Periodicities on Trees *

Dora Giammarresi

Dipartimento di Matematica Applicata e Informatica, Università "Cá Foscari" di Venezia.
via Torino, 155 - 30173 Venezia Mestre - ITALY (dora@dsi.unive.it)

Sabrina Mantaci   Filippo Mignosi   Antonio Restivo

Dipartimento di Matematica ed Applicazioni, Università di Palermo
via Archirafi, 34 - 90123 Palermo - ITALY
({sabrina,mignosi,restivo}@altair.math.unipa.it)

February 12, 1997

Abstract

We introduce the notion of periodicity for k-ary labeled trees: roughly speaking, a tree is periodic if it can be obtained by a sequence of concatenations of a smaller tree plus a "remainder". The period is the shape of such smaller tree (i.e. the corresponding unlabeled tree). This definition reduces to the classical one for string when restricted to the case of unary trees.

Then, we define the greatest common divisor of two unlabeled trees and relate right congruences to unlabeled trees. This allows us to give a characterization of tree periodicity in terms of right congruences and then to prove a periodicity theorem for trees that is a generalization to trees of the Fine and Wilf's periodicity theorem for words.

Keywords: Congruence, periodicity, labeled tree.

*Work partially supported by the ESPRIT II Basic Research Actions Program of the EC under Project ASMICS 2 (contract No. 6317) and in part by the Italian Ministry of Universities and Scientific Research MURST 40% *Algoritmi, Modelli di Calcolo, Strutture Informative.*

1 Introduction

In this paper we introduce the notion of periodicity for k-ary labeled trees. We look at a k-ary labeled tree as a generalization of a word, in the sense that words corresponds to the particular case of $k = 1$, i. e. to unary trees.

Our work can be considered as a part of a more general research program having the aim to extend concepts, methods and results of combinatorics on words to labeled trees (cf. [13, 14, 15]).

The study of periodicity plays an important role in combinatorics on words and presents some interesting applications in algebra, in formal languages and in the design of string searching algorithm (cf. [8, 12]). A central result in this theory is the Fine and Wilf periodicity theorem (cf. [4, 12]). This result and the notion of periodicity have been extended to two-dimensions, motivated by problems in pattern matching algorithms (cf.[7]). We feel confident that some interesting applications will come out also from the study of periodicity on trees.

We recall that a word w has period p if there exists a word u of length p such that w is prefix of a word in u^*. This means, roughly speaking, that w can be factorized as a repeated concatenation of the word u by itself, plus a "remainder", that is a prefix of the word u. Generalizing this notion to trees, we define the period of a labeled tree τ by means of a "factorization" of τ in terms of a smaller tree τ_0, plus a "remainder", that is a set of prefixes of τ_0. The period of τ is the shape of τ_0, i.e. the unlabeled tree corresponding to τ_0. This definition reduces to the classical one for words when restricted to the special case of unary trees.

The main result of the paper is a periodicity theorem for trees, that generalizes the Fine and Wilf theorem. In order to state this result we introduce the notion of greatest common divisor of two unlabeled trees and we prove its existence and unicity. We then relate unlabeled trees to right congruences on a free monoid, and we show that the greatest common divisor of two unlabeled trees corresponds to the join of the related congruences.

This allow us to characterize tree periodicity in terms of right congruences and then to prove the periodicity theorem by using algebraic arguments concerning the join and the restrictions of the right congruences.

The paper is organized as follows. We start in Section 2 by recalling some definitions on words and periodicities on words. Then, in Section 3, we focus our attention on labeled k-ary trees by introducing the basic definitions and all the related notations. Moreover we define some basic operations on trees (union, intersection, difference, concatenation and power) and give several examples. In particular, the power of a tree is used in Section 4 to give the definition of periodicity for trees.

Then, the remaining part of the paper is devoted to state and to prove the periodicity theorem for trees. In Section 5, we define the greatest common divisor of unlabeled trees and give an algorithm to compute it. In Section 6 we associate particular right congruences to unlabeled trees and prove some lemmas to relate the join of two right congruences and the greatest common divisor of the trees associated to such congruences. An algorithm to calculate the join of two right congruences is given in Section 7. Moreover we give the definition of set complete with respect to two given congruences together with an algorithm to decide whether a set is complete. Finally, in Section 8, we give a characterization of periodicity on trees in terms of right congruences and prove the main theorem.

2 Periodicities on words

We start by recalling some definitions of words, languages and prefixes that will be useful for a description of trees given in next sections. We also give the definition of periodicity for words and one of its characterizations that will be useful in the sequel. Moreover we state the Fine and Wilf's Theorem on periodicity for words. For the basic terminology and for the notion of periodicity on words we will refer to [12].

Given an arbitrary finite alphabet A, i.e. a finite set of symbols, the *free monoid* over A, denoted by A^*, is the set of all possible concatenations of symbols in A. The elements of the free monoid A^* are called *words*. The natural operation on the free monoid is concatenation of words. We will denote by ϵ the *empty word*, that is the identity in the free monoid A^* with respect to the concatenation of words. If w is a word over the alphabet A, we will denote by $|w|$ the lenght of w.

Given a word $w \in A^*$, a *prefix* of w is a word $u \in A^*$ such that $w = uv$ for some $v \in A^*$. A subset P of A^* is *prefix-closed* (or *closed by left factor*) if for any $w \in P$, every prefix of w belongs also to P.

We recall (cf. [2]) that a subset C of Σ^* is a *prefix code* if no word in C is prefix of another word in C. A prefix code is *maximal* if it is not a proper subset of another prefix code over the same alphabet.

We now recall the definition of periodicity on words.

Definition 2.1 *Let A be a finite alphabet, let $w \in A^*$ and let $p \in I\!N$. We say that w* has period p *if there exists a word $u \in A$ with $|u| = p$ such that w is a prefix of a word in u^*.*

Example 2.1 The word

$$w = \underbrace{a\,b\,b\,a}\underbrace{a\,b\,b\,a}\underbrace{a\,b\quad}$$

has period 4. In fact w is a prefix of a word in u^*, where $u = a\,b\,b\,a$.

We now give an algebraic charaterization of periodicity on words, that will be useful in the study of periodicity for trees. We recall first that, given three integers n, m and p, we say that n is equivalent to m modulo p, and we write $n \equiv m \pmod p$, if $n - m = kp$ for some $k \in I\!N$.

A word $w \in A^*$ of length n can always be represented as a map $w : I\!N \to A$ whose domain is the set $\{0, \ldots, n-1\}$ and such that $w(i)$ is the $(i+1)$-th letter in w. In fact it is convenient to consider the first letter of a word as the one corresponding to the position 0 of the domain. This allows us to give the following characterization of the notion of periodic words.

Proposition 2.1 *A word w* has period $p \in \mathbb{N}$ *if and only if for any pair of natural numbers* $0 \leq i, j \leq |w| - 1$ *such that* $i \equiv j \pmod p$, *we have that* $w(i) = w(j)$.

We can see from Example 2.1 that this definition is perfectly equivalent to the definition 2.1. In fact, if we consider the word w as an application from the set of natural numbers to the alphabet $\{a, b\}$, elements in the domain of w that are equivalent modulo 4, have the same image by the map w.

The periodicity theorem of Fine and Wilf considers the case in which a word has two different periods. We give here the statement:

Theorem 2.1 [Fine and Wilf's Theorem] *Let p_1 and p_2 be two positive integers. Let $w \in A^*$ be a word with periods p_1 and p_2. If $|w| \geq p_1 + p_2 - gcd(p_1, p_2)$ then w has also period $p = gcd(p_1, p_2)$.*

It is easy to prove that this bound is tight. It suffices to exibit two integers p and q and a word w with $|w| = p + q - gcd(p, q) - 1$ with periods p and q but not period $gcd(p, q)$. For example, we can take the Fibonacci's word of length 11:

$$a\,b\,a\,a\,b\,a\,b\,a\,a\,b\,a$$

This word has periods 5 and 8 but not period $1 = gcd(5, 8)$.

3 Trees

In this section we recall the notion of k-ary labeled tree. Most of the notations and the definitions here introduced are from Nivat [14]. We remark that the classical recursive definition of k-ary tree (cf. [11]) is equivalent to the one we use.

Moreover we define some operations on trees that will allow us to study the notion of periodicity on trees.

3.1 Basic definitions

Definition 3.1 *Let $\Sigma = \{1, \ldots, k\}$ and let A be a finite alphabet. A* k-ary (labeled) tree *over A is a map $\tau : \Sigma^* \to A$ whose domain $dom(\tau)$ is a finite and prefix-closed subset of Σ^*. The elements in $dom(\tau)$ are called* nodes. *We say that τ is an* unlabeled tree *if the alphabet A contains only one element (i.e. all nodes have the same label).*

Example 3.1 Let $\Sigma = \{1, 2\}$ and $A = \{a, b\}$. Consider the map $\tau : \Sigma^* \to A$ given by:

$\tau(\epsilon) = a$	$\tau(12) = b$	$\tau(121) = a$
$\tau(1) = b$	$\tau(21) = b$	$\tau(122) = a$
$\tau(2) = a$	$\tau(22) = a$	$\tau(221) = b$

This map is defined over the prefix-closed set $\{\epsilon, 1, 2, 12, 21, 22, 121, 122, 221\}$. Then τ it is a (labeled) binary tree.

In this paper, when we deal with k-ary trees, we will consider the alphabet $\Sigma = \{1, \ldots, k\}$ and we will refer to it as the *structure-alphabet*. We will refer to the set A as the *label-alphabet*. Moreover, we will denote labeled trees by Greek letters, whereas unlabeled trees will be denoted by Latin letters.

Notice that an unlabeled tree can be univocally individuated by its domain. Therefore we can always associate to any labeled tree τ the unique unlabeled tree t having the same domain. We call t the *shape* of τ and we denote it by $sh(\tau)$. Moreover we will denote by $|\tau|$ the *size* of τ, that is, the number of its nodes.

Let τ be a tree over A. If $x, y \in dom(\tau)$ are nodes of τ such that $x = yi$ for some $i \in \Sigma$, we say that y is the *father* of x and that x is the i-th *son* of y. A node without sons is called *leaf*. The node ϵ (i.e. the only node without father) is called *the root* of the tree.

We now give a definition that will be very useful in the sequel.

Definition 3.2 *Given a tree τ, the* border *of τ is the set $B(\tau) = \{xi \,|\, x \in dom(\tau), xi \notin dom(\tau)\}$.*

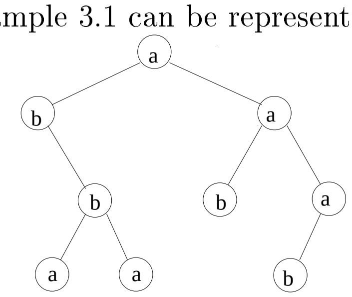
That is, the border of a k-ary tree τ is the set of nodes that are missing to make τ a complete k-ary tree plus the nodes corresponding to all children of the leaves of τ. The following lemma, whose proof is straightforward, holds.

Lemma 3.1 *The border $B(\tau)$ of a tree τ is a maximal prefix code.*

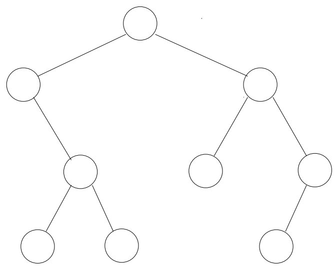
Remark that the definition of k-ary tree we gave above is completely equivalent to the classical notion of k-ary tree in terms of graphs. In fact we can represent a k-ary tree as a directed graph where the root is the only node without incoming edges, each node has out-degree k and the sons of each node are distinguished by numbers $1, 2, \ldots, k$ put as labels on the edges leaving that node (usually not explicitely written). Then, each node $v \in \Sigma^*$ in the tree corresponds to the vertex reached by the path from the root labeled v.

The following example is clarifying.

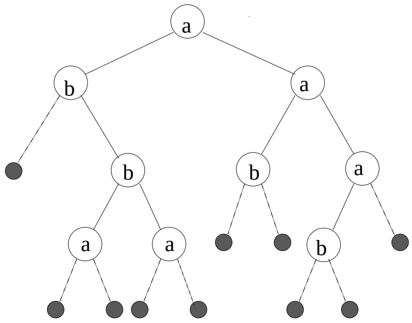
Example 3.2 The tree τ in Example 3.1 can be represented in the following way:



This tree has size $|\tau| = 9$. The shape of τ is represented by:



In this tree the set of leaves is given by $\{121, 122, 21, 221\}$ and the border is $B(\tau) = \{11, 1211, 1212, 1221, 1222, 211, 212, 222, 2211, 2212\}$. When we are interested in giving evidence to the border of a tree, we will draw it by means of dotted edges and dummy black nodes as in the figure that follows.



We will denote by $A_k^{\#}$ the set of all finite k-ary trees over A. When k is implicit we will simply write $A^{\#}$. A particular element of $A^{\#}$ is the tree of size 0, whose domain is the empty set, that is called the *empty tree*, and is denoted by Ω. The trees of size 1, whose domain is ϵ, are called the *punctual trees*. The punctual tree whose only node is labeled a will be simply denoted by a. The (unique) punctual unlabeled tree will be denoted by π.

Let τ and τ' be two k-ary trees. Then τ is a *subtree* of τ' if there exists a node $v \in dom(\tau')$ such that:

i) $v \cdot dom(\tau) = \{vu \mid u \in dom(\tau)\} \subseteq dom(\tau')$,

ii) $\tau(u) = \tau'(vu)$ for all $u \in dom(\tau)$.

In this case we say that τ is a subtree of τ' *rooted at* node v.

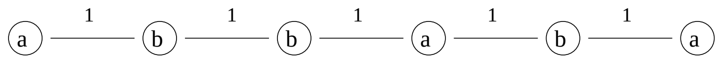
We remark that the definition of subtree we gave above is different from the usual one (it corresponds, instead, to a connected subgraph of a tree). Notice that, if $v = \epsilon$, then $S(\epsilon, \tau) = dom(\tau) \subseteq dom(\tau')$ and τ coincides with the restriction of τ' to $dom(\tau)$. In this case we write $\tau \subseteq \tau'$ and we say that τ is an *initial subtree* or a *prefix* of τ'. If $S(v, \tau) = v\Sigma^* \cap dom(\tau')$, then we say that τ is a *terminal subtree* or a *suffix* of τ'.

Remark 3.1 The definition of k-ary labeled trees, when restricted to the case $k = 1$, is consistent with the definition of words. In fact a word $w \in A^*$ can be described as a partial mapping from the set of natural numbers $I\!N$ (that is isomorphic to Σ^* when $\Sigma = \{1\}$) to the alphabet A: the domain of w is the set $\{0, 1, \ldots, |w| - 1\}$ that is isomorphic in Σ^* to the set $\{\epsilon, 1, 1^2, \ldots, 1^{|w|-1}\}$, that is a subset of $\{1\}^*$ closed under prefix. Then we can always consider a word as a labeled 1-ary tree over A. Moreover notice that, in the case of words, the border contains only one element $1^{|w|}$.

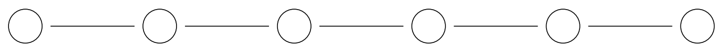
As example consider the word: $w = a\,b\,b\,a\,b\,a$. It can be seen as the unary tree defined by the map:

$w(\epsilon) = a$	$w(11) = b$	$w(1111) = b$
$w(1) = b$	$w(111) = a$	$w(11111) = a$

Then, w can be described by the following graph:



The shape of w is the following unlabeled 1-ary tree:



and the number of its nodes is actually the length of the word w. This justifies the choice of the notation $|\tau|$ to denote the size of a tree τ. Notice that a word has only one leaf and that its border contains only one element $B(w) = \{111111\}$.

In the sequel, to define operations among trees, we will use an intermediate object called *bush*. We recall that a subset S of Σ^* is *interval closed* if v and vab in S implies that va is also in S, for all $v \in \Sigma^*$ and $a, b \in \Sigma$.

Definition 3.3 *Let $\Sigma = \{1, \ldots, k\}$ and let A be a finite alphabet. A* k-ary (labeled) bush *over A is a partial mapping $\beta : \Sigma^* \to A$ whose domain, $dom(\beta)$, is a finite, interval closed subset of Σ^*. If the alphabet A contains just one element, we say that β is an* unlabeled bush.

As example consider the following.

Example 3.3 Let $\Sigma = \{1, 2\}$ and $A = \{a, b\}$. Then the map:

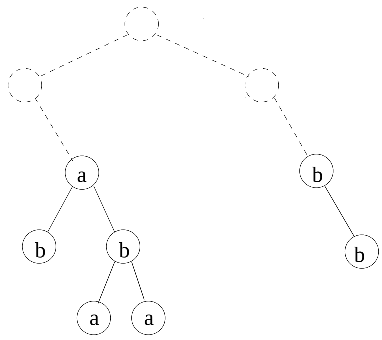
$\beta(12) = a$	$\beta(121) = a$	$\beta(122) = b$	$\beta(1221) = a$
$\beta(22) = b$	$\beta(222) = b$	$\beta(1222) = a$	

represents a bush whose domain is the set $dom(\beta) = \{12, 22, 121, 122, 222, 1221, 1222\}$.

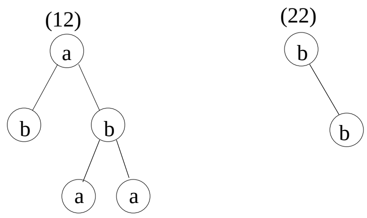
All the terminologies given for trees (node, father, son, leaf, shape, etc.) can be trasposed to bushes. *A root* of a bush is a node without father. Notice that a bush can have more than one root.

In a sense, a bush is a set of "tree-pieces". We can describe a bush as in the following example.

Example 3.4 The bush in Example 3.3 can be represented as:



or, equivalentely, as the following set of subtrees:



where we put an extra label on the roots to indicate the corresponding words in Σ^*.

A bush is *connected* if its domain contains only one root. This implies that a connected bush can be described by a connected graph. Given a bush β, a *maximal connected component* of β is a maximal connected bush β' such that $\beta' \subseteq \beta$, where "maximal" is referred to the number of nodes. The bush in Example 3.4 is not a connected bush since it has two maximal connected components. We remark that every bush can be partitioned in a unique way as a family of connected components (this is because connectivity in graphs is an equivalence relation).

Given two words $x, y \in \Sigma^*$ we denote by:

$$x^{-1}y = \begin{cases} y' & \text{if } y = xy' \\ undefined & \text{if } y \notin x\Sigma^* \end{cases}$$

We now define the translated of a bush by x^{-1} and by x, respectively. Let β be a bush and x be a word in Σ^*. The *translated* of β by x^{-1} is the bush $x^{-1}\beta$ defined as follows.

i) $dom(x^{-1}\beta) = x^{-1} \cdot dom(\beta) = \{x^{-1}y \mid y \in dom(\beta) \cap x\Sigma^*\}$

ii) $\forall y \in dom(x^{-1}\beta),\ x^{-1}\beta(y) = \beta(xy)$

The *translated* of β by x is the bush $x\beta$ defined as follows.

i) $dom(x\beta) = x \cdot dom(\beta) = \{xy | y \in dom(\beta)\}$

ii) $\forall y \in dom(x\beta),\ x\beta(y) = \beta(x^{-1}y)$

3.2 Basic operations

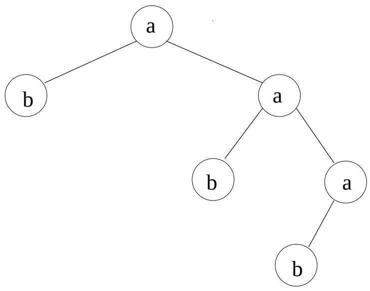
We now introduce operations among trees. We will implicitly assume that all the trees are labeled k-ary trees with the same k and that are defined over the same alphabet A. Moreover, some of the operations we define will require that the trees involved satisfy special conditions. We give first the following definition.

Let τ_1 and τ_2 be two labeled trees. We say that τ_1 and τ_2 are *compatible* if and only if they coincide as functions on the intersection of their domains. In formula, if $\tau_{|D}$ denotes the restriction of τ to $D \subseteq dom(\tau)$, we write:

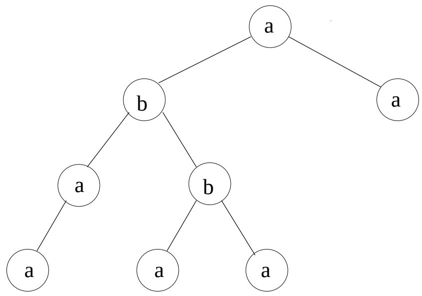
$$\tau_{1|(dom(\tau_1)\cap dom(\tau_2))} = \tau_{2|(dom(\tau_1)\cap dom(\tau_2))}.$$

We remark that two unlabeled trees are always compatible.

Example 3.5 The two trees τ_1 and τ_2 defined by the two graphs below, are compatible.



τ_1



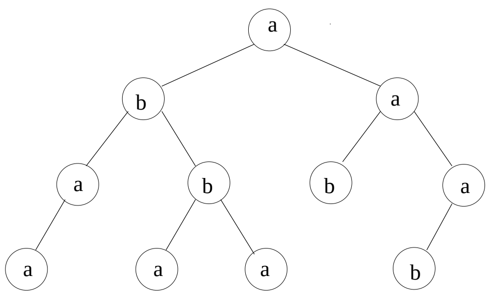
τ_2

We can now define several operations among trees. Let τ_1 and τ_2 be two compatible trees. The *union* of τ_1 and τ_2 is the tree $\tau_1 \oplus \tau_2$ defined by:

i) $dom(\tau_1 \oplus \tau_2) = dom(\tau_1) \cup dom(\tau_2)$;

ii) $\forall x \in dom(\tau_1 \oplus \tau_2),\ (\tau_1 \oplus \tau_2)(x) = \begin{cases} \tau_1(x) & \text{if } x \in dom(\tau_1) \\ \tau_2(x) & \text{otherwise} \end{cases}$

For instance, the union of the trees τ_1 and τ_2 in Example 3.5 is the tree $\tau_1 \oplus \tau_2$ described by the graph below.



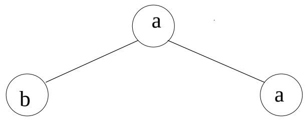
$\tau_1 \oplus \tau_2$

Let τ_1 and τ_2 be again two compatible trees. The *intersection* of τ_1 and τ_2 is the tree $\tau_1 \sqcap \tau_2$ such that:

i) $dom(\tau_1 \sqcap \tau_2) = dom(\tau_1) \cap dom(\tau_2)$;

ii) $\tau_1 \sqcap \tau_2 = \tau_{1|(dom(\tau_1)\cap dom(\tau_2))} = \tau_{2|(dom(\tau_1)\cap dom(\tau_2))}$

For example, the intersection of the trees τ_1 and τ_2 in Example 3.5 is the tree described by the graph below.



$\tau_1 \sqcap \tau_2$

Next operation we want define is the symmetric difference between two trees. It is not difficult to verify that, if we simply apply the corresponding set operation to the domains of the two trees, we do not get a tree but a bush. Since we want the result of an operation between trees to be still a tree (or, at most, a family of trees), we need to introduce more notations and defintions.

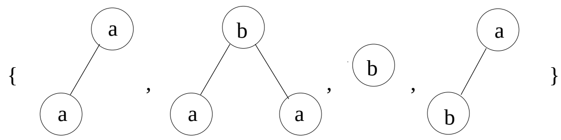
Let τ_1 and τ_2 be two compatible trees. We define, the bush $\tau_1 \diamond \tau_2$ as follows.

i) $dom(\tau_1 \diamond \tau_2) = dom(\tau_1 \oplus \tau_2) \setminus dom(\tau_1 \sqcap \tau_2)$;

ii) $\forall x \in dom(\tau_1 \diamond \tau_2),\ (\tau_1 \diamond \tau_2)(x) = \begin{cases} \tau_1(x) & \text{if } x \in dom(\tau_1) \\ \tau_2(x) & \text{otherwise} \end{cases}$

We are now ready to give the definition of the symmetric difference of two trees. Consider two compatible trees τ_1 and τ_2. Let $\beta = \tau_1 \diamond \tau_2 = \{\beta_1, \cdots, \beta_r\}$, where $\beta_1, \cdots, \beta_r$ denote the connected components of the bush β, and let $x_1, \cdots, x_r$ be the roots of $\beta_1, \cdots, \beta_r$, respectively. Remark that $x_i^{-1}\beta_i$ is a tree for any $i = 1, \ldots, r$. We refer to such a tree as the *tree corresponding to the connected component* β_i. The *symmetric difference* of τ_1 and τ_2 is the forest $\tau_1 \bigtriangleup \tau_2 = \{x_1^{-1}\beta_1, \cdots, x_r^{-1}\beta_r\}$.

For instance, the symmetric difference of the trees τ_1 and τ_2 in Example 3.5 is the following family of trees:



Remark 3.2 Notice that two words are compatible if one is prefix of the other one. Moreover the operations of union, intersection and symmetric difference applied to compatible words w_1 and w_2, give respectively the longest of the two words, say w_2, the smallest of the two words, say w_1 and the part where they differ, i.e. the word v such that $w_2 v = w_1$.

For example, given the words: $w_1 = a\,b\,b\,a\,b\,a$ and $w_2 = a\,b\,b\,a\,b\,a\,b\,a\,a$, we have:

$$w_1 \oplus w_2 = w_2$$

$$w_1 \sqcap w_2 = w_1$$

$$w_1 \bigtriangleup w_2 = b\,a\,a$$

We now define the concatenation between trees. Intuitively, to concatenate two trees τ_1 and τ_2 we "attach" the root of τ_2 to one of the elements of the border of τ_1. In general, as result, we get more than one tree since the border of a tree contains more than one element. Then the concatenation between two trees will be a set of trees containing as many trees as the elements in $B(\tau_1)$. We define first formally the concatenation of two trees at a given element of the border.

Let τ_1 and τ_2 be two trees, and let $B(\tau_1)$ denote the border of τ_1 (see Definition 3.2). The concatenation of τ_1 and τ_2 at $b \in B(\tau_1)$ is the tree $\tau_1(b)\tau_2$ defined as:

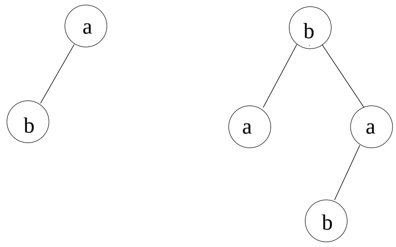
i) $dom(\tau_1(b)\tau_2) = dom(\tau_1) \cup b\, dom(\tau_2)$;

ii) $\forall x \in dom(\tau_1(b)\tau_2),\ (\tau_1(b)\tau_2)(x) = \begin{cases} \tau_1(x) & \text{if } x \in dom(\tau_1) \\ \tau_2(b^{-1}x) & \text{otherwise} \end{cases}$

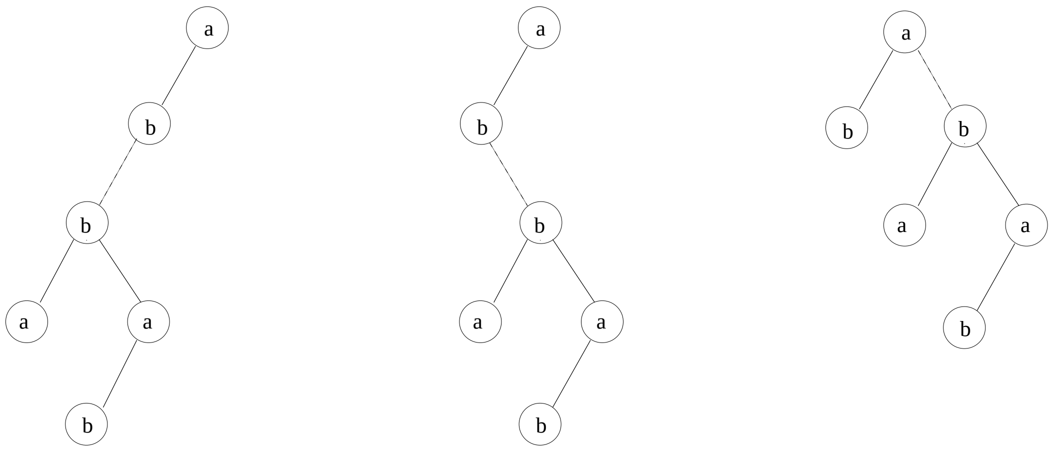
Definition 3.4 *Let τ_1 and τ_2 be two trees and let $B(\tau_1)$ be the border of τ_1. The* concatenation *of τ_1 and τ_2 is the set of trees:*

$$\tau_1 \cdot \tau_2 = \{\tau_1(b)\tau_2 \mid b \in B(\tau_1)\}.$$

Example 3.6 Given the trees τ_1 and τ_2



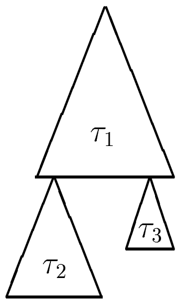
the concatenation $\tau_1 \cdot \tau_2$ is given by the following set of trees:



Remark 3.3 In the case of unary trees, i.e. of words, the result of this operation contains only one element that coincides with the word concatenation.

The notion of concatenation of two trees can be extended in the usual way to the concatenation of two sets of trees. If T_1 and T_2 are two families of trees, the concatenation of T_1 and T_2 is defined as the union of the sets $\tau_1 \cdot \tau_2$ where $\tau_1 \in T_1$ and $\tau_2 \in T_2$. It is denoted by $T_1 \cdot T_2$.

Notice that this operation is not associative. In general, given the trees τ_1,τ_2 and τ_3, we have that $\tau_1(\tau_2\tau_3) \subset (\tau_1\tau_2)\tau_3$. In fact the tree below belongs to $(\tau_1\tau_2)\tau_3$ but it does not belong to $\tau_1(\tau_2\tau_3)$.

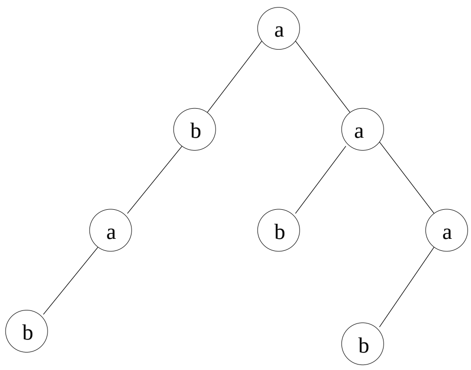


Definition 3.5 *Let τ be a tree. The set τ^n is defined recursively by $\tau^n = \tau^{n-1} \cdot \tau$ where $\tau^0 = \Omega$ and $\tau^1 = \{\tau\}$. Moreover, $\tau^{\#} = \cup_n \tau^n$.*

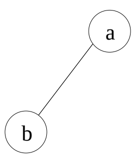
Notice that if we consider each element of the label-alphabet A as a punctual tree, $A^{\#}$ is exactly the set of all finite k-ary trees obtained as concatenation of all punctual trees in A. Then the notation $A^{\#}$ is coherent with the notation of $\tau^{\#}$.

Definition 3.6 *Given two trees τ_1, τ_2, we say that τ_1 is* a power *of τ_2 if $\tau_1 \in \tau_2^{\#}$. In this case, we say that τ_2 is a* root *of τ_1.*

Example 3.7 The following tree



is a power of the tree below.



We remark that, if we consider this definition over 1-ary trees, we retrieve the definition of power (and root) of a string over a finite alphabet A (cf. [12]).

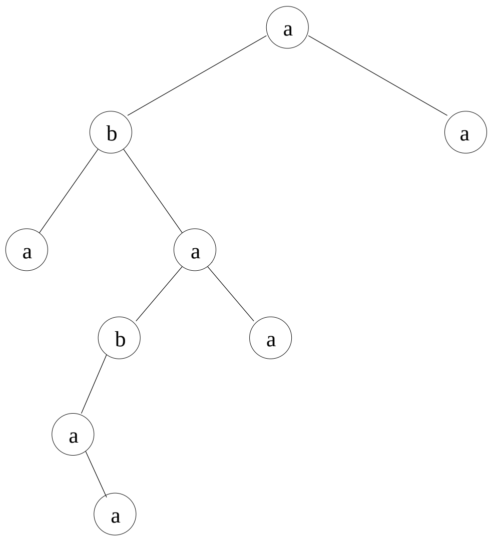
4 Periodicity on trees: definition and examples

In this section we generalize the notion of periodicity from words to labeled k-ary trees. We recall that a word w has period p if there exists a word u of length p such that w is a prefix of a word in u^*. This means, roughly speaking, that we can "factorize" the word w as concatenation of the word u by itself, plus a "remainder" that is a prefix of u.

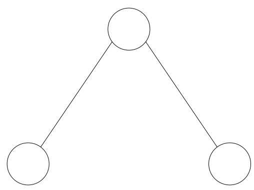
When we transpose this notions to trees we define periodicity of a labeled tree τ by means of a "factorization" of τ by a suitable tree τ_0, plus a "remainder" that is a set of prefixes of τ_0. In this case, the period is not a non negative integer like in the case of words, but it is an unlabeled tree, the shape of τ_0. Notice that in the case of unary trees, i. e. in the case of words, the shape is univocally determined by its size. The formal definition is given below.

Definition 4.1 *Let τ be a tree and t_0 be an unlabeled tree. We say that τ has period t_0 if there exists a tree τ_0 with $sh(\tau_0) = t_0$ such that τ is a prefix of an element of $\tau_0^{\#}$.*

Example 4.1 The tree below has as period the unlabeled tree t_0 drawn on its right:

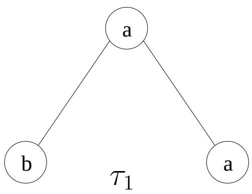


τ



t_0

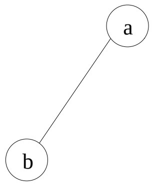
In fact τ is a prefix of a power of the following tree τ_1.



τ_1

In the sequel it will be useful to refer to the notion of *non trivial* period. If in Definition 4.1 we further require that τ_0 is prefix of τ, then we say that τ_0 is a non trivial period of τ. For instance, in the special case of words, a period p of a word w is non trivial if $p \leq |w|$. It is easy to verify that, in Example 4.1, t_0 is a non trivial period of τ.

The main problem we solve in this paper is that of giving a generalization of the Fine and Wilf's Theorem for words to labeled trees (cf. Theorem 2.1). We recall that this theorem considers the case when a word has two different periods: if a word w has two different periods p and q and it is "sufficiently long", then w has also as period the greatest common divisor of p and q. Also in the case of trees we study what happens if a tree has two different periods, and, we wonder under which conditions we can find a period that is "smaller" than the given ones. For instance, notice that the tree τ in Example 4.1 admits as period also the tree below.



t_2

Since we have to find a generalization of the Fine and Wilf's Theorem to trees, we have first to investigate what "greatest common divisor" and "sufficiently long" means in the case of trees. Moreover, according to our definition, a "period" is a shape of a tree, that is, an unlabeled tree. For this reason, most of considerations and results in the following sections will be referred to unlabeled trees.

5 Greatest common divisor of unlabeled trees

In this section we deal with unlabeled trees only. We give the definition of greatest common divisor of unlabeled trees and prove a theorem stating its existence and unicity.

We use a special terminology for unlabeled trees. This is motivated by the following remarks. In the case of words over the alphabet A, we use a multiplicative notation for the concatenation and, for instance, we say that w is a *power* of u if $w \in u^*$. However, if $|A| = 1$, A^* is isomorphic to the additive monoid of non negative integers and the additive notation is used for the concatenations of elements of A^*. In this case, if $w \in u^*$, we say that the integer corresponding to w is a *multiple* of the integer corresponding to u. In the same way, when no confusion arises, we will use the additive notation for unlabeled trees, whereas we use the multiplicative notation for labeled trees.

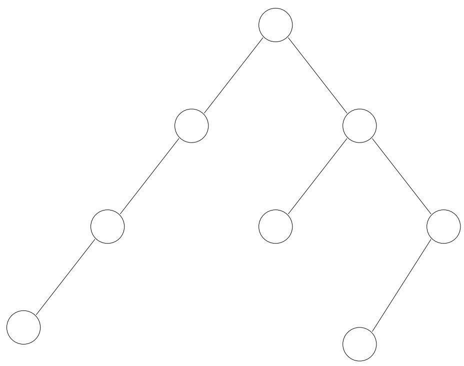
We start with the notion of "divisibility" for unlabeled trees. This notion has been also considered in [3].

Definition 5.1 *Given two unlabeled trees t_1 and t_2, we say that t_2* divides *t_1, and we write $t_2|t_1$, if $t_1 \in t_2^{\#}$. In this case we say that t_1 is a* multiple *of t_2 and that t_2 is a* divisor *of t_1.*

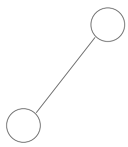
Remark that in the case of unary trees this corresponds to the notions of divisibility of integers.

Definition 5.2 *Given a set of unlabeled trees $T = \{t_1, \ldots, t_n\}$ and an unlabeled tree t, we say that t* divides *T, and we write $t|T$, if t divides all trees $t_1, \cdots, t_n \in T$.*

Example 5.1 The tree in the figure below



is a multiple of the following tree



We now give the definition of the greatest common divisor among trees.

Definition 5.3 *Let t_1, t_2 be two unlabeled trees. A tree t is the* greatest common divisor *of t_1 and t_2, and we denote it by $gcd(t_1, t_2)$, if:*

i) *t divides both t_1 and t_2;*

ii) *if there exists a tree $t' \in A^{\#}$ that divides both t_1 and t_2, then t' divides also t.*

It is useful to give also the definition of the greatest common divisor of a set of trees.

Definition 5.4 *Let $T = \{t_1, \ldots, t_n\}$ be a set of trees. A tree t is the* greatest common divisor *of T, and we write $t = gcd(T)$, if:*

i) *t divides T;*

ii) *if there exists a tree $t' \in A^{\#}$ that divides T, then t' divides also t.*

Before stating the main theorem of this section we prove two preliminary lemmas.

Recall that two unlabeled trees are always compatible. In this case the basic operation defined in Section 3.2 are *total* operations.

Lemma 5.1 *Let t_1 and t_2 be two unlabeled trees and let t be a divisor of t_1 and t_2. Then the following properties hold.*

1. *t divides $t_1 \cdot t_2$;*
2. *t divides $t_1 \sqcap t_2$;*
3. *t divides $t_1 \triangle t_2$;*
4. *t divides $t_1 \oplus t_2$.*

Proof: Since $t_1, t_2 \in t^{\#}$ then $t_1 \cdot t_2 \subset t^{\#}$. This proves statement *1.*

To prove *2.*, notice that both t_1 and t_2 are multiple of t, then both of them have t as prefix, that is, $t \subseteq t_1 \sqcap t_2$. Let p be the largest prefix of $t_1 \sqcap t_2$ that belongs to $t^{\#}$. We have to show that $p = t_1 \sqcap t_2$. Let us suppose by contradiction, that p is a proper prefix of $t_1 \sqcap t_2$, that is, $p \neq t_1 \sqcap t_2$. Then there exists a node in $t_1 \sqcap t_2$ that is an element of the border of p. This node must be necessairely the root of an occurence of t in t_1 and in t_2. But, if both t_1 and t_2 contain this node, then they both contain the whole occurrence of t having that node as root. Then we can find a tree $p' \in pt \subset t^{\#}$ contained in $t_1 \sqcap t_2$. This contradicts the hypothesis of maximality for p.

We now prove *3.*. Let $d \in t_1 \triangle t_2$. By definition of $t_1 \triangle t_2$, there exists a maximal connected component b in the bush $t_1 \diamond t_2$ such that d is the translated of b. It suffices to show that t divides b, to prove that t divides d. Since b is either a suffix of t_1 or of t_2 (and $t_1, t_2 \in t^{\#}$), its root R belongs to the domain of some occurrence of t in t_1 (or in t_2). Let R' be the root of this occurrence. If $R = R'$ then t divides b. Otherwise, the root R' belongs to the intersection $t_1 \sqcap t_2$. But in this case the entire occurrence of t must be contained in $t_1 \sqcap t_2$, because t divides $t_1 \sqcap t_2$ (by *2.*). Then $R \in t_1 \sqcap t_2$, and this contradicts the hypothesis that $b \in t_1 \diamond t_2$.

To prove *4.*, notice that $t_1 \oplus t_2$ consists in the concatenation of $t_1 \sqcap t_2$ with elements in $t_1 \triangle t_2$ at fixed elements of the border. By statements *1.* and *3.*, we have that t divides $t_1 \oplus t_2$. □

Lemma 5.2 *Let t_1 and t_2 be two unlabeled trees. Then, $gcd(t_1, t_2) = gcd(\{t_1 \sqcap t_2\} \cup (t_1 \triangle t_2))$.*

Proof: If $t = gcd(t_1, t_2)$ then $t|t_1$ and $t|t_2$. By Lemma 5.1, $t|(t_1 \sqcap t_2)$ and $t|(t_1 \triangle t_2)$. If t' divides both $t_1 \sqcap t_2$ and $t_1 \triangle t_2$, then it divides also t_1 and t_2. In fact t_1 (resp. t_2) is obtained by concatenating elements of $t_1 \triangle t_2$ to some elements of the border of $t_1 \sqcap t_2$. Then, by *3.* and *1.* of Lemma 5.1, t' divides t_1 and t_2, then t' divides also $t = gcd(t_1, t_2)$. Therefore, $t = gcd(\{t_1 \sqcap t_2\} \cup (t_1 \triangle t_2))$. □

Theorem 5.1 *Given two unlabeled trees t_1 and t_2, $gcd(t_1, t_2)$ exists and it is unique.*

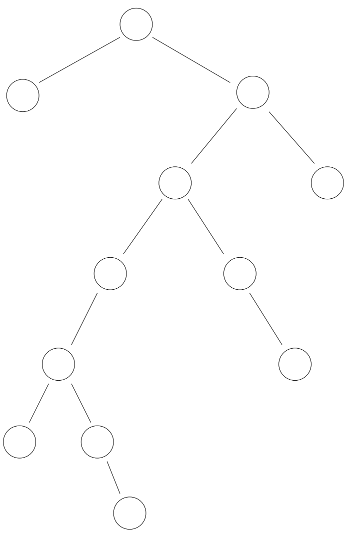
Proof: We describe an algorithm to compute the $gcd(t_1, t_2)$. If $t_1 = t_2$ then $gcd(t_1, t_2) = t_1 = t_2$. Suppose $t_1 \neq t_2$: we define $T_0 = \{t_1, t_2\}$ and for $k > 0$ we recursively compute sets

$$T_k = \bigcup_{t_i, t_j \in T_{k-1}} \{t_i \sqcap t_j\} \cup (t_i \bigtriangleup t_j)$$

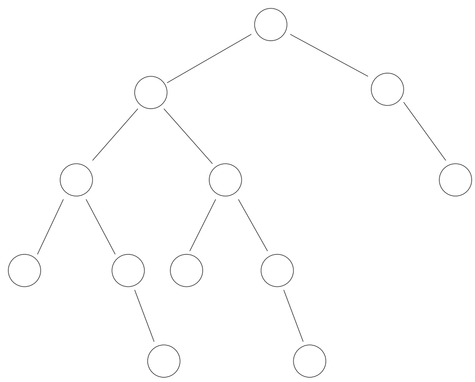
Notice that by applying recursively the argument of Lemma 5.2 we have that $gcd(t_1, t_2) = gcd(T_k)$ for all k. We stop when T_k contains only one element. Remark that, if $|T_k| \neq 1$, then $T_k \neq T_{k+1}$. It is easy to verify that, at each step k, we reduce the sizes of the trees in the set T_k. In particular, the maximal size of trees in T_{k+1} is strictly smaller than the maximal size of trees in T_k. This guarantees the termination of the algorithm. The greatest common divisor is the unique tree contained in the terminal set T_k. $\square$

Remark that the algorithm in the proof of Theorem 5.1 is a generalization to trees of the Euclid algorithm for the greatest common divisor of two integers. In fact, in the case of unary trees, (t_1, t_2) corresponds to a pair of integers and it is easy to verify that, if T_k corresponds to the pair (n_1, n_2), with $n_1 > n_2$, then T_{k+1} corresponds to the pair $(n_1 - n_2, n_2)$.

Example 5.2 By applying the algorithm of Theorem 5.1 to the following pair of trees,

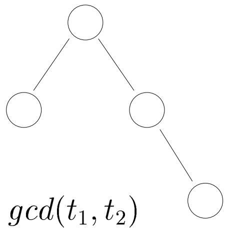


t_1



t_2

we obtain in two steps that the greatest common divisor of t_1 and t_2 is the tree below.



$gcd(t_1, t_2)$

6 Unlabeled trees and right congruences

In this section we associate right congruences to unlabeled trees. Since unlabeled trees are periods of labeled trees, this will allow us to describe periodicity on trees also in terms of right congruences generalizing the analogous characterization for words to trees (see Proposition 2.1). If t is an unlabeled tree, we will denote the corresponding right congruence by $\mathcal{R}(t)$. The main result of this section states that, given two trees t_1 and t_2, $\mathcal{R}(gcd(t_1,t_2)) = \mathcal{R}(t_1) \vee \mathcal{R}(t_2)$, where $\mathcal{R}(t_1) \vee \mathcal{R}(t_2)$ denotes the join of the congruences $\mathcal{R}(t_1)$ and $\mathcal{R}(t_2)$.

We start by giving notions and definitions on equivalences and congruences. Given two equivalence relations $\mathcal{E}_1$ and $\mathcal{E}_2$ on a same set X, we say that $\mathcal{E}_1$ is smaller than or equal to $\mathcal{E}_2$ (and we write $\mathcal{E}_1 \leq \mathcal{E}_2$) if each equivalence class of $\mathcal{E}_1$ is union of equivalence classes of $\mathcal{E}_2$.

Usually, if $\mathcal{E}_1 \leq \mathcal{E}_2$, one says that $\mathcal{E}_2$ is *finer* than $\mathcal{E}_1$ or that $\mathcal{E}_1$ is *coarser* than $\mathcal{E}_2$. It is easy to see that $\leq$ is an order relation between equivalences. The standard way to prove that $\mathcal{E}_1 \leq \mathcal{E}_2$ is to show that if $u\mathcal{E}_2 v$ then $u\mathcal{E}_1 v$, with $u, v \in \Sigma^*$.

Definition 6.1 *Let $\mathcal{E}_1$ and $\mathcal{E}_2$ be two equivalence relations on the same set X. The* join *of $\mathcal{E}_1$ and $\mathcal{E}_2$, denoted by $\mathcal{E}_1 \vee \mathcal{E}_2$, is the greatest equivalence smaller than or equal to both $\mathcal{E}_1$ and $\mathcal{E}_2$.*

Remark 6.1 From the definition it follows that $u(\mathcal{E}_1 \vee \mathcal{E}_2)v$, if and only if there exists a natural number $n \geq 0$ and a sequence $(w_1, w_2, \cdots, w_n)$ of elements of X such that $u\mathcal{E}_{j_1}w_1$, $w_1\mathcal{E}_{j_2}w_2$, $w_2\mathcal{E}_{j_3}w_3, \cdots$, $w_n\mathcal{E}_{j_{n+1}}v$, $j_i \in \{1,2\}$.

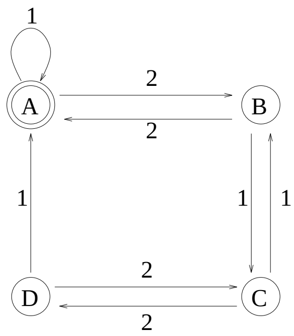
Definition 6.2 *A* right congruence *over a free monoid Σ^* is an equivalence relation $\mathcal{R}$ on the words of Σ^* such that, if $u\mathcal{R}v$, then $uw\mathcal{R}vw$ for all $w \in \Sigma^*$.*

Notice that the join of two right congruences is still a right congruence.

Given a right congruence $\mathcal{R}$ of finite index over Σ^*, we may describe it by a finite labeled directed graph to which we will refer as $G(\mathcal{R})$. The vertices $V_1, V_2, \ldots, V_n$ of $G(\mathcal{R})$ correspond to the equivalence classes $C_1, C_2, \ldots, C_n$ of $\mathcal{R}$. The vertex that corresponds to the class containing the empty word will be referred to as the *initial vertex*. The edges are defined in a way that there is an edge from vertex V_i to vertex V_j labeled $x \in \Sigma$ if $C_i x \subseteq C_j$. Notice that the graph $G(\mathcal{R})$ uniquely determines the right congruence $\mathcal{R}$. Indeed, for any word $w \in \Sigma^*$, the class of $\mathcal{R}$ containing w corresponds to the vertex reached by the (unique) path starting from the initial vertex and labeled w. Graph $G(\mathcal{R})$ will be referred to as *congruence graph*. Remark that $G(\mathcal{R})$ is a complete deterministic graph (i.e. for any letter $x \in \Sigma$ and for any vertex V, there is exactly one edge leaving V labeled x). Moreover, observe that $G(\mathcal{R})$ is always a connected graph. Conversely, for any complete deterministic connected graph G with a special vertex V_1 there exists a unique right congruence $\mathcal{R}$ such that $G = G(\mathcal{R})$.

A right congruence $\mathcal{R}$ is *strongly connected (s.c.)* if, for any $u \in \Sigma^*$, there exists $v \in \Sigma^*$ such that $uv\mathcal{R}\epsilon$, where ϵ is the empty word of Σ^*. This terminology is motivated by the fact that $\mathcal{R}$ is s.c. if and only if $G(\mathcal{R})$ is a strongly connected graph.

Example 6.1 The graph:



represents the strongly connected right congruence defined by the following relation between classes:

$A \cdot 1 \subseteq A$	$B \cdot 2 \subseteq A$	$D \cdot 1 \subseteq A$
$A \cdot 2 \subseteq B$	$C \cdot 1 \subseteq B$	$D \cdot 2 \subseteq C$
$B \cdot 1 \subseteq C$	$C \cdot 2 \subseteq D$	

The following lemma associates a maximal prefix code to a s.c. right congruence.

Lemma 6.1 *Let $\mathcal{R}$ be a s.c. right congruence and let $[\epsilon]$ denote the class of $\mathcal{R}$ containing the empty word ϵ. Then, $[\epsilon]$ is a submonoid of Σ^* generated by a maximal prefix code, referred to as $C(\mathcal{R})$. Moreover, if $\mathcal{R}$ has finite index then $C(\mathcal{R})$ is a rational set.*

Proof: Since $\mathcal{R}$ is a s.c. right congruence, then there exists a complete connected graph $G(\mathcal{R})$ that is the congruence graph for $\mathcal{R}$. Let us denote by V_1 the initial node, i.e. the node corresponding to class $[\epsilon]$. Remark that class $[\epsilon]$ contains the labels of all paths from V_1 to V_1. The operation of concatenation between two of such words is trivially an internal associative operation. Moreover the empty word belongs to $[\epsilon]$ and, therefore, $[\epsilon]$ is a submonoid of Σ^*.

The base of this monoid, that we denote by $C(\mathcal{R})$, is the set of non empty words in Σ^* that are labels of paths in $G(\mathcal{R})$ from V_1 to V_1 that reach V_1 only once. This is a prefix code. In fact none of such words can have another word of this set as prefix, otherwise the corresponding path in $G(\mathcal{R})$ would touch V_1 twice. Moreover such code is maximal because the graph $G(\mathcal{R})$ is complete.

Code $C(\mathcal{R})$ is also a rational set. In fact we can consider a graph G', obtained from $G(\mathcal{R})$ by adding a vertex V_f to the set of vertices of $G(\mathcal{R})$ and re-directing all the edges entering V_1 into V_f instead. We now consider the automaton having vertices of G' as the set of states, V_1 and V_f as as initial and final state, respectively and the transitions defined by graph G'. This automaton accepts words of the $C(\mathcal{R})$, that is, $C(\mathcal{R})$ is a rational set. $\square$

Now we reverse the correspondence established by the previuos lemma and associate a s.c. right congruence to a maximal prefix code. Let C be a maximal prefix code over the alphabet Σ and let us denote by $Pr(C)$ the set of *proper* prefixes of elements of C, that is:

$$Pr(C) = \{u \in \Sigma^* | u\Sigma^+ \cap C \neq \emptyset\}.$$

It is well known (cf. [2]) that, for any word $v \in \Sigma^*$ there exists a *unique* element $u \in Pr(C)$ such that $v \in C^*u$. This unique element is denoted by $r_C(v)$.

Definition 6.3 *Given a maximal prefix code C over Σ, and two words $u, v \in \Sigma^*$, we say that $u \equiv_C v$ if $r_C(u) = r_C(v)$.*

It is easy to verify that the equivalence $(\equiv_C)$ is a s.c. right congruence whose index is $|Pr(C)|$.

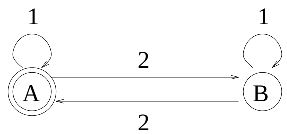
Remark 6.2 In the special case $|\Sigma| = 1$, a maximal prefix code C over Σ contains only one word, which is uniquely specified by its length p. In this case Σ^* is isomorphic to the additive monoid $\mathbb{N}$ of natural numbers and the congruence $\equiv_C$ corresponds to the equivalence *mod p* over $\mathbb{N}$.

Lemma 6.2 *Let $\mathcal{R}$ be a s.c. right congruence on Σ^*. Then $\mathcal{R} \leq (\equiv_{C(\mathcal{R})})$.*

Proof: Notice that $u \equiv_{C(\mathcal{R})} v$ implies that there exists $w \in Pr(C(\mathcal{R}))$ such that $u, v \in (C(\mathcal{R}))^* w = [\epsilon]w$. This implies that there exist $u', v' \in [\epsilon]$ such that $u = u'w$ and $v = v'w$. Since $u', v' \in [\epsilon]$, then $u'\mathcal{R}v'$ and therefore $u\mathcal{R}v$. $\Box$

Remark that, in general $\mathcal{R} \neq (\equiv_{C(\mathcal{R})})$. This is shown by the following example.

Example 6.2 Let us consider the right congruence defined by the following congruence graph:



We have that $C(\mathcal{R}) = \{1, 21^*2\}$, that is $\equiv_{C(\mathcal{R})}$ has an infinite number of equivalence classes, while $\mathcal{R}$ has only two classes.

The following lemma is an immediate consequence of the definitions.

Lemma 6.3 *Let C be a maximal prefix code over Σ^*. Then $C = C(\equiv_C)$.*

Lemma 6.4 *Let C_1 and C_2 be two maximal prefix codes over Σ. Then $C_1 \subseteq C_2^*$ if and only if $(\equiv_{C_2}) \leq (\equiv_{C_1})$.*

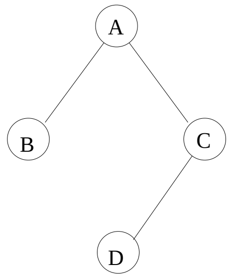
Proof: $(\Rightarrow)$ $u \equiv_{C_1} v$ implies that $u, v \in C_1^* w_1$, with $w_1 \in Pr(C_1)$. Since C_2 is a maximal prefix code there exists a unique element $w_2 \in Pr(C_2)$ such that $w_1 \in C_2^* w_2$. Since $C_1 \subseteq C_2^*$, then $u, v \in C_2^* w_2$, i.e. $u \equiv_{C_2} v$.

($\Leftarrow$) By definition, for any maximal prefix code C, $u \equiv_C \epsilon$ if and only if $u \in C^*$. Since $u \equiv_{C_1} \epsilon$ implies that $u \equiv_{C_2} \epsilon$,then $C_1^* \subseteq C_2^*$. $\square$

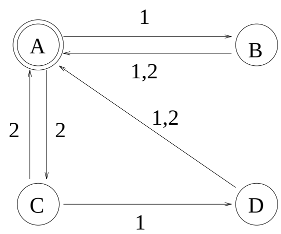
We now associate a right congruence to an unlabeled tree. Let t be an unlabeled tree and let $B(t)$ be the border of t (see Definition 3.2). Recall that the border $B(t)$ of t is a (finite) maximal prefix code. We define the right congruence associated to t as $\mathcal{R}(t) = (\equiv_{B(t)})$. Remark that $Pr(B(t)) = dom(t)$ and then the classes of $\mathcal{R}(t)$ are in a one-to-one correspondence to the elements of $dom(t)$.

Definition 6.4 *A right congruence $\mathcal{R}$ is a* tree congruence *if there exists an unlabeled tree t such that $\mathcal{R} = \mathcal{R}(t)$.*

Example 6.3 Given the following tree



the tree congruence associated to it is described by the following congruence graph:



We now prove some lemmas regarding tree congruences.

Lemma 6.5 *A right congruence $\mathcal{R}$ is a* tree congruence *if and only if $\mathcal{R} = (\equiv_{C(\mathcal{R})})$ and $C(\mathcal{R})$ is a* finite *prefix code.*

Proof: If $\mathcal{R}$ is a tree congruence, then $\mathcal{R} = \mathcal{R}(t) = (\equiv_{B(t)})$, for some tree t. By Lemma 6.3, $C(\mathcal{R}) = C(\equiv_{B(t)}) = B(t)$ and then $\mathcal{R} = (\equiv_{C(\mathcal{R})})$. Conversely, if $\mathcal{R} = (\equiv_{C(\mathcal{R})})$, consider the tree t defined by the domain $dom(t) = Pr(C(\mathcal{R}))$. One can easily verify that $\mathcal{R} = \mathcal{R}(t)$. $\square$

Lemma 6.6 *Let $\mathcal{R}$ be a right congruence and let $\mathcal{R}'$ be a tree congruence such that $\mathcal{R} \leq \mathcal{R}'$. Then $\mathcal{R}$ is s.c. and $\mathcal{R} \leq (\equiv_{C(\mathcal{R})}) \leq \mathcal{R}'$.*

Proof: Since $\mathcal{R} \leq \mathcal{R}'$, $\mathcal{R}'$ s.c. implies that, for any $u \in \Sigma^*$, there exists $v \in \Sigma^*$ such that $uv\mathcal{R}'\epsilon$. It follows that $uv\mathcal{R}\epsilon$, i.e. $\mathcal{R}$ is s.c. Now $[\epsilon]_{\mathcal{R}'} \subseteq [\epsilon]_{\mathcal{R}}$ implies that $C(\mathcal{R}') \subseteq C(\mathcal{R}))^*$ and then, by Lemma 6.4, an Lemma 6.2 we have $\mathcal{R} \leq \equiv_{C(\mathcal{R})} \leq \equiv_{C(\mathcal{R}')} = \mathcal{R}'$. $\square$

Lemma 6.7 *The join of two tree congruences is a tree congruence.*

Proof: Let $\mathcal{R}_1, \mathcal{R}_2$ be two tree congruences. Let $\mathcal{R}_1 \vee \mathcal{R}_2 = \mathcal{R}$. $\mathcal{R}$ is the greatest right congruence such that $\mathcal{R} \leq \mathcal{R}_1$ and $\mathcal{R} \leq \mathcal{R}_2$. Let $C(\mathcal{R})$ be the prefix code corresponding to $\mathcal{R}$. By the previous lemma one has:
$\mathcal{R} \leq (\equiv_{C(\mathcal{R})}) \leq \mathcal{R}_1$
$\mathcal{R} \leq (\equiv_{C(\mathcal{R})}) \leq \mathcal{R}_2$.
These inequalities imply that $\mathcal{R} = (\equiv_{C(\mathcal{R})})$. $\square$

Lemma 6.8 *Let t_1 and t_2 be two unlabeled trees and let $B(t_1)$, $B(t_2)$ be their borders. Then t_1 divides t_2 if and only if $B(t_2) \subseteq B(t_1)^*$.*

Proof: If t_1 divides t_2, then, by definition, any element of $B(t_2)$ is obtained as concatenation of elements of $B(t_1)$.
Conversely, suppose that $B(t_2) \subseteq (B(t_1))^*$. Let

$$m = max\{k \in I\!N \mid (B(t_1))^k \cap B(t_2) \neq \emptyset\}.$$

The proof is by induction on m. If $m = 1$, then $t_1 = t_2$ and trivially t_1 divides t_2. Now, let us assume that the statement is true for $j = 1, 2, ..., m-1$ and consider the set

$$(B(t_1))^m \cap B(t_2) = \{w_1, w_2, ..., w_r\}.$$

By definition, for any fixed index i, the word w_i can be decomposed as $w_i = u_i v_i$, with $u_i \in (B(t_1))^{m-1}$ and $v_i \in B(t_1)$. For any $v' \in B(t_1)$, consider the word $u_i v'$. Since $B(t_2)$ is a maximal prefix code, either $u_i v'$ is prefix of a word w' in $B(t_2)$, or there exists a word w' in $B(t_2)$ that is prefix of $u_i v'$. In the first case, $w' = u_i v' x$, for some $x \in \Sigma^*$, and, since $B(t_2) \subseteq (B(t_1))^*$, $v'x \in (B(t_1))^*$. By the maximality of m, $x = \epsilon$ and then $u_i v' \in B(t_2)$. In the second case, since $B(t_2)$ is prefix code, then $w' = u_i y$ with y prefix of v'. By the inclusion $B(t_2) \subseteq (B(t_1))^*$ one derives that $y = v'$. It follows that $u_i B(t_1) \subseteq B(t_2)$ for any $i = 1, 2, ..., r$. Consider now the set

$$C = B(t_2) \setminus [((B(t_1))^m \cap B(t_2)) \cup \{u_1, u_2, ..., u_r\}].$$

It is easy to verify that C is a maximal prefix code corresponding to the border of the tree obtained from t_2 by deleting the copies of the subtree t_1 in the bottom of t_2, having leaves of maximal depth. By the induction hypothesis, the statement follows. $\square$

Using previous lemmas we can now prove the following theorem.

Theorem 6.1 *Let t_1 and t_2 be two unlabeled trees. Then, $\mathcal{R}(gcd(t_1, t_2)) = \mathcal{R}(t_1) \vee \mathcal{R}(t_2)$*

Proof: Since the join of two tree congruences is a tree congruence (by Lemma 6.7) then there exists a tree t such that $\mathcal{R}(t_1) \vee \mathcal{R}(t_2) = \mathcal{R}(t)$. Moreover, by Lemmas 6.4 and 6.8, for any pair of trees t, t', we have that $\mathcal{R}(t) \leq \mathcal{R}(t')$ if and only if $B(t') \subseteq B(t)$, and this happens if and only if t divides t'. Therefore, t divides both t_1 and t_2 and then t divides $gcd(t_1, t_2)$. Conversely, since $gcd(t_1, t_2)$ divides both t_1 and t_2, then $\mathcal{R}(gcd(t_1, t_2)) \leq \mathcal{R}(t_1)$ and $\mathcal{R}(gcd(t_1, t_2)) \leq \mathcal{R}(t_2)$. It follows that $\mathcal{R}(gcd(t_1, t_2)) \leq \mathcal{R}(t)$, and then $gcd(t_1, t_2)$ divides t. $\square$

7 The join of right congruences and restrictions

In this section we describe and discuss an algorithm to compute the join of two right congruences of finite index over Σ^*. This algorithm can be implemented using a "union-find" data structure and its running time is dominated by the running time of "union-find"(cf. [1]). Moreover, since by Theorem 6.1 the join of two tree congruences $\mathcal{R}(t_1)$ and $\mathcal{R}(t_2)$ is the tree congruence $\mathcal{R}(gcd(t_1, t_2))$, this algorithm is a "fast way" to compute the greatest common divisor between two trees; more precisely, if n is the sum of the sizes of the two trees, the running time of the algorithm is smaller than or equal to a constant times $n\alpha(n)$, where α is the inverse of the Ackermann's function (cf. [1]).

In this section we consider also the restrictions of congruences to subsets of Σ^* and their relation with the join operation (recall that, if $\mathcal{R}_1$ and $\mathcal{R}_2$ are right congruences, then the join $\mathcal{R}_1 \vee \mathcal{R}_2$ is also a right congruence). This leads to the notion of *complete sets* with respect to two congruences. Finally we prove that there exist prefix closed sets that are complete with respect to two fixed congruences and that have "small" size.

For the sequel, we will denote by $\mathcal{R}^T$ the restriction of the equivalence $\mathcal{R}$ to the set $T \subseteq \Sigma^*$.

7.1 An algorithm for the join

In this section we deal with right congruences of finite index over Σ^*. We denote by $i(\mathcal{R})$ the index of the right congruence $\mathcal{R}$ (i.e. the number of classes that $\mathcal{R}$ defines over the set Σ^*).

We describe an algorithm to compute the join of two right congruences over Σ^* that operates on the corresponding congruence graphs. We first introduce some terminologies on graphs.

Given a directed labeled graph G, we indicate by $[V_i, x, V_j]$ the edge from vertex V_i to vertex V_j with label $x \in \Sigma$. We define an operation on G called *Merge* that unifies two vertices into one vertex as follows. Given vertices V_1, V_2 in G then *Merge*(V_1, V_2) transforms these two vertices into a unique vertex, say V'. The edges starting from V' are all the edges that were previously starting from V_1, V_2, paying attention to the deletion of possible duplicate edges, while the edges entering in V' are simply all the edges that were previously entering in V_1, V_2.

Given a letter $x \in \Sigma$, we say that a vertex V is *good for x* if there exists exactly one edge starting in V with label x. We say that V is *good* if V is good for any letter $x \in \Sigma$.

We now describe briefly and informally the algorithm. The inputs are two congruence graphs $G(\mathcal{R}_1)$ and $G(\mathcal{R}_2)$ whose vertices are $U_1, \ldots, U_n$ and $U'_1, \ldots, U'_m$, respectively, where U_1 and U'_1 are the initial vertices of $G(\mathcal{R}_1)$ $(G(\mathcal{R}_2)$, respectively. The output is a new graph G and a set of words $P \subseteq \Sigma^*$ that will be used later in this section.

Algorithm J

- **Initialization step:**
  Consider the two graphs $G(\mathcal{R}_1)$ and $G(\mathcal{R}_2)$ as a unique (not connected) graph, apply the operation $\mathit{Merge}(U_1, U'_1)$ and call U the resulting vertex. Denote by G the resulting connected graph; define a set of words $P \subseteq \Sigma^*$ and initialize it as $P = \{\epsilon\}$.

- **Main step:**
  Take a *bad* vertex V in G. Suppose that V is bad for the letter x. This fact implies that there exists a word $w \in P$ and two vertices V_1 and V_2 such that:

  1) there exists a path in G labeled w that starts in U and reaches V;

  2) the edges $[V, x, V_1], [V, x, V_2]$ are in G;

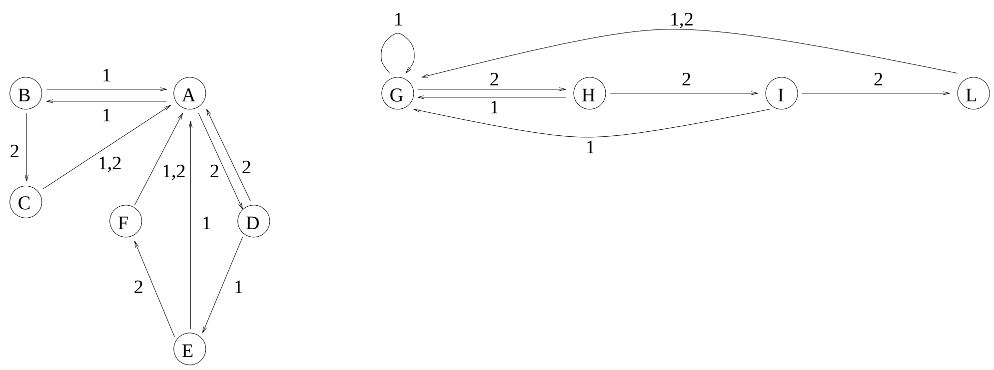
  3) the path labeled $w \cdot x$ in $G(\mathcal{R}_1)$ (in $G(\mathcal{R}_2)$ resp.) reaches *either* a vertex that has been merged in a previous step to obtain V_1 (V_2 resp.) *or* a vertex that has been merged in a previous step to obtain V_2 (V_1 resp.).

  Choose and fix the word w and vertices V_1 and V_2. Apply the operation $\mathit{Merge}(V_1, V_2)$ and add the new word $w \cdot x$ to the set P.

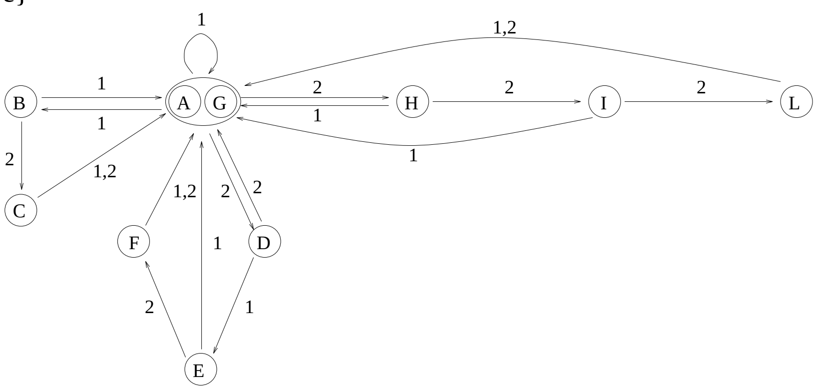
The algorithm works by performing the initialization step followed by the main step. Notice that the first time the main step is executed, the only *bad* vertex is U created in the initialization. Then, it runs by repeating the main step until there are no more *bad* vertices left.

Example 7.1 We show a run of the above algorithm applied to the congruences $\mathcal{R}_1$ and $\mathcal{R}_2$ over the alphabet $\Sigma = \{1, 2\}$ represented by the congruence graphs below.

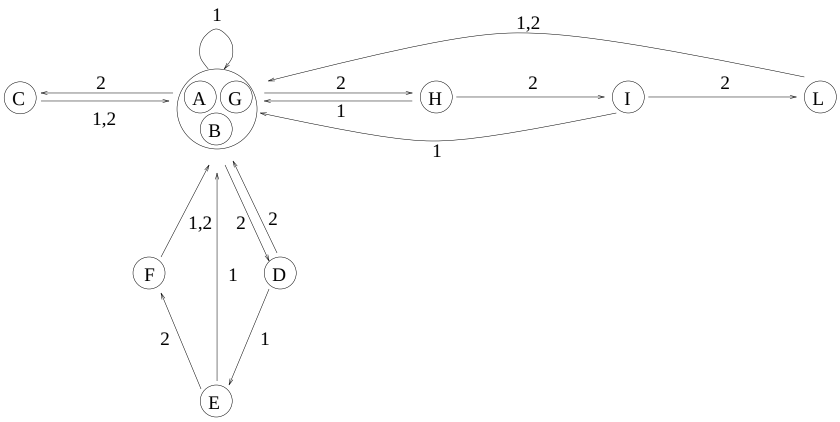
INPUT:



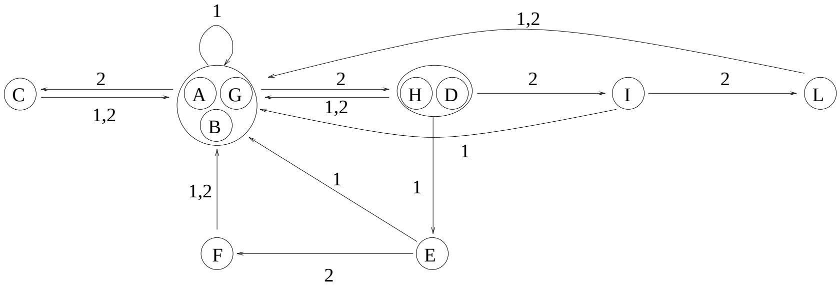
Step 1: P = {e}



Step 2: P = {e, 1}

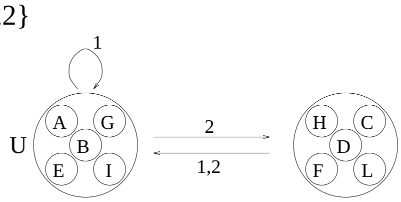


Step 3: P = {e, 1, 2}



After five more steps we obtain the following:

Final step: P = {e, 1, 2, 21, 22, 222}



Notice that, by construction, the set P output by the above algorithm is prefix-closed, hence it can be thought as an unlabeled tree. In fact, as we remarked before, an unlabeled tree is univocally defined by its domain (that is a prefix closed subset of Σ^*).

Remark 7.1 If the path in G starting from U and labeled z reaches some vertex W, then the vertices V_z and V'_z reached by the path labeled z in $G(\mathcal{R}_1)$ and in $G(\mathcal{R}_2)$, respectively, have been merged during the run of the algorithm in order to obtain vertex W (otherwise it is easy to see that in G there would remain at least one more bad vertex).

Remark 7.2 The algorithm J is "non-deterministic" in the sense that, after the initialization, each time we execute the main step, we make a choice (for a bad vertex). Different sequences of choices correspond to different runs of the algorithm on a given input. All such possible different runs of the algorithm produce the same graph G but different sets P's. In the case of unary alphabets the algorithm J is deterministic.

In the remaining part of this section we prove the correctness of the above algorithm.

Lemma 7.1 *Let G and P be respectively the graph and the subset of Σ^* output by the above algorithm. If for some pair of words $z, y \in \Sigma^*$ there exist two paths in G starting from the vertex U and labeled by z and y, that reach the same vertex W, then $z(\mathcal{R}_1^T \vee \mathcal{R}_2^T)y$ where $T = P \cup \{z, y\}$.*

Proof: Let us denote by G_m the graph we have after m steps of the algorithm. We claim that, if paths labeled z and y in $G(\mathcal{R}_1)$ (or, equivalentely, in $G(\mathcal{R}_2)$) reach the same vertex or two different vertices that have been merged at a certain step of the algorithm, in order to obtain a single vertex W in G_m, then $z(\mathcal{R}_1^T \vee \mathcal{R}_2^T)y$, where $T = P \cup \{z, y\}$.

The statement of the proposition follows by Remark 7.1 and by previous claim when we consider the the final step as *m-th* step.

Let us denote by V_z and V_y the vertices reached in $G(\mathcal{R}_1)$ (or, equivalently, in $G(\mathcal{R}_2)$) by paths labeled z and y respectively. If $V_z = V_y$ then previous claim is trivially true ($z\mathcal{R}_1 y$ or $z\mathcal{R}_2 y$ depending whether the vertex $V_z = V_y$ is in $G(\mathcal{R}_1)$ or in $G(\mathcal{R}_2)$).

Hence we can suppose that $V_z \neq V_y$ and we prove the claim by induction on m.

Base of the induction. If V_z and V_y are merged after the initialization step, the vertex W must be the vertex U otherwise $V_z = V_y$. There are 2 cases. First case: both paths in $G(\mathcal{R}_1)$ (or in $G(\mathcal{R}_2)$, equivalently), labeled by z and y starting from U_1 (from U'_1 resp.) reach the vertex U_1 (or U'_1 resp.); in this case again $z\mathcal{R}_1 y$ (or $z\mathcal{R}_2 y$ resp.). Second case: the path labeled z reaches U_1 in $G(\mathcal{R}_1)$ and the path labeled y reaches U'_1 in $G(\mathcal{R}_2)$ (or the analogous condition obtained by interchanging z with y); in this case $z\mathcal{R}_1\epsilon$, $\epsilon\mathcal{R}_2 y$ (or, analogously, $y\mathcal{R}_1\epsilon$, $\epsilon\mathcal{R}_2 z$). Since $\epsilon \in P$, the base of induction is proved.

Inductive step. Suppose now that V_z and V_y are merged into a single vertex W after the algorithm performed m steps with $m \geq 2$. If V_z and V_y were already merged into a single vertex $\hat{V}$ in G_{m-1} (that is, at the $(m-1)$*-th* step of the algorithm), then, by inductive hypothesis, the claim holds.

Suppose now that V_z and V_y are not merged into a single vertex in G_{m-1}; thus W must be the vertex resulting by the operation *merge* performed at the *m-th* step. Let us call V_1 and V_2 the vertices in G_{m-1} that "contain" V_z and V_y respectively; the vertex V'

is the result of the operation merge applied to the pair (V_1, V_2). Therefore there exist a vertex V in G_{m-1}, a word $w \in P$ and a letter $x \in \Sigma$, chosen by the algorithm such that:

1) there exists a path in G_{m-1} labeled w that starts in U and reaches V;

2) the edges $[V, x, V_1], [V, x, V_2]$ are in G_{m-1};

3) the path labeled $w \cdot x$ in $G(\mathcal{R}_1)$ (in $G(\mathcal{R}_2)$ resp.) reaches *either* a vertex that has been merged to obtain V_1 (V_2 resp.) *or* a vertex that has been merged to obtain V_2 (V_1 resp.).

By the inductive hypothesis applied to z and $w \cdot x$ at the $(m-1)$-*th* step, one has that $z(\mathcal{R}_1^T \vee \mathcal{R}_2^T) w \cdot x$; again by the inductive hypothesis applied to $w \cdot x$ and y at the $(m-1)$-*th* step, one has that $w \cdot x(\mathcal{R}_1^T \vee \mathcal{R}_2^T)y$, and, consequently, since $w \cdot x \in P$, $z(\mathcal{R}_1^T \vee \mathcal{R}_2^T)y$. □

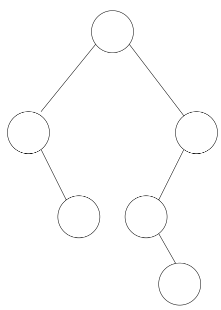
Theorem 7.1 *The graph G output by the algorithm is the congruence graph $G(\mathcal{R})$, where $\mathcal{R} = \mathcal{R}_1 \vee \mathcal{R}_2$.*

Proof: The graph G is a complete deterministic connected graph with special vertex U. Hence there exists a right congruence $\mathcal{R}$ such that $G = G(\mathcal{R})$.

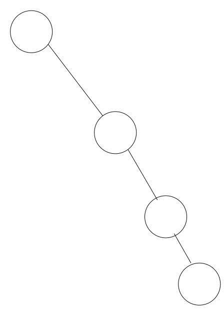
We have to prove that $\mathcal{R} = \mathcal{R}_1 \vee \mathcal{R}_2$. Let us prove first that $\mathcal{R} \leq \mathcal{R}_1 \vee \mathcal{R}_2$ by proving that $\mathcal{R} \leq \mathcal{R}_1$ and that $\mathcal{R} \leq \mathcal{R}_2$.
If $z\mathcal{R}_1 y$, then the two paths in $G(\mathcal{R}_1)$ starting from vertex U_1 and labeled respectively by z and y, reach the same vertex U_i. Therefore, the paths in G starting from the vertex U and labeled by z and y, reach the same vertex V (the one containing U_i), i.e. $z\mathcal{R}y$.
The proof that $\mathcal{R} \leq \mathcal{R}_2$ is analogous.

We have now to prove that $\mathcal{R}_1 \vee \mathcal{R}_2 \leq \mathcal{R}$. Suppose that $z\mathcal{R}y$. We have to prove that $z(\mathcal{R}_1 \vee \mathcal{R}_2)y$. If $z\mathcal{R}y$ then there exist two paths in G, starting from the vertex U and labeled by z and y, that reach same vertex W ; by applying Lemma 7.1, one get that there exists a set T such that $z(\mathcal{R}_1^T \vee \mathcal{R}_2^T)y$. Hence $z(\mathcal{R}_1 \vee \mathcal{R}_2)y$. □

Remark 7.3 The above algorithm gives the join of two general right congruences. In particular it can be applied to a pair of tree congruences $\mathcal{R}(t_1)$ and $\mathcal{R}(t_2)$. In this case the tree congruence corresponding to the output graph is, by Theorem 6.1, $\mathcal{R}(t)$ where $t = gcd(t_1, t_2)$. The right congruences given in Example 7.1 are actually the tree congruences $\mathcal{R}(t_1)$ and $\mathcal{R}(t_2)$ corresponding to the following pair of unlabeled trees:

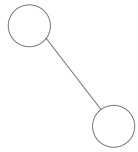


t_1



t_2

and the right congruence output by the algorithm is $\mathcal{R}(t)$, where t is the following tree



t

that is the greatest common divisor of t_1 and t_2.

7.2 Complete sets

Let T be a subset of Σ^*. We recall that $\mathcal{R}^T$ denotes the restriction of $\mathcal{R}$ to the set T, that is, for any $u, v \in \Sigma^*$, $(u, v) \in \mathcal{R}^T$ if and only if $(u, v) \in \mathcal{R}$ and $u, v \in T$. We remark that, even if $\mathcal{R}$ is a right congruence over Σ^*, $\mathcal{R}^T$ is not in general a right congruence, since T is not necessairely a concatenation-closed subset of Σ^*. In any case, $\mathcal{R}^T$ is always an equivalence over T.

Now let us consider two right congruences $\mathcal{R}_1$ and $\mathcal{R}_2$ and their restrictions to a fixed set T. We consider now the join $\mathcal{R}_1^T \vee \mathcal{R}_2^T$. We have that $\mathcal{R}_1^T \vee \mathcal{R}_2^T \geq (\mathcal{R}_1 \vee \mathcal{R}_2)^T$. Notice that in general the previous inequality is strict as shown in the following examples.

Example 7.2 Let Σ be a one-letter alphabet. In this case Σ^* is isomorphic to the additive monoid $I\!N$ of non negative integers. Let $\mathcal{C}_5$ and $\mathcal{C}_8$ be the congruences modulo 5 and modulo 8 respectively, and consider the set $T = \{0, 1, \cdots, 10\}$. The join $\mathcal{C}_5 \vee \mathcal{C}_8$ (and also its restriction $(\mathcal{C}_5 \vee \mathcal{C}_8)^T$) is the trivial congruence $\mathcal{C}_1$ having one class only, whereas $\mathcal{C}_5^T \vee \mathcal{C}_8^T$ has the following two equivalence classes: $C_1 = \{0, 2, 3, 5, 7, 8, 10\}$, $C_2 = \{1, 4, 6, 9\}$.

Example 7.3 Let $\mathcal{R}_1$ and $\mathcal{R}_2$ be the tree congruences in Example 7.1 and let $T' = \{\epsilon, 1, 12, 122, 1222\}$. The join $\mathcal{R}_1 \vee \mathcal{R}_2$ has two equivalence classes and, consequently also $(\mathcal{R}_1 \vee \mathcal{R}_2)^{T'}$ has two equivalence classes, while $\mathcal{R}_1^{T'} \vee \mathcal{R}_2^{T'}$ has three equivalence classes, as one can verify.

We can now give the following definition.

Definition 7.1 *Let $\mathcal{R}_1$ and $\mathcal{R}_2$ be two right congruences over Σ^* and let $P \subset \Sigma^*$. The set P is* complete *with respect to $\mathcal{R}_1$ and $\mathcal{R}_2$ if for any set $T \subseteq \Sigma^*$ such that $T \supseteq P$, $\mathcal{R}_1^T \vee \mathcal{R}_2^T = (\mathcal{R}_1 \vee \mathcal{R}_2)^T$.*

For instance, the sets T and T' given in Examples 7.2 and 7.3 are not complete sets with respect to the pairs of congruences $(\mathcal{C}_5, \mathcal{C}_8)$ and $(\mathcal{R}_1, \mathcal{R}_2)$ defined in the respective examples.

From the definition it follows that any set containing a complete set is also complete.

Remark that if a set P is a singleton, it is always true that $\mathcal{R}_1^P \vee \mathcal{R}_2^P = (\mathcal{R}_1 \vee \mathcal{R}_2)^P$, but P is not necessarly a complete set. Remark also that the empty set can be complete with respect to $\mathcal{R}_1$ and $\mathcal{R}_2$ and this happens when either $\mathcal{R}_1 \vee \mathcal{R}_2 = \mathcal{R}_1$ or $\mathcal{R}_1 \vee \mathcal{R}_2 = \mathcal{R}_2$.

In order to introduce next proposition notice that for any set T containing z and y, $z(\mathcal{R}_1 \vee \mathcal{R}_2)y$ if and only if $z(\mathcal{R}_1 \vee \mathcal{R}_2)^T y$. Moreover if $z(\mathcal{R}_1^T \vee \mathcal{R}_2^T)y$ then $z(\mathcal{R}_1 \vee \mathcal{R}_2)y$,

while in general it is false (as can be seen in previous example) that from the conditions $z(\mathcal{R}_1 \vee \mathcal{R}_2)y$ and T contains z and y it follows that $z(\mathcal{R}_1^T \vee \mathcal{R}_2^T)y$.
Here we give a characterization of complete sets:

Proposition 7.1 *A set P of words over Σ is complete with respect to $\mathcal{R}_1$ and $\mathcal{R}_2$ if and only if for any $z, y \in \Sigma^*$, $z(\mathcal{R}_1 \vee \mathcal{R}_2)y$ implies $z(\mathcal{R}_1^T \vee \mathcal{R}_2^T)y$ where $T = P \cup \{z, y\}$.*

Proof: $(\Rightarrow)$ Suppose that P is complete with respect to $\mathcal{R}_1$ and $\mathcal{R}_2$. Let $z(\mathcal{R}_1 \vee \mathcal{R}_2)y$ and let $T = P \cup \{z, y\}$. By definition, since P is complete and T contains P, $\mathcal{R}_1^T \vee \mathcal{R}_2^T = (\mathcal{R}_1 \vee \mathcal{R}_2)^T$. Since $z(\mathcal{R}_1 \vee \mathcal{R}_2)y$, then $z(\mathcal{R}_1 \vee \mathcal{R}_2)^T y$ and, consequently $z(\mathcal{R}_1^T \vee \mathcal{R}_2^T)y$.

$(\Leftarrow)$ Suppose now that for any $z, y \in \Sigma^*$, if $z(\mathcal{R}_1 \vee \mathcal{R}_2)y$ then $z(\mathcal{R}_1^T \vee \mathcal{R}_2^T)y$ where $T = P \cup \{z, y\}$. We have to prove that P is a complete set. Since for any $S \subseteq \Sigma^*$, $\mathcal{R}_1^S \vee \mathcal{R}_2^S \geq (\mathcal{R}_1 \vee \mathcal{R}_2)^S$, it remains to prove that if $P \subseteq S$ then $\mathcal{R}_1^S \vee \mathcal{R}_2^S \leq (\mathcal{R}_1 \vee \mathcal{R}_2)^S$, i.e. if $z(\mathcal{R}_1 \vee \mathcal{R}_2)^S y$ then $z(\mathcal{R}_1^S \vee \mathcal{R}_2^S)y$.
Let us suppose that $z(\mathcal{R}_1 \vee \mathcal{R}_2)^S y$ for z and y in S. Thus $z(\mathcal{R}_1 \vee \mathcal{R}_2)y$. By hypothesis $z(\mathcal{R}_1^T \vee \mathcal{R}_2^T)y$ where $T = P \cup \{z, y\}$. Since $T \subseteq S$ then $z(\mathcal{R}_1^S \vee \mathcal{R}_2^S)y$, and this concludes the proof. $\square$

The algorithm we are now giving allows to decide whether or not a finite set P is complete with respect to two given tree congruences $\mathcal{R}_1$ and $\mathcal{R}_2$.

The input of the algorithm is the set P and congruence graphs $G(\mathcal{R}_1)$ and $G(\mathcal{R}_2)$; notice that any equivalence class of $\mathcal{R}_1$, of $\mathcal{R}_2$ and of $\mathcal{R}_1 \vee \mathcal{R}_2$ is a rational set of words, as it can be proved using an analogous argument to the one used in Lemma 6.1 to prove that class $[\epsilon]$ is a rational set.

Algorithm C

1. Consider the multiset $\mathcal{M}$ of all the equivalence classes of $\mathcal{R}_1$ and of $\mathcal{R}_2$. For any $w \in P$ find in $\mathcal{M}$ all the elements that contain w and make the union of them.

2. By using the Algorithm J, find the graph $G(\mathcal{R}_1 \vee \mathcal{R}_2)$. Check for any class V of $\mathcal{R}_1 \vee \mathcal{R}_2$ whether there exists an element U of $\mathcal{M}$ such that $V \subseteq U$.

3. If for any class V of $\mathcal{R}_1 \vee \mathcal{R}_2$ the answer to previous check is "YES" then the output is "YES", i.e. P is complete, otherwise P is not complete and the output is "NO" .

Notice that, since a finite union of rational sets is a rational set, the multiset $\mathcal{M}$, after performing step 1, contains only rational sets; therefore it is possible to make checks in steps 2 because inclusion between rational sets is decidable.

Proposition 7.2 *Algorithm C is correct.*

Proof: We prove that P is complete if and only if for any class V of $\mathcal{R}_1 \vee \mathcal{R}_2$ there exists an element U of $\mathcal{M}$ such that $V \subseteq U$, or, equivalently, we prove that P is complete if and only if for any $z, y \in \Sigma^*$, $z(\mathcal{R}_1 \vee \mathcal{R}_2)y$ implies that there exists an element U of $\mathcal{M}$ such that $z, y \in U$. Therefore by Proposition 7.1, in order to prove the proposition it sufficies to prove that the following two statements are equivalents:

1) for any $z, y \in \Sigma^*$, $z(\mathcal{R}_1 \vee \mathcal{R}_2)y \Longrightarrow z(\mathcal{R}_1^T \vee \mathcal{R}_2^T)y$ where $T = P \cup \{z, y\}$;

2) for any $z, y \in \Sigma^*$, $z(\mathcal{R}_1 \vee \mathcal{R}_2)y \Longrightarrow$ there exists an element U of $\mathcal{M}$ such that $z, y \in U$.

But the above two statements are equivalent because both are equivalent to say that for any $z, y \in \Sigma^*$, if $z(\mathcal{R}_1 \vee \mathcal{R}_2)y$ then there exists a sequence $(v_1, v_2, \cdots, v_n)$ of elements of P such that $z\mathcal{R}_{j_1}v_1$, $v_1\mathcal{R}_{j_2}v_2$, $v_2\mathcal{R}_{j_3}v_3, \cdots$, $v_n\mathcal{R}_{j_{n+1}}y$, $j_i \in \{1, 2\}$ (cf. Remark 6.1), and this concludes the proof. □

We now consider complete sets that are prefix-closed. Since any set containing a complete set is also complete, we are interested on "small" prefix-closed complete sets. The existence and the effective construction of such sets is given by the following proposition.

Proposition 7.3 *Any set output of algorithm J is a prefix-closed complete set.*

Proof: Let us consider a run of the algorithm J and let P the output set. By construction, P is a prefix-closed set. In order to prove that P is a complete set, suppose that $z(\mathcal{R}_1 \vee \mathcal{R}_2)y$. Then, by Theorem 7.1 the paths in the graph G (output by the algorithm) starting from the vertex U and labeled by z and y, reach the same vertex V'. Therefore, by Lemma 7.1, $z(\mathcal{R}_1^T \vee \mathcal{R}_2^T)y$ where $T = P \cup \{z, y\}$. By Proposition 7.1, P is complete with respect $\mathcal{R}_1$ and $\mathcal{R}_2$, and this concludes the proof. □

The following corollary gives an upper bound on the size of a prefix-closed complete set of minimal size. Recall that $i(\mathcal{R})$ denotes the index of the congruence $\mathcal{R}$, i.e. the number of classes of $\mathcal{R}$.

Corollary 7.1 *Given two right congruences $\mathcal{R}_1$ and $\mathcal{R}_2$, there exists complete prefix-closed set P such that*

$$|P| = i(\mathcal{R}_1) + i(\mathcal{R}_2) - i(\mathcal{R}_1 \vee \mathcal{R}_2),$$

where $|P|$ is the cardinality of P.

Proof: Consider a set P output of the algorithm J. Its cardinality $|P|$ is equal to the number of steps performed by the algorithm J. Since, in each step, operation *Merge* unifies two vertices, we have that $|P| = i(\mathcal{R}_1) + i(\mathcal{R}_2) - i(\mathcal{R}_1 \vee \mathcal{R}_2)$. □

In the special case of $|\Sigma| = 1$, Σ^* is isomorphic to the set $I\!N$ of natural numbers and a prefix closed subset T of $I\!N$ is of the form $T = \{0, 1, \cdots, m-1\}$.

Corollary 7.2 *Let $|\Sigma| = 1$. If $m \geq i(\mathcal{R}_1) + i(\mathcal{R}_2) - i(\mathcal{R}_1 \vee \mathcal{R}_2)$ then T is complete.*

Proof: If $|\Sigma| = 1$, the algorithm J is deterministic (cf Remark 7.2). Then there exists a unique set P output of the algorithm J and its cardinality is $|P| = i(\mathcal{R}_1) + i(\mathcal{R}_2) - i(\mathcal{R}_1 \vee \mathcal{R}_2)$. Thus $P = \{0, 1, \cdots, i(\mathcal{R}_1) + i(\mathcal{R}_2) - i(\mathcal{R}_1 \vee \mathcal{R}_2)\}$. Since any set containing P is complete, the thesis follows. $\square$

8 A Periodicity Theorem

In this section we consider trees having two different periodicities. The notions and the results given in previous sections converge in the statement and in the proof of our main result that generalizes the Fine and Wilf's theorem to trees.

We first give a characterization of periodicity in terms of congruences, generalizing Proposition 2.1

Proposition 8.1 *Let τ be a labeled tree and t_0 be an unlabeled tree. Then, τ has period t_0 if and only if for each pair of nodes $x, y \in dom(\tau)$ such that $x\mathcal{R}(t_0)y$, $\tau(x) = \tau(y)$.*

Proof: ($\Rightarrow$) If τ has period t_0, then there exists a labeled tree τ_0 such that $sh(\tau_0) = t_0$, and a tree $\rho \in \tau_0^{\#}$ such that $\tau \subseteq \rho$. If $x\mathcal{R}(t_0)y$ then, by definition of $\mathcal{R}(t_0)$, $x, y \in B(t_0)^*z$ for some $z \in dom(t_0) = dom(\tau_0)$. In other words, all the elements in the class $B(t_0)^*z$ are the ones that are in the same position with respect to the occurrence of τ_0 in ρ to which they respectively belong. Since $\tau \subseteq \rho \in \tau_0^{\#}$, for each element $x \in B(t_0)^*z$, we have that $\tau(x) = \tau_0(z)$. In particular $\tau(x) = \tau(y)$.

($\Leftarrow$) Suppose that for all $x, y \in dom(\tau)$ such that $x\mathcal{R}(t_0)y$ we have that $\tau(x) = \tau(y)$. Notice that given two arbitrary unlabeled trees t_1 and t_2, we can find an unlabeled tree $s \in t_2^{\#}$ such that $t_1 \subseteq s$. In our case we can consider $t = sh(\tau)$ as prefix of an unlabeled tree $s \in t_0^{\#}$. In this way, $x\mathcal{R}(t_0)y$ means that x and y are in the same position relatively to the occurrences of t_0 in s to which they respectively belong. Since, by hypothesis, all the elements of a fixed class $B(t_0)^*z$ have the same label, we can define the tree τ_0 in the following way:

$$dom(\tau_0) = dom(t_0), \quad \forall z \in dom(\tau_0) \;\; \tau_0(z) = \tau(z).$$

It is easy to verify that τ is prefix of an element of $\tau_0^{\#}$, that is, τ has period t_0. $\square$

Given a labeled tree τ we define the *coarsest right congruence compatible with the labeling of* τ, denoted by $\mathcal{R}_\tau$, as follows: $\forall x, y \in dom(\tau)$, $x\mathcal{R}_\tau y \Leftrightarrow \tau(x) = \tau(y)$.

Recall that $\mathcal{R}^D$ denotes the restriction of an equivalence $\mathcal{R}$ to the set D. To simplify the notations, let us denote by D the domain of the tree τ. Then Proposition 8.1 can be re-stated as follows.

Corollary 8.1 *A tree $\tau \in A^{\#}$ has period t_0 if and only if $\mathcal{R}_\tau \leq (\mathcal{R}(t_0))^D$.*

At the end of Section 4 we stressed that, in order to generalize the Fine and Wilf theorem to trees we have to investigate what "greatest common divisor" and "suffiently long with respect to two periodicities" mean in the case of trees. In Section 5 we defined the greatest common divisor of two trees. The formalization of the latter notion in terms of completeness is given in the following definition.

Definition 8.1 *Let t, t_1 and t_2 be unlabeled trees. We say that t is complete with respect to t_1 and t_2 if $dom(t)$ is complete with respect to the congruences $\mathcal{R}(t_1)$ and $\mathcal{R}(t_2)$. Moreover we say that a labeled tree τ is complete with respect to t_1 and t_2 if its shape $sh(\tau)$ satisfies the same condition.*

Remark 8.1 In the special case of unary trees, i.e. in the case of words, t_1 and t_2 are uniquely specified by their sizes p_1 and p_2 respectively, $\mathcal{R}(t_1)$ and $\mathcal{R}(t_2)$ coincide with the equivalence $mod\ p_1$ and $mod\ p_2$ respectively (cf. Remark 6.2 and Lemma 6.5), τ corresponds to a word of length m and it is complete with respect to p_1 and p_2 if $m \geq p_1 + p_2 - gcd(p_1, p_2)$ (cf. Corollary 7.2).

Theorem 8.1 (Fine and Wilf's Theorem for Trees) *Let τ be a labeled tree having periods t_1 and t_2. If τ is complete with respect to t_1 and t_2 then τ has also period $gcd(t_1, t_2)$.*

Proof: Let us denote by $D = dom(\tau)$. By the hypothesis and by Corollary 8.1, $\mathcal{R}_\tau \leq (\mathcal{R}(t_1))^D$ and $\mathcal{R}_\tau \leq (\mathcal{R}(t_2))^D$. Then, by definition of join, $\mathcal{R}_\tau \leq (\mathcal{R}(t_1))^D \vee (\mathcal{R}(t_2))^D$. But D is complete with respect to $\mathcal{R}(t_1)$ and $\mathcal{R}(t_2)$, then $(\mathcal{R}(t_1))^D \vee (\mathcal{R}(t_2))^D = (\mathcal{R}(t_1) \vee \mathcal{R}(t_2))^D$. This implies that $\mathcal{R}_\tau \leq (\mathcal{R}(t_1) \vee \mathcal{R}(t_2))^D$. But by Theorem 6.1 $(\mathcal{R}(t_1) \vee \mathcal{R}(t_2))^D = \mathcal{R}(gcd(t_1, t_2))^D$, then we have that $\mathcal{R}_\tau \leq \mathcal{R}(gcd(t_1, t_2))$. This means that τ has period $gcd(t_1, t_2)$. $\square$

Remark 8.2 In the special case of unary trees we obtain, as a corollary of Theorem 8.1, the classical Fine and Wilf theorem for words (cf. Remark 8.1).

The tightness of previous theorem is proved in the case of non trivial periods (cf. Section 4).

Proposition 8.2 *Let t_1, t_2 and t be unlabeled trees such that t_1 or t_2 are prefixes of t and such that t is not complete with respect to t_1 and t_2. Then, there exists a (labeled) tree τ, with shape $sh(\tau) = t$, having (non trivial) periods t_1 and t_2 but not period $gcd(t_1, t_2)$.*

Proof:

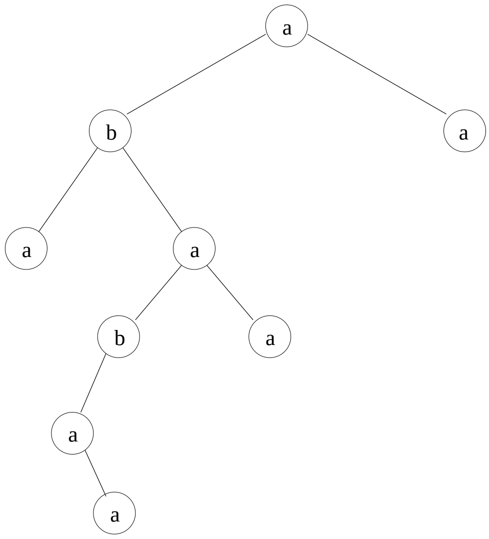
Let Σ be the structure alphabet of t, t_1, t_2. Let us denote by $D = dom(t)$. By Definition 7.1 and Definition 8.1, there exists a set $X \subseteq \Sigma^*$ such that $dom(t) \subseteq X$ and $(\mathcal{R}(t_1))^X \vee (\mathcal{R}(t_2))^X \neq (\mathcal{R}(t_1) \vee \mathcal{R}(t_2))^X$. This means that there exist $x, y \in X$ such that $(x, y) \in \mathcal{R}(t_1) \vee \mathcal{R}(t_2)$, but $(x, y) \notin (\mathcal{R}(t_1))^X \vee (\mathcal{R}(t_2))^X$. Since t_1 (or t_2) is a prefix of t, we can pick x' and $y' \in D$ such that $x\mathcal{R}(t_1)x'$ and $y'\mathcal{R}(t_1)y$ (or $x\mathcal{R}(t_2)x'$ and $y'\mathcal{R}(t_2)y$); this fact and Remark 6.1 imply that also $(x', y') \notin (\mathcal{R}(t_1))^X \vee (\mathcal{R}(t_2))^X$, i.e. $(\mathcal{R}(t_1))^D \vee (\mathcal{R}(t_2))^D \neq (\mathcal{R}(t_1) \vee \mathcal{R}(t_2))^D$.

Let $\mathcal{A}$ be the set of equivalence classes of $(\mathcal{R}(t_1))^D \vee (\mathcal{R}(t_2))^D$ and let $\Theta : \mathcal{A} \to A$ be a bijection between $\mathcal{A}$ and an alphabet A. Let τ be a labeled tree with $dom(\tau) = dom(t) = D$, defined as follows: For any $x \in D$, $\tau(x) = \Theta([x])$, where $[x]$ denotes the element of $\mathcal{A}$ containing x.

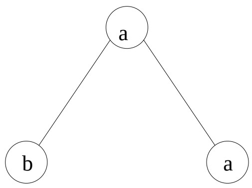
Clearly $\mathcal{R}_\tau = (\mathcal{R}(t_1))^D \vee (\mathcal{R}(t_2))^D$.

By Theorem 6.1 $(\mathcal{R}(t_1) \vee \mathcal{R}(t_2))^D = \mathcal{R}(gcd(t_1, t_2))^D$. Hence we have that $\mathcal{R}_\tau \not\leq \mathcal{R}(gcd(t_1, t_2))$. This implies, by Corollary 8.1, that τ has not period $gcd(t_1, t_2)$. $\square$

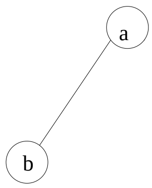
Example 8.1 Consider the following trees.



τ



τ_1



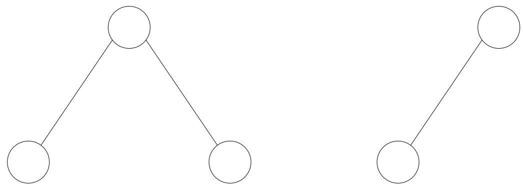
τ_2

It easy to verify that the tree τ has two different (non trivial) periods, $t_1 = sh(\tau_1)$ and $t_2 = sh(\tau_2)$, but it does not have period $gcd(t_1, t_2) = \pi$, the punctual tree. In fact τ is not complete with respect to t_1 and t_2.

Notice that if we want to make τ a complete tree with respect to t_1 and t_2 we should add the node 21. But, in this case, τ cannot have both periods t_1 and t_2 without having also period $gcd(t_1, t_2) = \pi$. In fact, in order to have period t_1, the node 21 must be labeled with a, whereas the period t_2 obliges vertex 21 to be labeled with a b. The only possibility to have both periods t_1 and t_2 would be if $a = b$ and, in such a case, the tree τ would have also period $\pi = gcd(t_1, t_2)$. Recall that the notion of completeness of a tree is effectively decidable by algorithm C.

Before concluding this paper, we remark that periodicity on trees is a phenomenon quite different from the corresponding one on words. This is true despite we defined periodicity on trees in a way that in the case of unary trees it is perfectly equivalent to the classical notion of periodicity on strings.

Although Corollary 7.1 guarantees the existence of "small" complete trees, however there exist particular pairs of unlabeled trees t_1 and t_2 to which respect we can find trees with an arbitrairely large size that are not complete. For example, consider the following trees t_1 and t_2.



t_1 t_2

All the trees whose domain is the set of all the prefixes of a word of the form

$$1^{d_1}21^{d_2}2\ldots 1^{d_n}2$$

or all the trees whose domain is the set of all prefixes of a word of the form

$$1^{d_1}21^{d_2}2\ldots 1^{d_n}21^{p}$$

where $d_1, d_2, \ldots, d_n$ are odd integers and p is an even integer, are not complete with respect to t_1 and t_2. Then, by choosing the d_i's, p and n sufficiently large, we obtain trees with an arbitrairely large size, that are not complete with respect to t_1 and t_2. This implies that we can have infinite trees that have two different periods t_1 and t_2 and not have period $gcd(t_1, t_2)$.

This situation never occurs in the case of words. In fact, in that case, the tree-congruences are only the congruences modulo p and the unlabeled words (or non-negative integers) that are not complete with respect to two given congruences modulo p and modulo q are exactly all the unlabeled words whose size is less or equal to $p+q-gcd(p,q)$. Therefore we cannot have words with an arbitrairely long size that are not complete with respect to t_1 and t_2.

We remark that it is possible to define "weaker" notion of periodicity for trees by using congruences that are not tree congruences. Also in this case we can state a "periodicity theorem", based on the results of Section 7, that have been proved without the restrictive hypothesis that the congruences are tree congruences (cf. [5] and [6]).

Acknoledgements

We thank an anonymous referee for his careful reading and suggestions.

References

[1] A. Aho, J. E. Hopcroft and J. D. Ullman. *The Design and the Analysis of Computer Algorithms.* Addison-Wesley, Reading, MA 1974.

[2] J. Berstel, D. Perrin. *Theory of Codes.* Academic Press, New York 1985.

[3] M. G. Castelli, D. Guaiana, S. Mantaci. Counting prime trees. Preprint of Dipartimento di Matematica ed Applicazioni dell'Universitá di Palermo No. 18, Marzo 1996.

[4] N. J. Fine, H. S. Wilf. Uniqueness Theorem for Periodic Functions. *Proc. Am. Mathematical Society* No. 16, (1965) pp.109-114.

[5] D. Giammarresi, S. Mantaci, F. Mignosi, A. Restivo. A periodicity theorem for trees. *Proc. 13th World Computer Congress - IFIP '94*, Hambourgh, Germany, 1994; vol. A-51 pp. 473–478. *Elsevier Science B.V. (North-Holland)*, 1994.

[6] D. Giammarresi, S. Mantaci, F. Mignosi, A. Restivo. Congruences, automata and periodicities. Proc. of the workshop *Semigroups, Automata and Languages*, Porto, June 1994. J.Almeida and P.Silva Eds. pp.125–135. World Scientific Publishing Co., 1995.

[7] R. Giancarlo, F. Mignosi. Generalizations of the periodicity Theorem of Fine and Wilf. *Proc. of CAAP 94*, Lecture Notes in Computer Science, no. 787, pp.130–141. Springer Verlag, 1994.

[8] L. J. Guibas, A. M. Odlyzko. Periods in Strings. Journal of Combinatorial Theory, Series A No. 30 (1981), pp. 19-42.

[9] F. Harary. *Graph Theory.* Addison-Wesley, Reading, MA 1969.

[10] J. E. Hopcroft, J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, Reading, MA 1979.

[11] D. E. Knuth. *The art of computer programming* vol.1. Addison-Wesley, Reading, MA 1968.

[12] M. Lothaire. *Combinatorics on Words.* Addison-Wesley, Reading, MA 1983.

[13] S. Mantaci, A. Restivo. Equations on trees. *Proceedings of MFCS'96*, Lecture Notes in Computer Science, no. 1113, pp. 443–456. Springer Verlag, 1996.

[14] M. Nivat. Binary tree codes. In *Tree Automata and Languages*, M. Nivat and A. Podelski Eds. pp. 1–19. Elsevier Science Publ. B.V. 1992.

[15] K. G. Subramanian, R. Siromoney, L. Mathew. Lyndon trees. *Theoretical Computer Science*, No 106, pp. 373-383, 1992.