ELSEVIER

Theoretical Computer Science 255 (2001) 483–509

Theoretical Computer Science

www.elsevier.com/locate/tcs

Codes and equations on trees ☆

Sabrina Mantaci * , Antonio Restivo

Dipartimento di Matematica ed Applicazioni, Università di Palermo, via Archirafi, 34, 90123 Palermo, Italy

Received September 1998; revised May 1999
Communicated by M. Nivat

Abstract

The objective of this paper is to study, by new formal methods, the notion of tree code introduced by Nivat in (Tree Automata and Languages, Elsevier, Amsterdam, 1992, pp. 1–19). In particular, we consider the notion of stability for sets of trees closed under concatenation. This allows us to give a characterization of tree codes which is very close to the algebraic characterization of word codes in terms of free monoids. We further define the stable hull of a set of trees and derive a defect theorem for trees, which generalizes the analogous result for words. As a consequence, we obtain some properties of tree codes having two elements. Moreover, we propose a new algorithm to test whether a finite set of trees is a tree code. The running time of the algorithm is polynomial in the size of the input. We also introduce the notion of tree equation as complementary view to tree codes. The main problem emerging in this approach is to decide the satisfiability of tree equations: it is a special case of second-order unification, and it remains still open.

1. Introduction

The theory of (word) codes, born in the context of information theory, has been later developed, as an independent subject, using both combinatorial and algebraic methods. This theory is now a part of theoretical computer science and is strongly related to combinatorics on words, automata theory and formal languages (cf. [1], but also [3, 5, 17]).

The objective of the theory of codes, from an elementary point of view, is the study of the properties concerning factorizations of words into a sequence of words taken from a given set. Actually, a set X of words is a code if any word over the same alphabet admits at most one factorization in elements of X. This notion can be extended

☆ Work partially supported by Progetto Cofinanziato MURST "Modelli di Calcolo Innovativi: Metodi sintattici e Combinatori".

* Corresponding author.
E-mail addresses: sabrina@altair.math.unipa.it (S. Mantaci), restivo@altair.math.unipa.it (A. Restivo).

0304-3975/01/$ - see front matter
PII: S0304-3975(99)00317-5

in a natural way to structures more general than words. For instance, in [4, 12, 24], the authors consider the code problem for traces and prove that, in some specific cases, it is possible to decide whether a finite set of traces is a code.

In this paper we approach the code problem for k-ary labeled trees. A k-ary labeled tree is here considered as a generalization of a word, in the sense that words correspond to the particular case of $k = 1$. The content of the present paper is a part of a general research program, aiming to extend concepts, methods and results of combinatorics on words to trees (cf. [8–10]). The notion of tree code was introduced by Nivat [25] where he proved, in particular, that it is possible to decide whether a finite set of trees is a tree code or not. Since this pioneering paper, no considerable progress has been made on the subject.

In this paper we first develop a mathematical framework for dealing with the notion of tree code. In particular we introduce clean and available definitions of factorization and factors of a tree, which allow us to give the concept of tree code in terms of these notions: a set T of trees is a tree code if any tree admits at most one factorization in elements of T.

We then consider the concept of stability (see also [14, 15]) and we characterize tree codes as the minimal generating sets of stable sets of trees. This characterization is very close to the algebraic one of word codes in terms of free monoids. Indeed, we will verify that, in the unary case (that is in the case of words) stable sets of words over an alphabet A correspond to free submonoids of A^*. From this characterization we also derive a defect theorem for trees, (cf. [2, 3] for the word case) which generalizes the analogous results for words. As a consequence, we give some necessary condition for a set of two trees to be a tree code.

Another relevant contribution of this paper is an efficent algorithm to test whether a finite set T of trees is a tree code. The algorithm is based on the construction of a directed graph $\mathcal{G}(T)$, whose size is at most $o(t^2)$, where t is the global size of T. This leads to a polynomial time algorithm to test the codicity of T.

In the last part of the paper we introduce the notion of tree equation as complementary to the one of tree codes. (cf. [3, 13, 16, 18, 19] for the corresponding problem on words) The search for solutions of tree equations gives rise to several new problems. In particular, the decidability of the existence of solutions for a given tree equation is left as an open problem. It generalizes at the same time both first-order unification and word unification and it appears to be a special case of second-order unification.

This paper is organized as follows: in Section 2 we give the main definitions used in the paper. In particular, we recall the notion of k-ary tree and all the terminology associated with this notion and we also give several definitons of concatenation between trees; this notion is in fact essential since it generalizes the notion of concatenation between words, that is the basic operation on the free monoid. In Section 3 we introduce the notion of factorization of a tree and the definition tree-code, and we also give a first characterization of tree codes. In Section 4 we consider the extension of the notion of stability to trees and we characterize tree codes in terms of stability. From this we derive, in Section 5, a defect theorem for trees. In Section 6 we consider the case of sets of trees of cardinality 2 and we compare the case of words with the case of trees.

Moreover we give, as an application of the defect theorem, the property that any k-ary tree over an alphabet A can be expressed in a unique way as power of a primitive tree. In Section 7 we give a new algorithm that permits to decide in polynomial time whether a given finite set of trees is a tree code. Finally in Section 8 we introduce the notion of tree-morphism and tree-equation as complementary to the one of tree codes. Moreover we introduce the problem of the solvability of tree equations which is a non-trivial generalization of the one-dimensional case (that is the satisfiability of word equations) whose solution is due to Makanin [19].

Other related problems about trees can be found in [6, 7, 27, 28].

Part of the results of this paper also appear in [21–23].

2. Basic definitions

In this section we give the main definitions used in the paper. We begin by recalling the formal definition of k-ary tree and the related terminology.

Definition 2.1. Let $\Sigma = \{1, 2, \ldots, k\}$ and let A be a finite alphabet. A *k-ary tree* over the label-alphabet A is a partial mapping

$$\tau : \Sigma^* \to A$$

whose domain $dom(\tau)$ is a finite and a prefix closed subset of Σ^*.

Let τ be a k-ary tree over an alphabet A. A *node* is an element $u \in dom(\tau)$. Given two nodes $u, v \in dom(\tau)$ we say that u is the *father* of v if $u = v \cdot i$ for some $i \in \Sigma$. In this case we say that v is the i-th son of u. A node without sons is called a *leaf*. The only node without father (ε) is called *the root*. The set

$$fr^+(\tau) = \{u \cdot i \mid u \in dom(\tau),\ i \in \Sigma,\ u \cdot i \notin dom(\tau)\}$$

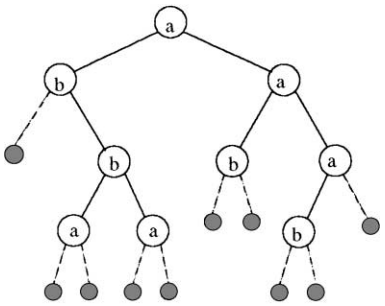
is called the *outer frontier* of τ. The size of τ, denoted by $|\tau|$, is the number of its nodes.

We denote by $(A, k)^{\#}$ the set of all finite k-ary trees over A. When k is implicit we write $A^{\#}$ instead of $(A, k)^{\#}$. In the examples given in this paper (when it is not specified) $A^{\#}$ denotes the set of all binary trees over A. Particular elements of $A^{\#}$ are the *empty tree*, denoted by Ω, whose domain is the empty set, and the *punctual trees*, that is, trees whose domain is $\{\varepsilon\}$. A punctual tree with label a will be simply denoted by a.

Remark 2.1. The definition of k-ary trees given here is consistent with the definition of word. In fact, a word $w \in A^*$ can be described as a partial mapping from the set of non-negative integers $\mathbb{N}$ (that is isomorphic to Σ^* when $\Sigma = \{1\}$) to a finite alphabet A, whose domain is the set $\{0, 1, \ldots, |w| - 1\}$ (isomorphic in Σ^* to the set $\{\varepsilon, 1, 1^2, \ldots, 1^{|w|-1}\}$, that is a prefix closed subset of $\{1\}^*$). Thus, we can always consider a word as a labeled 1-ary tree over A. Moreover, note that in the case of words the outer frontier contains only one element, $1^{|w|}$.

Note that in the figures of this paper, when we give evidence to the outer frontier, we draw all the edges outgoing from the node, denoting with the lacking sons dotted edges and black points.

Example 2.1. The binary tree graphically represented in the following figure



corresponds to the application $\tau : \{1,2\}^* \to \{a,b\}$ defined as
$\tau(\varepsilon) = a, \quad \tau(12) = b, \quad \tau(121) = a,$
$\tau(1) = b, \quad \tau(21) = b, \quad \tau(122) = a,$
$\tau(2) = a, \quad \tau(22) = a, \quad \tau(221) = b.$
The domain of this tree is the prefix closed set

$$dom(\tau) = \{\varepsilon, 1, 2, 12, 21, 22, 121, 122, 221\}$$

and the outer frontier is the set of words over $\{1,2\}^*$:

$$fr^+(\tau) = \{11, 1211, 1212, 1221, 1222, 211, 212, 2211, 2212, 222\}.$$

Remark 2.2. It can be easily proved by induction that a k-ary tree with n nodes has an outer frontier with cardinality equal to $(k-1)n+1$. From this formula one can also derive that in the case of words (that is, if $k=1$) the outer frontier contains one element independently of the size of the word.

2.1. Operations on trees

We can define several operations on trees. Most of these are already defined in [10] and are recalled here for the sake of completeness. First of all we are interested in the operation of concatenation between trees. Intuitively, the concatenation between two trees τ_1 and τ_2 is obtained by "attaching" the root of τ_2 to one of the elements of the outer frontier of τ_1. It is clear that this operation cannot have, as a result, a unique tree, since in general the outer frontier of a tree contains more than one element. Then the concatenation between two trees will be a set of trees containing as many elements as the number of elements in $fr^+(\tau_1)$. First, we define the concatenation of two trees at a given element of the outer frontier.

Definition 2.2. Let τ_1 and τ_2 be two k-ary trees and let $fr^+(\tau_1)$ denote the outer frontier of τ_1. The concatenation of τ_2 to τ_1 at $f \in fr^+(\tau_1)$ is the tree $\tau_1(f)\tau_2$ defined by

$$dom(\tau_1(f)\tau_2) = dom(\tau_1) \cup f \cdot dom(\tau_2),$$

$$\forall u \in dom(\tau_1(f)\tau_2), (\tau_1(f)\tau_2)(u) = \begin{cases} \tau_1(u) & \text{if } u \in dom(\tau_1), \\ \tau_2(v) & \text{if } fv = u \text{ and } v \in dom(\tau_2). \end{cases}$$

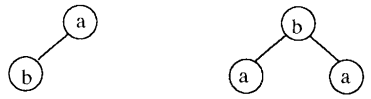
Definition 2.3. Let τ be a tree with outer frontier $fr^+(\tau) = \{f_1, f_2, \ldots, f_n\}$ and let $\tau_1, \ldots, \tau_n$ be a sequence of n trees (eventually empty). The *simultaneous concatenation* of $\tau_1, \ldots, \tau_n$ to all the nodes of the outer frontier of τ, denoted by $\tau\langle\tau_1, \tau_2, \ldots, \tau_n\rangle$ is the result of concatenating the trees $\tau_1, \tau_2, \ldots, \tau_n$ to τ at $f_1, f_2, \ldots, f_n$, respectively, i.e.,

$$\tau\langle\tau_1, \tau_2, \ldots, \tau_n\rangle = (\ldots((\tau(f_1)\tau_1)(f_2)\tau_2)\ldots(f_n)\tau_n).$$

Definition 2.4. Let τ_1 and τ_2 be two trees and let $fr^+(\tau_1)$ be the outer frontier of τ_1. The *concatenation* of τ_1 with τ_2 is the set:

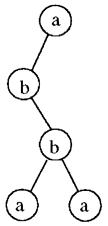
$$\tau_1\tau_2 = \{\tau_1(f)\tau_2 \mid f \in fr^+(\tau_1)\}.$$

Example 2.2. Given the binary trees

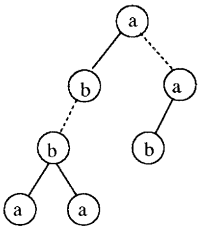


τ_1 τ_2

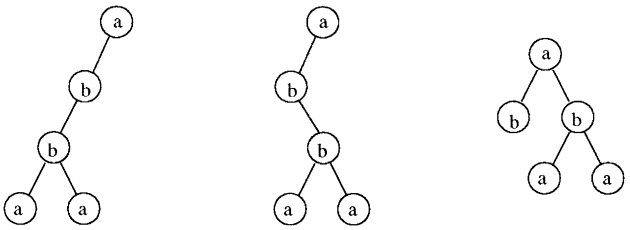
the following is the concatenation of τ_2 to τ_1 at element 12 of the outer frontier, that is, $\tau_1(12)\tau_2$:



The simultaneous concatenation of τ_2, Ω and τ_1 to the outer frontier of τ_1, that is, $\tau_1\langle\tau_2, \Omega, \tau_1\rangle$ is the tree



The concatenation of τ_1 and τ_2, that is, $\tau_1\tau_2$ is the set containing the following trees:



Definition 2.5. Given a set of trees $\mathscr{T}$, we say that T is *closed* (*under concatenation*) if $\tau_1, \tau_2 \in \mathscr{T}$ implies $\tau_1\tau_2 \subset \mathscr{T}$.

It is easy to verify that the intersection of a family of sets of trees closed under concatenation is a set of trees closed under concatenation. This fact allows us to give the following definitions:

Definition 2.6. Given a set of trees T, the *closure of* T (*w.r.t. concatenation*), denoted by $T^{\#}$, is the minimal set of trees closed under concatenation containing T.

A particular case is where $T = \{\tau\}$. In this case we write $\tau^{\#}$ instead of $\{\tau\}^{\#}$.

Definition 2.7. Given two trees τ, τ_0, we say that τ is a *power* of τ_0, if $\tau \in \tau_0^{\#}$.

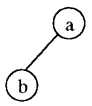
The notion of primitiveness plays an important role in the theory of words. This notion can be extended to trees:

Definition 2.8. Given a tree τ, we say that τ is a *primitive tree* if the condition $\tau \in \tau_0^{\#}$ implies $\tau = \tau_0$.

Example 2.3. Consider the binary trees τ_1 and τ_2 in the figure below:

τ_1 τ_2

τ_1 is a primitive tree since it is not a power of another tree, whereas τ_2 is not primitive since it is power of the (primitive) tree τ in the following figure:



τ

We remark that in the case of unary trees, the notion of primitive tree coincides with the one of primitive word.

Given a tree τ we can define on $dom(\tau)$ the prefix partial order relation. As notation, if $v_1, v_2 \in dom(\tau)$, we write $v_1 \leqslant v_2$ if v_1 is a prefix of v_2. If v_1 is a strict prefix of v_2 we write $v_1 < v_2$.

A *subtree* of a tree τ rooted at the node $v \in dom(\tau)$, is a tree τ_v such that $v \cdot dom(\tau_v) \subset dom(\tau)$ and $\tau_v(w) = \tau(vw)$. If $v = \varepsilon$, then we write $\tau_\varepsilon \subseteq \tau$ and we say that τ_ε is an *initial subtree* or a *prefix* of τ. If $v \cdot dom(\tau_v) = v\Sigma^* \cap dom(\tau)$, then we say that τ_v is a *terminal subtree* or *suffix* of the tree τ. In what follows, when it is not specified, τ_v will denote the terminal subtree of a tree τ, rooted at the node v. We say that two terminal subtrees of a tree τ, say τ_v and τ_w, are *independent* if neither $v \leqslant w$ nor $w \leqslant v$.

We also need some other operations on trees defined in [8]. These operations require the trees involved to be compatible.

Definition 2.9. Let τ and σ be two labeled trees. We say that τ and σ are *compatible* if they agree on the intersection of their domains. Formally, if $\tau_{|D}$ denotes the restriction of τ to the set $D \subseteq dom(\tau)$, the requirement is

$$\tau_{|(dom(\tau) \cap dom(\sigma))} = \sigma_{|(dom(\tau) \cap dom(\sigma))}.$$

Note that two unlabeled trees are always compatible.

Definition 2.10. Let τ and σ be two compatible trees. The *union* of τ and σ is the tree $\tau \oplus \sigma$ defined by

(i) $dom(\tau \oplus \sigma) = dom(\tau) \cup dom(\sigma)$, and

(ii)

$$\forall x \in dom(\tau \oplus \sigma),\ (\tau \oplus \sigma)(x) = \begin{cases} \tau(x) & \text{if } x \in dom(\tau), \\ \sigma(x) & \text{if } x \in dom(\sigma). \end{cases}$$

The *intersection* of τ and σ is the tree $\tau \sqcap \sigma$ such that

(i) $dom(\tau \sqcap \sigma) = dom(\tau) \cap dom(\sigma)$, and

(ii) $\tau \sqcap \sigma = \tau_{|(dom(\tau) \cap dom(\sigma))} = \sigma_{|(dom(\tau) \cap dom(\sigma))}$.

The *symmetric difference* of the compatible trees τ and σ is the set of trees defined as follows:

$$\tau \triangle \sigma = \{(\tau \oplus \sigma)_v \neq \Omega \mid v \in fr^+(\tau \sqcap \sigma)\}.$$

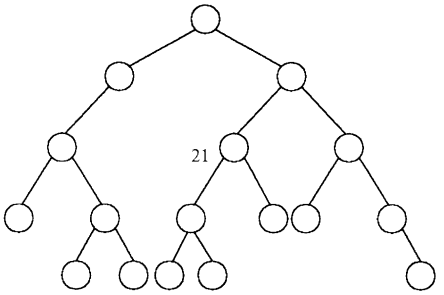
Informally, the symmetric difference of two trees is the set of subtrees that we get after deleting their intersection from the top of their union.

3. Factorizations and codes

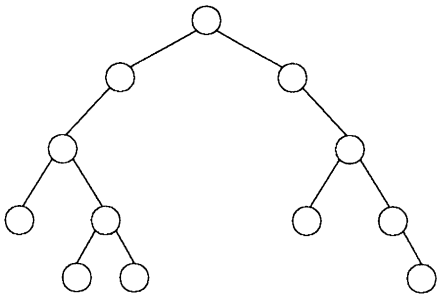
Definition 3.1. Let τ be a tree and $v \in dom(\tau)$. The *pruning* of a tree τ by a terminal subtree τ_v, denoted by $\tau\tau_v^{-1}$, is the restriction of τ to the domain $D = dom(\tau)\backslash v \cdot dom(\tau_v)$. In other words $\tau\tau_v^{-1}$ is the tree such that $(\tau\tau_v^{-1})(v)\tau_v = \tau$.

We can generalize this operation to the pruning of a tree by a set of independent terminal subtrees $\tau_{v_1}, \ldots, \tau_{v_n}$, and we denote it by $\tau(\tau_{v_1}, \ldots, \tau_{v_n})^{-1}$, as the restriction of τ to the domain $D = dom(\tau)\backslash(v_1\, dom(\tau_{v_1}) \cup \cdots \cup v_n\, dom(\tau_{v_n}))$.

Example 3.1. Consider the tree in the figure that follows:



The pruning of τ by τ_{21} is the tree



Definition 3.2. Given two trees τ and σ, the *right quotient* of τ by σ is the set

$$\tau\sigma^{-1} = \{\tau\tau_u^{-1}\, u \in dom(\tau),\ \tau_u = \sigma\}.$$

Consequently, it is empty if σ is not a suffix of τ. The *left quotient* of τ by σ is the set

$$\sigma^{-1}\tau = \{\tau_u \mid u \in fr^+(\sigma) \cap dom(\tau)\}$$

and it is empty if σ is not a prefix of τ.

Note that the pruning of τ by $\tau_u = \sigma$ yields an element of the quotient $\tau\sigma^{-1}$.

Given the partial order relation ($\leqslant$) defined on the elements of $dom(\tau)$, for every subset $S \subset dom(\tau)$ we can define the partial order relation induced by ($\leqslant$). Taken

$v, w \in S$, we say that w is a *successor* for v in S if $v < w$ and there is no $u \in S$ such that $v < u < w$.

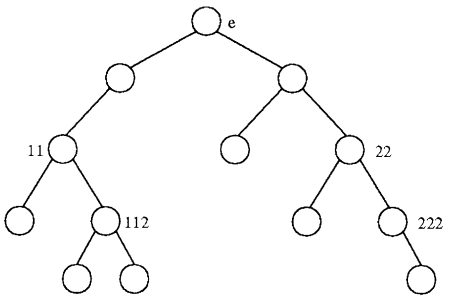
Definition 3.3. Let τ be a k-ary tree. A *factorization* of τ is a set $F \subseteq dom(\tau)$ such that $\varepsilon \in F$. Any $f \in F$ uniquely identifies a *factor* σ of the factorization F, which is the tree defined as

$$\sigma = \tau_f(f^{-1}\tau_{f_1}, \ldots, f^{-1}\tau_{f_k})^{-1}$$

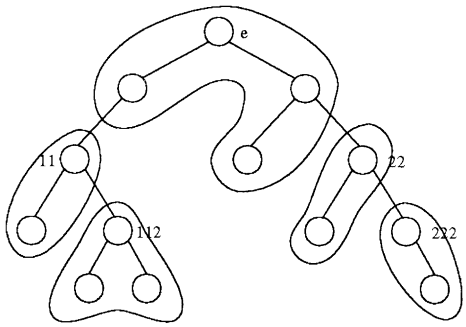
where $f_1, f_2, \ldots, f_k$ are all the successors of f in F.

To each factorization F of a tree τ uniquely corresponds the set of its factors, that we denote by $T(F)$. Informally speaking, the factors of the factorization are the "pieces" of trees individuated by the factorization, as we stress in the following example.

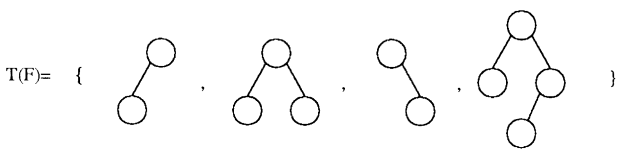
Example 3.2. Consider the tree



and the factorization $F = \{\varepsilon, 11, 22, 112, 222\}$. The factorization permits a "splitting" of the tree into pieces in the following way:



and so individuates the following set of factors for F:



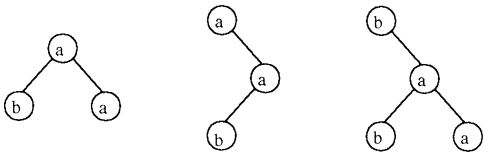
Definition 3.4. Given a set of trees T, a factorization F of a tree τ is called a *T-factorization* if all of its factors belong to T. We say that τ has a *T-factorization* or *it factorizes* in elements of T.

From the previous definitions one can easily derive the following proposition.

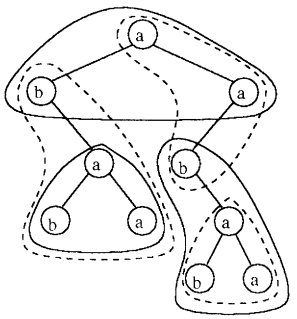
Proposition 3.1. *Given a set of trees T, a tree τ belongs to $T^{\#}$ if and only if τ factorizes in elements of T.*

A tree $\tau \in T$ admits in general more than one T-factorization, as shown by the following example.

Example 3.3. Consider the following set of trees:



The tree in the figure that follows has two different T-factorization, $F_1 = \{\varepsilon, 12, 21\}$ and $F_2 = \{\varepsilon, 1, 212\}$. We represent F_1 and F_2 in the same figure by denoting the factors of the first factorization with a continuous line and the factors of the second one by a dashed line:



Definition 3.5. Let T be a set of k-ary trees. We say that T is a *tree code* if every element in $T^{\#}$ has a unique factorization in the elements of T.

The following definition is related to the previous one.

Definition 3.6. Let T be a set of trees that is not a tree code. We say that a tree $\tau \in A^{\#}$ is *T-ambiguous* if it admits two different T-factorizations. A tree is called *minimal T-ambiguous* if it is T-ambiguous and all of its proper subtrees are not T-ambiguous.

Remark 3.1. Given a set of trees T that is not a tree code, we can always assume that there exist minimal T-ambiguous trees.

Given a tree $\tau \in T^{\#}$, we say that two T-factorizations F_1 and F_2 are *independent* if $F_1 \cap F_2 = \{\varepsilon\}$. Note that if a tree τ minimal T-ambiguous, then its factorizations are pairwise independent, otherwise we could split τ into a set of smaller trees having a double T-factorization.

The following theorem gives a property that characterizes sets of trees that are tree-codes.

Theorem 3.1. *Let T be a set of k-ary trees. T is a tree-code if and only if for all pairs of trees $\tau_1, \tau_2 \in T$, with $\tau_1 \neq \tau_2$, $\tau_1 T^{\#} \cap \tau_2 T^{\#} = \emptyset$.*

Proof. Let T be a tree code that is, for all $\tau \in T^{\#}$, τ has a unique factorization in elements of T. Then for all $\tau_1, \tau_2 \in T$ with $\tau_1 \neq \tau_2$, $\tau_1 T^{\#} \cap \tau_2 T^{\#} = \emptyset$ because otherwise we would have a tree $\tau \in \tau_1 T^{\#} \cap \tau_2 T^{\#}$ obtained in two different ways (since $\tau_1 \neq \tau_2$) as concatenation of elements in T.

Suppose now that for all τ_1, $\tau_2 \in T$ with $\tau_1 \neq \tau_2$, $\tau_1 T^{\#} \cap \tau_2 T^{\#} = \emptyset$. Suppose there is a tree τ in $T^{\#}$ having two different T-factorizations. W.l.o.g. we can consider τ minimal T-ambiguous. Let us denote by F_1 and F_2 two of its different T-factorizations. Let τ_ε^1 (τ_ε^2 respectively) be the factor of the factorizations F_1 (F_2 respectively) rooted at the root of τ. Since τ is minimal T-ambiguous, then $\tau_\varepsilon^1 \neq \tau_\varepsilon^2$; otherwise we could find a subtree of τ having two different T-factorizations. The existence of a double T-factorization for τ implies that $\tau \in \tau_\varepsilon^1 T^{\#} \cap \tau_\varepsilon^2 T^{\#} \neq \emptyset$ that contradicts the hypothesis. □

4. Stable sets

In this section we characterize a tree code T in terms of a property, called stability, of the set $T^{\#}$. We see that in the unary case (that is in the case of words) stable sets of words over an alphabet A correspond to free submonoids of A^*.

Definition 4.1. Let $\mathscr{T}$ be a set of trees closed under concatenation and containing the empty tree Ω. We say that a set of trees T is a *set of generators* (or a *generating set*) for $\mathscr{T}$ if $T^{\#} = \mathscr{T}$. Moreover we say that T is a *minimal generating set* for $\mathscr{T}$ if, for any set of generators T' of $\mathscr{T}$ we have that $T' \subseteq T$ implies $T' = T$.

Proposition 4.1. *Let $\mathscr{T}$ be a subset of $A^{\#}$ closed under concatenation and containing the empty tree Ω. Then $\mathscr{T}$ has a unique minimal set of generators $T = (\mathscr{T} - \Omega) - (\mathscr{T} - \Omega)^2$.*

Proof. We prove first that T is a set of generators for $\mathscr{T}$, that is $T^{\#} = \mathscr{T}$. Obviously $T^{\#} \subseteq \mathscr{T}$. We prove the opposite inclusion by induction on the size of the trees. First, Ω is in $T^{\#}$. If τ is in $(\mathscr{T} - \Omega)$ and $\tau \notin (\mathscr{T} - \Omega)^2$, then $\tau \in T \in \mathscr{T}^{\#}$. Otherwise $\tau \in \tau_1 \tau_2$ where $\tau_1, \tau_2 \in (\mathscr{T} - \Omega)$ and $|\tau_1|, |\tau_2| < |\tau|$. By the inductive hypothesis τ_1, $\tau_2 \in T^{\#}$ and this implies $\tau \in T^{\#}$.

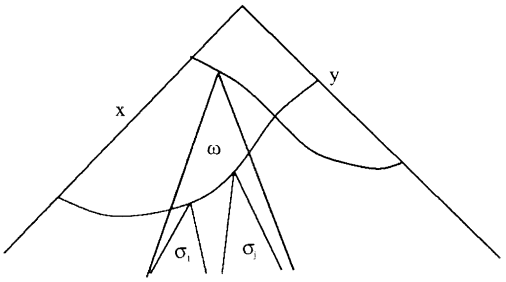
We now prove that T is the unique minimal set of generators for $\mathscr{T}$. Let X be another set of generators for $\mathscr{T}$, that is, $X^{\#} = \mathscr{T}$. We can suppose that $\Omega \notin X$. Taken $\tau \in T$, then $\tau \in X^{\#}$. Since $\tau \neq \Omega$ and $\tau \notin (\mathscr{T} - \Omega)^2$, τ is not concatenation of two words in $X^{\#}$. The only possibility is that $\tau \in X$ that is $T \subset X$. □

The following definition introduces the notion of stability for a set of trees (see also [14, 15]).

Definition 4.2. We say that a set $\mathscr{T} \subset A^{\#}$ closed under concatenation is *stable* (in $A^{\#}$) if for $\tau \in \mathscr{T}$, $x, y \in A^{\#}$ such that x and y are prefixes of τ,

$$\tau, x, y \in \mathscr{T};\ x^{-1}\tau, \quad y^{-1}\tau \subset \mathscr{T} \Rightarrow x \sqcap y \in \mathscr{T}, \quad x \triangle y \subset \mathscr{T}.$$

By the following figure we can illustrate the meaning of the definition of stable sets:



This definition means that if a tree τ has a double factorization in $\mathscr{T}$, the first one beginning with x and the second one beginning with y, then trees x and y "cut" each other into trees that are still in $\mathscr{T}$.

In the special case of unary trees, i.e., in the case of words, stable sets of trees correspond to stable submonoids of a free monoid. Recall that stability characterizes those submonoids that are themselves free (cf. [1]).

Let τ be a tree having a T-factorization F. We say that a subtree τ_u of τ *inherits* the T-factorization F if $u^{-1}(F \cap u \cdot dom(\tau_u))$ is a T-factorization for τ_u.

Remark 4.1. If a tree τ admits a T-factorization F, every terminal subtree τ_u of τ rooted at a node $u \in F$ inherits the T-factorization F. Moreover, every subtree of τ of the form $\tau\tau_u^{-1}$ with $u \in F$ inherits the T-factorization F.

The following theorem characterizes tree codes in terms of stable subsets of $A^{\#}$. This proof can be found also in [14].

Theorem 4.1. *Let $\mathscr{T}$ be a subset of $A^{\#}$ closed under concatenation and containing the empty tree. $\mathscr{T}$ is stable if and only if the minimal set of generators T of $\mathscr{T}$ is a tree code.*

Proof. Suppose that $\mathscr{T}$ is stable. If the minimal generating set T is not a tree code, then there exists a pair of trees $x, y \in T$ such that $xT^{\#} \cap yT^{\#} \neq \emptyset$. Consider $\tau \in xT^{\#} \cap yT^{\#}$.

Clearly, x and y are compatible, and are elements of $\mathscr{T}$. Moreover, by definition, the trees in $x^{-1}\tau$ and $y^{-1}\tau$ are in $\mathscr{T}$. Since $\mathscr{T}$ is stable, this implies that $x \triangle y \subset \mathscr{T}$ and $x \sqcap y \in \mathscr{T}$. Then x and y can be obtained as concatenations of $x \sqcap y$ with elements in $x \triangle y$, and so they belong to $(\mathscr{T} - \Omega)^2$. This contradicts with the hypothesis that $x, y \in T$.

Conversely, suppose T is a tree code. We have to prove that $\mathscr{T}$ is stable. Consider two compatible trees $x, y \in A^{\#}$ and $\tau \in A^{\#}$ with x and y prefixes of τ such that $x, y, \tau \in \mathscr{T}$, $x^{-1}\tau, y^{-1}\tau \subset \mathscr{T}$. We would like to prove that $x \triangle y \subset \mathscr{T}$ and $x \sqcap y \in \mathscr{T}$. Our hypothesis means that the tree τ admits two different $\mathscr{T}$-factorizations, the first one beginning with x and the second beginning with y. Let us denote by F_x and F_y these factorizations. Since T is a tree code, a straightforward argument shows that $F_x \cup F_y$ is a $\mathscr{T}$-factorization for τ. This implies that $x \sqcap y \in \mathscr{T}$ and $x \triangle y \subset \mathscr{T}$ as was to be proved. □

5. The defect theorem

In this section we introduce the notion of *stable hull* for a set of trees. This definition permits to extend the *defect theorem*, in its stronger form, to sets of trees. In particular, we find that if a set of trees T is not a tree code, then we can find a tree code C, with $|C| < |T|$, such that $T \subset C^{\#}$.

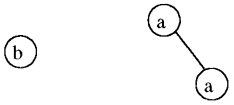
Proposition 5.1. *If $\{\mathscr{T}_i\}_{i \in I}$ is a family of stable subsets of $A^{\#}$, then $\mathscr{T} = \bigcap_{i \in I} \mathscr{T}_i$ is a stable subset of $A^{\#}$.*

Proof. Let τ be a tree in $\mathscr{T}$ and u, v with $u < v$ be two nodes of τ such that $\{\tau_u, \tau_v, \tau\tau_u^{-1}, \tau\tau_v^{-1} \in \mathscr{T}\}$; since each $\mathscr{T}_i$ $(i \in I)$ is stable, $\tau_u\tau_v^{-1} \in \mathscr{T}_i$ $(i \in I)$, and $\tau_u\tau_v^{-1} \in \mathscr{T}$. □

Given a set of trees T, Proposition 5.1 permits to define the *stable hull for T*.

Definition 5.1. Given a set of trees T, the *stable hull for T* is the minimal stable subset of $A^{\#}$ that contains T, denoted by $\mathscr{H}(T)$, and it is obtained as the intersection of all the stable subsets of $A^{\#}$ containing T. We denote by $H(T)$ the minimal set of generators of $\mathscr{H}(T)$.

Example 5.1. One can verify that the stable hull of the set of trees in Example 3.3 is the set of trees generated by the following pair of trees:



Corollary 5.1. *T is a tree code if and only if $T = H(T)$.*

Finally the following theorem generalizes the defect theorem to trees.

Theorem 5.1 (Defect Theorem for trees). *Let T be a set of trees that is not a tree code. Then $|H(T)|<|T|$.*

Proof. Let $\mathscr{H}(T)$ denote the stable hull for T and let $H=H(T)$ denote the minimal set of generators for $\mathscr{H}(T)$. By Theorem 4.1, H is a tree code. We have to prove that $|H|<|T|$. First we prove the following fact:

- Each element in H appears as prefix in at least one element of T.

In fact, suppose there exists an element σ in H that is not prefix of some element of T. Then consider the set $\mathscr{J}=\mathscr{H}-\sigma\mathscr{H}$. Trivially $T\subset\mathscr{J}$. Moreover $\mathscr{J}$ is closed under concatenation. In fact, suppose there exists a sequence of trees $\tau,\tau_1,\tau_2,\ldots,\tau_m\in\mathscr{J}$ such that the concatenation $\tau\langle\tau_1,\ldots,\tau_m\rangle\in\sigma\mathscr{H}$. This means that the tree $\tau\langle\tau_1,\ldots,\tau_m\rangle$ $(\in\sigma\mathscr{H})$ has two different H-factorizations (one beginning with the first factor of τ as an element of $\mathscr{J}$ and the other beginning with σ), which contradicts the fact that H is a tree code.

Now, take a tree $\tau\in\mathscr{J}$ and two compatible trees x and y both prefixes of τ such that $x,y\in\mathscr{J}$ and $x^{-1}\tau,y^{-1}\tau\subset\mathscr{J}$. Since $\mathscr{H}$ is stable, $x\sqcap y\in\mathscr{H}$ and $x\triangle y\subset\mathscr{H}$. Moreover, since $\tau_u\notin\sigma\mathscr{H}$ and for each $\tau'\in x^{-1}\tau\cup y^{-1}\tau$, $\tau'\notin\sigma\mathscr{H}$, $x\sqcap y\notin\sigma\mathscr{H}$ and for each $z\in x\triangle y$, $z\notin\sigma\mathscr{H}$. Consequently, $x\sqcap y\in\mathscr{J}$ and $x\triangle y\subset\mathscr{J}$. This proves that $\mathscr{J}$ is stable and contains T, contradicting the minimality of $\mathscr{H}$.

Define now the application $\alpha:T\to H$ defined as $\alpha(\tau)=\sigma$ if $\tau\in\sigma\mathscr{H}$. This application is surjective for what we proved above, but it is not injective since T is not a code; then there exists $\tau_1,\tau_2\in T$ such that $\tau_1T^{\#}\cap\tau_2T^{\#}\neq\emptyset$. Let $\tau\in\tau_1T^{\#}\cap\tau_2T^{\#}$. Since H is a code, τ has a unique H-factorization, and the first factor τ_ε of τ in the H-factorization is a prefix of both τ_1 and τ_2.

Since $\alpha:T\to S$ is surjective but not injective, $|T|>|H|$. □

Example 5.2. The set of trees in Example 5.1 is the generating set of the stable hull of the set of trees T in Example 3.3. Since T is not a code, the size of the generating set $H(T)$ is less than that of T.

6. Sets with two trees

In this section we consider sets with two trees. It is well known that the following proposition characterizes codes with two words.

Proposition 6.1. *A set $X=\{u,v\}$ of words in A^* is not a (word) code if and only if u and v are both powers of the same word.*

The proof is a consequence of the defect theorem (for words) in one direction, and is trivial in the other direction (since if $u=w^k$ and $v=w^h$ then $vu=uv$).

In the case of trees, the following proposition generalizes in one direction Proposition 6.1.

Proposition 6.2. *If $T=\{\tau_1,\tau_2\}$ is not a tree code, then τ_1 and τ_2 are powers of the same tree.*

Proof. The proof is a direct consequence of the defect theorem applied to the case $n=2$. □

As an application of Proposition 6.2, we extend a well-known combinatorial property of words (cf. [17]) to trees.

Theorem 6.1. *Let τ be a labeled tree. Then there exists a unique primitive tree σ such that $\tau\in\sigma^{\#}$.*

Proof. We prove the statement by induction on the size of the tree. If τ is a punctual tree a, then τ is a primitive tree and we are done.

Let us suppose now that the theorem is true for every tree τ' such that $|\tau'|<|\tau|$. If τ is primitive, the statement is verified. Otherwise, there exists a tree σ (with $|\sigma|<|\tau|$) such that $\tau\in\sigma^{\#}$. By the inductive hypothesis, the statement is true for σ; thus there exists a primitive tree ρ such that $\sigma\in\rho^{\#}$; hence $\tau\in\sigma^{\#}\subseteq\rho^{\#}$.

This primitive tree is unique. In fact if there were two different primitive trees σ_1 and σ_2 such that $\tau\in\sigma_1^{\#}$ and $\tau\in\sigma_2^{\#}$ then τ would have a double $\{\sigma_1,\sigma_2\}$-factorization, that is, $\{\sigma_1,\sigma_2\}$ is not a tree code. By Proposition 6.2, σ_1 and σ_2 must be powers of the same tree, and this contradicts the primitiveness of σ_1 and σ_2. □

We remark that, unlike the case of words, the sufficient condition in Proposition 6.2 is not necessary. In fact, we can give several examples of pairs of trees that are both power of the same tree and that are, at the same time, tree codes.

Example 6.1. The following pair of trees is a tree code, but the two trees are both powers of the same punctual tree a.

In what follows, we stress some differences and similarities between the word case and the tree case, regarding the problem of the characterization of sets with two elements.

Given two words u and v in a free monoid A^*, the following three conditions are equivalent:

(1) $u^*\cap v^*\neq\{\varepsilon\}$;
(2) $\{u,v\}$ is not a code;
(3) $u,v\in w^+$, for some $w\in A^*$.

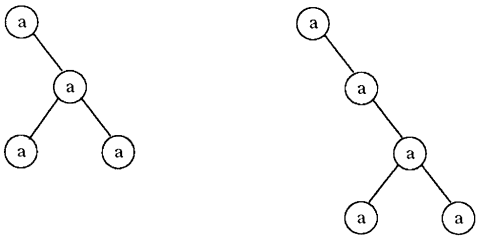
Given two labeled k-ary trees τ_1 and τ_2 over an alphabet A, consider the following conditions, corresponding to the above ones for trees:

$(1')$ $\tau_1^{\#} \cap \tau_2^{\#} \neq \{\Omega\}$;

$(2')$ $\{\tau_1, \tau_2\}$ is not a tree code;

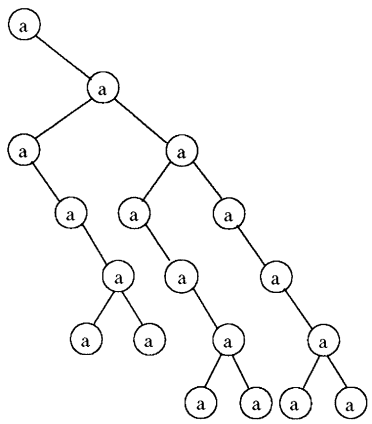
$(3')$ $\tau_1, \tau_2 \in \tau^{\#}$, for some $\tau \in A^{\#}$.

It is trivial that $(1') \Rightarrow (2')$. In fact if $\tau \in \tau_1^{\#} \cap \tau_2^{\#}$, then τ has two different $\{\tau_1\tau_2\}$-factorizations. Moreover $(2') \Rightarrow (3')$, as stated in Proposition 6.2. In any case, unlike the case of words, $(3') \not\Rightarrow (2')$ as proved by Example 6.1. Moreover $(2') \not\Rightarrow (1')$, that is, there exist pairs of trees that are not tree codes, but that do not have a common power. For example the following pair of trees



τ_1 τ_2

is not a tree code. In fact, it can be seen that the following tree have two different $\{\tau_1, \tau_2\}$-factorizations:



It can be also proved (cf. [26]) that τ_1 and τ_2 cannot have a common power.

These remarks show that the situation is more complicated for trees than for words. In particular, the problem of characterizing those pairs of trees that are tree codes remains open.

7. A test for tree-codes

In this section we give an algorithm to decide whether a finite set T of trees is a tree code. We will refer to this problem as *the code problem for trees*. In [25] Nivat proves

the decidability of the code problem for trees. Our algorithm, described in terms of graphs, efficiently estabilishes whether a given set of trees has the property of being a code. The conception of our algorithm is inspired by the approach of Matiyasevitch in [24] to handle the code problem for words and its extension to partially commutative alphabets.

We begin by formalizing the code problem for trees. By Theorem 3.1 the code problem for trees can be stated as follows:

- *Instance* 1: A set $T=\{\tau_1,\tau_2,\ldots,\tau_n\}$ of k-ary labeled trees over an alphabet A.
- *Question* 1: Are there pairs of trees $\tau_i,\tau_j\in T$ with $\tau_i\neq\tau_j$ such that

$$\tau_i\langle\lambda_1,\ldots,\lambda_l\rangle=\tau_j\langle\lambda'_1,\ldots,\lambda'_m\rangle$$

for some $\lambda_1,\ldots,\lambda_l,\lambda'_1,\ldots,\lambda'_m\in T^{\#}$?

Question 1 can be immediately be solved once we solve the following more general problem:

- *Instance* 2: A set $T=\{\tau_1,\ldots,\tau_n\}$ and a pair (σ,ρ) of k-ary labeled trees over A.
- *Question* 2: Does the relation

$$\sigma\langle\lambda_1,\ldots,\lambda_l\rangle=\rho\langle\lambda'_1,\ldots,\lambda'_m\rangle$$

hold for some $\lambda_1,\ldots,\lambda_l,\lambda'_1,\ldots,\lambda'_m\in T^{\#}$?

Depending on the different form of the pair of trees instanced, the possible answers to Question 2 are the following:

1. If the pair of trees is (Ω,Ω), the answer is trivially YES.
2. If the pair of trees is of the form

$$(a\langle\sigma_1,\ldots,\sigma_k\rangle,a'\langle\rho_1,\ldots,\rho_k\rangle)$$

with $a,a'\in A$ and $a\neq a'$, the answer is trivially NO.

3. If the pair of trees has the form

$$(a\langle\sigma_1,\ldots,\sigma_k\rangle,a\langle\rho_1,\ldots,\rho_k\rangle)$$

with $a\in A$, then the answer is YES if and only if the answer to *all of* the instances:

$$(\sigma_1,\rho_1),\ldots,(\sigma_k,\rho_k)$$

is YES.

4. If $(\sigma,\rho)=(\sigma,\Omega)$. The answer is YES if and only if *at least one* of the istances $(\sigma,\tau_1),\ldots,(\sigma,\tau_n)$ is YES, where $\tau_1,\ldots,\tau_n$ are all the trees in T.

Let $T=\{\tau_1,\ldots,\tau_n\}$ be a set of labeled k-ary trees over an alphabet A. As notation, we denote by $St(T)$ the set of all the terminal subtrees of the trees contained in the set T. Based on Question 2, the algorithm we propose to test if a set T is or not a code, consists of two phases:

- the first phase consists in the construction of a directed graph $\mathscr{G}(T)$, whose nodes, distingished into 'OR', 'AND' and 'FINAL' nodes, depending on the labels, are labeled by unordered pairs of elements of $St(T)$, and whose edges are labeled by elements in the set $\{\varepsilon,1,2,\ldots,k\}$;

- the second phase consists in a recursive assignment of a "quality" (ACCEPT or REJECT) to the nodes of the graph.

We will prove that T is a tree code if and only if all the initial nodes have quality REJECT.

7.1. The construction of the graph $\mathscr{G}(T)=(V(T),E(T))$

Initialization: We create $n(n-1)/2$ nodes and we label them by the pairs (τ_i,τ_j) with $\tau_i\neq\tau_j$, $\tau_i,\tau_j\in T$. We will refer to these nodes as *initial nodes*.

Recurrence: Starting from the initial nodes we proceed as follows. For each node x already in the graph, we update the graph by applying (depending on cases) one of the following rules:

- **The simplification rule**

  If the node x is labeled by a pair

  $$(\sigma,\rho)=(a\langle\sigma_1,\ldots,\sigma_k\rangle,a\langle\rho_1,\ldots,\rho_k\rangle)$$

  with $a\in A$, we qualify x as an AND node. In this case we consider the nodes $y_1,\ldots,y_k$ labeled $(\sigma_1,\rho_1),\ldots,(\sigma_k,\rho_k)$, respectively (if some of these nodes do not already exist, we create it) and create k new edges $(x,y_1),\ldots,(x,y_k)$ labeled $1,\ldots,k$, respectively.
- **The splitting rule**

  If the node x is labeled (σ,Ω) for some $\Omega\neq\sigma\in A^{\#}$, then we qualify x as an OR node. In this case we consider the nodes $z_1,\ldots,z_n$ labeled $(\sigma,\tau_1),\ldots,(\sigma,\tau_n)$ respectively, where $\tau_1,\ldots,\tau_n$ are the trees in T (if some of these nodes do not already exist, we create it) and we introduce n new edges $(x,z_1),\ldots,(x,z_n)$ labeled ε.
- **The stopping rule**

  If $(\sigma,\rho)=(\Omega,\Omega)$ or $(\sigma,\rho)=(a\langle\sigma_1,\ldots,\sigma_k\rangle,a'\langle\rho_1,\ldots,\rho_k\rangle)$ with $a\neq a'$ then the node x is called a FINAL node. FINAL nodes have no next nodes.

The procedure for the construction of the graph stops when for each node in the graph all its fan out is individuated.

7.2. Assignment of qualities to the nodes

Starting from the final nodes up to the initial nodes, we can recursively assign a quality ACCEPT/REJECT to the nodes of the graph $\mathscr{G}(T)$ by applying the rules that follow, that reflect the possible answers YES/NO to Question 2:

1. The node labeled (Ω,Ω) has the quality ACCEPT.
2. The nodes labeled $(a\langle\sigma_1,\ldots,\sigma_k\rangle,a'\langle\rho'_1,\ldots,\rho'_k\rangle)$ with $a\neq a'$ have the quality REJECT.
3. An AND node has the quality ACCEPT if all of its next nodes have the quality ACCEPT. It has the quality REJECT if at least one of its next nodes has the quality REJECT.
4. An OR node has the quality ACCEPT if at least one of its next nodes has the quality ACCEPT. It has the quality REJECT if all of its next nodes have the quality REJECT.

All the nodes that cannot be assigned a quality by this procedure, have the quality REJECT.

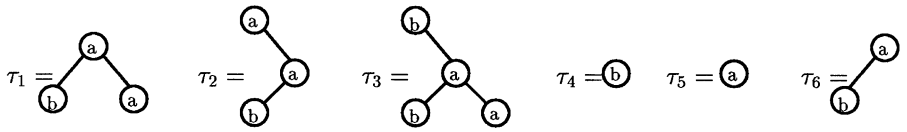
Once we have the graph $\mathcal{G}(T)$, the qualities assigned to the states allow to check the existence of a tree having a double T-factorization or, conversely, to prove that T is a tree code.

Theorem 7.1. *A set of trees $T=\{\tau_1,\ldots,\tau_n\}$ is a tree code if and only if all the initial nodes of the graph $\mathcal{G}(T)$ have the quality REJECT.*

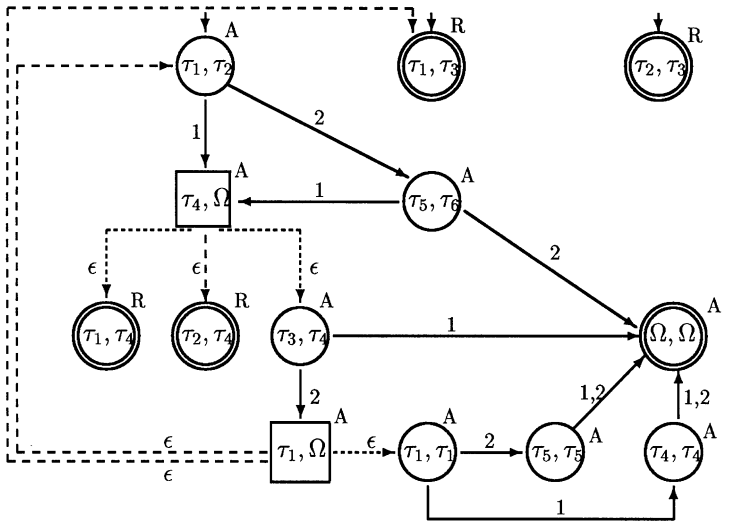
Proof. The proof of the correctness of the algorithm is an immediate consequence of the possible answers we can give to Question 2 depending on the pair of trees we are considering. Question 1 corresponds, in fact, to as many queries of Question 2 as the number of pairs of trees in T that are given as instances to Question 2. The assignment of the qualities ACCEPT/REJECT to the nodes corresponds to the answers YES/NO to Question 2. Moreover, the remaining nodes are assigned the quality REJECT since they belong to loops that do not lead to FINAL nodes (that is give rise to infinite trees). □

We remark that the graph we construct to decide the code problem for a set of trees T, when T is not a code, can also be used to find the set of all T-ambiguous trees, that is trees having at least two different T-factorizations, corresponding to "accepting" paths in the graph. Note that minimal T-ambiguous trees correspond to the minimals of such paths.

Example 7.1. Consider for example the set of trees $\{\tau_1,\tau_2,\tau_3\}$ (that is not a tree code) defined in the Example 3.3. Let us denote:



By applying the algorithm we can build the following graph and endow the nodes with the quality ACCEPT/REJECT (A/R).



Here, we draw with a square the OR nodes and with a circle the AND nodes, and with a double circle the FINAL nodes. The "non-deterministic" edges outgoing from the OR nodes, are dashed. The initial nodes are denoted by an incoming arrow.

It can be easily seen that the tree in Example 3.3 corresponds to an "accepting path" in the graph G, which means that all the next nodes of the AND vertices are taken into consideration and by choosing an accepting next node for each OR vertex. Note also that this tree is minimal T-ambiguous, since it corresponds to a minimal "accepting path".

Remark 7.1. By taking the global size of the set $T = \{\tau_1, \ldots, \tau_n\}$ as parameter, that is

$$t = \sum_{i=1}^{n} |\tau_i|,$$

one can see that the graph has at most t^2 nodes and $n^2 + kt^2$ edges (that is $o(t^2)$ edges). The assignment of the qualities is equivalent to an exploration of the graph, that is, deciding the code problem on a set T takes polynomial time in the global size of T.

8. Equations on trees

In the last part of this paper we introduce the notion of tree equations as complementary to the one of tree codes. It is well known that, if a set of words is not a code, then it satisfies some word equation without constant. Our generalization to trees of the notion of "equation", is, in the same sense, complementary to the notion of tree code introduced by Nivat, that is, if a set of trees is not a tree code, then it is a solution of some tree equation.

We first give some additional definition needed to define a tree equation and its solution. In particular, we introduce the notion of graded tree (that generalizes the notion of k-ary tree) and the notion of morphism between two sets of graded trees. Then we give the definition of tree equation and propose some related decidability and algorithmic problem.

8.1. Graded trees and morphisms

We introduce here a different notion of tree that generalizes the notion of k-ary tree. In this notion of *graded tree* the arity of each node is a function of the label contained in the given node. The introduction of this kind of trees is essential to give a general definition of tree equation. In the formal definition that follows we consider $\mathbb{N}$ (the set of non-negative integers) as a general infinitely numerable set, to denote the fact that the arity of each node can be arbitrarily large. We first define a graded alphabet:

Definition 8.1. A *graded alphabet* is an alphabet A endowed with an arity function $\alpha : A \to \mathbb{N}$ assigning a non-negative integer to each element in the alphabet.

Definition 8.2. Given a graded alphabet (A, α) we define a labeled tree over A with arity function α a partial mapping

$$\tau : \mathbb{N}^* \to A,$$

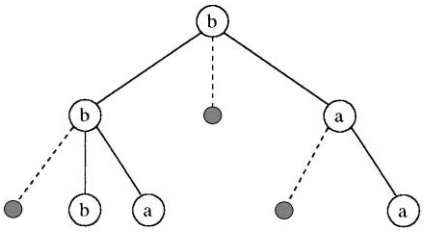
where the domain $dom(\tau)$ is a finite and prefix closed subset of $\mathbb{N}^*$ and, for all $v \in dom(\tau)$ and $i \in \mathbb{N}$, if $v \cdot i \in dom(\tau)$, then $i \leqslant \alpha(\tau(v))$. We say that τ is an α-*ary tree* over A.

The notions of *node*, *sons*, *father*, *root*, *leaf*, etc., are extended to α-ary trees in a trivial way. The outer frontier for α-ary trees is defined as the set

$$fr^+(\tau) = \{u \cdot i \mid u \in dom(\tau),\ i \leqslant \alpha(\tau(u)),\ u \cdot i \notin dom(\tau)\}.$$

We will denote by $(A, \alpha)^{\#}$ the set of all labeled trees over a graded alphabet (A, α).

Example 8.1. If $A = \{a, b\}$ and $\alpha(a) = 2$, $\alpha(b) = 3$, then the tree



is an α-ary tree over A, whose domain is the set $\{\varepsilon, 1, 3, 12, 13, 32\}$ and its outer frontier is the set $fr^+(\tau) = \{2, 11, 31, 121, 122, 123, 131, 132, 321, 322\}$.

If we are given a graded alphabet (A, α) where the arity function is a constant function, that is, $\alpha(a) = k$, $\forall a \in A$, then we have the traditional notion of k-ary tree.

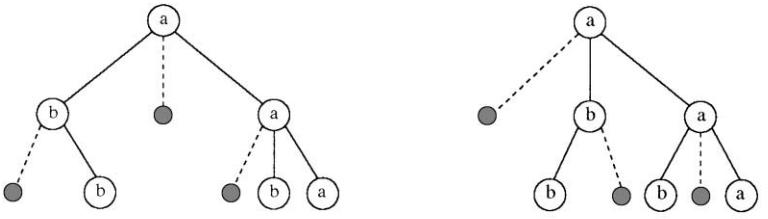
Definition 8.3. Given a finite alphabet A, an *ordered labeled tree* over A is a tree in which the sons of each node are ordered.

When we draw an ordered tree, we assume that the sons of each vertex are ordered from left to right. We will denote by $\mathcal{O}(A)$ the set of all ordered labeled trees over A. By convention, we will denote ordered trees by Latin letters, while α-ary (k-ary) trees will be denoted by Greek letters.

Note that an α-ary (k-ary) tree can be seen as a particular ordered tree in which the sons of each node are distinguished and they are ordered by means of their lexicographic order. Then, if (A, α) is a graded alphabet, the application is naturally defined as

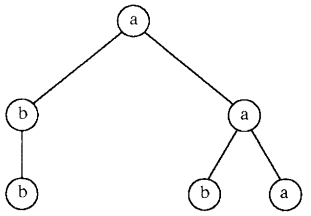
$$\omega : (A, \alpha)^{\#} \to \mathcal{O}(A)$$

that associates with an α-ary tree the corresponding ordered tree. We say that $\omega(\tau)$ is the *ordered tree associated* with τ. Note that this application is not injective. In fact, for instance, both trees τ_1 and τ_2 over (A,α) with $A=\{a,b\}$ and $\alpha(a)=3$ and $\alpha(b)=2$, drawn in the figure below



τ_1 τ_2

have the same ordered tree as image



Definition 8.4. Given two graded alphabets $\mathscr{A}=(A,\alpha)$ and $\mathscr{B}=(B,\beta)$, a *morphism* of $\mathscr{A}^{\#}$ into $\mathscr{B}^{\#}$ is an application

$$\varphi:\mathscr{A}^{\#}\to\mathscr{B}^{\#}$$

such that

1. φ preserves the cardinality of the outer frontier of trees, that is for any $a\in A$, $|fr^{+}(\varphi(a))|=\alpha(a)$;
2. φ preserves concatenation, i.e., for any tree τ with $|fr^{+}(\tau)|=n$ and for any sequence of n trees $\tau_1,\tau_2,\ldots,\tau_n$

$$\varphi(\tau\langle\tau_1,\tau_2,\ldots,\tau_n\rangle)=\varphi(\tau)\langle\varphi(\tau_1),\varphi(\tau_2),\ldots,\varphi(\tau_n)\rangle$$

A morphism $\varphi:\mathscr{A}^{\#}\to\mathscr{B}^{\#}$ is said to be *non-erasing* if the only tree having as image the empty tree (in $\mathscr{B}^{\#}$), is the empty tree in $\mathscr{A}^{\#}$. Since concatenation is preserved by morphisms, a morphism $\varphi:(A,\alpha)^{\#}\to(B,\beta)^{\#}$, where $A=\{a_1,\ldots,a_k\}$, can be uniquely determined by the set of trees $T=\{\varphi(a_1),\ldots,\varphi(a_k)\}\subseteq\mathscr{B}^{\#}$. Then $\varphi(\mathscr{A}^{\#})=T^{\#}$ that is, any tree $\tau\in\varphi(\mathscr{A}^{\#})$ is obtained as a concatenation of trees in T.

We denote by (B,k) the graded alphabet whose arity function is the constant function $k(b)=k$ for all $b\in B$. Clearly (B,k) is the alphabet defining k-ary trees over the alphabet B.

Remark 8.1. Note that, the condition $|fr^+(\varphi(a))| = \alpha(a)$ implies that it is possible to have morphisms $\varphi : (A,\alpha)^{\#} \to (B,k)^{\#}$ only if $\alpha(a) \geqslant k$ for all $a \in A$.

Remark 8.2. Let us consider non-erasing morphisms $\varphi : (A,\alpha)^{\#} \to (B,k)^{\#}$, with $k > 1$ (and then, by previous remark also $\alpha(a) > 1$ for all $a \in A$).

Recall that the function $f_k : \mathbb{N} \to \mathbb{N}$ defined as $f_k(n) = (k-1)n+1$ gives the relation between the size n of a k-ary tree and the size of its outer frontier $f_k(n)$, as proved by induction. Note that $f_k(n)$ is totally invertible if and only if $k = 2$.

Note also that in a non-erasing morphism $\varphi : (A,\alpha)^{\#} \to (B,k)^{\#}$, the arity of each $a \in A$ gives at the same time the cardinality of the outer frontier of $\varphi(a)$, and then gives information about the size of the tree $\varphi(a)$ for that noticed above. This fact suggests some remarks:

1. In the definition of morphisms, the need for dealing with α-ary (and not k-ary) trees, as regards the domain of the morphism, comes from the requirement that two different elements of A, can have as image trees with different sizes.
2. Once we fix the alphabets (A,α) and (B,k), there exists only a finite number of non-erasing morphisms $\varphi : (A,\alpha)^{\#} \to (B,k)^{\#}$.
3. There exist pairs of alphabets (A,α) and (B,k) such that there is no non-erasing morphism $\varphi : (A,\alpha)^{\#} \to (B,k)^{\#}$.
4. For all graded alphabets (A,α) and for all alphabets $(B,2)$ (that is, alphabets defining binary trees) there exist always non-erasing morphisms $\varphi : (A,\alpha)^{\#} \to (B,2)^{\#}$.

8.2. Equations on trees

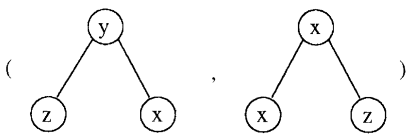
We are now ready to give the definition of tree equation:

Definition 8.5. Let $X = \{x_1,\ldots,x_m\}$ be a finite alphabet. A *tree equation* is a pair of ordered trees $t_1, t_2 \in \mathcal{O}(X)$. We denote it by (t_1,t_2) and we call $\{x_1,\ldots,x_m\}$ the set of indeterminates. A *k-ary solution* over the alphabet A to a tree equation (t_1,t_2) consists of:

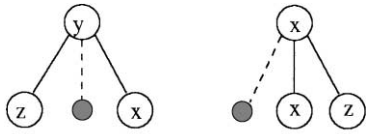
1. an arity function $\chi : X \to \mathbb{N}$ assigning to each symbol x in X an arity $\chi(x)$;
2. a pair of χ-ary trees τ_1 and τ_2 whose associated ordered trees are, respectively, t_1 and t_2;
3. a tree morphism $\varphi : (X,\chi)^{\#} \to (A,k)^{\#}$ such that $\varphi(\tau_1) = \varphi(\tau_2)$.

Usually we will refer to the multiset $\varphi(X) = \{\varphi(x_1),\ldots,\varphi(x_m)\}$ as *a k-ary solution* of the tree equation (t_1,t_2).

Example 8.2. Consider the tree equation (t_1,t_2) in the set of indeterminates $\{x,y,z\}$, expressed in the form

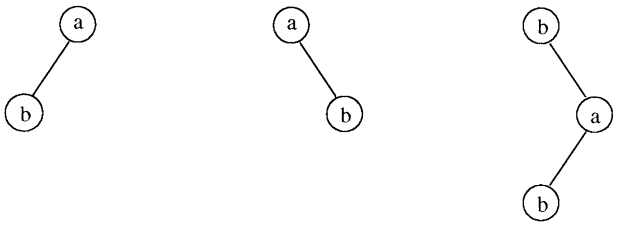


1. Consider the arity function χ over the set of variables $\{x, y, z\}$ defined as follows: $\chi(x) = \chi(y) = 3$ and $\chi(z) = 4$.
2. Consider the following pair of χ-ary trees τ_1 and τ_2 over X having as associated ordered trees t_1 and t_2, respectively,



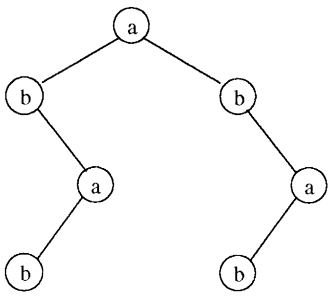
τ_1 τ_2

3. Finally consider the morphism $\varphi : (X, \chi)^{\#} \to (A, 2)^{\#}$ defined as follows on the elements of X:



$\varphi(x)$ $\varphi(y)$ $\varphi(z)$

This gives a solution to the tree equation (t_1, t_2). Indeed, one can easily verify that $\tau = \varphi(\tau_1) = \varphi(\tau_2)$ is the following binary tree:



τ

Remark 8.3. As mentioned at the beginning of this section, the notion of tree equation is indeed in a sort of a dual correspondence with the notion of tree code. In fact, given a set of trees $T \subset A^{\#}$, if T is not a tree code, then there is a tree τ having a double T-factorization, and then we can associate a variable x_i with each element τ_i of T and consider the pair of ordered trees over the variable alphabet $\{x_i\}_{i \in I}$ corresponding

to the factorizations of τ. Trivially this is a tree-equation having T as solution. For instance, the set of trees $T = \{\varphi(x), \varphi(y), \varphi(z)\}$ in Example 8.2 is not a tree code, but is in fact a solution of a tree equation.

From the definition, it follows that, in order to find a solution of a tree equation, one has to solve the following three steps:

Step 1: Find an arity function $\chi : X \to \mathbb{N}$ assigning to each symbol x in X an arity $\chi(x)$.

Step 2: Find a pair of χ-ary trees τ_1 and τ_2 whose associated ordered trees are respectively t_1 and t_2.

Step 3: Find a tree morphism $\varphi : (X, \chi)^{\#} \to (A, k)^{\#}$ such that $\varphi(\tau_1) = \varphi(\tau_2)$.

Open Problem. Given a tree equation (t_1, t_2) decide if it admits a k-ary solution, for a given k.

Remark 8.4. The special case $k = 1$ of the open problem corresponds to word equations. Indeed, in this case, by Remark 8.1 there exists a solution only if t_1 and t_2 are linear trees, i.e. they are words. The solvability of word equations has been proved by Makanin [19] (cf. also [13]).

Remark 8.5. By item 4 of Remark 8.2 one can derive that, if there exists a k-ary solution to a tree equation for $k > 2$, then there exists a solution for $k = 2$. So, in order to decide the existence of a $k \geqslant 2$ such that (t_1, t_2) has a k-ary solution, it suffices to decide the existence of a binary solution.

Note that, for $k \geqslant 2$, once Step 1 is solved (that is, the arity function over the variables is given), the problem of solvability of a tree equation becomes trivially decidable. In fact, as remarked before, when we know the arity of a variable, we can retrieve the size of its image by φ. In this case we have to test only a finite number of cases, since there exists only a finite number of pairs χ-ary trees that correspond, as ordered trees, to the pair of trees in the equation, and a finite number of morphisms between $(X, \chi)^{\#}$ and $(A, k)^{\#}$. Then for Steps 2 and 3, we are only faced with the problem of designing efficient algorithms to solve the respective problems. In particular, a linear algorithm solving Step 3 (that is the problem of solving "graded" tree equations) is given in [20], where the Martelli–Montanari algorithm for the first-order unification is generalized to the case where the variables are in all the nodes of the tree and not only in the leaves.

So the difficulty of the solvability of tree equations is in Step 1. In fact, since the arity of variables is unknown, we are not able to upper bound the size of the trees in the solution. This also explains why we need to define a tree equation in terms of ordered trees.

Actually our open problem generalizes at the same time both first-order unification and word unification problems and is a particular case of second-order unification.

The latter problem is in general undecidable (cf. [11]). However, we do not know if second-order unification becomes decidable for this particular subclass of equations. In fact, the proof of the undecidability of second-order unification given in [11] cannot be extended directly to our case. This shows the relevance of the solvability problem of tree equations.

The open problem can be generalized to the one of finding a solution to more general sets of graded trees $(A,\alpha)^{\#}$, as well as to equations with constants. As to the search of solutions over graded alphabets (A,α), previous remarks apply to the cases where either $\alpha(a)=1$ for all $a\in A$, or $\alpha(a)>1$ for all $a\in A$. But in the general case we do not know the procedure even to solve Step 3 because it is not clear whether it can be obtained as a combination of Makanin's algorithm and algorithm of [20]. This gives rise to a new open problem.

Acknowledgements

We wish to thank the authors of [14] for pointing out that our original definition of stability was not correct and providing the correct one.

References

[1] J. Berstel, D. Perrin, Theory of Codes, Academic Press, New York, 1985.
[2] J. Berstel, D. Perrin, J.F. Perrot, A. Restivo, Sur le theoreme des defaut, J. Algebra 60 (1979) 169–180.
[3] C. Choffrut, J. Karhumäki, Combinatorics on words, in: G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Languages, Vol. 3, Springer, Berlin, 1997.
[4] M. Chrobak, W. Rytter, Unique decipherability for partially commutative alphabets, Fund. Inform. X (1987) 323–336.
[5] S. Eilenberg, Automata, Languages and Machines, vol. A, Academic Press, New York, 1974.
[6] C.C. Elgot, S.L. Bloom, R. Tindell, On the algebraic structure of rooted trees, J. Comput. System Sci. 16 (1978) 362–399.
[7] F. Gécseg, M. Steinby, Tree languages, in: G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Languages, Vol. 3, Springer, Berlin, 1997.
[8] D. Giammarresi, S. Mantaci, F. Mignosi, A. Restivo, Congruence, Automata and Periodicities, in: J. Almeida, G.M.S. Gomes, P.V. Silva (Eds.), Proc. Semigroups, Automata and Languages, Portugal, 1994, (World Scientific Publishing, Singapore, 1996, pp. 125–135.
[9] D. Giammarresi, S. Mantaci, F. Mignosi, A. Restivo, A periodicity theorem for trees, Proc. 13th. World Computer Congress – IFIP'94, Vol. A-51, Elsevier Science B.V., North-Holland, Amsterdam, 1994, pp. 473–478.
[10] D. Giammarresi, S. Mantaci, F. Mignosi, A. Restivo, Periodicities on trees, Theoret. Comput. Sci. 205 (1-2) (1998) 145–181.
[11] W. Goldfarb, The undecidability of the second-order unification problem, Theoret. Comput. Sci. 13 (1981) 225–230.
[12] H.J. Hoogeboom, A. Muscholl, The code problem for traces – improving the boundaries, Theoret. Comput. Sci. 172 (1997) 309–321.
[13] J. Jaffar, Minimal and complete word unification, J. ACM 37 (1990) 47–85.
[14] J. Karhumäki, S. Mantaci, Defect theorems for trees, Fund. Inform. 38 (1999) 119–133.
[15] J. Karhumäki, S. Mantaci, Defect theorems for trees, (Conference Version), Proc. of Developments in Language Theory (1999), to appear.

[16] A. Lentin, M.P. Schützenberger, A combinatorial problem in the theory of free monoids, Proc. University of North Carolina, 1967, pp. 67–85.
[17] M. Lothaire, Combinatorics on Words, Addison-Wesley, Reading, MA, 1983.
[18] R.C. Lyndon, M.P. Schützenberger, The equation $a^m = b^n c^p$ in a free group, Michigan Math. J. 9 (1962) 289–298.
[19] G.S. Makanin, The problem of solvability of equations in a free semigroup, Math. USSR Sbornik 32 (1977) 129–198, (in AMS, 1979).
[20] S. Mantaci, D. Micciancio, An algorithm for the solution of tree equations, Proc. CAAP-TAPSOFT'97, Lecture Notes in Computer Science, Vol. 1214, 1997, pp. 417–428.
[21] S. Mantaci, A. Restivo, Equations on trees, Proc. MFCS'96, Lecture Notes in Computer Science, Vol. 1113, 1996, pp. 443–456.
[22] S. Mantaci, A. Restivo, Tree codes and equations, Proc. 3rd Internat. Conf. Developments in Language Theory, 1997, pp. 119–133.
[23] S. Mantaci, A. Restivo, On the defect theorem for trees, Proc. "8th Internat. Conf. Automata and Formal Languages", Salgótarján, Ungheria, Pub. Math, Debrecen 54-supplement (1999) 923–932.
[24] Y. Matyiasevitch, Mots et Codes. Cas décidables et indécidables di problème du codage pour les monoïdes partialment commutatifs, Quadrature. Mag. Math. Pure App. 27 (1997) 23–33.
[25] M. Nivat, Binary tree codes, Tree Automata and Languages, Elsevier, Amsterdam, 1992, pp. 1–19.
[26] L. Oget, A least common divisor for trees, Internal Report.
[27] J. Tatcher, Tree automata, an informal survey, in: A. Aho (Ed.), Currents in the Theory of Computing, Pretince-Hall, Englewood Cliffs, NJ, 1973.
[28] W. Thomas, Logical aspects in the theory of tree languages, in: B. Courcelle (Ed.), Proc. 9 colloquium on Trees in Algebra and Programming, Cambridge University Press, Cambridge, 1984.