Lower bound for deterministic semantic-incremental branching programs solving GEN

Dustin Wehr

January 7, 2011

Abstract

We answer a problem posed in [GKM08] regarding a restricted model of small-space computation, tailored for solving the GEN problem. They define two variants of "incremental branching programs", the *syntactic* variant defined by a restriction on the graph-theoretic paths in the program, and the more-general *semantic* variant in which the same restriction is enforced only on the consistent paths - those that are followed by at least one input. They show that exponential size is required for the syntactic variant, but leave open the problem of superpolynomial lower bounds for the semantic variant. Here we give an exponential lower bound for the semantic variant by generalizing lower bound arguments from [BCM$^+$09] [Weh10] for a similar restricted model tailored for solving a special case of GEN called Tree Evaluation.

1 How to read this paper

The introduction 2 (which is short and should be read entirely) defers several definitions to Section 3.1; the reader should refer there to read any unfamiliar definitons as they arise. The proof of our main result spans Sections 3.2, 4.1, 4.3. All but the most-casual readers should read Section 3.2, which sets up and outlines the proof. Sections 3.3 and 4.2 can safely be skipped by readers only interested in our main result.

2 Introduction

An instance T of m-GEN is just a function from $[m] \times [m]$ to $[m]$, where $[m] = \{1, \ldots, m\}$. T is a YES instance iff m is in the closure of the set $\{1\}$ under the operation T. Depending on the computation model, T is given as $m^2 \lceil \log m \rceil$ bits or, more naturally, as m^2 elements of $[m]$. The computation model we use, m-way branching programs (defn 1), is the standard one for studying the nonuniform space complexity of problems represented in the second way.

We refer to the m^2, $[m]$-valued input variables that define the m-GEN instances by the names $\{(x, y)\}_{x,y \in [m]}$, and we refer to input T's value of variable (x, y) by $T(x, y)$. Throughout, we only talk about deterministic branching programs. A **deterministic semantic-incremental** branching program (BP) solving m-GEN is an m-way BP B that computes m-GEN such that for every state q of B that queries a variable (x, y) and every input T that visits q, for both $z \in \{x, y\}$ either $z = 1$ or there is an earlier edge on the computation path of T labeled z. The main goal is Corollary 1 in Section 4.3:

> There is a constant $c > 0$ such that for infinitely-many m every deterministic semantic-incremental BP solving m-GEN has at least $2^{c\,m/\log m}$ states.

3 Preliminaries / Outline

3.1 Definitions

Definition 1 (k-way branching program)**.** A *deterministic k-way branching program B* computing a function $g : [k]^{|Vars|} \to Out$, where $Vars$ and Out are finite sets, is first of all a directed multi-graph whose nodes are called *states*, having a unique in-degree 0 state called the *start state*. Every state is labeled with an *input variable* (an element of $Vars$) except for $|Out|$ *output* states with out-degree 0 labelled with distinct *output values* (the elements of Out). Every state has k out-edges, labeled with distinct elements of $[k]$. An *input* I (a mapping $X \mapsto X^I$ from $Vars$ to $[k]$) defines a *computation path* from the start state through B in the obvious way: from a non-output state q labeled with $X \in Vars$, I follows the edge out of q labeled X^I. The computation path of I must be finite, ending at the output state labeled with $g(I)$. The *size* of B is its number of states. We say that B solves a decision problem if $|Out| = 2$.

Definition 2 (rooted dag, root u_{root}, leaf, child, parent, arc, size)**.** A **rooted dag** G is a directed acyclic graph with a unique out-degree 0 node called the **root**, denoted u_{root}. In-degree 0 nodes are called **leaves**. We refer to the edges of G as **arcs** in order to avoid confusion with the edges of a branching program. The nodes with arcs into u are the **children** of u and the nodes that receive arcs coming out of u are the **parents** of u.

Definition 3 (Dag Evaluation Problem)**.** An input is a 4-tupple $\langle G, k, \vec{l}, \vec{f} \rangle$. G is a connected rooted dag and $k \geq 2$ is an integer. $\vec{l}$ consists of a $\lceil \log k \rceil$-bit string specifying a value in $[k]$ for each leaf node of G, and $\vec{f}$ consists of a $k^d \lceil \log k \rceil$-bit string specifying a function from $[k]^d$ to $[k]$ for each non-leaf node of G with d children. Each non-leaf receives a value in the expected way; namely, by applying its function to the values of its children. The function version of the problem asks for the value of the root. The decision version asks if the root value is 1.

The next definition subsumes the previous one; it makes precise how inputs to the Dag Evaluation Problem are presented to k-way BPs, and introduces notation that we will use throughout this paper. The variable G denotes a connected rooted dag (see definition 2) with at least two nodes throughout this paper.

Definition 4 (DE_G : Dag Evaluation Problem for fixed dag G)**.** The size of an input to DE_G is determined by a parameter $k \geq 2$, and we write $\text{DE}_G(k)$ for the problem restricted to inputs with size paramter k. The $[k]$-valued input variables $Vars$ of $\text{DE}_G(k)$ are as-follows:

$$\begin{array}{rl} l_u & \text{for each leaf } u \in G \\ f_u(\vec{a}) & \text{for each node } u \in G \text{ of in-degree } d \geq 1 \text{ and each } \vec{a} \in [k]^d \end{array}$$

We write l_u^I and $f_u^I(\vec{a})$ for the value input $I\colon Vars \to [k]$ assigns to variables l_u and $f_u(\vec{a})$. For $u \in G$ we define u^I, the *value of I on u*, inductively: if u is a leaf then $u^I = l_u^I$, and if u has children $v_1, \ldots, v_d$ then $u^I = f_u^I(v_1^I, \ldots, v_d^I)$. $\text{DE}_G(k)$ is a decision problem; the output is YES if $u_{\text{root}}^I = 1$ and NO otherwise.

We generalize the definition from [BCM+09] [Weh10] of deterministic thrifty BPs solving the Tree Evaluation Problem (which is the Dag Evaluation Problem for the complete binary trees):

Definition 5. A k-way BP solving $\text{DE}_G(k)$ is thrifty if for every state q that queries an internal node variable $f_u(a_1, \ldots, a_d)$, if $v_1, \ldots, v_d$ are the children of u then every input I that visits q has $v_1^I = a_1, \ldots, v_d^I = a_d$.

Definition 6 (Black pebbling cost of G)**.** Let G be a rooted dag. A **pebbling configuration** C of G is given by a subset of the nodes of G which are said to be **pebbled**. A **complete pebbling sequence** for G is a sequence of pebbling configurations $\pi = C_1, \ldots, C_{t^*}$ such that every node is unpebbled in C_1, the root is pebbled in C_t, and for all $t \in \{1, \ldots, t^* - 1\}$, configuration C_{t+1} is obtained from C_t by one of the following types of **pebbling moves**:

1. If all the children of node u are pebble in C_t, then in C_{t+1} a pebble can placed on u and simultaneously zero or more of the children of u can have their pebbles removed.
2. A pebble is removed from some node.[1]

[1]We don't actually need to include this as a possible move.

We say π requires p pebbles if p is the maximum over all C_t of the number of nodes pebbled in C_t. Finally, the **pebbling cost** of G is the minimum number of pebbles required for a complete pebbling sequence for G.

3.2 Outline of proof

For arbitrary G and k, Theorem 1 gives lower bounds for thrifty BPs solving $\mathrm{DE}_G(k)$ in terms of k and the pebbling cost of G. Let T^h be the complete binary tree with $2^h - 1$ nodes. Theorem 1 is a generalization of the following result from [Weh10], stated in terms of the notation introduced above:

For any $h, k \geq 2$ every deterministic thrifty BP solving $\mathrm{DE}_{T^h}(k)$ has at least k^h states.

Theorem 3 uses Theorem 1 to get lower bounds for semantic-incremental BPs solving m-GEN in terms of the pebbling cost of dags with indegree 2.[2] The bulk of that proof consists of showing that for any G with indegree 2, there is a polynomial-bounded[3] reduction g from DE_G to GEN such that thrifty BPs can efficiently simulate semantic incremental BPs that solve instances of GEN from the range of g.[4] Corollary 1 uses Theorem 3 for each member of a particular hard-to-pebble family of dags, whose existence was proved in [PTC76].

3.3 Remarks on proofs by Gál, Koucký, McKenzie

The authors of [GKM08] obtain exponential lower bounds for syntactic incremental BPs solving m-GEN in two ways. Both methods also work for a nondeterminstic variant of syntactic incremental BPs. First, they use the Raz/McKenzie lower bounds for monotone circuits [RM99] to get a lower bound of 2^{n^ϵ} for some $\epsilon > 0$ and sufficiently large n[5]. The first method works for a possibly-larger larger class of BPs, but the definition of that class is not simple.[6] Their second method uses a probabilistic argument[7] combined with the same pebbling result that we use to get a lower bound of $2^{cn/\log n}$ for some $c > 0$ and sufficiently large n.

4 Results

4.1 Lower Bound for Thrifty BPs

Theorem 1. *If G has pebbling cost p then for any $k \geq 2$ every thrifty deterministic BP solving $DE_G(k)$ has at least k^p states.*

Proof. Fix G, k and a deterministic thrifty BP B that solves $\mathrm{DE}_G(k)$. Let n be the number of nodes in G and Q the states of B. If u is a non-leaf node with d children then the u variables are $f_u(\vec{a})$ for each $\vec{a} \in [k]^d$, and if u is a leaf node then there is just one u variable l_u. We sometimes say "f_u variable" just as an in-line reminder that u is a non-leaf node. When it is clear from the context that a state q is on the computation path of an input I, we just say "q queries u" instead of "q queries the thrifty u variable of I".

We want to assign a black pebbling sequence to each input; to do this we need the following lemma.

Lemma 1. *For any input I and non-leaf node u, there is at least one state q on the computation path of I that queries u,[8] and for every such q, for each child v of u there is a state on the computation path of I before q that queries v.*

[2]This could be generalized to work for families of dags with unbounded indegree, but we have no use for that generalization here.

[3]And very efficiently computable, though we don't need that fact.

[4]More precisely, if E is the set of $\mathrm{DE}_G(k)$ instances, and there is a size s semantic incremental BP solving the set of GEN instances $g(E)$, then there is a thrifty BP solving $\mathrm{DE}_G(k)$ of size at most s.

[5]ϵ not given explicitly.

[6]See section 3.1 "Tight Computation of GEN" of [GKM08].

[7]See Lemma 5.2 "Symmetrization Lemma" of [GKM08].

[8]Recall that "queries u" means queries the thrifty u variable of I.

Proof. Fix an input I. We prove the lemma for I starting with the root, and then the children of the root, and so on. Let $v_1, \ldots, v_d$ be the children of u_{root}. I must visit at least one state that queries its thrifty u_{root} variable, since otherwise B would make a mistake on an input J that is identical to I except

$$f_u^J(v_1^I, \ldots, v_d^I) = \begin{cases} 2 & \text{if } u_{\text{root}}^I = 1 \\ 1 & \text{otherwise} \end{cases}$$

Now let u be any non-leaf node and q any state on the computation path of I that queries u. Suppose the lemma does not hold for this q, so for some child v of u there is no state before q that queries v. For every $a \neq v^I$ there is an input I_a that is identical to I except $v^{I_a} = a$. Now I_a visits q since I and I_a have the same computation path up to q; hence the thrifty assumption is violated. □

We define the pebbling sequence for each input I by following the computation path of I from beginning to end, associating the t-th state visited by I with the t-th pebbling configuration C_t, such that C_{t+1} is either identical to C_t or follows from C_t by applying a valid pebbling move. Let $q_1, \ldots, q_{t^*}$ be the states on the computation path of I up to the state q_{t^*} immediately following the first state that queries the root; C_{t^*} will be the last configuration, and the only configuration where the root is pebbled. Note that q_1 must query a leaf by Lemma 1. We associate q_1 with the empty configuration C_1.

Assume we have defined the configurations $C_1, \ldots, C_t$ associated with the first $t < t^*$ states, and assume $C_1, \ldots, C_t$ is a valid sequence of configurations (where adjacent identical configurations are allowed), but neither it nor any prefix of it is a complete pebbling sequence. We also maintain that for all $t' \leq t$, if the node queried by $q_{t'}$ is not a leaf, then its children are pebbled in $C_{t'}$ and it is not. Let u be the node queried by q_t. By the I.H. u is not pebbled in C_t. We define C_{t+1} by saying how to obtain it by modifying C_t:

1. If u is the root, then $t+1 = t^*$ by the definition of q_{t^*}, and by the I.H. all the children of u are pebbled. Put a pebble on the root and remove the pebbles from its children. This completes the definition of the pebbling sequence for I.
2. If u is not the root or a leaf, then by the I.H. all the children of u are pebbled. For each child v of u: if there is a state q' after q_t that queries some parent of v, and no state between q_t and q' that queries v, then leave the pebble on v, and otherwise remove it.
3. If u is not the root, then place a pebble on it iff there is a state q' after q_t that queries some parent of u and no state between q_t and q' that queries u.

Let p^I be the maximum number of pebbled nodes over all the configurations we just defined. So $p^I \geq p$ since G has pebbling cost p. Let C_t be the earliest configuration with p^I pebbled nodes. Later we will need that q_t is not an output state, so we prove that now.

It suffices to show $t < t^*$, since then there must be at least one state q_{t+1} (possibly an output state) after q_t. We use the assumption that G is connected and has at least two nodes, so the root has degree $d \geq 1$. In the move from C_{t^*-1} to C_{t^*} one pebble is added and d pebbles are removed, so either C_{t^*} has fewer than p^I pebbled nodes (if $d > 1$) or else C_{t^*-1} is an earlier configuration with p^I pebbled nodes. Hence $t < t^*$.

Define the **critical state** r^I for I to be q_t. We refer to the nodes pebbled in C_t as the **bottleneck nodes** of I. The following fact is immediate from the pebbling sequence assignment.

Fact 1. *For any input I, if non-root node u has a pebble at a state q (i.e. the configuration associated with q), then there is a later state q' that queries some parent of u and no state between (inclusive) q and q' that queries u.*

Let D be the set of inputs I such that for every non-leaf node u, if $v_1, \ldots, v_d$ are the children of u then $f_u^I(\vec{a}) = 1$ except possibly when $\vec{a} = \langle v_1^I, \ldots, v_k^I \rangle$. So $|D| = k^n$. Let R be the states that are critical for at least one input in D, and for each $r \in R$ let D_r be the inputs in D with critical state r. The remainder of the proof of Theorem 1 is devoted to the proof of the next lemma.

Lemma 2. $|D_r| \leq k^{n-p}$ *for every* $r \in R$

Let us first see that the theorem follows from the lemma. Since $\{D_r\}_{r \in R}$ is a partition of D, by the lemma there must be at least $|D|/k^{n-p} = k^p$ sets in the partition, i.e. the set of critical states R has size at least k^p, which is what we wanted to show.

Consider a very simple cooperative two player game where Player 1 chooses $r \in R$ and an input I in D_r and gives r to Player 2. Both players know the branching program B. Player 2's goal is to determine I (which is the only thing Player 1 knows that Player 2 does not), which by the definition of D is equivalent to determining the node values of I. Player 1 gets to send an advice strings to Player 2, and it is her goal to minimize the length of the advice stirngs. The lemma says that, for any critical state r chosen by Player 1, advice strings in $[k]^{n-p}$ suffice to enable Player 2 to determine the input in D_r chosen by Player 1. We refer to the individual elements from $[k]$ of an advice string as *words*.

Fix r in R. Let $I \in D_r$ be an input chosen by Player 1, unknown to Player 2. Player 2 will use the advice, together with r and the thrifty property of B, to follow the computation path taken by I from r till I's output state. We will define the advice string so that each word tells Player 2 the value of a different node; when Player 2 learns (the value of) a node in this way, we say he *receives* the value of (I on) that node. There will be at least p nodes –specifically, the bottleneck nodes of I– that Player 2 will *not* receive the values of, but by using the thrifty property he will learn them nonetheless; when Player 2 learns a node in this way, we say he *deduces* the value of that node.

Let q be the state Player 2 is currently on, initially $q = r$. Let u be the node queried by q. Suppose u is an internal node, and let $f_u(a_1, \ldots, a_d)$ be the variable queried by q and $v_1, \ldots, v_d$ the children of u. Since B is thrifty, $a_1, \ldots, a_d$ are the values of I on $v_1, \ldots, v_d$. Hence, for each v_i, if Player 2 does not yet know $I(v_i)$ (meaning, he did not in some earlier state *receive* or *deduce* the value of v_i) then he deduces $I(v_i) = a_i$ now. Next, Player 2 needs to decide what edge out of q to follow (Player 2 does this step for all nodes u, including leaf nodes). If for some a he learned $I(u) = a$ at an earlier state, then he again takes the edge labeled a. Otherwise, we define the next unused word in the advice string to be $I(u)$, and Player 2 uses that word now.

It is clear that for some $m \leq n$, the protocol just defined will allow Player 2 to reach the output state of I and learn at least m node values along the way, using at most m words of advice. We will argue for a stronger proposition: for some $m \leq n - p$, a string of m words suffices to allow Player 2 to reach the output state and learn at least $m+p$ nodes along the way. That will finish the proof of the lemma, since then we can use the remaining $(n-p) - m$ words of the advice string for I for the values of the $\leq n - (m+p)$ remaining nodes (ordered by some globally-fixed order on the nodes of G) that Player 2 has not yet learned. Now, by Fact 1, for every bottleneck node u of I, some parent of u is queried at some state on the path from r to the output state of I. Furthermore, if u is ever queried on the path from r till the output state, then this must happen *after* some parent of u is queried. Hence, for every bottleneck node u, Player 2 will be able to *deduce* u before he is forced to use a word of the advice to *receive* the value of u. Since the nodes whose values are deduced by Player 2 are disjoint from the nodes whose values are received by Player 2, and Player 2 uses m words by assumption, in total Player 2 learns at least $m + p$ node values[9]. □

4.2 Lower bound for Thrifty BPs using easy inputs

For much of the proof of Theorem 1, we only considered the behavior of the thrifty BP on inputs from the following set:

Definition 7 (hard inputs for thrifty programs)**.** For given dag G and $k \geq 2$, let $D_{G,k}$ be the set of $\mathrm{DE}_G(k)$ inputs I such that for every non-leaf node u, if $v_1, \ldots, v_d$ are the children of u then $f_u^I(\vec{a}) = 1$ except possibly when $\vec{a} = \langle v_1^I, \ldots, v_k^I \rangle$.

The sets $D_{G,k}$ are a small fraction of the $\mathrm{DE}_G(k)$ inputs, and it is not hard to see that separating the YES and NO instances of $D_{G,k}$ is easy for unrestricted BPs. The next result shows that the bound of Theorem 1 holds even for thrifty BPs that are only required to be correct on inputs from $D_{G,k}$.

[9]We do not say *exactly* $m + p$ because technically, according to the given protocol, if the bottleneck configuration assigned to input I has $p' > p$ pebbles (which can happen), then Player 2 learns more than $m + p$ node values.

Theorem 2. *If G has pebbling cost p then for any $k \geq 2$ if B is a thrifty deterministic BP that computes a set consistent with $DE_G(k)$ for the inputs $D_{G,k}$, then B has at least k^p states.*

Proof. In the proof of Theorem 1, the only place we used the correctness of B on inputs outside of $D_{G,k}$ is in the proof of Lemma 1. We now proof it under the weaker assumptions.

Lemma 3. *For any input $I \in D_{G,k}$ and non-leaf node u, there is at least one state q on the computation path of I that queries u, and for every such q, for each child v of u there is a state on the computation path of I before q that queries v.*

Fix $I \in D_{G,k}$. Once again we prove the lemma for I starting with the root, and then the children of the root, and so on. Let $v_1, \ldots, v_d$ be the children of u_{root}. I must visit at least one state that queries its thrifty u_{root} variable, since otherwise B would make a mistake on an input $J \in D_{G,k}$ that is identical to I except

$$f_u^J(v_1^I, \ldots, v_d^I) = \begin{cases} 2 & \text{if } u_{\text{root}}^I = 1 \\ 1 & \text{otherwise} \end{cases}$$

Now let u be any non-leaf node and suppose there is some state on the computation path of I that queries u for which the lemma does not hold. Let q be the earliest such state. So for some child v of u there is no state before q that queries v. For any $a \neq v^I$ there is an input $I_a \in D_{G,k}$ that is identical to I except $v^{I_a} = a$ and $f_u^{I_a}(v_1^I, \ldots, v_d^I) = 1$. Suppose there is no state q' before q on the computation path of I that queries u. Then I and I_a have the same computation path up to q and so I_a visits q also, which violates the thrifty assumption. Hence there is a state q' before q on the computation path of I that queries u. By our choice of q, the lemma must hold for q'. This is a contradiction since the states given by the conclusion of the lemma for q' satisfy the conclusion of the lemma for q as well.

□

4.3 Lower Bound for Semantic-incremental BPs

Theorem 3. *If there is a rooted DAG G with $n \geq 2$ nodes, indegree 2 and pebbling cost p, then for any $k \geq 2$ and $m = 3kn + n + 1$ every deterministic semantic incremental BP solving m-GEN has at least k^p states.*

Proof. Let G, n, p, k, m be as in the statement of the theorem; these are fixed throughout the proof. The bulk of this argument is a reduction from $\text{DE}_G(k)$ to m-GEN; we map each instance I of $\text{DE}_G(k)$ to an instance T^I of m-GEN such that I is a YES instance iff T^I is. Let E be the set of inputs for $\text{DE}_G(k)$. We will show that if there is a semantic-incremental m-way BP of size s that computes m-GEN correctly on the inputs $\{T^I\}_{I \in E}$, then there is a thrifty k-way BP of size at most s that computes $\text{DE}_G(k)$. Then from Theorem 1 we get $s \geq k^p$.

Fix an order on the nodes of G. We will not differentiate between a node u and its index in $[n]$ given by this order. Let $u_{\text{root}} \in [n]$ be the (index of) the root of G. We divide the elements $\{n+2, \ldots, m\}$ into two parts, one of size nk and the other of size $\leq 2nk$,[10] and refer to them by the following mnemonic names:

- $\langle u, a \rangle$ for each node u and $a \in [k]$. T^I generates this element iff $u^I = a$.
- $\langle vu, a \rangle$ for each arc vu and $a \in [k]$. T^I generates this element iff $v^I = a$.[11]

Any way of assigning those $\leq 3nk$ names to distinct elements of $\{n+2, \ldots, m\}$ will suffice, except we require that $\langle u_{\text{root}}, 1 \rangle$ gets assigned to m, since then we will have that T^I is a YES instance of m-GEN iff I is a YES instance of $\text{DE}_G(k)$. Elements $\{1, \ldots, n+1\}$ will be generated by every T^I; their purpose is technical.

Now we give the reduction. Fix an instance I of $\text{DE}_G(k)$. First we make T^I generate each of the elements $\{2, \ldots, n+1\}$. For each $u \in [n]$:

$$T^I(1, u) := u + 1 \tag{1}$$

[10] $2n$ is a bound on the number of edges since non-leaf nodes have indegree at most 2.

[11] These elements may seem redundant, given the elements $\langle u, a \rangle$. That is correct if G has the property that no two nodes u_1, u_2 have the same children.

In a similar way, we make T^I generate the elements $\langle w, l_w^I \rangle$ for each leaf w. Fix an order $w_1, \ldots, w_l$ on the leaf nodes. For each $t \in [l-1]$ and $a \in [k]$:

$$\begin{aligned} T^I(1, n+1) &:= \langle w_1, l_{w_1}^I \rangle \\ T^I(1, \langle w_t, a \rangle) &:= \left\langle w_{t+1}, l_{w_{t+1}}^I \right\rangle \end{aligned} \tag{2}$$

For every non-leaf node $u \in [n]$ and $a, b_1, b_2 \in [k]$, if v_1 and v_2 are the left and right children of u then we add the following definitions. Equations (3) simply propogate the value of a node to its out-arcs. Let $b \overset{\text{def}}{=} f_u^I(b_1, b_2)$. Equation (4) expresses: If I gives the left in-arc of u value b_1 and the right in-arc of u value b_2, then I gives u the value $b = f_u^I(b_1, b_2)$.

$$\begin{aligned} T^I(u+1, \langle v_1, a \rangle) &:= \langle v_1 u, a \rangle \\ T^I(u+1, \langle v_2, a \rangle) &:= \langle v_2 u, a \rangle \end{aligned} \tag{3}$$

$$T^I(\langle v_1 u, b_1 \rangle, \langle v_2 u, b_2 \rangle) := \langle u, b \rangle \tag{4}$$

Let us call a variable (x, y) **used** if $T^I(x, y)$ is defined at this point, and **unused** otherwise. Examining the left sides of equations (1)–(4), it is clear that the set of used variables depends only on G and k (not on I). For every unused variable (x, y) define

$$T^I(x, y) := 1 \tag{5}$$

That completes the definition of T^I. It is straightforward to show that I is a YES instance of $\mathrm{DE}_G(k)$ iff T^I is a YES instance of m-GEN.

Now we show how to convert, without increasing the size, a semantic-incremental m-way BP B that computes m-GEN correctly on the inputs $\{T^I\}_{I \in E}$, into a thrifty k-way BP that computes $\mathrm{DE}_G(k)$. From the above definition of the inputs $\{T^I\}_{I \in E}$, an m-GEN variable (x, y) is a used variable iff it has exactly one of the following four forms, where i. corresponds to Equation (1), and ii. corresponds to Equation (2), etc:

i. $(1, u)$ for some $u \in [n]$
ii. $(1, \langle w_t, a \rangle)$ for some leaf node w_t and $a \in [k]$
iii. $(u+1, \langle v, b \rangle)$ for some $b \in [k]$ and some non-leaf node $u \in [n]$ with child v
iv. $(\langle v_1 u, b_1 \rangle, \langle v_2 u, b_2 \rangle)$ for some non-leaf node u with children v_1, v_2 and $b_1, b_2 \in [k]$

We say (x, y) is a type i. variable iff it has the form of i. above, and type ii., iii., iv. variables are defined analogously. Only variables of type ii. and iv. will be translated to $\mathrm{DE}_G(k)$ variables. The remaining used variables, of types i. and iii., are "**dummy**" variables, in the sense that the right sides of the corresponding defining equations (1) and (3) do not depend on I. So for any state q in B that is labeled with a dummy variable, there is at most one edge out of q that any input T^I can take; as a consequence these states will eventually be deleted. For the same reason, states labeled with unused variables will also be deleted.

Recall the ordering on the leaf nodes $w_1, \ldots, w_l$ that we fixed earlier. Let q be a state of B that queries a variable (x, y). If (x, y) is an unused variable then delete all edges out of q except the one labeled 1. Otherwise (x, y) is one of the variable types i.–iv., and should be handled as follows:

i. $(x, y) = (1, u)$ for some $u \in [n]$. Delete every edge out of q except the one labeled $u + 1$.
ii. If $(x, y) = (1, n+1)$, then delete every edge out of q except for the k edges labeled $\langle w_1, a \rangle$ for $a \in [k]$. Otherwise $(x, y) = (1, \langle w_t, a \rangle)$ for some $t \in [l]$ and $a \in [k]$. Delete every edge out of q except for the k edges labeled $\langle w_{t+1}, b \rangle$ for $b \in [k]$.
iii. $(x, y) = (u+1, \langle v, b \rangle)$ for some $b \in [k]$ and nodes u, v such that v is a child of u. Delete every edge out of q except the one edge labeled $\langle vu, b \rangle$.
iv. $(x, y) = (\langle v_1 u, b_1 \rangle, \langle v_2 u, b_2 \rangle)$ for some non-leaf node u with children v_1, v_2 and $b_1, b_2 \in [k]$. Delete every edge out of q except the k edges labeled $\langle u, a \rangle$ for $a \in [k]$.

At this point, a non-output state q has outdegree zero or one iff it is labeled with an unused or dummy variable. For any non-output state q with outdegree zero, change q to a reject state. Now consider q with outdegree one. Note its out-edge must be labeled with an element of the same form as the right side of one of Equations (1), (3) or (5). Let q' be the unique state that q transitions to. For every state q'' with an out-edge e to q, move the target node of e from q to q'. After all the edges into q have been moved, delete q. The last step is to rename the variable labels on the remaining states. Note that only types ii., and iv. remain. Rename them as follows:

$$(1, n+1) \mapsto l_{w_1} \qquad (1, \langle w_t, a\rangle) \mapsto l_{w_{t+1}}$$

$$(\langle v_1 u, b_1\rangle, \langle v_2 u, b_2\rangle) \mapsto f_u(b_1, b_2)$$

Let B' be the resulting branching program. Since we have obtained B' only by renaming variables and deleting edges and states, clearly its size is no greater than that of B. We need that B accepts T^I iff B' accepts I for every $I \in E$. It is clear that "bypassing" and then deleting the states labeled by unused and dummy variables, in the way done above, has no effect on any input T^I. Also, one can check that for every edge e we removed before deleting the dummy states, none of the inputs T^I can take edge e. Finally, B' is thrifty precisely because B is semantic-incremental.

□

Corollary 1. *There is a constant $c > 0$ such that for infinitely-many m every deterministic semantic-incremental BP solving m-GEN has at least $2^{c\,m/\log m}$ states.*

Proof. We will need a family of rooted DAGs that are much harder to pebble than the complete binary trees. [PTC76] provides such a family:

> *There is a family of rooted DAGs $\{G_t\}_{t\geq 1}$ with $\Theta(t)$ nodes and indegree two whose black pebbling cost is $\Omega(t/\log t)$.*

Let n_t be the number of nodes in G_t[12], and let $p(n_t)$ be the pebbling cost of G_t as a function of n_t. So $p(n_t) = \Omega(n_t/\log n_t)$. We use Theorem 3 on each of the G_t with $k = 2$. For $m_t := 3kn_t + n_t + 1 = 7n_t + 1 \leq 8n_t$, for every t we get lower bounds of $2^{p(n_t)} \geq 2^{p(m_t/8)}$ for semantic incremental BPs solving m_t-GEN. This suffices since $p(m_t/8) \geq c\,m_t/\log m_t$ for some constant $c > 0$ and all t. □

5 Acknowledgements

Thanks to Steve Cook and Pierre McKenzie for many helpful comments on drafts of this paper.

References

[BCM+09] Mark Braverman, Stephen Cook, Pierre McKenzie, Rahul Santhanam, and Dustin Wehr. Fractional pebbling and thrifty branching programs. In Ravi Kannan and K Narayan Kumar, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2009)*, volume 4 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 109–120, Dagstuhl, Germany, 2009. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[GKM08] Anna Gál, Michal Koucký, and Pierre McKenzie. Incremental branching programs. *Theor. Comp. Sys.*, 43(2):159–184, 2008.

[PTC76] Wolfgang J. Paul, Robert Endre Tarjan, and James R. Celoni. Space bounds for a game on graphs. In *STOC '76: Proceedings of the eighth annual ACM symposium on Theory of computing*, pages 149–160, New York, NY, USA, 1976. ACM.

[RM99] Ran Raz and Pierre McKenzie. Separation of the monotone nc hierarchy. *Combinatorica*, 19:403–435, 1999. 10.1007/s004930050062.

[12] $n_t \geq 2$ for all $t \geq 1$, but regardless we could always just start at some $t_0 > 1$ to ensure that, without changing the result.

[Weh10] Dustin Wehr. Pebbling and branching programs solving the tree evaluation problem, 2010. arXiv:1002.4676.