Branching Programs for Tree Evaluation

Mark Braverman[1], Stephen Cook[2], Pierre McKenzie[3], Rahul Santhanam[4], and Dustin Wehr[2]

[1] Microsoft Research
[2] University of Toronto
[3] Université de Montréal
[4] University of Edinburgh

Abstract. The problem $FT^h_d(k)$ consists in computing the value in $[k] = \{1, \ldots, k\}$ taken by the root of a balanced d-ary tree of height h whose internal nodes are labelled with d-ary functions on $[k]$ and whose leaves are labelled with elements of $[k]$. We propose $FT^h_d(k)$ as a good candidate for witnessing $\mathbf{L} \subsetneq \mathbf{LogDCFL}$. We observe that the latter would follow from a proof that k-way branching programs solving $FT^h_d(k)$ require $\Omega(k^{\text{unbounded function}(h)})$ size. We introduce a "state sequence" method that can match the size lower bounds on $FT^h_d(k)$ obtained by the Nečiporuk method and can yield slightly better (yet still subquadratic) bounds for some nonboolean functions. Both methods yield the tight bounds $\Theta(k^3)$ and $\Theta(k^{5/2})$ for deterministic and nondeterministic branching programs solving $FT^3_2(k)$ respectively. We propose as a challenge to break the quadratic barrier inherent in the Nečiporuk method by adapting the state sequence method to handle $FT^4_d(k)$.

1 Introduction

Let T^h_d be the balanced d-ary ordered tree T^h_d of height h, where we take *height* to mean the number of levels in the tree and we number the nodes as suggested by the heap data structure. Thus the root is node 1, and in general the children of node i are (when $d = 2$) nodes $2i, 2i + 1$ (see Figure 1). For every $d, h, k \geq 2$ we define the *Tree Evaluation* problem and its associated decision problem:

Definition 1.1 ($FT^h_d(k)$ and $BT^h_d(k)$)
Given: T^h_d with each non-leaf node i independently labelled with a function $f_i : [k]^d \to [k]$ and each leaf node independently labelled with an element from $[k]$.
Function evaluation problem $FT^h_d(k)$*: Compute the value $v_1 \in [k]$ of the root 1 of T^h_d, where in general $v_i = a$ if i is a leaf labelled a and $v_i = f_i(v_{j_1}, \ldots, v_{j_d})$ and the children of i are $j_1, \ldots, j_d$.*
Boolean evaluation problem $BT^h_d(k)$*: Decide whether $v_1 = 1$.*

In the context of uniform complexity measures such as Turing machine space we rewrite $FT^h_d(k)$ and $BT^h_d(k)$ as $FT_d(h,k)$ and $BT_d(h,k)$ to indicate that d is fixed but h,k are input parameters. It is not hard to show that for each $d \geq 2$ a deterministic logspace-bounded poly-time auxiliary pushdown automaton solves $BT_d(h,k)$, implying by [**?**] that $BT_d(h,k)$ belongs to the class **LogDCFL** of languages logspace reducible to a deterministic context-free language. We know $\mathbf{L} \subseteq \mathbf{LogDCFL} \subseteq \mathbf{P}$ (see [**?**] for up to date information on **LogDCFL**). The special case $BT_d(h,2)$ was investigated under a different name in [**?**] as part of an attempt to separate $\mathbf{NC}^1$ from $\mathbf{NC}^2$. In this paper, we suggest investigating the space complexity of $BT_d(h,k)$ and $FT_d(h,k)$.

We choose to study the Tree Evaluation problem as a particularly interesting candidate for non-membership in **L** or **NL** (deterministic of nondeterministic log space) because pebble games on trees provide natural space bounded algorithms for solving it: Black pebbling provides deterministic algorithms and, though we do not consider these in this paper, black-white pebbling provides nondeterministic algorithms. We choose k-way branching programs (BPs) as our model of Turing machine because the inputs to our problems are tuples of numbers in $[k]$.

For fixed d,h we are interested in how the size (number of states) of BPs solving $FT^h_d(k)$ and $BT^h_d(k)$ grows with k. One of our contributions is an alternative approach to Nečiporuk's lower bound method [**?**] for this size. Applied to the problem $BT^h_d(k)$, our "state sequence" approach does as well as (but, so far, no better than) Nečiporuk's method. On the other hand, our approach does not suffer in principle from the quadratic limitation inherent in Nečiporuk's method. Hence there is hope that the approach can be extended. The current bottleneck stands at height 4. Proving our conjectured lower bound of $\Omega(k^7/\lg k)$ (writing lg for $\log_2$) for the size of deterministic BPs solving $BT^4_3(k)$ would constitute a breakthrough and would overcome the n^2 Nečiporuk limitation. However we do not yet know how to do this.

The more specific contributions of this paper are the following:

- we observe that for any $d \geq 2$ and unbounded $r(h)$, a lower bound of the form $\Omega(k^{r(h)})$ on the size of BPs solving $FT^h_d(k)$ would prove $BT_d(h,k) \notin \mathbf{L}$;
- we prove tight black pebbling bounds for T^h_d and transfer the upper bounds to size upper bounds of the form $k^{\Omega(h)}$ for deterministic k-way BPs for $FT^h_d(k)$ and $BT^h_d(k)$;
- we prove tight size bounds of $\Theta(k^{2d-1})$ and $\Theta(k^{2d-1}/\lg k)$ for deterministic k-way BPs solving $FT^3_d(k)$ and $BT^3_d(k)$ respectively;
- we prove tight size bounds of $\Theta(k^{3d/2-1/2})$ for nondeterministic k-way BPs solving $BT^3_d(k)$; in terms of input length, this implies an $\Omega(n^{3/2}/(\lg n)^{3/2})$ bound for the number of *states* in nondeterministic binary BPs of arbitrary outdegree, which improves slightly on the former $\Omega(n^{3/2})$ bound obtained for the number of *edges* [**?**,**?**] in such BPs;
- we give examples of functions, such as the restriction $SumMod^3_2(k)$ of $FT^3_2(k)$ in which the root function is fixed to the sum modulo k, and the function $Children^4_2(k)$ which is required to simultaneously compute the root values of two instances of $FT^3_2(k)$, for which the state sequence method yields a

better k-way BP size lower bound than a direct application of Nečiporuk's method ($\Omega(k^3)$ versus $\Omega(k^2)$ for $SumMod_2^3(k)$, and $\Omega(k^4)$ versus $\Omega(k^3)$ for $Children_2^4(k)$).

Section 2 defines branching programs and pebbling. Section 3 relates pebbling and branching programs to Turing machine space, and proves the pebbling bounds exploited in Section 4 to prove BP size upper bounds. BP lower bounds obtained using the Nečiporuk method are stated in Subsection 4.1. Our state sequence method is introduced in Subsection 4.2. An Appendix contains most of the proofs left out of the main text.

2 Preliminaries

We assume some familiarity with complexity theory, such as can be found in [**?**]. We write $[k]$ for $\{1, 2, \ldots, k\}$ and let $k \geq 2$.
Warning: Recall that the *height* of a tree is the number of levels in the tree, as opposed to the distance from root to leaf. Thus T_2^2 has just 3 nodes.

2.1 Branching programs

Many variants of the branching program model have been studied [**?**,**?**]. Our definition below is inspired by Wegener [**?**, p. 239], by the k-way branching program of Borodin and Cook [**?**] and by its nondeterministic variant [**?**,**?**]. We depart from the latter however in two ways: nondeterministic branching program labels are attached to states rather than edges (because we think of branching program states as Turing machine configurations) and cycles in branching programs are allowed (because our lower bounds apply to this more powerful model).

Definition 2.1 (Branching programs) *A* nondeterministic k-way branching program *B computing a total function $g : [k]^m \to R$, where R is a finite set, is a directed rooted multi-graph whose nodes are called* states. *Every edge has a label from $[k]$. Every state has a label from $[m]$, except $|R|$* final *sink states consecutively labelled with the elements from R. An input $(x_1, \ldots, x_m) \in [k]^m$ activates, for each $1 \leq j \leq m$, every edge labelled x_j out of every state labelled j. A* computation *on input $\vec{x} = (x_1, \ldots, x_m) \in [k]^m$ is a directed path consisting of edges activated by $\vec{x}$ which begins with the unique start state (the root), and either it is infinite, or it ends in the final state labelled $g(x_1, \ldots, x_m)$, or it ends in a nonfinal state labelled j with no out edge labelled x_j (in which case we say the computation* aborts*). At least one such computation must end in a final state. The* size *of B is its number of states. B is* deterministic k-way *if every non-sink state has precisely k outedges labelled $1, \ldots, k$. B is* binary *if $k = 2$.*

We say that B solves a decision problem (relation) if it computes the characteristic function of the relation.

A k-way branching program computing the function $FT_d^h(k)$ requires k^d k-ary arguments for each internal node i of T_d^h in order to specify the function f_i, together with one k-ary argument for each leaf. Thus in the notation of Definition 1.1 $FT_d^h(k)$: $[k]^m \to R$ where $R = [k]$ and $m = \frac{d^{h-1}-1}{d-1} \cdot k^d + d^{h-1}$. Also $BT_d^h(k)$: $[k]^m \to \{0,1\}$.

We define $\#\mathsf{detFstates}_d^h(k)$ (resp. $\#\mathsf{ndetFstates}_d^h(k)$) to be the mininum number of states required for a deterministic (resp. nondeterministic) k-way branching program to solve $FT_d^h(k)$. Similarly we define $\#\mathsf{detBstates}_d^h(k)$ and $\#\mathsf{ndetBstates}_d^h(k)$ to be the number of states for solving $BT_d^h(k)$.

The next lemma is easy to prove and shows that the function problem is not much harder to solve than the Boolean problem.

Lemma 2.2 $\#\mathsf{detBstates}_d^h(k) \leq \#\mathsf{detFstates}_d^h(k) \leq k \cdot \#\mathsf{detBstates}_d^h(k)$ *and* $\#\mathsf{ndetBstates}_d^h(k) \leq \#\mathsf{ndetFstates}_d^h(k) \leq k \cdot \#\mathsf{ndetBstates}_d^h(k)$.

2.2 Pebbling

The pebbling game for dags was defined by Paterson and Hewitt [**?**] and was used as an abstraction for deterministic Turing machine space in [**?**]. Black-white pebbling was introduced in [**?**] as an abstraction of nondeterministic Turing machine space (see [**?**] for a survey).

We will only make use of a simple 'black pebbling' game in this paper. Here a pebble can be placed on any leaf node, and in general if all children of a node i have pebbles, then one of the pebbles on the children can be slid to i (this is a "sliding" move). The goal is to pebble the root. A *pebbling* of a tree T using p pebbles is any sequence of pebbling moves on nodes of T which starts and ends with no pebbles, and at some point the root is pebbled, and no configuration has more than p pebbles.

Our motivation for choosing this definition is that we want pebbling algorithms for trees to closely correspond to k-way branching program algorithms for the tree evaluation problem.

We use $\#\mathsf{pebbles}(T)$ to denote the minimum number of pebbles required to pebble T. The following is an adaptation of results and techniques that have been known since work of Loui, Meyer auf der Heide and Lengauer-Tarjan [**?**,**?**,**?**] in the late '70s. We allow sliding moves.

Theorem 2.3. *For every* $d, h \geq 2$, $\#\mathsf{pebbles}(T_d^h) = (d-1)h - d + 2$.

3 Connecting TMs, BPs, and Pebbling

Let $FT_d(h,k)$ be the same as $FT_d^h(k)$ except now the inputs vary with both h and k, and we assume the input to $FT_d(h,k)$ is a binary string X which codes h and k and codes each node function f_i for the tree T_d^h by a sequence of k^d binary numbers and each leaf value by a binary number in $[k]$, so X has length

$$|X| = \Theta(d^h k^d \lg k) \tag{1}$$

The output is a binary number in $[k]$ giving the value of the root. The problem $BT_d(h,k)$ is the Boolean version of $FT_d(h,k)$: The input is the same, and the instance is true iff the value of the root is 1.

Obviously $BT_d(h,k)$ and $FT_d(h,k)$ can be solved in polynomial time, but we can prove a stronger result.

Theorem 3.1. *For each $d \geq 2$ the problem $BT_d(h,k)$ is in* **LogDCFL**.

The best known upper bounds on the number of states required by a BP to solve $FT_d^h(k)$ grow as $k^{\Omega(h)}$. The next result shows (Corollary 3.3) that any provable nontrivial dependency on h, for the power of k expressing the minimum number of such states, would separate **L**, and perhaps **NL** (deterministic and nondeterministic log space), from **LogDCFL**.

Theorem 3.2. *For each $d \geq 2$, if $BT_d(h,k)$ is in* **L** *(resp.* **NL***) then there is a constant ω_d and a function $c_d(h)$ such that $\#\mathsf{detFstates}_d^h(k) \leq c_d(h)k^{\omega_d}$ (resp. $\#\mathsf{ndetFstates}_d^h(k) \leq c_d(h)k^{\omega_d}$) for all $h,k \geq 2$.*

Proof. By Lemma 2.2, arguing for $\#\mathsf{detBstates}_d^h(k)$ and $\#\mathsf{ndetBstates}_d^h(k)$ instead of $\#\mathsf{detFstates}_d^h(k)$ and $\#\mathsf{ndetFstates}_d^h(k)$ suffices. In general a Turing machine which can enter at most C different configurations on all inputs of a given length n can be simulated (for inputs of length n) by a binary (and hence k-ary) branching program with C states. Each Turing machine using space $O(\lg n)$ has at most n^c possible configurations on any input of length $n \geq 2$, for some constant c. By (1) the input for $BT_d(h,k)$ has length $n = \Theta(d^h k^d \lg k)$, so there are at most $(d^h k^d \lg k)^{c'}$ possible configurations for a log space Turing machine solving $BT_d(h,k)$, for some constant c'. So we can take $c_d(h) = d^{c'h}$ and $\omega_d = c'(d+1)$. □

Corollary 3.3 *Fix $d \geq 2$ and any unbounded function $r(h)$. If $\#\mathsf{detFstates}_d^h(k)$ (resp. $\#\mathsf{ndetFstates}_d^h(k)$) $\in \Omega(k^{r(h)})$ then $BT_d(h,k) \notin$* **L** *(resp. $\notin$* **NL***).*

The next result connects pebbling upper bounds with BP upper bounds.

Theorem 3.4. *If T_d^h can be pebbled with p pebbles, then deterministic branching programs with $O(k^p)$ states can solve $FT_d^h(k)$ and $BT_d^h(k)$.*

Corollary 3.5 $\#\mathsf{detFstates}_d^h(k) = O(k^{\#\mathsf{pebbles}(T_d^h)})$.

4 Branching Program Bounds

In this section we prove optimal bounds (up to a constant factor) for the number of states required for both deterministic and nondeterminisitc k-way branching progams to solve the Boolean problems $BT_d^3(k)$ for all trees of height 3. (The bound is obviously $\Theta(k^d)$ for trees of height 2, because there are $d + k^d$ input variables.) We also prove bounds for the function problem $FT_d^h(k)$.

For the deterministic case our nearly best bounds come from pebbling via Theorem 3.4, although we can improve on them for $BT_2^h(k)$ by a factor of $\lg k$.

Theorem 4.1 (BP Upper Bounds).

$$\#\mathsf{detBstates}_d^h(k) = O(k^{(d-1)h-d+2}) \tag{2}$$
$$\#\mathsf{detFstates}_d^h(k) = O(k^{(d-1)h-d+2}) \tag{3}$$
$$\#\mathsf{ndetBstates}_2^3(k) = O(k^{5/2}) \tag{4}$$
$$\#\mathsf{detBstates}_d^h(k) = O(k^{(d-1)(h-1)+1}/\lg k^{d-1}),\ \textit{for } h \geq 3. \tag{5}$$

We can combine the above upper bounds with the Nečiporuk lower bounds in Subsection 4.1, Figure 1, to obtain the following tight bounds.

Corollary 4.2 (Height 3 trees)

$$\#\mathsf{ndetBstates}_2^3(k) = \Theta(k^{5/2})$$
$$\#\mathsf{detBstates}_d^3(k) = \Theta(k^{2d-1}/\lg k)$$
$$\#\mathsf{detFstates}_d^3(k) = \Theta(k^{2d-1}).$$

4.1 The Nečiporuk method

The Nečiporuk method still yields the strongest explicit binary branching program size lower bounds known today, namely $\Omega(\frac{n^2}{(\lg n)^2})$ for deterministic [**?**] and $\Omega(\frac{n^{3/2}}{\lg n})$ for nondeterministic (albeit for a weaker nondeterministic model in which states have bounded outdegree [**?**], see [**?**]).

By *applying the Nečiporuk method* to a k-way branching program B computing a function $f : [k]^m \to R$, we mean the following well known steps [**?**]:

1. Upper bound the number $N(s, v)$ of (syntactically) distinct branching programs of type B having s non-sink states, each labelled by one of v variables.
2. Pick a partition $\{V_1, \ldots, V_p\}$ of $[m]$.
3. For $1 \leq i \leq p$, lower bound the number $r_{V_i}(f)$ of restrictions $f_{V_i} : [k]^{|V_i|} \to R$ of f obtainable by fixing values of the variables in $[m] \setminus V_i$.
4. Then $\text{size}(B) \geq |R| + \sum_{1 \leq i \leq p} s_i$, where $s_i = \min\{\ s : N(s, |V_i|) \geq r_{V_i}(f)\ \}$.

Theorem 4.3. *Applying the Nečiporuk method yields Figure 1.*

Remark 4.4. The $\Omega(n^{3/2}/(\lg n)^{3/2})$ binary nondeterministic BP lower bound for the $BT_d^h(k)$ problem and in particular for $BT_2^3(k)$ applies to the number of *states* when these can have arbitrary outdegree. This seems to improve on the best known former bound of $\Omega(n^{3/2}/\lg n)$, slightly larger but obtained for the weaker model in which states have bounded degree, or equivalently, for the switching and rectifier network model in which size is defined as the number of edges [**?**,**?**].

Let $Children_d^h(k)$ have the same input as $FT_d^h(k)$ with the exception that the root function is deleted. The output is the tuple $(v_2, v_3, \ldots, v_{d+1})$ of values for the children of the root.

Model	Lower bound for $FT_d^h(k)$	Lower bound for $BT_d^h(k)$
Deterministic k-way branching program	$\boxed{\frac{d^{h-2}-1}{4d(d-1)^2} \cdot k^{2d-1}}$	$\boxed{\frac{d^{h-2}-1}{3d(d-1)} \cdot \frac{k^{2d-1}}{\lg k}}$
Deterministic binary branching program	$\frac{d^{h-2}-1}{6d(d-1)} \cdot k^{2d} = \Omega(n^2/(\lg n)^2)$	$\frac{d^{h-2}-1}{4d(d-1)} \cdot \frac{k^{2d}}{\lg k} = \Omega(n^2/(\lg n)^3)$
Nondeterministic k-way BP	$\frac{d^{h-2}-1}{2d-2} \cdot k^{\frac{3d}{2}-\frac{1}{2}} \sqrt{\lg k}$	$\boxed{\frac{d^{h-2}-1}{2d-2} \cdot k^{\frac{3d}{2}-\frac{1}{2}}}$
Nondeterministic binary BP	$\frac{d^{h-2}-1}{2d-2} \cdot k^{\frac{3d}{2}} \lg k = \Omega(n^{3/2}/\lg n)$	$\frac{d^{h-2}-1}{2d-2} \cdot k^{\frac{3d}{2}} = \Omega(n^{3/2}/(\lg n)^{3/2})$

Fig. 1. Size bounds, expressed in terms of $n = \Theta(k^d \lg k)$ in the binary cases, obtained by applying the Nečiporuk method. Rectangles indicate optimality in k when $h = 3$ (Cor. 4.2). Improving any entry to $\Omega(k^{\text{unbounded } f(h)})$ would prove $\mathbf{L} \subsetneq \mathbf{P}$ (Cor. 3.3).

Theorem 4.5. *For any $d, h \geq 2$, the best k-way deterministic BP size lower bound attainable for $Children_d^h(k)$ by applying the Nečiporuk method is $\Omega(k^{2d-1})$.*

Let $SumMod_d^h(k)$ have the same input as $FT_d^h(k)$ with the exception that the root function is preset to the sum modulo k. In other words the output is $v_2 + v_3 + \cdots + v_{d+1} \mod k$.

Theorem 4.6. *The best k-way deterministic BP size lower bound attainable for $SumMod_2^3(k)$ by applying the Nečiporuk method is $\Omega(k^2)$.*

4.2 The state sequence method

Here we give alternative proofs for some of the lower bounds given in Section 4.1. These proofs are more intricate than the Nečiporuk proofs but they do not suffer a priori from a quadratic limitation. The method also yields stronger lower bounds to $Children_2^4(k)$ and $SumMod_2^3(k)$ than those obtained by applying the Nečiporuk's method as expressed in Subsection 4.1 (see Theorems 4.5 and 4.6).

Theorem 4.7. $\#\mathsf{ndetBstates}_2^3(k) \geq k^{2.5}$ *for sufficiently large k.*

Proof. Consider an input I to $BT_2^3(k)$. We number the nodes in T_2^3 as in Figure 1, and let v_j^I denote the value of node j under input I. We say that a state in a computation on input I *learns* v_j^I if that state queries $f_j^I(v_{2j}^I, v_{2j+1}^I)$ (recall $2j, 2j+1$ are the children of node j).

Definition [Learning Interval] *Let B be a k-way nondeterministic BP that solves $BT_2^3(k)$. Let $\mathcal{C} = \gamma_0, \gamma_1, \cdots, \gamma_T$ be a computation of B on input I. We say that a state γ_i in the computation is* critical *if one of the following holds:*

1. *$i = 0$ or $i = T$*
2. *γ_i learns v_2^I and there is an earlier state which learns v_3^I with no intervening state that learns v_2^I.*

3. *γ_i learns v_3^I and no earlier state learns v_3^I unless an intervening state learns v_2^I.*

We say that a subsequence $\gamma_i, \gamma_{i+1}, \cdots \gamma_j$ is a learning interval *if γ_i and γ_j are consecutive critical states. The interval is* type 3 *if γ_i learns v_3^I, and otherwise the interval is* type 2.

Thus the first learning interval is type 2, begins with γ_0, and ends at the first state which learns v_3^I ,or at γ_T if no state in the computation learns v_3^I. After that all type 2 learning intervals begin with a state learning v_2^I and end with γ_T or with a state learning v_3^I, and *vice versa* for learning intervals of type 3. Type 2 learning intervals never learn v_3^I until the last state, and type 3 learning intervals never learn v_2^I until the last state.

Now let B be a nondeterministic branching program that solves $BT_2^3(k)$. For $j \in \{2,3\}$ let Γ_j be the set of all states of B which query the input function f_j. We will prove the theorem by showing that for large k

$$|\Gamma_2| + |\Gamma_3| > k^2\sqrt{k}. \tag{6}$$

Given four numbers a_4, a_5, a_6, a_7 in $[k]$ we define $F_{yes}^{a_4,a_5,a_6,a_7}$ to be the set of all YES inputs I to B whose four leaves are labelled a_4, a_5, a_6, a_7 respectively, whose middle node functions f_2^I and f_3^I are identically 0 except $f_2^I(a_4, a_5) = v_2^I$ and $f_3^I(a_6, a_7) = v_3^I$. Since I is a YES input it follows that $f_1^I(v_2^I, v_3^I) = 1$.

Note that each member I of $F_{yes}^{a_4,a_5,a_6,a_7}$ is uniquely specified by a triple

$$(v_2^I, v_3^I, f_1^I) \text{ where } f_1^I(v_2^I, v_3^I) = 1 \tag{7}$$

and hence $F_{yes}^{a_4,a_5,a_6,a_7}$ has exactly $k^2(2^{k^2-1})$ members.

For $j \in \{2,3\}$ and $a, b \in [k]$ let $\Gamma_j^{a,b}$ be the subset of Γ_j consisting of those states which compute $f_j(a, b)$. Then Γ_j is the disjoint union of $\Gamma_j^{a,b}$ over all pairs (a, b) in $[k] \times [k]$. Hence to prove (6) it suffices to show

$$|\Gamma_2^{a_4,a_5}| + |\Gamma_3^{a_6,a_7}| > \sqrt{k} \tag{8}$$

for large k and all a_4, a_5, a_6, a_7 in $[k]$. We will show this by showing

$$(|\Gamma_2^{a_4,a_5}| + 1)(|\Gamma_3^{a_6,a_7}| + 1) \geq k/2 \tag{9}$$

for all $k \geq 2$. (Note that given the product, the sum is minimized when the summands are equal.)

For each input I in $F_{yes}^{a_4,a_5,a_6,a_7}$ we associate a fixed accepting computation $\mathcal{C}(I)$ of B on input I.

For $a, b \in [k]$ and $f : [k] \times [k] \to \{0,1\}$ with $f(a, b) = 1$ we use (a, b, f) to denote the input I in $F_{yes}^{a_4,a_5,a_6,a_7}$ it represents as in (7).

Lemma. *Assume (9) is false. Then $\exists a, b \in [k]$ and $\exists f, g : [k] \times [k] \to \{0,1\}$ such that $f(a, b) = 1$ and $g(a, b) = 0$ and for all states γ, δ, if there is a learning interval beginning with γ and ending with δ in the computation $\mathcal{C}(a, b, f)$ and the interval is type 2 then $\exists b' \in [k]$ such that $g(a, b') = 1$ and $\mathcal{C}(a, b', g)$ contains a*

type 2 learning interval beginning with γ and ending with δ, and if the interval is type 3 then $\exists a' \in [k]$ such that $g(a', b) = 1$ and the computation $\mathcal{C}(a', b, g)$ contains a type 3 learning interval beginning with γ and ending with δ.

We first show that (9), and hence the theorem, follows from the lemma. If (9) is false, then by the lemma we can show that B accepts the NO input (a, b, g). The accepting computation is obtained by modifying the accepting computation $\mathcal{C}(a, b, f)$ by replacing each learning interval beginning with some γ and ending with some δ in that computation by the corresponding learning interval in the computation $\mathcal{C}(a, b', g)$ or $\mathcal{C}(a', b, g)$. This works because a type 2 learning interval never computes v_3 and a type 3 learning interval never computes v_2.

It remains to prove the lemma. We may assume that the branching program B has a unique initial state γ_0 and a unique accepting state δ_{ACC}.

For $j \in \{2, 3\}$, $a, b \in [k]$ and $f : [k] \times [k] \to \{0, 1\}$ with $f(a, b) = 1$ define $\varphi_j(a, b, f)$ to be the set of all state pairs (γ, δ) such that there is a type j learning interval in $\mathcal{C}(a, b, f)$ which begins with γ and ends with δ. Note that if $j = 2$ then $\gamma \in (\Gamma_2^{a_4,a_5} \cup \{\gamma_0\})$ and $\delta \in (\Gamma_3^{a_6,a_7} \cup \{\delta_{ACC}\})$, and if $j = 3$ then $\gamma \in \Gamma_3^{a_6,a_7}$ and $\delta \in (\Gamma_2^{a_4,a_5} \cup \{\delta_{ACC}\})$

To complete the definition, define $\varphi_j(a, b, f) = \varnothing$ if $f(a, b) = 0$.

For $j \in \{2, 3\}$ and $f : [k] \times [k] \to \{0, 1\}$ we define a function $\varphi_j[f]$ from $[k]$ to sets of state pairs as follows:

$$\varphi_2[f](a) = \bigcup_{b \in [k]} \varphi_2(a, b, f) \ \subseteq S_2$$

$$\varphi_3[f](b) = \bigcup_{a \in [k]} \varphi_3(a, b, f) \ \subseteq S_3$$

where $S_2 = (\Gamma_2^{a_4,a_5} \cup \{\gamma_0\}) \times (\Gamma_3^{a_6,a_7} \cup \{\delta_{ACC}\})$ and $S_3 = \Gamma_3^{a_6,a_7} \times (\Gamma_2^{a_4,a_5} \cup \{\delta_{ACC}\})$.

For each f the function $\varphi_j[f]$ can be specified by listing a k-tuple of subsets of S_j, and hence there are at most $2^{k|S_j|}$ distinct such functions as f ranges over the 2^{k^2} Boolean functions on $[k] \times [k]$, and hence there are at most $2^{k(|S_2|+|S_3|)}$ pairs of functions $(\varphi_2[f], \varphi_3[f])$. By our assumption that (9) is false, we have $|S_2| + |S_3| < k$. Hence by the pigeonhole principle there must exist distinct Boolean functions f, g such that $\varphi_2[f] = \varphi_2[g]$ and $\varphi_3[f] = \varphi_3[g]$.

Since f and g are distinct we may assume that there exist a, b such that $f(a, b) = 1$ and $g(a, b) = 0$. Since $\varphi_2[f](a) = \varphi_2[g](a)$, if (γ, δ) are the endpoints of a type 2 learning interval in $\mathcal{C}(a, b, f)$ there exists b' such that (γ, δ) are the endpoints of a type 2 learning interval in $\mathcal{C}(a, b', g)$ (and hence $g(a, b') = 1$). Similarly for type 3 intervals. This proves the lemma and completes the proof of Theorem 4.7. □

Theorem 4.8. *Every deterministic branching program that solves $BT_2^3(k)$ has at least $k^3/\lg k$ states for sufficiently large k.*

Proof. We modify the proof of Theorem 4.7. Let B be a deterministic BP which solves $T_2(3, k)$, and for $j \in \{2, 3\}$ let Γ_j be the set of states in B which query f_j.

It suffices to show that for sufficiently large k

$$|\Gamma_2| + |\Gamma_3| \geq k^3/\lg k. \tag{10}$$

For $r, s \in [k]$ we define the set $F^{r,s}$ of inputs I of B to be those which satisfy the following restrictions:

- The leaves satisfy $v_4^I = v_6^I = r$ and $v_5^I = v_7^I = s$.
- For $j \in \{2, 3\}$ the node function f_j satisfies $f_j(x, y) = 0$ unless $x = r$ and $y = s$. (Here we assume $[k] = \{0, \ldots, k-1\}$.)

Each member of I of $F^{r,s}$ is uniquely specified by a triple

$$(a, b, f)$$

where $a, b \in [k]$ and $f : [k] \times [k] \to \{0, 1\}$; namely $v_2^I = a, v_3^I = b, f_1 = f$. So for each r, s there are exactly $k^2 2^{k^2}$ inputs in $F^{r,s}$.

For $j \in \{2, 3\}$ let $\Gamma_j^{r,s}$ be the set of states of B which query $f_j(r, s)$. Then Γ_j is the disjoint union

$$\Gamma_j = \bigcup_{r,s \in [k]} \Gamma_j^{r,s}$$

To prove (10) it suffices to show that for sufficiently large k and all r, s in $[k]$

$$|\Gamma_2^{r,s}| + |\Gamma_3^{r,s}| \geq k/\lg k. \tag{11}$$

We may assume there are unique start, accepting, and rejecting states γ_0, δ_{ACC}, δ_{REJ}. Fix $r, s \in [k]$.

For each root function $f : [k] \times [k] \to \{0, 1\}$ we define the functions

$$\psi_2[f] : [k] \times (\Gamma_2^{r,s} \cup \{\gamma_0\}) \to (\Gamma_3^{r,s} \cup \{\delta_{ACC}, \delta_{REJ}\})$$
$$\psi_3[f] : [k] \times \Gamma_3^{r,s} \to (\Gamma_2^{r,s} \cup \{\delta_{ACC}, \delta_{REJ}\})$$

by $\psi_2[f](a, \gamma) = \delta$ if δ is the next critical state after γ in a computation with input (a, b, f) (this is independent of b), or $\delta = \delta_{REJ}$ if there is no such critical state. Similarly $\psi_3[f](b, \delta) = \gamma$ if γ is the next critical state after δ in a computation with input (a, b, f) (this is independent of a), or $\delta = \delta_{REJ}$ if there is no such critical state.

CLAIM: The pair of functions $(\psi_2[f], \psi_3[f])$ is distinct for distinct f.

For suppose otherwise. Then there are f, g such that $\psi_2[f] = \psi_2[g]$ and $\psi_3[f] = \psi_3[g]$ but $f(a, b) \neq g(a, b)$ for some a, b. But then the sequences of critical states in the two computations $C(a, b, f)$ and $C(a, b, g)$ must be the same, and hence the computations either accept both (a, b, f) and (a, b, g) or reject both. So the computations cannot both be correct.

Now we prove (11) from the CLAIM. Let $s_2 = |\Gamma_2^{r,s}|$ and let $s_3 = |\Gamma_3^{r,s}|$, and let $s = s_2 + s_3$. Then the number of distinct pairs (ψ_2, ψ_3) is at most

$$(s_3 + 2)^{k(s_2+1)}(s_2 + 2)^{ks_3} \leq (s + 2)^{k(s+1)}$$

and since there are 2^{k^2} functions f we have

$$2^{k^2} \leq (s+2)^{k(s+1)}$$

so taking logs, $k^2 \leq k(s+1)\lg(s+2)$ so $k/\lg(s+2) \leq s+1$, and (11) follows. This proves the CLAIM and completes the proof of Theorem 4.8. □

Recall from Theorem 4.5 that applying the Nečiporuk method to $Children_2^4(k)$ yields an $\Omega(k^3)$ size lower bound and from Theorem 4.6 that applying it to $SumMod_2^3(k)$ yields $\Omega(k^2)$. The next two theorems are proved in the Appendix.

Theorem 4.9. *Any deterministic k-way BP for $Children_2^4(k)$ has at least $k^4/2$ states.*

Theorem 4.10. *Any deterministic k-way BP for $SumMod_2^3(k)$ requires at least k^3 states.*

5 Conclusion

Our main open question is whether we can adapt the state sequence method to break the $\Omega(n^2)$ barrier for the size of deterministic branching programs. In particular, can the method be extended to handle trees of height 4? Specifically, can we prove a lower bound of $\Omega(k^7/\lg k)$ for $BT_3^4(k)$ (see Theorem 4.1)?

Another question arises from the $O(k^{5/2})$ upper bound arising in Theorem 4.1. Is there a pebbling to justify such a non-integral exponent? As it turns out, the answer is yes. One can introduce fractional black-white pebbling and develop an interesting theory. Our work on that issue will be the subject of another paper.

Acknowledgment James Cook played a helpful role in the early parts of this research. The second author is grateful to Michael Taitslin for suggesting a version of the tree evaluation problem in which the nodes are labelled by quasi groups (see [**?**]).

Appendix

Proof of Theorem 2.3. $\#\mathsf{pebbles}(T_d^h) = (d-1)h - d + 2$.

Proof. For $h = 2$ this gives $\#\mathsf{pebbles}(T_d^2) = d$, which is obviously correct. In general we show $\#\mathsf{pebbles}(T_d^{h+1}) = \#\mathsf{pebbles}(T_d^h)+d-1$, from which the theorem follows.

The following pebbling strategy gives the upper bound: Let the root be node 1 and the children be $2 \ldots d+1$. Pebble the nodes $2 \ldots d+1$ in order using the optimal number of pebbles for T_d^{h-1}, leaving a black pebble at each node. Note that for the black pebble game, the complexity of pebbling in the game where a pebble remains on the root is the same as for the game where the root has a black pebble on it at some point. The maximum number of pebbles at any point on the tree is $d - 1 + \#\mathsf{pebbles}(T_d^{h-1})$. Now slide the black pebble from node 1 to the root, and then remove all pebbles.

For the lower bound, consider the time t at which the children of the root all have black pebbles on them. There must be a final time t' before t at which one of the sub-trees rooted at $2, 3, \ldots d+1$ had $\#\mathsf{pebbles}(T_d^h)$ pebbles on it. This is because pebbling any of these subtrees requires at least $\#\mathsf{pebbles}(T_d^h)$ pebbles, by definition. At time t', all the other subtrees must have at least 1 black pebble each on them. If not, then there is a subtree T which does not, and it would have to be pebbled before time t, which contradicts the definition of t'. Thus at time t', there are at least $\#\mathsf{pebbles}(T_d^h) + d - 1$ pebbles on the tree. □

Proof of Theorem 3.1. For each $d \geq 2$, $BT_d(h,k) \in$ **LogDCFL**.

Proof. By [**?**] if suffices to show that $BT_d(h,k)$ is solved by some deterministic auxiliary pushdown automaton M in lg space and polynomial time. The algorithm for M is to use its stack to perform a depth-first search of the tree T_d^h, where for each node i it keeps a partial list of the values of the children of i, until it obtains all d values, at which point it computes the value of i and pops its stack, adding that value to the list for the parent node. □

Proof of Theorem 3.4. If T_d^h can be pebbled with p pebbles, then deterministic branching programs with $O(k^p)$ states can solve $FT_d^h(k)$ and $BT_d^h(k)$.

Proof. Consider the sequence $C_0, C_1, \ldots C_\tau$ of pebble configurations for a pebbling of T_d^h using p pebbles. We may as well assume that the root is pebbled in configuration C_τ, since all pebbles could be removed in one more step at no extra cost in pebbles. We design a branching program B for solving $FT_d^h(k)$ as follows. For each pebble configuration C_t, program B has k^p states; one state for each possible assignment of a value from $[k]$ to each of the p pebbles. Hence B has $O(k^p)$ states, since τ is a constant independent of k. Consider an input I to $FT_d^h(k)$, and let v_i be the value in $[k]$ which I assigns to node i in T_d^h (see Definition 1.1). We design B so that on I the computation of B will be a state sequence $\alpha_0, \alpha_1, \ldots, \alpha_\tau$, where the state α_t assigns to each pebble the value v_i of the node i that it is on. (If a pebble is not on any node, then its value is 1.)

For the initial pebble configuration no pebbles have been assigned to nodes, so the initial state of B assigns the value 1 to each pebble. In general if B is in a state α corresponding to configuration C_t, and the next configuration C_{t+1} places a pebble j on node i, then the state α queries the node i to determine v_i, and moves to a new state which assigns v_i to the pebble j and assigns 1 to any pebble which is removed from the tree. Note that if i is an internal node, then all children of i must be pebbled at C_t, so the state α 'knows' the values $v_{j_1}, \ldots, v_{j_d}$ of the children of i, so α queries $f_i(v_{j_1}, \ldots, v_{j_d})$.

When the computation of B reaches a state α_τ corresponding to C_τ, then α_τ determines the value of the root (since C_τ has a pebble on the root), so B moves to a final state corresponding to the value of the root. □

Proof of Theorem 4.1.

$$\#\mathsf{detBstates}_d^h(k) = O(k^{(d-1)h-d+2}) \tag{12}$$

$$\#\mathsf{detFstates}_d^h(k) = O(k^{(d-1)h-d+2}) \tag{13}$$

$$\#\mathsf{ndetBstates}_2^3(k) = O(k^{5/2}) \tag{14}$$

$$\#\mathsf{detBstates}_2^h(k) = O(k^h/\lg k), \text{ for } h \geq 3 \tag{15}$$

Proof. The first two bounds follow immediately from Theorems 2.3 and 3.4. We now prove (14) for the case $d = 2$. The general case is similar.

Let $\ell \leq \lg k + 2$ be an even integer at least as big as the number of bits in the binary notation for k.

Let v_i denote the value of node i in T_2^3, where the nodes are numbered as in Fig. 1. The following algorithm by a nondeterministic k-way BP solves BT_2^3 in $O(k^{5/2})$ states.

1. Allocate $O(k^2)$ states to compute the leaf values v_4 and v_5 and use them to compute v_2.
2. Allocate $O(k^2 \cdot 2^{\ell/2}) = O(k^{5/2})$ states to remember just the $\ell/2$ most significant bits of v_2 and to compute v_3 as above.
3. Allocate $O(k^2)$ states to guess at the low-order $\ell/2$ bits of v_2 to reconstruct v_2 and use this and v_3 to compute v_1. If $v_1 \neq 1$ then REJECT.
4. Forget all node values except the guessed $\ell/2$ low order bits of v_2. Now recompute v_2 as in step 1, and ACCEPT iff the guessed values are correct.

To prove (15) we use a branching program which follows the following algorithm. Here we have a parameter m, which is optimized when $m = \lg k - \lg\lg k$. In the following analysis we estimate the number of states required up to a constant factor.

1) Compute v_2 (the value of node 2 in the heap ordering), using the black pebbling algorithm for the principal left subtree. This requires k^{h-1} states. Divide the k possible values for v_2 into k/m blocks of size m.

2) Remember the block number for v_2, and compute v_3. This requires $k/m \times k^{h-1} = k^h/m$ states.

3) Remember v_3 and the block number for v_2. Compute $f_1(a, v_3)$ for each of the m possible values a for v_2 in its block number, and keep track of the set of a's for which $f_1 = 1$. This requires $k \times k/m \times m \times 2^m = k^2 2^m$ states.

4) Remember just the set of possible a's (within its block) from above (there are 2^m possibilities). Compute v_2 again and accept or reject depending on whether v_2 is in the subset. This requires $k^{h-1}2^m$ states.

The total number of states has order the $\max\{k^h/m, k^{h-1}2^m\}$, which is minimized when $m = \lg k - \lg\lg k$. □

Proof of Theorem 4.3. Applying the Nečiporuk method yields Figure 1.

Proof. We have $N_{\text{det}}^{k\text{-way}}(s, v) \leq v^s \cdot (s + |R|)^{sk}$ for the number of deterministic BPs and $N_{\text{nondet}}^{k\text{-way}}(s, v) \leq v^s \cdot (|R|+1)^{sk} \cdot (2^s)^{sk}$ for nondeterministic BPs having s non-sink states, each labelled with one of v variables. To see $N_{\text{nondet}}^{k\text{-way}}(s, v)$, note that an edge labelled $i \in [k]$ can connect a state S to zero or one state among the sink states and can connect S independently to any number of states among the non-sink states.

The only decision to make when applying the Nečiporuk method is the choice of the partition of the input variables. Here every entry in Figure 1 is obtained using the same partition (with the proviso that a k-ary variable in the partition is replaced by $\lg k$ binary variables when we treat 2-way branching programs).

We will only partition the set V of k-ary $FT_d^h(k)$ or $BT_d^h(k)$ variables that pertain to internal tree nodes other than the root (we will neglect the root and leaf variables). Each internal tree node has $d-1$ siblings and each sibling involves k^d variables. By a *litter* we will mean any set of d k-ary variables that pertain to precisely d such siblings. We obtain our partition by writing V as a union of

$$k^d \cdot \Sigma_{i=0}^{h-3} d^i = k^d \cdot \frac{d^{h-2} - 1}{d - 1}$$

litters. (Specifically, each litter can be defined as

$$\{f_i(j_1, j_2, \ldots, j_d), f_{i+1}(j_1, j_2, \ldots, j_d), \ldots, f_{i+d-1}(j_1, j_2, \ldots, j_d)\}$$

for some $1 \leq j_1, j_2, \ldots, j_d \leq k$ and some d siblings $i, i+1, \ldots, i+d-1$.)

Consider such a litter L. We claim that $|D|^{k^d}$ distinct functions $f_L : [k]^d \to D$ can be induced by setting the variables outside of L, where $|D| = k$ in the case of $FT_d^h(k)$ and $|D| = 2$ in the case of $BT_d^h(k)$. Indeed, to induce any such function, fix the "descendants of the litter L" to make each variable in L relevant to the output; then, set the variables pertaining to the immediate ancestor node ν of the siblings forming L to the appropriate k^d values, as if those were the final output desired; finally, set all the remaining variables in a way such that the values in ν percolate from ν to the root.

It remains to do the calculations. We illustrate two cases. Similar calculations yield the other entries in Figure 1.

Nondeterministic k-way branching programs computing $FT_d^h(k)$. Here $|R| = k$. In a correct program, the number s of states querying one of the d litter L variables must satisfy

$$k^{k^d} \leq N_{\text{nondet}}^{k\text{-way}}(s,d) \leq d^s \cdot (k+1)^{sk} \cdot (2^s)^{sk} \leq s^s \cdot k^{2sk} \cdot (2^s)^{sk}$$

since $d \leq s$ (because $FT_d^h(k)$ depends on all its variables), and thus

$$k^d \lg k \leq s(\lg s + 2k \lg k) + s^2 k.$$

Suppose to the contrary that $s < (k^{\frac{d-1}{2}}\sqrt{\lg k})/2$. Then

$$s(\lg s + 2k\lg k) + s^2 k < s(\frac{d-1}{2}\lg k + \frac{\lg\lg k}{2} + 2k\lg k) + s^2 k < s(sk) + s^2 k < k^d \lg k$$

for large k and all $d \geq 2$, a contradiction. Hence $s \geq (k^{\frac{d-1}{2}}\sqrt{\lg k})/2$. Since this holds for every litter, the total number of states in the program is at least

$$k + k^d \cdot \frac{d^{h-2}-1}{d-1} \cdot (k^{\frac{d-1}{2}}\sqrt{\lg k})/2 \geq \frac{d^{h-2}-1}{2d-2} \cdot k^{\frac{3d}{2}-\frac{1}{2}}\sqrt{\lg k}.$$

Nondeterministic binary (ie 2-way) branching programs deciding $BT_d^h(k)$. Here $|R| = 2$. When the program is binary, the d variables in the litter L become $d\lg k$ Boolean variables. The number s of states querying one of these $d\lg k$ variables then verifies

$$2^{k^d} \leq N_{\text{nondet}}^{k\text{-way}}(s, d\lg k) \leq (d\lg k)^s \cdot (2+1)^{2s} \cdot (2^s)^{2s} < (s\lg k)^s \cdot 2^{4s+2s^2}$$

since $d \leq s$ and thus

$$k^d \leq s\lg s + s\lg\lg k + 4s + 2s^2 \leq 3s^2 + 5s\lg\lg k.$$

It follows that $s \geq k^{\frac{d}{2}}/2$. Hence the total number of states in a binary nondeterministic program deciding $BT_d^h(k)$ is at least

$$k^d \cdot \frac{d^{h-2}-1}{d-1} \cdot k^{d/2}/2 \geq \frac{d^{h-2}-1}{2(d-1)} \cdot k^{\frac{3d}{2}} = \frac{d^{h-2}-1}{2(d-1)} \cdot \frac{(k^d\lg k)^{3/2}}{(\lg k)^{3/2}} = \Omega(n^{3/2}/(\lg n)^{3/2})$$

where $n = \Theta(k^d \lg k)$ is the length of the binary encoding of $BT_d^h(k)$. □

Proof of Theorem 4.5. For any fixed $d, h \geq 2$, the best k-way deterministic BP size lower bound attainable for $Children_d^h(k)$ by the Nečiporuk method is $\Omega(k^{2d-1})$.

Proof. The function $Children_d^h(k) : [k]^m \to R$ has $m = \Theta(k^d)$. Any partition $\{V_1, \ldots, V_p\}$ of the set of k-ary input variables thus has $p = O(k^d)$. Claim: for each i, the best attainable lower bound on the number of states querying variables from V_i is $O(k^{d-1})$.

Consider such a set V_i, $|V_i| = v \geq 1$. Here $|R| = k^d$, so the number $N_{\text{det}}^{k\text{-way}}(s, v)$ of distinct deterministic BPs having s non-sink states querying variables from V_i satisfies

$$\begin{aligned} N_{\text{det}}^{k\text{-way}}(s, v) &\geq 1^s \cdot (s + |R|)^{sk} \\ &\geq (1 + k^d)^{sk} \\ &\geq k^{dsk}. \end{aligned}$$

Hence the estimate used in the Nečiporuk method as an upper bound to $N_{\text{det}}^{k\text{-way}}(s, v)$ will be at least k^{dsk}. On the other hand, the number of functions $f_{V_i} : [k]^v \to R$ obtained by fixing variables outside of V_i cannot exceed $k^{O(k^d)}$ since the number of variables outside V_i is $\Theta(k^d)$. Hence the best lower bound on the number of states querying variables from V_i obtained by applying the method will be no larger than the smallest s verifying $k^{ck^d} \leq k^{dsk}$ for some c depending on d and k. This proves our claim since then this number is at most $s = O(k^{d-1})$.

Proof of Theorem 4.6. The best k-way deterministic BP size lower bound attainable for $SumMod_2^3(k)$ by the Nečiporuk method is $\Omega(k^2)$.

Proof. The function $SumMod_2^3(k) : [k]^m \to R$ has $m = \Theta(k^2)$. Consider a set V_i in any partition $\{V_1, \ldots, V_p\}$ of the set of k-ary input variables, $|V_i| = v$. Here $|R| = k$, so the number $N_{\text{det}}^{k\text{-way}}(s, v)$ of distinct deterministic BPs having s non-sink states querying variables from V_i satisfies

$$\begin{aligned} N_{\text{det}}^{k\text{-way}}(s, v) &\geq 1^s \cdot (s + |R|)^{sk} \\ &\geq (1 + k)^{sk} \\ &\geq k^{sk}. \end{aligned}$$

If V_i contains a leaf variable, then perhaps the number of functions induced by setting variables complementary to V_i can reach the maximum k^{k^2}. Nečiporuk would conclude that k states querying the variables from such a V_i are necessary. Note that there are at most 4 sets V_i containing a leaf variable (hence a total of $4k$ states required to account for the variables in these 4 sets). Now suppose that V_i does not contain a leaf variable. Then setting the variables complementary to V_i can either induce a constant function (there are k of those), or the sum of a constant plus a variable (there are at most $k \cdot |V_i|$ of those) or the sum of two of the variables (there are at most $|V_i|^2$ of those). So the maximum number of induced functions is $|V_i|^2 = O(k^4)$. The number of states querying variables from V_i is found by Nečiporuk to be $s \geq 4/k$. In other words $s = 1$. So for any of the at least $p - 4$ sets in the partition not containing a leaf variable, the method gets one state. Since $p - 4 = O(k^2)$, the total number of states accounting for all the V_i is $O(k^2)$.

Proof of theorem 4.9 Any deterministic k-way BP computing $Children_2^4(k)$ has at least $k^4/2$ states.

Proof. Let E_{4true} be the set of all inputs I to $Children_2^4(k)$ such that $f_2^I = f_3^I = +_k$ (addition mod k), and for $i \in \{4,5,6,7\}$ f_i^I is identically 0 except for $f_i^I(v_{2i}^I, v_{2i+1}^I)$.

Let B be a branching program as in the theorem. For each $I \in E_{4true}$ let $\mathcal{C}(I)$ be the computation of B on input I.

For $r, s \in [k]$ let $E_{4true}^{r,s}$ the set of inputs I in E_{4true} such that for $i \in \{4,5,6,7\}$, $v_{2i}^I = r$ and $v_{2i+1}^I = s$. Then for each pair r, s each input I in $E_{4true}^{r,s}$ is completely specified by the quadruple $v_4^I, v_5^I, v_6^I, v_7^I$, so $|E_{4true}^{r,s}| = k^4$.

For $r, s \in [k]$ and $i \in \{4,5,6,7\}$ let $\Gamma_i^{r,s}$ be the set of states of B that query $f_i(r,s)$, and let

$$\Gamma^{r,s} = \Gamma_4^{r,s} \cup \Gamma_5^{r,s} \cup \Gamma_6^{r,s} \cup \Gamma_7^{r,s} \tag{16}$$

The theorem follows from the following Claim.

CLAIM 1: $|\Gamma^{r,s}| \geq k^2/2$ for all $r, s \in [k]$.

To prove CLAIM 1, suppose to the contrary for some r, s

$$|\Gamma^{r,s}| < k^2/2 \tag{17}$$

We associate a pair

$$T(I) = (\gamma^I, v_i^I)$$

with I as follows: γ^I is the last state in the computation $\mathcal{C}(I)$ that is in $\Gamma^{r,s}$ (such a state clearly exists), and $i \in \{4,5,6,7\}$ is the node queried by γ^I. (Here v_i^I is the value of node i).

We also associate a second triple $U(I)$ with each input I in $E_{4true}^{r,s}$ as follows:

$$U(I) = \begin{cases} (v_4^I, v_5^I, v_3^I) \text{ if } \gamma^I \text{ queries node 4 or 5} \\ (v_6^I, v_7^I, v_2^I) \text{ otherwise.} \end{cases}$$

CLAIM 2: As I ranges over $E_{4true}^{r,s}$, $U(I)$ ranges over at least $k^3/2$ triples in $[k]^3$.

To prove CLAIM 2, consider the the subset E' of inputs in $E_{4true}^{r,s}$ whose values for nodes 4,5,6,7 have the form a, b, a, c for arbitrary $a, b, c \in [k]$. For each such I in E' an adversary trying to minimize the number of triples $U(I)$ must choose one of the two triples $(a, b, a +_k c)$ or $(a, c, a +_k b)$. There are a total of k^3 distinct triples of each of the two forms, and the adversary must choose at least half the triples from one of the two forms, so there must be at least $k^3/2$ distinct triples of the form $U(I)$. This proves CLAIM 2.

On the other hand by (17) there are fewer than $k^3/2$ possible values for $T(I)$. Hence there exist inputs $I, J \in E_{4true}^{r,s}$ such that $U(I) \neq U(J)$ but $T(I) = T(J)$. Since $U(I) \neq U(J)$ but $v_i^I = v_i^J$ (where i is the node queried by $\gamma^I = \gamma^J$) it follows that either $v_2^I \neq v_2^J$ or $v_3^I \neq v_3^J$, so I and J give different values to the function $Children_2^4(k)$. But since $T(I) = T(J)$ if follows that the two computations $\mathcal{C}(I)$ and $\mathcal{C}(J)$ are in the same state $\gamma^I = \gamma^J$ the last time any of the nodes $\{4,5,6,7\}$ is queried, and the answers $v_i^I = v_i^J$ to the queries are the same, so both computations give identical outputs. Hence one of them is wrong. □

Proof of theorem 4.10 Any deterministic k-way BP for $SumMod_2^3(k)$ requires as least k^3 states.

Proof. We adapt the previous proof. Now $E^{r,s}$ is the set of inputs I to $SumMod_2^3(k)$ such that for $i \in \{2,3\}$, f_i^I is identically zero except for $f_i^I(r,s)$, and $v_4^I = v_6^I = r$ and $v_5^I = v_7^I = s$. Note that an input to $E^{r,s}$ can be specified by the pair (v_2^I, v_3^I), so $E^{r,s}$ has exactly k^2 elements. Define

$$\Gamma^{r,s} = \Gamma_2^{r,s} \cup \Gamma_3^{r,s}$$

Now we claim that an input I in $E^{r,s}$ can be specified by the pair (γ^I, v_i^I), where γ^I is the last state in the computation $\mathcal{C}(I)$ that is in $\Gamma^{r,s}$, and $i \in \{2,3\}$ is the node queried by γ^I.

The Claim holds because (γ^I, v_i^I) determines the output of the computation, which in turn (together with v_i^I) determines v_j^I, where j is the sibbling of i.

From the Claim it follows that $|\Gamma^{r,s}| \geq k$ for all $r, s \in [k]$, and hence there must be at least k^3 states in total.