CS 301

Lecture 26 – Conclusion

Stephen Checkoway

May 2, 2018

What's this class good for anyway?

- Thinking logically will help any time you want to make an argument

What's this class good for anyway?

- Thinking logically will help any time you want to make an argument
- Regular expressions are incredibly useful; learn to use them in your favorite programming language (and when not to use them)

What's this class good for anyway?

- Thinking logically will help any time you want to make an argument
- Regular expressions are incredibly useful; learn to use them in your favorite programming language (and when not to use them)
- Context-free grammars are important for programming languages and compilers

What's this class good for anyway?

- Thinking logically will help any time you want to make an argument
- Regular expressions are incredibly useful; learn to use them in your favorite programming language (and when not to use them)
- Context-free grammars are important for programming languages and compilers
- Decidability helps you think about what problems you cannot solve with computers

What's this class good for anyway?

- Thinking logically will help any time you want to make an argument
- Regular expressions are incredibly useful; learn to use them in your favorite programming language (and when not to use them)
- Context-free grammars are important for programming languages and compilers
- Decidability helps you think about what problems you cannot solve with computers
- Complexity helps you think about what problems you can solve or verify quickly

So what's next?

- Algorithms! It turns out models of computation really do matter; computers aren't Turing machines; $O(n \log n)$ is great, $O(n^2)$ can be too slow to use

So what's next?

- Algorithms! It turns out models of computation really do matter; computers aren't Turing machines; $O(n \log n)$ is great, $O(n^2)$ can be too slow to use
- Sometimes, $O(n^2)$ works great and an equivalent $O(n \log n)$ algorithm takes longer (small inputs)

So what's next?

- Algorithms! It turns out models of computation really do matter; computers aren't Turing machines; $O(n \log n)$ is great, $O(n^2)$ can be too slow to use
- Sometimes, $O(n^2)$ works great and an equivalent $O(n \log n)$ algorithm takes longer (small inputs)
- More computability!

So what's next?

- Algorithms! It turns out models of computation really do matter; computers aren't Turing machines; $O(n \log n)$ is great, $O(n^2)$ can be too slow to use
- Sometimes, $O(n^2)$ works great and an equivalent $O(n \log n)$ algorithm takes longer (small inputs)
- More computability!
- More complexity! We've only scratched the surface

More computability and complexity?

- What if we give our TM access to an "oracle" that in a single step can decide a language like A_{TM}, what languages can you decide with this new capability?

More computability and complexity?

- What if we give our TM access to an "oracle" that in a single step can decide a language like A_{TM}, what languages can you decide with this new capability?
- Time bounds aren't the only option, we can consider space constraints too

More computability and complexity?

- What if we give our TM access to an "oracle" that in a single step can decide a language like A_{TM}, what languages can you decide with this new capability?
- Time bounds aren't the only option, we can consider space constraints too
- What languages can you decide if you use a polynomial amount of space? PSPACE

More computability and complexity?

- What if we give our TM access to an "oracle" that in a single step can decide a language like A_{TM}, what languages can you decide with this new capability?
- Time bounds aren't the only option, we can consider space constraints too
- What languages can you decide if you use a polynomial amount of space? PSPACE
- What languages can you decide with a nondeterministic TM in a polynomial amount of space? $\mathrm{NSPACE}(f(n)) \subseteq \mathrm{SPACE}(f^2(n))$ (Savitch's theorem); compare to time where there's an exponential blow up

More computability and complexity?

- What if we give our TM access to an "oracle" that in a single step can decide a language like A_{TM}, what languages can you decide with this new capability?
- Time bounds aren't the only option, we can consider space constraints too
- What languages can you decide if you use a polynomial amount of space? PSPACE
- What languages can you decide with a nondeterministic TM in a polynomial amount of space? $\mathrm{NSPACE}(f(n)) \subseteq \mathrm{SPACE}(f^2(n))$ (Savitch's theorem); compare to time where there's an exponential blow up
- What if you have a read-only input and a logarithmic amount of space to work with? (Effectively, you have a constant number of pointers) L

More computability and complexity?

- What if we give our TM access to an "oracle" that in a single step can decide a language like A_{TM}, what languages can you decide with this new capability?
- Time bounds aren't the only option, we can consider space constraints too
- What languages can you decide if you use a polynomial amount of space? PSPACE
- What languages can you decide with a nondeterministic TM in a polynomial amount of space? $\mathrm{NSPACE}(f(n)) \subseteq \mathrm{SPACE}(f^2(n))$ (Savitch's theorem); compare to time where there's an exponential blow up
- What if you have a read-only input and a logarithmic amount of space to work with? (Effectively, you have a constant number of pointers) L
- Same thing but with nondeterminism. $\mathrm{NL} = \mathrm{co\text{-}NL}$ (bizarre!)

More computability and complexity?

- What if we give our TM access to an "oracle" that in a single step can decide a language like A_{TM}, what languages can you decide with this new capability?
- Time bounds aren't the only option, we can consider space constraints too
- What languages can you decide if you use a polynomial amount of space? PSPACE
- What languages can you decide with a nondeterministic TM in a polynomial amount of space? $\mathrm{NSPACE}(f(n)) \subseteq \mathrm{SPACE}(f^2(n))$ (Savitch's theorem); compare to time where there's an exponential blow up
- What if you have a read-only input and a logarithmic amount of space to work with? (Effectively, you have a constant number of pointers) L
- Same thing but with nondeterminism. $\mathrm{NL} = \mathrm{co\text{-}NL}$ (bizarre!)
- What if you give the TM access to randomness and allow the TM to be wrong sometimes?

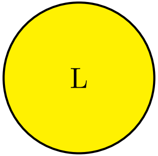
More computability and complexity?

- What if we give our TM access to an "oracle" that in a single step can decide a language like A_{TM}, what languages can you decide with this new capability?
- Time bounds aren't the only option, we can consider space constraints too
- What languages can you decide if you use a polynomial amount of space? PSPACE
- What languages can you decide with a nondeterministic TM in a polynomial amount of space? $\mathrm{NSPACE}(f(n)) \subseteq \mathrm{SPACE}(f^2(n))$ (Savitch's theorem); compare to time where there's an exponential blow up
- What if you have a read-only input and a logarithmic amount of space to work with? (Effectively, you have a constant number of pointers) L
- Same thing but with nondeterminism. $\mathrm{NL} = \mathrm{co\text{-}NL}$ (bizarre!)
- What if you give the TM access to randomness and allow the TM to be wrong sometimes?
- What if you require it give the right answer and "on average" takes polynomial time but on some inputs can take more?

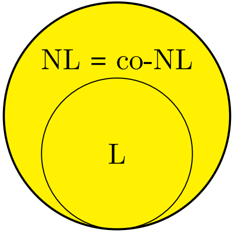
More computability and complexity?

- What if we give our TM access to an "oracle" that in a single step can decide a language like A_{TM}, what languages can you decide with this new capability?
- Time bounds aren't the only option, we can consider space constraints too
- What languages can you decide if you use a polynomial amount of space? PSPACE
- What languages can you decide with a nondeterministic TM in a polynomial amount of space? $\mathrm{NSPACE}(f(n)) \subseteq \mathrm{SPACE}(f^2(n))$ (Savitch's theorem); compare to time where there's an exponential blow up
- What if you have a read-only input and a logarithmic amount of space to work with? (Effectively, you have a constant number of pointers) L
- Same thing but with nondeterminism. $\mathrm{NL} = \mathrm{co\text{-}NL}$ (bizarre!)
- What if you give the TM access to randomness and allow the TM to be wrong sometimes?
- What if you require it give the right answer and "on average" takes polynomial time but on some inputs can take more?
- What if instead of TMs, you have circuits?

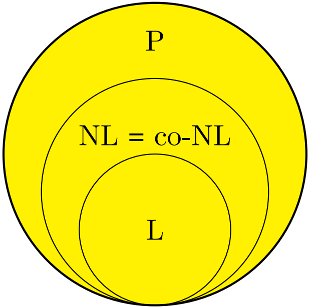
Complexity "zoo"



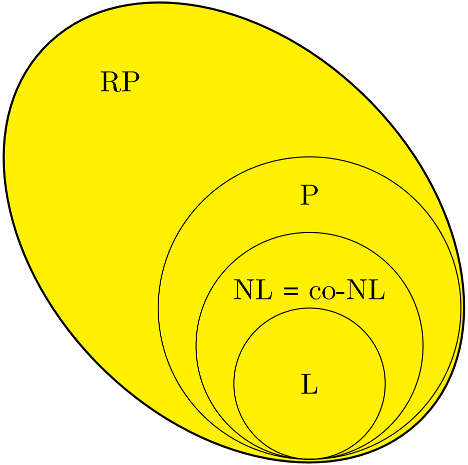
Complexity "zoo"



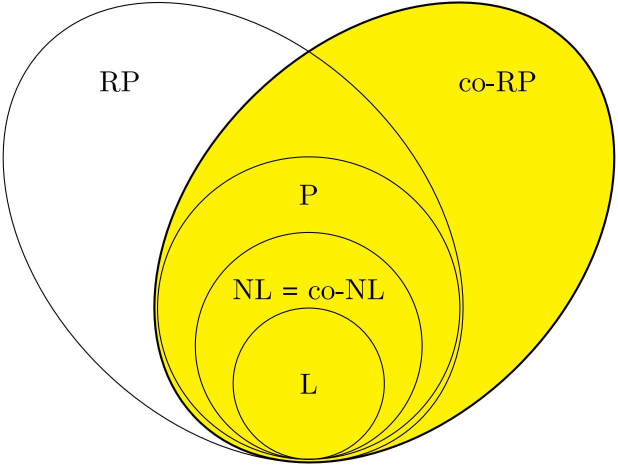
Complexity "zoo"



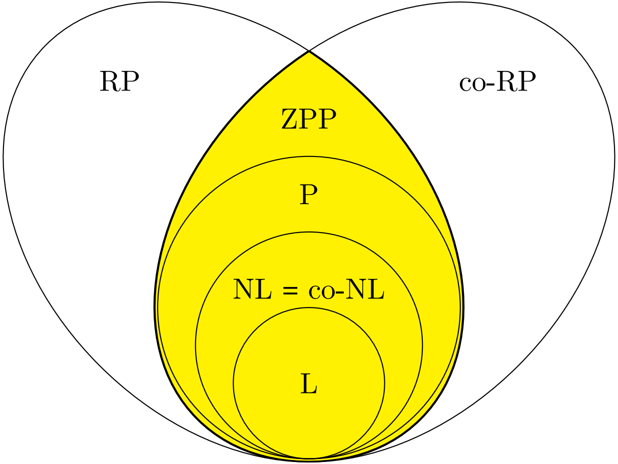
Complexity “zoo”



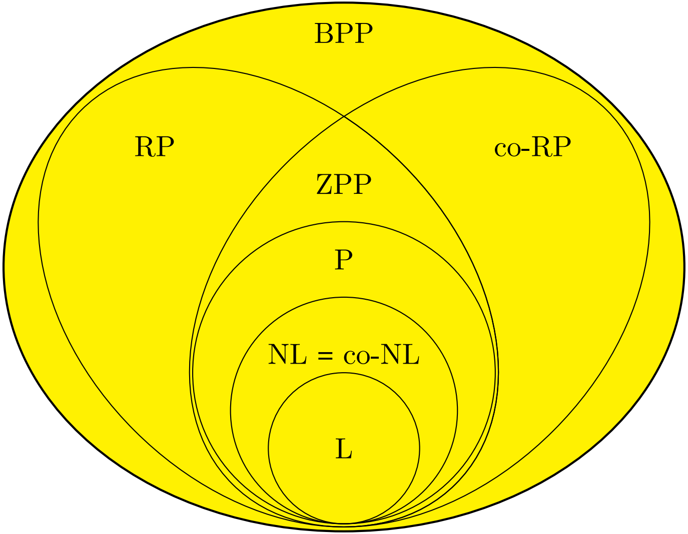
Complexity "zoo"



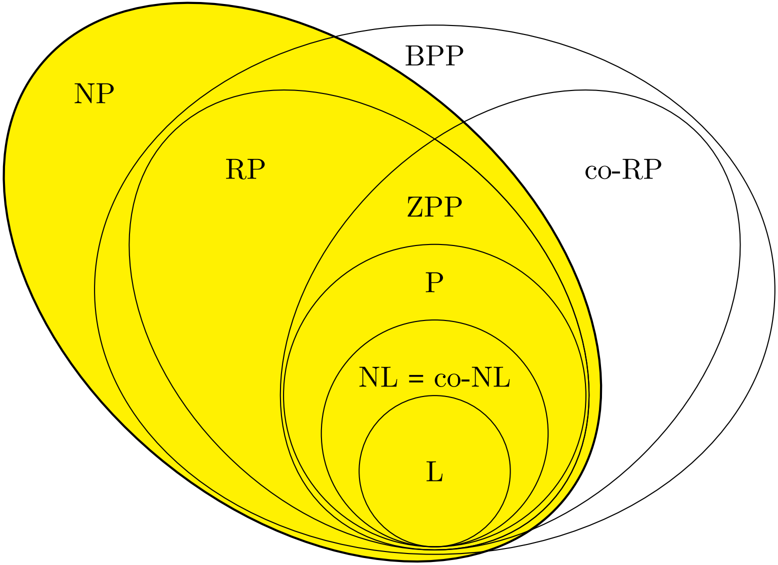
Complexity "zoo"



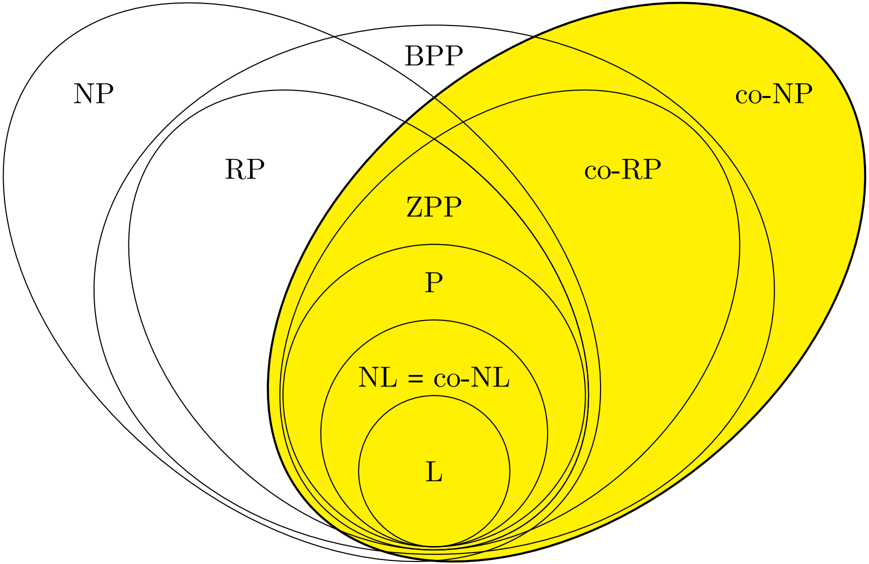
Complexity "zoo"



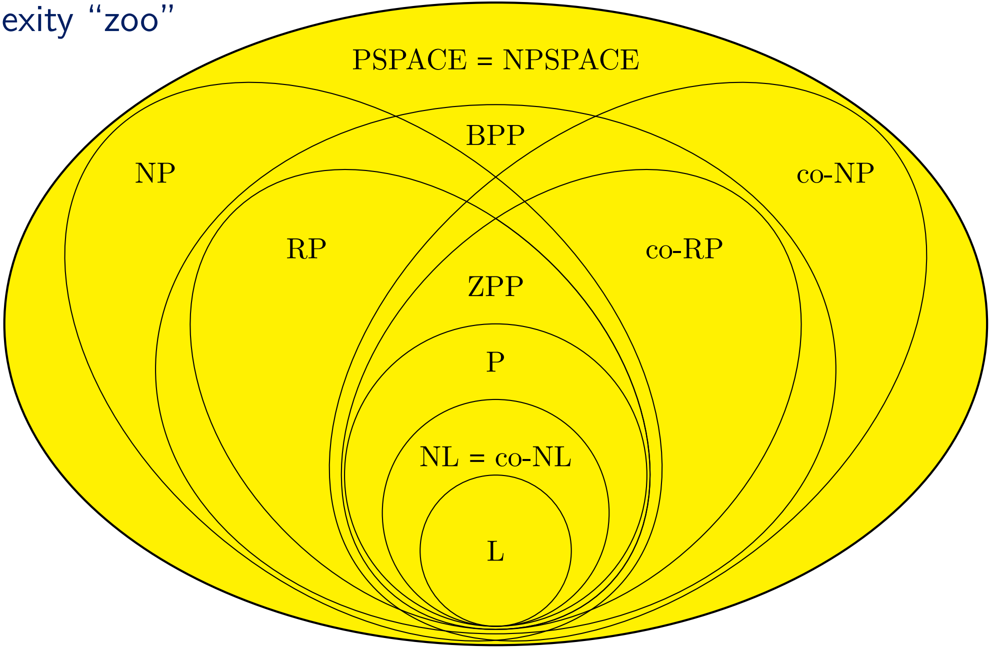
Complexity "zoo"



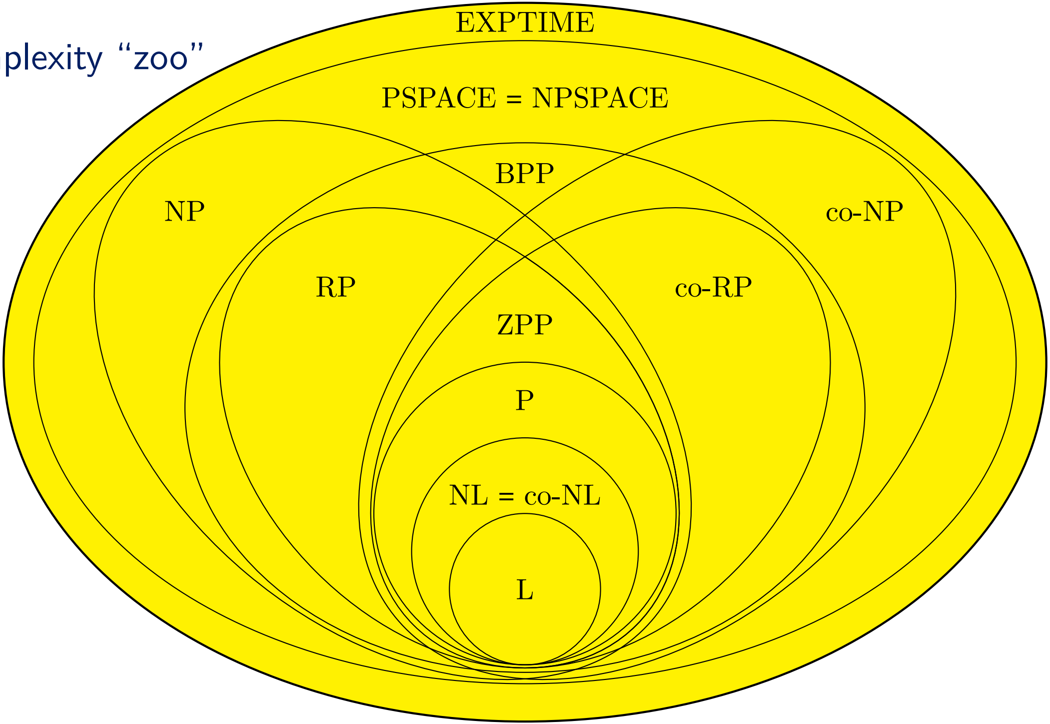
Complexity “zoo”



Complexity "zoo"



Complexity "zoo"



Exam topics

Broadly speaking: Everything through today

- Regular languages, context-free languages
- DFAs, NFAs, regular expressions, CFGs, PDAs, TMs
- Conversions between the various machines, grammars, and expressions (where doable).
- Converting a CFG to CNF
- Closure properties of regular, context-free, decidable, and Turing-recognizable languages
- Decision problems from language theory (e.g., A_{DFA}, EQ_{TM}, ALL_{CFG})
- Mapping reductions
- Polynomial time mapping reductions
- P, NP, $\mathrm{EXPTIME}$
- What it means for a language to be NP-complete

Types of exam questions

The questions from the exam fall into these types

- True/false questions with explanation
- Constructions
- Proofs
- **One extra credit problem**

Exam question break down (probably; the exam is still being written)

- Five true/false questions (4 points each)
- Two constructions (20 points each)
- Four proofs (20 points, 15 points, 15 points, 20 points)
- Extra credit (20 points, no partial credit)

Things that won't be on the exam

- Pumping lemma for context-free languages questions
- Proving that a particular language is NP-complete (you may be asked to prove that under some assumptions, some language is NP-complete, but you won't be asked to give a polynomial time reduction)

Examples

1. Regular languages are closed under perfect shuffle

$$\{a_1 b_1 a_2 b_2 \cdots a_n b_n \mid \text{each } a_i, b_i \in \Sigma,\ a_1 a_2 \cdots a_n \in A \text{ and } b_1 b_2 \cdots b_n \in B\}$$

2. Turing-recognizable languages are closed under intersection
3. Prove that if $A \le_{\mathrm{p}} B$ and $B \le_{\mathrm{p}} C$, then $A \le_{\mathrm{p}} C$
4. Convert a CFG to a PDA
5. $\textsc{Composites} = \{\langle n \rangle \mid n > 0 \text{ is a composite integer}\} \in \mathrm{NP}$
6. Any others you want me to do