Arithmetic for Rooted Trees

Fabrizio Luccio

Published online: 29 November 2016
© Springer Science+Business Media New York 2016

Abstract We introduce a new arithmetic for non-empty rooted unordered trees. After discussing tree representation and enumeration, we define the operations of tree addition, multiplication, and stretch, and prove their properties. Using these operations all trees can be generated from a starting tree of one vertex. We show how a given tree can be obtained as the sum or as the product of two trees, and define 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 introduce tree equations whose coefficients are integers and whose unknowns are trees. We show how to solve some tree equations as an introduction to the field, and suggest more advanced examples. Finally we briefly discuss how our arithmetic might be useful in different applications. To the best of our knowledge our proposal is new and may be susceptible of variations and improvements.

Keywords Arithmetic · Rooted tree · Prime tree · Tree equation

This article is part of the Topical Collection on 50th Anniversary

A preliminary version of this work was presented at the 8th International Conference on Fun with Algorithms (FUN 2016). Proc. published by Dagstuhl Pub. Co.

Fabrizio Luccio
luccio@di.unipi.it

1 Dipartimento di Informatica, Università di Pisa, Pisa, Italy
1 Introduction

The title of this paper may call to mind different scenarios according to the cultural or professional background of the readers. Mathematicians would probably think of yet another speculation in coalgebra. Physicists may expect a new attempt to the normalization of quantum fields. Computing scientists, the readers to whom this work is mainly directed, may think of a new approach for handling data structures. Nobody would probably imagine the genuine purpose of this work, namely formulating and solving arithmetic equations on trees.

In the literature several operators on trees have been defined in the wide field of associative and co-associative Hopf algebras. Based on a preliminary study [11], J.-L. Loday presented his Arithmetree [12] built on three elementary operations of “grafting”, “over”, and “under” on planary binary trees (called p.b.t., i.e. binary trees in which all internal vertices have exactly two children). Based on these operations, a definition of non-commutative addition and multiplication followed, with the purpose of studying the algebraic structure of the whole set of trees. The study was naturally extended to trees whose internal vertices have at least two children. Primality under Loday multiplication was further investigated in [2].

The algebra for binary trees was then refined in [5] where trees evolve as search trees under insertion, and the whole theory is reframed under a new combinatorial approach.

More generally different operations have been defined on rooted trees of all types in the framework of Hopf algebras [7]. In particular the theory has been applied to renormalization of quantum fields [9], and for representing the solution of functional equations [6].

The purpose of this work is different. We are not investigating vector spaces on trees, but we simply treat them as primitive objects on which arithmetic operations are performed. We shall define such operations and investigate their properties and consequences, up to the formulation of equations whose unknowns are trees. Out of the former studies, the only one that may have some relation with ours is that of [12] and its follow-ups. Still our trees have no limitation in shape; our addition and multiplication generate a tree while in the previous studies addition and multiplication generate a forest; the computational complexity of all steps is crucial in our approach and is ignored by the others; and, above all, our aim and consequences are completely different.

Another line of research, however, has to be mentioned here. Born in the field of graph theory and algorithms, several investigations have been directed to graph multiplication: from the seminal work of Sibidussi [14], to the factorization algorithms of [3] and [16]. In this context prime graphs and graph factorization have been considered under various definitions of multiplication, see [1]. The approach adopted in all these works has some resemblance with ours. However, if applied to trees as special graphs, all the definitions and results on tree multiplication are unrelated to the ones that we present in this work.
2 Tree Enumeration

Trees are basic structures in combinatorial algorithms and have been the object of
countless works in the computing field. We study trees from this perspective, starting
with their representation and enumeration as the basis of any further discussion.

Preliminary notation and assumptions are as follows:

- We refer to rooted unordered trees simply called trees. Our trees are non empty.
  The symbol $1$ denotes the tree containing exactly one vertex, and is the basic
element of our construction.
- In a tree $T$, $r(T)$ 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 rooted at the
children of $x$ are called subtrees of $x$. $s_T$ denotes the number of subtrees of $r(T)$.
- A tree $T$ can be represented as a binary sequences $S_T$ (the original reference
for ordered trees is [15]). 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 Fig. 1, and has
the recursive structure $1S_1 \ldots S_k 0$, where each $S_i$ is a sequence representing a
subtrees of $r(T)$. The sequence for tree 1 is 10. 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, implying that in general many different sequences represent $T$. For this
reason a canonical form for trees is 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 fam-
ilies $F_1, F_2, \ldots$ as shown in Fig. 2, where $F_i$ contains the trees of $i$ vertices. So the
trees are ordered for increasing number of vertices, and inside each family the order-
ing is determined by the canonical form as explained below. Trees and sequences are
then numbered with increasing natural numbers.

- 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 $F_{n_U}$
precede the trees of $F_{n_T}$ in the ordering.
- The families $F_1, F_2$ contain one tree each numbered 1, 2.

![Fig. 1 Tree representation as a binary sequence. $S_1, S_2, S_3$ represent the subtrees of the root of $T$](image)
The canonical families of trees $F_1$ to $F_6$ and the corresponding tree enumeration

- The ordering of the trees in $F_{n>2}$ is based on the ordering of the preceding families, see Fig. 2. 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 $F_n$, where the digits in the multiset indicate the number of vertices of the subtrees of the root. For $F_6$ in Fig. 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 $F_3$ as third subtree, following the ordering in $F_3$; ...; 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 $F_n$ is the one of height 2 with $n - 1$ subtrees of the root of one vertex each and sequence 1101010...100; and the last tree is the “chain” of $n$ vertices and sequence 11...100...0. As said the binary sequences representing the trees in $F_n$ are ordered for increasing values, see Fig. 3.

Many (not necessarily all) trees of each family $F_n$ can be generated from the ones in $F_{n-1}$ using the following:

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

For example the four trees of $F_4$ in Fig. 2 can be built by DR from the two trees of $F_3$. The nine trees of $F_5$ can be built by DR from the four trees of $F_4$, with the exception of tree 13. The twenty trees of $F_6$ can be built by DR from the nine trees of $F_5$, 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
The binary sequences representing the trees of $F_1$ to $F_6$

|   |   |   |
|---|---|---|
| 1 | 10 | 18 1101010101000 |
| 2 | 1100 | 19 1101010110000 |
| 3 | 110100 | 20 1101011010000 |
| 4 | 111000 | 21 1101100110000 |
| 5 | 11010100 | 22 1101101010000 |
| 6 | 11011000 | 23 1101101100000 |
| 7 | 11101000 | 24 1101110100000 |
| 8 | 11110000 | 25 1101111000000 |
| 9 | 1101010100 | 26 1101110100000 |
| 10 | 1101011000 | 27 1101111000000 |
| 11 | 1101101000 | 28 1110011100000 |
| 12 | 1110110000 | 29 1110101010000 |
| 13 | 1110111000 | 30 1110101110000 |
| 14 | 1111010000 | 31 1111011010000 |
| 15 | 1111011000 | 32 1111011100000 |
| 16 | 1111100000 | 33 1111101100000 |
| 17 | 1111110000 | 34 1111110100000 |

For $F_n$ we immediately have $f_n > 2^{n-2}$ for increasing $n$; but an analysis attributed to George Pólya [8] shows that the asymptotic value of this function is much higher and can be approximated as:

$$f_n \sim 0.44 \cdot 2.96^n \cdot n^{-3/2}.$$  \hspace{1cm} (1)

Then the minimum length of the sequences representing the trees of $F_n$ 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 Fig. 3), then these four digits could be removed, leaving a sequence of $2n - 4$ bits to represent a tree. We maintain our representation since it allows to work easily on trees, leaving the construction of a shorter efficient coding as a challenging open problem.

To conclude this introduction we note that an arbitrary tree $T$ can be represented in canonical form in polynomial time with Algorithm CF of Fig. 4. An elementary analysis shows that the algorithm is correct and both steps 1, 2 can be executed in total $O(n^2)$ time.
3 Arithmetic Operations and Tree Generation

Let us start with the basic operations of addition (symbol +) and multiplication (symbol \( \cdot \), or simple concatenation). Let \( A, B \) be two arbitrary trees. Referring to the simple examples of Fig. 5 we pose:

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

**Definition 2 (Multiplication)** \( T = A \cdot B \) is built by merging \( r(B) \) with each vertex \( x \in A \) so that all the subtrees of \( r(B) \) become new subtrees of \( x \). We have \( A \cdot 1 = 1 \cdot A = A \).
In both operations it is immaterial in which order the subtrees are attached to the new parents. We also define the unary operation stretch (symbol over-bar) whose interest will be made clear in the following:

**Definition 3 (Stretch)** \( T = \bar{\bar{A}} \) consists of a new root \( r(T) \) with \( A \) attached as a subtree.

In all expressions parentheses have the usual role, stretch has precedence over multiplication, and multiplication has precedence over addition.

A solid arithmetic system must permit the construction of all the elements upon which the system works, through the application of the defined operators onto a finite set of *generators*. In our arithmetic all trees can be built 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 \( F_i \) with \( 1 \leq i \leq n - 1 \) can be generated by the trees of the preceding families, each tree \( T \) in \( F_n \) can also be generated. In fact if \( r(T) \) has one subtree \( T_1 \) then \( T \) can be generated as \( \bar{T}_1 \); if \( r(T) \) has \( k \geq 2 \) subtrees \( T_1, T_2, \ldots, T_k \) then \( T \) can be generated as \( U + V \) where \( U \) is \( T \) deprived of \( T_k \) and \( V \) is \( T \) deprived of \( T_1, T_2, \ldots, T_{k-1} \).

Let us now review the properties of our operations. From the definitions of addition and multiplication two immediate consequences 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 1 and 2, 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 \( F_{n_T} \). 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 \( F_{n_U} \).

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

\(^1\)Stretch has been included in the operation set to permit 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.
Proposition 3 \textit{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 1 and 3 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 the 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 \(r(X)\). For \(n_A > n_B\) our conditions are only necessary.

Proposition 4 \textit{For} \(n_A = n_B\) \textit{we have} \(A \cdot B = B \cdot A\) \textit{if and only if} \(A = B\).

\textit{Proof} 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 \(A = B\). \(\square\)

Proposition 5 \textit{For} \(n_A > n_B\ \textit{we have} A \cdot B = B \cdot A\ \textit{only if the following conditions are all verified}:

\begin{enumerate}[\textit{(i)}]
  \item \(n_A/e_A = n_B/e_B\);
  \item \(B\) is a proper subtree of \(A\);
  \item if \(s_A \geq s_B\) all the subtrees of \(r(B)\) must be equal to some subtrees of \(r(A)\).
\end{enumerate}

\textit{Proof} Let \(T = A \cdot B\) and \(U = B \cdot A\).

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

\textit{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 \(U\).

\textit{Condition} (iii). By construction the \(s_B\) subtrees of \(r(B)\) appear also in \(T\) as subtrees of \(r(T)\) where they are the ones with fewer vertices because all the others
have at least $n_B$ vertices. And the $s_A$ subtrees of $r(A)$ appear also in $U$ as subtrees of $r(U)$ 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 $r(U)$ have more vertices than the subtrees of $r(B)$ since $n_A > n_B$. For having $T = U$ the $s_B$ subtrees of $r(B)$ that appear as subtrees of $r(T)$ must be equal to $s_B$ subtrees of $r(U)$ 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 $r(A)$. This also implies that if $s_A = s_B$ then $A = B$.

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 $C = I$.

### 4 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 4** (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 $T$. 
Note that tree 1 is add-prime and mult-prime. 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 7**  $T$ is add-prime if and only if $r(T)$ has only one subtree.

**Proof** By contradiction. If part: for an arbitrary tree $X = A + B$ with $A, B \neq 1$, $r(X)$ has at least two subtrees, then $T \neq X$ for any pair $A, B \neq 1$. Only if part: if $r(T)$ 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 $T_1, \ldots, T_{k-1}$.

As a consequence of Proposition 7, deciding if a tree is add-prime is computationally “easy”. We also immediately have:

**Proposition 8** For $n \geq 2$ the number of add-prime trees is $f_{n-1}$.

From (1) we have: $f_{n-1}/f_n \rightarrow \sim 0.34$ for $n \rightarrow \infty$, that is the add-prime trees in $F_n$ are asymptotically about one third of the total. Each of the remaining add-composite (i.e., non add-prime) trees $T$ can be factorized in $s_T$ factors, that can be recursively factorized until all the add-prime factors are reached. We immediately have:

**Proposition 9** Add-factorization of any tree $T$ is unique.

For mult-primality we start with two immediate statements respectively deriving from Proposition 1 and from the definition of multiplication:

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

**Proposition 11** If $r(T)$ 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 $r(T)$ 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 $F_6$ (see Fig. 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.

The problem of deciding if $n_T$ is prime is polynomial in $\log n_T$. Since if $n_T$ is prime $T$ is mult-prime, checking this condition is computationally easy. 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 arbitrary 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 small, 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 \( r(B) \) are also subtrees of \( r(T) \). Namely:

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

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

In the mult-composite tree \( Z \) of Fig. 6, if the first subtree of \( r(Z) \) (containing one vertex) is a subtree of maximal cardinality of one of the factors, \( B \) in this case, then \( B \) consists of a root plus the first two subtrees of \( r(Z) \). Similarly, if the third subtree of \( r(Z) \) 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 \( r(Z) \). We pose:

**Notation 1** For an arbitrary tree \( T \): (i) \( G_1, \ldots, G_r \) are the groups of subtrees of \( r(T) \) 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 \( r(T) \) with up to \( g_i \) vertices.

Based on Proposition 1 and Notation 1 we can build the primality Algorithm MP of Fig. 7 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

```
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 \( r(T) \) as in Notation 1;
3. for \( 1 \leq i \leq r - 1 \)
   copy T into Z;
   traverse Z in preorder
   for any 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.
```

Fig. 7 Structure of Algorithm MP for deciding if a tree \( T \) of \( n \) vertices is mult-prime
Proposition 13 Mult-primality of any tree $T$ can be decided in time polynomial in $n_T$.

Proof 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 conservative analysis of the algorithm shows what follows. 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 $O(n^4)$.

The analysis of Algorithm MP given above is rough because the number of vertices decreases during the traversal of $T$, so the stated bound $O(n^4)$ is exceedingly high. In fact our aim was merely to show that the algorithm requires polynomial time.

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 $r(B)$, while the tree $Z$ is reduced to $A$. In particular $B$ is the last factor of a product of mult-prime trees, with $T = T_1 \cdot T_2 \cdots \cdot T_k \cdot B$. If Algorithm MP is not interrupted with the return statement, all these factors can be detected. As a consequence we have:

Proposition 14 Mult-factorization of any tree $T$ is unique.

Proof By contradiction assume that $T$ has two different factorizations $T_1 \cdot T_2 \cdots \cdot T_k$ and $S_1 \cdot S_2 \cdots \cdot S_h$ 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 $T_1 \cdot T_2 \cdots \cdot T_i = S_1 \cdot S_2 \cdots \cdot S_j$. 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.
Finally note that counting the number of add-prime trees is simple (Proposition 8), but an even approximate count for mult-prime trees is much more difficult. We pose:

**Open Problem 1** For a composite integer \( n \) determine the number of mult-prime trees of \( n \) vertices.

## 5 Negative Trees

Once the basic arithmetic operations have been established, it is natural to define their inverses. We pose:

**Definition 5** (Subtraction) \( A = T - B \) is defined if and only if all the subtrees of \( r(B) \) are also subtrees of \( r(T) \). 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 \( T - 1 = T \).

**Definition 6** (Division) \( A = T/B \) is defined if and only if there exists a subset \( \Psi \) of the vertices of \( T \) such that each \( v \in \Psi \) has exactly the subtrees of \( r(B) \); and, depriving \( T \) of such subtrees, a tree \( T' \) is obtained having exactly the vertices of \( \Psi \). Then \( A = T' \). This is the inverse of the multiplication \( T = A \cdot B \), with \( n_A = n_T/n_B \). We have \( T / 1 = T \).

Also the operation of stretch has an inverse (symbol underline):

**Definition 7** (Un-stretch) \( T = A \) is defined if and only if \( r(A) \) has exactly one subtree \( T \), and we pose \( A = T \).

In all expressions addition and subtraction, multiplication and division, and stretch and un-stretch, have the same precedence.

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 circllet. 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 the root. Tree \( 1 \) is neutral and we have \( 1 = -1 \). This tree has a companion role of integer 0 in integer arithmetic.

Addition and subtraction between \( A \) and \( B \) keep their definition, with the additional condition that positive and negative subtrees with identical shape cancel each other.

![Addition between a positive tree A and a negative tree B](image-url)
other out in the result. So for \( A \) positive and \( B \) negative, with \( n_A \geq n_B \), addition is defined if and only if the subtraction \( A - (-B) \) is defined, see Fig. 8. Multiplication and division between \( A \) and \( B \) are defined as if they were both positive, then letting the result be positive if \( A \) and \( B \) are both positive or both negative, and letting the result be negative otherwise.

6 Tree Equations

The definition of negative trees leads us into the probably most intriguing part of our work, namely introducing of tree equations. Unknowns and terms will be trees, but integers will appear as multiplicative coefficients or exponents.

In this work we thoroughly study some simple equations only, as an introduction to a possibly wide field of investigation. In general one may consider equations of different degrees with different number of variables, ask questions on the existence and on the number of solutions, study the computational complexity of finding them. In a sense tree equations are the companions of Diophantine equations in integer arithmetic, with the request that their solutions must be trees.

Denote trees and integers with capital and lower case letters respectively. The simplest linear equation has only one unknown \( X \) and is expressed as:

\[
aX + C = 1, \text{ i.e } aX = -C
\]  

We have:

**Proposition 15** Equation (2) admits a solution if and only if the following Condition 1 holds. In the affirmative case the solution can be found in polynomial time in the size of the problem description.

**Condition 1** Since \( aX \) equals the addition of \( a \) copies of \( X \), equation (2) admits a solution (in fact, a unique solution) if and only if the \( s_C \) subtrees of \( r(C) \) can be divided in \( g \geq 1 \) groups \( G_1, \ldots, G_g \) of identical subtrees, where each \( G_i \) has cardinality \( k_i \cdot a \) with \( k_i \geq 1 \). If this condition is fulfilled, the solution \( X \) is built as a root with \( s_C/a \) subtrees attached to it and divided in \( g \) groups of \( k_i \) subtrees identical to the ones of \( G_i \). The whole process can be easily built in time polynomial in \( n_C \) starting with the transformation of \( C \) in canonical form.

The proof of Proposition 15 is implicit in the discussion of Condition 1. An example of equation (2) is equation E1 in Fig. 9. Note that \( X \) and \( C \) must have opposite sign.

A linear tree equation in two unknowns \( X, Y \) can be expressed as:

\[
aX + bY + C = 1, \text{ i.e } aX + bY = -C
\]  

We have:
Fig. 9 Solution of the equations: E1: $2X + C = 1$. E2: $3X + 2Y + C = 1$. E3: $2X + 3Y + C = 1$

Proposition 16 Equation (3) admits a solution if and only if one of the following Conditions 2 or 3 holds. In the affirmative case the solution can be found in polynomial time in the size of the problem description.

Condition 2 For trees $X, Y$ of equal sign. The $s_C$ subtrees of $r(C)$ 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 $C$ has the opposite sign of $X, Y$. See Equation E2 in Fig. 9.

Condition 3 For trees $X, Y$ of opposite sign. W.l.o.g. let the subtrees of $r(X)$ be divided in $k + h$ groups $G_1, \ldots, G_{k+h}$ of identical subtrees, and the subtrees of $r(Y)$ 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 $r(C)$ 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:

$$
a x_i - b y_i = c_i \quad \text{for } 1 \leq i \leq k \quad \text{(i)}
$$
$$
a x_i = c_i \quad \text{for } k + 1 \leq i \leq k + h \quad \text{(ii)}
$$

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:

$$
c_i / \gcd(a, b) \text{ integer} \quad \text{for } 1 \leq i \leq k \quad \text{(iii)}
$$
$$
c_i / a \text{ integer} \quad \text{for } k + 1 \leq i \leq k + h \quad \text{(iv)}
$$
for a value of $k$ established as the minimum value for which condition (iv) holds (this fixes also the value of $h$). Then if all conditions (iii) hold the system is solved, 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 pairs. The solution takes time polynomial in $n_C, n_X$ and $n_Y$. In particular 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 preceding one by standard methods. See equation E3 in Fig. 9.

The proof of Proposition 16 is implicit in the discussion of Conditions 2 and 3.

An alternative necessary (but not sufficient) condition for the existence of a solution of equation (3) can be built on the yet unknown numbers of vertices $n_X, n_Y$ of trees $X$ and $Y$. As this condition is very simple, it may be worth to test it before trying to apply Conditions 2 and 3. Recalling that the number of vertices of a product $kT$ equals $kn_T - k + 1$, we have that, for tree equation (3) to be solvable, one of the algebraic integer equations:

$$an_X + bn_Y = n_C + a + b - 1, \text{ for } X, Y \text{ of equal sign},$$

$$an_X - bn_Y = n_C + a - b - 1, \text{ for } X, C \text{ of equal sign and } Y \text{ of different sign},$$

must have an integer solution in $n_X, n_Y$. Such a solution exists if and only if $n_C + a + b - 1$, or $n_C + a - b - 1$, is divided by $\gcd(a, b)$; a condition that can be verified in linear time in the size of the parameters. The reader may check equations (4) and (5), and the condition for their integer solution, on the equations E2 and E3 of Fig 9, respectively.

A natural extension of equation (3) is:

$$\sum_{i=1}^{k} a_i X_i + C = 1, \text{ i.e } \sum_{i=1}^{k} a_i X_i = -C, \text{ for } k > 2$$

The decision on whether equation (6) admits a solution, and the construction of such a solution in the affirmative case, can be attained with a simple extension of Condition 3 applied to the $2^k$ combinations of positive and negative signs of the trees $X_i$.

Tree equations of degree greater than one are more difficult to handle. Let us start with the simple quadratic equation:

$$aX^2 + C = 1, \text{ i.e } aX^2 = -C$$

We have:

**Proposition 17** Equation (7) admits a solution if and only if the requirements in the following Condition 4 are fulfilled. If exists, the solution can be found in polynomial time in the size of the problem description.

**Condition 4** A solution of equation (7) exists if the two steps of the following procedure can be completed. In this case the solution itself is determined. Starting with the formal transformation $Y = X^2$: 1) Solve the equation in $Y$: $aY + C = 1$, if possible, by applying Proposition 15. 2) Solve the equation: $X^2 = Y$, i.e., compute the square root $X$ of $Y$. For such a square root to exist two necessary conditions must

 Springer
hold, due to the multiplication of $X$ by itself: (i) $n_X^2 = n_Y$, i.e. $n_Y$ must be a perfect square; (ii) $2s_X = s_Y$, i.e. $s_Y$ must be even. (These two conditions should be checked before proceeding). Furthermore, if such a tree $X$ exists, the subtrees of its root can be determined by Proposition 12 as the $s_Y/2$ smaller subtrees of $Y$. Then build the tree $T$ with these subtrees, compute $T^2$, and check if $T^2 = Y$. If so we have $X = T$ and the equation is solved. Note that all these operations can be done in polynomial time.

The proof of Proposition 17 is implicit in the discussion of Conditions 4.

For quadratic or higher degree tree equation in two or more unknowns we are unable to give interesting results at the moment. For the simplest of them:

$$aX^2 + bY + C = 1, \text{ i.e. } aX^2 + bY = -C \quad (8)$$

we are only able to state a necessary condition for the existence of a solution. With an argument similar to the one developed with the integer equations (4) and (5) we have that, for the tree equation (8) to be solvable, one of the algebraic integer equations:

$$an_X^2 + bn_Y = nC + a + b - 1, \text{ for } Y \text{ positive,} \quad (9)$$

$$an_X^2 - bn_Y = nC + a - b - 1, \text{ for } Y \text{ negative,} \quad (10)$$

must have an integer solution in $n_X, n_Y$. Verifying this condition is an NP-complete problem [4]. In general we pose:

**Open Problem 2** Find a solid approach for deciding whether equation (8) admits a solution, and find the solution in the affirmative case.

We conclude this section with a “more ambitious” problem expressed by the tree equation:

$$X^k + Y^k = Z^k \quad (11)$$

with the question of deciding wether there is a solution $X, Y, Z$ for any value $k \geq 2$. In fact even for $k = 2$ 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 quasi-Pythagorean trees with such numbers of vertices is left as an open problem.

### 7 Possible Applications and Extensions

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

Essentially all trees used in computer algorithms are rooted, but 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. Although the subtrees are essentially unordered at any vertex, they must be stored in some standard form to be represented, e.g. following an alphanumeric label order of similar. Or, of course, following our canonical order.

Three actions often required in a hierarchical structure are the following:

1. Join two independent trees $A, B$ to form a new tree $T$ by merging the roots of $A, B$: e.g. merge two XML files.
2. Add a new subtree $B$ to the root of a tree $T$: e.g. add a new task to a public authority.
3. Join two independent trees $A, B$ to form a new tree $T$ with $A, B$ subtrees of the root: e.g. join two phylogenetic trees under a common ancestor.

In our arithmetic:

- action 1 is directly represented as $T = A + B$;
- action 2 is represented as $T = T + \bar{B}$;
- action 3 is represented as $T = \bar{A} + \bar{B}$.

These actions can be respectively undone as:

- $A = T - \bar{B}$; $B = T - A$;
- $T = T - \bar{B}$;
- $A = T - \bar{B}$, $B = T - \bar{A}$.

These inverse actions, for example, are basic tools for scheduling multithreaded computations [10].

An important extension of action 2 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 $r(T) = v_0$ 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 $v$. Note that this operation has an impact on the canonical form of the whole tree.

Other operations can be considered and their representation investigated along the lines above. In particular multiplication may be performed on some subtrees only, for example be limited at the leaves.

A possible application of tree equations is in data compression where the form $a_1X_1 + a_2X_2 + \ldots + a_kX_k = C$ can be the basis for representing $C$ through the representation of $X_1, \ldots, X_k, a_1, \ldots, a_k$, thereby reducing the storage space from $\Theta(n_C)$ to $\Theta(n_{X_1} + \ldots + n_{X_k} + \log a_1 + \ldots + \log a_k)$: a substantial saving if $a_1, \ldots, a_k$ are large. Also multiplication may be significant 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 useful in tree compression, or be of practical interest in the
reverse-engineering operation of deciding if a tree has been generated as a sum or a product.

Obviously this is a brief sketch of possible problems. The usefulness of tree operations and equations might come to light in different applications once their properties are known.

8 Concluding Remarks

This paper proposes a new arithmetic for unordered rooted trees, conceived in a computational environment. A commutative addition and a non-commutative multiplication between trees are defined, together with a unary tree stretch operation that creates a new root. After discussing the properties of the resulting structure with particular attention to primality, the three inverse operations of subtraction, division, and un-stretch are introduced, leading to the definition of negative trees and to the formulation of tree equations. In a sense equations on trees are a counterpart of Diophantine equations on integers.

When discussing the state of the art, we have underlined that arithmetic operations on trees have been discussed for decades in the field of colagebras, but the focus has always been on the abstract algebraic structure of the system without attention to computational aspects. Tree equations in particular seem not to have been considered before.

In setting up our theory different attempts have been made before choosing the present formulation. Let us mention a few points. First note that the empty tree, say $\Phi$, is not present in our proposal where the atomic element is tree $\mathbf{1}$ containing exactly one vertex. Here $\mathbf{1}$ is the equivalent of 0 in integer arithmetic, to get $A + \mathbf{1} = T$ since the roots of the two trees are merged. If $\Phi$ existed, $A + \Phi$ could not be defined because $\Phi$ has no root.

While the proposed tree multiplication seems to be a natural extension of integer multiplication, different definitions of tree addition have been considered. In particular $A + B$ may be defined as the creation of a new root to which $A$ and $B$ are attached as subtrees. Addition with $\Phi$ would then be allowed, although with the result $A + \Phi \neq A$. This addition also plays the role of stretching because a new root is created. However its combination with multiplication would not be sufficient to build all trees from a finite set of generators, thus requiring the introduction of a third operation as in our case where stretch has been added primarily for this purpose. An operation aimed at increasing the height of the trees is in fact present in most of the algebraic proposals mentioned above, as well as in language theory (see for example [13]).

In conclusion our proposal arises from a deep study of different alternatives, but is susceptible of a wealth of possible variations and improvements. Further studies on tree arithmetic are definitely needed.

Acknowledgment Many thanks are due to Federico Poloni, Mahdi Amani, Günter Rote, and Giovanni Viglietta, for their comments and suggestions.
References

1. Bray, N., Weisstein, E.W.: Graph Product. Math World - A Wolfram Web Resource. http://mathworld.wolfram.com/GraphProduct.html
2. Bruno, A., Yasaki, D.: The arithmetic of trees. Involve \textbf{4}(1), 1–11 (2011)
3. Feigenbaum, J., Hershberger, J., Schäffer, A.A.: A polynomial time algorithm for finding the prime factors of Cartesian-product graphs. Discret. Appl. Math. \textbf{12}, 123–138 (1985)
4. Garey, M.R., Johnson, D.S.: Computers and Intractability, Appendix A7.2. Freeman & Co., San Francisco (1979)
5. Hivert, F., Novelli, J.-C., Thibon, J.-Y.: The algebra of binary search trees. Theor. Comput. Sci. \textbf{339}, 129–165 (2005)
6. Hivert, F., Novelli, J.-C., Thibon, J.-Y.: Trees, functional equations, and combinatorial Hopf algebras. Eur. J. Comb. \textbf{29}, 1682–1695 (2008)
7. Hoffman, M.E.: Combinatorics of rooted trees and Hopf algebras. Trans. Am. Math. Soc. \textbf{355-9}, 3795–3811 (2003)
8. Knuth, D.E.: The Art of Computer Programming, vol. 1, Sec. 2.3.4.4. Addison-Wesley, Reading (1968)
9. Kreimer, D.: On the Hopf algebra structure of perturbative quantum field theory. Adv. Theor. Math. Phys. \textbf{2}, 303–334 (1998)
10. Leiserson, C.E., Scharldl, T.B., Suksompong, W.: Upper bounds on number of steals in rooted trees. Theory Comput. Syst. \textbf{58}, 223–240 (2016)
11. Loday, J.L., Frabetti, A., Chapoton, F., Goichot, F.: Dialgebras and Related Operads. Lecture Notes in Mathematics, vol. 1763, pp. 7–66. Springer-Verlag, Berlin (2001)
12. Loday, J.L.: Arithmetree. J. Algebra \textbf{258}(1), 275–309 (2002)
13. Pair, C., Quèrè, A.: Définition et étude des bilangages réguliers. Inf. Control. \textbf{13}, 565–593 (1968)
14. Sabidussi, G.: Graph multiplication. Math. Z. \textbf{72}, 446–457 (1960)
15. Zaks, S.: Lexicographic generation of ordered trees. Theor. Comput. Sci. \textbf{10}, 63–82 (1980)
16. Zmazek, B., Zerownik, J.: Weak reconstruction of small product graphs. Discret. Math. \textbf{307}, 641–649 (2007)