Algebrization

Patrick Norton

December 4, 2024

Oracle

“Magic” function: can solve some problem in a single step

Can be any function, but we care when the function is hard to compute

Relativization

Statement about *theorems*, not classes or problems

$\mathcal{C} \subseteq \mathcal{D}$ algebrizes:

$$\mathcal{C}^A \subseteq \mathcal{D}^{\tilde{A}}$$

$\mathcal{C} \not\subseteq \mathcal{D}$ algebrizes:

$$\mathcal{C}^{\tilde{A}} \not\subseteq \mathcal{D}^A$$

for all oracles A and extensions $\tilde{A}$

Relativizing techniques

Not just theorems, also techniques!

Relativizes when logic is still valid

Replace $\mathcal{C}$ with $\mathcal{C}^A$ everywhere

Diagonalization

Goal: construct a language L not in some class $\mathcal{C}$

Idea: go through every machine in $\mathcal{C}$ and make sure it's incorrect for some string in L

Relativizing technique!

$P \neq NP$ does not relativize

- Idea: find an oracle powerful enough to make the difference between P and NP disappear
- If our oracle A is PSPACE-complete, $P^A = PSPACE = NP^A$

$\mathsf{P} = \mathsf{NP}$ does not relativize

Time to diagonalize!

Goal: construct a language L such that for some oracle A, $L \in \mathsf{NP}^A \setminus \mathsf{P}^A$

Define

$$L(A) = \{x \in \{0,1\}^* \mid \text{there is } y \in A \text{ such that } |y| = |x|\}$$

$L(A) \in \mathrm{NP}^A$

NP: languages that can be verified in polynomial time
NP^A: NP but with access to oracle A
$L(A)$: all strings with the same length as something in A

Certificate: a string in A of length n

Verify: Pass certificate to oracle A and verify it outputs 1

Diagonalizing A: definitions

- Let $\{P_i\}$ be a sequence of all oracle machines in P
- For each P_i, let $p_i(n)$ be a polynomial upper bound on the runtime of P_i over all inputs of length n
- Next: construct A
- Do this by constructing a chain

$$A(1) \subseteq \cdots \subseteq A(i) \subseteq \cdots$$

- Each $A(i)$ corresponds to a machine P_i

Diagonalizing A

- For each P_i:
- Let $n_i > n_{i-1}$ such that $p_i(n_i) < 2^{n_i}$
- This ensures P_i will not query some string of length n_i
- Further, it ensures we don't modify anything a previous machine queries
- Run P_i on input 0^n with oracle $A(i-1)$
- Now, we do different things depending on whether P_i accepts

If P_i accepts

- We want to make sure that there are no strings of length n_i in A
- Hence, we say $A(i) = A(i-1)$ and thus add nothing new

If P_i rejects

- This is the tricker half
- Since $p_i(n_i) < 2^{n_i}$, we know there is some string x of length n_i that P_i never queried
- Hence, if we add that string to $A(i)$, it won't affect the output of P_i
- So set $A(i) = A(i-1) \sqcup \{x\}$

Algebrization

- While relativization was enough to rule out all techniques at the time, new ones have come about since
- Let's rule out all these new ones too
- We can think of an oracle as being a collection of functions $f_n : \{0,1\}^n \to \{0,1\}$ for each n
- What if we extended f to be over a finite field?

Algebrization

- Ok, maybe that's a few too many functions
- So, let's restrict to polynomials over finite fields
- Formal definition: An extension $\tilde{A}$ of an oracle A is a collection of polynomials $\tilde{A}_{n,\mathbb{F}}$ such that $A_n(x) = \tilde{A}_{n,\mathbb{F}}(x)$ for all $x \in \{0,1\}^n$

Algebrizing results

Big difference: only one oracle gets extended

$\mathcal{C} \subseteq \mathcal{D}$ algebrizes:

$$\mathcal{C}^A \subseteq \mathcal{D}^{\tilde{A}}$$

$\mathcal{C} \not\subseteq \mathcal{D}$ algebrizes:

$$\mathcal{C}^{\tilde{A}} \not\subseteq \mathcal{D}^A$$

for all oracles A and extensions $\tilde{A}$

Algebrizing theorems

- Also like before, proof techniques can algebrize
- A proof technique algebrizes if the logic is still valid when you replace every class $\mathcal{C}$ with either $\mathcal{C}^A$ or $\mathcal{C}^{\tilde{A}}$, as appropriate

Why algebrization

Good news! This is still enough to cause problems

$P \neq NP$ does not algebrize

- Idea: same as for relativization, we just need to make sure $NP^{\tilde{A}} = PSPACE = P^{A}$
- From Babai, Fortnow, and Lund 1990, if A is PSPACE-complete, then so is $\tilde{A}$ so long as each polynomial in $\tilde{A}$ is multilinear
- So we know $NP^{\tilde{A}} = PSPACE$ and from earlier, we know $P^{A} = PSPACE$

$\mathsf{P} = \mathsf{NP}$ does not algebrize

- Diagonalization returns!
- Same as before: construct an A such that $L(A) \in \mathsf{NP}^A \setminus \mathsf{P}^{\tilde{A}}$
- What's different?
- We have to ensure queries "off the cube" don't break

$\mathsf{P} = \mathsf{NP}$ does not algebrize

Reuse definitions from before:

- P_i a sequence of P machines
- Polynomial p_i upper bound on running time of P_i
- n_i such that $p_i(n_i) < 2^{n_i}$

P = NP does not algebrize

- P_i a sequence of P machines
- Polynomial p_i upper bound on running time of P_i
- n_i such that $p_i(n_i) < 2^{n_i}$

Let $\mathcal{Y}_{\mathbb{F}}$ be the set of points queried by P_i for some field $\mathbb{F}$

The point w is the string we want to be 1

We want:

1. $\tilde{A}_{n,\mathbb{F}}(y) = 0$ for all $y \in \mathcal{Y}_{\mathbb{F}}$
2. $\tilde{A}_{n,\mathbb{F}}(w) = 1$
3. $\tilde{A}_{n,\mathbb{F}}(x) = 0$ for all $x \in \{0,1\}^n \setminus \{w\}$

References

Aaronson, Scott and Avi Wigderson (Feb. 2009). "Algebrization: A New Barrier in Complexity Theory". In: *ACM Trans. Comput. Theory* 1.1. ISSN: 1942-3454. DOI: 10.1145/1490270.1490272.

Babai, L., L. Fortnow, and C. Lund (1990). "Nondeterministic exponential time has two-prover interactive protocols". In: *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*, 16–25 vol.1. DOI: 10.1109/FSCS.1990.89520.