# EXTREMELY UNIFORM BRANCHING PROGRAMS

Michaël Cadilhac[(A)] Andreas Krebs[(B)]
Pierre McKenzie[(C)]

[(A)]WSI at Universität Tübingen michael@cadilhac.name

[(B)]WSI at Universität Tübingen mail@krebs-net.de

[(C)]DIRO at U. de Montréal and Chaire Digiteo ENS Cachan-École Polytechnique
mckenzie@iro.umontreal.ca

***Abstract***
*We propose a new descriptive complexity notion of uniformity for branching programs solving problems defined on structured data. We observe that FO[=]-uniform (n-way) branching programs are unable to solve the tree evaluation problem studied by Cook, McKenzie, Wehr, Braverman and Santhanam [8] because such programs possess a variant of their thriftiness property. Similarly, FO[=]-uniform (n-way) branching programs are unable to solve the* P*-complete GEN problem because such programs possess the incremental property studied by Gál, Koucký and McKenzie [10].*

## 1 Introduction

A curious phenomenon arises in the quest for lower bounds on models such as boolean circuits: beyond the *Shannon effect* [17, p. 87], referring to the fact that most functions within the customary range of interest are hard but no specific hard function can be identified, one faces what we might call the *compactness effect*: within the range of interest, desirable lower bounds on a family $(C_n)_{n>0}$ of boolean circuits seem no easier to prove when a uniformity condition that somehow binds together the $C_n$'s is imposed on the family (although see [14, 4, 15, 12]).

Uniformity refers to restrictions imposed on an infinite family of combinatorial objects serving as a computing device to curb the ability of such a device to capture undecidable languages. Many different uniformity notions have been proposed since Borodin [6] and Cook [9] first characterized Turing machine-based complexity classes using logspace-uniform boolean circuit families (see Vollmer's book [16] for an extensive treatment of circuit uniformity issues or the introductory section of the conference paper [4] for a brief historical account).

[(A)]The first and second authors are supported by the Deutsche Forschungsgemeinschaft.

[(C)]The work of this paper was done during a stay of the third author at the Universität Tübingen and is supported by NSERC of Canada and Digiteo.

Branching programs are another combinatorial object that can serve to capture computation. In their nonuniform guise, polynomial size branching programs are well known to capture (nonuniform) logarithmic space. A wealth of restricted branching program variants have been studied (see [18]).

This paper deals with *uniform* branching programs. Uniform branching programs were used in statements of Barrington's theorem [1, 2]. More generally, a branching program can be viewed as a labelled graph, so that every uniformity notion invented for boolean circuits can be adapted for branching programs. But in this paper we adopt a different viewpoint.

Our main contribution is to propose a uniformity notion adapted to computational problems that involve some sort of underlying structure — specifically, array-valued variables where the set of variables itself obeys a structure. Rather than flatten the instances of such a problem by encoding them as strings (as is done in Turing machine complexity), we wish to decouple the "variable access" aspect of solving such a problem from the "computation" aspect inherent to the data itself. Our novel idea is to impose fine-grained uniformity to the variable access mechanism of the computing device.

Consider for example a computational problem such as checking if the rank of $k$ matrices $M_1, M_2, \ldots, M_k$ of size $n \times n$ and with values in $\{1, 2, \ldots, n\}$ are the same. Informally, the decoupling between *data access* and *computation* is done by allowing the program to access the entries of $M_m$ with tuples $(i, j)$. Compared to the usual binary representation one has no need to compute the position $(n^2 \cdot (m-1) + i \cdot n + j) \cdot \lceil \log n \rceil$ and evaluate the following $\log n$ bits, but can directly address the $(i, j)$-th entry of a matrix. While such operations as multiplication and addition of matrix entries may seem well within small complexity classes, the resources needed to decode the input may make the problem unwillingly harder. Technically, we implement the separation by letting inputs be mappings from an *index set* to integer values. Furthermore, these index sets are themselves broken up into two parts: an array indexing part and a structural part. In the example above, the structural part is the set of symbols $\{M_1, M_2, \ldots, M_k\}$, hence instances assign matrices to the variables. Having the index sets distinguish between variable access and array indexes allows applying uniformity constraints to only one of the two, hence enabling a trade-off in uniformity.

Specifically, we investigate the impact of applying, in our setting, the tight uniformity notions, successfully developed in the context of circuit complexity, to the branching program model. This uniformity revolves around first order logic with very weak predicates, but constrains only the array indexing part, leaving the structural part nonuniform. See Section 3 for precise definitions. By considering problems that are in part very uniform we hope to better understand the core complexity of a problem rather than the complexity that comes from the encoding of the input.

We conclude with two consequences of imposing such uniformity to $n$-way branching programs: FO[=]-uniform branching programs, even when constant symbols are allowed in the logic, can neither solve the tree evaluation problem from [8] nor the P-complete GEN problem [7, 13].

## 2 Preliminaries

For $n > 0$, we write $[n]$ for the set $\{1, 2, \ldots, n\}$. Vectors are written in bold, i.e., $\boldsymbol{s} = (s_1, s_2, \ldots)$. We will rely on first-order logic (FO) to express uniformity constraints. The use of logic is similar to [5], except that we will divide the universe in two parts: one composed of numerical values, that behaves as in the usual case, and one composed of objects (or *structural values*), for which any predicate will be made available (see Section 3 for formal definitions).

We will want the inputs of the problems to be encoded in a very natural way: even in a uniform complexity class, we still want to ensure that natural accessing operations can be performed. We thus pair problems under study with the natural structure of their inputs, in a general way.

Consider the simple problem where the input is a number $i \in [n]$ together with a linear array $\ell$ of $n$ values in $[n]$, and the goal is to check if the $i$-th entry of the array $\ell$ is equal to 1. While this is solved by most programming languages using a array access such as $\ell[i] == 1$, we suddenly need addition and multiplication to do this if we have a binary encoded input. For instance, in such an encoding where $i$ is given before $\ell$, we need to test if the bits at position:

$$\left(\sum_{k=1}^{\lceil \log n \rceil} \mathrm{BIT}(k) \cdot 2^{(k-1)}\right) + \lceil \log n \rceil, \quad \ldots, \quad \left(\sum_{k=1}^{\lceil \log n \rceil} \mathrm{BIT}(k) \cdot 2^{(k-1)}\right) + 2 \cdot \lceil \log n \rceil - 2$$

are equal to 0 and the bit at position:

$$\left(\sum_{k=1}^{\lceil \log n \rceil} \mathrm{BIT}(k) \cdot 2^{(k-1)}\right) + 2 \cdot \lceil \log n \rceil - 1$$

is equal to 1. These computations are rather obscuring the real essence of the problem. This illustrates why such a simple problem cannot be computed in FO$[<, +]$-uniform $\mathrm{AC}^0$.

Since we want to consider more difficult problems but even stricter uniformities we need a different way to encode the input. Consider the problem above again, we want our branching program to be able to access both $i$ and the entries of $\ell$ directly. Let $S = \{\texttt{int}, \texttt{lst}\}$ be a set of two objects. Then an instance corresponding to $(i, \ell)$ is an assignment from $[n] \times S$ to $[n]$, such that $(j, \texttt{lst})$ is mapped to the $j$-th element of $\ell$, and $(j, \texttt{int})$ to the value of $i$ (disregarding $j$).

Our more general framework allows for assignments from elements of $[n]^c \times S_n$, where $c = 2$ encodes matrices, for instance. Note that we also allow $S_n$ to depend on $n$, allowing to encode growing collections of objects.

**Definition 2.1 (Index sets, instances, problems).** *Given a constant $c \geq 0$ and a family of finite sets $(S_n)_{n>0}$, the sets of the form $I_n = [n]^c \times S_n$ are called* index sets. *We will refer to $[n]^c$ as the* numerical part *of $I_n$, and to $S_n$ as the* structural part. *An* instance *on $I_n$ is then a mapping from $I_n$ to $[n]$, for some $n > 0$. Finally, a* problem *is a set of instances, i.e., it is a subset of $\bigcup_{n>0} [n]^{I_n}$. Throughout the paper, we use the notations $c, S_n$, and $I_n$ with the understanding that we are referring to the defining elements of the index sets of a problem.*

**Example 2.2.** • *The sets $S_n$ of Definition 2.1 can be arbitrarily complex, and as such can be seen as finite oracles. We present three possibilities for index sets to represent languages $\subseteq \Sigma^*$, where the index sets are independent of the language represented. (1) Letting $c = 1$ and $S_n = \{\top\}$ for any $n > 0$, an instance is a function from $[n]$ to $[n]$, i.e., $S_n$ can be disregarded. For $n$ large enough, we can encode the elements of $\Sigma$ with elements of $[n]$, hence we can see an instance $A$ as a function from $[n]$ to $\Sigma$, which describes the word $a_1 a_2 \dots a_n$ such that $a_i = A(i)$. (2) Letting $c = 0$ and $S_n = \Sigma^n$ for any $n > 0$, a problem can be seen as a function from $\Sigma^+$ to $\mathbb{N}$ mapping words $w$ to a value in $[|w|]$; restricting this value to $\{1, 2\}$, such functions can be seen as characteristic functions of languages $\subseteq \Sigma^+$. (3) With $c = 0$ and $S_n$ containing the $n$-th word of $\Sigma^+$, under some ordering, a problem can be seen as a language $\subseteq \Sigma^+$ by considering that the $n$-th word of $\Sigma^+$ is in the language iff there is an instance on $I_n$ in the problem, whichever value it takes.*

- *If $c = 1$ and for every $n$, $S_n$ is of size $n$, then the index sets can be seen as so-called* structures with numbers *[11, p. 186], which are used in descriptive complexity to allow counting without imposing a linear order on $S_n$.*

- *Let $T_d^h$ be the complete $d$-ary tree of height $h$. Let $V_d^h$ be the set of vertices of $T_d^h$. Fix $h, d > 1$. The tree evaluation problem $\mathrm{TE}_d^h$ is defined with $c = d$, $S_n = V_d^h$. Given an instance $A\colon I_n \to [n]$, we define the* value of $A$ on $u \in V_d^h$, *written $u^A$, inductively by: if $u$ is a leaf, then $u^A = A((1, \dots, 1), u)$, otherwise, if $u$ has children $v_1, \dots, v_d$, then $u^A = A((v_1^A, \dots, v_d^A), u)$. Finally, $\mathrm{TE}_d^h$ consists of such instances that give the value 1 to the root of $T_d^h$.*

- *The problem GEN is defined with $c = 2$, $S_n = \{\top\}$. An instance is hence a function $([n] \times [n]) \to [n]$, for some $n$. Then such a function $f$ is in GEN iff $n$ belongs to the closure of $\{1\}$ under $f$, i.e., $n$ belongs to the smallest set $E$ containing 1 and such that $i, j \in E \Rightarrow f(i, j) \in E$. When considering GEN in the following, we will no longer specify $S_n$.*

**Definition 2.3 (Branching programs).** *A* deterministic $n$-way branching program *$B$ recognizing a problem $P \subseteq [n]^{I_n}$, for some $n > 0$, is a directed rooted multigraph whose nodes are called* states. *Every state is labeled with an element from $I_n$, called the* query *of the state, and some states are additionally labeled as accepting. Every edge is labeled with an element from $[n]$, called the* value *of the edge. Every state accessible from the initial state has no more than one outedge with a given value. An instance $A$ on $I_n$ activates, for each index $\boldsymbol{q} \in I_n$, every edge valued $A(\boldsymbol{q})$ out of every state querying $\boldsymbol{q}$. A* computation *on an instance $A$ is a directed path consisting of edges activated by $A$ that begins with the unique start state (the root) and either: (1) is infinite and contains no accepting state, (2) is finite, contains no accepting state, and ends in a state querying $\boldsymbol{q}$ with no outedge labeled $A(\boldsymbol{q})$, (3) is finite and contains an accepting state. In the first two cases, $A \notin P$, and in the third, $A \in P$. The size of $B$ is its number of states.*

# 3 Uniform Branching Programs

**Definition 3.1 ($\mathcal{L}$-uniformity for a logic $\mathcal{L}$).** *Let $(B_n)_{n>0}$ be a family of branching programs such that $B_n$ takes instances on $I_n$ as input. For a logic $\mathcal{L}$, the family is said to be $\mathcal{L}$-uniform if:*

- *there is a constant $\ell > 0$ such that for each $n > 0$, there is a labeling in $(I_n)^\ell$ of the states of $B_n$, i.e., an injective mapping from the states of $B_n$ to $(I_n)^\ell$. We will often identify a state with its label;*
- *there are two functions* init *and* fin *such that for all $n > 0$:*

$$\begin{aligned} \mathrm{init}(n) &\in (I_n)^\ell \text{ and } \mathrm{init}(n) \text{ is the initial state of } B_n \ , \\ \mathrm{fin}(n) &\subseteq (I_n)^\ell \text{ and } \mathrm{fin}(n) \text{ are the accepting states of } B_n \ ; \end{aligned}$$

- *there is a predicate $\varphi_{\mathrm{Q}}$ of $\mathcal{L}$ such that for all $n > 0$:*

$$I_n \models \varphi_{\mathrm{Q}}(\boldsymbol{q}, \boldsymbol{i}) \quad \Leftrightarrow \quad \text{state } \boldsymbol{q} \in (I_n)^\ell \text{ of } B_n \text{ queries } \boldsymbol{i} \in I_n \ ;$$

- *there is a predicate $\varphi_{\mathrm{C}}$ of $\mathcal{L}$ such that for all $n > 0$:*

$$I_n \models \varphi_{\mathrm{C}}(\boldsymbol{q}, \boldsymbol{q'}, j) \quad \Leftrightarrow \quad \exists \text{ an edge valued } j \in [n] \text{ between states } \boldsymbol{q} \text{ and } \boldsymbol{q'} \text{ of } B_n \ .$$

**Remark 3.2.** *As with $FO[<]$-uniform circuits [5], we allow the labeling to be polynomial in the size of $I_n$. Note however that the size of an uniform branching program family depends on the $S_n$'s of the index sets: in the general case, $B_n$ can have a size up to $|I_n|^\ell$, and $|S_n|$ could grow arbitrarily. In this paper, however, those sets $|S_n|$ will always be of constant size, hence in our uniform setting, branching program families will be of polynomial size w.r.t. $n$.*

It remains to specify the logics which will fully determine the uniformity conditions. The universe of our logics consist of index sets for which we disregard $c$. Hence we will have a numerical set, typically of the form $[n]$ for some $n > 0$, and a structural part, i.e., the finite sets $S_n$. We will say that a variable ranging over the former part is a *numerical* variable, and over the latter part, a *structural* variable. In our context of uniformity, we will allow any computation on the structural part of the input while imposing restrictions on the processing abilities of the numerical part:

**Definition 3.3 (**FO[pred] **logic).** *Let* pred *be a set of numerical predicates, i.e., of predicates $P$ mapping each $n > 0$ to a subset of $[n]^k$ for some $k \geq 0$, which is deemed true of a vector $\boldsymbol{x} \in [n]^k$ over the universe $[n]$ if $\boldsymbol{x} \in P(n)$. The logic* FO[pred] *consists of formulas built using the predicates from the set* pred *on the numerical part and* any predicate *on the structural part, combined with the logical connectors $\wedge, \vee, \neg$, and using two existential quantifiers, $\exists^{\mathsf{num}} x$ where $x$ ranges over the numerical part and $\exists^{\mathsf{struct}} s$ where $s$ ranges over the structural part. In particular, we write* cst *for the set of numerical constants.*

*Convention.* We will use the same name for a variable in the logic and a valuation for it, when it does not add confusion.

**Example 3.4.** *Consider $S_n$ as a set of $2^n - 1$ elements, and let $\mathsf{tree}(\cdot,\cdot)$ be a predicate on $S_n$ such that $(S_n, \mathsf{tree})$ is a complete directed tree for each $n > 0$. Let moreover* prime *be a (structural, 0-ary) predicate indicating whether $2^n - 1$ is prime. Then the formula:*

$$\varphi(i) \equiv (\exists^{\mathsf{struct}} s_1 \exists^{\mathsf{struct}} s_2 \exists^{\mathsf{struct}} s_3)[\mathsf{tree}(s_1, s_2) \wedge \mathsf{tree}(s_2, s_3)] \wedge (i = 4 \to \mathsf{prime})$$

*is an* FO[=, cst]*formula, as the only numerical predicates used are $=$ and the constant 4, such that $I_n \models \varphi(i)$ is true if $n \geq 3$ (as $s_1, s_2, s_3$ describe a path of length 2 in the complete tree) and, in case $i = 4$, $2^n - 1$ is prime.*

# 4 Intrinsic Thriftiness of FO[=]-Uniform detBPs

In this section, we present general properties of FO[=]-uniform detBP families, that closely resemble the *incremental* and *thriftiness* properties of [10] and [8], respectively. The underlying common idea is that whenever a query is made on a given index (i.e., an element of an index set) in a computation, then the numerical values in the index have been obtained as the result of a query earlier in the computation:

**Definition 4.1.** *A branching program $B$ taking inputs from $I_n$ is said to be* thrifty modulo *$M$ for a set $M \subseteq [n]$ if for every state querying $((i_1, \ldots, i_c), s)$ and all paths (consistent or inconsistent) from the initial state to that state, it holds that each of $i_1, \ldots, i_c$ appears as the value of an edge in the path (we also say that* the value has been discovered earlier*), or is in $M$. We say that $B$ is* thrifty *if it is thrifty modulo $\{1\}$.*

**Remark 4.2.** *In the context of [10] and [8], both* incrementality *and* thriftiness *were defined relative to a specific problem. For the GEN problem, which is the focus of [10], our notion and that of incrementality match. In the tree evaluation problem, the focus of [8], there is a slight discrepancy: the restriction of [8] is that for any instance $A$, any query to $A(\boldsymbol{v}, u)$ is such that $\boldsymbol{v}$ is the correct vector of values for the children of node $u$; our notion allows queries of a node function on values that are not that of its children. It is however not hard to apply the main results of [8] in our context, as the proof of the forthcoming Theorem 5.3 shows.*

**Lemma 4.3.** *Let $\varphi(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{s})$ be an* FO[=] *formula where $\boldsymbol{x}, \boldsymbol{y}$ are tuples of numerical variables and $\boldsymbol{s}$ consists of structural variables. For any large enough $n > 0$, if for some values for $\boldsymbol{x}, \boldsymbol{s}$ there is a unique value for $\boldsymbol{y}$ such that $I_n \models \varphi(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{s})$, then the value of each element of $\boldsymbol{y}$ is one of those in $\boldsymbol{x}$.*

*Proof.* Using the notation of the lemma, suppose for a contradiction that there is an $i$ such that $y_i$ is not one of the $x_j$'s.

Now let $m \in [n]$ be a value different from any $x_j$'s or $y_j$'s. Such a value exists if $n$ is large enough. Let $\boldsymbol{y'}$ be equal to $\boldsymbol{y}$ except at positions $j$ with the same value as $y_i$, where it is set to $m$.

It is readily seen that there is no winning strategy for Spoiler in the EF-game for FO[=] on $\boldsymbol{y}$ and $\boldsymbol{y'}$, hence $I_n \models \varphi(\boldsymbol{x}, \boldsymbol{y'}, \boldsymbol{s})$ also, a contradiction. □

**Corollary 4.4.** *If $(B_n)_{n>0}$ is a* FO[=]*-uniform family of detBPs, then for $n > 0$ large enough:*

- $I_n \models \varphi_{\mathrm{Q}}(\boldsymbol{q}, \boldsymbol{i})$ *implies that the numerical part of* $\boldsymbol{i}$ *appears in* $\boldsymbol{q}$*;*
- $I_n \models \varphi_{\mathrm{C}}(\boldsymbol{q}, \boldsymbol{q'}, j)$ *implies that each numerical component of* $\boldsymbol{q'}$ *either appears in* $\boldsymbol{q}$ *or is* $j$*.*

**Proposition 4.5.** *Let $(B_n)_{n>0}$ be an* FO[=]*-uniform family of deterministic branching programs. For $n > 0$ large enough, $B_n$ is thrifty modulo the label of its initial state.*

*Proof.* Let $n > 0$ be large enough for Corollary 4.4 to apply. For a state $\boldsymbol{q}$ of $B_n$, we define $\mathrm{dist}(\boldsymbol{q})$ to be the length of the longest path in $B_n$ from the initial state to $\boldsymbol{q}$, or $\perp$ if none exists. We show by induction on $\mathrm{dist}(\boldsymbol{q})$ that for any path from the initial state to $\boldsymbol{q}$ in $B_n$, if $L$ is the set of labels on the edges of the path, it holds that $(\forall i)[q_i \in [n] \to q_i \in L \cup \mathrm{init}(n)]$.

*Base case.* If $\mathrm{dist}(\boldsymbol{q}) = 0$, then $\boldsymbol{q}$ is the initial state, labeled $\mathrm{init}(n)$.

*Induction step.* Suppose $\mathrm{dist}(\boldsymbol{q}) = m > 0$, and consider a path from the initial state of $B_n$ to $\boldsymbol{q}$, whose second to last state is $\boldsymbol{q'}$. Note that $\mathrm{dist}(\boldsymbol{q'})$ is at most $m - 1$, hence the induction hypothesis applies to it. In particular, the components of $\boldsymbol{q'}$ are values of the edges of the path under consideration or in $\mathrm{init}(n)$. Now we have that $I_n \models \varphi_{\mathrm{C}}(\boldsymbol{q'}, \boldsymbol{q}, j)$ where $j$ is the value of the last edge of the path, and by Corollary 4.4, this implies that each numerical component of $\boldsymbol{q}$ appears in $\boldsymbol{q'}$ or is $j$. Hence for each $i$, $q_i$ appears on a value of an edge of the path, or in $\mathrm{init}(n)$. This concludes the induction step.

We conclude that $B_n$ is thrifty modulo $\mathrm{init}(n)$. For any state $\boldsymbol{q}$ accessible from the initial state, the query at $\boldsymbol{q}$ is done, by Corollary 4.4, at indexes appearing in $\boldsymbol{q}$. By the previous fact, these indexes are those appearing on every path from the initial state to $\boldsymbol{q}$, hence they have been discovered earlier. □

Moreover, it is easily seen that constants in the uniformity language do not extend the expressiveness of detBPs:

**Proposition 4.6.** *For any* FO[=, cst]*-uniform family of detBPs, there exists an* FO[=]*-uniform family of detBPs for the same problem.*

*Proof.* Let $(B_n)_{n>0}$ be an FO[=, cst]-uniform family of detBPs. Let $C$ be the (ordered) set of constants appearing in the defining formulas $\varphi_{\mathrm{Q}}$ and $\varphi_{\mathrm{C}}$ of $(B_n)_{n>0}$. Extend the state labels of the $B_n$'s by $|C|$ additional copies of $I_n$, of which we only use the very first component. Rewrite the formulas $\varphi_{\mathrm{Q}}$ and $\varphi_{\mathrm{C}}$ into $\psi_{\mathrm{Q}}$ and $\psi_{\mathrm{C}}$, respectively, in such a way that instead of relying on the constants of $C$, they use the extra components. Modify the init function of the family into a function $\mathrm{init}'$ mapping $n \in \mathbb{N}$ to a tuple of $(I_n)^{\ell+|C|}$ consisting of the concatenation of the vector $\mathrm{init}(n)$ and the values for $C$ with universe $[n]$. Do likewise for the function fin, ending up with the function $\mathrm{fin}'$. The detBP family described by $\mathrm{init}', \mathrm{fin}', \psi_{\mathrm{Q}}, \psi_{\mathrm{C}}$ is an FO[=]-uniform family of detBPs for the problem decided by $(B_n)_{n>0}$, concluding the proof. □

The following technical proposition gives sufficient conditions for a detBP family which is thrifty modulo to be made thrifty. This applies to problems such as GEN and $\mathrm{TE}^h_d$ for which a constant number of numerical values can be skipped, i.e., for which a constant number of entries

in the input can be supposed to be known beforehand, without impacting the complexity of the problem:

**Proposition 4.7.** *Let $(B_n)_{n>0}$ be a detBP family for a problem $P$ where the index sets $I_n = [n]^c \times S_n$ are such that the $S_n$'s are constant. Suppose that for all $n > 0$, $B_n$ is thrifty modulo some set $M_n \cup \{1\}$, where $1 \notin M_n$ and the size of $M_n$ is a constant $m$ independent of $n$. Let $\mathsf{up}_n \colon [n-m] \to [n]$ be a family of injective mappings, which we naturally extend to tuples, i.e., $\mathsf{up}_n(i_1, i_2, \ldots, i_c) = (\mathsf{up}_n(i_1), \mathsf{up}_n(i_2), \ldots, \mathsf{up}_n(i_c))$. For an instance $A$ on $I_{n-m}$, define the following instance on $I_n$:*

$$\mathsf{up}_n(A) \colon (\boldsymbol{i}, s) \mapsto \begin{cases} 1 & \text{if } \boldsymbol{i} \text{ contains elements of } M_n \ , \\ 1 & \text{if } \mathsf{up}_n^{-1}(\boldsymbol{i}) \text{ is undefined,} \\ \mathsf{up}_n(A(\mathsf{up}_n^{-1}(\boldsymbol{i}), s)) & \text{otherwise.} \end{cases}$$

*If the $\mathsf{up}_n$ family is such that $A \in P \Leftrightarrow \mathsf{up}(A) \in P$, then there is a family $(B'_n)_{n>0}$ of thrifty detBPs for $P$ where $B'_n$ is of the size of $B_{n+m}$.*

*Proof.* Fix an $n > m$. As the family solves $P$, in particular $B_n$ decides $\mathsf{up}(A) \in P$ for every instance $A$ on $I_{n-m}$. In all the computations of $B_n$ on instances of the form $\mathsf{up}(A)$, any query pertaining to an element in $M_n$ will be valued 1. Hence define $B'_{n-m}$ as the branching program $B_n$ where:

- For any state querying an index $(\boldsymbol{i}, s)$ where (1) $\boldsymbol{i}$ contains an element of $M_n$ or (2) $\mathsf{up}_n^{-1}(\boldsymbol{i})$ is undefined, the query is replaced by $((1, \ldots, 1), s)$, and all outedges of the state are rerouted to the destination of the one valued 1;
- All the other queries to indexes $(\boldsymbol{i}, s)$ are changed to queries to indexes $(\mathsf{up}_n^{-1}(\boldsymbol{i}), s)$;
- All edge values $j$ are changed to $\mathsf{up}^{-1}(j)$ if it is defined, or removed if it is not;
- The initial and final states are kept the same.

Clearly, $B'_{n-m}$ solves $P$ for all instances $A$ on $I_{n-m}$. Moreover, $B_n$ being thrifty modulo $M_n$, $B'_{n-m}$ is thrifty. □

# 5 Consequences of FO[=]-Uniformity

Recall the definition of GEN from Example 2.2. When the $n \times n$ table describing GEN instances is restricted to have a single nontrivial row (i.e., a row containing non-one entries), GEN is known to be complete for Logspace [3]. Let GEN-1 be the problem specified by $c = 1, S_n = \{\top\}$ and containing instances $A \colon [n] \to [n]$ for which there is a $k$ such that $A^k(1) = n$. Then:

**Theorem 5.1.** *GEN-1 is in* FO[=]*-uniform detBP.*

*Proof.* In GEN-1, it suffices to remember the latest value produced so far, hence the FO[=]-uniform detBP family specified by:

- $\ell = 1, \mathrm{init}(k) = 1, \mathrm{fin}(k) = \{k\}$;

- $\varphi_{\mathrm{Q}}(q,i) \equiv (q = i)$;
- $\varphi_{\mathrm{C}}(q,q',i) \equiv (q' = i)$;

solves GEN-1. □

**Theorem 5.2.** *There is no* FO[=, cst]*-uniform family of detBPs for GEN.*

*Proof.* For a contradiction, suppose that there exists an FO[=, cst]-uniform family of detBPs for GEN. By Proposition 4.6, there is such a family $(B_n)_{n>0}$ which is FO[=]-uniform.

Let $n > 0$ be large enough, so that, by Proposition 4.5, $B_n$ is thrifty modulo the set $M_n \cup \{1\}$ of values in the label $\mathrm{init}(n)$ of its initial state — we suppose that $1 \notin M_n$. Note that $|M_n| \leq c$, and we may, for $n$ large enough, add to $M_n$ values different from 1 so that $|M_n| = c$. Order $M_n$ as $m_1 < m_2 < \ldots < m_c$. For $i < n - c$, let $\mathsf{up}_n(i)$ be $i + \max\{j \mid m_j \leq i\}$, where $\max \emptyset = 0$. Additionally, we set $\mathsf{up}_n(n-c) = n$. For an instance $A$ on $I_{n-c}$, that is, an array of size $(n-c) \times (n-c)$, and with the notation of Proposition 4.7, it holds that $A$ is in GEN iff $\mathsf{up}(A)$ is in GEN, as there is no element of $M_n \setminus \{n\}$ in $\mathsf{up}(A)$.

By Proposition 4.7, there is a thrifty polysize family of detBPs for GEN. But this contradicts the fact that incremental (i.e., thrifty in our terminology) branching programs require exponential size to solve GEN [10, Cor. 4.12]. □

Recall the definition of $\mathrm{TE}_2^h$ from Example 2.2.

**Theorem 5.3.** *For $h$ big enough, there is no* FO[=, cst]*-uniform family of detBPs for* $\mathrm{TE}_2^h$.

*Proof.* For a contradiction, suppose that there exists an FO[=, cst]-uniform family of detBPs for $\mathrm{TE}_2^h$, for some large $h$. By Proposition 4.6, there is such a family $(B_n)_{n>0}$ which is FO[=]-uniform. We now proceed as in Theorem 5.2 in order to apply Proposition 4.7. By Proposition 4.5, for $n$ big enough, $B_n$ is thrifty modulo the set $M_n \cup \{1\}$ of values in the label $\mathrm{init}(n)$ of its initial state — we again suppose that $1 \notin M_n$. Note that $|M_n| \leq c$, and we may, for $n$ large enough, add values different than 1 into $M_n$ so that $|M_n| = c$. Order $M_n$ as $m_1 < m_2 < \ldots < m_c$. For $i \leq n - c$, let $\mathsf{up}_n(i)$ be $i + \max\{j \mid m_j \leq i\}$, where $\max \emptyset = 0$. With the notation of Proposition 4.7, it holds that $A$ is in $\mathrm{TE}_2^h$ iff $\mathsf{up}(A)$ is in $\mathrm{TE}_2^h$, as there is no element of $M_n$ in $\mathsf{up}(A)$.

Our goal is to rely on a lower bound of [8]; to do so, we must convert a thrifty detBP family to a family that has the stronger thriftiness property of [8]: a branching program may only query $A(\boldsymbol{v}, u)$ if $\boldsymbol{v}$ is the correct vector of values for the children of node $u$.

Let $N = 2^h - 1$, and fix an $n > 0$. We adapt the proof of Theorem 5 of [8] and show that there exists a thrifty (in the sense of [8]) detBP family of size $O(((Nn)^c \times 2^h)^\ell)$ for $\mathrm{TE}_2^h$, contradicting the lower bound of $n^h$ of [8, Theorem 33], for $n, h$ big enough.

Let $A$ be an instance from $I_n$ to $[n]$. We design from $A$ an instance $A'$ from $I_{(Nn+2)}$ to $[Nn+2]$ that encodes the tree structure $T_2^h$ into the index sets. To do so, let us view $[Nn+2]$ as the set $\{1\} \cup \{\langle i, u\rangle \mid i \in [n] \wedge u \in V_2^h\} \cup \{\bot\}$; recall that $V_2^h$ is the set of vertices of the full binary

tree $T_2^h$. Now for $u \in V_2^h$ and $i_1, i_2 \in [Nn+2]$ interpreted as previously, define:

$$A'((i_1,i_2),u) = \begin{cases} \langle A((j_1,j_2),u),u\rangle & \text{if } i_1 = \langle j_1, v_1\rangle \wedge i_2 = \langle j_2, v_2\rangle \wedge v_1, v_2 \text{ are the children of } u \\ \langle A((1,1),u),u\rangle & \text{if } i_1 = i_2 = 1 \wedge u \text{ is a leaf} \\ \bot & \text{otherwise.} \end{cases}$$

Now $B_{Nn+2}$ solves such instances $A'$, and thriftiness indicates that the function is only queried on values discovered earlier. Hence, each a query is either (1) about a leaf at position $(1,1)$, or (2) about an inner function on the value of $A$ on the two children of a node, or (3) about unrelated indexes, in which case the answer is 1. We now construct a detBP $B'_n$ from $B_{Nn+2}$. First, we hardwire the queries of type 3 to the 1 output, i.e., we change the query to a dummy leaf query at position $(1,1)$, and branch on all output values to the state in $B_{Nn+2}$ corresponding to the output 1. Second, if a state queries a leaf at position $(1,1)$, then it does the same in $B'_n$. Thirdly, a query to $((\langle j_1, v_1\rangle, \langle j_2, v_2\rangle), u)$ is replaced by a query to $(j_1, j_2, u)$. Finally, edge values of the form $\langle j, u\rangle$ are replaced by $j$.

The family $(B'_n)_{n>0}$ thus obtained is a nonuniform thrifty detBP family in the sense of [8] for $\mathrm{TE}_2^h(k/N)$ of size smaller than $h^k$. □

# 6 Discussion

We have proposed a new notion of uniformity that allows imposing fine-grained restrictions on the data access components of a computing device such as a family of branching programs.

This uniformity can be relaxed by the addition of more predicates to the logic. The framework thus provides a way to gradually relax uniformity. For example, we have shown here that FO[=]-uniform branching programs solving the tree evaluation problem are "thrifty" in the sense of [8]. What properties would less uniform branching programs for the tree evaluation possess? Would these pose more incisive lower bound challenges than thriftiness?

More generally, we have observed that FO[=]-uniform deterministic branching programs are weaker than the thrifty or incremental detBPs from the literature. Those rather immediate consequences of our definitions were merely intended as proofs of concept. But it seems reasonable to expect further applications of this framework. For example, would applying it to circuit complexity raise a need for manageable new circuit complexity arguments?

# References

[1] D. A. M. BARRINGTON, Bounded-width polynomial size branching programs recognize exactly those languages in $\mathrm{NC}^1$. *Journal of Computer and System Sciences* 38 (1989), 150–164.

[2] D. A. M. BARRINGTON, N. IMMERMAN, H. STRAUBING, On uniformity within $\mathrm{NC}^1$. *Journal of Computer and System Sciences* 41 (1990) 3, 274–306.

[3] D. A. M. BARRINGTON, P. MCKENZIE, Oracle branching programs and Logspace versus P. *Inf. Comput.* 95 (1991) 1, 96–115.

[4] C. BEHLE, A. KREBS, K.-J. LANGE, P. MCKENZIE, The Lower Reaches of Circuit Uniformity. In: *MFCS*. 2012, 590–602.

[5] C. BEHLE, K.-J. LANGE, FO[<]-Uniformity. In: *Proc. 21st Annual IEEE Conference on Computational Complexity (CCC'06)*. 2006, 183 – 189.

[6] A. BORODIN, On relating time and space to size and depth. *SIAM Journal on Computing* 6 (1977), 733–744.

[7] S. COOK, An Observation on Time-Storage Trade Off. *J. Comput. Syst. Sci.* 9 (1974) 3, 308–316.

[8] S. COOK, P. MCKENZIE, D. WEHR, M. BRAVERMAN, R. SANTHANAM, Pebbles and Branching Programs for Tree Evaluation. *ACM TOCT* 3 (2012) 2, 4.

[9] S. A. COOK, Deterministic CFL's are accepted simultaneously in polynomial time and log squared space. In: *Proceedings 11th Theory of Computing*. ACM Press, 1979, 338–345.

[10] A. GÁL, M. KOUCKÝ, P. MCKENZIE, Incremental Branching Programs. *Theory Comput. Syst.* 43 (2008) 2, 159–184.

[11] N. IMMERMAN, *Descriptive Complexity*. Springer, 1999.

[12] H. JAHANJOU, E. MILES, E. VIOLA, Succinct and explicit circuits for sorting and connectivity. *Electronic Colloquium on Computational Complexity (ECCC)* 21 (2014), 37.

[13] N. D. JONES, W. T. LAASER, Complete problems for deterministic polynomial time. *Theoretical Computer Science* 3 (1976), 105–117.

[14] K. LUOSTO, Equicardinality on Linear Orders. In: *19th IEEE Symp. on Logic in Computer Science*. 2004, 458–465.

[15] R. SANTHANAM, R. WILLIAMS, On Medium-Uniformity and Circuit Lower Bounds. In: *IEEE Conference on Computational Complexity*. 2013, 15–23.

[16] H. VOLLMER, *Introduction to Circuit Complexity – A Uniform Approach*. Texts in Theoretical Computer Science, Springer Verlag, 1999.

[17] I. WEGENER, *The Complexity of Boolean Functions*. Wiley-Teubner series in computer science, B. G. Teubner & John Wiley, Stuttgart, 1987.

[18] I. WEGENER, *Branching Programs and Binary Decision Diagrams*. SIAM Monographs on Discrete Mathematics and Applications, SIAM, Philadelphia, 2000.