Circuits

Lecture 11
Uniform Circuit Complexity
Recall
Recall

- Non-uniform complexity
Recall

- Non-uniform complexity
- $P/1 \not\subseteq$ Decidable
Recall

- Non-uniform complexity
- $P/1 \not\subseteq \text{Decidable}$
- $NP \subseteq P/\log \Rightarrow NP = P$
Recall

- Non-uniform complexity
  - $P/1 \not\subseteq$ Decidable
  - $NP \subseteq P/log \implies NP = P$
  - $NP \subseteq P/poly \implies PH = \Sigma_2^P$
Recall

- Non-uniform complexity
  - $P/1 \not\subseteq$ Decidable
  - $NP \subseteq P/log \implies NP = P$
  - $NP \subseteq P/poly \implies PH = \Sigma^p_2$
- Circuit Complexity
Recall

- Non-uniform complexity
  - $P/1 \nsubseteq$ Decidable
  - $NP \subseteq P/log \Rightarrow NP = P$
  - $NP \subseteq P/poly \Rightarrow \text{PH} = \Sigma_2^P$

- Circuit Complexity
  - $\text{SIZE}(\text{poly}) = P/poly$
Recall

- Non-uniform complexity
  - $P/1 \not\subseteq \text{Decidable}$
  - $\text{NP} \subseteq P/\log \Rightarrow \text{NP} = P$
  - $\text{NP} \subseteq P/poly \Rightarrow \text{PH} = \Sigma_2^P$

- Circuit Complexity
  - SIZE(poly) = $P/poly$
  - SIZE-hierarchy
Recall

- Non-uniform complexity
  - $P/1 \not\subseteq$ Decidable
  - $NP \subseteq P/\log \Rightarrow NP = P$
  - $NP \subseteq P/poly \Rightarrow PH = \Sigma_2^P$

- Circuit Complexity
  - $\text{SIZE}(\text{poly}) = P/poly$
  - SIZE-hierarchy
    - SIZE($T'$) $\subseteq$ SIZE($T$) if $T = \Omega(t2^t)$ and $T' = O(2^t/t)$
Recall

- Non-uniform complexity
  - P/1 \not\subseteq Decision
  - NP \subseteq P/log \Rightarrow NP = P
  - NP \subseteq P/poly \Rightarrow PH = \Sigma_2^P

- Circuit Complexity
  - SIZE(poly) = P/poly

- SIZE-hierarchy
  - SIZE(T′) \subseteq SIZE(T) if T=\Omega(t2^t) and T′=O(2^t/t)
  - Most functions on t bits (that ignore last n-t bits) are in SIZE(T) but not in SIZE(T′)
Uniform Circuits
Uniform Circuits

- Uniform circuit family: constructed by a TM
Uniform Circuits

- Uniform circuit family: constructed by a TM
- Undecidable languages are undecidable for these circuits families
Uniform Circuits

Uniform circuit family: constructed by a TM

Undecidable languages are undecidable for these circuits families

Can relate their complexity classes to classes defined using TMs
Uniform Circuits

- Uniform circuit family: constructed by a TM
  - Undecidable languages are undecidable for these circuits families
  - Can relate their complexity classes to classes defined using TMs
- Logspace-uniform:
Uniform Circuits

Uniform circuit family: constructed by a TM

Undecidable languages are undecidable for these circuits families

Can relate their complexity classes to classes defined using TMs

Logspace-uniform:

An $O(\log n)$ space TM can compute the circuit
$\text{NC}^i$ and $\text{AC}^i$
$\text{NC}^i$ and $\text{AC}^i$

$\text{NC}^i$: class of languages decided by bounded fan-in logspace-uniform circuits of polynomial size and depth $O(\log^i n)$
NC$^i$ and AC$^i$

- **NC$^i$**: class of languages decided by bounded fan-in logspace-uniform circuits of polynomial size and depth $O(\log^i n)$
- **AC$^i$**: Similar, but unbounded fan-in circuits
NC\textsuperscript{i} and AC\textsuperscript{i}

- NC\textsuperscript{i}: class of languages decided by bounded fan-in logspace-uniform circuits of polynomial size and depth $O(\log^i n)$
- AC\textsuperscript{i}: Similar, but unbounded fan-in circuits
- NC\textsuperscript{0} and AC\textsuperscript{0}: constant depth circuits
NC^i and AC^i

- **NC^i**: class of languages decided by bounded fan-in logspace-uniform circuits of polynomial size and depth \(O(\log^i n)\)

- **AC^i**: Similar, but unbounded fan-in circuits

- **NC^0** and **AC^0**: constant depth circuits

- **NC^0** output depends on only a constant number of input bits
NC\textsuperscript{i} and AC\textsuperscript{i}

\begin{itemize}
  \item NC\textsuperscript{i}: class of languages decided by bounded fan-in logspace-uniform circuits of polynomial size and depth \(O(\log^n n)\)
  \item AC\textsuperscript{i}: Similar, but unbounded fan-in circuits
  \item NC\textsuperscript{0} and AC\textsuperscript{0}: constant depth circuits
    \begin{itemize}
      \item NC\textsuperscript{0} output depends on only a constant number of input bits
      \item NC\textsuperscript{0} \nsubseteq AC\textsuperscript{0}: Consider \(L = \{1,11,111,...\}\)
    \end{itemize}
\end{itemize}
NC and AC
NC and AC

\[ \text{NC} = \bigcup_{i>0} \text{NC}^i. \text{ Similarly AC.} \]
NC and AC

\[ \text{NC} = \bigcup_{i > 0} \text{NC}^i. \text{ Similarly AC.} \]

\[ \text{NC}^i \subseteq \text{AC}^i \subseteq \text{NC}^{i+1} \]
NC and AC

_NC = \bigcup_{i>0} NC^i_. Similarly AC.

_NC^i \subseteq AC^i \subseteq NC^{i+1}_

Clearly NC^i \subseteq AC^i_
NC and AC

\( NC = \bigcup_{i>0} NC^i \). Similarly AC.

\( NC^i \subseteq AC^i \subseteq NC^{i+1} \)

Clearly \( NC^i \subseteq AC^i \)

\( AC^i \subseteq NC^{i+1} \) because polynomial fan-in can be reduced to constant fan-in by using a log depth tree
NC and AC

\( NC = \bigcup_{i>0} NC^i \). Similarly AC.

\( NC^i \subseteq AC^i \subseteq NC^{i+1} \)

Clearly \( NC^i \subseteq AC^i \)

\( AC^i \subseteq NC^{i+1} \) because polynomial fan-in can be reduced to constant fan-in by using a log depth tree.

So \( NC = AC \)
\( \text{NC} \subseteq \text{P} \)
NC ⊆ P

Generate circuit of the right input size and evaluate on input
$$\text{NC} \subseteq \text{P}$$

- Generate circuit of the right input size and evaluate on input
- Generating the circuit
\( \mathsf{NC} \subseteq \mathsf{P} \)

- Generate circuit of the right input size and evaluate on input
- Generating the circuit
  - in logspace, so poly time; also circuit size is poly
\[ \text{NC} \subseteq \text{P} \]

- Generate circuit of the right input size and evaluate on input
- Generating the circuit
  - in logspace, so poly time; also circuit size is poly
- Evaluating the gates
NC ⊆ P

- Generate circuit of the right input size and evaluate on input
  - Generating the circuit
    - in logspace, so poly time; also circuit size is poly
  - Evaluating the gates
    - Poly(n) gates
\( \text{NC} \subseteq \text{P} \)

- Generate circuit of the right input size and evaluate on input
  - Generating the circuit
    - in logspace, so poly time; also circuit size is poly
  - Evaluating the gates
    - \( \text{Poly}(n) \) gates
    - Per gate takes \( O(1) \) time + time to look up output values of (already evaluated) gates
NC $\subseteq$ P

- Generate circuit of the right input size and evaluate on input
- Generating the circuit
  - in logspace, so poly time; also circuit size is poly
- Evaluating the gates
  - Poly(n) gates
  - Per gate takes $O(1)$ time + time to look up output values of (already evaluated) gates
- Open problem: Is NC = P?
Motivation for NC
Motivation for NC

- Fast parallel computation is (loosely) modeled as having poly many processors and taking poly-log time
Motivation for NC

- Fast parallel computation is (loosely) modeled as having poly many processors and taking poly-log time

- Corresponds to NC (How?)
Motivation for NC

Fast parallel computation is (loosely) modeled as having poly many processors and taking poly-log time.

- Corresponds to NC (How?)
- Depth translates to time
Motivation for NC

- Fast parallel computation is (loosely) modeled as having poly many processors and taking poly-log time
  - Corresponds to NC (How?)
  - Depth translates to time
  - Total “work” is size of the circuit
An example
An example

PARITY in NC$^1$
An example

PARITY in NC$^1$

PARITY = \{ x \mid x \text{ has odd number of 1s} \}
An example

PARITY in $\mathbf{NC}^1$

PARITY = \{ $x$ | $x$ has odd number of 1s \}

Circuit should evaluate $x_1 \oplus x_2 \oplus ... \oplus x_n$
An example

实例

PARITY in $\text{NC}^1$

PARITY = \{ x \mid x \text{ has odd number of } 1s \}

Circuit should evaluate $x_1 \oplus x_2 \oplus \ldots \oplus x_n$

Tree of $n-1$ XOR gates: log $n$ deep
An example

- **PARITY in NC^1**
  - **PARITY** = \{ x | x has odd number of 1s \}
  - Circuit should evaluate \( x_1 \oplus x_2 \oplus \ldots \oplus x_n \)
  - Tree of n-1 XOR gates: log n deep
  - Each XOR gate implemented in depth 3
An example

- **PARITY in $\text{NC}^1$**
  - $\text{PARITY} = \{ x \mid x \text{ has odd number of } 1\text{s} \}$
  - Circuit should evaluate $x_1 \oplus x_2 \oplus ... \oplus x_n$
    - Tree of $n-1$ XOR gates: $\log n$ deep
    - Each XOR gate implemented in depth 3
Another example
Another example

PATH ∈ AC¹
Another example

PATH ∈ AC¹

“Boolean” Matrix Multiplication
Another example

PATH $\in \text{AC}^1$

“Boolean” Matrix Multiplication

$Z = XY$: $z_{ij} = \lor_{k=1..n} (x_{ik} \land y_{kj})$
Another example

PATH ∈ AC¹

"Boolean" Matrix Multiplication

Z=XY: \( z_{ij} = \bigvee_{k=1..n} (x_{ik} \land y_{kj}) \)

AC⁰ circuit (OR gate with fan-in n, binary AND gates)
Another example

PATH ∈ AC¹

“Boolean” Matrix Multiplication

\[ Z = XY: z_{ij} = \bigvee_{k=1..n} (x_{ik} \land y_{kj}) \]

AC⁰ circuit (OR gate with fan-in n, binary AND gates)

If X adjacency matrix (with self-loops), \( X^t_{ij} = 1 \) iff path from i to j of length t or less
Another example

PATH ∈ AC¹

"Boolean" Matrix Multiplication

\[ Z=XY: z_{ij} = \bigvee_{k=1..n} (x_{ik} \land y_{kj}) \]

AC⁰ circuit (OR gate with fan-in n, binary AND gates)

If X adjacency matrix (with self-loops), \(X^t_{ij}=1\) iff path from i to j of length t or less

\(X^m_{ij}\) for \(m \geq n\) is the transitive closure
Another example

PATH ∈ AC¹

"Boolean" Matrix Multiplication

Z=XY: \( z_{ij} = \bigvee_{k=1..n} (x_{ik} \land y_{kj}) \)

AC⁰ circuit (OR gate with fan-in n, binary AND gates)

If X adjacency matrix (with self-loops), \( X^t_{ij} = 1 \) iff path from i to j of length t or less

\( X^m_{ij} \) for \( m \geq n \) is the transitive closure

O(log n) matrix multiplications to compute \( X^n_{ij} \)
Another example

PATH ∈ AC^1

"Boolean" Matrix Multiplication

Z = XY: \( z_{ij} = \bigvee_{k=1..n} (x_{ik} \wedge y_{kj}) \)

AC^0 circuit (OR gate with fan-in n, binary AND gates)

If X adjacency matrix (with self-loops), \( X^t_{ij} = 1 \) iff path from i to j of length t or less

\( X^m_{ij} \) for \( m \geq n \) is the transitive closure

O(log n) matrix multiplications to compute \( X^n_{ij} \)

Total depth O(log n)
$\text{NC}^1 \subseteq L$
$\textbf{NC}^1 \subseteq \textbf{L}$

- Generate circuit (implicitly) and evaluate
\[ \text{NC}^1 \subseteq L \]

- Generate circuit (implicitly) and evaluate
- cf. NC \subseteq P. But now, to conserve space, a recursive evaluation (rather than bottom-up).
\( \text{NC}^1 \subseteq \text{L} \)

- Generate circuit (implicitly) and evaluate

- cf. \( \text{NC} \subseteq \text{P} \). But now, to conserve space, a recursive evaluation (rather than bottom-up).

- For each gate, recursively evaluate each input wire
\textbf{NC}^1 \subseteq L

Generate circuit (implicitly) and evaluate

cf. NC \subseteq P. But now, to conserve space, a recursive evaluation (rather than bottom-up).

For each gate, recursively evaluate each input wire

Storage: A path to the current node, from the output node: since bounded fan-in, takes O(1) bits per node; since logspace uniform that is sufficient to compute the node id in logspace
\[ \text{Generate circuit (implicitly) and evaluate} \]

\[ \text{cf. } \text{NC} \subseteq \text{P}. \text{ But now, to conserve space, a recursive evaluation (rather than bottom-up).} \]

\[ \text{For each gate, recursively evaluate each input wire} \]

\[ \text{Storage: A path to the current node, from the output node: since bounded fan-in, takes } O(1) \text{ bits per node; since logspace uniform that is sufficient to compute the node id in logspace} \]

\[ \text{And at each node along the path, the input wire values evaluated results so far (again } O(1) \text{ bits per node)} \]
\[ \text{NC}^1 \subseteq \text{L} \]

Generate circuit (implicitly) and evaluate

- cf. NC \(\subseteq\) P. But now, to conserve space, a recursive evaluation (rather than bottom-up).

For each gate, recursively evaluate each input wire

- Storage: A path to the current node, from the output node: since bounded fan-in, takes \(O(1)\) bits per node; since logspace uniform that is sufficient to compute the node id in logspace

- And at each node along the path, the input wire values evaluated results so far (again \(O(1)\) bits per node)

- Length of path = depth of circuit = \(O(\log n)\)
NL ⊆ AC$^1$
$$\text{NL} \subseteq \text{AC}^1$$

Recall $\text{PATH} \in \text{AC}^1$
\( \text{NL} \subseteq \text{AC}^1 \)

- Recall PATH \( \in \text{AC}^1 \)
- Also recall PATH is NL-complete
NL ⊆ AC¹

Recall PATH ∈ AC¹

Also recall PATH is NL-complete

with respect to log-space reductions
\[ \text{NL} \subseteq \text{AC}^1 \]

- Recall \( \text{PATH} \in \text{AC}^1 \)
- Also recall \( \text{PATH} \) is NL-complete
  - with respect to log-space reductions
  - in fact, with respect to NC\(^1\) reductions
\( \text{NL} \subseteq \text{AC}^1 \)

- Recall \( \text{PATH} \in \text{AC}^1 \)
- Also recall \( \text{PATH} \) is \( \text{NL} \)-complete
  - with respect to log-space reductions
  - in fact, with respect to \( \text{NC}^1 \) reductions
- For \( \text{NL} \) machine \( M \), can build (in log-space) an \( \text{NC}^1 \) circuit which on input \( x \), outputs \( (i,j)^{th} \) entry of the adjacency matrix of configuration graph of \( M(x) \). (Configuration 0, the start configuration depends on \( x \).)
Recall $\text{PATH} \in \text{AC}^1$

Also recall $\text{PATH}$ is NL-complete

with respect to log-space reductions

in fact, with respect to $\text{NC}^1$ reductions

For NL machine $M$, can build (in log-space) an $\text{NC}^1$ circuit which on input $x$, outputs $(i,j)^{th}$ entry of the adjacency matrix of configuration graph of $M(x)$. (Configuration 0, the start configuration depends on $x$.)

Combining the $\text{NC}^1$ circuits for reduction and the $\text{AC}^1$ circuit for $\text{PATH}$, we get an $\text{AC}^1$ circuit
Summary: \( NC^i \) and \( AC^i \)
Summary: $\text{NC}^i$ and $\text{AC}^i$

$\text{NC}^i \subseteq \text{AC}^i \subseteq \text{NC}^{i+1} \subseteq \text{NC} = \text{AC} \subseteq \text{P}$
Summary: \( NC^i \) and \( AC^i \)

- \( NC^i \subseteq AC^i \subseteq NC^{i+1} \subseteq NC = AC \subseteq P \)
- \( NC^0 \not\subseteq AC^0 \not\subseteq NC^1 \subseteq L \subseteq NL \subseteq AC^1 \)
Summary: $\text{NC}^i$ and $\text{AC}^i$

- $\text{NC}^i \subseteq \text{AC}^i \subseteq \text{NC}^{i+1} \subseteq \text{NC} = \text{AC} \subseteq \mathbb{P}$
- $\text{NC}^0 \subsetneq \text{AC}^0 \subsetneq \text{NC}^1 \subseteq \text{L} \subseteq \text{NL} \subseteq \text{AC}^1$
- $\text{AC}^0 \subsetneq \text{NC}^1$ as $\text{PARITY} \not\in \text{AC}^0$ (later)
Summary: $NC^i$ and $AC^i$

- $NC^i \subseteq AC^i \subseteq NC^{i+1} \subseteq NC = AC \subseteq P$
- $NC^0 \varsubsetneq AC^0 \varsubsetneq NC^1 \subseteq L \subseteq NL \subseteq AC^1$
- $AC^0 \varsubsetneq NC^1$ as $PARITY \not\in AC^0$ (later)
- Open: whether $NC^i \varsubsetneq AC^i \varsubsetneq NC^{i+1}$ for larger $i$
Summary: $\text{NC}^i$ and $\text{AC}^i$

- $\text{NC}^i \subseteq \text{AC}^i \subseteq \text{NC}^{i+1} \subseteq \text{NC} = \text{AC} \subseteq \text{P}$
- $\text{NC}^0 \not\subseteq \text{AC}^0 \not\subseteq \text{NC}^1 \subseteq \text{L} \subseteq \text{NL} \subseteq \text{AC}^1$
  - $\text{AC}^0 \not\subseteq \text{NC}^1$ as $\text{PARITY} \not\in \text{AC}^0$ (later)

- Open: whether $\text{NC}^i \not\subseteq \text{AC}^i \not\subseteq \text{NC}^{i+1}$ for larger $i$

- Open: Is $\text{NC} = \text{P}$? (Can all polynomial time decidable languages be sped up to poly-log time using parallelization?)
Zoo
DC Uniform
DC Uniform

_topic_ Recall Uniform circuit family: circuits in the family can be generated by a TM
DC Uniform

- Recall Uniform circuit family: circuits in the family can be generated by a TM.
- Suppose circuits are super-polynomially large. Cannot be logspace-uniform or P-uniform.
DC Uniform

Recall Uniform circuit family: circuits in the family can be generated by a TM

Suppose circuits are super-polynomially large. Cannot be logspace-uniform or P-uniform.

DC uniform allows exponentially large circuits
DC Uniform

- Recall Uniform circuit family: circuits in the family can be generated by a TM
- Suppose circuits are super-polynomially large. Cannot be logspace-uniform or P-uniform.
  - DC uniform allows exponentially large circuits
  - Still requires polynomial time implicit computation of the circuit
DC Uniform

- Recall Uniform circuit family: circuits in the family can be generated by a TM

- Suppose circuits are super-polynomially large. Cannot be logspace-uniform or P-uniform.

  - DC uniform allows exponentially large circuits

  - Still requires polynomial time implicit computation of the circuit

- Coincides with EXP (Why?)
O(1) depth DC Uniform
O(1) depth DC Uniform

Restricted to constant depth, $2^{\text{poly}(n)}$ size, unbounded fan-in DC uniform circuit families decide exactly languages in PH.
Restricted to constant depth, $2^{\text{poly}(n)}$ size, unbounded fan-in DC uniform circuit families decide exactly languages in PH.

Given a DC uniform circuit (w.l.o.g alternating levels of AND and OR gates, and NOT gates only at the input level) of depth $k$, an equivalent quantified expression with $k$ alternations.
O(1) depth DC Uniform

- Restricted to constant depth, $2^{\text{poly}(n)}$ size, unbounded fan-in
  DC uniform circuit families decide exactly languages in PH

- Given a DC uniform circuit (w.l.o.g alternating levels of AND and OR gates, and NOT gates only at the input level) of depth $k$, an equivalent quantified expression with $k$ alternations

- Given a quantified expression with $k$ alternations, an equivalent DC uniform circuit of depth $k+2$
O(1) depth DC Uniform
O(1) depth DC Uniform

From circuit to quantified expression
O(1) depth DC Uniform

- From circuit to quantified expression

- Consider game played on the circuit: adversary picks an input edge to the AND level and Alice picks an input edge to the OR level, going through levels top to bottom
O(1) depth DC Uniform

- From circuit to quantified expression

Consider game played on the circuit: adversary picks an input edge to the AND level and Alice picks an input edge to the OR level, going through levels top to bottom.

Alice wins if adversary “breaks off the path” (by picking either a non-wire edge or a wire not continuing the path), or if the path terminates at literal of value 1 (w/o breaking).
O(1) depth DC Uniform

- From circuit to quantified expression

Consider game played on the circuit: adversary picks an input edge to the AND level and Alice picks an input edge to the OR level, going through levels top to bottom.

Can check in poly time

Alice wins if adversary “breaks off the path” (by picking either a non-wire edge or a wire not continuing the path), or if the path terminates at literal of value 1 (w/o breaking)
O(1) depth DC Uniform

- From circuit to quantified expression

Consider game played on the circuit: adversary picks an input edge to the AND level and Alice picks an input edge to the OR level, going through levels top to bottom.

Alice wins if adversary “breaks off the path” (by picking either a non-wire edge or a wire not continuing the path), or if the path terminates at literal of value 1 (w/o breaking).

Input accepted by the circuit iff Alice has a winning strategy (i.e., if the quantified expression is true)
**O(1) depth DC Uniform**

- From circuit to quantified expression

Consider game played on the circuit: adversary picks an input edge to the AND level and Alice picks an input edge to the OR level, going through levels top to bottom.

Alice wins if adversary “breaks off the path” (by picking either a non-wire edge or a wire not continuing the path), or if the path terminates at literal of value 1 (w/o breaking).

Input accepted by the circuit iff Alice has a winning strategy (i.e., if the quantified expression is true).

Each edge has a polynomially long label, and quantified variables take values from the same domain. Checking if edge is a correct wire in poly time (uniformity).
O(1) depth DC Uniform
O(1) depth DC Uniform

From quantified expression to circuit:
O(1) depth DC Uniform

- From quantified expression to circuit:
  - Circuit has depth-2 sub-circuits evaluating the poly-time condition for each possible assignment of the quantified variables.
O(1) depth DC Uniform

- From quantified expression to circuit:
  - Circuit has depth-2 sub-circuits evaluating the poly-time condition for each possible assignment of the quantified variables.
  - Hang these sub-circuits at the leaves of a k-level AND-OR tree appropriately.
O(1) depth DC Uniform

From quantified expression to circuit:

- Circuit has depth-2 sub-circuits evaluating the poly-time condition for each possible assignment of the quantified variables.

- Hang these sub-circuits at the leaves of a k-level AND-OR tree appropriately.

- Circuit can be implicitly computed in polynomial time. Size $2^{O(\text{total length of variables})}$
Today
Today

$NC^i$ and $AC^i$
Today

- $\text{NC}^i$ and $\text{AC}^i$
- DC-uniform
Today

- $\text{NC}^i$ and $\text{AC}^i$
- DC-uniform
- PH and EXP
Today

- NC\textsuperscript{i} and AC\textsuperscript{i}
- DC-uniform
- PH and EXP
- Later, more circuits and non-uniform computation (time permitting)
Today

- $\text{NC}^i$ and $\text{AC}^i$
- DC-uniform
- PH and EXP
- Later, more circuits and non-uniform computation (time permitting)
- $\text{PARITY} \notin \text{AC}^0$
Today

- $\text{NC}^i$ and $\text{AC}^i$
- DC-uniform
- PH and EXP
- Later, more circuits and non-uniform computation (time permitting)
- $\text{PARITY} \notin \text{AC}^0$
- Decision trees, Branching programs
Today

- $\text{NC}^i$ and $\text{AC}^i$
- DC-uniform
- PH and EXP

Later, more circuits and non-uniform computation (time permitting)

- $\text{PARITY} \not\in \text{AC}^0$
- Decision trees, Branching programs
- Connections between circuit lowerbounds and other complexity class separations