In [Hab92] this informal definition did not cause any troubles, but in Section 6.3 we have to use, and manipulate, these expressions in a proof. Therefore we need a more formal definition of compatible functions. We define the expressions with the aid of universal algebra theory (recall Section 2.3), which results in the following definitions.

Definition 6.3 (compatible functions) Let $C \subseteq \mathbf{HRG}$, $\mathbf{V}$ a set of values and $\mathfrak{A} = (\mathbf{V}, \alpha)$ a Γ-algebra.

i. Let I be a finite index set, $f : \mathbf{HG} \times I \to \mathbf{V}$ and

$$f' = \{f'_{U,i} : (E_U \times I \to \mathbf{V}) \to \mathbf{V}\}_{U \in \mathbf{HG}, i \in I}$$

a family of functions. Then f is called (C, f')-*compatible* if, for all $G = (N, T, P, Z) \in C$ and all derivations $A^\bullet \Rightarrow_G U \Rightarrow^*_G H$, with $A \in N$, $H \in \mathbf{HG}_T$ and for all $i \in I$, $f(H, i) = f'_{U,i}(\gamma)$, where the assignment function $\gamma : E_U \times I \to \mathbf{V}$ is given by $\gamma(e, j) = f(H(e), j)$ for all $(e, j) \in E_U \times I$, where, for each $e \in E_U$, $H(e)$ is defined by the decomposition theorem (Theorem 4.27). Note that for $e \notin \mathit{lead}(U)$, $H(e) = e^\bullet$ and thus $f(H(e), j)$ is a constant value in $\mathbf{V}$.

ii. Let I be a finite index set. A function $f : \mathbf{HG} \times I \to \mathbf{V}$ is called $(C, \mathfrak{A})$-*compatible* if there exists a family of functions $f' = \{f'_{U,i}\}_{U \in \mathbf{HG}, i \in I}$ such that f is (C, f')-compatible and for each right-hand side U of some production in C and each $i \in I$, a $\Gamma(E_U \times I)$-expression $s_{U,i} \in T_\Gamma(E_U \times I)$ can effectively be found, such that $f'_{U,i} = s^{\mathfrak{A}}_{U,i}$. Such an expression $s_{U,i}$ is said to *correspond* to U and i, or to $f'_{U,i}$. Moreover, for each hyperedge e in C and for each $i \in I$, T_Γ must contain an expression $\kappa_{e,i}$ such that $\mathfrak{m}^{\mathfrak{A}}(\kappa_{e,i}) = f(e^\bullet, i)$.

iii. A function $f_0 : \mathbf{HG} \to \mathbf{V}$ is $(C, \mathfrak{A})$-*compatible* if a finite index set I, a function $f : \mathbf{HG} \times I \to \mathbf{V}$ and an index $i_0 \in I$ exist, such that f is $(C, \mathfrak{A})$-compatible and $f_0 = f(-, i_0)$. □

Note that the f' function in Definition 6.2 is replaced by a family of functions, which is also denoted by f'. Furthermore, Definition 6.3 is more general than Definition 6.1, since it allows for expressions in algebras with arbitrary signature, instead of expressions consisting of only variables, $+$, $\cdot$, max, min and the constants. When the algebra $\mathfrak{A}$ in the definition is the algebra $\mathfrak{N}$ for the arithmetic calculus, as defined in Definition 5.1, with the possible addition of min and without raising to a power, the two definitions are almost equivalent. We do not need the special value for undefined, since all our functions are always defined.

6.3 Realizable functions

As already mentioned in the introduction of this chapter, we will define a property for functions on hypergraphs that is closely related to the compatibility property. But where the compatibility property was defined for a class of hyperedge replacement grammars, this new property, called the realizability property, is defined for a subset of the set of all substitution operators. Informally, a function is realizable if it can be realized by a total deterministic top-down tree transducer, which transduces expressions over a subset of **SUB** (which evaluate to hypergraphs), into expressions in an algebra.

Definition 6.4 (realizable functions) Let $\mathfrak{A} = (\mathbf{V}, \alpha)$ be a Γ-algebra and $\Delta \subseteq$ **SUB**. A function $f_0 : \mathbf{HG} \to \mathbf{V}$ is called $(\Delta, \mathfrak{A})$-*realizable*, if for every finite $\Sigma \subseteq \Delta$, a total deterministic top-down finite tree transducer $\mu = (Q, \Sigma, \Gamma, R, \{q_d\})$ can effectively be constructed, such that for all $t \in WT_\Sigma$ it holds that $\mathfrak{m}^{\mathfrak{A}}(\mu(q_d, t)) = f_0(val(t))$. □

Remark 6.5 The numerical functions in Chapter 5 are all $(\mathbf{SUB}, \mathfrak{N})$-realizable.□

The compatibility property is defined on a class of hyperedge replacement grammars, whereas the realizability property is defined on a set of substitution operators. The following definition will associate a class of hyperedge replacement grammars with a set of substitution operators, in a way similar to the associated grammars in Definition 4.39.

Definition 6.6 (associated grammars) Let $\Delta \subseteq$ **SUB**. The class of all Δ-*associated hyperedge replacement grammars*, denoted by $\mathbf{HRG}^{\bowtie}_{\Delta}$, is the class of all grammars $(N, T, P, Z) \in \mathbf{HRG}$, such that for all $A \to H$ with $A \to H \in P$ or $H = Z$, there exists a $\sigma \in \Delta$ and $A_1, \ldots, A_k \in \mathbf{\Omega}$, such that $A \to \sigma[A_1 \cdots A_k] \bowtie A \to H$, where $k = rank(\sigma)$. Clearly, for all $\Delta \subseteq \mathbf{SUB}$, $\mathbf{HRG}^{\bowtie}_{\Delta} \subseteq \mathbf{HRG}$. □

Intuitively, $\mathbf{HRG}^{\bowtie}_{\Delta}$ consists of all hyperedge replacement grammars such that the axiom and the right-hand sides of the productions are in Δ (modulo $\bowtie$).

Remark 6.7 The two special sets of hypergraph replacement grammars that we know, **HRG** and **ERG** can be "generated" from a set of substitution operators as follows.

i. For $\mathbf{HRG}^{\bowtie}_{\Delta}$ to be equal to $\mathbf{HRG}$, it is necessary that Δ contains, for every rule in $\mathbf{HRG}$, a substitution operator that is associated with the rule. Thus $\mathbf{HRG} = \mathbf{HRG}^{\bowtie}_{\mathbf{SUB}}$.

ii. For $\mathbf{HRG}^{\bowtie}_{\Delta} = \mathbf{ERG}$ the case is more complicated, since in $\mathbf{ERG}$, the axioms must be graphs, whereas $\mathbf{HRG}^{\bowtie}_{\Delta}$ has no restrictions on the axioms. Therefore, we must add this restriction, yielding

$$\mathbf{ERG} = \{(N,T,P,Z) \in \mathbf{HRG}^{\bowtie}_{\mathbf{SUB}^{1,1}\cap\mathbf{GR}} \mid Z \in \mathbf{GR}\}.$$

From now on we will only consider $(C,\mathfrak{A})$-compatibility for classes $C = \mathbf{HRG}^{\bowtie}_{\Delta}$. This is a restriction with respect to [Hab92], but in [Hab92] the only classes that are really used, are $\mathbf{HRG}$ and $\mathbf{ERG}$. □

Using Definition 6.6, we arrive at the following theorem, which states that every compatible function over a class of hyperedge replacement grammars, as defined in Definition 6.6, can be realized by a total deterministic top-down tree transducer, which transduces an expression over $\mathbf{SUB}$ into an expression over the underlying algebra.

Theorem 6.8 Let $\Delta \subseteq \mathbf{SUB}$, $\mathfrak{A} = (\mathbf{V},\alpha)$ a Γ-algebra and $f_0 : \mathbf{HG} \to \mathbf{V}$ a $(\mathbf{HRG}^{\bowtie}_{\Delta},\mathfrak{A})$-compatible function. Then f_0 is $(\Delta,\mathfrak{A})$-realizable.

Proof Let I be the finite index set, $f : \mathbf{HG} \times I \to \mathbf{V}$ the function, $i_0 \in I$ the index and $f' = \{f'_{U,i}\}_{U\in\mathbf{HG},i\in I}$ the family of functions corresponding to the $(\mathbf{HRG}^{\bowtie}_{\Delta},\mathfrak{A})$-compatible function f_0. Furthermore, let Σ be an arbitrary finite subset of Δ. Now construct a total deterministic top-down tree transducer $\mu = (I,\Sigma,\Gamma,R,\{i_0\})$ and a hyperedge replacement grammar $G = (\{A\},\mathbf{\Omega},P,Z)$, with $A \in \mathbf{\Omega}$ an arbitrary label and Z an arbitrary singleton such that $l(Z) = A$, where R and P are defined as follows. For each $H \in \Sigma_k$, $k \geq 0$ and each $i \in I$, let $A \to U_H$ be a production such that $A \to H[A\cdots A] \bowtie A \to U_H$ and let $s_{U_H,i} \in T_\Gamma(E_{U_H} \times I)$ be the $\Gamma(E_{U_H}\times I)$-expression corresponding to U_H and i. U_H can be determined effectively from H, by relabeling the hyperedges $var_H(\ell)$ for $1 \leq \ell \leq k$. Note that $E_H = E_{U_H}$ and $type(H) = type(U_H)$. Now, for each $H \in \Sigma_k$, $k \geq 0$, the production $A \to U_H$ is in P. Furthermore, for each $H \in \Sigma_k$, $k \geq 0$ and each $i \in I$, the rule

$$i[H[x_1\cdots x_k]] \to s^{\mathfrak{F}_\Gamma(I[\mathbf{X}_k])}_{U_H,i}(\varphi_{H,i})$$

is in R, where $\mathfrak{F}_\Gamma(I[\mathbf{X}_k])$ is the free Γ-algebra which is generated by $I[\mathbf{X}_k]$ and $s^{\mathfrak{F}_\Gamma(I[\mathbf{X}_k])}_{U_H,i}(\varphi_{H,i}) \in T_\Gamma(I[\mathbf{X}_k])$ is the result of the substitution of $\varphi_{H,i}(e,j)$ for all variables $(e,j) \in E_H \times I$ in $s_{U_H,i}$. The assignment function $\varphi_{H,i} : E_H \times I \to T_\Gamma(I[\mathbf{X}_k])$ is defined by

$$\varphi_{H,i}(e,j) = \begin{cases} j[x_\ell] & \text{if } e = var_H(\ell) \text{ for some } 1 \leq \ell \leq k \\ \kappa_{e,j} & \text{otherwise.} \end{cases}$$

Clearly, the hyperedge replacement grammar G is in $\mathbf{HRG}^{\bowtie}_{\Delta}$. Note that $\mathcal{L}(G)$ is not uniquely determined since Z, and in particular its type, is arbitrary. This will cause no problems, since we are only interested in the derivations in G, not in $\mathcal{L}(G)$.

We will now show that for every $i \in I$ and every $t \in WT_\Sigma$, $\mathfrak{m}^{\mathfrak{A}}(\mu(i,t)) = f(\mathit{val}(t), i)$. The proof is by induction to the structure of t, where it is simultaneously shown that for every $t = \sigma[t_1 \cdots t_k] \in WT_\Sigma$, with $t_1, \ldots, t_k \in WT_\Sigma$, there is a derivation $A^\bullet \Rightarrow_G U_\sigma \Rightarrow_G^* \mathit{val}(t)$.

First the case for $t \in \Sigma_0$. Let $H = \mathit{val}(t) = t$, thus $H \in \Sigma_0 \subseteq \Delta_0$. By the definition of G, we have a production $A \to U_H$, with $U_H = H$ since $\mathit{lead}(U_H) = \varnothing$, and thus a direct derivation $A^\bullet \Rightarrow_G H$. Since f is $(\mathbf{HRG}_\Delta^{\bowtie}, f')$-compatible, $G \in \mathbf{HRG}_\Delta^{\bowtie}$ and $\mathit{lead}(H) = \varnothing$, we find that $f(H,i) = f'_{H,i}(\gamma)$, with $\gamma(e,j) = f(e^\bullet, j)$ for all $(e,j) \in E_H \times I$. Furthermore, since $\mathit{rank}(t) = 0$, $\mathfrak{m}^{\mathfrak{A}}(\varphi_{H,i}(e,j)) = \mathfrak{m}^{\mathfrak{A}}(\kappa_{e,j}) = f(e^\bullet, j)$ for all $(e,j) \in E_H \times I$ and thus $\gamma = \mathfrak{m}^{\mathfrak{A}} \circ \varphi_{H,i}$. So we find that

$$\begin{aligned}
\mathfrak{m}^{\mathfrak{A}}(\mu(i,t)) &= \mathfrak{m}^{\mathfrak{A}}(\mu(i,H)) \\
&= \mathfrak{m}^{\mathfrak{A}}(s^{\mathfrak{F}_\Gamma}_{U_H,i}(\varphi_{H,i})) \\
&= s^{\mathfrak{A}}_{U_H,i}(\mathfrak{m}^{\mathfrak{A}} \circ \varphi_{H,i}) \\
&= s^{\mathfrak{A}}_{U_H,i}(\gamma) \\
&= f'_{H,i}(\gamma) \\
&= f(H,i),
\end{aligned}$$

which concludes the induction basis.

Now the case for $t = \sigma[t_1 \cdots t_k]$, $k \geq 1$, $\sigma \in \Sigma_k$, $t_1, \ldots, t_k \in WT_\Sigma$. Let

$$\begin{aligned}
H &= \mathit{val}(t) \\
&= \mathit{val}(\sigma[t_1 \cdots t_k]) \\
&= \mathit{sub}_\sigma(\mathit{val}(t_1), \ldots, \mathit{val}(t_k)) \\
&= \mathrm{RPL}(\sigma, \mathit{rpl}), \text{ where } \mathit{rpl}(\mathit{var}_\sigma(\ell)) = \mathit{val}(t_\ell) \text{ for } 1 \leq \ell \leq k.
\end{aligned}$$

Now, by one of the induction hypotheses, for each $\mathit{val}(t_\ell)$, $1 \leq \ell \leq k$, we have a derivation $A^\bullet \Rightarrow_G^* \mathit{val}(t_\ell)$. By the definition of G, we have a production $A \to U_\sigma$, with $A \to U_\sigma \bowtie A \to \sigma[A \cdots A]$. Recall that $E_{U_\sigma} = E_\sigma$ and $\mathit{type}(U_\sigma) = \mathit{type}(\sigma)$. We now have a direct derivation $A^\bullet \Rightarrow_G U_\sigma$ and, for all $e \in \mathit{lead}(U_\sigma)$, a derivation $e^\bullet \Rightarrow_G^* H(e)$, where $H(e) = \mathit{val}(t_\ell)$ such that $e = \mathit{var}_\sigma(\ell)$. Hence, $\mathrm{RPL}(U_\sigma, \mathit{rpl}) = H$, where for all $e \in \mathit{lead}(U_\sigma)$, $\mathit{rpl}(e) = H(e)$. We now can apply the decomposition theorem (Theorem 4.27) to find that there is a derivation $A^\bullet \Rightarrow_G U_\sigma \Rightarrow_G^* \mathit{val}(t)$. Since f is $(\mathbf{HRG}_\Delta^{\bowtie}, f')$-compatible and $G \in \mathbf{HRG}_\Delta^{\bowtie}$, we have $f(H,i) = f'_{U_H,i}(\gamma)$, with $\gamma : E_{U_H} \times I \to \mathbf{V}$ given by $\gamma(e,j) = f(H(e), j)$ for all $(e,j) \in E_{U_H} \times I$. So we find that

$$\begin{aligned}
\mathfrak{m}^{\mathfrak{A}}(\mu(i,t)) &= \mathfrak{m}^{\mathfrak{A}}(\mu(i, \sigma[t_1 \cdots t_k])) \\
&= \mathfrak{m}^{\mathfrak{A}}(s^{\mathfrak{F}_\Gamma(I[\mathbf{X}_k])}_{U_\sigma,i}(\varphi_{\sigma,i})^{\mathfrak{F}_\Gamma}(\psi)),
\end{aligned}$$

where $\psi : I[\mathbf{X}_k] \to T_\Gamma$ is defined by $\psi(j[x_\ell]) = \mu(j, t_\ell)$ for $1 \leq \ell \leq k$. Now by the associativity of the composition of substitutions, we may combine $\varphi_{\sigma,i}$ and ψ into one substitution $\varphi'_{\sigma,i}$, yielding

$$\mathfrak{m}^{\mathfrak{A}}(\mu(i,t)) = \mathfrak{m}^{\mathfrak{A}}(s^{\mathfrak{F}_\Gamma(I[\mathbf{X}_k])}_{U_\sigma,i}(\varphi_{\sigma,i})^{\mathfrak{F}_\Gamma}(\psi))$$

$$\begin{aligned} &= \mathfrak{m}^{\mathfrak{A}}(s^{\mathfrak{F}_\Gamma}_{U_\sigma,i}(\varphi'_{\sigma,i})) \\ &= s^{\mathfrak{A}}_{U_\sigma,i}(\mathfrak{m}^{\mathfrak{A}} \circ \varphi'_{\sigma,i}) \end{aligned}$$

where $\varphi'_{\sigma,i} : E_{U_\sigma} \times I \to T_\Gamma$ is the composition of the substitutions $\varphi_{\sigma,i}$ and ψ, defined, for all $(e,j) \in E_{U_\sigma} \times I$, by

$$\varphi'_{\sigma,i}(e,j) = \begin{cases} \mu(j,t_\ell) & \text{if } e = var_\sigma(\ell) \text{ for some } 1 \leq \ell \leq k \\ \kappa_{e,j} & \text{otherwise.} \end{cases}$$

Now we can apply the other induction hypothesis to find that $\mathfrak{m}^{\mathfrak{A}}(\mu(j,t_\ell)) = f(val(t_\ell),j) = f(H(var_\sigma(\ell)),j)$, so for $e \in lead(U_\sigma)$, $\mathfrak{m}^{\mathfrak{A}}(\mu(j,t_\ell)) = f(H(e),j) = \gamma(e,j)$, where ℓ is such that $e = var_\sigma(\ell)$. For $e \notin lead(U)$, $f(e^\bullet,j)$ is represented by an expression $\kappa_{e,j}$, such that $\mathfrak{m}^{\mathfrak{A}}(\kappa_{e,j}) = f(e^\bullet,j) = f(H(e),j) = \gamma(e,j)$. Thus for all $(e,j) \in E_{U_\sigma} \times I$, $\mathfrak{m}^{\mathfrak{A}}(\varphi'_{\sigma,i}(e,j)) = \gamma(e,j)$, thus $\mathfrak{m}^{\mathfrak{A}} \circ \varphi'_{\sigma,i} = \gamma$. Now continuing the above equations we find that

$$\begin{aligned} \mathfrak{m}^{\mathfrak{A}}(\mu(i,t)) &= s^{\mathfrak{A}}_{U_\sigma,i}(\mathfrak{m}^{\mathfrak{A}} \circ \varphi'_{\sigma,i}) \\ &= s^{\mathfrak{A}}_{U_\sigma,i}(\gamma) \\ &= f'_{U_\sigma,i}(\gamma) \\ &= f(H,i). \end{aligned}$$

This concludes the induction step. Thus for every $i \in I$ and every $t \in WT_\Sigma$, $\mathfrak{m}^{\mathfrak{A}}(\mu(i,t)) = f(val(t),i)$.

So clearly, for every finite $\Sigma \subseteq \Delta$, we can construct a total deterministic finite tree transducer μ, with initial state i_0, such that for all $t \in WT_\Sigma$, $\mathfrak{m}^{\mathfrak{A}}(\mu(i_0,t)) = f(-,i_0)(val(t)) = f_0(val(t))$. Thus f_0 is $(\Delta,\mathfrak{A})$-realizable. □

The reverse of the theorem does not seem to be true without some restrictions on the finite tree transducers in the definition of realizable functions.

Since the tree transducers act on expressions that evaluate to hypergraphs, instead of on hypergraphs themselves, it is not guaranteed for all tree transducers $\mu = (Q,\Sigma,\Gamma',R,\{q_d\})$ and for all states $q \in Q \setminus \{q_d\}$, that if two expressions t' and t'' evaluate to the same hypergraph ($val(t') = val(t'')$), they have the same image under μ for state q (it is possible that $\mu(q,t') \neq \mu(q,t'')$). Thus, two equal hypergraphs could yield different function values, which is not allowed for compatible functions. To obtain a result for the reverse of Theorem 6.8, a more restrictive definition of realizable functions can be used.

Theorem 6.4' (strict realizable functions) Let $\mathfrak{A} = (\mathbf{V},\alpha)$ be a Γ-algebra and $\Delta \subseteq \mathbf{SUB}$. A function $f_0 : \mathbf{HG} \to \mathbf{V}$ is called *strict* $(\Delta,\mathfrak{A})$*-realizable*, if for every finite $\Sigma \subseteq \Delta$, a total deterministic top-down finite tree transducer $\mu = (Q,\Sigma,\Gamma,R,\{q_d\})$ can effectively be constructed, such that for all $t \in WT_\Sigma$ it holds that $\mathfrak{m}^{\mathfrak{A}}(\mu(q_d,t)) = f_0(val(t))$ and for all $t',t'' \in WT_\Sigma$ and all $q \in Q$, if $val(t') = val(t'')$, then $\mathfrak{m}^{\mathfrak{A}}(\mu(q,t')) = \mathfrak{m}^{\mathfrak{A}}(\mu(q,t''))$. □

Note that the finite tree transducers that are constructed in the proof of Theorem 6.8 all satisfy the restrictions in Definition 6.4'. This can be seen from the first induction hypothesis, that states that for every $i \in I$ and every $t \in WT_\Sigma$, $\mathfrak{m}^{\mathfrak{A}}(\mu(i,t)) = f(val(t), i)$. Hence, Theorem 6.8 can be strengthened as follows.

Theorem 6.8' Let $\Delta \subseteq \mathbf{SUB}$, $\mathfrak{A} = (\mathbf{V}, \alpha)$ a Γ-algebra and $f_0 : \mathbf{HG} \to \mathbf{V}$ a $(\mathbf{HRG}^{\bowtie}_{\Delta}, \mathfrak{A})$-compatible function. Then f_0 is strict $(\Delta, \mathfrak{A})$-realizable. □

Clearly, every strict realizable function is also a realizable function.

6.4 Realizable predicates

Besides functions on hypergraphs, Habel also considered predicates on hypergraphs. Predicates on hypergraphs are for example the questions whether or not a hypergraph is totally disconnected, whether or not a hypergraph contains a Eulerian path or whether or not a hypergraph is k-colorable for a fixed $k \in \mathbb{N}$. To be able to compute these predicates, and to answer these questions, based on the derivation of a hypergraph in a hyperedge replacement grammar, the predicates must be compatible with the derivation proces. The notion of compatible predicates was introduced in [HabKreVog89][2]. The definition of compatible predicates in [Hab92] does not have the flaws that the definition of compatible functions has. We will therefore use Definition VI.6.1 from this source, which is included hereafter.

Definition 6.9 (compatible predicates, Habel)

i. Let $C \subseteq \mathbf{HRG}$, I a finite set, called the index set, ψ a predicate defined on pairs $(H, i) \in \mathbf{HG} \times I$ and ψ' a decidable predicate on triples (R, ass, i), with $R \in \mathbf{HG}$, a mapping $ass : E_R \to I$ and $i \in I$. Then ψ is called (C, ψ')-*compatible* if for all $G = (N, T, P, Z) \in C$ and all derivations $A^\bullet \Rightarrow_G R \Rightarrow^*_G H$ with $A \in N \cup T$ and $H \in \mathbf{HG}_T$, and for all $i \in I$, $\psi(H, i)$ holds iff there is a mapping $ass : E_R \to I$ such that $\psi'(R, ass, i)$ holds and $\psi(H(e), ass(e))$ holds for all $e \in E_R$.

ii. A predicate ψ_0 on $\mathbf{HG}$ is called *C-compatible* if predicates ψ and ψ' and an index $i_0 \in I$ exist such that ψ is (C, ψ')-compatible and $\psi_0 = \psi(-, i_0)$. □

Now a predicate on hypergraphs is nothing more than a function from the set of hypergraphs into the set of boolean constants, $\mathbb{B}$. In fact it turns out that every compatible predicate can be represented by a compatible function. In order to show this, we first need an algebra which represents the boolean calculus.

Definition 6.10 (boolean calculus) The set of *booleans*, denoted by $\mathbb{B}$, is defined as $\mathbb{B} = \{\mathit{false}, \mathit{true}\}$. The set of *boolean symbols*, denoted by $\mathbf{\Gamma}^{\mathbb{B}}$, is the ranked alphabet defined by $\mathbf{\Gamma}^{\mathbb{B}}_0 = \{\mathit{false}, \mathit{true}\}$, $\mathbf{\Gamma}^{\mathbb{B}}_1 = \{\neg\}$ and $\mathbf{\Gamma}^{\mathbb{B}}_k = \{\vee, \wedge\}$ for $k \geq 1$. The *boolean calculus* is the $\mathbf{\Gamma}^{\mathbb{B}}$-algebra $\mathfrak{B} = (\mathbb{B}, \alpha)$, where the operations are defined by

[2] In contradiction to what this thesis suggests, compatible predicates were introduced prior to compatible functions.

i. $\alpha_0(\mathit{false}) = \mathit{false}$ and $\alpha_0(\mathit{true}) = \mathit{true}$,

ii. $\alpha_1(\neg) : \mathbb{B} \to \mathbb{B}$ is defined by $\alpha_1(\neg)(p) = \neg p$ for all $p \in \mathbb{B}$,

iii. $\alpha_1(\vee) : \mathbb{B} \to \mathbb{B}$ and $\alpha_1(\wedge) : \mathbb{B} \to \mathbb{B}$ are respectively defined by $\alpha_1(\vee)(p) = p$ and $\alpha_1(\wedge)(p) = p$ for all $p \in \mathbb{B}$,

iv. for $k \geq 2$, $\alpha_k(\vee) : \mathbb{B}^k \to \mathbb{B}$ and $\alpha_k(\wedge) : \mathbb{B}^k \to \mathbb{B}$ are respectively defined by

$$\alpha_k(\vee)(p_1, \ldots, p_k) = \textstyle\bigvee_{1 \leq i \leq k} p_i$$

and

$$\alpha_k(\wedge)(p_1, \ldots, p_k) = \textstyle\bigwedge_{1 \leq i \leq k} p_i$$

for all $p_1, \ldots, p_k \in \mathbb{B}$. □

Using this algebra, we arrive at the following theorem, which is basically the same as Theorem VII.2.4 in [Hab92], but now we do not have to represent predicates as functions into $\mathbb{N}$, using 0 to represent *false*, 1 to represent *true*, *max* to represent the logical or ($\vee$) and $\prod$ to represent the logical and ($\wedge$), since we can use $\mathfrak{B}$ for a more elegant representation.

Theorem 6.11 Let $C \subseteq \mathbf{HRG}$ and ψ_0 be a C-compatible predicate. Then the function $\varphi_0 : \mathbf{HG} \to \mathbb{B}$, given by $\varphi_0(H) = \mathit{true}$ iff $\psi_0(H)$, is $(C, \mathfrak{B})$-compatible, where $\mathfrak{B}$ is the boolean calculus.

Proof The proof is very similar to the proof of Theorem VII.2.4 in [Hab92]. Let ψ_0 be a C-compatible predicate, ψ and ψ' the corresponding predicates, I the corresponding index set and $i_0 \in I$ the index such that $\psi_0 = \psi(-, i_0)$. Then define the function $\varphi : \mathbf{HG} \times I \to \mathbb{B}$ as

$$\varphi(H, i) = \begin{cases} \mathit{true} & \text{if } \psi(H, i) \text{ holds} \\ \mathit{false} & \text{otherwise,} \end{cases}$$

and the family of functions φ' as

$$\varphi' = \{\varphi'_{R,i} : (E_R \times I \to \mathbb{B}) \to \mathbb{B}\}_{R \in \mathbf{HG}, i \in I}$$

where for each $R \in \mathbf{HG}$ and $i \in I$, $\varphi'_{R,i}$ is defined by

$$\varphi'_{R,i}(\gamma) = \begin{cases} \mathit{true} & \text{if } \psi'(R, \mathit{ass}, i) \text{ holds for some } \mathit{ass} : E_R \to I \\ & \quad \text{with } \gamma(e, \mathit{ass}(e)) = \mathit{true} \text{ for all } e \in E_R \\ \mathit{false} & \text{otherwise.} \end{cases}$$

Clearly, $\varphi_0(H) = \varphi(H, i_0)$. Now we have to show that φ is (C, φ')-compatible. Let $H \in \mathbf{HG}_T$ and $A^\bullet \Rightarrow_G R \Rightarrow_G^* H$ be a derivation of H in some $G = (N, T, P, Z) \in$

C. Furthermore, for all $e \in E_R$, let $l_R(e)^\bullet \Rightarrow_G H(e)$ be the fibre of $R \Rightarrow_G^* H$, induced by e.

If $\varphi(H, i) = \mathit{true}$, then $\psi(H, i)$ is satisfied. Since ψ is (C, ψ')-compatible, there is a mapping $\mathit{ass} : E_R \to I$ such that $\psi'(R, \mathit{ass}, i)$ and $\psi(H(e), \mathit{ass}(e))$ for all $e \in E_R$ hold. Thus, $\varphi(H(e), \mathit{ass}(e)) = \mathit{true}$ for all $e \in E_R$. Now define $\gamma : E_R \times I \to \mathbb{B}$ by $\gamma(e, j) = \varphi(H(e), j)$ for $(e, j) \in E_R \times I$. Then $\varphi'_{R,i}(\gamma) = \mathit{true}$.

Conversely, if $\varphi'_{R,i}(\gamma) = \mathit{true}$, with $\gamma(e, j) = \varphi(H(e), j)$ for $(e, j) \in E_R \times I$, then there is a mapping $\mathit{ass} : E_R \to I$ such that for all $e \in E_R$, $\psi'(R, \mathit{ass}, i)$ holds and $\varphi(H(e), \mathit{ass}(e)) = \mathit{true}$. By definition of φ, $\psi(H(e), \mathit{ass}(e))$ holds for all $e \in E_R$. Now since ψ is (C, ψ')-compatible, we find that $\psi(H, i)$ holds. Thus $\varphi(H, i) = \mathit{true}$.

Now $\varphi(H, i) = \varphi'_{R,i}(\gamma)$, where $\gamma(e, j) = \varphi(H(e), j)$ for $(e, j) \in E_R \times I$, thus φ is (C, φ')-compatible. Moreover, each $\varphi'_{R,i}(\gamma)$ can be expressed as $\varphi'_{R,i}(\gamma) = s^{\mathfrak{B}}_{R,i}(\gamma)$, where $s_{R,i} \in T_{\mathbf{\Gamma}^{\mathbb{B}}}(E_R \times I)$ is a $\mathbf{\Gamma}^{\mathbb{B}}(E_R \times I)$-expression corresponding to R and i. The expression $s_{R,i}$ can be expressed as

$$s_{R,i} = \bigvee_{\mathit{ass} \in \mathrm{ASS}_{R,i}} [\![\bigwedge_{e \in E_R} [\![(e, \mathit{ass}(e))]\!]]\!],$$

where

$$\mathrm{ASS}_{R,i} = \{\mathit{ass} : E_R \to I \mid \psi'(R, \mathit{ass}, i) \text{ is satisfied}\}.$$

If $\mathrm{ASS}_{R,i} = \varnothing$ then $s_{R,i} = \mathit{false}$. Now $\mathrm{ASS}_{R,i}$ can be effectively constructed since E_R and I are finite and ψ' is a decidable predicate, and thus $s_{R,i}$ can be effectively constructed from R and i. Hence, the function φ is $(C, \mathfrak{B})$-compatible. □

If a predicate ψ_0 on hypergraphs is defined to be equal to a function $\psi_0 : \mathbf{HG} \to \mathbb{B}$, then we come to the following corollary.

Corollary 6.12 Let $\Delta \subseteq \mathbf{SUB}$ and ψ_0 a $\mathbf{HRG}^{\bowtie}_{\Delta}$-compatible predicate. Then the function $\varphi_0 : \mathbf{HG} \to \mathbb{B}$, defined by $\varphi_0(H) = \mathit{true}$ iff $\psi_0(H)$ holds, is $(\Delta, \mathfrak{B})$-realizable.

Proof The corollary follows directly from Theorem 6.11 and Theorem 6.8. □

Chapter seven

Decidability results

The final chapter of this thesis presents some decidability results on hypergraph expression languages, which are defined as regular tree languages. It is shown that it is decidable whether or not a predicate on a hypergraph, that is realizable by a finite tree transducer that transforms a hypergraph expression into a boolean expression, holds for the evaluation of some, or all, expressions in a regular tree language over **SUB**. Also it is shown that it is decidable whether or not the value of a numerical function on hypergraphs, that is realizable by a finite tree transducer that transforms a hypergraph expression into a numerical expression, is bounded. The metatheorems in [Hab92] follow from the theorems in this chapter. The proofs of these results use results from tree language theory.

7.1 Decision problems

In this section it is shown that it is decidable whether or not a regular hypergraph expression language generates hypergraph expressions that evaluate to a hypergraph, for which a realizable predicate holds. The proof uses the tree language of all boolean expressions that evaluate to *true*, which is a recognizable tree language. Recall the definition of the boolean calculus in Definition 6.10.

Definition 7.1 Let $\mathfrak{B} = (\mathbb{B}, \alpha)$ be the $\mathbf{\Gamma}^{\mathbb{B}}$-algebra for the boolean calculus and $\mathbf{\Gamma}$ a finite subset of $\mathbf{\Gamma}^{\mathbb{B}}$. Then the set of expressions over $\mathbf{\Gamma}$, which evaluate to *true*, denoted by $\mathbf{TRUE}_{\mathbf{\Gamma}}$, is defined as $\mathbf{TRUE}_{\mathbf{\Gamma}} = \{t \in T_{\mathbf{\Gamma}} \mid \mathfrak{m}^{\mathfrak{B}}(t) = \mathit{true}\}$. □

Lemma 7.2 Let $\mathfrak{B} = (\mathbb{B}, \alpha)$ be the $\mathbf{\Gamma}^{\mathbb{B}}$-algebra for the boolean calculus and $\mathbf{\Gamma}$ a finite subset of $\mathbf{\Gamma}^{\mathbb{B}}$. Then $\mathbf{TRUE}_{\mathbf{\Gamma}} \in \mathbf{RECOG}$.

Proof Construct a regular tree grammar $G = (\{F, T\}, \mathbf{\Gamma}, R, T)$, with the set of rules R such that

i. if *false* $\in \mathbf{\Gamma}_0$ then $F \to \mathit{false}$ is in R and if *true* $\in \mathbf{\Gamma}_0$ then $T \to \mathit{true}$ is in R,

ii. if $\neg \in \mathbf{\Gamma}_1$ then $F \to \neg[T]$ and $T \to \neg[F]$ are in R,

iii. for $k \geq 1$ and $\vee, \wedge \in \mathbf{\Gamma}_k$, $F \to \vee[F \cdots F]$ and all $F \to \wedge[A_1 \cdots A_k]$, such that $\exists 1 \leq i \leq k : A_i = F$, are in R and

iv. for $k \geq 1$ and $\wedge, \vee \in \mathbf{\Gamma}_k$, $T \to \wedge[T \cdots T]$ and all $T \to \vee[A_1 \cdots A_k]$, such that $\exists 1 \leq i \leq k : A_i = T$, are in R.

It is easy to see that $\mathcal{L}(G) = \mathbf{TRUE}_{\mathbf{\Gamma}}$. □

We are now ready to prove one of the major theorems of this thesis.

Theorem 7.3 (metatheorem for decision problems) Let $\Delta \subseteq \mathbf{SUB}$ and $\varphi_0 : \mathbf{HG} \to \mathbb{B}$ a $(\Delta, \mathfrak{B})$-realizable function. Then for all $G \in \mathbf{RTG}_\Delta$, it is decidable whether or not $\varphi_0(\mathit{val}(t)) = \mathit{true}$ for some $t \in \mathcal{L}(G)$.

Proof Let $G = (N, \Sigma, P, S) \in \mathbf{RTG}_\Delta$. Clearly Σ is a finite subset of Δ. Thus, since φ_0 is $(\Delta, \mathfrak{B})$-realizable, a total deterministic top-down finite tree transducer $M = (Q, \Sigma, \mathbf{\Gamma}, R, \{q_d\})$ can be constructed, such that for all $t \in WT_\Sigma$, $\mathfrak{m}^{\mathfrak{B}}(M(t)) = \varphi_0(\mathit{val}(t))$. $\mathbf{\Gamma}$ is the finite subset of $\mathbf{\Gamma}^{\mathbb{B}}$, that consists of all symbols with their rank, that are used in the right-hand sides of the rules in R. Now $M^{-1}(\mathbf{TRUE}_{\mathbf{\Gamma}})$ is the set of all $t \in WT_\Sigma$ such that $\mathfrak{m}^{\mathfrak{B}}(M(t)) = \mathit{true}$. Since $\mathbf{TRUE}_{\mathbf{\Gamma}} \in \mathbf{RECOG}$ (Lemma 7.2) and $\mathbf{RECOG}$ is effectively closed under inverse top-down tree transducers (Theorem 3.59), $M^{-1}(\mathbf{TRUE}_{\mathbf{\Gamma}})$ is effectively a recognizable tree language. Now $M^{-1}(\mathbf{TRUE}_{\mathbf{\Gamma}}) \cap \mathcal{L}(G)$ is the set of all $t \in \mathcal{L}(G)$ such that $\mathfrak{m}^{\mathfrak{B}}(M(t)) = \mathit{true}$ (and thus $\varphi_0(\mathit{val}(t)) = \mathit{true}$). Since $\mathbf{RECOG}$ is effectively closed under intersection (Theorem 3.14) and the emptiness problem for recognizable tree languages is decidable (Theorem 3.15), we find that it is decidable whether $\varphi_0(\mathit{val}(t))$ holds for some $t \in \mathcal{L}(G)$, since this is true iff $M^{-1}(\mathbf{TRUE}_{\mathbf{\Gamma}}) \cap \mathcal{L}(G) \neq \varnothing$. □

Remark 7.4 Note that if φ_0 is $(\Delta, \mathfrak{B})$-realizable, then so is $\neg\varphi_0$ (simply "invert" the rules which have the initial state as left-hand side). Therefore it follows from Theorem 7.3, that for every $G \in \mathbf{RTG}_\Delta$, it is decidable whether or not $\varphi_0(val(t))$ holds for all $t \in \mathcal{L}(G)$, since this is equal to the problem whether or not there is *no* $t \in \mathcal{L}(G)$ such that $\neg\varphi_0(val(t))$. □

The metatheorem for decision problems in [Hab92] (Theorem VI.4.1) follows from this theorem, the equivalence between the two formalisms for the generation of hypergraphs and the relation between compatible and realizable functions.

Theorem 7.5 (metatheorem for decision problems, Habel) Let $\Delta \subseteq \mathbf{SUB}$ and ψ_0 be a $\mathbf{HRG}_\Delta^{\bowtie}$-compatible predicate. Then for all $G_h \in \mathbf{HRG}_\Delta^{\bowtie}$, it is decidable whether

i. $\psi_0(H)$ holds for some $H \in \mathcal{L}(G_h)$ and

ii. $\psi_0(H)$ holds for all $H \in \mathcal{L}(G_h)$.

Proof According to Corollary 6.12, the function $\varphi_0 : \mathbf{HG} \to \mathbb{B}$, given by $\varphi_0(H) =$ *true* iff $\psi_0(H)$, is $(\Delta, \mathfrak{B})$-realizable, where $\mathfrak{B}$ is the $\mathbf{\Gamma}^{\mathbb{B}}$-algebra for the boolean calculus. Let $G_h \in \mathbf{HRG}_\Delta^{\bowtie}$ be an arbitrary hyperedge replacement grammar. According to Theorem 4.20 we may assume that G_h is typed (the construction in the proof of this theorem assures that the typed equivalent of G_h is still in $\mathbf{HRG}_\Delta^{\bowtie}$). Let $G_t = (N, \Sigma, R, S) \in \mathbf{RTG}_{\mathbf{SUB}}$ be a typed regular tree grammar such that $G_h \bowtie G_t$. Such a G_t exists (and can be found effectively) according to Theorem 4.40. From Theorem 4.41 we find that $\mathcal{L}(G_h) = val(\mathcal{L}(G_t))$. Furthermore, from Definition 4.39 and the definition of $\mathbf{HRG}_\Delta^{\bowtie}$ (Definition 6.6), we find that Σ is a finite subset of Δ and thus $G_t \in \mathbf{RTG}_\Delta$. Hence, the statements in the metatheorem follow directly from Theorem 7.3 and Remark 7.4. □

7.2 Boundedness problems

We now will show that it is decidable for a realizable function f_0, whether or not a regular tree grammar, generating hypergraph expressions, generates expressions that evaluate to hypergraphs whose function values under f_0 grow beyond any bound. These results were presented in another form in [Eng94].

We use numerical functions that can be realized by a top-down finite tree transducer, which transforms the hypergraph expressions into expressions in the $\mathbf{\Gamma}^{\mathbb{N}}$-algebra $\mathfrak{N} = (\mathbb{N}, \alpha)$ for the arithmetic caclulus, which was defined in Definition 5.1. For the simplicity of the proofs, we will use trees in which each node has at most two subtrees. These trees are called binary trees, defined as follows.

Definition 7.6 A ranked alphabet Γ is called *binary* if $\Gamma_k = \varnothing$ for $k = 1$ and for all $k > 2$. A tree over a binary ranked alphabet is called a *binary* tree. □

We want to make sure that we do not lose any expressive power, and therefore we prove the following lemma, which states that every expression in $\mathfrak{N}$ can be transformed into an equivalent binary expression (also in $\mathfrak{N}$).

Lemma 7.7 For each finite $\Gamma \subseteq \mathbf{\Gamma}^{\mathbb{N}}$, there is a binary ranked alphabet $\Gamma' \subseteq \mathbf{\Gamma}^{\mathbb{N}}$ and a linear tree homomorphism $\xi : T_\Gamma \to T_{\Gamma'}$, such that for every $t \in T_\Gamma$, $\xi(t) \in T_{\Gamma'}$ and $\mathfrak{m}^{\mathfrak{N}}(\xi(t)) = \mathfrak{m}^{\mathfrak{N}}(t)$.

Proof Let Γ be an arbitrary finite subset of $\mathbf{\Gamma}^{\mathbb{N}}$ and let Γ' be such that $\Gamma'_0 = \Gamma_0$ and $\Gamma'_2 = \{\oplus, \odot, \mathbf{max}, \mathbf{pow}\}$. Clearly, $\Gamma' \subseteq \mathbf{\Gamma}^{\mathbb{N}}$ and Γ' is binary. Now construct a linear tree homomorphism ξ, as follows. $\xi_0 : \Gamma_0 \to T_{\Gamma'}$ is defined by $\xi_0(\sigma) = \sigma$ for all $\sigma \in \Gamma_0$ and for all $k \geq 1$, $\xi_k : \Gamma_k \to T_{\Gamma'}(\mathbf{X}_k)$ is defined by $\xi_k(\sigma) = \sigma[x_1\sigma[x_2 \ldots \sigma[x_{k-1}x_k] \ldots]]$ for all $\sigma \in \Gamma_k$ (in particular, $\xi_1(\sigma) = x_1$). Since the operations corresponding to $\oplus$, $\odot$ and $\mathbf{max}$ in $\mathfrak{N}$ are associative and since $\mathbf{pow}$ has only rank 2, it can be shown by induction to the structure of t, that for every $t \in T_\Gamma$, $\xi(t) \in T_{\Gamma'}$ and $\mathfrak{m}^{\mathfrak{N}}(\xi(t)) = \mathfrak{m}^{\mathfrak{N}}(t)$. □

The value of a function g is bounded on a set S iff the set $g(S)$ is finite. If S has the property that $g(S)$ is finite iff S is finite, we only have to show that S is finite. If S is a regular tree language, we can use results from the theory of tree grammars to show its finiteness. To accomplish that a set of hypergraph expressions has this property, we need to remove all subexpressions that do not contribute to the function value, e.g. subexpressions of the form $0 + t$, $t + 0$, $1 \cdot t$, $t \cdot 1$, t^1 and 1^t. Expressions that do not have these subexpressions are called ascending expressions. Again, each (binary) expression in $\mathfrak{N}$ can be effectively transformed into an equivalent ascending expression.

Definition 7.8 For each $\Gamma \subseteq \mathbf{\Gamma}^{\mathbb{N}}$, the set of *ascending* expressions over Γ, denoted by AT_Γ, is defined recursively as follows.

i. If $\sigma \in \Gamma_0$, then $\sigma \in AT_\Gamma$.

ii. For $\sigma \in \Gamma_k \backslash \{\mathbf{max}\}$, $k \geq 1$ and $t_1, \ldots, t_k \in AT_\Gamma$, if $\mathfrak{m}^{\mathfrak{N}}(t_i) < \mathfrak{m}^{\mathfrak{N}}(\sigma[t_1 \cdots t_k])$ for all $1 \leq i \leq k$, then $\sigma[t_1 \cdots t_k] \in AT_\Gamma$.

iii. If $\mathbf{max} \in \Gamma_k$, $k \geq 1$ and $t_1, \ldots, t_k \in AT_\Gamma$, then $\mathbf{max}[t_1 \cdots t_k] \in AT_\Gamma$.

Lemma 7.9 For every finite binary $\Gamma \subseteq \mathbf{\Gamma}^{\mathbb{N}}$, there is a linear total deterministic bottom-up finite tree transducer $\tau : T_\Gamma \to T_\Gamma$, such that for every $t \in T_\Gamma$, $\tau(t) \in AT_\Gamma$ and $\mathfrak{m}^{\mathfrak{N}}(\tau(t)) = \mathfrak{m}^{\mathfrak{N}}(t)$.

Proof Let $\Gamma \subseteq \mathbf{\Gamma}^{\mathbb{N}}$ be an arbitrary finite binary ranked alphabet. Now construct a total deterministic bottom-up finite tree transducer $\tau = (Q, \Gamma, \Gamma, R, Q)$, where $Q = \{q_0, q_1, q_{\geq 2}\}$ and R has the following rules.

i. $C_0 \to q_0[C_0]$, $C_1 \to q_1[C_1]$ and $C_n \to q_{\geq 2}[C_n]$ for $C_n \in \Gamma$ such that $n \geq 2$,

ii. $\oplus \to q_0[\oplus]$,
$\oplus[q_0[x_1]q_0[x_2]] \to q_0[C_0]$,
$\oplus[q_0[x_1]q[x_2]] \to q[x_2]$ and $\oplus[q[x_1]q_0[x_2]] \to q[x_1]$ for $q \in \{q_1, q_{\geq 2}\}$,
$\oplus[q[x_1]q'[x_2]] \to q_{\geq 2}[\oplus[x_1x_2]]$ for $q, q' \in \{q_1, q_{\geq 2}\}$,

iii. $\odot \to q_1[\odot]$,
$\odot[q_0[x_1]q_0[x_2]] \to q_0[C_0]$,
$\odot[q_0[x_1]q[x_2]] \to q_0[C_0]$ and $\odot[q[x_1]q_0[x_2]] \to q_0[C_0]$ for $q \in \{q_1, q_{\leq 2}\}$,
$\odot[q_1[x_1]q_{\geq 2}[x_2]] \to q_{\geq 2}[x_2]$ and $\odot[q_{\geq 2}[x_1]q_1[x_2]] \to q_{\geq 2}[x_1]$,
$\odot[q_1[x_1]q_1[x_2]] \to q_1[C_1]$ and $\odot[q_{\geq 2}[x_1]q_{\geq 2}[x_2]] \to q_{\geq 2}[\odot[x_1 x_2]]$,

iv. $\mathbf{pow}[q_0[x_1]q_0[x_2]] \to q_1[C_1]$ (note that we define $0^0 = 1$),
$\mathbf{pow}[q_0[x_1]q[x_2]] \to q_0[C_0]$ for $q \in \{q_1, q_{\geq 2}\}$,
$\mathbf{pow}[q_1[x_1]q[x_2]] \to q_1[C_1]$ for $q \in Q$,
$\mathbf{pow}[q_{\geq 2}[x_1]q_0[x_2]] \to q_1[C_1]$, $\mathbf{pow}[q_{\geq 2}[x_1]q_1[x_2]] \to q_{\geq 2}[x_1]$,
$\mathbf{pow}[q_{\geq 2}[x_1]q_{\geq 2}[x_2]] \to q_{\geq 2}[\mathbf{pow}[x_1 x_2]]$,

v. $\mathbf{max}[q_0[x_1]q_0[x_2]] \to q_0[C_0]$,
$\mathbf{max}[q_0[x_1]q[x_2]] \to q[x_2]$ and $\mathbf{max}[q[x_1]q_0[x_2]] \to q[x_1]$ for $q \in \{q_1, q_{\geq 2}\}$,
$\mathbf{max}[q_1[x_1]q_{\geq 2}[x_2]] \to q_{\geq 2}[x_2]$ and $\mathbf{max}[q_{\geq 2}[x_1]q_1[x_2]] \to q_{\geq 2}[x_1]$,
$\mathbf{max}[q_1[x_1]q_1[x_2]] \to q_1[C_1]$ and $\mathbf{max}[q_{\geq 2}[x_1]q_{\geq 2}[x_2]] \to q_{\geq 2}[\mathbf{max}[x_1 x_2]]$.

When the tree transducer arrives in state q_0, the value of the processed subtree is 0, for state q_1 it is 1 and for state $q_{\geq 2}$ it is larger than 1. It can be shown by induction to the structure of t, that for every $t \in T_\Gamma$, $\tau(t) \in AT_\Gamma$ and $\mathfrak{m}^{\mathfrak{N}}(\tau(t)) = \mathfrak{m}^{\mathfrak{N}}(t)$. □

As was the case with subexpressions of the form $0 + t$ et cetera, we also have to remove expressions of the form $\mathbf{max}[t_1 t_2]$, since they can lead to infinitely many trees with the same value. We therefore show that a tree transducer can be constructed that removes **max** from the expressions by guessing its value (and thus the appropriate subtree).

Lemma 7.10 For each finite $\Gamma \subseteq \mathbf{\Gamma}^{\mathbb{N}}$, a linear top-down finite tree transducer $\eta \subseteq T_\Gamma \times T_\Gamma$ can be constructed, such that $\eta(AT_\Gamma) \subseteq AT_{\Gamma \setminus \{\mathbf{max}\}}$ and, for every $L \subseteq AT_\Gamma$, $\mathfrak{m}^{\mathfrak{N}}(L)$ is finite iff $\mathfrak{m}^{\mathfrak{N}}(\eta(L))$ is finite.

Proof Let $\Gamma \subseteq \mathbf{\Gamma}^{\mathbb{N}}$ be an arbitrary finite ranked alphabet. Construct the (non-deterministic) tree transducer $\eta = (\{q\}, \Gamma, \Gamma, R, \{q\})$, where q is the only state and R contains the following rules.

i. For $\sigma \in \Gamma_0$, the rule $q[\sigma] \to \sigma$.

ii. For $k \geq 1$, $\mathbf{max} \in \Gamma_k$, the rule(s) $q[\mathbf{max}[x_1 \cdots x_k]] \to x_i$, for $1 \leq i \leq k$.

iii. For $k \geq 1$, $\sigma \in \Gamma_k \setminus \{\mathbf{max}\}$, the rule $q[\sigma[x_1 \cdots x_k]] \to \sigma[q[x_1] \cdots q[x_k]]$.

Note that the non-determinism is caused only by the multiple rules for **max**. For every expression of the form $\sigma[t_1 \cdots t_k]$, with $\sigma \in \Gamma \setminus \{\mathbf{max}\}$, η merely copies the expression deterministically to the output, computing the value for $\mathfrak{m}^{\mathfrak{N}}(\sigma[t_1 \cdots t_k])$ from the operation corresponding to σ and the values $\mathfrak{m}^{\mathfrak{N}}(t_i)$ for $1 \leq i \leq k$. For expressions of the form $\mathbf{max}[t_1 \cdots t_k]$, η guesses the subtree which has the highest value for $\mathfrak{m}^{\mathfrak{N}}(t_i)$, $1 \leq i \leq k$. Among these guesses, there is at least one good guess for all **max** expressions in a tree. Thus, if $t \in AT_\Gamma$ is an arbitrary ascending expression and $\eta(t) = \{s \in AT_{\Gamma\backslash\{\mathbf{max}\}} \mid q[t] \Rightarrow^* s\}$ is the image of t under η, then $\mathfrak{m}^{\mathfrak{N}}(t) = max\{\mathfrak{m}^{\mathfrak{N}}(s) \mid s \in \eta(t)\}$. This uses the fact that the operations corresponding to $\oplus$, $\odot$ and **pow** are monotonic in all arguments. Thus, for all $L \subseteq AT_\Gamma$, $\mathfrak{m}^{\mathfrak{N}}(\eta(L))$ is finite iff $\mathfrak{m}^{\mathfrak{N}}(L)$ is finite. Furthermore, by induction to the structure of t it can be shown that for every $t \in AT_\Gamma$, $\eta(t) \in AT_{\Gamma\backslash\{\mathbf{max}\}}$. □

We now have a set of ascending expressions that do not contain the **max** symbol. We now find that for this set, the value of the meaning function $\mathfrak{m}^{\mathfrak{N}}$ is bounded iff the set itself is finite.

Lemma 7.11 For each $L \subseteq AT_{\mathbf{\Gamma}^{\mathbb{N}}\backslash\{\mathbf{max}\}}$, $\mathfrak{m}^{\mathfrak{N}}(L)$ is finite iff L is finite.

Proof For each $t = \sigma[t_1 \cdots t_k] \in AT_{\mathbf{\Gamma}^{\mathbb{N}}\backslash\{\mathbf{max}\}}$, $k \geq 1$, $\sigma \in \{\oplus, \odot, \mathbf{pow}\}$ and $t_1, \ldots, t_k \in AT_{\mathbf{\Gamma}^{\mathbb{N}}\backslash\{\mathbf{max}\}}$, by definition $\mathfrak{m}^{\mathfrak{N}}(t_i) < \mathfrak{m}^{\mathfrak{N}}(\sigma[t_1 \cdots t_k])$ for $1 \leq i \leq k$. By induction to the structure of t, one can show that $\mathfrak{m}^{\mathfrak{N}}(t) \geq height(t)$ for all $t \in AT_{\mathbf{\Gamma}^{\mathbb{N}}\backslash\{\mathbf{max}\}}$. Thus, if L is infinite, there must be trees of arbitrary height, and thus of arbitrary large value. □

Now we are finally ready to prove one of the most important results in this thesis. It states that we can decide for a realizable function, given a regular tree grammar, whether or not the function value can grow beyond any bound for a hypergraph, that is the result of the evaluation of a hypergraph expression generated by the grammar.

Theorem 7.12 (boundedness theorem) Let $\Delta \subseteq \mathbf{SUB}$ and $f_0 : \mathbf{HG} \to \mathbb{N}$ a $(\Delta, \mathfrak{N})$-realizable function. Then it is decidable for a regular tree grammar G over Δ, whether or not there is a $v \in \mathbb{N}$, such that for all $t \in \mathcal{L}(G)$, $f_0(val(t)) \leq v$.

Proof The proof is similar to the proof of Proposition 15 in [Eng94]. Let $G = (N, \Sigma, R, S)$ be an arbitrary regular tree grammar over Δ. We have to show that it is decidable for G, whether or not $f_0(val(\mathcal{L}(G)))$ is bounded or, equivalently, whether or not $f_0(val(\mathcal{L}(G)))$ is finite. Clearly, Σ is a finite subset of Δ, so there exists a total deterministic top-down tree transducer $\mu : T_\Sigma \to T_\Gamma$, where $\Gamma \subseteq \mathbf{\Gamma}^{\mathbb{N}}$ is the finite set of all operators used in μ, such that for all $t \in WT_\Sigma$, $\mathfrak{m}^{\mathfrak{N}}(\mu(t)) = f_0(val(t))$. Thus $\mathfrak{m}^{\mathfrak{N}}(\mu(\mathcal{L}(G))) = f_0(val(\mathcal{L}(G)))$. Now, by Lemma 7.7, a binary ranked alphabet $\Gamma' \subseteq \mathbf{\Gamma}^{\mathbb{N}}$ and a linear tree homomorphism $\xi : T_\Gamma \to T_{\Gamma'}$ can be constructed, such that for every $t \in T_\Gamma$, $\xi(t) \in T_{\Gamma'}$ and $\mathfrak{m}^{\mathfrak{N}}(\xi(t)) = \mathfrak{m}^{\mathfrak{N}}(t)$. Thus $\mathfrak{m}^{\mathfrak{N}}(\xi(\mu(\mathcal{L}(G)))) = f_0(val(\mathcal{L}(G)))$ and $\xi(\mu(\mathcal{L}(G))) \subseteq T_{\Gamma'}$. Hence, $\xi(\mu(\mathcal{L}(G)))$ contains only binary trees. By Lemma 7.9, there is a total deterministic bottom-up finite tree transducer $\tau : T_{\Gamma'} \to T_{\Gamma'}$, such that for all $t \in T_{\Gamma'}$, $\tau(t) \in AT_{\Gamma'}$ and $\mathfrak{m}^{\mathfrak{N}}(\tau(t)) = \mathfrak{m}^{\mathfrak{N}}(t)$. Hence, $\tau(\xi(\mu(\mathcal{L}(G)))) \subseteq AT_{\Gamma'}$ and $\mathfrak{m}^{\mathfrak{N}}(\tau(\xi(\mu(\mathcal{L}(G))))) = \mathfrak{m}^{\mathfrak{N}}(\xi(\mu(\mathcal{L}(G)))) = f_0(val(\mathcal{L}(G)))$. Now by Lemma 7.10, we can construct a non-deterministic top-down finite tree transducer $\eta \subseteq T_{\Gamma'} \times T_{\Gamma'}$, such that for every $L \subseteq AT_{\Gamma'}$, $\eta(L) \subseteq AT_{\Gamma'\backslash\{\mathbf{max}\}}$ and $\mathfrak{m}^{\mathfrak{N}}(L)$ is finite iff $\mathfrak{m}^{\mathfrak{N}}(\eta(L))$ is finite. So $\eta(\tau(\xi(\mu(\mathcal{L}(G))))) \subseteq AT_{\Gamma'\backslash\{\mathbf{max}\}}$ and $f_0(val(\mathcal{L}(G)))$ is finite iff $\mathfrak{m}^{\mathfrak{N}}(\eta(\tau(\xi(\mu(\mathcal{L}(G))))))$ is finite. By Lemma 7.11, $f_0(val(\mathcal{L}(G)))$ is finite iff $\eta(\tau(\xi(\mu(\mathcal{L}(G)))))$ is finite. Since $\mathcal{L}(G) \in \mathbf{RECOG}$ and since $\eta \circ \tau \circ \xi \circ \mu \in (\mathbf{B} \cup \mathbf{T})^*$, it follows from the definition of **SUR** that $\eta(\tau(\xi(\mu(\mathcal{L}(G))))) \in \mathbf{SUR}$. Now, by

Theorem 3.62, the finiteness problem is solvable for **SUR**. Hence, the finiteness problem is solvable for $f_0(val(\mathcal{L}(G)))$. □

Remark 7.13 Note that if μ in the proof of Theorem 7.12 is linear, then so is $\eta \circ \tau \circ \xi \circ \mu$, and thus $\eta(\tau(\xi(\mu(\mathcal{L}(G))))) \in \mathbf{RECOG}$, since **RECOG** is closed under linear top-down or bottom-up tree transducers. Now the finiteness problem for **RECOG** is decidable by Theorem 3.16, so if μ is linear, the proof can be simplified. □

This result is related to the metatheorem for boundedness problems in [Hab92] (Theorem VII.3.1). In fact, Habels result follows from Theorem 7.12 for certain classes of hyperedge replacement grammars.

Theorem 7.14 (metatheorem for boundedness problems) Let $\Delta \subseteq \mathbf{SUB}$ and $f_0 : \mathbf{HG} \to \mathbb{N}$ be a $(\mathbf{HRG}^{\bowtie}_{\Delta}, \mathfrak{N})$-compatible function. Then for all $G_h \in \mathbf{HRG}^{\bowtie}_{\Delta}$, it is decidable whether or not there is a $v \in \mathbb{N}$, such that $f_0(H) \leq v$ for all $H \in \mathcal{L}(G)$.

Proof The proof is similar to the proof of Theorem 7.5. According to Theorem 6.8, f_0 is $(\Delta, \mathfrak{N})$-realizable. Now let G_h be an arbitrary hyperedge replacement grammar in $\mathbf{HRG}^{\bowtie}_{\Delta}$. According to Theorem 4.20 we may assume that G_h is typed (the construction in the proof of this theorem assures that the typed equivalent of G_h is still in $\mathbf{HRG}^{\bowtie}_{\Delta}$). Let $G_t = (N, \Sigma, R, S) \in \mathbf{RTG}_{\mathbf{SUB}}$ be a typed regular tree grammar such that $G_h \bowtie G_t$. Such a G_t exists (and can be found effectively) according to Theorem 4.40. From Theorem 4.41 we find that $\mathcal{L}(G_h) = val(\mathcal{L}(G_t))$. Furthermore, from Definition 4.39 and the definition of $\mathbf{HRG}^{\bowtie}_{\Delta}$ (Definition 6.6), we find that Σ is a finite subset of Δ and thus $G_t \in \mathbf{RTG}_{\Delta}$. Hence, the statement in the metatheorem follows directly from Theorem 7.12. □

Corollary 7.15 (metatheorem for boundedness problems, Habel) Let $\Delta \subseteq \mathbf{SUB}$ and f_0 be a $(\mathbf{HRG}^{\bowtie}_{\Delta}, max, +, \cdot)$-compatible function. Then, for all $G \in \mathbf{HRG}^{\bowtie}_{\Delta}$, it is decidable whether or not there is a natural number $v \in \mathbb{N}$, such that $f_0(H) \leq v$ for all $H \in \mathcal{L}(G)$.

Proof Following our own, equivalent, definition, f_0 is a $(\mathbf{HRG}^{\bowtie}_{\Delta}, \mathfrak{N})$-compatible function. Therefore the corollary follows directly from Theorem 7.14. □

Example 7.16 Let $f_0 : \mathbf{HG} \to \mathbb{N}$ be the function defined by $f_0(H) = |E_H|$. Thus f_0 determines the number of hyperedges in a hypergraph. From Section 5.1 we know that f_0 is $(\mathbf{SUB}, \mathfrak{N})$-realizable. So from Theorem 7.12 we find that it is decidable for a regular tree grammar G over **SUB**, whether or not there is a $v \in \mathbb{N}$, such that for all $t \in \mathcal{L}(G)$, $f_0(val(t)) \leq v$.

For the regular tree grammar G_t from Example 4.46, we find that $\mathcal{L}(G_t) = \{\sigma_0([\sigma_1)^n[\sigma_2]()]^n \mid n \in \mathbb{N}\}$. Now for $n \in \mathbb{N}$, $val(\sigma_0([\sigma_1)^n[\sigma_2]()]^n) = W_{n+2}$. Furthermore, for all $n \in \mathbb{N}$, $f_0(W_n) = 2n$. So clearly, for G_t, $f_0(val(G_t))$ is not bounded since $val(G_t)$ contains infinitely many hypergraphs. □

seventyfour

References

[BauCou87] M. Bauderon, B. Courcelle, *Graph Expressions and Graph Rewritings*, Mathematical Systems Theory, Vol. 20, 1987, pp. 83–127.

[Eng74] Joost Engelfriet, *Tree Automata and Tree Grammars*, Lecture Notes, Institute of Mathematics, University of Aarhus, Denmark, 1974.

[Eng75] Joost Engelfriet, *Bottom-up and Top-down Tree Transformations - a Comparison*, Mathematical Systems Theory, Vol. 9, 1975, pp. 198–231.

[Eng77] Joost Engelfriet, *Top-down Tree Transducers with Regular Look-ahead*, Mathematical Systems Theory, Vol. 10, 1977, pp. 289–303.

[Eng94] Joost Engelfriet, *Graph Grammars and Tree Transducers*, proceedings CAAP'94, Sophie Tison (ed.), Lecture Notes in Computer Science, Vol. 787, Springer Verlag, Berlin, Germany, pp. 15–36.

[EngHey91] Joost Engelfriet, Linda Heyker, *The String Generating Power of Context Free Hypergraph Grammars*, Journal of Computer and System Sciences, Vol. 43, October 1991, pp. 328–360.

[GecSte84] Ferenc Gécseg, Magnus Steinby, *Tree Automata*, Akadémiai Kiadó, Budapest, Hungary, 1984.

[Hab92] Annegret Habel, *Hyperedge Replacement: Grammars and Languages*, Lecture Notes in Computer Science, Vol. 643, Springer Verlag, Berlin, Germany, July 1992.

[HabKre87] Annegret Habel, H. J. Kreowski, *May We Introduce to You: Hyperedge Replacement*, in: Graph Grammars and Their Applications to Computer Science, Lecture Notes in Computer Science 291, 1987, pp. 15–26.

[HabKreVog89] Annegret Habel, H. J. Kreowski, W. Vogler, *Metatheorems for Decision Problems on Hyperedge Replacement Graph Languages*, Acta Informatica, Vol. 26, 1989, pp. 657–677.

[HabKreVog91] Annegret Habel, H. J. Kreowski, W. Vogler, *Decidable Boundedness Problems for Sets of Graphs Generated by Hyperedge Replacement*, Theoretical Computer Science 89, 1991, pp. 33–62.

[HopUll79] John E. Hopcroft, Jeffrey D. Ullman, *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley Publishing Company, Reading, Massachusetts, US, 1979.

[Wec92] Wolfgang Wechler, *Universal Algebra for Computer Scientists*, Springer-Verlag, Berlin Heidelberg, Germany, 1992.

seventysix