Extremely Uniform Branching Programs

Logic-based uniformity in structured problems

2nd Highlights of Logic, Games and Automata

M. Cadilhac, A. Krebs, and P. McKenzie

September 4th, 2014
Paris, France
Extremely Uniform Branching Programs

Logic-based uniformity in structured problems

2nd Highlights of Logic, Games and Automata

M. Cadilhac, A. Krebs, and P. McKenzie

Université de Montréal

September 4th, 2014
Paris, France
Extremely Uniform Branching Programs

Logic-based uniformity in structured problems

2nd Highlights of Logic, Games and Automata

M. Cadilhac, A. Krebs, and P. McKenzie

September 4th, 2014
Paris, France
Extremely Uniform Branching Programs

Logic-based uniformity in structured problems

2nd Highlights of Logic, Games and Automata

M. Cadilhac, A. Krebs, and P. McKenzie

September 4th, 2014

Paris, France
Extremely Uniform Branching Programs

Logic-based uniformity in structured problems

2nd Highlights of Logic, Games and Automata

M. Cadilhac, A. Krebs, and P. McKenzie

Université de Montréal

September 4th, 2014

Paris, France
Extremely Uniform Branching Programs

Logic-based uniformity in structured problems

2nd Highlights of Logic, Games and Automata

M. Cadilhac, A. Krebs, and P. McKenzie

September 4th, 2014
Paris, France
Extremely Uniform Branching Programs

Logic-based uniformity in structured problems

2nd Highlights of Logic, Games and Automata

M. Cadilhac, A. Krebs, and P. McKenzie

Université de Montréal

September 4th, 2014
Paris, France
Extremely Uniform Branching Programs

Logic-based uniformity in structured problems

2nd Highlights of Logic, Games and Automata

M. Cadilhac, A. Krebs, and P. McKenzie

September 4th, 2014
Paris, France
Study of low complexity classes:
   ▶ Models: Boolean circuits,
Study of low complexity classes:

- Models: Boolean circuits,

\[ C_n \]

Nondecidable languages, hard to show general results.

Uniformity: specify \( n \) for different input sizes.

Lead to natural characterizations:

- \( L \)-unif. polysize circuits = \( P \)
- \( \text{ALOGTIME-unif. NC}^1 = \text{DLOGTIME-unif. NC}^1 = \text{ALOGTIME} \)
- \( \text{DLOGTIME-unif. AC}^0 = \text{FO}^{+,\times} \)

Going below \( \text{DLOGTIME} \) (prompted by [Roy and Straubing, 2007, Behle and Lange, 2006]):
Study of low complexity classes:

- Models: Boolean circuits, branching programs
Study of low complexity classes:

- Models: Boolean circuits, branching programs
Context

Study of low complexity classes:

- Models: Boolean circuits, branching programs

![Branching Program Diagram](image-url)
Context

Study of low complexity classes:

- Models: Boolean circuits, branching programs
- Family $(C_n)_{n>0}$ for different input sizes
Context

Study of low complexity classes:

- Models: Boolean circuits, branching programs
- Family \((C_n)_{n \geq 0}\) for different input sizes
  - Nondecidable languages,
Context

Study of low complexity classes:

- Models: Boolean circuits, branching programs
- Family \((C_n)_{n>0}\) for different input sizes
  - Nondecidable languages,
    \[ C_n \text{ accepts } \Sigma^n \text{ iff } n \in \text{HALT} \]
Context

Study of low complexity classes:

- Models: Boolean circuits, branching programs
- Family $(C_n)_{n>0}$ for different input sizes
  - Nondecidable languages,
Context

Study of low complexity classes:
- Models: Boolean circuits, branching programs
- Family $(C_n)_{n>0}$ for different input sizes
  - Nondecidable languages, hard to show general results
Context

Study of low complexity classes:

- Models: Boolean circuits, branching programs
- Family \((C_n)_{n>0}\) for different input sizes
  - Nondecidable languages, hard to show general results

Best to date: \(\mathsf{NEXP} \neq \mathsf{ACC}^0\)

[Williams, 11]
Context

Study of low complexity classes:

- Models: Boolean circuits, branching programs
- Family \((C_n)_{n>0}\) for different input sizes
  - Nondecidable languages, hard to show general results
Context

Study of low complexity classes:

- Models: Boolean circuits, branching programs
- Family \((C_n)_{n>0}\) for different input sizes
  - Nondecidable languages, hard to show general results
- Uniformity: specify \(n \mapsto C_n\)
Study of low complexity classes:

- Models: Boolean circuits, branching programs
- Family \((C_n)_{n>0}\) for different input sizes
  - Nondecidable languages, hard to show general results
- Uniformity: specify \(n \mapsto C_n\)
Context

Study of low complexity classes:
  - Models: Boolean circuits, branching programs
  - Family \((C_n)_{n>0}\) for different input sizes
    - Nondecidable languages, hard to show general results
  - Uniformity: specify \(n \mapsto C_n\)

Lead to natural characterizations:
Context

Study of low complexity classes:

- Models: Boolean circuits, branching programs
- Family \((C_n)_{n>0}\) for different input sizes
  - Nondecidable languages, hard to show general results
- Uniformity: specify \(n \mapsto C_n\)

Lead to natural characterizations:

- L-unif. polysize circuits = \(P\)
Context

Study of low complexity classes:

- Models: Boolean circuits, branching programs
- Family \((C_n)_{n>0}\) for different input sizes
  - Nondecidable languages, hard to show general results
- Uniformity: specify \(n \mapsto C_n\)

Lead to natural characterizations:

- \(L\)-unif. polysize circuits = \(P\)
- \(ALOGTIME\)-unif. \(NC^1\)
Context

Study of low complexity classes:

- Models: Boolean circuits, branching programs
- Family \( (C_n)_{n>0} \) for different input sizes
  - Nondecidable languages, hard to show general results
- Uniformity: specify \( n \mapsto C_n \)

Lead to natural characterizations:

- L-unif. polysize circuits = P
- ALOGTIME-unif. \( NC^1 \)

\[ \log n \downarrow \text{depth} \]
Context

Study of low complexity classes:

- Models: Boolean circuits, branching programs
- Family \((C_n)_{n>0}\) for different input sizes
  - Nondecidable languages, hard to show general results
- Uniformity: specify \(n \mapsto C_n\)

Lead to natural characterizations:

- \(L\text{-unif. polysize circuits} = P\)
- \(ALOGTIME\text{-unif. NC}^1\)
Context

Study of low complexity classes:

- Models: Boolean circuits, branching programs
- Family \((C_n)_{n > 0}\) for different input sizes
  - Nondecidable languages, hard to show general results
- Uniformity: specify \(n \mapsto C_n\)

Lead to natural characterizations:

- L-unif. polysize circuits = P
- ALOGTIME-unif. \(\text{NC}^1\) = DLOGTIME-unif. \(\text{NC}^1\) = ALOGTIME
Context

Study of low complexity classes:

- Models: Boolean circuits, branching programs
- Family \((C_n)_{n>0}\) for different input sizes
  - Nondecidable languages, hard to show general results
- Uniformity: specify \(n \mapsto C_n\)

Lead to natural characterizations:

- L-unif. polysize circuits = P
- ALOGTIME-unif. NC\(^1\) = DLOGTIME-unif. NC\(^1\) = ALOGTIME
- DLOGTIME-unif. AC\(^0\)
Context

Study of low complexity classes:

- Models: Boolean circuits, branching programs
- Family \((C_n)_{n>0}\) for different input sizes
  - Nondecidable languages, hard to show general results
- Uniformity: specify \(n \mapsto C_n\)

Lead to natural characterizations:

- \(L\)-unif. polysize circuits = \(P\)
- \(ALOGTIME\)-unif. \(NC^1 = DLOGTIME\)-unif. \(NC^1 = ALOGTIME\)
- \(DLOGTIME\)-unif. \(AC^0\)
Context

Study of low complexity classes:

- Models: Boolean circuits, branching programs
- Family \((C_n)_{n>0}\) for different input sizes
  - Nondecidable languages, hard to show general results
- Uniformity: specify \(n \mapsto C_n\)

Lead to natural characterizations:

- L-unif. polysize circuits = P
- ALOGTIME-unif. NC¹ = DLOGTIME-unif. NC¹ = ALOGTIME
- DLOGTIME-unif. AC⁰
Context

Study of low complexity classes:

- Models: Boolean circuits, branching programs
- Family \((C_n)_{n>0}\) for different input sizes
  - Nondecidable languages, hard to show general results
- Uniformity: specify \(n \mapsto C_n\)

Lead to natural characterizations:

- L-unif. polysize circuits = P
- ALOGTIME-unif. \(NC^1 = DLOGTIME\)-unif. \(NC^1 = ALOGTIME\)
- DLOGTIME-unif. \(AC^0 = FO[+, \times]\)
Context

Study of low complexity classes:

- Models: Boolean circuits, branching programs
- Family \((C_n)_{n>0}\) for different input sizes
  - Nondecidable languages, hard to show general results
- Uniformity: specify \(n \mapsto C_n\)

Lead to natural characterizations:

- \(L\)-unif. polysize circuits = \(P\)
- \(ALOGTIME\)-unif. \(NC^1 = DLOGTIME\)-unif. \(NC^1 = ALOGTIME\)
- \(DLOGTIME\)-unif. \(AC^0 = FO[+, \times]\)

Going below DLOGTIME (prompted by [Roy and Straubing, 2007, Behle and Lange, 2006]): logics
Our contributions

Get rid of circuit encoding, by means of logic-based uniformity

- Allows for logics that cannot decode
Our contributions

Get rid of circuit encoding, by means of logic-based uniformity

- Allows for logics that cannot decode
- Allows to consider structured inputs (more natural than words) without requiring more uniformity power
Our contributions

Get rid of circuit encoding, by means of logic-based uniformity

- Allows for logics that cannot decode
- Allows to consider structured inputs (more natural than words) without requiring more uniformity power
- Depending on input structure, balance uniformity
Our contributions

Get rid of circuit encoding, by means of logic-based uniformity

- Allows for logics that cannot decode
- Allows to consider structured inputs (more natural than words) without requiring more uniformity power
- Depending on input structure, balance uniformity

An application with a weak uniformity logic
Outline

Presenting inputs

Uniform branching programs

$\text{FO}[=, \text{cst}]-\text{uniform branching programs}$

Further research
Outline

Presenting inputs

Uniform branching programs

FO[=, cst]-uniform branching programs

Further research
A proposal for structured inputs

Definition (Structured problem)

- Index set: $I_n = [n]^c \times S_n$
- Instance: mapping $A: I_n \rightarrow [n]$ for some $n$
A proposal for structured inputs

Definition (Structured problem)

- Index set: \( I_n = [n]^c \times S_n \)
- Instance: mapping \( A: I_n \to [n] \) for some \( n \)
A proposal for structured inputs

Definition (Structured problem)

- Index set: \( I_n = [n]^c \times S_n \)
- Instance: mapping \( A: I_n \rightarrow [n] \) for some \( n \)
A proposal for structured inputs

Definition (Structured problem)

- Index set: $I_n = [n]^c \times S_n$
- Instance: mapping $A: I_n \to [n]$ for some $n$
- Problem: set of instances
A proposal for structured inputs

Definition (Structured problem)

- Index set: \( I_n = [n]^c \times S_n \)
- Instance: mapping \( A: I_n \rightarrow [n] \) for some \( n \)
- Problem: set of instances

Examples:
- \( S_n \) = set of pointers,
- \( A(i, v) = \ast(v + i) \)
- \( S_n = [k] \), \( c = 1 \), encodes \( k \)-bounded-degree graphs
- More generally, \( S_n \) can represent a parameter.
A proposal for structured inputs

Definition (Structured problem)

- Index set: $I_n = [n]^c \times S_n$
- Instance: mapping $A: I_n \rightarrow [n]$ for some $n$
- Problem: set of instances

Examples:

- $S_n = \text{set of pointers, } A(i, v) = *(v + i)$
A proposal for structured inputs

**Definition (Structured problem)**

- Index set: \( I_n = [n]^c \times S_n \)
- Instance: mapping \( A: I_n \rightarrow [n] \) for some \( n \)
- Problem: set of instances

**Examples:**

- \( S_n = \) set of pointers, \( A(i, v) = * (v + i) \)
- \( S_n = [k], c = 1, \) encodes \( k \)-bounded-degree graphs
  \( A(s, d) = d \)-th neighbor of \( s \)
A proposal for structured inputs

Definition (Structured problem)

- Index set: \( I_n = [n]^c \times S_n \)
- Instance: mapping \( A: I_n \rightarrow [n] \) for some \( n \)
- Problem: set of instances

Examples:

- \( S_n = \) set of pointers, \( A(i, v) = \ast (v + i) \)
- \( S_n = [k], c = 1, \) encodes \( k \)-bounded-degree graphs
  \( A(s, d) = d \)-th neighbor of \( s \)
A proposal for structured inputs

Definition (Structured problem)

- Index set: \( I_n = [n]^c \times S_n \)
- Instance: mapping \( A: I_n \rightarrow [n] \) for some \( n \)
- Problem: set of instances

Examples:

- \( S_n = \) set of pointers, \( A(i, v) = *(v + i) \)
- \( S_n = [k], c = 1, \) encodes \( k \)-bounded-degree graphs
  \( A(s, d) = d \)-th neighbor of \( s \)
A proposal for structured inputs

Definition (Structured problem)

- Index set: \( I_n = [n]^c \times S_n \)
- Instance: mapping \( A: I_n \to [n] \) for some \( n \)
- Problem: set of instances

Examples:

- \( S_n = \) set of pointers, \( A(i, v) = * (v + i) \)
- \( S_n = [k], c = 1, \) encodes \( k \)-bounded-degree graphs \( A(s, d) = d \)-th neighbor of \( s \)
- More generally, \( S_n \) can represent a parameter
Outline

Presenting inputs

Uniform branching programs

FO[=, cst]-uniform branching programs

Further research
A take on logical uniformity

- Usual strategy: $C_n \mapsto L_{C_n} \subseteq \Sigma^*$ then descr. cpx. on strings

$$\exists x (Q_a x \land \forall y (x < y \rightarrow Q_b y))$$
A take on logical uniformity

- Usual strategy: \( C_n \mapsto L_{C_n} \subseteq \Sigma^* \) then descr. cpx. on strings
- Here: remove the need for the logic to decode

\[ \exists x \left( Q_a x \land \forall y \left( x < y \rightarrow Q_b y \right) \right) \]
A take on logical uniformity

- Usual strategy: \( C_n \mapsto L_{C_n} \subseteq \Sigma^* \) then descr. cpx. on strings \( \exists x \left( Q_a x \land \forall y \left( x \leq y \rightarrow Q_b y \right) \right) \)
- Here: remove the need for the logic to decode

Definition \((L\text{-uniform branching program})\)

Family \((B_n)_{n>0}:\)
- Labeling of states of \( B_n \) in \((I_n)^\ell\)
A take on logical uniformity

- Usual strategy: $C_n \mapsto L_{C_n} \subseteq \Sigma^*$ then descr. cpx. on strings $\exists x \left( Q_a x \land \forall y \left( x < y \rightarrow Q_b y \right) \right)$
- Here: remove the need for the logic to decode

Definition ($L$-uniform branching program)

Family $(B_n)_{n > 0}$:
- Labeling of states of $B_n$ in $(I_n)^\ell$
- $\text{init}(n) \in (I_n)^\ell$, $\text{fin}(n) \subseteq (I_n)^\ell$, $I_n \rightarrow [n]$, $I_n = [n]^c \times S_n$
A take on logical uniformity

- Usual strategy: \(C_n \mapsto L_{C_n} \subseteq \Sigma^*\) then descr. cpx. on strings \(\exists x (Q_a x \land \forall y (x < y \rightarrow Q_b y))\)
- Here: remove the need for the logic to decode

Definition (\(L\)-uniform branching program)

Family \((B_n)_{n \geq 0}\):
- Labeling of states of \(B_n\) in \((I_n)^\ell\)
- \(\text{init}(n) \in (I_n)^\ell, \text{fin}(n) \subseteq (I_n)^\ell\),
- \(I_n \models \varphi_Q(q, i)\) iff state \(q\) of \(B_n\) queries \(i \in I_n\)
- \(I_n \models \varphi_C(q, q', j)\) iff edge valued \(j \in [n]\) between \(q, q'\)
A take on logical uniformity

- Usual strategy: $C_n \mapsto L_{C_n} \subseteq \Sigma^*$ then descr. cpx. on strings $\exists x (Q_a x \land \forall y (x < y \rightarrow Q_b y))$
- Here: remove the need for the logic to decode

**Definition (\(\mathcal{L}\)-uniform branching program)**

Family \((B_n)_{n>0}\):

- Labeling of states of \(B_n\) in \((I_n)^\ell\)
- \(\text{init}(n) \in (I_n)^\ell\), \(\text{fin}(n) \subseteq (I_n)^\ell\),
- \(I_n \models \varphi_Q(q, i)\) iff state \(q\) of \(B_n\) queries \(i \in I_n\)
- \(I_n \models \varphi_C(q, q', j)\) iff edge valued \(j \in [n]\) between \(q, q'\)
A take on logical uniformity

- Usual strategy: \( C_n \mapsto L_{C_n} \subseteq \Sigma^* \) then descr. cpx. on strings

\[
\exists x \left( Q_a x \land \forall y \left( x < y \rightarrow Q_b y \right) \right)
\]

- Here: remove the need for the logic to decode

**Definition (\( \mathcal{L} \)-uniform branching program)**

Family \( (B_n)_{n>0} \):

- Labeling of states of \( B_n \) in \( (I_n)^\ell \)
- \( \text{init}(n) \in (I_n)^\ell, \text{fin}(n) \subseteq (I_n)^\ell \),
- \( I_n \models \varphi_Q(q, i) \) iff state \( q \) of \( B_n \) queries \( i \in I_n \)
- \( I_n \models \varphi_C(q, q', j) \) iff edge valued \( j \in [n] \) between \( q, q' \)

- When \( \mathcal{L} \) is powerful enough, similar to unary shuffled coding
Outline

Presenting inputs

Uniform branching programs

**FO[=, cst]-uniform branching programs**

Further research
Uniformity logic under consideration
Free access to variables, paid access to array indexes

- Finer-grained uniformity differentiating variables ($S_n$) and array-indexes ($[n]^c$)
Uniformity logic under consideration
Free access to variables, paid access to array indexes

- Finer-grained uniformity differentiating variables \( (S_n) \) and array-indexes \( ([n]^c) \)

**Definition (FO[=, cst])**

First order logic with numerical constants and \( = \) on numbers and *any* predicate on \( S_n \)
Uniformity logic under consideration
Free access to variables, paid access to array indexes

- Finer-grained uniformity differentiating variables ($S_n$) and array-indexes ($[n]^c$)

Definition (FO[=, cst])
First order logic with numerical constants and $=$ on numbers and any predicate on $S_n$

Used as uniformity logic $\rightarrow$ unrestricted power on choosing variable, very restricted on choosing array index
Thrifty branching programs

Definition (Thrifty branching program)

When a node queries \(((i_1, \ldots, i_c), v) \in I_n\), then each \(i_j\) is a previous answer.
Thrifty branching programs

Definition (Thrifty branching program)
When a node queries \(((i_1, \ldots, i_c), v) \in I_n\), then each \(i_j\) is a previous answer.
Thrifty branching programs

Definition (Thrifty branching program)

When a node queries \(((i_1, \ldots, i_c), v) \in I_n\), then each \(i_j\) is a previous answer.
Thrifty branching programs

Definition (Thrifty branching program)

When a node queries \(((i_1, \ldots, i_c), v) \in I_n\), then each \(i_j\) is a previous answer.

Generalizes previous similar problem-based definitions [Gál et al., 2008, Cook et al., 2012]
Thrifty branching programs

Definition (Thrifty branching program)
When a node queries \(((i_1, \ldots, i_c), v) \in I_n\), then each \(i_j\) is a previous answer.

Generalizes previous similar problem-based definitions [Gál et al., 2008, Cook et al., 2012]

Lemma

\[ FO[=, \text{cst}] \text{-uniform det. branching programs can be made thrifty} \]
Thrifty branching programs

Definition (Thrifty branching program)
When a node queries \(((i_1, \ldots, i_c), v) \in I_n\), then each \(i_j\) is a previous answer.

Generalizes previous similar problem-based definitions [Gál et al., 2008, Cook et al., 2012]

Lemma
\(\text{FO}[=, \text{cst}]-\text{uniform det. branching programs can be made thrifty}\)

Derived from known lower bounds:

Theorem
\(\text{GEN and TreeEval have no uniform family of det. branching programs}\)
Thrifty branching programs

Definition (Thrifty branching program)

When a node queries \(((i_1, \ldots, i_c), v) \in I_n\), then each \(i_j\) is a previous answer.

Generalizes previous similar problem-based definitions
[Gál et al., 2008, Cook et al., 2012]

Lemma

\(\text{FO[=, cst]-uniform det. branching programs can be made thrifty}\)

Derived from known lower bounds:

Theorem

\(\text{GEN and TreeEval have no uniform family of det. branching programs}\)
Thrifty branching programs

Definition (Thrifty branching program)

When a node queries \(((i_1, \ldots, i_c), v) \in I_n\), then each \(i_j\) is a previous answer

Generalizes previous similar problem-based definitions [Gál et al., 2008, Cook et al., 2012]

Lemma

\[ FO[\leq, \text{cst}]-\text{uniform det. branching programs can be made thrifty} \]

Derived from known lower bounds:

Theorem

\emph{GEN and TreeEval have no uniform family of det. branching programs}
Thrifty branching programs

Definition (Thrifty branching program)

When a node queries \(((i_1, \ldots, i_c), v) \in I_n\), then each \(i_j\) is a previous answer.

Generalizes previous similar problem-based definitions [Gál et al., 2008, Cook et al., 2012]

Lemma

\(\text{FO}[=, \text{cst}]-\text{uniform det. branching programs can be made thrifty}\)

Derived from known lower bounds:

Theorem

\(\text{GEN and TreeEval have no uniform family of det. branching programs}\)
Presenting inputs

Uniform branching programs

FO[=, cst]-uniform branching programs

Further research
Further Research

- Derive lower bounds without relying on thriftiness
Further Research

- Derive lower bounds without relying on thriftiness
- Study expressiveness; is there an equivalent circuit/logic?
Further Research

- Derive lower bounds without relying on thriftiness
- Study expressiveness; is there an equivalent circuit/logic?
- Extend $\text{FO}[=, \text{cst}]$ to include successor, order, . . .
Further Research

- Derive lower bounds without relying on thriftiness
- Study expressiveness; is there an equivalent circuit/logic?
- Extend FO[@=, cst] to include successor, order, ...
- Balance power on \( I_n = [n]^c \times S_n \)
Further Research

- Derive lower bounds without relying on thriftiness
- Study expressiveness; is there an equivalent circuit/logic?
- Extend FO[=, cst] to include successor, order, ...
- Balance power on $I_n = [n]^c \times S_n$
- Apply similar framework to circuits (Traditionally FO[$<$, $\times 2$]-unif. NC$^1$ is already ALOGTIME)
Further Research

- Derive lower bounds without relying on thriftiness
- Study expressiveness; is there an equivalent circuit/logic?
- Extend FO[=, cst] to include successor, order, ...
- Balance power on $I_n = [n]^c \times S_n$
- Apply similar framework to circuits (Traditionally FO[<, $\times 2$]-unif. NC$^1$ is already ALOGTIME)
Thank you, merci!

Presenting inputs

Uniform branching programs

FO[=, cst]-uniform branching programs

Further research

Uniformity
Map $n \mapsto C_n$

Structured prob.
Set of maps
$l_n \rightarrow [n]$, $l_n = [n]^c \times S_n$

$L$-uniform
Rely on query form. $\varphi_Q(q, i)$ and connect. form. $\varphi_C(q, q', j)$
Bibliography I

On uniformity within NC\(^1\).

Fo[\(<\)]-uniformity.

Pebbles and branching programs for tree evaluation.
*TOCT*, 3(2):4.
