An Arithmetic for Rooted Trees*

Fabrizio Luccio

Dipartimento di Informatica, University of Pisa, Italy
luccio@di.unipi.it

Abstract

We propose a new arithmetic for non-empty rooted unordered trees simply called trees. After discussing tree representation and enumeration, we define the operations of tree addition, multiplication, and stretch, prove their properties, and show that all trees can be generated from a starting tree of one vertex. We then show how a given tree can be obtained as the sum or product of two trees, thus defining *prime trees* with respect to addition and multiplication. In both cases we show how primality can be decided in time polynomial in the number of vertices and prove that factorization is unique. We then define negative trees and suggest dealing with tree equations, giving some preliminary examples. Finally we comment on how our arithmetic might be useful, and discuss preceding studies that have some relations with ours. The parts of this work that do not concur to an immediate illustration of our proposal, including formal proofs, are reported in the Appendix.

To the best of our knowledge our proposal is completely new and can be largely modified in cooperation with the readers. To the ones of his age the author suggests that "many roads must be walked down before we call it a theory".

1998 ACM Subject Classification E.1 Data Structures, G.2.0 [Discrete Mathematics] General, G.2.2 [Discrete Mathematics] Graph Theory

Keywords and phrases Arithmetic, Rooted tree, Prime tree, Tree equation

Digital Object Identifier 10.4230/LIPIcs.FUN.2016.23

1 Basic properties and notation

- We refer to **rooted unordered trees** simply called trees. Our trees are non empty. **1** denotes the tree containing exactly one vertex, and is the basic element of our theory.
- In a tree T, $\boldsymbol{r}(T)$ denotes the root of T; $x \in T$ denotes any of its vertices; n_T and e_T respectively denote the numbers of vertices and leaves. A subtree is the tree composed of a vertex x and all its descendants in T. The subtrees routed at the children of x are called subtrees of x. s_T denotes the number of subtrees of $\boldsymbol{r}(T)$.
- A tree T can be represented as a binary sequences S_T (the original reference for ordered trees is [11]). In our scheme T is traversed in left to right preorder inserting 1 in the sequence for each vertex encountered, and inserting 0 for each move backwards. Then S_T is composed of $2n$ bits as shown in Figure 1, and has a balanced parenthesis recursive structure 1 S_1 . . . S_k 0 where the S_i are the sequences representing the subtrees of $\boldsymbol{r}(T)$. The sequences for tree **1** is 10. Note that all the prefixes of S_T have more 1's than 0's except for the whole sequence that has as many 1's as 0's.

Since T is unordered, the order in which the subsequences S_i appear in S_T is immaterial (i.e., in general many different sequences represent T). However a *canonical form* for trees is

* This work was partially supported by MIUR of Italy under PRIN 2012C4E3KT national research project AMANDA – Algorithmics for Massive and Networked DAta.

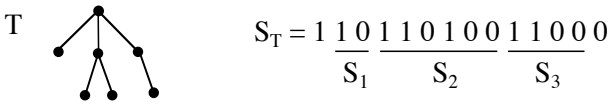

Figure 1 Tree representation as a binary sequence. S_1, S_2, S_3 represent the subtrees of the root of T.

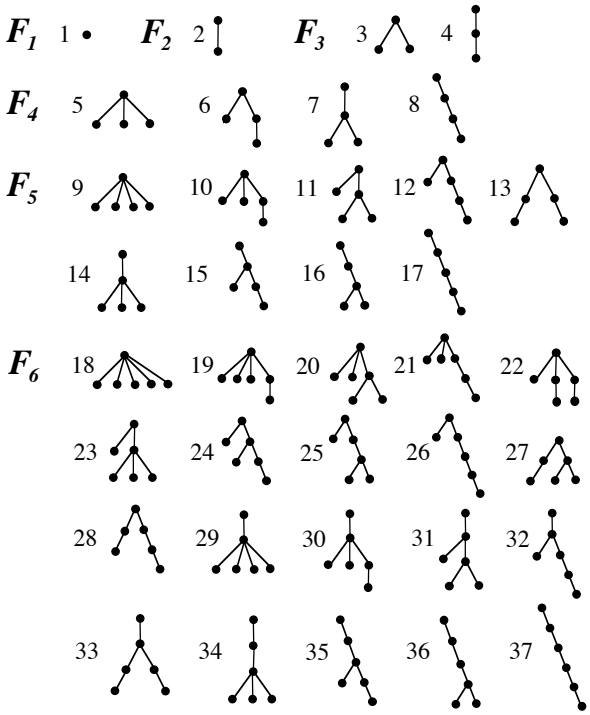

Figure 2 The canonical families of trees $\mathcal{F}_1$ to $\mathcal{F}_6$ and the corresponding tree enumeration.

established so that their sequences will be uniquely determined, and will result to be ordered for increasing values if they are interpreted as binary numbers. To this end the trees are grouped into consecutive families $\mathcal{F}_1, \mathcal{F}_2, \ldots$ as shown in Figure 2, where $\mathcal{F}_i$ contains the trees of i vertices. So the trees are ordered for increasing number of vertices, and inside each family the ordering is determined by the canonical form as follows. Trees and sequences are then numbered with increasing natural numbers (see Appendix A for the sequences of the trees of $\mathcal{F}_1$ to $\mathcal{F}_6$).

- If the sequences are interpreted as binary numbers, for two trees U, T with $n_U < n_T$ we have $S_U < S_T$ because the initial character of each sequence is 1 and S_U is shorter than S_T. This is consistent with the property that the trees of $\mathcal{F}_{n_U}$ precede the trees of $\mathcal{F}_{n_T}$ in the ordering.
- The families $\mathcal{F}_1, \mathcal{F}_2$ contain one tree each numbered 1, 2.
- The ordering of the trees in $\mathcal{F}_{n>2}$ is based on the ordering of the preceding families. Consider the multisets of positive integers whose sum is $n-1$. E.g., for $n = 6$ these multisets are: 1,1,1,1,1 - 1,1,1,2 - 1,1,3 - 1,2,2 - 1,4 - 2,3 - 5 ordered for non-decreasing value of the digits left to right. Each multiset corresponds to a group of consecutive trees in $\mathcal{F}_n$, where the digits in the multiset indicate the number of vertices of the subtrees of the root. For $\mathcal{F}_6$ in Figure 2, multiset 1,1,1,1,1 refers to tree 18; multiset 1,1,1,2 refers to

tree 19; multiset 1,1,3 refers to trees 20 and 21 that have the two trees of $\mathcal{F}_3$ as third subtree, following the ordering in $\mathcal{F}_3$; ...; multiset 2,3 refers to trees 27 and 28; the last multiset 5 refers to trees 29 to 37 whose roots have only one child.

So the first tree in $\mathcal{F}_n$ is the one of height 2 with $n-1$ subtrees of the root of one vertex each and sequence 1 1 0 1 0 1 0 . . . 1 0 0; and the last tree is the "chain" of n vertices and sequence 1 1 . . . 1 0 0 . . . 0. As said the binary sequences representing the trees in $\mathcal{F}_n$ are ordered for increasing values, see Appendix A.

Many of these trees (not necessarily all) of each family $\mathcal{F}_n$ can be generated from the ones in $\mathcal{F}_{n-1}$ using the following:

Doubling Rule DR. From each tree T in $\mathcal{F}_{n-1}$ build two trees T_1, T_2 in $\mathcal{F}_n$ by adding a new vertex as the leftmost child of $\boldsymbol{r}(T)$, or adding a new root and appending T to it as a unique subtree.

For example the four trees of $\mathcal{F}_4$ in Figure 2 can be built by **DR** from the two trees of $\mathcal{F}_3$. The nine trees of $\mathcal{F}_5$ can be built by **DR** from the four trees of $\mathcal{F}_4$, with the exception of tree 13. The twenty trees of $\mathcal{F}_6$ can be built by **DR** from the nine trees of $\mathcal{F}_5$, with the exception of trees 27 and 28. In fact the number of extra trees that cannot be built with **DR** increases sharply with n. Letting f_n denote the number of trees in $\mathcal{F}_n$ we immediately have $f_n \geq 2^{n-2}$ for $n \geq 2$. But a deep analysis [3, 8] has shown that the asymptotic value of this function is much higher, and can be approximated as:

$$f_n \sim 0.44 \cdot 2.96^n \cdot n^{-3/2}. \tag{1}$$

Then the minimum length of the sequences representing the trees of $\mathcal{F}_n$ is given approximately by:

$$\log_2(0.44 \cdot 2.96^n \cdot n^{-3/2}) \sim 1.57\, n - 1.5 \log_2 n - 1.19$$

much less than the $2n$ bits of our proposal. We only note that for $n \geq 2$ all the binary sequences representing our trees begin with two 1's and end with two 0's (see the listing in the Appendix A), then these four digits could be removed, leaving a sequence of $2n-4$ bits to represent a tree. We shall see that our representation is amenable at working easily on the trees, so we maintain it, leaving the construction of a shorter efficient coding as a challenging open problem.

An arbitrary tree T can be transformed into its canonical form with Algorithm CF of Figure 3. An elementary analysis shows that the algorithm is correct and each of its steps **1**, **2** can be executed in total $O(n^2)$ time. The algorithm may possibly be improved, however our present aim is just showing that the problem can be solved in polynomial time.

2 Operators and tree generation

Our basic operations are addition (symbol +) and multiplication (symbol $\cdot$, or simple concatenation) defined as follows. Referring to Figure 4, let A, B be two arbitrary trees:

Addition. $T = A + B$ is built by merging the two roots $\boldsymbol{r}(A)$, $\boldsymbol{r}(B)$ into a new root $\boldsymbol{r}(T)$. That is the subtrees of A and B (if any) become the subtrees of $\boldsymbol{r}(T)$. We have $A + \mathbf{1} = \mathbf{1} + A = A$.

algorithm CF(T)

1. forany vertex $x \in T$
  count the number of vertices $n_1, \ldots n_k$ of its subtrees;
  reorder these subtrees for non decreasing values of the n_i;
  let $G_1, \ldots, G_r$ be the groups of subtrees with the same number $g_1, \ldots, g_r$ of vertices, with all $g_i > 2$;
  // reordering is necessary but not sufficient for having T in canonical form
  // the trees in all G_i must be be arranged in canonical order

2. forany $x \in T$, down-top from the vertices closest to the leaves
  forany group $G_i = \{T_1, \ldots, T_s\}$
    compute the representing sequences $S_1, \ldots, S_s$;
    order $S_1, \ldots, S_s$ for increasing binary value;
    permute $T_1, \ldots, T_s$ accordingly.

Figure 3 Structure of Algorithm CF for transforming an arbitrary tree T of n vertices in canonical form. CF requires polynomial time in n.

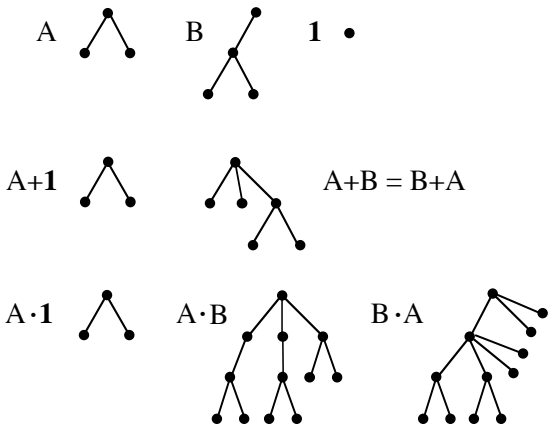

Figure 4 Examples of addition and multiplication.

Multiplication. $T = A \cdot B$ is built by merging $\boldsymbol{r}(B)$ with each vertex $x \in A$ so that all the subtrees of $\boldsymbol{r}(B)$ become new subtrees of x. We have $A \cdot \mathbf{1} = \mathbf{1} \cdot A = A$.

In both operations it is immaterial in which order the subtrees are attached to the new parents. We also define the operation stretch (symbol over-bar) whose interest will be made clear in the following:

Stretch. $T = \bar{A}$ consists of a new root $\boldsymbol{r}(T)$ with A attached as a subtree.

In the notation stretch has precedence over multiplication, and multiplication has precedence over addition. Two propositions immediately follow:

▶ **Proposition 1.** *For $T = A + B$ we have $n_T = n_A + n_B - 1$. For $T = A \cdot B$ we have $n_T = n_A n_B$. For $T = \bar{A}$ we have $n_T = n_A + 1$.*

▶ **Proposition 2.** *Addition is commutative and associative. That is $A + B = B + A$ and $(A + B) + C = A + (B + C)$.*

For a positive integer $k > 1$ and a tree A we can define the product $T = kA$ (not to be confused with the product of trees) as the sum of k copies of A. Due to Propositions 2 and 1, the k copies of A can be combined in any order and we have $n_T = k\,n_A - k + 1$. For any given k, the trees of n_T vertices obtained as a product kA are only f_{n_A}, that is they constitute an exponentially small fraction of all the trees in $\mathcal{F}_{n_T}$. For example the "even" trees (k even) are a small minority among all the trees with the same number of vertices. Similarly we can define the stretch-product $U = k\bar{A}$ as A stretched k times, and we have $n_U = n_A + k$. Again for any given k, the trees of n_U vertices obtained as a stretch-product $k\bar{A}$ are only f_{n_A} and constitute an exponentially small fraction of all the trees in $\mathcal{F}_{n_U}$.

Studying associativity and commutativity in tree multiplication is more complicated. From the definition of multiplication we have with simple reasoning:

▶ **Proposition 3.** *Multiplication is associative.*

That is $(A \cdot B) \cdot C = A \cdot (B \cdot C)$. For a positive integer $k > 1$ and a tree A we can define the power $T = A^k$ as the product of k copies of A. Due to Propositions 3 and 1 the multiplications can be done in any order and we have $n_T = n_A^k$. Again, for any given k, the different trees of n_T vertices obtained as $T = A^k$ are only f_{n_A}.

Multiplication is generally not commutative. For a product $A \cdot B$ we consider two cases $n_A = n_B$ and $n_A > n_B$ ($n_B > n_A$ is symmetric), for which we pose the conditions below. Recall that, for any tree X, e_X and s_X respectively denote the number of leaves of X and the number of subtrees of $\boldsymbol{r}(X)$. For $n_A > n_B$ our conditions are only necessary.

▶ **Proposition 4** (Proof in the Appendix B)**.** *For $n_A = n_B$ we have $A \cdot B = B \cdot A$ if and only if $A = B$.*

▶ **Proposition 5** (Proof in the Appendix B)**.** *For $n_A > n_B$ we have $A \cdot B = B \cdot A$ only if the following conditions are all verified:*

(i) *$n_a/e_A = n_B/e_B$;*
(ii) *B is a proper subtree of A;*
(iii) *if $s_A \geq s_B$ all the subtrees of $\boldsymbol{r}(B)$ must be equal to some subtrees of $\boldsymbol{r}(A)$.*

The trees $3 = A$ and $2 = B$ of Figure 2 do not comply with conditions (i) and (ii) of Proposition 5 and we have $A \cdot B = 22$ different from $B \cdot A = 20$. Commutative products are in fact quite rare. An example with $A \cdot B = B \cdot A$ is shown in Figure 5 where the three conditions of Proposition 5 are verified. In this particular case we have $A = B^2$ hence $A \cdot B = B^3$. Finally multiplication is generally not distributive over addition. From Proposition 1 we can immediately prove:

▶ **Proposition 6.** *$(A + B) \cdot C = A \cdot C + B \cdot C$ if and only if $C = \mathbf{1}$.*

A basic fact about our arithmetic is that all trees can be generated by the single *generator* **1** using addition and stretch.[1] Namely:

- Tree **1** is the generator of itself.
- Assuming inductively that each of the trees in $\mathcal{F}_i$ with $1 \leq i \leq n-1$ can be generated by the trees of the preceding families, then each tree T in $\mathcal{F}_n$ can also be generated. In fact if $\boldsymbol{r}(T)$ has one subtree T_1 then T can be generated as $\bar{T}_1$; if $\boldsymbol{r}(T)$ has $k \geq 2$ subtrees

[1] Stretch been included in the operation set to allow the construction of all trees starting from a finite set of generators. The reader may check that addition and multiplication, or stretch and multiplication, are not sufficient for this purpose. It may be noted that a similar need has arisen in tree algebras (not arithmetic) studied in language theory, e.g. see [7].

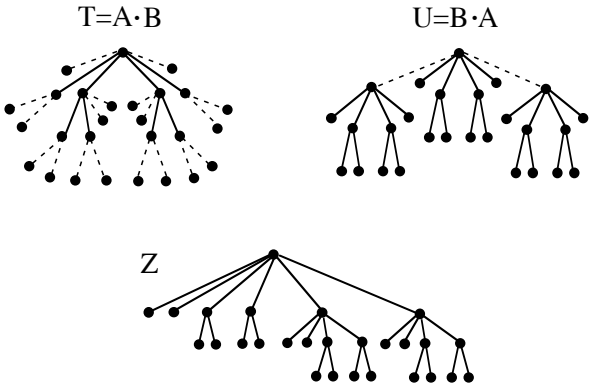

Figure 5 An example of commutative product $A \cdot B = B \cdot A$ for B subtree of A. The two trees are shown in solid and dashed lines, respectively. Z is $A \cdot B$ in canonical form.

$T_1, T_2, \ldots, T_k$ then T can be generated as $U + V$ where U is T deprived of T_k and V is T deprived of $T_1, T_2, \ldots, T_{k-1}$.

3 Prime trees

In the arithmetic of natural numbers the basic operations are addition and multiplication, with $x + 0 = x$ and $x \cdot 1 = x$. Prime numbers under addition have no sense, since all x greater than 1 can be constructed as the sum of two smaller terms other than 0 and x. In our arithmetic for trees, instead, primality occurs in relation with addition and multiplication. In this whole section we refer to trees T with $n_T > 1$. We pose:

▶ **Definition 7.**

(i) T is *prime under addition (*shortly *add-prime)* if can be generated by addition only if the terms are **1** and T (tree **1** has a companion role of integer 0 in $\mathbb{N}$).

(ii) T is *prime under multiplication (*shortly *mult-prime)* if can be generated by multiplication only if the factors are **1** and T.

The definition of mult-primality is the natural counterpart of the one of primality in $\mathbb{N}$. As it may be expected its consequences are not easy to study. For add-primality, instead, the situation is quite simple. We have:

▶ **Proposition 8** (Proof in the Appendix B)**.** *T is add-prime if and only if $\boldsymbol{r}(T)$ has only one subtree.*

As a consequence of Proposition 8 deciding if a tree is add-prime is computationally "easy". From Proposition 8 , and from the construction given in the **DR** rule we have:

▶ **Proposition 9.** *For $n \geq 2$ the number of add-prime trees is f_{n-1}.*

From Equation (1) we have: $f_{n-1}/f_n \to \sim 0.34$ for $n \to \infty$, that is the add-prime trees in $\mathcal{F}_n$ are asymptotically about one third of the total. Each of the remaining *add-composite* (i.e., non add-prime) trees T can be uniquely factorized in s_T factors.

For mult-primality we start with two immediate statements respectively derived from Proposition 1, and from the definition of multiplication for trees with at least two vertices:

▶ **Proposition 10.** *If n is a prime number all the trees with n vertices are mult-prime.*

▶ **Proposition 11.** *If $\boldsymbol{r}(T)$ has only one subtree then T is mult-prime.*

The converse of Propositions 10 and 11 do not hold in our arithmetic. That is if n_T is a composite number or $\boldsymbol{r}(T)$ has more than one subtree, tree T may still be mult-prime. In a sense mult-prime trees are more numerous than primes in $\mathbb{N}$. For example out of the twenty trees in $\mathcal{F}_6$ (see Figure 2) only trees 20, 22, 24, and 28 are *mult-composite* (i.e. non mult-prime), as they can be built as $2 \cdot 3$, $3 \cdot 2$, $4 \cdot 2$, and $2 \cdot 4$, respectively.

Since if n_T is prime T is mult-prime, and the problem of deciding if n_T is prime is polynomial in $\log n_T$, deciding if T is mult-prime is straightforward for n_T prime. However the problem is difficult for n_T composite because T may be mult-prime or mult-composite. An algorithm for n_T composite may consist of building all the products $A \cdot B$ and $B \cdot A$ of two trees A, B of a, b vertices respectively for all the factorizations of n_T as $a \cdot b$, and comparing T with these products looking for a match. However this method is impracticable unless n_T is very smal, then we must find a different way to decide mult-primality. To this end consider a property of product trees based on the observation that, if $T = A \cdot B$, all the subtrees of $\boldsymbol{r}(B)$ are also subtrees of $\boldsymbol{r}(T)$. Namely:

▶ **Proposition 12** (Proof in the Appendix B)**.** *Let $T = A \cdot B$ with $A, B \neq \mathbf{1}$, and let Y be a subtree of $\boldsymbol{r}(B)$ with maximum number n_Y of vertices. Then the subtrees of $\boldsymbol{r}(B)$ are exactly the subtrees of $\boldsymbol{r}(T)$ with at most n_Y vertices.*

In the mult-composite tree Z of Figure 5, if the first subtree of $\boldsymbol{r}(Z)$ (containing one vertex) is a subtree of maximal cardinallity of one of the factors, B in this case, then B consists of a root plus the first two subtrees of $\boldsymbol{r}(Z)$. Similarly, if the third subtree of $\boldsymbol{r}(Z)$ is a subtree of maximal cardinality of one of the factors, A in this case, then A consists of a root plus the first four subtrees of $\boldsymbol{r}(Z)$. We pose:

▶ **Notation 13.** *For an arbitrary tree T:* (i) *$G_1, \ldots, G_r$ are the groups of subtrees of $\boldsymbol{r}(T)$ with the same number $g_1, \ldots, g_r$ of vertices, $g_1 < g_2 < \cdots < g_r$;* (ii) *$H_i = \bigcup_{j=1}^{i} G_j$, $1 \leq i \leq r$, i.e. each H_i is the group of subtrees of $\boldsymbol{r}(T)$ with up to g_i vertices.*

Based on Propositions 12 and Notation 131 we can build the primality Algorithm MP of Figure 6 that requires polynomial time in the number of vertices. Since all trees with a prime number n of vertices are mult-prime, MP is intended for testing trees with n composite. However MP works for all trees and can always be applied to avoid a preliminary test for the primality of n.

▶ **Proposition 14.** *Mult-primality of any tree T can be decided in time polynomial in n_T.*

Proof. Proof in the Appendix B, based on the analysis of algorithm MP. ◀

Note that if T is mult-composite Algorithm MP allows to find a pair of factors A, B at no extra cost, with B mult-prime. In fact, if a cycle i of step **3** is completed, the algorithm is interrupted on the **return** statement and the group H_i contains exactly the subtrees of $\boldsymbol{r}(B)$, while the tree Z is reduced to A. In particular B is the last factor of a product of mult-prime trees, with $T = T_1 \cdot T_2 \cdots T_k \cdot B$. If Algorithm MP is not interrupted with the **return** statement, all these factors can be detected. As a consequence we have:

▶ **Proposition 15** (Proof in the Appendix B)**.** *Mult-factorization of any tree T is unique.*

Finally note that counting the number of add-prime trees is simple (Proposition 9), but an even approximate count for mult-prime trees is much more difficult. We pose:

▶ **Open Problem.** *For a composite integer n determine the number of mult-prime trees of n vertices.*

algorithm MP(T)

1. CF(T);
// transform T in canonical form with Algorithm CF of Figure 4
2. let $H_1, \ldots, H_r$ be the groups of subtrees of $\boldsymbol{r}(T)$ as in Notation 1;
3. for $1 \leq i \leq r-1$
copy T into Z;
traverse Z in preorder
forany vertex x encountered in the traversal
if x has all the subtrees of H_i **erase** these subtrees in Z
else exit from the i-th cycle;
return MULT-COMPOSITE;
4. return MULT-PRIME.

Figure 6 Structure of Algorithm MP for deciding if a tree T of n vertices is mult-prime.

4 Negative trees, with a window on tree equations

Once addition and multiplication are known, it is natural to define the inverse operations.

We define the **subtraction** $A = T - B$ if and only if all the subtrees of $\boldsymbol{r}(B)$ are also subtrees of $\boldsymbol{r}(T)$. Then A equals T deprived of such subtrees. This is the inverse of the addition $T = A + B$, with $n_A = n_T - n_B + 1$. We have $T - \mathbf{1} = T$.

We define the **division** $A = T/B$ if and only if there exists a subset Ψ of the vertices of T such that each $v \in \Psi$ has exactly the subtrees of $\boldsymbol{r}(B)$, and the tree T' obtained as T deprived of such subtrees has exactly the vertices of Ψ. Then $A = T'$. This is the inverse of the multiplication $T = A \cdot B$, with $n_A = n_T/n_B$. We have $T \,/\, \mathbf{1} = T$.

Also the operation of stretch has an inverse. We define the **un-stretch** (symbol underline) if and only if $\boldsymbol{r}(A)$ has exactly one subtree T, and we pose $\underline{A} = T$. In the notation un-stretch has precedence over multiplication and stretch has precedence over un-stretch.

As negative numbers arose from subtraction in integer arithmetic, the more intriguing concept of negative trees arises here from tree subtractions. We propose the following definition. All the vertices of a tree T are either **positive** (then T is positive) or **negative** (then T is negative), except for the root that is **neutral**. Positive and negative vertices are respectively indicated with a black dot or an empty circlet. The root is also indicated with a black dot. Changing the sign of a tree amounts to changing the nature of all its vertices except for the root. Tree $\mathbf{1}$ is neutral and we have $\mathbf{1} = -\,\mathbf{1}$.

Addition and subtraction between A and B keep their definition with the additional condition that if A is positive and B is negative all the subtrees of $\boldsymbol{r}(B)$ are also subtrees of $\boldsymbol{r}(A)$ or vice-versa, and positive and negative subtrees with identical shape cancel each other out in the result (See Figure 7). Multiplication and division between A and B also keep their definition with the additional condition that if A and B are both positive or both negative the result is positive, otherwise is negative.

At this point we may open a window on **tree equations** whose terms have all the nature of a tree, but integers may appear as multiplicative coefficients or exponents. In a sense they are companions of the Diophantine equations with integers, but the solutions are now required to be trees. We may consider equations of different degrees with different number of variables, ask questions on the existence and on the number of solutions, study

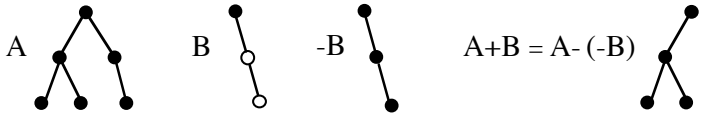

Figure 7 Addition between a positive tree A and a negative tree B.

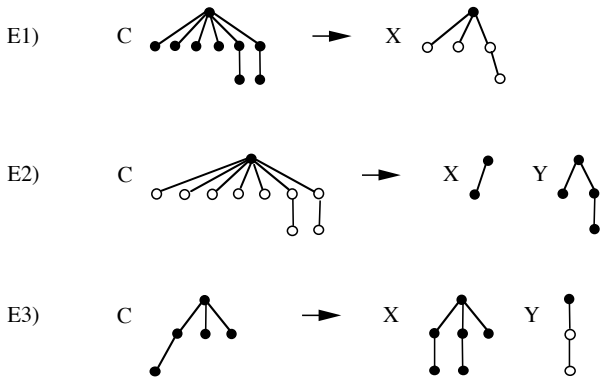

Figure 8 Solution of the tree equations: **E1**: $2X + C = \mathbf{1}$. **E2**: $3X + 2Y + C = \mathbf{1}$. **E3**: $2X + 3Y + C = \mathbf{1}$.

the computational complexity of finding them. In fact we give only some examples, leaving the field completely open.

Denote trees and integers with capital and lower case letters respectively. The simplest equation is linear and has only one unknown X. We put:

$$aX + C = \mathbf{1}, \qquad i.e. aX = -C\,. \tag{2}$$

Equation (2) admits exactly one solution if and only if the s_C subtrees of $\boldsymbol{r}(C)$ can be divided in $g \geq 1$ groups $G_1, \ldots, G_g$ of identical subtrees, where each G_i has cardinality $k_i a$ for $k_i \geq 1$, see example E1 in Figure 8. In this case X has s_C/a subtrees that can be divided in g groups of k_i subtrees identical to the ones of G_i. This solution can be easily built in time polynomial in n_C starting with the transformation of C in canonical form. Note that X and C have opposite sign.

A standard linear tree equation in two unknowns X, Y can be expressed as:

$$aX + bY + C = \mathbf{1}, \qquad i.e. aX + bY = -C\,. \tag{3}$$

This equation is companion of the diophantine equation $ax + by = c$ widely used in modular algebra, that admits an integer solution if and only if c is divided by $gcd(a, b)$. In general equation (3) admits a solution if and only if one of two non trivial conditions 1 and 2 holds, corresponding respectively to trees X, Y of equal sign or of opposite sign as in the examples E2, E3 of Figure 8. These conditions are reported in Appendix C. In both cases the solution can be built in time polynomial in n_C. Referring to the number of vertices, a necessary condition for the solution is that the integer equation $a\,n_X + b\,n_Y = n_C + a + b - 1$ (Case 1), or $a\,n_X - b\,n_Y = n_C + a - b - 1$ (Case 2), has an integer solution in n_X, n_Y, that is $n_C + a + b - 1$, or $n_C + a - b - 1$, is divided by $gcd(a, b)$ as in the examples above.

Higher degree equations are more difficult to handle. For the quadratic tree equation:

$$aX^2 + bY + C = \mathbf{1}, \qquad i.e. aX^2 + bY = -C \tag{4}$$

a necessary condition for the solution is the existence of two integers n_X, n_Y satisfying the algebraic equation $a\,n_X^2 + b\,n_Y = n_C + a + b - 1$ for Y positive, or $a\,n_X^2 - b\,n_Y = n_C + a - b - 1$ for Y negative, a well known NP-complete problem. To find a reasonably interesting approach for deciding whether equation (4) has a solution is left as an open problem.

A "more ambitious" problem can be expressed as:

$$X^n + Y^n = Z^n \tag{5}$$

with the question of deciding wether equation (5) has a tree solution X, Y, Z for any $n \geq 2$. In fact even for $n = 2$ the problem is not simple. Due to Proposition 1 we have the necessary condition $n_X^2 + n_Y^2 - 1 = n_Z^2$ for its solution, i.e. the existence of a "quasi-Pythagorean" triple of integers. In fact such triples exist, as for example $\{4, 7, 8\}$, but the existence of Pythagorean trees with such numbers of vertices is left as an open problem.

5 Possible applications and extensions

While the main purpose of the present study is the one of defining arithmetic concepts outside the realm of numbers, let us briefly discuss what the role of our proposal in applications might be.

Essentially all trees used in computer algorithms are rooted, and different families have been defined among them to deal with particular problems. We do not put any restriction on the tree structure. The trees considered here simply correspond to nested sets as for example hierarchical structures in computer science; or department plans in business organization; or phylogenetic trees in biology, etc. Note that the subtrees are essentially unordered at any vertex, although they must be stored in some standard form to be represented, e.g. following an alphanumeric label order of similar. Or, of course, our canonical order.

Some actions generally required in a hierarchical structure are the following. (i) Join two independent trees A, B to form a new tree T by merging the roots of A, B: e.g. merging two XML files. (ii) Add a new subtree B to the root of a tree T: e.g. adding a new task to a public authority. (iii) Join two independent trees A, B to form a new tree T with A, B subtrees of the root: e.g. joining two phylogenetic trees under a common ancestor. In our arithmetic action (i) is directly represented as $T = A + B$; action (ii) is represented as $T = T + \bar{B}$; and action (iii) is represented as $T = \bar{A} + \bar{B}$. These actions can be respectively undone as: (i') $A = T - B, B = T - A$; (ii') $T = T - \bar{B}$; and (iii') $A = \underline{T - \bar{B}}$, $B = \underline{T - \bar{A}}$. These inverse actions, for example, are basic tools for scheduling multithreaded computations [4].

An important extension of action (ii) is inserting a new subtree A at a given vertex v of T. This is obtained by an iterative operation along the path $\pi = (v_0, v_1, \ldots, v_k)$, from $\boldsymbol{r}(T) = v_0$ to $v = v_k$. Letting $T_0, \ldots, T_k$ be the subtrees rooted at vertices $v_0, \ldots, v_k$, hence $T = T_0$, we set $S_i = T_i - \bar{T}_{i+1}$ for $i = 0, 1, \ldots, k-1$; then we set $T_k = T_k + \bar{A}$; then we set $T_{i-1} = S_{i-1} + \bar{T}_i$ for $i = k, k-1, \ldots, 1$, where $T_0 = T$ gives the transformed tree. A similar operation is required to extract a subtree A at vertex v. Propositions 2 and 3 hold for the subtrees rooted at v, with obvious effects on the whole tree. Other operations can be considered and their representation investigated along the lines above. In particular multiplication may be performed on subtrees only, and even be limited at leaves.

We have cast a glance at tree equations as an invitation to look into this new field. A possible application is in data compression where the form $a_1X_1 + a_2X_2 + \cdots + a_kX_k = C$ can be the basis for representing C through the representation of $X_1, \cdots, X_k$, $a_1, \cdots, a_k$, thereby reducing the storage space from $\Theta(n_C)$ to $\Theta(n_{X_1} + \cdots + n_{X_k} + \log a_1 + \cdots + \log a_k)$: a substantial saving if $a_1, \cdots, a_k$ are large. Also multiplication may be useful in data

compression because the information contained in a product $A \cdot B$ is fully present in its factors, so the storage space needed for the product can be reduced from $\Theta(n_A \cdot n_B)$ to $\Theta(n_A + n_B)$. So the concept of primality may be of practical interest in the reverse-engineering operation of deciding if a tree has been generated as a sum or a product.

6 Other studies on tree arithmetic

Up to now only one major line of research, that we call LBY, has been directed to defining arithmetic on trees. Opened by J.L. Loday *et al* in connection with dendriform algebras [5], it was then developed by J.L. Loday himself who gave a full description of arithmetic operations on binary trees and their properties, showing an embedding of $\mathbb{N}$ in the subsets of all binary trees of n vertices [6]. A. Bruno and D. Yasaki worked on Loday's theory introducing primality and counting properties on subsets of trees in [2]. LBY is limited to binary trees, which carries simpler consequences than in our general case. A non-commutative tree addition is defined in LBY attaching the second addend to a deepest leaf of the first one, and this operation is given in two versions for building any tree from one generator (as in our proposal two different operations are needed). From this construction stems a definition of tree multiplication to produce trees different from our products. Several interesting properties are derived, including some counting arguments on the different families of trees built. The most relevant extension done by Bruno and Yasaki over Loday's theory is the definition and treatment of prime trees under multiplication. Aside from proceeding with similar purposes, none of the definitions and results of LBY applies to our theory, or vice-versa.

Another study on tree arithmetic, due to R. Sainudiin, is aimed at using binary trees for treating mapped partitions of a special class of intervals [10], and has nothing to share with LBY and with our theory. None of these works deals with aspect of computational complexity related to the operations on trees.

Along an independent line of research several papers have been directed to define graph multiplication, from the seminal work of G. Sibidussi [9] to the one of B. Zmazek and J. Zerownik [12]. In this context prime graphs and graph factorization have been considered under various operations of multiplication, see [1]. Again, if applied to trees as special graphs, all the definitions and results on tree multiplication are unrelated to ours.

Acknowledgements. Many thanks are due to Federico Poloni, Mahdi Amani, and to the conference reviewers, for their comments and suggestions.

References

1 N. Bray and E.W. Weisstein. Graph Product. *Math World* - A Wolfram Web Resource. *http://mathworld.wolfram.com/GraphProduct.html*

2 A. Bruno and D. Yasaki. The Arithmetic of Trees. *Involve* 4 (1) (2011), pp. 1–11.

3 S. Finch. Two Asymptotic Series. `http://www.people.fas.harvard.edu/~sfinch/`

4 C.E. Leiserson, T.B. Schardl, and W. Suksompong. Upper Bounds on Number of Steals in Rooted Trees. *Theory of Computing Systems* 58 (2016), pp. 223–240.

5 J.L. Loday, A. Frabetti, F. Chapoton, and F. Goichot. Dialgebras and related operands. *Lecture Notes in Mathematics* 1763, Springer-Verlag, Berlin (2001), pp. 7–66.

6 J.L. Loday. Arithmetree. *J. Algebra* 258 (1) (2002), pp. 275–309.

7 C. Pair and A. Quère. Définition et étude des bilangages réguliers. *Information and Control* 13 (1968), pp. 565–593.

8 J.M. Plitkin and J.W. Rosenthal. How to obtain an asymptotic expansion of a sequence from an analytic identity satisfied by its generating function. *J. Australian Math Soc.* (Ser. A) 56 (1994), pp. 131–143.

9 G. Sabidussi. Graph Multiplication. *Mathematische Zeitschrift* 72 (1960), pp. 446–457.

10 R. Sainudiin. Algebra and Arithmetic of Plane Binary Trees: Theory & Applications of Mapped Regular Pavings.
`http://www.math.canterbury.ac.nz/r.sainudiin/talks/MRP_UCPrimer2014.pdf`

11 S. Zaks. Lexicographic Generation of Ordered Trees. *Theoretical Computer Science* 10 (1980), pp. 63–82.

12 B. Zmazek and J. Zerownik. Weak Reconstruction of Small Product Graphs. *Discrete Mathematics* 307 (2007), pp. 641–649.

A List of sequences

The binary sequences representing the trees of the first six canonical families.

1	10		
		18	110101010100
2	1100	19	110101011000
		20	110101101000
3	110100	21	110101110000
4	111000	22	110110011000
		23	110110101000
5	11010100	24	110110110000
6	11011000	25	110111010000
7	11101000	26	110111100000
8	11110000	27	111001101000
		28	111001110000
9	1101010100	29	111010101000
10	1101011000	30	111010110000
11	1101101000	31	111011010000
12	1101110000	32	111011100000
13	1110011000	33	111100110000
14	1110101000	34	111101010000
15	1110110000	35	111101100000
16	1111010000	36	111110100000
17	1111100000	37	111111000000

B Proofs of propositions 4, 5, 8, 12, 14, 15

Proof of Proposition 4. The if part is immediate. For the only if part let $T = A \cdot B$ and $U = B \cdot A$. From the construction of the two products we immediately have $e_T = n_A e_B$ and $e_U = n_B e_A$. If $T = U$ we have $e_T = e_U$ then $n_A e_B = n_B e_A$, then $e_A = e_B$ since $n_A = n_B$. Note that T and U contain $e_A = e_B$ subtrees rooted in the former leaves of A and B respectively, each coinciding with B and A respectively. Each of these subtrees contains $n_B = n_A$ vertices, while all the other subtrees of T, U contain a different number of vertices. Then for having $T = U$ the former two groups of subtrees should be identical, that is each subtree coinciding with B in T must be equal to a subtree coinciding with A in U. That is $A = B$. ◀

Proof of Proposition 5. Let $T = A \cdot B$ and $U = B \cdot A$.

Condition (i) Immediate from the observation that $T = U$ implies $e_T = e_U$ (see the proof of Proposition 4).

Condition (ii) As in the proof of Proposition 4, consider the subtrees of T, U respectively attached to the former leaves of A in T and of B in U. Since $n_A e_B = n_B e_A$ (see the proof above) and $n_A > n_B$ we have $e_A > e_B$. In T there are e_A such subtrees of n_B vertices and in U there are e_B such subtrees of n_A vertices. For having $T = U$ the above subtrees of T (all coinciding with B) should be present also in U where, by the construction of $B \cdot A$, they must appear as subtrees of the copies of A in U.

Condition (iii) By construction the s_B subtrees of $\boldsymbol{r}(B)$ appear also in T as subtrees of $\boldsymbol{r}(T)$ where they are the ones with fewer vertices because all the others have at least n_B vertices. And the s_A subtrees of $\boldsymbol{r}(A)$ appear also in U as subtrees of $\boldsymbol{r}(U)$ where they are the ones with fewer vertices because all the others have at least n_A vertices. Note that all these other subtrees of $\boldsymbol{r}(U)$ have more vertices than the subtrees of $\boldsymbol{r}(B)$ since $n_A > n_B$. For having $T = U$ the s_B subtrees of $\boldsymbol{r}(B)$ that appear as subtrees of $\boldsymbol{r}(T)$ must be equal to s_B subtrees of $\boldsymbol{r}(U)$ and, for what just seen about these subtrees, they must be equal to s_B subtrees among the ones with fewer vertices, i.e. with subtrees of $\boldsymbol{r}(A)$. This also implies that if $s_A = s_B$ then $A = B$. ◀

Proof of Proposition 8. By contradiction. *If part*: for an arbitrary tree $X = A + B$ with $A, B \neq \mathbf{1}$, $\boldsymbol{r}(X)$ has at least two subtrees, then $T \neq X$ for any pair $A, B \neq \mathbf{1}$. *Only if part*: if $\boldsymbol{r}(T)$ has $k > 1$ subtrees $T_1, \ldots, T_k$ then $T = U + V$, where for example U is equal to T deprived of T_k and V is equal to T deprived of $T_1, \ldots, T_{k-1}$. ◀

Proof of Proposition 12. Since $T = A \cdot B$, the subtree Y has been inserted at $\boldsymbol{r}(T)$ as the largest subtree of $\boldsymbol{r}(B)$. Then also the subtrees of $\boldsymbol{r}(T)$ with at most n_Y vertices must have been inserted at $\boldsymbol{r}(T)$ as subtrees of $\boldsymbol{r}(B)$ since they have too few vertices for deriving from former subtrees of $\boldsymbol{r}(A)$ whose vertices are merged with B in T. Furthermore the remaining subtrees of $\boldsymbol{r}(T)$ cannot be subtrees of $\boldsymbol{r}(B)$ since they have too many vertices by the hypothesis that Y is a largest subtree of $\boldsymbol{r}(B)$. ◀

Proof of Proposition 14. Refer to Algorithm MP.

Correctness. Only step **3** requires an analysis. Z is the changing version of T and is restored at each i-th cycle. If one of the groups H_i of subtrees can be erased from Z at all vertices encountered in the traversal, the cycle is completed and the algorithm terminates declaring that T is mult-composite. In fact tree B, whose root has the subtrees in H_i, is one of the factors of T (see Proposition 12). If none of the i-cycles can be completed, that is no H_i can be found as being the group of subtrees of x in all vertices x of Z, the tree T is mult-prime as declared in step **4**.

Complexity. A superficial analysis of the algorithm is the following. Step **1** requires $O(n^2)$ time as discussed for algorithm CF. Step **2** is executed with a linear time scan because the tree is now in canonical form and the number of vertices in each subtree of the root has been computed by algorithm CF in step **1**. Step **3** requires $O(n)$ copy operations of T into Z in $O(n^2)$ time, and $O(n)$ traversals each composed of $O(n)$ steps, for a total of $O(n^2)$ steps. At each step at vertex x the subtrees in H_i must be compared with the subtrees of x with the same cardinality; this can be done by representing such subtrees with their binary sequences S and comparing these sequences. In the worst case vertex x has $O(n)$ subtrees of length $O(n)$, so that building and comparing all the sequences takes time $O(n^2)$, and the total

time required by step **3** is $O(n^4)$. Note that this analysis is very rough because the number of vertices of T decreases during the traversal, so the stated bound $O(n^4)$ is exceedingly high. ◀

Proof of Proposition 15. By contradiction assume that T has two different factorizations $T_1 \cdot T_2 \cdot \cdots T_k$ and $S_1 \cdot S_2 \cdot \cdots S_h$ in multi-prime factors. Tracing back from k and h, let T_i and S_j be the first pair of factors encountered with $T_i \neq S_j$. Then we have $T_1 \cdot T_2 \cdot \cdots T_i = S_1 \cdot S_2 \cdot \cdots S_j$. By Proposition 12 T_i must contain S_j as one of its factors (or vice-versa), against the hypothesis that T_i is mult-prime. ◀

C Solution of linear tree equations in two unknowns

The linear tree equation:

$$aX + bY + C = \mathbf{1}, \qquad i.e. \quad aX + bY = -C$$

admits a solution if and only if one of the following two conditions holds, corresponding respectively to trees X, Y of equal sign or of opposite sign.

1. The s_C subtrees of $\boldsymbol{r}(C)$ can be divided in $g \geq 1$ groups $G_1, \ldots, G_g$ and $h \geq 1$ groups $H_1, \ldots, H_h$ of identical subtrees, where each G_i has cardinality $g_i a$ for $g_i \geq 1$ and each H_i has cardinality $h_i b$ for $h_i \geq 1$. In this case X has $\sum_{i=1}^{g} g_i$ subtrees divided in g groups of g_i subtrees identical to the ones of G_i; and Y has $\sum_{i=1}^{h} h_i$ subtrees divided in h groups of h_i subtrees identical to the ones of H_i. This solution can be built in time polynomial in n_C. Note that X and Y have the same sign, and C has opposite sign. See Equation E2 in Figure 8.
2. Let the unknown trees X and Y have opposite sign. W.l.o.g. let the subtrees of $\boldsymbol{r}(X)$ be divided in $k + h$ groups $G_1, \ldots, G_{k+h}$ of identical subtrees, and the subtrees of $\boldsymbol{r}(Y)$ be divided in k groups $H_1, \ldots, H_k$ of identical subtrees, with $k \geq 1$ and $h \geq 0$. And let the subtrees of $\boldsymbol{r}(C)$ be divided in $k + h$ groups $C_1, \ldots, C_{k+h}$ of identical subtrees. x_i, y_i, c_i respectively denote the cardinalities of G_i, H_i, C_i.
   To allow the addition $aX + bY$ the subtrees in H_i must be identical to the ones in G_i for $1 \leq i \leq k$; the subtrees in C_i must be identical to the ones in G_i for $1 \leq i \leq k + h$; and we have the system of diophantine equations:

$$ax_i - by_i = c_i \qquad \text{for} \qquad 1 \leq i \leq k \tag{6}$$
$$ax_i = c_i \qquad \text{for} \qquad k + 1 \leq i \leq k + h \tag{7}$$

   whose integer solutions (if any) state that the a copies of the subtrees of G_i suffice to elide the b copies of the subtrees of H_i in C, for $i \leq k$; and a copies of the subtrees in G_i appear as subtrees of C_i, for $i > k$. The system can be solved under the conditions:

$$c_i / gcd(a, b) \text{ integer} \qquad \text{for} \qquad 1 \leq i \leq k \tag{8}$$
$$c_i / a \text{ integer} \qquad \text{for} \qquad k + 1 \leq i \leq k + h \tag{9}$$

   for a value of k established as the minimum value for which condition (9) holds (this fixes also the value of h). Then if all conditions (8) hold the system is solved in time polynomial in n_C and two trees X, Y satisfying equation (3) are immediately built from the values of x_i, y_i, out a potentially infinite number of solutions. In particular note that, for all i, the values x_i, y_i must be both positive to represent subset cardinalities. If this does not happen, an alternative positive solution is built from the other by standard methods. See equation E3 in Figure 8.