On the Cook-Mertz Tree Evaluation procedure

Oded Goldreich
Department of Computer Science
Weizmann Institute of Science, Rehovot, Israel.

December 27, 2024

Abstract

The input to the Tree Evaluation problem is a binary tree of height h in which each internal vertex is associated with a function mapping pairs of ℓ-bit strings to ℓ-bit strings, and each leaf is assigned an ℓ-bit string. The desired output is the value of the root, where the value of each internal node is defined by applying the corresponding function to the value of its children.

We provide an exposition and a digest of the recent result of Cook and Mertz (*ECCC*, TR23-174), which asserts that Tree Evaluation problem can be solved in space $O((h+\ell)\cdot\log\ell)$. In particular, we point out that the algebraic manipulation (using roots of unity) performed in the original work is merely a special case of univariate polynomial interpolation. Using this observation we provide a more transparent exposition of their main result as well as its low order quantitative improvement (i.e., space complexity $O(\ell+h\cdot\log\ell)$).

Our exposition refers to the "global storage" model rather than to the "catalytic storage" model used by Cook and Mertz, which can be viewed as a special case. We believe that the global storage model is more flexible and intuitive, but our exposition can be easily adapted to the catalytic storage model.

Contents

ISSN 1433-8092

1 Introduction

We provide an alternative exposition of a result of Cook and Mertz [5, Thm. 10] that asserts that the Tree Evaluation problem can be solved in space $O(\log n \cdot \log\log n)$, where n denotes the length of the input. For the history and significance of this problem, see [5]. Here we only mention that the Tree Evaluation problem was promoted in [2] towards separating $\mathcal{L}$ from $\mathcal{P}$. The problem was conjectured to require space $\Omega(\log^2 n)$, but this conjecture was already refuted in [3, 4], who showed an algorithm that uses $O((\log^2 n)/\log\log n)$ space. The result of [5] is far more dramatic.

More precisely, the input to the Tree Evaluation problem, denoted $\texttt{TrEv}_{h,\ell}$, is a rooted binary tree of height h in which internal nodes represent arbitrary gates mapping pairs of ℓ-bit strings to ℓ-bit strings, and each leaf carries an ℓ-bit string. Specifically, nodes in the tree are labelled by strings of length at most h such that the nodes $u0$ and $u1$ are the children of the node $u \in U \stackrel{\rm def}{=} \bigcup_{i=0}^{h-1}\{0,1\}^i$. For every $u \in U$, the internal node u is associated with a gate $f_u : \{0,1\}^{\ell+\ell} \to \{0,1\}^\ell$, and the leaf $u \in \{0,1\}^h$ is assigned the value $v_u \in \{0,1\}^\ell$. Hence, the input is the description of all $|U| = 2^h - 1$ gates (i.e., all f_u's) and the values assigned to the 2^h leaves; that is, the length of the input is $(2^h - 1)\cdot(2^{2\ell}\cdot\ell) + 2^h\cdot\ell = \exp(\Theta(h+\ell))$. The desired output is v_λ such that for every $u \in U$ it holds that

$$v_u = f_u(v_{u0}, v_{u1}). \tag{1}$$

The straightforward recursive algorithm for $\texttt{TrEv}_{h,\ell}$ (spelled out at the very beginning of Section 2) has space complexity $O(h\cdot\ell)$. In contrast, the procedure suggested by Cook and Mertz yields a dramatic improvement: Specifically, [5, Thm. 10] asserts that $\texttt{TrEv}_{h,\ell}$ has space complexity $O((\ell+h)\cdot\log\ell)$, whereas [5, Thm. 15] asserts space complexity $O(\ell + h\cdot\log\ell)$.

We provide an exposition of the foregoing results. In particular, we point out that the algebraic manipulation (relying on roots of unity) performed in [5] is merely a special case of univariate polynomial interpolation. This observation allows for a more transparent presentation of the proof of [5, Thm. 10] as well as of the low order quantitative improvement of [5, Thm. 15].

Another significant difference between our exposition and the one of [5] is that we work with the "global storage" model (used by us in [6, Sec. 5.2.4.2]), whereas [5] works with the "catalytic storage" model (introduced in [1]). We believe that the global storage model is more flexible and intuitive, but our exposition can be easily adapted to the catalytic storage model.

Organization. Section 2 presents the core of this survey, which is an alternative exposition of the proof of [5, Thm. 10]. The digest presented in Section 3 leads directly to an alternative proof of [5, Thm. 15]. The global storage model is reviewed in Section 4 and some tedious details that were avoided in Section 2 are provided in Section 5.

Follow-up work. Building on the current exposition, we were able to provide a small quantitative improvement over the results of Cook and Mertz [5]. This is presented in [7].

2 An Alternative Exposition of [5, Thm. 10]

We start by spelling out the straightforward recursive algorithm for $\texttt{TrEv}_{h,\ell}$.

The straightforward recursive algorithm. Observing that the value at node u is determined by the values at its two children, we compute v_u by first making a recursive call for the value of v_{u0} and then making a recursive call for the value of v_{u1}. Hence, before making the second recursive call, we maintain the (ℓ-bit long) value v_{u0} in the local memory of the current execution (which refers to node u). Once we obtain v_{u1}, we compute v_u and output it. The crucial point is that each level of recursion uses a local memory that is different from the memory that is used by other levels. Hence, the space complexity of the algorithm that unravels the recursion is $O(h \cdot \ell)$.

The improved Tree Evaluation algorithm (establishing [5, Thm. 10]) uses the same recursive strategy, but implements it in a highly sophisticated manner that avoids storing v_{u0} while computing v_{u1}. Specifically, rather than holding ℓ bits in the local memory of each recursive level, we shall store only $O(\log \ell)$ bits per level. This sounds miraculous, and this miracle is enabled by using a sophisticated accounting that relies on a non-standard model.

Towards the improved (recursive) algorithm. The first step towards improving the space complexity of $\texttt{TrEv}_{h,\ell}$ is conceptual: It consists of abandoning the paradigm of "good programming" under which a recursive call uses a different work space than the execution that calls it. Instead, we shall use the same *global space* for both executions, whereas only a much smaller work space will be allocated to each recursive level as its *local space*. The resulting *global storage model* and its relation to the standard model are spelled-out in Section 4. (We mention that this model was used by us in [6, Sec. 5.2.4.2], amd that the "catalytic space model" used by [5] is a special case.)

The key question is how to implement the foregoing recursion better (in the global storage model). For starters, suppose that the global memory holds three ℓ-bit strings, denoted x, y and z. Further suppose that we have a procedure that, for any $u \in U$ and $\sigma \in \{0,1\}$, when invoked with $(u\sigma, x, y, z)$ on its global storage, returns $(u\sigma, x, y, z \oplus v_{u\sigma})$ on the global storage, where $v_{u\sigma}$ is recursively defined as in Eq. (1). Then, when invoked with (u, x, y, z) on its global storage, we can return $(u, x, y, z \oplus v_u)$ (such that $v_u = f_u(v_{u0}, v_{u1})$) by proceeding as follows:

1. Making a recursive call with $(u0, y, z, x)$ on the global storage, we update the global storage to $(u0, y, z, x')$, where $x' \overset{\text{def}}{=} x \oplus v_{u0}$.

   (Note that we re-arranged the parts of the global storage so that the variable holding x is updated (to a value denoted x') and the other variables are left intact.)

2. Similarly, making a recursive call on $(u1, x', z, y)$, we update the global storage to $(u1, x', z, y')$, where $y' \overset{\text{def}}{=} y \oplus v_{u1}$.

3. *Miraculously* compute $z' \overset{\text{def}}{=} z \oplus f_u(v_{u0}, v_{u1})$ based on $x' = x \oplus v_{u0}$ and $y' = y \oplus v_{u1}$, while preserving the values of x' and y'.

4. Making a recursive call on $(u0, y', z', x')$, we update the global storage to $(u0, y', z', x)$.

   (Note that $x' \oplus v_{u0}$ equals the original value of x.)

5. Making a recursive call on $(u1, x, z', y')$, we update the global storage to $(u1, x, z', y)$.

6. Return (u, x, y, z').

Indeed, the problem is with the *miraculous step* (i.e., Step 3): We wish to compute $z \oplus f_u(v_{u0}, v_{u1})$, but we do not have v_{u0} and v_{u1}, but rather versions of these values that are masked by the original values of x and y, respectively. There is hope for such a miracle only if we have a few versions of this masking. Suppose, for example, that f_u were a linear (over GF(2)) function and that we have the values of $f_u(x',y')$ and $f_u(x,y)$; then, using $f_u(x',y') \oplus f_u(x,y) = f_u(x' \oplus x, y' \oplus y)$, we can obtain $f_u(x' \oplus x, y' \oplus y) = f_u(v_{u0}, v_{u1})$. This ignores the problem of having to store both $f_u(x',y')$ and $f_u(x,y)$. The last problem can be overcome if we deal with the bits of these ℓ-bit values one at a time; that is, for each $i \in [\ell]$, we first compute the i^{th} of $f_u(x',y')$ and then compute the i^{th} of $f_u(x,y)$, thus obtaining the i^{th} bit of $f_u(v_{u0}, v_{u1})$.

Hence, if all f_u's were linear functions, then the foregoing miracle (in Step 3) would have become a reality. Needless to say, we do not want to assume that the f_u's are linear. The alternative of using multi-linear extensions (of functions describing the output bits) arises naturally. Specifically, we shall compute the value of a corresponding tree in which the input functions (i.e., the f_u's) are replaced by their multi-linear extensions.

Multi-linear extensions and interpolation Indeed, we considering multi-linear extensions of the corresponding functions, where these extensions are in a (prime) field $\mathcal{K}$ that contains at least $2\ell + 2$ elements. Specifically, for every $u \in U$ and $i \in [\ell]$, let $f_{u,i}(x,y)$ equal the i^{th} bit of $f_u(x,y)$. Next, we define $\widehat{f}_{u,i} : \mathcal{K}^\ell \times \mathcal{K}^\ell \to \mathcal{K}$ to be the multi-linear extension of $f_{u,i} : \{0,1\}^\ell \times \{0,1\}^\ell \to \{0,1\}$. Now, suppose that we are given the values of $\widehat{f}_{u,i}(j\widehat{x} + v_0, j\widehat{y} + v_1)$ for every $j \in \{1, ..., 2\ell + 1\} \subset \mathcal{K}$, where $j \cdot (z_1, ..., z_\ell) = (jz_1, ..., jz_\ell)$. Using polynomial interpolation (on the degree 2ℓ univariate polynomial in j that arises from fixing $u, i, \widehat{x}, \widehat{y}, v_0$ and v_1), we obtain the value $\widehat{f}_{u,i}(0\widehat{x}+v_0, 0\widehat{y}+v_1) = f_{u,i}(v_0, v_1)$.

Note, however, that a naive implementation of this interpolation involves operating on these $2\ell + 1$ values (after storing them in memory). Fortunately, *the interpolation formula is a linear combination of these $2\ell + 1$ values, and so we need not store these values but can rather operate on them on-the-fly* (while only storing the partial linear combination computed so far). Specifically, the interpolation formula has the form

$$\widehat{f}_{u,i}(0\widehat{x} + v_0, 0\widehat{y} + v_1) \;=\; \sum_{j \in [2\ell+1]} c_j \cdot \widehat{f}_{u,i}(j\widehat{x} + v_0, j\widehat{y} + v_1) \tag{2}$$

where the c_j's are fixed constants in $\mathcal{K}$ (which depend on $[2\ell + 1] \subset \mathcal{K}$ and $\mathcal{K}$ only). Hence the l.h.s of Eq. (2) can be computed in $2\ell + 1$ iterations such that we enter the j^{th} iteration with the partial sum of the first $j - 1$ terms (i.e., $\sum_{j' \in [j-1]} c_{j'} \cdot \widehat{f}_{u,i}(j'\widehat{x} + v_0, j'\widehat{y} + v_1) \in \mathcal{K}$), compute the j^{th} term (i.e., $c_j \cdot \widehat{f}_{u,i}(j\widehat{x} + v_0, j\widehat{y} + v_1)$), and update the partial sum accordingly. Note that the fact that the interpolation points equal $[2\ell + 1]$ is immaterial; any $2\ell + 1$ interpolation points would do (but, of course, the coefficients c_j will chance accordingly).

Actually, as observed in [5], using specific interpolation points allows for a more explicit interpolation that merely sums-up the values (rather than using a more general linear combination). Specifically, these interpolation points are powers of an m^{th} root of unity (in $\mathcal{K}$), where $m > \ell' \stackrel{\text{def}}{=} 2\ell$ and $m < |\mathcal{K}| = O(\ell)$. Denoting such a root by ω, we observe that for any multi-linear polynomial $p : \mathcal{K}^{\ell'} \to \mathcal{K}$ it holds that

$$\sum_{j \in [m]} p(\omega^j r_1 + s_1, ..., \omega^j r_{\ell'} + s_{\ell'}) = m \cdot p(s_1, ..., s_{\ell'}). \tag{3}$$

(Eq. (3) can be proved by considering each monomial separately.)[1]

The improved (recursive) algorithm. For sake of simplicity, we first assume that we have oracle access to $F : U \times [\ell] \times \mathcal{K}^{2\ell} \to \mathcal{K}$ defined by

$$F(u, i, \widehat{x}, \widehat{y}) \stackrel{\rm def}{=} \widehat{f}_{u,i}(\widehat{x}, \widehat{y}). \tag{4}$$

The global memory that we use will hold three ℓ-long sequences over $\mathcal{K}$, denoted $\widehat{x}$, $\widehat{y}$ and $\widehat{z}$, as well as a string of length at most h, denoted u. Now, suppose that we have a procedure that, for any $u \in U$ and $\sigma, \tau \in \{0,1\}$, when invoked with $(u\sigma, \tau, \widehat{x}, \widehat{y}, \widehat{z})$ on its global memory, returns $(u\sigma, \widehat{x}, \widehat{y}, \widehat{z} + (-1)^\tau \cdot v_{u\sigma})$ on the global memory, where $v_{u\sigma} \in \{0,1\}^\ell \subset \mathcal{K}^\ell$ is recursively defined as in Eq. (1).[2] The procedure that we detail next will achieve an analogous effect on $(u, \tau, \widehat{x}, \widehat{y}, \widehat{z})$, where the point is that this procedure uses the same global memory as the procedure that it calls (while using only a small abount of local memory). In fact, we describe a recursive procedure that, on input of the form $(u, \cdot, \cdot, \cdot, \cdot)$, makes calls regarding inputs of the form $(u0, \cdot, \cdot, \cdot, \cdot)$ and $(u1, \cdot, \cdot, \cdot, \cdot)$.

Algorithm 1 (the recursive procedure): *Let the v_u's be recursively defined as in* Eq. (1). *Then, on input $(u, \tau, \widehat{x}, \widehat{y}, \widehat{z}) \in U \times \{0,1\} \times \mathcal{K}^{3\ell}$, placed on its global memory, the procedure returns $(u, \widehat{x}, \widehat{y}, \widehat{z} + (-1)^\tau v_u)$, on its global memory, where $v_u = f_u(v_{u0}, v_{u1})$. The recursive procedure does so by proceeding in m iterations.*[3]

(In iteration $j \in [m]$, for each $i \in [\ell]$, we increment the current value of the ith element of $\widehat{z}$ by $(-1)^\tau \cdot \widehat{f}_{u,i}(\omega^j \widehat{x} + v_{u0}, \omega^j \widehat{y} + v_{u1})/m$, while maintaining $(u, \widehat{x}, \widehat{y})$ intact. Recall that, by Eq. (3), $\sum_{j \in [m]} \widehat{f}_{u,i}(\omega^j \widehat{x} + v_{u0}, \omega^j \widehat{y} + v_{u1})/m$ equals $\widehat{f}_{u,i}(v_{u0}, v_{u1})$.)

The jth iteration proceeds as follows.

1. *Making a recursive call with $(u0, 0, \widehat{y}, \widehat{z}, \omega^j \widehat{x})$ on the global memory, we update the global memory to $(u0, \widehat{y}, \widehat{z}, \widehat{x}')$, where $\widehat{x}' \stackrel{\rm def}{=} \omega^j \widehat{x} + v_{u0}$.*
2. *Making a recursive call on $(u1, 0, \widehat{x}', \widehat{z}, \omega^j \widehat{y})$, we update the global memory to $(u1, \widehat{x}', \widehat{z}, \widehat{y}')$, where $\widehat{y}' \stackrel{\rm def}{=} \omega^j \widehat{y} + v_{u1}$.*

[1]For any $I \subseteq [\ell']$, it holds that

$$\begin{aligned}
\sum_{j\in[m]} \prod_{i\in I} (\omega^j r_i + s_i) &= \sum_{j\in[m]} \sum_{S\subseteq I} \left(\prod_{i\in S} \omega^j r_i\right) \cdot \left(\prod_{i\in I\setminus S} s_i\right) \\
&= \sum_{S\subseteq I} \sum_{j\in[m]} \omega^{j\cdot|S|} \left(\prod_{i\in S} r_i\right) \cdot \left(\prod_{i\in I\setminus S} s_i\right) \\
&= \sum_{S\subseteq I} \left(\sum_{j\in[m]} \omega^{j\cdot|S|}\right) \cdot \left(\prod_{i\in S} r_i\right) \cdot \left(\prod_{i\in I\setminus S} s_i\right) \\
&= m \cdot \prod_{i\in I} s_i,
\end{aligned}$$

where the last equality uses $\sum_{j\in[m]} \omega^{js} = 0$ for $s \in [\ell'] \subseteq [m-1]$ and $\sum_{j\in[m]} \omega^0 = m$.

[2]The variable/parameter τ allows us to either add or subtract the value $v_{u\sigma}$. In our recursive calls, we shall need both options.

[3]The following description is for the case of $u \in U$. In case $u \in \{0,1\}^h$, we may just obtain v_u from the input oracle (e.g., augment F such that $F(u) = v_u$).

3. *For each $i \in [\ell]$, letting $\widehat{z}_i$ denote the* i^{th} *element of $\widehat{z} \in \mathcal{K}^\ell$, we compute $\widehat{z}_i + (-1)^\tau \cdot F(u, i, \widehat{x}', \widehat{y}')/m$ by making an oracle call to F, and update the value of $\widehat{z}_i$ accordingly. Note that in the* i^{th} *sub-step only the* i^{th} *element of the sequence $\widehat{z}$ is updated* (and that division by m compensates for the factor of m in Eq. (3)).

4. *Making a recursive call on $(u0, 1, \widehat{y}', \widehat{z}, \widehat{x}')$, we update the global memory to $(u0, \widehat{y}', \widehat{z}, \omega^j\widehat{x})$.* (Note that $\widehat{x}' - v_{u0} = \omega^j\widehat{x}$.)

5. *Making a recursive call on $(u1, 1, \omega^j\widehat{x}, \widehat{z}, \widehat{y}')$, we update the global memory to $(u1, \omega^j\widehat{x}, \widehat{z}, \omega^j\widehat{y})$.*

6. *Re-arrange the global memory to contain $(u, \widehat{x}, \widehat{y}, \widehat{z})$, while noting that each $\widehat{z}_i$ got incremented by $(-1)^\tau \cdot \widehat{f}_{u,i}(\omega^j\widehat{x} + v_{u0}, \omega^j\widehat{y} + v_{u1})/m$.*

Using Eq. (3)*, we note that* (after the m iterations) *the value of each $\widehat{z}_i$ equals the initial value plus $(-1)^\tau \cdot \widehat{f}_{u,i}(v_{u0}, v_{u1}) = (-1)^\tau \cdot f_{u,i}(v_{u0}, v_{u1})$*

The correctness of Algorithm 1 follows from Eq. (3), and when invoked on input $(\lambda, 0, 0^\ell, 0^\ell, 0^\ell)$ it returns $(\lambda, 0^\ell, 0^\ell, v_\lambda)$. Algorithm 1 uses a global memory of length $h + O(1) + (3 + o(1)) \cdot \log_2 |\mathcal{K}|^\ell = O(h + \ell \cdot \log \ell)$, where the $o(1) \cdot \log_2 |\mathcal{K}|^\ell$ term accounts for the space complexity of various manipulations (including maintaining the counter $i \in [\ell]$), and a local memory of length $\log_2 m = O(\log \ell)$, which is used only for recording $j \in [m]$.

Using a composition lemma akin [6, Lem. 5.10] (reproduced as Lemma 5), it follows that the Tree Evaluation problem (with parameters h and ℓ) can be solved in space $O(h + \ell \log |\mathcal{K}|) + h \cdot O(\log \ell) = O((h + \ell) \cdot \log \ell)$, when using oracle access to F. Observing that F can be evaluated in linear space (i.e., space linear in $h + O(\ell \log |\mathcal{K}|)$)[4] and using a naive composition (see Section 5 for details), it follows that

Theorem 2 (Cook and Mertz [5, Thm. 10]): *The space complexity of* $\texttt{TrEv}_{h,\ell}$ *is* $O((h + \ell) \cdot \log \ell)$.

Recalling that the length of the input to $\texttt{TrEv}_{h,\ell}$ is exponential in $h + \ell$, it follows that $\texttt{TrEv}_{h,\ell}$ is solved in space $O(\log n \cdot \log \log n)$, where $n = \exp(\Theta(h + \ell))$.

3 Digest and Beyond

As stated in the introduction, we believe that the model of global storage (as outlined in [6, Def. 5.8] and reproduced in Section 4) is more flexible and intuitive than the model of catalytic storage used in [5], which may be viewed as a special case.[5] Hence, we used the global storage model rather than the catalytic storage model in our exposition.

As stated in Section 2, the interpolation formula given in Eq. (3), which relies on an m^{th} root of unity, is inessential for the proof of Theorem 2. More generally, recalling that the $\widehat{f}_{u,i}$'s are

[4]Recall that computing F calls for computing the corresponding $\widehat{f}_{u,i}$, which is a multi-linear extension of $f_{u,i}$. As for computing $\widehat{f}_{u,i}$, it requires obtaining all values of $f_{u,i}$ (cf. Footnote 7).

[5]In particular, the catalytic model presumes that the memory is structured (e.g., partitioned into registers that hold values of some semigroup) and supports specific operations (i.e., "clean computation"). To see that the catalytic model is a special case of the global model, we note a correspondence between the catalytic (resp., ordinary) storage of the catalytic model and the global (resp., local) memory of the model defined in Section 4. Specifically, a "clean computation" (in the catalytic model) that results in adding a desired value to one set of registers while keeping the other registers intact can be emulated by a corresponding transformation of the global storage.

polynomials of total degree 2ℓ, we can use univariate polynomial interpolation based on any $2\ell' + 1$ points (on a line that passes through the desired point), while noting that such interpolation can be represented by a linear combination of the polynomial's values (with coefficients that depend on the interpolation points and the desired point). This is indeed captuted by Eq. (2). The only advantage of using Eq. (3) is that the interpolation formula is a simple sum (i.e., all coefficients are 1).

Capitalizing on the last paragraph, we can reduce the length of global storage used by the recursive procedure from $O(h+\ell \log \ell)$ to $O(h+\ell)$. This can be done by viewing the f_u's as functions from $[k]^k \times [k]^k$ to $[k]^k$, where $k^k = 2^\ell$ (i.e., $k = \Theta(\ell/\log \ell)$), and using low degree extensions of the corresponding $f_{u,i}$'s (i.e., $f_{u,i}(x, y) \in [k]$ is the i^{th} symbol in $f_u(x, y) \in [k]^k$).[6] Specifically, these extensions are $2k$-variate polynomials of individual degree $k - 1$ over $\mathcal{K}$, where $\mathcal{K}$ is a finite field of size poly(k) that is greater than $m = 2k^2$ (and $[k] \subset \mathcal{K}$).[7] Thus, $\widehat{f}_{u,i} : \mathcal{K}^k \times \mathcal{K}^k \to \mathcal{K}$ has total degree $2k \cdot (k - 1) < m$, whereas its input length (i.e., $\log_2 |\mathcal{K}^{2k}|$) equals $\log_2(\text{poly}(k)^{2k}) = O(\ell)$. Consequently, we can obtain the value of $\widehat{f}_{u,i}(v_0, v_1)$ by univariate polynomial interpolation from the values of $\widehat{f}_{u,i}(j\widehat{x} + v_0, j\widehat{y} + v_1)$ for all $j \in [m]$. Hence, the revised recursive procedure uses a global space of length $O(h + \ell)$ and local space of length $\log_2 m = O(\log \ell)$.

Again, using a composition lemma akin [6, Lem. 5.10], it follows that $\texttt{TrEv}_{h,\ell}$ can be solved in space $O(\ell + h \cdot \log \ell)$, when using oracle access to $F : U \times [\ell] \times \mathcal{K}^{2k} \to \mathcal{K}$, which in turn can be evaluated in linear space (i.e., space linear in $O(h + \ell)$).[8] Hence (see Section 5 for details), we obtain

Theorem 3 (Cook and Mertz [5, Thm. 15]): *The space complexity of* $\texttt{TrEv}_{h,\ell}$ *is* $O(\ell + h \cdot \log \ell)$.

In particular, for $h = O(\ell/\log \ell)$, the problem can be solved in logarithmic space (because, in this case, the input length is $n = \exp(\Theta(\ell))$, whereas $O(\ell + h \cdot \log \ell) = O(\ell) = O(\log n)$).

Needless to say, the question of whether $\texttt{TrEv}_{h,\ell}$ can be solved in $O(\ell + h)$ space remains open. The stumbling block for our approach is that we use $O(\log \ell)$ bits of local storage for indexing poly(ℓ) different evaluations of $\widehat{f}_{u,i}$.

4 The Global Storage Model (Following [6, Sec. 5.2.4.2])

(The global storage model was introduced in [6, Sec. 5.2.4] in order to facilitate a modular presentation of Reingold's UCONN algorithm [8].)

The aim of this model is to support a composition result that is beneficial in the context of recursive calls. The basic idea is deviating from the paradigm that allocates *separate* input/output and query devices to *each level in the recursion*, and combining all these devices in a single ("global")

[6]More generally, we may replace $\{0, 1\}^\ell$ by S^k such that $k = \Theta(\ell/\log \ell)$ and $S \subset \mathcal{K}$ has size $2^{\ell/k}$.

[7]Indeed, for simplicity, we assume that $\mathcal{K}$ is of prime cardinality. In general, for $S \subset \mathcal{K}$, the low degree extension of $f : S^t \to S$ is given by $\widehat{f} : \mathcal{K}^t \to \mathcal{K}$ such that

$$\widehat{f}(x_1, ..., x_t) = \sum_{a_1,...,a_t \in S} \left(\prod_{i \in [t]} \chi_{a_i}(x_i) \right) \cdot f(a_1, ..., a_t),$$

where $\chi_a(x) \overset{\text{def}}{=} \prod_{b \in S \setminus \{a\}} (x - b)/(a - b)$ is a degree $|S| - 1$ univariate polynomial.

[8]We can also assign to F the task of providing the coefficients of the linear combination used in the interpolation. As detailed in Section 5, these coefficients can be computed in $o(\ell)$ space.

device, which will be used by all levels of the recursion. That is, rather than following the "structured programming" methodology of using locally designated space for passing information to the subroutine, we use the "bad programming" methodology of passing information through global variables. (As usual, this notion is formulated by referring to the model of multi-tape Turing machine, but it can be formulated in any other reasonable model of computation.)

Definition 4 (following [6, Def. 5.8]): *A* global-tape oracle machine *is defined as an oracle machine* (cf. [6, Def. 1.11])*, except that the input, output and oracle tapes are replaced by a single* global-tape*. In addition, the machine has a constant number of work tapes, called the* local-tapes*. The machine obtains its input from the global-tape, writes each query on this very tape, obtains the corresponding answer from this tape, and writes its final output on this tape.* (We stress that, as a result of invoking the oracle f, the contents of the global-tape changes from q to $f(q)$.)[9] *In addition, the machine can use the global-tape also for its internal computations. The space complexity of such a machine is stated when referring separately to its use of the global-tape and to its use of the local-tapes.*

Note that in our presentation of Theorem 2 we also used oracle calls to a function F. This was done for the sake of simplicity, and these oracle calls (unlike the recursive calls) can be modeled by the usual mechanism (of oracle tapes).

Composing global-tape oracle machines. As stated above, global-tape oracle machines are beneficial in the context of *recursive composition*, as indicated by Lemma 5 (which relies on this model in a crucial way). The key observation is that all levels in the recursive composition may re-use the same global storage, and only the local storage gets added. Consequently, we have the following composition lemma.

Lemma 5 (recursive composition in the global-tape model [6, Lem. 5.10]): *Suppose that there exists a* global-tape oracle machine *that, for every $i = 1, ..., t-1$, computes f_{i+1} by making oracle calls to f_i while using a global-tape of length L and a local-tape of length l_i, which also accounts for the machine's state. Then, f_t can be computed by a* standard oracle machine *that makes calls to f_1 and uses space $L + \sum_{i=1}^{t-1}(l_i + \log_2 l_i)$.*

Proof Sketch: We compute f_t by allocating space for the emulation of the global-tape and the local-tapes of each level in the recursion. We emulate the recursive computation by capitalizing on the fact that all recursive levels use the same global-tape (for making queries and receiving answers). Recall that in the actual recursion, each level may use the global-tape arbitrarily as long as when it returns control to the invoking machine the global-tape contains the correct answer. Thus, the emulation may do the same, and emulate each recursive call by using the space allocated for the global-tape as well as the space designated for the local-tape of this level. The emulation should also store the locations of the other levels of the recursion on the corresponding local-tapes, which is accounted for by the extra $\sum_{i=1}^{t-1} \log_2 l_i$ term. ■

[9]This means that the prior contents of the global-tape (i.e., the query q) is lost (i.e., it is replaced by the answer $f(q)$). Thus, if we wish to keep such prior contents, then we need to copy it to a local-tape. We also stress that, according to the standard oracle invocation conventions, the head location after the oracle responds is at the left-most cell of the global-tape.

5 Tedious Details

Recall that Theorems 2 and 3 were proved by using a composition lemma akin [6, Lem. 5.10], which implies a space bound on a procedure (in the standard model) that computes $\texttt{TrEv}_{h,\ell}$ when given oracle access to F. As for F, it was assigned the task for computing the $\widehat{f}_{u,i}$'s, providing the values of the v_u's for all $u \in \{0,1\}^\ell$, and computing the coefficients of the linear combination that underlies the interpolation procedure.[10] We stress that, while the latter v_u's and all $f_{u,i}$'s appear explicitly in the input to $\texttt{TrEv}_{h,\ell}$, the $\widehat{f}_{u,i}$'s and the interpolation coefficients need to be computed. We address both tasks in the more general setting of the proof of Theorem 3.

Computing the $\widehat{f}_{u,i}$'s. As stated in Footnote 7, for a prime field $\mathcal{K}$, the low degree extension of $f_{u,i} : [k]^k \to [k]$ is given by $\widehat{f}_{u,i} : \mathcal{K}^k \to \mathcal{K}$ such that

$$\widehat{f}_{u,i}(x_1, ..., x_k) = \sum_{a_1,...,a_k \in [k]} \left(\prod_{s \in [k]} \chi_{a_s}(x_s) \right) \cdot f_{u,i}(a_1, ..., a_k), \tag{5}$$

where $\chi_a(x) \stackrel{\text{def}}{=} \prod_{b \in [k] \setminus \{a\}} (x-b)/(a-b)$ is a degree $k-1$ univariate polynomial over $\mathcal{K}$. Hence, $\widehat{f}_{u,i}$ can be computed by going over all possible $(a_1, ..., a_k) \in [k]^k$ (and all $s, b \in [k]$), which can be done using $\log_2(k^k \cdot k^2) = (1+o(1)) \cdot \ell$ space.

Computing the interpolation coefficients. The desired coefficients for the interpolation (of a univariate polynomial (based on m evaluation points)) are the first row of the inverse of an m-by-m Vandermonde matrix over $\mathcal{K}$. While the Vandermonde matrix has a simple explicit form (i.e., its entries are powers of the evaluation points), its inverse has a simple explicit form only in some cases (i.e., for some structured sequence of evaluation points). We can use such a structured sequence, but prefer not to rely on such low level considerations. Instead, we use the fact that matrix inversion is in NC, and hence can be computed in polylogarithmic space, whereas here we the input length is $m^2 \cdot \log_2 |\mathcal{K}| = \text{poly}(\ell)$. Hence, the inverse of the relevant matrix can be computed in space $\text{poly}(\log(\ell)) = o(\ell)$.

Conclusion. The function F can be evaluated in space that is linear in the length of the main part of its input (i.e., linear in $|\widehat{x}| + |\widehat{y}|$, which is $2\ell \cdot \log_2 |\mathcal{K}| = O(\ell \log \ell)$ in the proof of Theorem 2 and $2k \cdot \log_2 |\mathcal{K}| = O(\ell)$ in the proof of Theorem 3). Composing the (standard model) procedure that computes $\texttt{TrEv}_{h,\ell}$ (when given oracle access to F) with the algorithm for computing F, we derive the claimed results.

Acknowledgments

I am grateful to Amnon Ta-Shma, Ben Chen, and Madhu Sudan for helpful discussions and comments.

[10] Computing the coefficients is needed either when proving Theorem 3 or when using Eq. (2) (rather than Eq. (3)).

References

[1] Harry Buhrman, Richard Cleve, Michal Koucky, Bruno Loff, and Florian Speelman. Computing with a full memory: catalytic space. In *46th ACM Symposium on the Theory of Computing*, pages 857–866, 2014.

[2] Stephen Cook, Pierre McKenzie, Dustin Wehr, Mark Braverman, and Rahul Santhanam. Pebbles and branching programs for tree evaluation. *ACM Trans. Comput. Theory*, Vol. 3 (2), Art. 4:1–4:43, 2012.

[3] James Cook and Ian Mertz. Catalytic approaches to the tree evaluation problem. In *52nd STOC*, pages 752–760, 2020.

[4] James Cook and Ian Mertz. Encodings and the tree evaluation problem. *ECCC*, TR21-054, 2021.

[5] James Cook and Ian Mertz. Tree Evaluation is in Space $O(\log n \cdot \log\log n)$. *ECCC*, TR23-174, 2023.

[6] Oded Goldreich. *Computational Complexity: A Conceptual Perspective*. Cambridge University Press, 2008.

[7] Oded Goldreich. Solving Tree Evaluation in $o(\log n \cdot \log\log n)$ Space. *ECCC*, TR24-124, 2024.

[8] Omer Reingold. Undirected ST-Connectivity in Log-Space. In *37th ACM Symposium on the Theory of Computing*, pages 376–385, 2005.