1 Counting

In a *counting problem*, we are interested in *how many* solutions to a given problem exist. We will be interested in problems described by NP-relations, for instance: How many perfect matching does a graph have? How many satisfying assignments does a boolean formula have?

Unlike decision problems, where the Turing Machine that solves the problem always outputs 0 or 1, in a counting problem the output can be any number. So it is common to represent counting problems as functions from $\{0,1\}^* \to \mathbb{N}$, where $\mathbb{N} = \{0, 1, 2, \dots\}$.

Definition 1. Let R be an NP-search relation (with polynomial bound p). The *counting problem* $\#R$ associated to R is the function

$$\#R(x) = |\{y \colon R(x, y) \text{ is true and } |y| = p(|x|)\}|.$$

The class #P consists of all counting problems associated to some NP-relation R.

Clearly #P should be at least as hard as NP: If we could count the number of satisfying assignment to a boolean formula, then we should know if one exists. To formalize this intuition, we cannot simply say that NP $\subseteq$ #P because NP is a class of decision problems, while #P is a class of counting problems. The easiest way to do so is via oracles.

To define an appropriate oracle for #P, we need a complete problem, and for this we need to define reductions between counting problems. For us, it will be adequate to say that counting problem f reduces to counting problem g if there is a polynomial-time reduction that maps instances x of f into instances y of g such that $g(y) = f(x)$. Under this definition, #SAT is NP-complete: If we examine the Cook-Levin reduction we notice that it preserves witnesses uniquely (when we reduce from L to SAT, every witness y for L maps into a unique witness for SAT), so in particular it preserves the number of witnesses. This kind of reduction that preserves the number of solutions is called a *parsimonious reduction*.

One way to state the fact that #P is at least as hard as NP is via the containment $\mathrm{P}^{\mathrm{SAT}} \subseteq \mathrm{P}^{\#\mathrm{SAT}}$. Could it be that they are equal? In 1991 Seinosuke Toda showed that this is quite implausible:

Theorem 2 (Toda's theorem)**.** *For all k, $\Sigma_k \subseteq \mathrm{P}^{\#\mathrm{SAT}}$.*

So an oracle for counting SAT-solutions would allow us to solve all problems in the polynomial-time hierarchy. On the other hand, it seems unlikely that an oracle for SAT would allow us the same, because if $\Sigma_k \subseteq \mathrm{P}^{\mathrm{SAT}}$ for all k, then the polynomial-time hierarchy would collapse to $\mathrm{P}^{\mathrm{SAT}}$.

There is another interesting difference between NP and #P. Recall the problem of finding perfect matchings in graphs. This problem is in P. However, its counting version is as hard as #SAT: $\mathrm{P}^{\#\mathrm{MATCHING}} = \mathrm{P}^{\#\mathrm{SAT}}$. Matching is a natural problem whose search version is easy, but its counting version is extremely hard.

2 Finding unique assignments

Before we analyze the complexity of counting, let's look at a warm-up problem. This will allow us to introduce two concepts that will play a role later: *promise problems* and *hash functions*.

Consider a version of SAT where you are promised the following: Either the formula has no satisfying assignment, or it has exactly one satisfying assignment. The goal is to tell which is the case. We will call this problem USAT for "unique boolean formula satisfiability." How hard is this problem? Clearly, USAT can be no harder than SAT, but is it really any easier?

The problem USAT is not a standard decision problem. Its instances are not arbitrary boolean formulas, but they are formulas with the special property that they have either zero or one satisfying assignment. (The question whether an instance satisfies this special property or not is in itself an NP-complete problem.) This kind of problem is known as a promise problem.

Formally, a *promise problem* is a function $P\colon \{0,1\}^* \to \{0,1,?\}$, where the ? symbol stands for "don't care". We say a Turing Machine M solves P if for every x such that $P(x) \neq ?$, $M(x) = P(x)$. A randomized Turing Machine M solves P if for every x such that $P(x) \neq ?$, $\Pr[M(x) = P(x)] \geq 2/3$.

In the case of USAT, we have $\mathrm{USAT}(\phi) = 1$ if ϕ has one satisfying assignment, $\mathrm{USAT}(\phi) = 0$ if ϕ is unsatisfiable, and $\mathrm{USAT}(\phi) = ?$ if ϕ has two or more satisfying assignments.

It turns out that if we allow the use of randomized algorithms, USAT is no easier than SAT:

Theorem 3 (Valiant and V. Vazirani)**.** *There is a randomized polynomial-time reduction R that maps CNF formulas ϕ (on n variables) to CNF formulas ϕ' such that*

$$\phi \in \mathrm{SAT} \Longrightarrow \Pr[\mathrm{USAT}(\phi') = 1] \geq 1/8n$$
$$\phi \notin \mathrm{SAT} \Longrightarrow \mathrm{USAT}(\phi') = 0.$$

From this theorem we see that $\mathrm{SAT} \in \mathrm{RP}^{\mathrm{USAT}}$: To tell if ϕ is in SAT given a USAT oracle, run the reduction from the theorem for $16n$ times, run the USAT oracle on all instances, accept if any one of them accepts, and reject otherwise.

We now turn to the proof of this theorem. Here is the idea of the proof. Suppose we knew that $\phi(x)$ had either 0 or s satisfying assignments ($1 \leq s \leq 2^n$). We try to design ϕ' in a way that isolates a single satisfying assignment of ϕ by imposing some extra condition on x. In case ϕ is unsatisfiable, it remains unsatisfiable.

How do we impose this extra condition? The idea is to use randomness: Suppose we selected a set $T \subseteq \{0,1\}^n$ by including each element of $\{0,1\}^n$ independently at random with probability $1/s$. Then the probability that exactly one satisfying assignment of ϕ makes it into T is $s \cdot (1/s) \cdot (1 - 1/s)^{s-1} \geq 1/e$.

So with probability at least $1/e$ the expression $\phi(x) \wedge (x \in T)$ has exactly one satisfying assignment. But there are two problems: First, we don't know the value s. Second, the above expression is not a CNF formula. The size of the set T might be very large and it is not clear how to write the expression as a CNF in x.

Let's deal with the second problem first. The idea is to derandomize the construction of T via *hash functions*. For a moment let's make the additional assumption that $s = 2^{i-1}$ for some i. A family of functions $H = \{h\colon \{0,1\}^n \to \{0,1\}^i\}$ is a *pairwise-independent hash function family* if for every pair $x, y \in \{0,1\}^n$, where $x \neq y$, the values $h(x)$ and $h(y)$ are uniformly distributed and pairwise independent when h is random. More precisely, for every $x, y \in \{0,1\}^n$, $x \neq y$, and $z, w \in \{0,1\}^i$:

$$\Pr[h(x) = z] = 2^{-i} \qquad \text{and} \qquad \Pr[h(x) = z \mid h(y) = w] = \Pr[h(x) = z].$$

One example of a pairwise independent family is the family $h_{A,b}(x) = xA + b$. Here we think of the input x as an n-dimensional vector with $0,1$ entries, A is a uniformly random $n \times i$ matrix with $0,1$ entries, b is a uniformly random vector in $\{0,1\}^n$, and all operations (vector-matrix multiplication and vector addition) are modulo 2.

Now we will replace the condition $x \in T$ above with the condition $h(x) = 0$ for a random $h \sim H$. We will show that the probability that $\phi(x) \wedge (h(x) = 0)$ only worsens from $1/e$ to $1/8$. Yet now the expression $\phi(x) \wedge (h(x) = 0)$ can be represented by a CNF formula $\phi'(x, z)$ with the same number of satisfying assignments. For example, if we use the above hash family $h_{A,b}(x) = Ax + b$, then the expression $\phi(x) \wedge (h(x) = 0)$ can be computed by a circuit of size $O(|\phi|n^2)$, which can then be converted into a CNF formula $\phi'(x, z)$ with the same number of satisfying assignments using the standard reduction from circuits to formulas.

It remains to deal with the problem of not knowing the value of s (and the assumption that $s = 2^i$). The solution is to guess at random a value of i so that s is approximately equal to 2^{i-1}, and everything will still work. Since there are only n possible values of i ($1 \leq i \leq n$), we guess the correct one with probability $1/n$, and the reduction succeeds in producing ϕ' with a unique assignment with probability at least $1/8n$.

Proof of Theorem 3. Consider the following reduction R: Choose a random $i \in [1, n+1]$ and let $h\colon \{0,1\}^n \to \{0,1\}^i$ be a random hash function from the above family H of pairwise independent hash function. Create the circuit

$$C(x) = \begin{cases} 1, & \text{if } \phi(x) \text{ is true and } h(x) = 0, \\ 0, & \text{otherwise} \end{cases}$$

transform it into a formula ϕ' with the same number of satisfying assignments, and output ϕ'.

Clearly, if ϕ is unsatisfiable, ϕ' will also be unsatisfiable. We now argue that if i satisfies the condition $2^{i-1} \geq |S| > 2^{i-2}$, then $\phi(x)$ is true and $h(x) = 0$ simultaneously for exactly one assignment x with probability at least $1/8$. Since i satisfies this condition with probability at least $1/n$, ϕ' has a unique satisfying assignment with probability at least $1/8n$.

Let S be the set of all x such that $\phi(x) = 0$. Then

$$\begin{aligned}
\Pr[\exists! x \in S\colon h(x) = 0] &= \sum_{x \in S} \Pr[h(x) = 0 \text{ and } h(x') \neq 0 \text{ for all } x' \neq x,\, x' \in S] \\
&= \sum_{x \in S} \Pr[h(x') \neq 0 \text{ for all } x' \neq x,\, x' \in S \mid h(x) = 0] \Pr[h(x) = 0] \\
&= \sum_{x \in S} (1 - \Pr[h(x') \neq 0 \text{ for some } x' \neq x,\, x' \in S \mid h(x) = 0]) \Pr[h(x) = 0] \\
&\geq \sum_{x \in S} \Big(1 - \sum_{x' \in S, x' \neq x} \Pr[h(x') \neq 0 \mid h(x) = 0]\Big) \Pr[h(x) = 0] \\
&= \frac{|S|}{2^i}\Big(1 - \frac{|S| - 1}{2^i}\Big) \geq \frac{1}{8},
\end{aligned}$$

as desired. □

3 Approximate counting

Since the problem of counting the number of assignments to NP-relations is quite hard, it is natural to ask if we can get an approximation.

Definition 4. Let R be an NP-relation. We say a Turing Machine A *approximately counts* $\#R$ if on input x and ε, A runs in time polynomial in x and ε and outputs a number such that

$$(1 - \varepsilon)\#R(x) \leq A(x, \varepsilon) \leq \#R(x).$$

For randomized Turing Machines, we require the condition to hold with probability $2/3$ over the randomness of A. Let $m = p(n)$ be the length of the witnesses in R. When $\varepsilon = 2^{-m}$, then approximate counting is the same as exact counting, but we allow A to run in exponential time. When ε is not as large, can we do better?

Suppose we have a Turing Machine A that approximately counts #SAT. Then $A(\phi, 1/2)$ always outputs 0 when ϕ is unsatisfiable, and some number larger than 0 when ϕ is satisfiable. Therefore, A can be used to solve SAT and all problems in $\mathrm{P}^{\mathrm{SAT}}$. However, unlike exact counting, approximate counting does not require much more power to implement:

Theorem 5. *There is a randomized oracle Turing Machine $A^?$ such that A^{SAT} approximately counts* #SAT*.*

Since all NP-relations have parsimonious reductions to SAT, approximate counting for any NP-relation can be done by a randomized algorithm with a SAT oracle. The proof of Theorem 5 follows by reduction to the following promise problem L:

Input: A boolean formula ϕ on n variables and a parameter $1 \leq i \leq n + 1$.
Yes instances: ϕ such that $2^{i-1} \geq \#\mathrm{SAT}(\phi) > 2^{i-2}$.
No instances: ϕ such that $\#\mathrm{SAT}(\phi) \leq 2^{i-4}$.

Claim 6. *There is a randomized oracle Turing Machine $M^?$ such that M^{SAT} solves the promise problem L.*

We now sketch how to use M^{SAT} for approximate counting as in Theorem 5. Given $M^?$, we design $A^?$ in two stages. First, we give an algorithm that works only for large values of ε (say, $\varepsilon = 7/8$). Then we show how to extend the algorithm to also work for small values of ε.

Let's first pretend that M^{SAT} is deterministic and show how to get a deterministic A^{SAT}. Consider the following algorithm:

$A^?$: On input ϕ with n variables and $\varepsilon \geq 7/8$,
    For $i := n + 1$ down to 1
        If $M^?(\phi, i)$ accepts, output 2^{i-4} and halt.
    Ouptut 0.

Let's analyze A^{SAT}. If ϕ is unsatisfiable, M^{SAT} will reject in all iterations, so the output is 0. Now assume ϕ is satisfiable. Notice that as long as $2^{i-4} \geq \#\mathrm{SAT}(\phi)$, M^{SAT} won't accept, so $\#\mathrm{SAT}(\phi) > A^{\mathrm{SAT}}(\phi, 7/8)$. On the other hand, $M^?$ must have accepted by the time that $2^{i-1} \geq \#\mathrm{SAT}(\phi) > 2^{i-2}$, so $\frac{1}{8}\#\mathrm{SAT}(\phi) \leq A^{\mathrm{SAT}}(\phi, 7/8)$.

Now let's describe how to handle smaller values of ε: On input ϕ, we create the formula $\psi = \phi_1 \wedge \cdots \wedge \phi_k$, where $\phi_1, \ldots, \phi_k$ are identical copies of ϕ, but over disjoint sets of variables. Then $\#\mathrm{SAT}(\psi) = \#\mathrm{SAT}(\phi)^k$, so $A^{\mathrm{SAT}}(\psi, 7/8)$ is between $(7/8)\#\mathrm{SAT}(\phi)^k$ and $\#\mathrm{SAT}(\phi)$. This suggests the following procedure for $A^{\mathrm{SAT}}(\phi, \varepsilon)$ when $\varepsilon \leq 7/8$: Set $k = 3/\varepsilon$, create ψ, run $A^{\mathrm{SAT}}(\psi, 7/8)$ and output the kth root of the answer. Then

$$\left(\tfrac{1}{8}\right)^{1/k}\#\mathrm{SAT}(\phi) \leq A^{\mathrm{SAT}}(\phi, \varepsilon) \leq \#\mathrm{SAT}(\phi)$$

For $k = 3/\varepsilon$, $(1/8)^{1/k} \geq 1 - \varepsilon$ and we get the desired answer.

When $M^?$ is a randomized Turing Machine, the same analysis works, provided we begin by amplifying the success probability of $M^?$ (to say $1/3n$) so that the total probability of error in some invocation of M^{SAT} never adds up to more than $1/3$.

It remains to prove Claim 6. The proof is quite similar to the proof of Theorem 3.

Proof of Claim 6. Let H be a pairwise-independent hash function family from $\{0,1\}^n$ to $\{0,1\}^i$. The Turing Machine A^{SAT} does the following:

1. Choose a random h from H.
2. Create a boolean formula ψ such that ψ is satisfiable if and only if $\phi(x)$ is true and $h(x) = 0$ for some x.
3. Ask the oracle if ψ is satisfiable and output its answer.

Let S be the set of satisfying assignments of ϕ. Let's argue about the no instances first. First assume $|S| = \#\mathrm{SAT}(\phi) \leq 2^{i-4}$. Then

$$\Pr[\exists x \in S \colon h(x) = 0] \leq \sum_{x \in S} \Pr[h(x) = 0] = \sum_{x \in S} 2^{-i} \leq 2^{i-4} \cdot 2^{-i} = \frac{1}{16}.$$

If $2^{i-1} \leq |S| < 2^{i-2}$, then by the same proof as in Theorem 3,

$$\Pr[\exists! x \in S \colon h(x) = 0] \geq \frac{1}{8}.$$

Therefore A^{SAT} accepts yes instances with probability at least $1/8$, and accepts no instances with probability at most $1/16$. Amplifying the success probability to $2/3$, we obtain a randomized Turing Machine for L. □

4 Proof of Toda's Theorem

he starting point for the proof of Toda's theorem is Theorem 3. By this theorem if we had a way to tell whether a formula had one satisfying assignment or zero satisfying assignmnets, then, using randomness, we could solve NP complete problems.

Now suppose that we could tell if a formula had an even number or an odd number of satisfying assignments. In particular we can then tell one from zero satisfying assignmnets, so we can solve everything in NP. In fact we can now solve everything in the polynomial hierarchy.

Lemma 7. *For every k there exists a randomized polynomial-time algorithm R that on input a quantified boolean formula ϕ with k alternations produces an unquantified boolean formula ψ such that*

$$\phi \in \exists_k \mathrm{SAT} \longrightarrow \Pr[\#\mathrm{SAT}(\psi) \textit{ is odd}] \geq 2/3$$
$$\phi \notin \exists_k \mathrm{SAT} \longrightarrow \#\mathrm{SAT}(\psi) \textit{ is even.}$$

It looks like we are almost there: To solve $\exists_k$SAT, we map ϕ to ψ, use the #SAT oracle to count the number of satisfying assignments of ψ and return the parity. In fact this is sufficient to show that $\Sigma_k \subseteq \mathrm{BPP}^{\#\mathrm{SAT}}$, but we promised $\Sigma_k \subseteq \mathrm{P}^{\#\mathrm{SAT}}$. We will deal with this issue later.

4.1 The parity quantifier and proof of Lemma 7

Lemma 7 says the determining if a quantified formula is true can be reduced to computing the *parity* of the number of assignments of some other formula. It will be convenient to view parity as a *quantifier* (like $\exists$ and $\forall$) over the resulting formula: Given a formula ϕ, we say the quantified formula "$\oplus x : \phi(x)$" is true if $\phi(x)$ is true for an odd number of x, and false otherwise. We define the decision problem

$$\oplus\mathrm{SAT} = \{\phi : \text{"}\oplus x : \phi(x)\text{" is true}\}.$$

Just like SAT asks whether a formula is satisfiable, $\oplus$SAT asks if it has an odd number of satisfying assignments. We can do the same for quantified formulas: Define

$$\oplus\exists_k\text{SAT} = \{\phi : \text{"}\oplus x \exists y_1 \forall y_2 \ldots Q y_k : \phi(x, y_1, \ldots, y_k)\text{" is true}\}$$

and $\oplus\forall_k$SAT similarly. In particular the problem $\oplus\exists_k$SAT is hard for $\exists_k$. So to prove Lemma 7, it is sufficient to design a reduction R such that

$$\phi \in \oplus\exists_k\text{SAT} \longrightarrow \Pr[\psi \in \oplus\text{SAT}] \geq 2/3$$
$$\phi \notin \oplus\exists_k\text{SAT} \longrightarrow \psi \notin \oplus\text{SAT}.$$

We start with the case $k = 1$. We are given $\phi(x, y)$ and want to determine if "$\oplus x \exists y : \phi(x, y)$" is true. Let $|x| = |y| = n$. For the moment, let's forget about x and focus on y. What can we do? Using Theorem 3, we can randomly produce a formula $\phi'(x, y)$ such that if $\phi(x, y)$ is satisfiable for some y, then $\phi'(x, y)$ has a unique satisfying assignment with probability $1/8n$, and otherwise it is not satisfiable.

Let us think wishfully and suppose that instead of working with probability $1/8n$, the reduction from Theorem 3 worked with probability one. Then the formula "$\oplus y : \phi'(x, y)$" would be equivalent to "$\exists y$: $\phi(x, y)$", and so the $\oplus$SAT instance "$\oplus x \oplus y$: $\phi'(x, y)$" and the $\oplus\exists$SAT instance "$\oplus x \exists y$: $\phi(x, y)$" would also be equivalent.

Unfortunately the reduction from Theorem 3 sometimes fails; what we will do instead is obtain a random $\phi'(x, y)$ such that "$\oplus y : \phi'(x, y)$" and "$\exists y : \phi(x, y)$" are equivalent with very high probability over the choice of ϕ'. We will make this probability as high as $1 - \frac{1}{6}2^{-n}$. Then by the union bound we have that

$$\Pr[\text{For all } x\text{: "}\oplus y : \phi'(x, y)\text{" is equivalent to "}\exists y : \phi(x, y)\text{"}] \geq 5/6.$$

so in particular

$$\Pr[\text{The formulas "}\oplus x \oplus y : \phi'(x, y)\text{" and "}\oplus x \exists y : \phi(x, y)\text{" are equivalent}] \geq 5/6.$$

Let's now see how to construct ϕ' from ϕ. We run the reduction $m = O(n^2)$ times independently to produce formulas $\phi'_1(x, y)$ up to $\phi'_m(x, y)$. If $\phi(x, y)$ is satisfiable (in y), then with probability $1 - \frac{1}{6}2^{-n}$ at least one of these formulas has a unique satisfying assignment, and otherwise none of them has a satisfying assignment.

We are left with the following task: Given formulas $\phi'_1, \ldots, \phi'_m$ produce a single formula ϕ' such that "$\oplus y : \phi'(x, y)$" is true iff at least one of "$\oplus y : \phi'_i(x, y)$" is true. This can be done using the following general construction: Given two unquantified formulas $\psi(y), \psi'(y)$ define

- $\psi \cdot \psi'$ as the formula $\psi(y) \wedge \psi'(z)$, where y and z are disjoint sets of variables. It is easy to check that $\#\text{SAT}(\psi \cdot \psi') = \#\text{SAT}(\psi) \cdot \#\text{SAT}(\psi')$.
- $\psi + \psi'$ as the formula $(w \wedge \psi(y)) \vee (\overline{w} \wedge \psi'(y))$, where w is an additional boolean variable. Then $\#\text{SAT}(\psi + \psi') = \#\text{SAT}(\psi) + \#\text{SAT}(\psi')$.

- 1 as an arbitrary formula with exactly one satisfying assignment.

Then we can set

$$\phi'(x,y) = 1 + (1 + \phi'_1(x,y)) \cdot (1 + \phi'_2(x,y)) \dots (1 + \phi'_m(x,y))$$

where we think of the formulas as formulas over y, and x is just a free variable that gets copied in the process of constructing ϕ'. By construction ϕ' has an odd number of satisfying assignments (in y) iff at least one of the ϕ'_i does.

This concludes the case $k = 1$. In general, to go from $\oplus\exists_k$SAT to $\oplus\exists_{k-1}$SAT we carry out exactly the same argument to eliminate the outermost existential quantifier of ϕ. We then obtain an instance ψ of $\oplus\forall_{k-1}$SAT. Now observe that $\psi \in \oplus\forall_{k-1}$SAT iff $\overline{\psi} \in \oplus\exists_{k-1}$SAT, and the inductive step is done. We can arrange the probabilities so that the reduction from $\oplus\exists_k$SAT to $\oplus\exists_{k-1}$SAT succeeds with probability at least $1/6k^2$. Then even after we put everything together the reduction will work with probability $1 - \sum_k (1/6k^2) \geq 2/3$.

4.2 Derandomizing the reduction

So far we showed that $\Sigma_k \in \mathrm{BPP}^{\#\mathrm{SAT}}$ for all k. We now show how to go to $\Sigma_k \in \mathrm{P}^{\#\mathrm{SAT}}$.

Lemma 8. *There is a deterministic reduction that runs in time $O(n)$ and on input a formula ψ, produces a formula ψ' such that for every N:*

$$\begin{aligned} \#\mathrm{SAT}(\psi) = 0 \pmod N \quad &\Longrightarrow \quad \#\mathrm{SAT}(\psi') = 0 \pmod{N^2} \\ \#\mathrm{SAT}(\psi) = -1 \pmod N \quad &\Longrightarrow \quad \#\mathrm{SAT}(\psi') = -1 \pmod{N^2} \end{aligned}$$

Therefore, telling if a formula has an even or odd number of assignments reduces to telling if some other formula has 0 or 3 assignments modulo 4, which in turn reduces to telling if some other formula has 0 or 7 assigments modulo 8, and so on. Each time we apply the lemma the size of the formula increases by a constant factor. If we apply the lemma $\log_2 m$ times to ψ for some m, we obtain a formula ψ' of size $\mathrm{poly}(m)|\phi|$ and so that if we can tell if ψ' has zero or nonzero assignments modulo 2^m, then we can tell if ψ has an even or odd number of assignments.

Now we put the two lemmas together. First consider the reduction R from Lemma 7. It is randomized, but we think of it as a deterministic procedure that takes ϕ and a random string r of length $m-1$ and produces a formula ψ_r that depends on r. Applying Lemma 8 to ψ_r $\log_2 m$ times we obtain a formula ψ'_r.

We now consider the quantity $K = \sum_{r \in \{0,1\}^{m-1}} \#\mathrm{SAT}(\psi'_r)$, which counts the number of *pairs* (x, r) such that assignment x satisfies formula ψ'_r.

If $\phi \notin \exists_k$SAT, then regardless of the choice of r, $\#\mathrm{SAT}(\psi'_r)$ must equal 0 modulo 2^m. It follows that $K = 0$ modulo 2^m.

If $\phi \in \exists_k$SAT, let p be the fraction of strings r such that $\#\mathrm{SAT}(\psi'_r)$ is odd. We then have that for a p fraction of strings r, $\#\mathrm{SAT}(\psi'_r) = -1$ modulo 2^m, and for the remaining $1 - p$

fraction, $\#\mathrm{SAT}(\psi'_r) = 0$ modulo 2^m. Since $p \in [2/3, 1]$, it follows that K must fall in the range $[-\frac{2}{3}2^{m-1}, -2^{m-1}]$ modulo 2^m, so that $K \neq 0$ modulo 2^m.

We now have our $\mathrm{P}^{\#\mathrm{SAT}}$ algorithm for $\exists_k\mathrm{SAT}$. On input ϕ, we run the reductions from Lemmas 7 and 8, and ask the oracle to count the number of pairs (x, r) such that x satisfies ψ'_r. (This is a #SAT type question.) If the answer divides 2^m, we reject; otherwise we accept.

We now prove Lemma 8. First let us construct a polynomial p such that

$$\begin{aligned} s = 0 \pmod N \quad &\Longrightarrow \quad p(s) = 0 \pmod{N^2} \\ s = -1 \pmod N \quad &\Longrightarrow \quad p(s) = -1 \pmod{N^2} \end{aligned}$$

It is not difficult to find such a p: If s^2 factors into p the first property is satisfied. Now if $s = -1$ modulo N, then $s^3 = -1$ modulo N, so $s^3(s^3 + 2) = (s^3 + 1)^2 - 1 = -1$ modulo N^2. We can set $p(s) = s^3(s^3 + 2)$.

Now the formula $\psi' = \psi^3 \cdot (\psi^3 + 2)$ proves the lemma.