# The Tree Evaluation Problem: Towards Separating P from NL

## Lecture #6: 26 February 2014

**Lecturer: Stephen A. Cook**

**Scribe Notes by: Venkatesh Medabalimi**

## 1. Introduction

Consider the sequence of complexity classes :

$$\mathsf{AC}^0(6) \subseteq \mathsf{NC}^1 \subseteq \mathrm{L} \subseteq \mathsf{NL} \subseteq LogCFL \subseteq \mathsf{AC}^1 \subseteq \mathsf{NC}^2 \subseteq \mathbf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PH}$$

As of today, we do not know if even one among the above sequence of containments is strict. In fact, it is open if $\mathsf{AC}^0(6)$=PH ! In this lecture by Steve we learn about an attempt aimed at separating **P** from NL using a specific problem called the Tree Evaluation Problem. Most of the contents of the talk are derived from the work "Pebbles and Branching Programs for tree evaluation" [1] by Steve Cook, Pierre McKenzie, Dustin Wehr, Mark Braverman and Rahul Santhanam. The problem chosen in this program is inspired by Michael Taitslin's FOCS 2005 submission, an attempt to separate NL and **P**.

## 2. Definitions

Let $T^h$ be the balanced rooted binary tree of height $h$, i.e having $h$ levels.Let $[k] = \{0, 1, ..., k-1\}$. We number the nodes of $T^h$ using heap numbering. The root is numbered one and in general the children of node $i$ are numbered $2i$ and $2i + 1$.

**Definition 1.** *In the Tree Evaluation Problem(TEP) on $T^h$ the input is a table defining a function $f_i : [k] \times [k] \to [k]$ for each internal node in $T^h$ and a number in $[k]$ for each leaf in $T^h$. The two versions of TEP which are of interest to us are*

- *$FT^h(k)$: For a given input, find the value of the root of $T^h$.*
- *$BT^h(k)$: In this boolean problem for a given input we need to determine whether the value of the root is* 1*.*

One can easily see that $TEP$ is in **P**. We wish to show that $TEP \notin L$ (and $TEP \notin \mathsf{NL}$).

Natural ways to solve the tree evaluation problem can be described using what we call pebbling algorithms. We use them to get our upper bounds for the tree evaluation problem.

**Definition 2** (Black Pebbling)**.** *The legal moves in a black pebbling game are as follows.*

- *Place a pebble on any leaf.*
- *If both children of node $i$ are pebbled, slide one of these pebbles to node $i$.*
- *Remove a pebble at any time.*

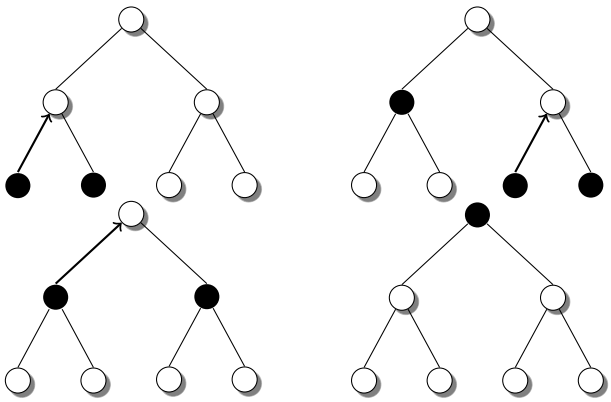

FIGURE 1. This figure illustrates a black pebbling of $T^3$ using 3 pebbles.

The goal in a black pebbling game is to place a pebble on the root. Figure 1 gives an example of a black pebbling scheme to pebble $T^3$ using 3 pebbles.

**Theorem 3.** *h pebbles are necessary and sufficient to black pebble* $T^h$

*Proof.* This is easily proved by induction on h. Another argument for the lower bound is to observe that the first time when the paths from root to all leaves are blocked forms a bottleneck pebbling configuration with at least $h$ pebbles. □

So black pebbling can be implemented by a turing machine with $O(k^h)$ states. So $FT^h(k) \in DSPACE(h \log k)$. The input size of a TEP instance is $n \approx (2^h - 1)k^2 \log k \implies \log n \approx h + \log k$.(So black pebbling isn't a log space algorithm.)

## 3. Branching Programs

**Definition 4** (k-way Branching Programs)**.** *A k-way branching program solving* $FT^h(k)$ *is a directed acyclic multigraph B with one source node* $\gamma_0$ *(the start state) and k sink nodes (the output states) labeled* $\{0, 1, ..., k-1\}$*. Each non-output state* $\gamma$ *has a label* $< i, a, b >$*, where i is a node in* $T^h$ *and* $a, b \in [k]$ *(if i is a leaf in* $T^h$*, then a,b, are missing). (The intention is that state gamma queries the function* $f_i(a, b)$*.) The state* $\gamma$ *has k out edges labeled* $0, 1, ..., k-1$*) (multiple edges can go to the same node). The* computation *of B on input I (describing an instance of* $FT^h(k)$*)) is a path in B, starting with the start state* $\gamma_0$*, and ending in an output state, where for each non-output state* $\gamma$ *querying* $f_i(a, b) = c$*, the next edge in the path is the one labeled c.*

**Theorem 5.** *For every* $k, h$*, there is a BP solving* $FT^h(k)$ *with* $O(k^h)$ *states. (Here the constant depends on h and is about* $2^h$*).*

*Proof.* We can derive such a BP from an efficient pebbling algorithm. Think of $B$ as layered, moving from left to right. Each level $i$ is associated with a node in $T^h$ – the next node to be pebbled in the pebbling algorithm. Thus there are $N + 1$ levels, where $N = 2^h - 1$ is the number of nodes in the tree $T^h$. At each level the states in the branching program correspond to different pebbling

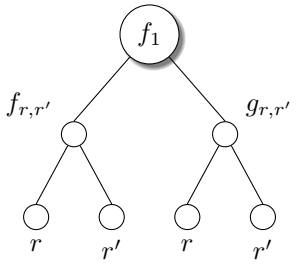


FIGURE 2. This figure depicts the sets of inputs $E^{r,r'}$ we consider to obtain $k^3$ lower bound for $FT^3(k)$.

configurations of $T^h$ that arise when the node corresponding to that level is pebbled. Since the maximum number of pebbles used in any configuration for an efficient pebbling algorithm on $T^h$ is $h$ the number of states in each level is $O(k^h)$. Since $N = 2^h - 1$, the number of states in the BP is $O(k^h)$. □

**CONJECTURE**: Every deterministic k-way BP solving $FT^h(k)$ has at least $k^h$ states, for $k, h \geq 2$.

There's a $100 prize for the first person to disprove this conjecture– see "Barriers Workshop Slides" on Steve's website for more on this.Instead, if you prove it you will be famous for the following reason.

**Theorem 6.** *To show $L \neq P$ it suffices to show that any k-way BP solving $FT^h(k)$ requires $\Omega(k^{c_h})$ states for some unbounded sequence $c_h$. (The constant in $\Omega$ depends on $h$.)*

*Proof.* Observe that a general TEP problem instance has input size $n = (2^h k^2 \log k)$ bits, so $\log n = (h + \log k)$.Let $\{c_h\}$ be any unbounded sequence, indexed by $h$. Suppose we can show that any k-way BP solving $FT^h(k)$ requires at least $\Omega(k^{c_h})$ states as a function of $k$. Then the space required is $c_h \log k = \omega(\log n)$ if we take $h = O \log k$, as $k \to \infty$. □

**Theorem 7.** *Conjecture 3 holds for $h = 2, 3$. (It's wide open for $h = 4$).*

*Proof.* The proof is obvious for $h = 2$ since there must be a state which queries each of the $k^2$ inputs $f_1(a, b)$ for $a, b \in [k]$.

(Proof for $h = 3$): Assume that $f_1(a, b) = (a + b) \mod k$. Let $B$ be a deterministic BP solving TEP for $T_2^3$. Let $C(I) = C$ be a complete computation of $B$ on input $I$. For $r, r' \in [k]$, let $E^{r,r'}$ be the set of all inputs in which the leaves $v_4, v_5, v_6, v_7$ are assigned $r, r', r, r'$ respectively(so an input in $E^{r,r'}$ is specified by the functions $f, g$ assigned to nodes 2 and 3). Figure 2 describes these inputs. Let $Q_{r,r'}$ be the set of all states q which query either $f(r, r')$ or $g(r, r')$. The theorem follows if we prove the following claim.

Claim 1: $|Q_{r,r'}| \geq k$.(So $B$ has at least $k^3$ states, since this is true for all pairs $k^2$ pairs $(r, r')$)

Fix $r, r'$. For every pair $a, b \in [k]$ define input $I_{a,b}$ as follows.Let $I_{a,b}$ be the input in $E^{r,r'}$ such that $f(r, r') = a$ and $g(r, r') = b$, and $f(x, y) = g(x, y) = 0$ if $(x, y) \neq (r, r')$.(Note that for input $I_{a,b}$ we have $v_2 = a$ and $v_3 = b$. Now claim 1 follows from the next claim:

Claim 2: For each state $\gamma$ in $Q_{r,r'}$ there are at most $k$ pairs $(a, b)$ such that $\gamma$ is the last state in $C(I_{a,b})$ which is in $Q_{r,r'}$.

Note that claim 1 and so the theorem would follow if we prove claim 2.

*Proof.* (of claim 2) Suppose otherwise, so some $\gamma$ in $Q_{r,r'}$ is the last such state in $C(I_{a,b})$ for more than $k$ pairs $(a, b)$. Then there exist $a, a', b \in [k]$ such that $a \neq a'$ and $\gamma$ is the last such state in both $C(I_{a,b})$ and $C(I_{a',b})$. Then the output of both computations is the same, but $a + b \mod k \neq a' + b \mod k$. □

This proves that the number of states needed in a $k-$way BP solving $FT^3$ is at least $k^3$, resolving the conjecture for $h = 3$. □

What about the boolean problem $BT^3(k)$?

**Theorem 8.** *An optimal k-way BP solving $BT^3(k)$ has $\Theta(k^3/\log k)$ states.*

*Proof.* (Outline for the Upper bound) Learn node 2, but partition possible values into $k/m$ blocks of size $m = \log k - \log\log k$, and remember block number. Now query node 3, and use $v_3$ to make $m$ queries to node 1 using the the m possible values of $v_2$, and remember the list of possible values of $v_1$. Now query node two again, to obtain the correct value of the root.

To get the lower bound, we cannot use the above method we used for $FT^3$ because of the following reason.

Claim: $BT^3$ can be solved in $O(k^2 \log k)$ states when the root function is $+ \mod k$.

*Proof.* For each setting of the four leaves put in $O(\log k)$ states which, for each $j$, query $v_2$ and $v_3$ in order to determine the $j^{th}$ bits in the binary notation for $v_2, v_3$.This allows us, for example to determine whether $v_2 + v_3 = k + 1$. Note that if $v_2 + v_3 = 1 \mod k$, then $v_2 + v_3$ is either 1 or $k + 1$. Since there are $O(k^2)$ settings of the values for leaves this BP has $O(k^2 \log k)$ states. □

In light of the above claim we have to make some changes to the approach we took in theorem 7. We proceed similarly except instead of fixing $f_1$ to be addition mod $k$, we let a boolean function $f_1$ be part of the input. So there are $2^{K^2}$ possibilities for $f_1$. As before let $Q_{r,r'}$ be the set of states which query either $f_2(r, r')$ or $f_3(r, r')$. It suffices to show that $|Q_{r,r'}| \geq k/\log k$. Let $s = |Q_{r,r'}|$. For each fixed $f_1$(and fixed values for all inputs except $f_2(r, r')$ and $f_3(r, r')$) the branching program has the same $s$ states, but the edges and labels between them change as $f_1$ changes. There are $2^{k^2}$ possibilities for $f_1$, but at most $(s+2)^{sk}$ ways of putting in labeled edges between the $s$ states. This is because including the two binary output states there are $s + 2$ choices for the destination of each labeled edge. By taking logs of both sides and assuming $s + 2 \leq k$ (since otherwise we are done) we

obtain $s \geq k/\log k$ as required. Since the same is true for all pairs $r, r'$ any BP solving $BT^3(k)$ has at least $k^3/\log k$ states. □

## 4. Thrifty Branching Programs

**Definition 9.** *Let $B$ be a BP which solves $BT^h$ or $FT^h$. Let $I$ be an input to $B$, and let $C(I)$ be the resulting computation (sequence of states). A query to a node $i$ during $C(I)$ is* thrifty *if either $i$ is a leaf, or $i$ is an internal node and the query is $f_i(a, b)$, where a,b are the values of the children of $i$. We say $B$ is a* thrifty *BP if every query for every input $I$ is thrifty.*

Note that if B follows the pebbling algorithm, then B is thrifty.

**Theorem 10.** *Every thrifty BP solving $BT^h$ or $FT^h$ has at least $k^h$ states.*

*Proof.* (David Liu's MSc thesis [2]) Consider a computation $C(I)$ of $B$ on input $I$.

Claim: Every node is queried, and for every non-leaf node $j$ queried in $C(I)$, both children of $j$ were queried earlier.

*Proof.* Suppose the root node is not queried in $C(I)$ then consider any input which is same as $I$ but differs from $I$ on only the value of the thrifty query at the root. Clearly $B$ would give a wrong answer for $I'$. So $C(I)$ makes a query to the root node. Now assume for some node $j$ there exists a child $i$ such that $B$ does not query $i$ in the path leading to some state $\gamma$ that queries $j$. Then consider the input $I'$ which is same as $I$ but different from it only in the value of the thrifty query at node $i$. Observe that for $I'$, $B$ would make a non-thrifty query at $j$. It also follows that every node is queried at least once. □

Lets define a *critical state* for the root to be the last state in $C(I)$ which queries the root. The critical state for a non root node $i$ is the last state which queries $i$ before the critical state for the parent of $i$.

We now use the critical states to define a black pebbling. We make one move at each critical state.

- At a critical state for a leaf, put a pebble on the leaf.
- At the critical state of an internal node $i$, slide a pebble on one of its children to $i$, and remove the pebble on the other child.

This is a valid pebbling.

Define the *supercritical state* to be the critical state in $C(I)$ immediately after the state which causes $h$ pebbles to be present.

Associate a tag $U(I) = (\gamma, v, x)$ with $I$ as follows:

- $\gamma$ is the supercritical state
- $v \in [k]^{N-h}$ ($N$ is the number of nodes in $T^h$) is string expressing all correct node values except the first $h$ values learned by $C(I)$ after $\gamma$.
- $x \in [k]^R$ gives all the non-thrifty values for I (i.e. the entire input except the values for the nodes.)

Claim: $U(I)$ uniquely determines $I$.

*Proof.* Note that the $v$ and $x$ in the tag together specify $I$ except for the values of the pebbled nodes. We can determine the values of the pebbled nodes by following the computation path starting from $\gamma$ onwards. Note that all these pebbles will be removed in $C(I)$ subsequently and by the way we defined black pebbling, the parent of any pebbled node is queried immediately before the pebble is removed. Also by the way we chose critical states no pebbled node is ever queried. Since this query is thrifty we can deduce the value of its pebbled children. □

From the above claim it follows that there must be at least $k^h$ supercritical states $\gamma$.This is because there are $k^{h+(N-h)+R}$ different inputs $I$, and $k^{(N-h)+R}$ different pairs $v, x \in U(I)$. □

Moving a little further from thrifty we explored if allowing queries to be on completely wrong values can add power to thrifty branching programs. We realized they don't.

**Definition 11** (Wrong-Wrong)**.** *A query $f_i(a, b)$ is wrong-wrong iff $a \neq v^I_{2i}$ and $b \neq v^I_{2i}$.*

**Theorem 12** (Dustin Wehr's MSc Thesis [3])**.** *: For any $h, k \geq 2$, if $B$ is a deterministic BP that solves $BT^h(k)$ and all queries are either thrifty or wrong-wrong, then $B$ has at least $k^h$ states.*

Another restricted class of branching programs that have the same lower bound are the read-once branching programs.

**Definition 13** (Read-Once Branching Programs)**.** *A deterministic read-once branching program is one in which no input variable is queried more than once along any path in the branching program.*

**Theorem 14** (James Cook/Siu Man Chan and later David Liu)**.** *Any deterministic read-once branching program solving $FT^h$ has atleast $k^h$ states.*

These are two very different and interesting proofs.

## 5. Non Deterministic Branching Programs

**Definition 15** (Nondeterministic Branching Program)**.** *A nondeterministic $k$-way branching program solving $FT^h$ is a directed rooted multigraph with one source node $\gamma_0$ (the start state) and $k$ sink nodes (the output states) labeled $\{0, 1, ..., k-1\}$. Each non-output state gamma has a label $< i, a, b >$, where $i$ is a node in $T^h$ and $a, b \in [k]$ (if $i$ is a leaf in $T^h$, then a,b, are missing). (The intention is that state gamma queries the function $f_i(a, b)$.) The state $\gamma$ has out edges with label in $[k]$ (multiple edges can go to the same node and some entries in $[k]$ may not be used as labels). A* computation *on input $I$ (describing an instance of $FT^h(k)$) is a path in $B$, starting with the start state $\gamma_0$ and proceeding such that for each non-output state $\gamma$ querying $f_i(a, b) = c$ (or a leaf $v = c$), the next edge in the path is any edge labeled $c$. A computation path on input $I$ either ends in a final state labeled $FT^h(I)$ or it ends in a non-final state labeled querying $f_i(a, b) = c$ (or a leaf $v = c$) with no out-edge labeled $c$(In this case we say the computation aborts). For every input $I$ at least one such computation must end in a final state.*

We define what we call as Black/White pebbling to describe one of the ways to obtain the upper bounds for tree evaluation problem in the non-deterministic setting.

**Definition 16** (Black/White pebbling)**.** *The legal moves in a black/white pebbling game are as follows*

- *A white pebble can be placed at any node at any time.*
- *A white pebble can be removed if the node is a leaf or both its children have pebbles.*
- *A black pebble can be placed at any leaf.*
- *If both children of node $i$ are pebbled, place a black pebble at $i$ and remove any black pebbles at the children.*
- *Remove a black pebble at any time.*

The goal of a black/white pebbling scheme is to start and end with no pebbles but to have a pebble at the root at some time. The minimum number of black/white pebbles needed, the B/W pebbling number for $T^h$ is $\lceil h/2 \rceil + 1$. Figure 3 describes how $T^4$ can be black/white pebbled with 3 pebbles.

**Corollary 17.** *A non-deterministic BP can solve $BT^h$ with $O(k^{\lceil h/2 \rceil + 1})$ states.*

We observe that for the height 3 case Neciporuk gives a $k^{2.5}$ lower bound that doesn't match the $k^3$ upperbound obtained from Black/White pebbling. This brings us to the notion of fractional pebbling [1] where we allow pebbles to be partly white and partly black. We show that $h/2 + 1$ fractional pebbles are necessary and sufficient for $T^h$. Hence $\Theta(k^{2.5})$ is the non-deterministic complexity for BPs solving $BT^3$.

## 6. Main Open Questions

To conclude we list some of the main open questions apart from the conjectures stated earlier in the lecture.

- Extend the deterministic $k^h$ lower bound to apply to $FT^4$? (This would be a minor breakthrough, since Neciporuk does not apply – at least not to degree 4 trees. Refer page 6:43 in [1] for more on this.).
- Come up with lower bounds for the Nondeterministic Semantic Read-once Branching Programs. Semantic Read-Once restriction requires that a BP query any node only once along an accepting path.
- Come up with lower bounds for the Nondeterministic thrifty BPs solving TEP.

## References

[1] S. Cook, P. McKenzie, D. Wehr, M. Braverman, and R. Santhanam, *Pebbles and branching programs for tree evaluation*, ACM Transactions on Computation Theory (TOCT), 3 (2012), p. 4.

[2] D. Liu, *Pebbling arguments for tree evaluation*, CoRR, abs/1311.0293 (2013).

[3] D. Wehr, *Pebbling and branching programs solving the tree evaluation problem*, CoRR, abs/1002.4676 (2010).

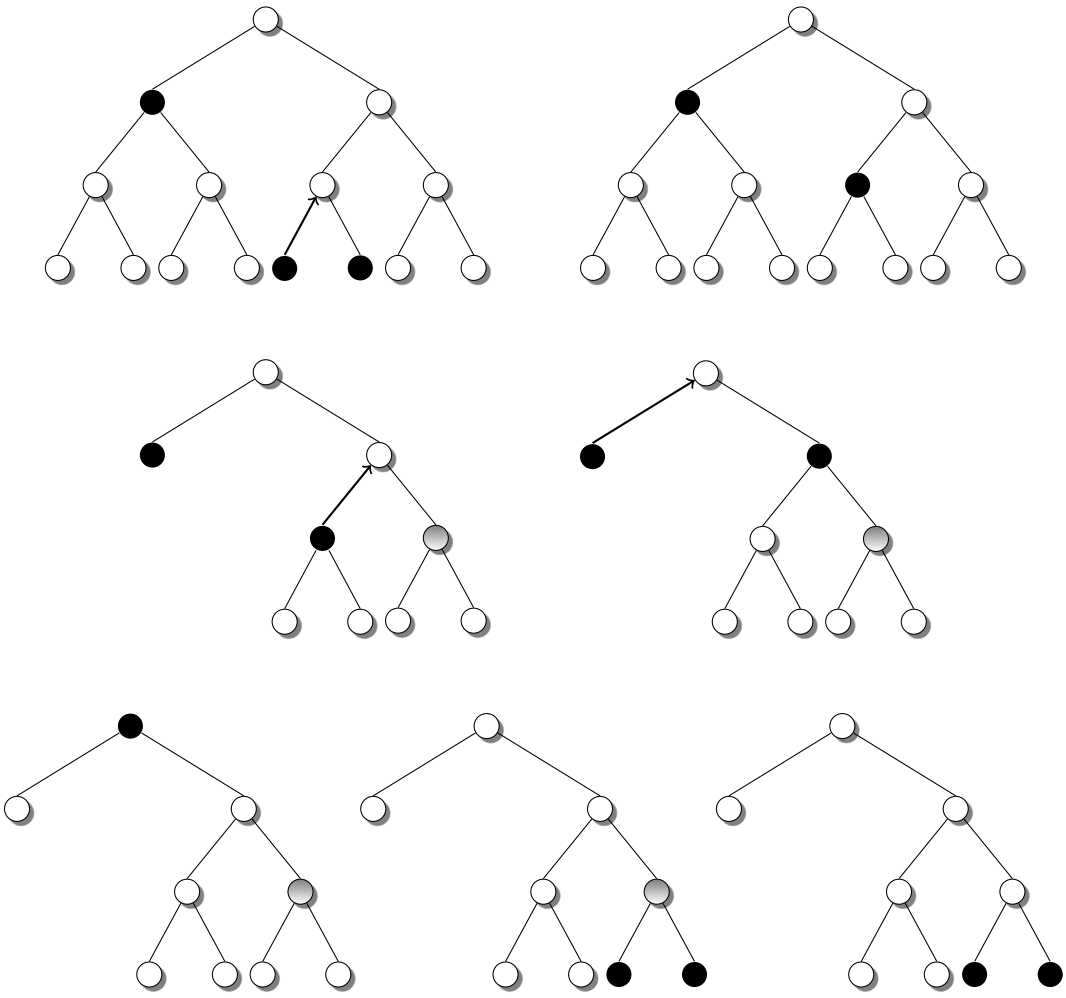

FIGURE 3. This figure shows a black/white pebbling of $T^4$ using 3 pebbles. We start with pebbling the root of left subtree