## Ryan Williams: Simulating Time with Square-Root Space

## James Cook, Ian Mertz: Tree Evaluation Is In Space $\mathcal{O}(\log n \log\log n)$

**presented by Petr Chmel**

**Theorem 1** (Simulating time in square-root space)**.**
For multi-tape Turing machines and every function $t(n) \geq n$,

$$\text{TIME}[t(n)] \subseteq \text{SPACE}[\sqrt{t(n)\log t(n)}].$$

**Theorem 2** (Simulating time in square-root space, talk version)**.**
For multi-tape Turing machines and every function $t(n) \geq n^2$,

$$\text{TIME}[t(n)] \subseteq \text{SPACE}[\sqrt{t(n)}\,\log t(n)].$$

**Proposition 1** (Obtaining oblivious Turing machines; Hennie, Stearns'66, Pippenger, Fischer'79, Fortnow, Lipton, van Melkebeek, Viglas'05)**.**
For every time-$t(n)$ multitape Turing machine $M$, there is an equivalent time-$T(n)$ two-tape Turing machine $M'$ which is oblivious, with $T(n) \leq \mathcal{O}(t(n)\log t(n))$. Furthermore, given $n$ and $i \in [T(n)]$ specified in $\mathcal{O}(\log t(n))$ bits, the two head positions of $M'$ on a length-$n$ input at time step $i$ can be computed in $\text{poly}(\log t(n))$ time.

**Corollary 1** (Separation of linear space and almost-quadratic time)**.**
For space constructible $s(n) \geq n$ and all $\varepsilon > 0$, $\text{SPACE}[s(n)] \not\subset \text{TIME}[s(n)^{2-\varepsilon}]$.

**Corollary 2** (A specific language requiring almost-quadratic time)**.**
The language $L = \{\langle M, x, 1^k\rangle \mid |M| \leq k \text{ and } M(x) \text{ halts in space } k\}$ requires $n^{2-\varepsilon}$ time to solve on a multitape Turing machine, for every $\varepsilon > 0$.

## Part 2: Tree Evaluation

**Definition 1** (Tree Evaluation)**.**
Tree Evaluation on trees of bit-length $b$, maximum height $h$, and fan-in $d$ is the following problem. On input, you are given a rooted $d$-ary tree of maximum height $h$ such that each leaf is labelled by a bit-string of length $b$, and every internal node $u$ has a function $f_u : \{0,1\}^{d\cdot b} \to \{0,1\}^b$. The goal is to compute the value of the tree at the root.

**Theorem 3** (Tree Evaluation in small space)**.**
Tree Evaluation on trees of bit-length $b$, maximum height $h$, and fan-in *at most* $d$, can be computed in $\mathcal{O}(d \cdot b + h\log(d \cdot b))$ space.

**Theorem 4** (Tree Evaluation in small space, talk version)**.**
Tree Evaluation on trees of bit-length $b$, maximum height $h$, and fan-in 2, can be computed in $\mathcal{O}((b + h) \cdot \log b)$ space.

**Definition 2** (Register program)**.**
A register program over an alphabet $\Sigma$ consists of a collection of memory locations $R = \{R_1, \ldots, R_s\}$, called registers, each of which can hold one element from $\Sigma$, and an ordered list of instructions in the form of updates to some register $R_i$ based on the current values of the register and an input $x \in \{0,1\}^n$.

**Definition 3** (Clean computation)**.**
Let $\mathcal{R}$ be a ring and let $f$ be a function whose output can be represented in $\mathcal{R}$. A register program over the alphabet $\mathcal{R}$ with $s$ registers *cleanly computes* $f$ into a register $R_o$ if for all possible $x_1, \ldots, x_n \in \{0,1\}^b$ and $\tau_1, \ldots, \tau_s \in \mathcal{R}$, if the program is run after initializing each register $R_i = \tau_i$, then at the end of the execution, $R_o = \tau_o + f(x_1, \ldots, x_n)$ and $R_i = \tau_i$ for all $i \neq o$.

**Definition 4** (Inverse clean register program)**.**
If $P$ is a register program that cleanly computes $f(x_1, \ldots, x_n)$, an inverse to $P$ is any program $P^{-1}$ that cleanly computes $-f(x_1, \ldots, x_n)$.

**Observation 1** (Inverse clean register programs exist)**.**
For any clean register program $P$, its inverse exists.

**Definition 5** (Uniform register programs)**.**
A family of register programs $P = \{P_n\}_{n\in\mathbb{N}}$ is space $c(n)$-*uniform* if there is an algorithm using space $c(n)$ which, given $(t, x)$ and access to an array of registers, performs the $t$-th instruction of $P_n$ on input $x \in \{0,1\}^n$.

**Proposition 2** (Register programs and space complexity)**.**
For $n \in \mathbb{N}$, let $c := c(n), s := s(n), t := t(n) \in \mathbb{N}$, and let $\mathcal{R} := \mathcal{R}_n$ be a ring. Let $f := f_n$ be a Boolean function on $n$ variables and let $P := P_n$ be a space $c$-uniform register program with $s$ registers over $\mathcal{R}$ and which has $t$ instructions in total, that cleanly computes $f$. Then $f$ can be computed in space $\mathcal{O}(c + s\log|\mathcal{R}| + \log t)$.

**Definition 6** (Root of unity)**.**
An element $\omega$ of a field $\mathcal{K}$ is a root of unity of order $m$ if $\omega^m = 1$. It is a primitive root of unity, if, moreover, $\omega^k \neq 1$ for all integer $0 < k < m$.

**Proposition 3** (Roots of unity are useful)**.**
Let $\mathcal{K}$ be a finite field and let $\omega$ be a primitive root of unity of order $m$ in $\mathcal{K}$. Then, for all $0 < b < m$, $\sum_{j=1}^{m} \omega^{jb} = 0$. (Note that for $b = 0$, the sum equals $-1$.)

**Lemma 1** (Roots of unity are *really* useful)**.**
Let $\mathcal{K}$ be a finite field, let $m = |\mathcal{K}| - 1$ and let $\omega$ be a primitive root of unity of order $m$ in $\mathcal{K}$. Let $d < m$ and let $\tau_i, x_i$ be elements of $\mathcal{K}$ for $i \in [d]$. Then

$$\sum_{j=1}^{m}\prod_{i=1}^{d}(\omega^j\tau_i + x_i) = -1 \cdot \prod_{i=1}^{d} x_i.$$

Moreover, let $q : \mathcal{K}^n \to \mathcal{K}$ be a degree-$d$ polynomial over $\mathcal{K}$. Then

$$\sum_{j=1}^{m} -1 \cdot q(\omega^j\tau_1 + x_1, \ldots, \omega^j\tau_n + x_n) = q(x).$$

**Lemma 2** (Clean computation of one node of TEP)**.**
Let $\mathcal{K}$ be a finite field such that $m := |\mathcal{K}| - 1 > 2b$. Let $u$ be a non-leaf node in the instance of Tree Evaluation with bit-length $b$ and height $h$. Let $P_\ell, P_r$ be register programs which cleanly compute the values $v_\ell, v_r \in \{0,1\}^b$ at $u$'s children into registers $R_\ell, R_r \in \mathcal{K}^b$, respectively, and let $P_\ell^{-1}, P_r^{-1}$ be their inverses. Let $f_u : \{0,1\}^{2b} \to \{0,1\}^b$ be the function at node $u$.

Then there exists a register program $P_u$ that cleanly computes $v_u = f(v_\ell, v_r) \in \{0,1\}^b$ into registers $R_u \in \mathcal{K}^b$, as well as an inverse program $P_u^{-1}$. Both $P_u, P_u^{-1}$ consist of $m$ copies each of $P_\ell, P_r, P_\ell^{-1}, P_r^{-1}$ and $5mb$ other instructions. Additionally, both programs only use registers $R_u$ plus any other registers used by $P_\ell, P_r$.