Applied General Topology
© Universidad Politécnica de Valencia
Volume 9, No. 1, 2008
pp. 1-14

Topological and categorical properties of binary trees

H. Pajoohesh

To my friend, Tehmineh Fadaei for her 40'th birthday

Abstract. Binary trees are very useful tools in computer science for estimating the running time of so-called comparison based algorithms, algorithms in which every action is ultimately based on a prior comparison between two elements. For two given algorithms A and B where the decision tree of A is more balanced than that of B, it is known that the average and worst case times of A will be better than those of B, i.e., $\overline{T}_A(n) \leq \overline{T}_B(n)$ and $T_A^W(n) \leq T_B^W(n)$. Thus the most balanced and the most imbalanced binary trees play a main role. Here we consider them as semilattices and characterize the most balanced and the most imbalanced binary trees by topological and categorical properties. Also we define the composition of binary trees as a commutative binary operation, *, such that for binary trees A and B, $A * B$ is the binary tree obtained by attaching a copy of B to any leaf of A. We show that $(T, *)$ is a commutative po-monoid and investigate its properties.

2000 AMS Classification: 06A12, 06F05, 16B50.

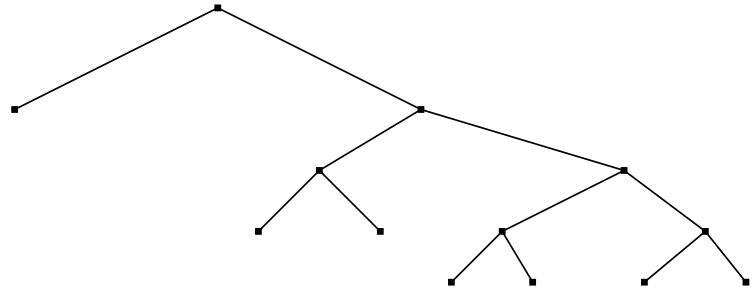
Keywords: Algorithm, decision tree, lower topology, semilattice.

1. Introduction

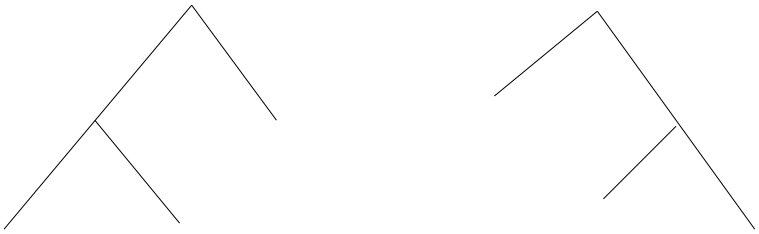
Here we bring some definitions and statements from [11]:

Definition 1.1. *Binary trees are reversed trees with a root node, in which every internal node has exactly two children. The order of the leaves is insignificant, so a tree is determined (up to permutation on the leaves) by the lengths of the paths from the root node to each leaf (the distance of the leaf from the root). Thus we can represent equivalence classes of the rooted binary trees with n leaves by sequences of n non-negative integers, which give the path-length of each leaf.*

For example, the path-length sequence $< 1\,3\,3\,4\,4\,4\,4 >$ represents a binary tree with $n = 7$ leaves, of which one has path-length 1, two have path-length 3, and four have path-length 4. This is shown in the following:



Remark 1.2. Notice that both of the following binary trees are the same for us.



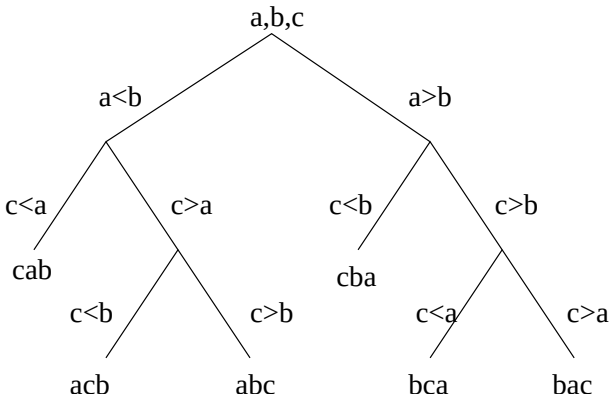
We bring the following Theorem from [6]:

Theorem 1.3. *If we denote the set of binary trees with n leaves by* T_n, $< x_1 \dots x_n > \in T_n$ *if and only if* $\sum_{i=1}^{n} 1/2^{x_i} = 1$.

Our interest in decision trees stems from the fact that in order to carry out the running time analysis of so-called "comparison-based algorithms", i.e. algorithms in which every action is ultimately based on a prior comparison between two elements, the notion of a decision tree is a fundamental tool [2]. Most sorting and searching algorithms are comparison-based. Decision trees are binary trees representing the comparisons carried out during the computation of a comparison-based algorithm. The path-length from the root ("input") to a leaf ("output") gives the comparison time for the algorithm (i.e. the total number of comparisons) to compute the output corresponding to the given input.

The concept of the path-length of a tree is of great importance in the analysis of algorithms, since this quantity is often directly related to the execution time [6]. If we consider decision trees, then the path-length represents the number of comparisons made while producing the result for a given input list. Therefore, the sum of the path-length sequence represents the total number of comparisons made by the sorting algorithm over all possible permutations of the input list.

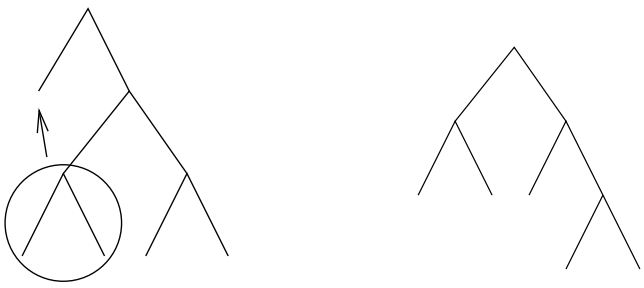
For example, if we consider some sorting algorithm A which sorts a list of size 3 then this will produce a decision tree with six leafs, each of which corresponds to the computation involved in producing the sorted list for one of the six possible input lists which can exist. Assume that the path-length sequence for the decision tree of A is $< 2\,3\,3\,2\,3\,3 >$,(See the following diagram). As mentioned previously, the path-length sequences form an equivalence class so we can represent this tree with the sequence $< 2\,2\,3\,3\,3\,3 >$. From this sequence we can see that two of the input lists took two comparisons to be sorted and the other four input lists took three comparisons each.



Definition 1.4. *Consider* $x =< a_1 \dots a_j \, \mathbf{p} \, a_{j+2} \dots \mathbf{q} + \mathbf{1} \, \mathbf{q} + \mathbf{1} \dots a_n >$, $y =< a_1 \dots a_j \, \mathbf{p} + \mathbf{1} \, \mathbf{p} + \mathbf{1} a_{j+3} \dots \mathbf{q} \dots a_n > \in T_n$. *Then we say* y *can be obtained from* x *via a ternary exchange. If* $q = p + 1$ *then we call this ternary exchange, a minimal ternary exchange.*

For every $x =< x_1 \dots x_n > \in T_n$, the level of x is defined by $L(x) = (n-1)(n+2)/2 - \sum_{i=1}^{n} x_i$.

This type of balancing exchange has the effect of moving a subtree consisting of two leaves at level q in the tree to a leaf (which then becomes an internal node) at level p in the tree. An example of this is shown in the following:

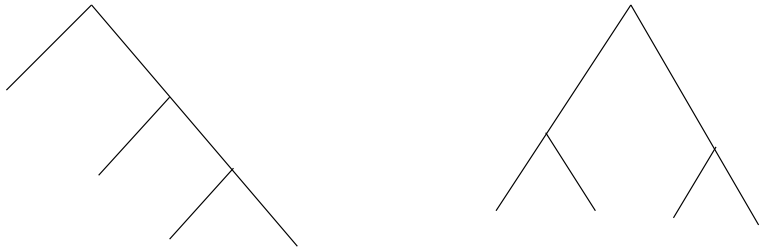


<1 3 3 3 3> <2 2 2 3 3>

It is obvious that these minimal balancing exchanges have the effect of increasing the level of balance of a tree by a value of one. Consequently, they also have the effect of decreasing the sum of the path-lengths of a sequence by a value of one. Therefore, every time we increase the level of balance of a tree's path-length sequence (through minimal balancing exchanges), we simultaneously decrease the sum of its path-lengths.

Definition 1.5. *If $x, y \in T_n$ then we define $x \leq y$ if there are $l_1, ..., l_m \in T_n$ such that $y = l_1, x = l_m$ and for each i, $1 \leq i < m$, l_{i+1} is obtained from l_i via a ternary exchange. So if $x \leq y$ then we say that x is more balanced than y.*

Example 1.6. In the following diagram, $x =< 1\,2\,3\,3 >$ and $y =< 2\,2\,2\,2 >$, $p = 1$ and $q + 1 = 3$. So y is obtained from x via a ternary exchange which is actually a minimal ternary exchange because $q = p + 1$. Thus y is more balanced than x, what is seen in the diagrams as well.



x **y**

The path-length from the root ("input") to a leaf ("output") gives the comparison time for the algorithm (i.e. the total number of comparisons) to compute the output corresponding to the given input. So the total time is the summation of all path-lengths from the root to the leaves. For instance in Example 1.6 we can see that for x, the total summation of path-lengths is $1 + 2 + 3 + 3 = 9$ while for y, $2 + 2 + 2 + 2 = 8$. So it is seen in this example that when y is more balanced than x then the summation of the path length sequence for y is less than the summation of the path length sequence for x. It is true in general that if the binary tree corresponding to the c.b. algorithm (comparison based algorithm)A is less than or equal to the binary tree corresponding to the c.b. algorithm B then the running(total) time of A (which is actually the sum of path length sequence of the binary tree corresponding to it) is less than the running time of B. In other words A is faster than B. In other words when $x =< x_1 \, ... \, x_n >$ is more balanced than $y =< y_1 \, ... \, y_n >$ then $\sum_{i=1}^{n} x_i \leq \sum_{i=1}^{n} y_i$

The average time analysis and the worst and the best case analysis time for a c.b. algorithm A which is defined as the usual, is denoted respectively by $\overline{T(A)}$,$T^W(A)$ and $T^B(A)$. It was shown in [9, 10] that given two algorithms A and B where the decision tree of A is more balanced than that of B, we have that the average and worst case times of A will be better than those of B, i.e., $\overline{T}_A(n) \leq \overline{T}_B(n)$ and $T_A^W(n) \leq T_B^W(n)$.

Proposition 1.7 ([11]). *T_n with the above order is a lattice.*

Definition 1.8. *The smallest element of T_n is called the most balanced binary tree and the biggest one is called the most imbalanced binary tree.*

The notion of balance is very important in the context of running times of algorithms. A binary tree is the most balanced if the height of the left subtree of every node never differs by more than ± 1 from the height of its right subtree [7]. If a decision tree is balanced then its sum of path-lengths will be minimised [6]. This in turn means that the computation time for the algorithm which produced this decision tree will be minimised.

2. Topologies on ordered trees

In this section we look at binary trees as a semi-lattice. They are DCPOs with the induced order. We study the Scott and lower topology on them and will show that every join preserving map from a binary tree to another is Scott continuous. Then we characterize the most imbalanced binary trees by using the lower topology.

Definition 2.1. *Consider the poset X and $x, y \in X$ such that $x < y$. We say x is a lower cover of y and will denote $x \prec y$ if there is no element between them. In other words $x \prec y$ if and only if $\{t | x < t < y\} = \varnothing$.*

Definition 2.2. *An ordered tree is a conditionally complete $\vee$-semilattice T so that for every $x \in T$,*

i) $|\{y \in T | y \prec x\}| = 2$ *or* 0
ii) $|\{y \in T | x \prec y\}| = 1$ *or* 0

By a conditionally complete join semilattice we mean a $\vee$-semilattice such that every non-empty subset has a join.

Definition 2.3. *For every ordered tree, T by $\bigvee T$ we mean the biggest element of T and by $m(T)$ we mean the set of minimal elements of T.*

Every partial order on a DCPO (A poset such that every directed set has a join) induces two topologies on this set, Scott topology and lower topology and their join which is called Lawson topology.

We recall that if $(X, \leq)$ is a DCPO, then $A \subseteq X$ is called Scott open if it is an upperset- which means that if $a \in A$ and $a \leq b$ then $b \in A$- and whenever for directed set $D \subseteq X$, $\bigvee D \in A$ then $D \cap A \neq \varnothing$. The collection of the Scott open sets is a topology and is called Scott topology. The topology generated by $X - \uparrow x$, $x \in X$ where $\uparrow x = \{y \in X | y \geq x\}$ is called lower topology and is denoted by α.

We study these topologies in this section.

Theorem 2.4. *Let T_1, T_2 be ordered trees. Then every join preserving map f from T_1 to T_2 preserves arbitrary join and hence is Scott continuous.*

Proof. Consider $\{x_i | i \in I\} \subseteq T_1$. We show that $f(\bigvee_{i\in I} x_i) = \bigvee_{i\in I} f(x_i)$. Since f is join preserving and hence order preserving which gives

$$\bigvee_{i\in I} f(x_i) \leq f(\bigvee_{i\in I} x_i). \qquad (*)$$

Now if $m, n \prec \bigvee x_i$, $m \neq n$, then it is obvious that there are x_l, x_h, where $h, l \in I$, $x_l \neq x_h$, such that $x_l \leq m$ and $x_h \leq n$. Thus $x_l \vee x_h = \bigvee x_i$, so $f(x_l) \vee f(x_h) = f(\bigvee x_i)$. This shows

$$f(\bigvee x_i) \leq \bigvee f(x_i). \qquad (**)$$

From (*) and (**) we get $f(\bigvee x_i) = \bigvee f(x_i)$ and the proof is complete. □

Definition 2.5. *Let T be an ordered tree, then for every $x \in T$, we define the length of x be the distance of x to $\bigvee T$ and we denote it by $l(x)$. More precisely for x let $x = x_0 \prec x_1 \prec ... \prec \bigvee T = x_m$ then $l(x) = m$. Also we define the length of T, $l(T) = max\{l(x) | x \in m(T)\}$.*

Remark 2.6. By the construction of ordered trees for every element of an ordered tree T, there is a unique path which connects it to $\bigvee T$.

Definition 2.7. *For every ordered tree, we define, $S(T)$, to be the increasing sequence corresponding to the distance of the minimal elements to $\bigvee T$.*

Definition 2.8. *In the lower topology, α, $A \in \alpha$ is called principal if there exists $x \in A$ such that $A = \downarrow x$. Obviously every minimal element is an open set. We call them trivial open sets.*

Definition 2.9. *The principal α-open set V is called atomic if $c(V) = \{W \in \alpha | W \subsetneq V$, W is nontrivial and principal $\}$ with inclusion is a chain. It is called strongly atomic if $c(V)$ has more than one element.*

Theorem 2.10. *Let T be an ordered tree with n minimal elements. Then*

1. *Every principal open set in the lower topology is atomic if and only if for some $n \in \{2, 3, ...\}$, $S(T) = (1, ..., n-2, n-1, n-1)$.*
2. *Every strongly atomic open set consists at least 3 minimal elements with 3 different lengths.*

Proof.

1. Since every principal open set is atomic, T itself is atomic. Let $x_1, y_1 \prec \bigvee T$. We show that one of x_1 and y_1 is a minimal element. Equivalently we show that $\downarrow x_1 \cap T = \{x\}$ or $\downarrow y_1 \cap T = \{y\}$. First of all since T is a lower set $\downarrow x_1 = \downarrow x_1 \cap T$ and $\downarrow y_1 = \downarrow y_1 \cap T$. Clearly these two sets are non-empty. Now if $\downarrow x_1 \cap T$ and $\downarrow y_1 \cap T$ both have more than one element, then both $\downarrow x_1 = \downarrow x_1 \cap T$ and $\downarrow y_1 = \downarrow y_1 \cap T$ are nontrivial principal open sets which are contained in T and neither of them contains the other one which contradicts the atomic property of T. Thus either x_1 or y_1 is a minimal element. Let x_1 be minimal element. Notice that if both are minimal elements then $S(T) = (1, 1)$. Now by assumption $\downarrow y_1$ is atomic. Now if $x_2, y_2 \prec y_1$ then again with the same reason x_2 is a minimal element or y_2 is a minimal element.

Let x_2 be a minimal element. Then again if we consider y_2 and assume that $x_3, y_3 \prec y_2$ then we get another minimal element, x_3 and so on. Thus we get a sequence of minimal elements $x_1, x_2, ..., x_{n-2}$ where by their structure $l(x_i) = i$ for $i = 1, 2, ..., n-2$. Now when we consider y_{n-2} since T has n minimal elements this time x_{n-1} and y_{n-1} both are minimal elements and their lengths are both $n-1$. Thus $S(T) = (1, ..., n-2, n-1, n-1)$.

Now assume that $S(T) = (1, ..., n-2, n-1, n-1)$. We have to show that every principal open set is atomic. Let K be principal and $\bigvee K = e$. If $l(e) = s$ and $m, n \prec e$ we show that m is a minimal element or n is a minimal element. If neither of them is a minimal element then $T(m) = \{x \in T | x \leq m\}$ is an ordered tree so there are at least two minimal elements with the same length. We have the same result for n, too. Thus again $T(n) = \{x \in T | x \leq m\}$ has two minimal elements with the same length. Thus in the ordered tree there are 4 minimal elements with at most 2 distinct lengths where the construction of $S(T)$ doesn't allow this.

Thus either m or n is a minimal element. If both are minimal elements then $c(K) = \varnothing$ and hence K is atomic. Now if m is a minimal element but n is not, then $\downarrow n \in c(K)$ and is the maximum element of $c(K)$. Inductively we can get a finite sequence which shows that $c(K)$ is a chain.

(2) Let M be a strongly atomic open set and $l(\bigvee M) = s$. Then by what we have proved already if $x, y \prec \bigvee M$, one of them is a minimal element. Let x be a minimal element then $l(x) = s+1$. Now if $z, t \prec y$ and both are minimal elements then $c(M) = \{\{y, z, t\}\}$, has one element which contradicts M being strongly atomic. Thus one of z and t is a minimal element and the other is not, which means that we will have another two lengths aside from the length of x and the proof is complete.

□

Remark 2.11. We can improve the above theorem to an infinite ordered tree and notice that in this case we will have:

(1) Every principal open set in the lower topology is atomic if and only if $S(T) = (1, 2, 3, ...)$.
(2) Every strongly atomic open set consists at least 3 minimal elements with 3 different lengths.

3. Category of ordered trees

In this section we introduce the category of binary trees whose objects are binary trees and morphisms are strictly length preserving maps. Then we characterize the most balanced binary trees with 2^k leaves, $k \in \{0, 1, 2, ...\}$ as weak initial objects of this category.

Definition 3.1. *Let T_1, T_2 be ordered trees. A map $f : m(T_1) \to m(T_2)$ is length preserving if for minimal elements $m, n \in T_1$, $l(m) \leq l(n)$ implies $l(f(m)) \leq l(f(n))$.*

It is obvious that $l(x) = l(y)$ implies $l(f(x)) = l(f(y))$.

Theorem 3.2. *Let T_1, T_2 be ordered trees. Then every length preserving map $f : T_1 \to T_2$ can be extended to an order preserving map, $g : T_1 \to T_2$.*

Proof. Define $g : T_1 \to T_2$ such that for $a \in m(T_1)$, $g(a) = f(a)$ and for the other points $g(a) = \bigvee T_2$. Then clearly g is order preserving. □

Theorem 3.3. *The class of (finite) ordered trees with length preserving maps is a category. We denote it by ($\mathcal{FLT}$) $\mathcal{LT}$.*

Proof. This is because the composition of two length preserving maps is a length preserving map. □

Theorem 3.4. *$T_1, T_2 \in \mathcal{FLT}$ are isomorphic if and only if $S(T_1) = S(T_2)$.*

Proof. First assume that T_1 is isomorphic to T_2. Thus there are length preserving maps, $f : m(T_1) \to m(T_2)$ and $g : m(T_2) \to m(T_1)$ such that $f \circ g = id_{m(T_2)}$ and $g \circ f = id_{m(T_1)}$. This implies $|m(T_1)| = |m(T_2)|$. Also notice that $l(x) < l(y)$, $x, y \in T_1$, implies $l(f(x)) < l(f(y))$, because if $l(f(x)) = l(f(y))$ then $l(x) = l(g(f(x)) = l(g(f(y)) = l(y)$. Also if $l(x) = l(y)$ then $l(f(x)) = l(f(y))$, because if for example $l(f(x)) < l(f(y))$, then by the same proof $l(x) = l(g(f(x))) < l(g(f(y))) = l(y)$. Thus we can say $l(x) = l(y)$ if and only if $l(f(x)) = l(f(y))$. So the proof will be complete if we show that $l(x) = l(f(x))$. But $|\{l(x) : x \in T_1\}| = |\{l(x) : x \in T_2\}|$, also for every $a \in T_1$, $|\{x \in T_1 : l(x) = l(a)\}| = |\{x \in T_2 : l(x) = l(f(a))\}|$.

Now the only thing that has remained is, $\{l(x)|x \in T_1\} = \{l(x)|x \in T_2\}$. We prove this by induction on the cardinality of $|\{l(x)|x \in T_1\}|$. If $|\{l(x)|x \in T_1\}| = 1$, then $\{l(x)|x \in T_i\} = \{k_i\}$, for $i = 1, 2$. Then T_i has 2^{k_i}, where $i = 1, 2$, minimal elements and since $m(T_1) = m(T_2)$, $2^{k_1} = 2^{k_2}$ and hence $k_1 = k_2$.

Now assume that $|\{l(x)|x \in T_1\}| = |\{l(x)|x \in T_2\}| = p$ and let $l(T_i) = m_i$, for $i = 1, 2$. We show that $m_1 = m_2$. For this we derive isomorphic ordered trees T_i' from T_i for $i = 1, 2$ such that $l(T_i') = l(T_i) - 1$. For this take $T_i' = T_i - \{x \in m(T_i)|l(x) = m_i\}$, for $i = 1, 2$. Now $m(T_i') = A_i \dot{\cup} B_i$, for $i = 1, 2$ where $A_i = m(T_i) \cap m(T_i')$ and $B_i = m(T_i') - A_i$. Notice that $f(A_1) = A_2$ and $g(A_2) = A_1$. By the definition of ordered trees, $2|B_1| = |\{x \in m(T_1) : l(x) = m_1\}| = |\{x \in m(T_2) : l(x) = m_2\}| = 2|B_2|$. Since $|B_1| = |B_2|$, there is a one one and onto function, $h : B_1 \to B_2$. Now define $f' : m(T_1') \to m(T_2')$ such that $f'(x) = f(x)$, if $x \in A_1$ and $f'(x) = h(x)$ if $x \in B_1$ and define $g' : m(T_2') \to m(T_1')$ such that $g'(x) = g(x)$, if $x \in A_2$ and $g'(x) = h^{-1}(x)$ if $x \in B_2$. Then f' and g' are length preserving maps such that $g' \circ f' = id_m(T_1')$ and $f' \circ g' = id_m(T_2')$. Thus T_1' and T_2' are isomorphic.

Now if $n_1 = max\{l(x)|x \in T_1, l(x) < m_1\}$ and $n_2 = max\{l(x)|x \in T_2, l(x) < m_2\}$. Then $m_1 - n_1 = m_2 - n_2$. Assume the contrary for example $m_1 - n_1 < m_2 - n_2$ then if we reduce the length of T_1 and T_2 like above $m_1 - n_1$ times then the new ordered trees obtained from T_1 and T_2 are isomorphic, because in every stage of reducing the length we get isomorphic ordered trees. But the new ordered tree obtained from T_1 has $p-1$ different lengths and the new ordered tree obtained by T_2 has again p different lengths. But it is impossible to have two isomorphic ordered trees with different number of different lengths. So $m_1 - n_1 = m_2 - n_2$. Now we reduce the length $m_1 - n_1$ times in both ordered trees, then we will have two isomorphic ordered trees with $p-1$ different lengths and by using induction and by the proof, $\forall x \in T_1, l(x) < m_1, l(x) = l(f(x))$. Thus in particular $n_1 = n_2$ and since $m_1 - n_1 = m_2 - n_2$ we get $m_1 = m_2$. This proves that if for $x \in T_1$, $l(x) = m_1$ then $l(x) = l(f(x))$ and also we showed that $\forall x \in T_1, l(x) < m_1, l(x) = l(f(x))$. Thus for every $x \in T_1$, $l(x) = l(f(x))$ and the proof of this part is complete.

The converse of the theorem is trivial because of the definition of length, length preserving maps and $S(T)$. □

Now we introduce a useful subcategory of $\mathcal{LT}$ ($\mathcal{FLT}$) whose objects are the object of $\mathcal{LT}$ ($\mathcal{FLT}$) but whose morphisms are strictly length preserving maps-that is if T_1, T_2 are trees and $f : m(T_1) \to m(T_2)$, whenever $l(x) < l(y)$ for $x, y \in T_1$, then $l(f(x)) < l(f(y))$. We denote this subcategory by $\mathcal{SLT}$ ($\mathcal{FSLT}$).

Lemma 3.5. *If $f \in Mor(\mathcal{LT})$ is a morphism of $\mathcal{SLT}$ then $l(f(x)) = l(f(y))$ if and only if $l(x) = l(y)$.*

Proof. We know from length preserving maps, $l(x) = l(y)$ implies $l(f(x)) = l(f(y))$. Also if $l(f(x)) = l(f(y))$ then $l(x) = l(y)$ because if for example $l(x) < l(y)$ then $l(f(x)) < l(f(y))$ which contradicts $l(f(x)) = l(f(y))$. □

Theorem 3.6. *$T_1, T_2 \in \mathcal{FSLT}$ are isomorphic if and only if $S(T_1) = S(T_2)$.*

Remark 3.7. Theorems 3.4 and 3.6 are not true for infinite ordered trees. For example consider T_1 and T_2 such that $S(T_1) = (1, 2, 3, ...)$ and $S(T_2) = (2, 3, 4, ...)$. Define $f : m(T_1) \to m(T_2)$ such that $l(f(x)) = l(x) + 1$ and $g : m(T_2) \to m(T_1)$ such that $l(g(x)) = l(x) - 1$. Then $g \circ f = id_{m(T_1)}$ and $f \circ g = id_{m(T_2)}$. So T_1 is isomorphic to T_2 but $S(T_1) \neq S(T_2)$.

Now we recall from (cf [1]) that the skeleton subcategory of a category is a full subcategory such that for any object in the original category, there exists a unique isomorphic object in the skeleton subcategory

Theorem 3.8. *The skeletons of $\mathcal{FSLT}$ and $\mathcal{FLT}$ are the same.*

Proof. It is true because we can characterize the skeletons of both $\mathcal{SLT}$ and $\mathcal{LT}$ by the path-length sequences which is exactly the characterization of binary trees. □

Corollary 3.9. *Consider the binary tree T:*

(1) *T is the most imbalanced iff when we consider it as a semilattice every open set in T is atomic*
(2) *If T is the most balanced then it has no atomic open set and so has no strongly atomic open set.*

Proof. By the above theorem and Theorem 2.10. □

Now why have we introduced this special subcategory of $\mathcal{LT}$?

Because by this subcategory we can state some of the important issues in computer science which are useful in practice as categorical properties.

One of the most important types of binary trees in computer science are the most balanced binary trees. In the following theorem we characterize these binary trees by categorical structures. First we recall the following definition.

Definition 3.10. *An object, W in the category $\mathcal{C}$ is called weak initial if for every object of $\mathcal{C}$ there is at least one morphism from W to it.*

Theorem 3.11. *Every object of $\mathcal{LT}$ ($\mathcal{FLT}$) is weak initial.*

Proof. Consider A an object of $\mathcal{LT}$. Now for every object C of $\mathcal{LT}$, if $z \in m(C)$, then define $f : m(A) \to m(C)$, such that for every $x \in m(A)$, $f(x) = z$. □

Theorem 3.12. *$T \in obj(\mathcal{SLT})$ is weak initial if and only if it is the most balanced binary tree with 2^k leaves, $k \in \{0, 1, 2, ...\}$.*

Proof. Let T be the most balanced binary tree with 2^k leaves, $k \in \{0, 1, 2, ...\}$. Then for every binary tree W, define a constant map from $m(T)$ to $m(W)$ which maps every leaf of T to a fixed leaf of W. Then this map is strictly length preserving. Conversely let T be a binary tree such that for every binary tree W there is a strictly length preserving map from $m(T)$ to $m(W)$. If T is not the most balanced binary tree with 2^k leaves, $k \in \{0, 1, 2, ...\}$ then T has at least two leaves with different lengths. But in this case there is not any strictly length preserving map from T to the binary tree with 2 leaves. □

4. Composition of comparison based algorithms

Sometimes a c.b. algorithm (comparison based algorithm) $\mathcal{B}$ acts on some outputs of the c.b. algorithm $\mathcal{A}$. Here we study this situation, and give a formula for it. We try to show it as an algebraic operation.

Let the binary tree corresponding to $\mathcal{A}$ be $< a_1 ... a_m >$ and the binary tree corresponding to $\mathcal{B}$ be $< b_1 ... b_n >$. Let $\mathcal{C}$ be the algorithm obtained when $\mathcal{B}$ acts on the output of $\mathcal{A}$ at the leaf x_k of $\mathcal{A}$. We can obtain the binary tree corresponding to $\mathcal{C}$ by attaching a copy of B to the leaf x_k. So $C = [< c_1 ... c_{k-1}\, c_{k,1} ... c_{k,n}\, c_{k+1} ... c_m >]$ such that $c_i = a_i$ for $i = 1, ..., k-1, k+1, ..., m$ and $c_{k,i} = a_k + b_i$, for $i = 1, ..., n$ and has $(m-1)+n$ leaves. By $[< c_1 ... c_{k-1}\, c_{k,1} ... c_{k,n}\, c_{k+1} ... c_m >]$ we mean the increasing sequence obtaining from the sequence $< c_1 ... c_{k-1}\, c_{k,1} ... c_{k,n}\, c_{k+1} ... c_m >$. If $\mathcal{B}$ acts on more than one output of the algorithm $\mathcal{A}$ the final binary tree is obtained similarly.

Now consider c.b. algorithm $\mathcal{C}$ obtained when $\mathcal{B}$ acts on all outputs of the c.b. algorithm $\mathcal{A}$. So if we consider their correspondence binary trees, a copy of the binary tree corresponding to $\mathcal{A}$ is attached to every leaf of the binary tree corresponding to $\mathcal{A}$. Thus in this case the decision binary tree corresponding to $\mathcal{C}$ will be $[< c_{ij} >]$ such that $c_{ij} = a_i + b_j$, where $i = 1, ..., m$ and $j = 1, ..., n$. We denote $\mathcal{C}$ by $A * B$.

Now we try to formalize this definition. We bring the following from [3].

Definition 4.1. *A po-groupoid (or m-poset), $(M, ., \leq)$ is a poset M with a binary multiplication, ., which satisfies the isotonicity condition*

$$a \leq b \text{ implies } x.a \leq x.b \text{ and } a.x \leq b.x \qquad (*)$$

for all $a, b, x \in M$. When multiplication is (commutative) associative, M is called a (commutative) po-semigroup. A po-semigroup with identity 1, such that $x.1 = 1.x = x$ for all $x \in M$ is called a partly ordered monoid, or po-monoid. A po-group is $(G, +, \leq)$ such that $(G, +)$ is a group which satisfies $(*)$. An l-group is a po-group such that $(G, \leq)$ is a lattice.

If T_n is the set of binary trees with n leaves, $n \in \mathbb{N}$, then as was mentioned when $A \in T_n$ and $B \in T_m$ then $A * B \in T_{mn}$. Take $T = \bigcup_{n\in\mathbb{N}} T_n$. Then $* : T \times T \to T$ defined by $*(A, B) = A * B$ is a binary operation on T. We call $*$ product. Now we define $\leq_T$ on T as follows:
$A \leq_T B$ if and only if there is $n \in \mathbb{N}$ such that $A, B \in T_n$ and $A \leq B$.

Now we have the following theorem:

Theorem 4.2. *$(T, *, \leq_T)$ is a commutative po-monoid.*

Proof. One can easily see that if $A \in T_n$ and $B \in T_m$ then $A * B = B * A$. Also the binary tree with one leaf $(< 0 >)$, is the identity because for every $A \in T_n$ where $n \in \mathbb{N}$, $A* < 0 >=< 0 > *A = A$. Also notice that if $A \leq_T B$ then for every $C \in T$, $A * C \leq_T B * C$. Because if for example the binary tree A is obtained from the binary tree B via a ternary exchange then $A * C$ is obtained from $B * C$ via m ternary exchanges, where $C \in T_m$. □

Remark 4.3. Notice that $(T, *, \leq_T)$ is not group-because, for example there is no binary tree L such that $L* < 11 >=< 11 > *L =< 0 >$.

Corollary 4.4. *If the c.b. algorithm A is faster than the c.b. algorithm B, then for every c.b. algorithm C, $A * C$ is faster than $B * C$.*

Corollary 4.5. *If the algorithm C is the product of the c.b. algorithm B with A and the algorithm D is the product of A with B. Then C and D are the same.*

For proving the next theorem we need the definition of a multi-set.

Definition 4.6. *A multi-set is a finite set-like object in which order is ignored but multiplicity is explicity significant. Thus contrary to sets, multi-sets allow for the repetition of elements. Therefore multi-sets $\{1, 2, 3\}$ and $\{3, 2, 1\}$ are considered to be equivalent, but $\{1, 2, 2, 3\}$ and $\{1, 2, 3\}$ differ. Also for the*

multi-sets A and B by $A - B$ we mean the set A with the elements in common with the set B removed according to their multiplicity. For example if $\{1,2,2,2,3\}$ and $\{1,2,2,3,4\}$ then $A - B = \{2\}$.

Theorem 4.7. *Consider A, B and C in T then:*

(1) *$A^m = B^m$, where $m \in \mathbb{N}$ implies $A = B$.*
(2) *$A * C = B * C$ implies $A = B$*

Proof.

(1) It is enough to show that $A^2 = B^2$ implies $A = B$. Since $A^2 = B^2$, there exists some $n \in \mathbb{N}$ such that $A \in T_n$ and $B \in T_n$. Let $A =< a_1 \dots a_{n-1}\, a_{n-1} >$ and $B =< b_1 \dots b_{n-1}\, b_{n-1} >$. Assume that $A \neq B$ and let i be the first position that a_i and b_i differ. Assume $a_i < b_i$. Since $a_j = b_j$, for $j = 1, 2, \dots, i-1$, the multi-sets $\{a_j + a_k | j, k = 1, \dots, i-1\}$ and $\{b_j + b_k | j, k = 1, \dots, i-1\}$ are equal. Thus the multi-sets $A^2 - \{a_j + a_k | j, k = 1, \dots, i-1\}$ and $B^2 - \{b_j + b_k | j, k = 1, \dots, i-1\}$ are equal. Notice that since these sequences are increasing and $A^2 = B^2$, the smallest number that appears in A^2 is $2a_1$ and in B^2 is $2b_1$, so $a_1 = b_1$. Thus $i > 1$. Now the smallest element that appears in $A^2 - \{a_j + a_k | j, k = 1, \dots, i-1\}$ is $a_1 + a_i$ and the smallest element that appears in $B^2 - \{b_j + b_k | j, k = 1, \dots, i-1\}$ is $b_1 + b_i = a_1 + b_i > a_1 + a_i$ which contradicts $A^2 - \{a_j + a_k | j, k = 1, \dots, i-1\} = B^2 - \{b_j + b_k | j, k = 1, \dots, i-1\}$. Thus $A^2 = B^2$. Similar method yields that $A^m = B^m$ implies $A = B$.
(2) By the same technique we get $A = B$.

□

Theorem 4.8. *For the c.b. algorithms A_1, A_2, where $A_1 \in T_n$ and $A_2 \in T_m$:*

(1) *$\overline{T}_{A_1 * A_2}(nm) = \overline{T}_{A_1}(n) + \overline{T}_{A_2}(m)$.*
(2) *$T^W_{A_1 * A_2}(nm) = T^W_{A_1}(n) + T^W_{A_2}(m)$.*
(3) *$T^B_{A_1 * A_2}(nm) = T^B_{A_1}(n) + T^B_{A_2}(m)$.*

Proof.

(1) Let $A_1 =< x_1 \dots x_n >$ and $A_2 =< y_1 \dots y_m >$ then $A_1 * A_2 = [< z_{ij} >]$, $z_{ij} = x_i + y_j$, where $i = 1, \dots, n$ and $j = 1, \dots, m$. Thus $nm\overline{T}_{A_1 * A_2}(nm) = \sum_{i,j} z_{ij} = \sum_{i,j}(x_i + y_j) = \sum_i(\sum_j(x_i + y_j)) = \sum_i(mx_i + \sum_j y_j) = m\sum_i x_i + n\sum_j y_j = mn\overline{T}_{A_1}(n) + nm\overline{T}_{A_2}(m)$. So $\overline{T}_{A_1 * A_2}(nm) = \overline{T}_{A_1}(n) + \overline{T}_{A_2}(m)$
(2) $A_1 * A_2$ is obtained when we attach a copy of A_2 to every leaf of A_1. So the longest path-length in $A_1 * A_2$ is the leaf which is the result of the longest length in A_2 attaching to the longest length in A_1. So the longest length in $A_1 * A_2$ is the summation of longest lengths in A_1 and A_2. So $T^W_{A_1 * A_2}(nm) = T^W_{A_1}(n) + T^W_{A_2}(m)$
(3) It is proved similar to the previous part.

□

Definition 4.9. *A nontrivial Binary tree is called prime if it can not been written as the product of two nontrivial binary trees* (*binary trees with at least two leaves*) *otherwise it is decomposable.*

Remark 4.10. Notice that the binary tree with one leaf is not neither prime nor decomposable, we denote it by 1.

Theorem 4.11. *If the binary tree T is decomposable, then it is not atomic.*

Proof. Let T be the product of 2 nontrivial binary trees A and B. Then the length of every leaf of T is at least 2. Now consider $\bigvee T$, then there are x, y such that $x, y \prec \bigvee T$. Since T doesn't have any leaf with length less than 2, x, y are not leaves. Now $\{t|t \leq x\}$ and $\{t|t \leq y\}$ neither A and B contains the other one. So $c(T)$ is not a chain and so T is not atomic. □

Theorem 4.12. *Every element of T_n, when n is prime natural number is a prime binary tree.*

Proof. If $C \in T_n$ is the product of A and B, where $A \in T_j$ and $B \in T_k$, $j, k \in \{2, 3, ...\}$ then $A * B \in T_{jk}$. Thus $n = jk$ for $j, k \in \{2, 3, ...\}$, which is a contradiction. □

Remark 4.13. The converse of the previous theorem is not true. For example $< 1\,2\,3\,3 > \in T_4$ is prime while 4 is not a prime number.

I tried to prove a theorem like the fundamental theorem of arithmetic for binary trees in the sense that "Every binary tree is 1 or prime or can be decomposed uniquely apart from rearrangement to the prime binary trees." I worked on it for a while and I could not prove it. Then I decided to leave it in the article as a Conjecture. But while Mr Diarmuid Early was helping me with the English he found the following counter example.

Example 4.14. Consider $A =< 1\,2\,2 >$, $B =< 1\,3\,3\,3\,5\,5\,5\,5 >$, $C =< 1\,2\,3\,3 >$ and $D =< 1\,2\,4\,4\,4\,4 >$. Four of these binary trees are prime and $A * B = C * D =< 2\,3\,3\,4\,4\,4\,5\,5\,5\,5\,5\,5\,6\,6$
$6\,6\,7\,7\,7\,7\,7\,7\,7 >$. So it is seen that you can write a binary tree as the product of prime binary trees in several ways.

Acknowledgements. I would like to thank Dr Michel Schellekens and Mr Diarmuid Early for their useful comments.

References

[1] J. Adamek, H. Herrlich and G. Strecker, *Abstract and Concrete Categories*, John Wiley and Sons, Inc., 1990.

[2] A. Aho, J. Hopcroft and J. Ullman, *Data Structures and Algorithms*, Addison-Wesley Series in Computer Science and Information Processing, Addison-Wesley, 1983.

[3] G. Birkhoff. *Lattice Theory*, American Mathematical Society Colloquium Publications, Volume XXV, 1967.

[4] B. C. Pierce, *Basic category theory for computer scientists*, Foundations of computing series 1991.

[5] G. K. Gierz, K. H. Hofmann, K. Keimel, J. D. Lawson, M. W. Mislove and D. S. Scott, *A Compendium of Continuous Lattices*, Springer-Verlag, Berlin, 1980.

[6] D. Knuth, *The Art of Computer Programming* vol. 1, Addison-Wesley, 1973.

[7] D. Knuth, *The Art of Computer Programming* vol. 3, Addison-Wesley, 1973.

[8] R. Kopperman and R.Wilson, *On the role of finite, hereditarily normal spaces and maps in the genesis of compact Hausdorff spaces*, Topology Appl. **135** (2004), 265–275.

[9] M. O'Keeffe, H. Pajoohesh and M. Schellekens, *On the relation between balance and speed of algorithms*, Hadronic Journal, to appear.

[10] M. O'Keeffe, H. Pajoohesh and M. Schellekens, *Decision trees of algorithms and a semivaluation to measure their distance*, Electron. Notes Theor. Comput. Sci. (Proceeding of MFCSIT 2004), to appear.

[11] D. Stott Parker and P. Ram, *The construction of Huffman codes is a submodular ("convex") optimization problem over a lattice of binary trees*, Sociely for industrial and Applied Mathematics **28**, no. 5, 1875–1905.

Received June 2005

Accepted November 2005

H. Pajoohesh (hpajoohesh@mec.cuny.edu)
Medgar Evers College, Department of Mathematics, 1150 Carroll Street, Brooklyn, NY 11225-2298, USA.