Listings and Logics

Yijia Chen
Shanghai Jiaotong University

June, 2011

(Joint work with **Jörg Flum**, Freiburg)

This is a follow-up to

Theorem (Krajíček and Pudlák, 1989; Sadowski, 2002; *C.* and Flum, 2010)

The following are all equivalent.

- $\textsc{List}(\mathbf{P}, \mathrm{TAUT}, \mathbf{P})$.
- $\mathrm{LFP}_{\mathrm{inv}}$ *captures* $\mathbf{P}$.
- TAUT *has a polynomial optimal proof system.*
- TAUT *has an almost optimal algorithm.*

Contents

Logics for **P**

Logics for **P**

There is an effective enumeration of all polynomial time computable subsets of graphs in terms of corresponding polynomial time machines

$$\mathbb{M}_1, \mathbb{M}_2, \ldots.$$

Logics for **P**

There is an effective enumeration of all polynomial time computable subsets of graphs in terms of corresponding polynomial time machines

$$\mathbb{M}_1, \mathbb{M}_2, \ldots.$$

For two isomorphic graphs G and H, the machine $\mathbb{M}_i$ might accept G but reject H. I.e., $\mathbb{M}_i$ does not necessarily decide a class of graphs that are closed under isomorphism, or a graph property.

Logics for **P**

There is an effective enumeration of all polynomial time computable subsets of graphs in terms of corresponding polynomial time machines

$$\mathbb{M}_1, \mathbb{M}_2, \ldots.$$

For two isomorphic graphs G and H, the machine $\mathbb{M}_i$ might accept G but reject H. I.e., $\mathbb{M}_i$ does not necessarily decide a class of graphs that are closed under isomorphism, or a graph property.

Question (Chandra and Harel, 1982)

Is there an effective enumeration of polynomial time computable graph properties in terms of corresponding polynomial time machines?

Logics for **P**

There is an effective enumeration of all polynomial time computable subsets of graphs in terms of corresponding polynomial time machines

$$\mathbb{M}_1, \mathbb{M}_2, \ldots.$$

For two isomorphic graphs G and H, the machine $\mathbb{M}_i$ might accept G but reject H. I.e., $\mathbb{M}_i$ does not necessarily decide a class of graphs that are closed under isomorphism, or a graph property.

Question (Chandra and Harel, 1982)

Is there an effective enumeration of polynomial time computable graph properties in terms of corresponding polynomial time machines?

Question (Gurevich, 1988)

Is there a logic capturing **P***?*

Logics for any complexity class C

Logics for any complexity class C

We can replace **P** by any complexity class.

Logics for any complexity class C

We can replace **P** by any complexity class.

Theorem (Fagin, 1974)

Existential second-order logic captures **NP**. *Therefore, we have an effective enumeration of nondeterministic polynomial time computable graph properties in terms of corresponding nondeterministic polynomial time machines.*

Logics for any complexity class C

We can replace **P** by any complexity class.

Theorem (Fagin, 1974)

Existential second-order logic captures **NP**. *Therefore, we have an effective enumeration of nondeterministic polynomial time computable graph properties in terms of corresponding nondeterministic polynomial time machines.*

But for any natural complexity class $C \subseteq \mathbf{P}$ it not known whether there is a logic capturing C, which is equivalent to the question:

> *Is there an effective enumeration of graph properties in C in terms of corresponding machines with resource bound according to C, or C-machines?*

Immerman and Vardi's Theorems

Immerman and Vardi's Theorems

Theorem

For $C = \mathbf{L}, \mathbf{NL}, \mathbf{P}$ we have an effective enumeration of ordered graph properties in C in terms of corresponding C-machines.

Immerman and Vardi's Theorems

Theorem

For $C = \mathbf{L}, \mathbf{NL}, \mathbf{P}$ we have an effective enumeration of ordered graph properties in C in terms of corresponding C-machines.

- DTC, *deterministic transitive closure logic*, *captures* **L** *on ordered structures.*
- TC, *transitive closure logic*, *captures* **NL** *on ordered structures.*
- LFP, *least fixed-point logic*, *captures* **P** *on ordered structures.*

Logics for any complexity class C (cont'd)

Logics for any complexity class C (cont'd)

For two complexity classes $C \subseteq C'$:

Is there an effective enumeration of graph properties in C in terms of corresponding C'-machines?

Logics for any complexity class C (cont'd)

For two complexity classes $C \subseteq C'$:

> *Is there an effective enumeration of graph properties in C in terms of corresponding C'-machines?*

Theorem (*C.* and Flum, 2009)

We have an effective enumeration of graph properties in **P** *in terms of corresponding* **NP***-machines.*

Listings

Listings

Definition

Let C and C' be complexity classes, and $Q \subseteq \Sigma^*$. A listing of the C-subsets of Q by C'-machines is an algorithm $\mathbb{L}$ that outputs Turing machines $\mathbb{M}_1, \mathbb{M}_2, \ldots$ of type C' such that

$$\{L(\mathbb{M}_i) \mid i \geq 1\} = \{X \subseteq Q \mid X \in C\},$$

where $L(\mathbb{M}_i)$ is the language accepted by $\mathbb{M}_i$,

Listings

Definition

Let C and C' be complexity classes, and $Q \subseteq \Sigma^*$. A listing of the C-subsets of Q by C'-machines is an algorithm $\mathbb{L}$ that outputs Turing machines $\mathbb{M}_1, \mathbb{M}_2, \ldots$ of type C' such that

$$\{L(\mathbb{M}_i) \mid i \geq 1\} = \{X \subseteq Q \mid X \in C\},$$

where $L(\mathbb{M}_i)$ is the language accepted by $\mathbb{M}_i$,

We write $\textsc{List}(C, Q, C')$ if there is a listing of the C-subsets of Q by C'-machines.

Our previous results

Our previous results

Theorem (*C.* and Flum, 2010)

If $\textsc{List}(\mathbf{P}, \mathrm{TAUT}, \mathbf{P})$*, then there is a logic for* $\mathbf{P}$*.*

Our previous results

Theorem (*C.* and Flum, 2010)

If $\textsc{List}(\mathbf{P}, \mathrm{TAUT}, \mathbf{P})$*, then there is a logic for* $\mathbf{P}$*.*

$\textsc{List}(\mathbf{P}, \mathrm{TAUT}, \mathbf{P})$ *if and only if the logic* $\mathrm{LFP}_{\mathrm{inv}}$*, the "order-invariant least fixed-point logic* LFP*,"* [Blass and Gurevich, 1988] *captures* $\mathbf{P}$*.*

Our previous results

Theorem (*C.* and Flum, 2010)

If $\textsc{List}(\mathbf{P}, \mathrm{TAUT}, \mathbf{P})$*, then there is a logic for* $\mathbf{P}$*.*

$\textsc{List}(\mathbf{P}, \mathrm{TAUT}, \mathbf{P})$ *if and only if the logic* $\mathrm{LFP}_{\mathrm{inv}}$*, the "order-invariant least fixed-point logic* LFP*,"* [Blass and Gurevich, 1988] *captures* $\mathbf{P}$*.*

What if we replace **P** by **L** or **NL**?

$\textsc{List}(\mathbf{L}, \mathrm{TAUT}, \mathbf{L})$

Theorem

Assume $\textsc{List}(\mathbf{L}, \mathrm{TAUT}, \mathbf{L})$. *Then* $\textsc{List}(\mathbf{P}, \mathrm{TAUT}, \mathbf{P})$. *Hence,* $\mathrm{LFP}_{\mathrm{inv}}$ *captures* $\mathbf{P}$.

$\textsc{List}(\mathbf{L}, \mathrm{TAUT}, \mathbf{L})$

Theorem

Assume $\textsc{List}(\mathbf{L}, \mathrm{TAUT}, \mathbf{L})$. *Then* $\textsc{List}(\mathbf{P}, \mathrm{TAUT}, \mathbf{P})$. *Hence,* $\mathrm{LFP}_{\mathrm{inv}}$ *captures* $\mathbf{P}$.

We will need the following fact:

Lemma

TAUT *has padding: there is a function* $\mathrm{pad} : \Sigma^* \times \Sigma^* \to \Sigma^*$ *such that:*

(i) *It is computable in logarithmic space.*

(ii) *For any* $x, y \in \Sigma^*$, $\big(\mathrm{pad}(x, y) \in \mathrm{TAUT} \iff x \in \mathrm{TAUT}\big)$.

(iii) *For any* $x, y \in \Sigma^*$, $|\mathrm{pad}(x, y)| > |x| + |y|$.

(iv) *There is a logspace algorithm which, given* $\mathrm{pad}(x, y)$ *recovers* y.

$\textsc{List}(\mathbf{L}, \mathrm{TAUT}, \mathbf{L})$

Theorem

Assume $\textsc{List}(\mathbf{L}, \mathrm{TAUT}, \mathbf{L})$. *Then* $\textsc{List}(\mathbf{P}, \mathrm{TAUT}, \mathbf{P})$. *Hence,* $\mathrm{LFP}_{\mathrm{inv}}$ *captures* $\mathbf{P}$.

We will need the following fact:

Lemma

TAUT *has padding: there is a function* $\mathrm{pad} : \Sigma^* \times \Sigma^* \to \Sigma^*$ *such that:*

(i) *It is computable in logarithmic space.*

(ii) *For any* $x, y \in \Sigma^*$, $\big(\mathrm{pad}(x, y) \in \mathrm{TAUT} \iff x \in \mathrm{TAUT}\big)$.

(iii) *For any* $x, y \in \Sigma^*$, $|\mathrm{pad}(x, y)| > |x| + |y|$.

(iv) *There is a logspace algorithm which, given* $\mathrm{pad}(x, y)$ *recovers* y.

Remark

In all the following results, the only properties we need for TAUT*: 1) it's* **coNP***-completeness; 2) it has padding.*

Proof of $\textsc{List}(\mathbf{L}, \mathsf{TAUT}, \mathbf{L}) \Rightarrow \textsc{List}(\mathbf{P}, \mathsf{TAUT}, \mathbf{P})$

Proof of $\textsc{List}(\mathbf{L}, \mathrm{TAUT}, \mathbf{L}) \Rightarrow \textsc{List}(\mathbf{P}, \mathrm{TAUT}, \mathbf{P})$

Let $\mathbb{M}$ be a machine. We set

$$\mathit{Comp}(\mathbb{M}) := \big\{\mathrm{pad}(x, \langle x, c\rangle) \mid x \in \Sigma^* \text{ and } c \text{ is a computation of } \mathbb{M} \text{ accepting } x\big\}.$$

Proof of $\textsc{List}(\mathbf{L}, \mathrm{TAUT}, \mathbf{L}) \Rightarrow \textsc{List}(\mathbf{P}, \mathrm{TAUT}, \mathbf{P})$

Let $\mathbb{M}$ be a machine. We set

$$\mathit{Comp}(\mathbb{M}) := \big\{\mathrm{pad}(x, \langle x, c\rangle) \mid x \in \Sigma^* \text{ and } c \text{ is a computation of } \mathbb{M} \text{ accepting } x\big\}.$$

Note from $\mathrm{pad}(x, \langle x, c\rangle)$ we can recover both x and c in logarithmic space.

Proof of $\textsc{List}(\mathbf{L}, \mathrm{TAUT}, \mathbf{L}) \Rightarrow \textsc{List}(\mathbf{P}, \mathrm{TAUT}, \mathbf{P})$

Let $\mathbb{M}$ be a machine. We set

$$Comp(\mathbb{M}) := \{\mathrm{pad}(x, \langle x, c\rangle) \mid x \in \Sigma^* \text{ and } c \text{ is a computation of } \mathbb{M} \text{ accepting } x\}.$$

Note from $\mathrm{pad}(x, \langle x, c\rangle)$ we can recover both x and c in logarithmic space.

- $comp(\mathbb{M})$ is in $\mathbf{L}$.
- $comp(\mathbb{M}) \subseteq \mathrm{TAUT}$ if and only if $L(\mathbb{M})$ is a subset of TAUT.

Proof of $\textsc{List}(\mathbf{L}, \mathsf{TAUT}, \mathbf{L}) \Rightarrow \textsc{List}(\mathbf{P}, \mathsf{TAUT}, \mathbf{P})$ (cont'd)

Proof of $\textsc{List}(\mathbf{L}, \text{TAUT}, \mathbf{L}) \Rightarrow \textsc{List}(\mathbf{P}, \text{TAUT}, \mathbf{P})$ (cont'd)

For two machines $\mathbb{D}$ and $\mathbb{D}'$ let $\mathbb{D}'(\mathbb{D})$ be a machine that on every input x

1. simulates $\mathbb{D}$ on x and let c be the corresponding computation;
2. simulates $\mathbb{D}'$ on $\text{pad}(x, \langle x, c \rangle)$.

Proof of $\textsc{List}(\mathbf{L}, \text{TAUT}, \mathbf{L}) \Rightarrow \textsc{List}(\mathbf{P}, \text{TAUT}, \mathbf{P})$ (cont'd)

For two machines $\mathbb{D}$ and $\mathbb{D}'$ let $\mathbb{D}'(\mathbb{D})$ be a machine that on every input x

1. simulates $\mathbb{D}$ on x and let c be the corresponding computation;
2. simulates $\mathbb{D}'$ on $\text{pad}(x, \langle x, c \rangle)$.

Let $\mathbb{D}$ be a **P**-machine that accepts a subset of TAUT. Then $comp(\mathbb{D})$ is accepted by an **L**-machine $\mathbb{D}'$. It is easy to see

$$L(\mathbb{D}'(\mathbb{D})) = L(\mathbb{D}).$$

Proof of $\textsc{List}(\mathbf{L}, \text{TAUT}, \mathbf{L}) \Rightarrow \textsc{List}(\mathbf{P}, \text{TAUT}, \mathbf{P})$ (cont'd)

For two machines $\mathbb{D}$ and $\mathbb{D}'$ let $\mathbb{D}'(\mathbb{D})$ be a machine that on every input x

1. simulates $\mathbb{D}$ on x and let c be the corresponding computation;
2. simulates $\mathbb{D}'$ on $\text{pad}(x, \langle x, c \rangle)$.

Let $\mathbb{D}$ be a **P**-machine that accepts a subset of TAUT. Then $comp(\mathbb{D})$ is accepted by an **L**-machine $\mathbb{D}'$. It is easy to see

$$L(\mathbb{D}'(\mathbb{D})) = L(\mathbb{D}).$$

If $\mathbb{D}'$ accepts a subset of TAUT, then so does $\mathbb{D}'(\mathbb{D})$ for every $\mathbb{D}$.

Proof of $\textsc{List}(\mathbf{L}, \mathrm{TAUT}, \mathbf{L}) \Rightarrow \textsc{List}(\mathbf{P}, \mathrm{TAUT}, \mathbf{P})$ (cont'd)

For two machines $\mathbb{D}$ and $\mathbb{D}'$ let $\mathbb{D}'(\mathbb{D})$ be a machine that on every input x

1. simulates $\mathbb{D}$ on x and let c be the corresponding computation;
2. simulates $\mathbb{D}'$ on $\mathrm{pad}(x, \langle x, c \rangle)$.

Let $\mathbb{D}$ be a **P**-machine that accepts a subset of TAUT. Then $comp(\mathbb{D})$ is accepted by an **L**-machine $\mathbb{D}'$. It is easy to see

$$L(\mathbb{D}'(\mathbb{D})) = L(\mathbb{D}).$$

If $\mathbb{D}'$ accepts a subset of TAUT, then so does $\mathbb{D}'(\mathbb{D})$ for every $\mathbb{D}$.

Then the following enumeration witnesses $\textsc{List}(\mathbf{P}, \mathrm{TAUT}, \mathbf{P})$:

$$\left(\mathbb{D}'_j(\mathbb{D}_i)\right)_{i,j \geq 1},$$

where $(\mathbb{D}_i)_{i \geq 1}$ is an enumeration of **P**-clocked machines and $(\mathbb{D}'_j)_{j \geq 1}$ witnesses $\textsc{List}(\mathbf{L}, \mathrm{TAUT}, \mathbf{L})$. □

Order invariance

Order invariance

Let L be one of DTC, TC and LFP.

Order invariance

Let L be one of DTC, TC and LFP.

For every vocabulary τ we let

$$\tau_< := \tau \,\dot\cup\, \{<\}.$$

Order invariance

Let L be one of DTC, TC and LFP.

For every vocabulary τ we let

$$\tau_< := \tau \dot{\cup} \{<\}.$$

Let φ be an $L[\tau_<]$-sentence and $n \geq 1$. φ is $\leq n$-invariant if for all τ-structures $\mathcal{A}$ with $|A| \leq n$ and all orderings $<_1$ and $<_2$ on A we have

$$(\mathcal{A}, <_1) \models_L \varphi \iff (\mathcal{A}, <_2) \models_L \varphi.$$

Order invariance

Let L be one of DTC, TC and LFP.

For every vocabulary τ we let

$$\tau_< := \tau \,\dot\cup\, \{<\}.$$

Let φ be an $L[\tau_<]$-sentence and $n \geq 1$. φ is $\leq n$-invariant if for all τ-structures $\mathcal{A}$ with $|A| \leq n$ and all orderings $<_1$ and $<_2$ on A we have

$$(\mathcal{A}, <_1) \models_L \varphi \iff (\mathcal{A}, <_2) \models_L \varphi.$$

We define

$$L\text{-Inv} := \big\{(\varphi, n) \mid \varphi \ L\text{-sentence}, \ \ n \geq 1, \text{ and } \varphi \leq n\text{-invariant}\big\}.$$

The invariant logics L_{inv}

The invariant logics L_{inv}

For every vocabulary τ we set

$$L_{\mathrm{inv}}[\tau] := L[\tau_<]$$

The invariant logics L_{inv}

For every vocabulary τ we set

$$L_{\mathrm{inv}}[\tau] := L[\tau_<]$$

For every $\varphi \in L_{\mathrm{inv}}[\tau]$ and every τ-structure $\mathcal{A}$

$$\mathcal{A} \models_{L_{\mathrm{inv}}} \varphi$$
$$\iff \Big((\varphi, |A|) \in L\text{-}\textsc{Inv} \text{ and } (\mathcal{A}, <) \models_L \varphi \text{ for some/all orderings } < \text{ on } A\Big).$$

The invariant logics L_{inv} (cont'd)

The invariant logics L_{inv} (cont'd)

Question

Does $\mathsf{LFP}_{\mathrm{inv}}/\mathsf{TC}_{\mathrm{inv}}/\mathsf{DTC}_{\mathrm{inv}}$ *capture* **P**/**NL**/**L**?

The invariant logics L_{inv} (cont'd)

Question

Does $\mathsf{LFP}_{\mathrm{inv}}/\mathsf{TC}_{\mathrm{inv}}/\mathsf{DTC}_{\mathrm{inv}}$ *capture* **P**/**NL**/**L**?

Lemma

Let K *be a class of structures.*

$$K \text{ is in } \mathbf{P}/\mathbf{NL}/\mathbf{L} \iff \text{for some } \varphi \in \mathsf{LFP}_{\mathrm{inv}}/\mathsf{TC}_{\mathrm{inv}}/\mathsf{DTC}_{\mathrm{inv}} \quad K = \left\{\mathcal{A} \mid \mathcal{A} \models_{\mathsf{LFP}_{\mathrm{inv}}/\mathsf{TC}_{\mathrm{inv}}/\mathsf{DTC}_{\mathrm{inv}}} \varphi\right\}.$$

The invariant logics L_{inv} (cont'd)

Question

Does $\mathsf{LFP}_{\mathrm{inv}}/\mathsf{TC}_{\mathrm{inv}}/\mathsf{DTC}_{\mathrm{inv}}$ *capture* **P**/**NL**/**L**?

Lemma

Let K *be a class of structures.*

$$K \text{ is in } \mathbf{P}/\mathbf{NL}/\mathbf{L} \iff \text{for some } \varphi \in \mathsf{LFP}_{\mathrm{inv}}/\mathsf{TC}_{\mathrm{inv}}/\mathsf{DTC}_{\mathrm{inv}} \quad K = \{\mathcal{A} \mid \mathcal{A} \models_{\mathsf{LFP}_{\mathrm{inv}}/\mathsf{TC}_{\mathrm{inv}}/\mathsf{DTC}_{\mathrm{inv}}} \varphi\}.$$

Question

Is there an algorithm $\mathbb{A}$ *deciding* $(\varphi, |A|) \in L\text{-}\textsc{Inv}$ *in such a way that for every fixed* $\varphi \in L$*:*

- *if* $L = \mathsf{LFP}$*, then* $\mathbb{A}$ *runs in time* $\|\mathcal{A}\|^{O(1)}$*;*
- *if* $L = \mathsf{TC}$*, then* $\mathbb{A}$ *runs in nondeterministic space* $O(\log \|\mathcal{A}\|)$*;*
- *if* $L = \mathsf{DTC}$*, then* $\mathbb{A}$ *runs in deterministic space* $O(\log \|\mathcal{A}\|)$*?*

Linking Listings to L_{inv}

Recall:

Theorem (*C.* and Flum, 2010)

$\textsc{List}(\mathbf{P}, \text{TAUT}, \mathbf{P})$ *if and only if* LFP_{inv} *captures* $\mathbf{P}$.

Linking Listings to L_{inv}

Recall:

Theorem (*C.* and Flum, 2010)

$\textsc{List}(\mathbf{P}, \text{TAUT}, \mathbf{P})$ *if and only if* LFP_{inv} *captures* $\mathbf{P}$.

Theorem

- $\textsc{List}(\mathbf{NL}, \text{TAUT}, \mathbf{NL})$ *if and only if* TC_{inv} *captures* $\mathbf{NL}$.
- $\textsc{List}(\mathbf{L}, \text{TAUT}, \mathbf{L})$ *if and only if* DTC_{inv} *captures* $\mathbf{L}$.

Linking Listings to L_{inv}

Recall:

Theorem (*C.* and Flum, 2010)

$\textsc{List}(\mathbf{P}, \text{TAUT}, \mathbf{P})$ *if and only if* LFP_{inv} *captures* $\mathbf{P}$.

Theorem

- $\textsc{List}(\mathbf{NL}, \text{TAUT}, \mathbf{NL})$ *if and only if* TC_{inv} *captures* $\mathbf{NL}$.
- $\textsc{List}(\mathbf{L}, \text{TAUT}, \mathbf{L})$ *if and only if* DTC_{inv} *captures* $\mathbf{L}$.

Corollary

DTC_{inv} *captures* $\mathbf{L} \Longrightarrow \text{TC}_{\text{inv}}$ *captures* $\mathbf{NL} \Longrightarrow \text{LFP}_{\text{inv}}$ *captures* $\mathbf{P}$.

Linking Listings to L_{inv}

Recall:

Theorem (*C.* and Flum, 2010)

$\textsc{List}(\mathbf{P}, \text{TAUT}, \mathbf{P})$ *if and only if* LFP_{inv} *captures* $\mathbf{P}$.

Theorem

- $\textsc{List}(\mathbf{NL}, \text{TAUT}, \mathbf{NL})$ *if and only if* TC_{inv} *captures* $\mathbf{NL}$.
- $\textsc{List}(\mathbf{L}, \text{TAUT}, \mathbf{L})$ *if and only if* DTC_{inv} *captures* $\mathbf{L}$.

Corollary

DTC_{inv} *captures* $\mathbf{L} \Longrightarrow \text{TC}_{\text{inv}}$ *captures* $\mathbf{NL} \Longrightarrow \text{LFP}_{\text{inv}}$ *captures* $\mathbf{P}$.

Remark

It is not known whether the existence of a logic capturing $\mathbf{P}$ *is implied by the existence of a logic capturing* $\mathbf{L}$.

$\textsc{List}(\mathbf{L}, \mathrm{TAUT}, \mathbf{L})$ implies that $\mathrm{DTC}_{\mathrm{inv}}$ captures $\mathbf{L}$

$\textsc{List}(\mathbf{L}, \mathrm{TAUT}, \mathbf{L})$ implies that $\mathrm{DTC}_{\mathrm{inv}}$ captures $\mathbf{L}$

Recall for every $\varphi \in \mathrm{DTC}_{\mathrm{inv}}[\tau]$ and every τ-structure $\mathcal{A}$

$$\mathcal{A} \models_{\mathrm{DTC}_{\mathrm{inv}}} \varphi$$
$$\iff \Big((\varphi, |A|) \in L\text{-}\textsc{Inv} \text{ and } (\mathcal{A}, <) \models_L \varphi \text{ for some } < \text{ on } A\Big).$$

$\textsc{List}(\mathbf{L}, \mathrm{TAUT}, \mathbf{L})$ implies that $\mathrm{DTC}_{\mathrm{inv}}$ captures $\mathbf{L}$

Recall for every $\varphi \in \mathrm{DTC}_{\mathrm{inv}}[\tau]$ and every τ-structure $\mathcal{A}$

$$\mathcal{A} \models_{\mathrm{DTC}_{\mathrm{inv}}} \varphi$$
$$\iff \Big((\varphi, |A|) \in L\text{-}\textsc{Inv} \text{ and } (\mathcal{A}, <) \models_L \varphi \text{ for some } < \text{ on } A\Big).$$

So our goal is to construct an algorithm $\mathbb{A}$ deciding $(\varphi, |A|) \in \text{DTC-}\textsc{Inv}$ in such a way that for every fixed $\varphi \in \mathrm{DTC}$ the algorithm $\mathbb{A}$ runs in deterministic space $O(\log \|\mathcal{A}\|) = O(\log |A|)$.

$\textsc{List}(\mathbf{L}, \mathrm{TAUT}, \mathbf{L})$ implies that $\mathrm{DTC}_{\mathrm{inv}}$ captures $\mathbf{L}$ (cont'd)

List(**L**, TAUT, **L**) implies that $\mathrm{DTC}_{\mathrm{inv}}$ captures **L** (cont'd)

Recall that $(\varphi, n) \in$ DTC-Inv, i.e., φ is $\leq n$-invariant, if for all τ-structures $\mathcal{A}$ with $|A| \leq n$ and all orderings $<_1$ and $<_2$ on A we have

$$(\mathcal{A}, <_1) \models_{\mathsf{DTC}} \varphi \iff (\mathcal{A}, <_2) \models_{\mathsf{DTC}} \varphi.$$

$\textsc{List}(\mathbf{L}, \text{TAUT}, \mathbf{L})$ implies that DTC_{inv} captures $\mathbf{L}$ (cont'd)

Recall that $(\varphi, n) \in \text{DTC-}\textsc{Inv}$, i.e., φ is $\leq n$-invariant, if for all τ-structures $\mathcal{A}$ with $|A| \leq n$ and all orderings $<_1$ and $<_2$ on A we have

$$(\mathcal{A}, <_1) \models_{\text{DTC}} \varphi \iff (\mathcal{A}, <_2) \models_{\text{DTC}} \varphi.$$

Therefore the following padded version of DTC-$\textsc{Inv}$ is in **coNP**:

$$Q := \left\{ (\varphi, n, \underbrace{1 \cdots 1}_{n^{|\varphi|} \text{ times}}) \;\middle|\; (\varphi, n) \in \text{DTC-}\textsc{Inv} \right\}.$$

List(**L**, TAUT, **L**) implies that $\mathrm{DTC}_{\mathrm{inv}}$ captures **L** (cont'd)

Recall that $(\varphi, n) \in$ DTC-Inv, i.e., φ is $\leq n$-invariant, if for all τ-structures $\mathcal{A}$ with $|A| \leq n$ and all orderings $<_1$ and $<_2$ on A we have

$$(\mathcal{A}, <_1) \models_{\mathrm{DTC}} \varphi \iff (\mathcal{A}, <_2) \models_{\mathrm{DTC}} \varphi.$$

Therefore the following padded version of DTC-Inv is in **coNP**:

$$Q := \left\{ (\varphi, n, \underbrace{1 \cdots 1}_{n^{|\varphi|} \text{ times}}) \;\middle|\; (\varphi, n) \in \text{DTC-Inv} \right\}.$$

Hence, there is a logspace reduction α from Q to TAUT:

$$(\varphi, n, \ldots) \mapsto \alpha(\varphi, n, \ldots).$$

Since TAUT is paddable, we can assume that φ and n can be recovered from $\alpha(\varphi, n, \ldots)$.

$\textsc{List}(\mathbf{L}, \mathrm{TAUT}, \mathbf{L})$ implies that $\mathrm{DTC}_{\mathrm{inv}}$ captures $\mathbf{L}$ (cont'd)

$\textsc{List}(\mathbf{L}, \mathrm{TAUT}, \mathbf{L})$ implies that $\mathrm{DTC}_{\mathrm{inv}}$ captures $\mathbf{L}$ (cont'd)

Let $\varphi \in \mathrm{DTC}$.

$\textsc{List}(\mathbf{L}, \mathrm{TAUT}, \mathbf{L})$ implies that $\mathrm{DTC}_{\mathrm{inv}}$ captures $\mathbf{L}$ (cont'd)

Let $\varphi \in \mathrm{DTC}$. We consider the set

$$Q(\varphi) := \left\{ \alpha(\varphi, n, \ldots) \,\middle|\, n \in \mathbb{N} \text{ and } (\varphi, n) \in \text{DTC-}\textsc{Inv} \right\}.$$

$\textsc{List}(\mathbf{L}, \mathrm{TAUT}, \mathbf{L})$ implies that $\mathrm{DTC}_{\mathrm{inv}}$ captures $\mathbf{L}$ (cont'd)

Let $\varphi \in \mathrm{DTC}$. We consider the set

$$Q(\varphi) := \left\{ \alpha(\varphi, n, \ldots) \,\middle|\, n \in \mathbb{N} \text{ and } (\varphi, n) \in \text{DTC-}\textsc{Inv} \right\}.$$

Assume that $(\varphi, n) \in \text{DTC-}\textsc{Inv}$ for all $n \in \mathbb{N}$.

List(**L**, TAUT, **L**) implies that $\mathrm{DTC}_{\mathrm{inv}}$ captures **L** (cont'd)

Let $\varphi \in \mathrm{DTC}$. We consider the set

$$Q(\varphi) := \left\{ \alpha(\varphi, n, \ldots) \,\middle|\, n \in \mathbb{N} \text{ and } (\varphi, n) \in \text{DTC-Inv} \right\}.$$

Assume that $(\varphi, n) \in$ DTC-Inv for all $n \in \mathbb{N}$. Recall we can recover φ and n from $\alpha(\varphi, n, \ldots)$ in logspace. So $Q(\varphi)$ is infinite and in **L**.

$\textsc{List}(\mathbf{L}, \text{TAUT}, \mathbf{L})$ implies that DTC_{inv} captures $\mathbf{L}$ (cont'd)

Let $\varphi \in \text{DTC}$. We consider the set

$$Q(\varphi) := \left\{ \alpha(\varphi, n, \ldots) \,\middle|\, n \in \mathbb{N} \text{ and } (\varphi, n) \in \text{DTC-}\textsc{Inv} \right\}.$$

Assume that $(\varphi, n) \in \text{DTC-}\textsc{Inv}$ for all $n \in \mathbb{N}$. Recall we can recover φ and n from $\alpha(\varphi, n, \ldots)$ in logspace. So $Q(\varphi)$ is infinite and in $\mathbf{L}$.

Then $\textsc{List}(\mathbf{L}, \text{TAUT}, \mathbf{L})$ implies that $Q(\varphi)$ is one of the $L(\mathbb{M}_i)$ in the corresponding listing:

$$\mathbb{M}_1, \mathbb{M}_2, \ldots,$$

say $\mathbb{M}_{i_\varphi}$.

$\textsc{List}(\mathbf{L}, \mathrm{TAUT}, \mathbf{L})$ implies that $\mathrm{DTC}_{\mathrm{inv}}$ captures $\mathbf{L}$ (cont'd)

$\textsc{List}(\mathbf{L}, \mathrm{TAUT}, \mathbf{L})$ implies that $\mathrm{DTC}_{\mathrm{inv}}$ captures $\mathbf{L}$ (cont'd)

Now we consider a first algorithm $\mathbb{I}$ which on every instance (φ, n) of DTC-$\textsc{Inv}$

1. $k \leftarrow 1$;
2. generates the machine $\mathbb{M}_k$ in the listing $\textsc{List}(\mathbf{L}, \mathrm{TAUT}, \mathbf{L})$;
3. simulates all $\mathbb{M}_1, \ldots, \mathbb{M}_k$ on input $(\varphi, n, 1 \cdots 1)$ using space $k \cdot \log n$;
4. if one $\mathbb{M}_i$ accepts $\alpha(\varphi, n, \ldots)$, then accepts;
5. $k \leftarrow k + 1$ and goes to 2.

$\textsc{List}(\mathbf{L}, \mathrm{TAUT}, \mathbf{L})$ implies that $\mathrm{DTC}_{\mathrm{inv}}$ captures $\mathbf{L}$ (cont'd)

Now we consider a first algorithm $\mathbb{I}$ which on every instance (φ, n) of DTC-$\textsc{Inv}$

1. $k \leftarrow 1$;
2. generates the machine $\mathbb{M}_k$ in the listing $\textsc{List}(\mathbf{L}, \mathrm{TAUT}, \mathbf{L})$;
3. simulates all $\mathbb{M}_1, \ldots, \mathbb{M}_k$ on input $(\varphi, n, 1 \cdots 1)$ using space $k \cdot \log n$;
4. if one $\mathbb{M}_i$ accepts $\alpha(\varphi, n, \ldots)$, then accepts;
5. $k \leftarrow k + 1$ and goes to 2.

If $(\varphi, n) \in$ DTC-$\textsc{Inv}$ for all $n \in \mathbb{N}$, recall $\mathbb{M}_{i_\varphi}$ decides $\left\{\alpha(\varphi, n, \ldots) \middle| \ n \in \mathbb{N}\right\}$ in logspace,

$\textsc{List}(\mathbf{L}, \text{TAUT}, \mathbf{L})$ implies that DTC_{inv} captures $\mathbf{L}$ (cont'd)

Now we consider a first algorithm $\mathbb{I}$ which on every instance (φ, n) of DTC-$\textsc{Inv}$

1. $k \leftarrow 1$;
2. generates the machine $\mathbb{M}_k$ in the listing $\textsc{List}(\mathbf{L}, \text{TAUT}, \mathbf{L})$;
3. simulates all $\mathbb{M}_1, \ldots, \mathbb{M}_k$ on input $(\varphi, n, 1 \cdots 1)$ using space $k \cdot \log n$;
4. if one $\mathbb{M}_i$ accepts $\alpha(\varphi, n, \ldots)$, then accepts;
5. $k \leftarrow k + 1$ and goes to 2.

If $(\varphi, n) \in$ DTC-$\textsc{Inv}$ for all $n \in \mathbb{N}$, recall $\mathbb{M}_{i_\varphi}$ decides $\left\{\alpha(\varphi, n, \ldots) \middle| \ n \in \mathbb{N}\right\}$ in logspace, i.e.,

$$O\left(\log\left|(\varphi, n, \underbrace{1 \cdots 1}_{n^{|\varphi|} \text{ times}})\right|\right) = O(|\varphi| \cdot \log n) = O(\log n),$$

where the second equality is by that φ is fixed.

$\textsc{List}(\mathbf{L}, \mathrm{TAUT}, \mathbf{L})$ implies that $\mathrm{DTC}_{\mathrm{inv}}$ captures $\mathbf{L}$ (cont'd)

Now we consider a first algorithm $\mathbb{I}$ which on every instance (φ, n) of DTC-$\textsc{Inv}$

1. $k \leftarrow 1$;
2. generates the machine $\mathbb{M}_k$ in the listing $\textsc{List}(\mathbf{L}, \mathrm{TAUT}, \mathbf{L})$;
3. simulates all $\mathbb{M}_1, \ldots, \mathbb{M}_k$ on input $(\varphi, n, 1 \cdots 1)$ using space $k \cdot \log n$;
4. if one $\mathbb{M}_i$ accepts $\alpha(\varphi, n, \ldots)$, then accepts;
5. $k \leftarrow k + 1$ and goes to 2.

If $(\varphi, n) \in$ DTC-$\textsc{Inv}$ for all $n \in \mathbb{N}$, recall $\mathbb{M}_{i_\varphi}$ decides $\left\{\alpha(\varphi, n, \ldots) \middle| \ n \in \mathbb{N}\right\}$ in logspace, i.e.,

$$O\left(\log\left|(\varphi, n, \underbrace{1 \cdots 1}_{n^{|\varphi|} \text{ times}})\right|\right) = O(|\varphi| \cdot \log n) = O(\log n),$$

where the second equality is by that φ is fixed. Hence, the algorithm $\mathbb{I}$ also uses space $O(\log n)$.

$\textsc{List}(\mathbf{L}, \mathrm{TAUT}, \mathbf{L})$ implies that $\mathrm{DTC}_{\mathrm{inv}}$ captures $\mathbf{L}$ (cont'd)

$\textsc{List}(\mathbf{L}, \mathrm{TAUT}, \mathbf{L})$ implies that $\mathrm{DTC}_{\mathrm{inv}}$ captures $\mathbf{L}$ (cont'd)

We consider a second algorithm $\mathbb{B}$ which on every instance (φ, n):

$\textsc{List}(\mathbf{L}, \mathrm{TAUT}, \mathbf{L})$ implies that $\mathrm{DTC}_{\mathrm{inv}}$ captures $\mathbf{L}$ (cont'd)

We consider a second algorithm $\mathbb{B}$ which on every instance (φ, n):

1. for every $i = 1, 2, 3, \ldots$, every $\mathcal{A}$ and every two orderings $<_1$ and $<_2$ checks whether $\Big((\mathcal{A}, <_1) \models_{\mathrm{DTC}} \varphi \iff (\mathcal{A}, <_1) \models_{\mathrm{DTC}} \varphi \Big)$; stops once the equivalence does not hold;
2. if $n < i$, then accepts else rejects.

List(**L**, TAUT, **L**) implies that $\mathrm{DTC}_{\mathrm{inv}}$ captures **L** (cont'd)

We consider a second algorithm $\mathbb{B}$ which on every instance (φ, n):

1. for every $i = 1, 2, 3, \ldots$, every $\mathcal{A}$ and every two orderings $<_1$ and $<_2$ checks whether $\Big((\mathcal{A}, <_1) \models_{\mathrm{DTC}} \varphi \iff (\mathcal{A}, <_1) \models_{\mathrm{DTC}} \varphi \Big)$; stops once the equivalence does not hold;
2. if $n < i$, then accepts else rejects.

If $(\varphi, n) \notin$ DTC-Inv for some $n \in \mathbb{N}$, then Step 1 eventually halts with the minimum i with $(\varphi, i) \notin$ DTC-Inv.

List(**L**, TAUT, **L**) implies that $\mathrm{DTC}_{\mathrm{inv}}$ captures **L** (cont'd)

We consider a second algorithm $\mathbb{B}$ which on every instance (φ, n):

1. for every $i = 1, 2, 3, \ldots$, every $\mathcal{A}$ and every two orderings $<_1$ and $<_2$ checks whether $\Big((\mathcal{A}, <_1) \models_{\mathrm{DTC}} \varphi \iff (\mathcal{A}, <_1) \models_{\mathrm{DTC}} \varphi \Big)$; stops once the equivalence does not hold;
2. if $n < i$, then accepts else rejects.

If $(\varphi, n) \notin$ DTC-Inv for some $n \in \mathbb{N}$, then Step 1 eventually halts with the minimum i with $(\varphi, i) \notin$ DTC-Inv. Thus Step 2 answers correctly.

List(**L**, TAUT, **L**) implies that $\mathrm{DTC}_{\mathrm{inv}}$ captures **L** (cont'd)

We consider a second algorithm $\mathbb{B}$ which on every instance (φ, n):

1. for every $i = 1, 2, 3, \ldots$, every $\mathcal{A}$ and every two orderings $<_1$ and $<_2$ checks whether $\Big((\mathcal{A}, <_1) \models_{\mathrm{DTC}} \varphi \iff (\mathcal{A}, <_1) \models_{\mathrm{DTC}} \varphi \Big)$; stops once the equivalence does not hold;
2. if $n < i$, then accepts else rejects.

If $(\varphi, n) \notin$ DTC-Inv for some $n \in \mathbb{N}$, then Step 1 eventually halts with the minimum i with $(\varphi, i) \notin$ DTC-Inv. Thus Step 2 answers correctly.

The space used by $\mathbb{B}$ in the first step only depends on φ, hence by fixing φ the total space that $\mathbb{B}$ needs is

$$O(\log n).$$

$\textsc{List}(\mathbf{L}, \mathrm{TAUT}, \mathbf{L})$ implies that $\mathrm{DTC}_{\mathrm{inv}}$ captures $\mathbf{L}$ (cont'd)

$\textsc{List}(\mathbf{L}, \mathrm{TAUT}, \mathbf{L})$ implies that $\mathrm{DTC}_{\mathrm{inv}}$ captures $\mathbf{L}$ (cont'd)

Recall our goal is to construct an algorithm $\mathbb{A}$ deciding $(\varphi, n) \in \text{DTC-}\textsc{Inv}$ in such a way that for every fixed $\varphi \in \mathrm{DTC}$ the algorithm $\mathbb{A}$ runs in deterministic space $O(\log n)$.

$\textsc{List}(\mathbf{L}, \mathrm{TAUT}, \mathbf{L})$ implies that $\mathrm{DTC}_{\mathrm{inv}}$ captures $\mathbf{L}$ (cont'd)

Recall our goal is to construct an algorithm $\mathbb{A}$ deciding $(\varphi, n) \in \text{DTC-}\textsc{Inv}$ in such a way that for every fixed $\varphi \in \mathrm{DTC}$ the algorithm $\mathbb{A}$ runs in deterministic space $O(\log n)$.

The desired algorithm $\mathbb{A}$ on every instance (φ, n)

1. $\ell \leftarrow 1$;
2. simulates both $\mathbb{I}$ and $\mathbb{B}$ using space at most ℓ;
3. if one of the simulation halts, then accepts or rejects accordingly;
4. otherwise $\ell \leftarrow \ell + 1$ and goes to 1.

$\textsc{List}(\mathbf{L}, \mathrm{TAUT}, \mathbf{L})$ implies that $\mathrm{DTC}_{\mathrm{inv}}$ captures $\mathbf{L}$ (cont'd)

Recall our goal is to construct an algorithm $\mathbb{A}$ deciding $(\varphi, n) \in \text{DTC-}\textsc{Inv}$ in such a way that for every fixed $\varphi \in \mathrm{DTC}$ the algorithm $\mathbb{A}$ runs in deterministic space $O(\log n)$.

The desired algorithm $\mathbb{A}$ on every instance (φ, n)

1. $\ell \leftarrow 1$;
2. simulates both $\mathbb{I}$ and $\mathbb{B}$ using space at most ℓ;
3. if one of the simulation halts, then accepts or rejects accordingly;
4. otherwise $\ell \leftarrow \ell + 1$ and goes to 1.

Then for every fixed $\varphi \in \mathrm{DTC}$

$\textsc{List}(\mathbf{L}, \mathrm{TAUT}, \mathbf{L})$ implies that $\mathrm{DTC}_{\mathrm{inv}}$ captures $\mathbf{L}$ (cont'd)

Recall our goal is to construct an algorithm $\mathbb{A}$ deciding $(\varphi, n) \in \text{DTC-}\textsc{Inv}$ in such a way that for every fixed $\varphi \in \mathrm{DTC}$ the algorithm $\mathbb{A}$ runs in deterministic space $O(\log n)$.

The desired algorithm $\mathbb{A}$ on every instance (φ, n)

1. $\ell \leftarrow 1$;
2. simulates both $\mathbb{I}$ and $\mathbb{B}$ using space at most ℓ;
3. if one of the simulation halts, then accepts or rejects accordingly;
4. otherwise $\ell \leftarrow \ell + 1$ and goes to 1.

Then for every fixed $\varphi \in \mathrm{DTC}$

- if $(\varphi, n) \in \text{DTC-}\textsc{Inv}$ for all $n \in \mathbb{N}$, then $\mathbb{I}$ uses space $O(\log n)$, and so does $\mathbb{A}$;

$\textsc{List}(\mathbf{L}, \mathrm{TAUT}, \mathbf{L})$ implies that $\mathrm{DTC}_{\mathrm{inv}}$ captures $\mathbf{L}$ (cont'd)

Recall our goal is to construct an algorithm $\mathbb{A}$ deciding $(\varphi, n) \in \text{DTC-}\textsc{Inv}$ in such a way that for every fixed $\varphi \in \mathrm{DTC}$ the algorithm $\mathbb{A}$ runs in deterministic space $O(\log n)$.

The desired algorithm $\mathbb{A}$ on every instance (φ, n)

1. $\ell \leftarrow 1$;
2. simulates both $\mathbb{I}$ and $\mathbb{B}$ using space at most ℓ;
3. if one of the simulation halts, then accepts or rejects accordingly;
4. otherwise $\ell \leftarrow \ell + 1$ and goes to 1.

Then for every fixed $\varphi \in \mathrm{DTC}$

- if $(\varphi, n) \in \text{DTC-}\textsc{Inv}$ for all $n \in \mathbb{N}$, then $\mathbb{I}$ uses space $O(\log n)$, and so does $\mathbb{A}$;
- if $(\varphi, n) \notin \text{DTC-}\textsc{Inv}$ for some $n \in \mathbb{N}$, then $\mathbb{B}$ uses space $O(\log n)$, and so does $\mathbb{A}$.

$\square$

Optimality

Theorem (Krajíček and Pudlák, 1989; Sadowski, 2002; *C.* and Flum, 2010)

The following are all equivalent.

- $\textsc{List}(\mathbf{P}, \mathrm{TAUT}, \mathbf{P})$.
- $\mathrm{LFP}_{\mathrm{inv}}$ *captures* $\mathbf{P}$.
- TAUT *has a polynomial optimal proof system.*
- TAUT *has an almost optimal algorithm.*

Space optimality

Theorem

The following are all equivalent.

- $\textsc{List}(\mathbf{L}, \text{TAUT}, \mathbf{L})$.
- DTC_{inv} *captures* $\mathbf{L}$.
- TAUT *has a space optimal logspace proof system.*
- TAUT *has an almost space optimal algorithm.*

Logspace proof systems

Logspace proof systems

Definition

1. A logspace proof system for TAUT is a surjective function $P : \Sigma^* \to \text{TAUT}$ computable in logarithmic space.

Logspace proof systems

Definition

1. A logspace proof system for TAUT is a surjective function $P : \Sigma^* \to \text{TAUT}$ computable in logarithmic space.
2. Let $P, P' : \Sigma^* \to \text{TAUT}$ be logspace proof systems for TAUT. We say that P logspace simulates P' if there exists a logspace computable function $g : \Sigma^* \to \Sigma^*$ such that $P(g(w)) = P'(w)$ for every $w \in \Sigma^*$.

Logspace proof systems

Definition

1. A logspace proof system for TAUT is a surjective function $P : \Sigma^* \to$ TAUT computable in logarithmic space.
2. Let $P, P' : \Sigma^* \to$ TAUT be logspace proof systems for TAUT. We say that P logspace simulates P' if there exists a logspace computable function $g : \Sigma^* \to \Sigma^*$ such that $P(g(w)) = P'(w)$ for every $w \in \Sigma^*$.
3. A logspace proof system for TAUT is space optimal if it logspace simulates every logspace proof system for TAUT.

Almost space optimal algorithms

Almost space optimal algorithms

Definition

A deterministic algorithm $\mathbb{A}$ deciding TAUT is almost space optimal for TAUT if for every deterministic algorithm $\mathbb{B}$ which decides TAUT there is a $d \in \mathbb{N}$ such that for all $x \in \text{TAUT}$

$$s_{\mathbb{A}}(x) \leq d \cdot (s_{\mathbb{B}}(x) + \log |x|),$$

where $s_{\mathbb{A}}(x)$ is the space required by $\mathbb{A}$ on x, and similarly for $s_{\mathbb{B}}(x)$.

Space optimality implies time optimality

Space optimality implies time optimality

Corollary

If TAUT *has an almost space algorithm, then* TAUT *has an almost (time) optimal algorithm.*

Space optimality implies time optimality

Corollary

If TAUT *has an almost space algorithm, then* TAUT *has an almost (time) optimal algorithm.*

Definition

A deterministic algorithm $\mathbb{A}$ deciding TAUT is (time) optimal for TAUT if for every deterministic algorithm $\mathbb{B}$ which decides TAUT and for all $x \in$ TAUT

$$t_{\mathbb{A}}(x) \leq (t_{\mathbb{B}}(x) + |x|)^{O(1)},$$

where $t_{\mathbb{A}}(x)$ is the time required by $\mathbb{A}$ on x, and similarly for $t_{\mathbb{B}}(x)$.

Thank You!