Seven Trees in One*

Yuval Filmus

September 16, 2009

1 Introduction

This lecture follows the paper "Seven Trees in One" [1], with some material taken from [3] (see also [2]).

We will consider classes of binary trees — in fact, polynomials in binary trees. But first, what is a binary tree? Our binary trees (excluding the empty tree) have a recursive definition: a single vertex $\circ$ is a binary tree, and if T_1, T_2 are binary trees, then (T_1, T_2) is also a binary tree. We can write this formally as follows:

$$T = 1 + T^2.$$

But what does this mean?

First, let us try to understand the expression $1 + T^2$. This means "either 1 or T^2". Next, 1 represents some atom. And what does T^2 stand for? These are just pairs of binary trees. So each tree is equivalent to either an atom or to a pair of binary trees. The question is, equivalent in what sense?

Before answering this question, let us try to hypothesize what polynomials we can expect to be equal. We can calculate as follows (for example in $\mathbb{C}[T]$):

$$T^3 = T \cdot T^2 = T(T-1) = T^2 - T = -1.$$

So T is a 6th-degree primitive root of unity (certainly $T \neq -1$). Written differently, $T^6 = 1$, but we wouldn't expect *that* to be true. However it seems reasonable to expect

$$T^7 = T.$$

Hence, *seven trees in one*.

The simplest semantics one can think of is that $P = Q$ if there is a bijection between the sets P and Q. However, this definition is quite boring since for any non-constant P we have $|P| = |T|$, and so $P = Q$ for *any* infinitenon-constant polynomials P, Q. What can we do then?

*After a paper [1] by Andreas Blass

2 Definition of equality

Let us try a more sophisticated definition of equality. We will say that $P = Q$ if there is an algorithm that takes P and produces Q. We can think of P as a sum of monomials, and the algorithms takes as input the monomial number and an appropriate tuple of trees. Its output is also a monomial number and an appropriate tuple. The trees themselves have the following API:

1. Determine whether a tree is a leaf (single vertex) or of the form (T_1, T_2).
2. In case a tree is of the form (T_1, T_2), get T_1 and T_2.
3. Produce a single vertex.
4. Given trees T_1, T_2, produce a tree (T_1, T_2).

Does this definition help us? Consider the following bijection between T and $T + 1$:

1. The input $\circ$ is mapped to the 1.
2. A left-winding path of length $l > 0$ is mapped to a left-winding path of length $l - 1$.
3. Any other tree is mapped to itself.

It is easy to check that this is a bijection. Clearly, it can also be implemented algorithmically. So we have

$$T = T + 1.$$

From this it follows that

$$\begin{aligned} T^2 = T \cdot T = T(T+1) = T^2 + T = T^2 + (T+1) \\ = (T^2 + T) + 1 = T^2 + 1 = T, \\ T + T = T^2 + T + 1 = T^2 + T = T^2 = T. \end{aligned}$$

From these three equations we can deduce again that $P = Q$ for any non-constant P, Q.

The algorithm described above terminates because any left-winding path must terminate (all the trees we considered were finite). Suppose now that we also allow infinite binary trees — for example, all subtrees of the usual infinite binary tree. Now our previous algorithm gets stuck for an infinite left-winding path. But is it meaningful to allow infinite trees? If we look carefully at our definition, we see that when the time comes to output a tree, we simply take several subtrees of the input ("pointers") and combine them. It doesn't matter whether these are finite or not. It turns out that the implied definition will give us everything that we want.

2.1 The definition

We define $P \equiv_C Q$ if there is an algorithm from P to Q which always stops, and produces a bijection, for the class of infinite binary trees (subtrees of the usual infinite binary tree). Note that this class also includes all the finite binary trees.

2.2 Elementary properties of equality

We now prove a few properties of $\equiv_C$, culminating in the fact that it is an equivalence relation.

Lemma 1 (Compactness)**.** *Any algorithm proving $P \equiv_C Q$ has bounded running time.*

Proof. We can look at the running of the algorithm as a tree of computation. At each vertex there is a query of the form "$\circ$ or (T_1, T_2)?", and in the leaves we actually output something. Suppose now that the running time is unbounded. We thus have a binary tree with unbounded depth, so by König's lemma it must contain an infinite path (this path can be built by starting with the root and choosing always the edge leading to a subtree with unbounded depth). This inifinite path corresponds to a non-terminating computation, since we can actually produce an infinite binary tree conforming to this recipe (this corresponds to the infinite left-winding path in the example discussed earlier). □

Lemma 2 (Uniqueness of labels)**.** *In any algorthim proving $P \equiv_C Q$, during any output step each input "pointer" is used at most once.*

The pointers mentioned in the lemma are just the unexplored parts of the tree at the time of output. For example, suppose the input is T, we queried the input and it turns out it's of the form (T_1, T_2) (we call this a *tagged tree*). We then queried T_1 and it turns out that T_1 is a leaf. So the general form of the tree at this point is $(\circ, T_2)$, and the lemma is claiming that if we output at this point, then we use T_2 at most once.

Proof. We consider first the special case $Q = T$.

At each output step, the output is a tagged tree, viz. a finite binary tree whose leaves are either $\circ$ or pointers. We can consider the collection O of all of them. This collection must represent each possible binary tree exactly once. Moreover, by lemma 1 this collection is finite, so there is some bound D on the depth of all trees in O.

Now suppose O contains some binary tree B with repeated labels. Choose one of the repeated labels S, and instantiate B by replacing S with a left winding path of length D, and all other labels by a vertex. Denote this tree B_0, so that B_0 is an instance of B. Now attach to the end of each of these paths a right winding path, of different length for each. Denote this new tree B_1. This B_1 must be an instantiation of some $C \in O$. The set of labels whose descendants are the left-then-right winding paths must consist of different labels because

their descendants are some left-then-right winding paths with the right winding part of different lengths. Moreover, these labels cannot appear anywhere else since these paths are of length D, and there are no other vertices that deep in B_1. We see immediately that, in fact, B_0 is also an instantiation of C, so that the mapping is not injective. This proves the lemma for $Q = T$.

Clearly the same method of proof works for any monomial T^k (trivially for $T^0 = 1$). A general polynomial is a disjoint sum of monomials, and considering each of them separately we again find the lemma correct. □

Theorem 3. *The relation $\equiv_C$ is an equivalence relation.*

Proof. We have to prove three things: reflexivity, symmetry and transitivity.

Reflexivity is easy: clearly $P \equiv_C P$ by simply copying the input. Transitivity is also simple: if you have an algorithm transforming P to Q, and another one transforming Q to R, you can run the first algorithm without emitting any output, and then run the second algorithm on the interim results virtually — some of the queries you will be able to answer right away, and some of them you simply "forward" to the actual input.

The hardest part is proving symmetry. Suppose there is some algorithm transforming P to Q. We can look at this algorithm as a collection of transformations $A \to B$, where A and B are tuples of tagged trees (labelled with monomial numbers). For each monomial in P or Q, the relevant A's or B's form a partition of that monomial.

To build an algorithm transforming Q to P, we need to determine for each input what B covers it. The way to do this is as follows. Given a tuple of trees, we will "explore" the trees by querying the structure of each member of the tuple up to some depth k, where k is the maximum depth of a B tree (by lemma 1 there is a finite number of them). Given this structure, we can associate the proper B for the input, and apply the appropriate transformation $B \to A$. The resulting algorithm always terminates and is a bijection transforming Q to P. □

3 When are two polynomials equal

We now have a definition for equality, but how do we know whether two polynomials are equal? In order to answer that, we have to find some invariant which every transformation algorithm keeps. First, we introduce a normal form for algorithms:

Lemma 4. *Each algorithm proving that $P \equiv_C Q$ can be transformed into an algorithm which at each output step uses only pointers which were not queried. Moreover, it must use each of them exactly once.*

An example of an algorithm not consistent with this normal form is an algorithm transforming T to T by first exploring the root of its input, and then copying the root pointer to the output.

Proof. Suppose that in some output step we use some pointer T_1 which was queried. We can represent the knowledge about the subtree rooted in T_1 as a tagged tree, and replace T_1 in the output with that tagged tree. This way we only use pointers which were not queried. We must use each of these pointers because otherwise the mapping won't be injective, and by lemma 2 we can't use any of them more than once. □

Theorem 5. *If $P \equiv_C Q$ then there is a set of rules $(i, A) \to (j, B)$ such that:*

1. *i is an index of some monomial $T^d \in P$,*
2. *A is a d-tuple of tagged trees,*
3. *j is an index of some monomial $T^e \in Q$,*
4. *B is an e-tuple of tagged trees,*
5. *A and B have the same number of tags,*
6. *For each index i of a monomial in $T^d \in P$, the set of all A's appearing in rules of the form $(i, A) \to \cdot$ forms a partition of T^d which results from repeated queries of T^d,*
7. *For each index j of a monomial in $T^e \in Q$, the set of all B's appearing in rules of the form $\cdot \to (j, B)$ forms a partition of T^e.*

Such a set of rules (without the requirement that the A's result from repeated querying) is called a very elementary bijection in [1].

Proof. This follows immediately from lemma 4. □

Let's summarize what we know about the set of rules:

1. For each monomial $T^d \in P$, the different A's form a partition of T^d which results from repeated querying of T^d.
2. For each monomial $T^e \in Q$, the different B's form a partition of T^e.
3. In each rule, the number of tags in A and B is the same.

What can we say about a partition of T^d? Let's look onat a simple example. We know that $\circ$ and (T_1, T_2) partition the monomial T. It would be natural to express that as $T = 1 + T^2$. This inspires the following definitions:

1. The weight $w(A)$ of a tuple A of tagged trees with i tags is T^i.
2. The weight $w(S)$ of a set S of tuples is the sum of the weights of the individual tuples.

Looking at the process of repeated querying, we immediately deduce the following lemma:

Lemma 6. *For of any set $\S = \{(i, A)\}$ partitioning a polyonimal P and resulting from repeated querying, we have $w(S) = P$ in the semiring $R_T = \mathbb{N}[T]/(T = 1 + T^2)$.*

The semiring $\mathbb{N}[T]/(T = 1 + T^2)$ is defined by identifying two polynomials which can be proven equal using the normal axioms of semirings together with the equation $T = 1 + T^2$.

Proof. Each step of querying has the effect of substituting $1 + T^2$ for T in the intermediate expression for the total weight, so it does not change the total weight modulo $T = 1 + T^2$. The lemma follows since the weight before making any queries is exactly P. □

The lemma was easy because the partition was obtained by repeated querying. But in fact, we can prove it for any partition as follows.

Lemma 7. *The weight of any finite set $\{(i, A)\}$ partitioning a polynomial P is equal to P in the semiring $R_T = \mathbb{N}[T]/(T = 1 + T^2)$.*

Proof. Let d be the maximum depth of any tree in the partition. For any tuple A of tagged trees, we can use repeated querying to form a partition $W_d(A)$, all of whose tags (if any) are at depth exactly d. By lemma 6, $w(A) = w(W_d(A))$. In particular, $w(\{i, A\}) = w\left(\bigcup\{i, W_d(A)\}\right)$. The right hand side is independent of the partition, so by lemma 6 it must equal P. □

We immediately deduce the following theorem.

Theorem 8. *If $P \equiv_C Q$ then $P = Q$ in the semiring $R_T = \mathbb{N}[t]/(T = 1 + T^2)$.*

Proof. Consider the normal form given by theorem 5. Since in each rule $(i, A) \to (j, B)$ we have $w(A) = w(B)$, from lemma 7 we deduce that in R_T we have $P = w(\{(i, A)\}) = w(\{(j, B)\}) = Q$. □

Is the converse also true? To answer this, we have to provide a formal description for when two polynomial are equal in R_T.

Lemma 9. *All equations in the semiring R_T can be deduced from the following axiom schemes and deduction rules, where all the variables are fully parenthesized terms using the atoms $1, T$ and the operators $+, \cdot$:*

1. *Equality: $A = B$, $A = B \vdash B = A$, $A = B \wedge B = C \vdash A = C$.*
2. *Compatibility: $A = B \vdash A + C = B + C$, $A = B \vdash AC = BC$.*
3. *Addition: $A + B = B + A$, $(A + B) + C = A + (B + C)$.*
4. *Multiplication: $AB = BA$, $(AB)C = A(BC)$.*
5. *Distributivity: $A(B + C) = AB + AC$.*
6. *The special axiom $T = 1 + T^2$.*

Proof. This can be taken as a definition of the semiring modulo the relation. □

We can easily adjust the definition of $\equiv_C$ to accomodate these fully paranthesized terms:

1. 1 means an atom.
2. T means a binary tree (accessible with the old API).
3. $A + B$ means either an instance of A or an instance of B.
4. $A \cdot B$ means an instance of A together with an instance of B.

We can think of an algorithm as being presented its input and presenting its output in this form. The original definition is now a special case given some explicit encoding of the polynomials involved.

It is now easy to see that any derivation of $P = Q$ in R_T translates to an algorithm in our sense, since all the axiom schemes and derivation rules are valid for $\equiv_C$.

Theorem 10. *We have $P \equiv_C Q$ if and only if $P = Q$ in R_T.*

Proof. We already know one direction from theorem 8. For the other direction, we only need to verify that all the axiom schemes (and the axiom $T = 1 + T^2$) and derivation rules are valid if we replace equality in R_T by $\equiv_C$. We have already proved all axioms for equality in theorem 3. All the rest is very simple to verify. □

4 Deciding equality in R_T

In the previous section we have reduced the question of equivalence in the sense of $\equiv_C$ to equality in the semiring R_T. It is easy to give necessary conditions for equality there: if we take any semiring with an element x satisfying $x = 1 + x^2$, then $P \equiv Q \pmod{R_T}$ implies that $P(x) = Q(x)$ in the new semiring. There are two canonical examples, which we have already seen in the introduction.

Lemma 11. *If $P \equiv Q \pmod{R_T}$ then $P(\alpha) = Q(\alpha)$ for $\alpha = e^{2\pi i/6}$ and $P(\aleph_0) = Q(\aleph_0)$.*

Proof. In the ring $\mathbb{C}$, α satisfies $\alpha = 1 + \alpha^2$, as we have seen in the introduction. In the semiring of cardinals, $\aleph_0 = 1 + \alpha_0^2$. □

It follows that $T = T^n$ can only be true if $n \equiv 1 \pmod 6$. What about the converse? We will see that the two conditions mentioned in lemma 11 are in fact also sufficient. There are two cases. If $P = n$ for some $n \in \mathbb{N}$ then the conditions imply that also $Q = n$ so that trivially $P = Q$. Otherwise, P and Q must both be non-constant polynomials satisfying $P(\alpha) = Q(\alpha)$. What does this imply?

Lemma 12. *If $P(\alpha) = Q(\alpha)$ then for some polynomial R, $P + X = Q + X \pmod{T = 1 + T^2}$.*

Proof. Let us consider P, Q as elements of $\mathbb{Q}[T]$. Since $(P - Q)(\alpha) = 0$, $P - Q$ must be divisible by the minimal polynomial of α (over the rationals), namely $T^2 - T + 1$. Employing the division algorithm, we see that the quotient S has integral coefficients. Let us write it as $S = S_+ - S_-$ where S_+, S_- have positive coefficients, so that we have

$$P - Q = (T^2 + 1 - T)(S_+ - S_-).$$

If we expand this and put all negative terms in the other side of the equation, we get

$$P + TS_+ + (T^2 + 1)S_- = Q + (T^2 + 1)S_+ + TS_-.$$

Since all the polynomials have positive coefficients, this equation is actually true in the semiring $\mathbb{N}[T]$. Moreover, calculating in R_T we get

$$\begin{aligned} P + TS_+ + (T^2 + 1)S_- &= Q + (T^2 + 1)S_+ + TS_- \\ &\equiv Q + TS_+ + (T^2 + 1)S_- \pmod{T = 1 + T^2}. \end{aligned}$$

The lemma follows with $X = TS_+ + (T^2 + 1)S_-$. □

In order to complete the proof, we need to find out a way to cancel elements in R_T. We cannot always do that, for example $T \equiv 1 + T^2 \equiv 1 + T + T^3$ but $1 + T^3 \not\equiv 0$. However, if both P and Q are non-constant, it turns out that cancellation does work.

We begin with consequences of the simple observation $T \equiv T + (1 + T^3)$ mentioned above.

Lemma 13. *Define $[0] = 1 + T^3$. This element satisfies the following properties in R_T:*

1. $T + [0] = T$.
2. $P + [0] = P$ *for non-constant* P.
3. $T \cdot [0] = [0]$.
4. $P \cdot [0] = [0]$ *for all* $P \neq 0$.

Proof. The first equation follows by the computation mentioned earlier:

$$T + [0] = T + 1 + T^3 = 1 + T^2 = T.$$

Given any non-constant P, we can take any non-constant monomial of it, and repeatedly apply the rule $T^k = T^{k+1} + T^{k-1}$ until we reach T. Then we can add or remove $[0]$, and take our steps back. For example,

$$T^2 = T^3 + T + [0] = T^2 + [0]\,.$$

This proves the second equation.

The third equation is another simple calculation:

$$T \cdot [0] = T(1 + T^3) = T + T^4 = 1 + T^2 + T^4 = 1 + T^3 = [0]\,.$$

The fourth equation follows from distributivity. □

Using this new kind of infinite zero, we can define inverses. Since $1+T^3 = [0]$, we can think of T^3 as an inverse of 1, and use it to define all other inverses.

Lemma 14. *Define* $[-1] = T^3$ *and for any* $P \neq 0$, $[-P] = [-1]\,P$. *These elements satisfy the following properties in* R_T*:*

1. $1 + [-1] = [0]$.
2. $P + [-P] = [0]$.

Proof. Both parts are easy calculations:

$$\begin{aligned} 1 + [-1] &= 1 + T^3 = [0]\,, \\ P + [-P] &= P + P \cdot [-1] = P(1 + [-1]) = P \cdot [0] = [0]\,. \end{aligned}$$ □

We can now prove a cancellation lemma:

Lemma 15 (Cancellation)**.** *If* $P + X \equiv Q + X \pmod{R_T}$ *for non-constant* P, Q *then* $P \equiv Q \pmod{R_T}$.

Proof. We can assume $X \neq 0$. The proof is extremely easy:

$$P = P + [0] = P + X + [-X] = Q + X + [-X] = Q + [0] = Q.$$ □

It follows that the two conditions mentioned as necessary in lemma 11 are also sufficient.

Theorem 16. *We have* $P \equiv Q \pmod{T = 1 + T^2}$ *if and only if* $P(\alpha) = Q(\alpha)$ *for* $\alpha = e^{2\pi i/6}$ *and* $P(\aleph_0) = Q(\aleph_0)$.

Proof. We have already proved that these conditions are necessary in lemma 11. Now suppose both are true. By substituting $\aleph_0$, we se that either both P and Q are constant, or both are non-constant. If both are constant then by substituting α we see that this constant must be the same and so $P = Q$ trivially. If both are non-constant this is true by combining lemmas 12 and 15. □

Corollary 17. *We have* $P \equiv_C Q$ *if and only if* $P(\alpha) = Q(\alpha)$ *for* $\alpha = e^{2\pi i/6}$ *and* $P(\aleph_0) = Q(\aleph_0)$.

This implies that indeed $T^7 \equiv_C T$, just as we wanted in the introduction.

5 Notes

Let us note the correspondence between our sources and our presentation. The original paper [1] starts by defining very elementary bijections, which are very similar to the normal form of theorem 5, and proving that their existence is equivalent to equality in R_T. It then uses a normal form argument to prove theorem 16.

Our proof of theorem 16 is taken from [3], where the cancellation lemma is proved for general equations $T = P(T)$, where $P(T) \in \mathbb{N}[T]$ must satisfy $P(0) \neq 0$ and $\deg P \geq 2$. Moreover, assuming $T - P(T)$ is primitive (its coefficients have no common factor) and has no repeated complex roots, then a form of theorem 16 is true (we have to test all complex roots). The paper [2] additionally describes a criterion of equality for linear P.

The paper [1] is motivated by a 1990 comment by the eminent category theorist Lawvere. The papers [3] and [2] are follow-ups by formal type theorists.

The definition of $\equiv_C$ is our own idea, and it is geared toward proving compactness. We feel it is more natural than that of very elementary bijections. The original paper [1] goes on to prove another equivalent definition, basically existence of a proof that $P = Q$ in some intuinionstic logic with added axioms describing the properties of binary trees. Unfortunately the proof uses topos theory, and is not accessible to the general mathematical public.

5.1 Future research

We have seen two definitions of equality. Under one of them all polynomials collapse to the uninteresting semiring $\mathbb{N}[T]/(T = 1 + T)$, and under another to the interesting one $\mathbb{N}[T]/(T = 1+T^2)$. There are semirings which are in between the two, for example we can add the axiom $T = T + 2$ without collapsing everything to $\mathbb{N}[T]/(T = 1 + T)$. The question is whether we can supply a definition of equality which would produce such an intermediate semiring.

One way of doing that would be to restrict the set of binary trees on which the algorithm must stop in the definition of $\equiv_C$. In order to prove compactness we only need all trees with finitely many computable infinite paths (a single one would do for the proof, but we need the set to satisfy $T = 1 + T^2$). What happens if we define equality in this sense?

References

[1] Andreas Blass. Seven Trees in One. *J. Pure Appl. Algebra*, 103:1–21, 1995.

[2] Marcelo Fiore. Isomorphisms of generic recursive polynomial types. In *31st Symposium on Principles of Programming Languages (POPL 2004)*, pages 77–88. ACM Press, 2004.

[3] Marcelo Fiore and Tom Leinster. Objects of categories as complex numbers. *Advances in Mathematics*, 190:264–277, 2005.