Let's now see the algorithmic method in action. We prove:

Theorem 1 ([Wil11]). *There is an algorithm that solves satisfiability for $\mathcal{ACC}_d$ circuits of size 2^{n^ϵ} in time 2^{n-n^ϵ}, where ϵ depends on d. In particular, the algorithm solves $\mathsf{CAPP}_{1,1/2}$ for such circuits.*

Corollary 2 ([Wil11; MW18]). $\mathcal{NTIME}[2^{\mathrm{polylog}(n)}] \not\subset \mathcal{ACC}$.

Note that this parameter setting is "in the middle" between the two parameter settings we've seen, and therefore it gives a lower bound in $\mathcal{NQP}$ (which is, again, "in the middle" between the two conclusions we've seen).

1 About $\mathcal{ACC}$ circuits

We first define the class and then motivate it.

Definition 3. *A circuit family is in $\mathcal{ACC}_{d,m}$ is all circuits are of depth d, with gates of unbounded fan-in over the basis $\{\wedge, \vee, \neg, \mathrm{mod}\, m\}$, where $\mathrm{mod}\, m(x_1, ..., x_n)$ is a Boolean gate (i.e., the output is 0 or 1, not in $\mathbb{Z}_m$) that outputs 1 iff the sum of its inputs is $0 \bmod m$.*

We identify the class of circuit families with the class of languages they decide, and also define $\mathcal{ACC} = \cup_{d,m} \mathcal{ACC}_{d,m}$. If we don't specify the size, we mean circuits of arbitrary polysize.

There are inherent motivations to study this class (e.g., it captures computation over solvable monoids [BT88]), but the main motivation comes from historical progress in complexity theory. Starting from $\mathcal{AC}^0$ – arguably a natural starting-point – we know that $\oplus$ is hard for this class [FSS84; Yao85; Hås87], so a natural question is what happens if we artificially allow the circuit to compute $\oplus$. We obtain $\mathcal{AC}^0[\oplus]$, for which MAJ is hard [Raz87; Smo87]. In fact, the proof expends to $\mathcal{AC}^0[q]$ for any prime power q (i.e., the gates compute the Boolean $\mathrm{mod}\, q$ function).

The known techniques break down when allowing m that is not a prime power. This is because they rely on approximations by low-degree polynomials over $\mathbb{F}_q$, but when working with a composite m, we now have a ring rather than a field. An open question is whether MAJ is hard for $\mathcal{ACC}$ – we still don't know the answer.

Example 4. *For concreteness, from now on we fix $m = 6$, the smallest non-prime composite. In fact, it will be easier to work with circuits over the basis $\{\wedge, \vee, \neg, \mathrm{mod}2, \mathrm{mod}3\}$, relying on the fact that $\mathrm{mod}6$ gates can be simulated by $\mathrm{mod}2$ and $\mathrm{mod}3$ gates and vice versa.*[1]

A structural result. Despite not having lower bounds for $\mathcal{ACC}$, we did have a structural result since the early 1990s, which simulates any $\mathcal{ACC}$ circuit by an object that is similar to a low-degree polynomial over $\mathbb{Z}$.

[1] Specifically $\mathrm{mod}6(x_1, ..., x_n) = \mathrm{mod}2(x_1, ..., x_n) \wedge \mathrm{mod}3(x_1, ..., x_n)$; and in the other direction, $\mathrm{mod}2(x_1, ..., x_n) = \mathrm{mod}6(x_1, x_1, x_1, ..., x_n, x_n, x_n)$ and $\mathrm{mod}3(x_1, ..., x_n) = \mathrm{mod}6(x_1, x_1, ..., x_n, x_n)$.

Definition 5. *A $\mathcal{SYM}^+$ circuit has a bottom layer of $\wedge$ gates, and a top gate computing a symmetric function (i.e. a function of the form $z_1, ..., z_m \to g(\sum_i z_i)$ for some $g\colon \mathbb{Z} \to \{0,1\}$ where the summation is over the integers).*

Note that a $\mathcal{SYM}^+$ gates computes $g\left(\sum_S x_S\right)$ where each x_S is a monomial (corresponding to the children of an $\wedge$ gate). In particular, if each $\wedge$ gate has small fan-in, then this is similar to a low-degree polynomial, where the main difference is that we add an arbitrary g at the end. We refer to the fan-in of $\wedge$ gates as the degree.

Lemma 6 ([Yao90; BT94; AG94])**.** *There is an algorithm that gets as input an $\mathcal{ACC}$ circuit of size s and outputs a functionally equivalent $\mathcal{SYM}^+$ circuit of degree* $\text{polylog}(s)$ *and size* $2^{\text{polylog}(s)}$.

Note that if the original size was $s = 2^{n^\epsilon}$, then the new size is $2^{n^{O(\epsilon)}}$. One might expect that this structural result would be enough to prove a lower bound on $\mathcal{ACC}$ using "combinatorial" methods, and this might still be true, but nobody knows how to do this (i.e., without algorithmic methods)

2 The algorithm

The main component of the algorithm is a batch-evaluation algorithm, which evaluates a $\mathcal{SYM}^+$ circuit of size S on all 2^n inputs in time $\approx 2^n + S$ instead of $\approx 2^n \cdot S$.

Lemma 7. *There is an algorithm that gets as input an n-bit $\mathcal{SYM}^+$ circuit of size S whose symmetric function can be computed in time* $\text{poly}(S)$*, and outputs the evaluations on all inputs in time* $(2^n + \text{poly}(S)) \cdot \text{poly}(n)$.

Proof. We'll use the following result:

Theorem 8 ([Cop82])**.** *There's an algorithm that multiplies an $N \times N^{.01}$ matrix by an $N^{.01} \times N$ matrix in time $\tilde{O}(N^2)$, where the entries are Boolean and the arithmetic is over the integers.*

We bipartition the variables into $X = x_1, ..., x_{n/2}$ and $Y = x_{n/2+1}, ..., x_n$. Consider a matrix A whose $N = 2^{n/2}$ rows correspond to all assignments in X and whose S columns correspond to the $\wedge$ gate of the circuit, where $A_{i,j} = 1$ iff the assignment to X doesn't falsify the j^{th} gate. Analogously, construct B whose columns are assignments to Y and rows are gates, and $B_{j,i} = 1$ iff the assignment doesn't falsify the j^{th} gate. Note that in $C = (A \cdot B)$, each cell corresponds to a full assignment to $x_1, ..., x_n$ and its value is the number of gates satisfied by this assignment.

Our algorithm builds A and B, each in time $\tilde{O}(N \cdot S)$, and runs the matrix multiplication algorithm, in time $\tilde{O}(N^2)$. The last step is to transform C into a matrix whose entries represent the truth-value of the circuit. The brute-force approach (computing the symmetric function on each entry) is too expensive. Instead, we prepare a size-S truth-table of the symmetric function in time $\text{poly}(S)$, and then go over all entries of C, and for each entry perform truth-table lookup, in total time $2^n \cdot \text{polylog}(S)$. ∎

Remark 9. *The algorithm described above works in the random access model (RAM), whereas most of complexity theory works in the multitape TM model (indeed, this is the model in which we proved all results so far, even if we did not explicitly state it). This doesn't matter when polynomial overheads are immaterial, but now polynomial overheads do matter. Nevertheless, this still doesn't harm the final result: by [GS89], we can simulate RAM in the multitape model in quasilinear* non-deterministic *time (in fact, we can even do so for non-deterministic RAM), and for the algorithmic method it is fine if the* CAPP *algorithm is non-deterministic.*

Remark 10. *An alternative algorithm that proves the lemma using dynamic programming (instead of matrix multiplication) appears in [Wil11].*

Final algorithm. Our strategy is to take an $\mathcal{ACC}$ circuit, transform it into a $\mathcal{SYM}^+$ circuit, and then evaluate the latter on all inputs (which, in particular, solves satisfiability). Assuming that the original circuit was of size $s = 2^{n^\epsilon}$, the $\mathcal{SYM}^+$ circuit will be of size $S = 2^{n^{O(\epsilon)}}$, and the algorithm runs in time approximately $2^n + \text{poly}(S)$.

This still isn't good enough, and the last missing observation is that we're paying a lot for the number of variables, but have room to spare in terms of size. So we'll trade off the number of variables for the size, with a simple trick. Given an initial circuit $C(x_1, ..., x_n)$, we create

$$C'(x_1, ..., x_{n-\ell}) = \bigvee_{x_{n-\ell+1}, ..., x_n} C(x_1, ..., x_n) \ .$$

Note that C' is satisfiable iff C is satisfiable, and that the number of variables of C' is $n - \ell$ but its size is $|C| \cdot 2^\ell$. Taking $\ell = n^\epsilon$, we now transform C' into a $\mathcal{SYM}^+$ circuit of size $S = 2^{n^{O(\epsilon)}}$ with $n - \ell$ variables, and run the batch-evaluation algorithm in time

$$(2^{n-\ell} \cdot \text{poly}(S)) \cdot \text{poly}(n) \leq \tilde{O}\left(2^{n-n^\epsilon}\right) \ .$$

References

[AG94] Eric Allender and Vivek Gore. "A uniform circuit lower bound for the permanent". In: *SIAM Journal on Computing* 23.5 (1994), pp. 1026–1049.

[BT88] David A. Mix Barrington and Denis Thérien. "Finite monoids and the fine structure of NC^1". In: *Journal of the ACM* 35.4 (1988), pp. 941–952.

[BT94] Richard Beigel and Jun Tarui. "On ACC". In: vol. 4. 4. 1994, pp. 350–366.

[Cop82] D. Coppersmith. "Rapid multiplication of rectangular matrices". In: *SIAM Journal on Computing* 11.3 (1982), pp. 467–471.

[FSS84] Merrick Furst, James B. Saxe, and Michael Sipser. "Parity, circuits, and the polynomial-time hierarchy". In: *Mathematical Systems Theory* 17.1 (1984), pp. 13–27.

[GS89] Yuri Gurevich and Saharon Shelah. "Nearly linear time". In: *Logic at Botik, Symposium on Logical Foundations of Computer Science*. Lecture Notes in Computer Science. 1989, pp. 108–118.

[Hås87] Johan Håstad. *Computational Limitations of Small-depth Circuits*. MIT Press, 1987.

[MW18] Cody Murray and R. Ryan Williams. "Circuit Lower Bounds for Nondeterministic Quasi-Polytime: An Easy Witness Lemma for NP and NQP". In: *Proc. 50th Annual ACM Symposium on Theory of Computing (STOC)*. 2018, pp. 890–901.

[Raz87] Alexander A. Razborov. "Lower bounds on the size of constant-depth networks over a complete basis with logical addition". In: *Mathematical Notes of the Academy of Science of the USSR* 41.4 (1987), pp. 333–338.

[Smo87] Roman Smolensky. "Algebraic Methods in the Theory of Lower Bounds for Boolean Circuit Complexity". In: *Proc. 19th Annual ACM Symposium on Theory of Computing (STOC)*. 1987, pp. 77–82.

[Wil11] Ryan Williams. "Non-uniform ACC circuit lower bounds". In: *Proc. 26th Annual IEEE Conference on Computational Complexity (CCC)*. 2011, pp. 115–125.

[Yao85] Andrew C-C. Yao. "Separating the Polynomial-time Hierarchy by Oracles". In: *Proc. 26th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 1985, pp. 1–10.

[Yao90] Andrew Chi-Chih Yao. "On *ACC* and Threshold Circuits". In: *Proc. 31st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 1990, pp. 619–627.