Introduction to Complexity Theory*
Lecture 10: The counting class $\#\mathcal{P}$

Notes taken by Oded Lachish, Yoav Rodeh and Yael Tauman

February 21, 1999

Summary: Up to this point in the course, we've focused on decision problems where the questions are YES/NO questions. Now we are interested in counting problems. In $\mathcal{NP}$ an element was in the language if it had a short checkable witness. In $\#\mathcal{P}$ we wish to count the number of witnesses to a specific element. We first define the complexity class $\#\mathcal{P}$, and classify it with respect to other complexity classes. We then prove the existence of $\#\mathcal{P}$-complete problems, and mention some natural ones. Then we try to study the relation between $\#\mathcal{P}$ and $\mathcal{NP}$ more exactly, by showing we can probabilistically approximate $\#\mathcal{P}$ using an oracle in $\mathcal{NP}$. Finally, we refine this result by restricting the oracle to a weak form of SAT (called $uniqueSAT$).

1 Defining $\#\mathcal{P}$

We used the notion of an $\mathcal{NP}$-relation when defining $\mathcal{NP}$. Recall:

Definition 1 *($\mathcal{NP}$ relation) : An $\mathcal{NP}$ relation is a relation $R \subseteq \Sigma^\star \times \Sigma^\star$ such that:*

- *R is polynomial time decidable.*
- *There exists a polynomial $p(\cdot)$ such that for every $(x, y) \in R$, it holds that $|y| \le p(|x|)$.*

Given an $\mathcal{NP}$-relation R we defined:

Definition 2 $L_R \stackrel{\text{def}}{=} \{x \in \Sigma^\star \mid \exists y \ s.t. \ (x, y) \in R\}$

We regard the y's that satisfy $(x, y) \in R$ as witnesses to the membership of x in the language L_R. The decision problem associated with R, is the question: Does there exist a witness to a given x? This is our definition of the class $\mathcal{NP}$. Another natural question we can ask is: How many witnesses are there for a given x? This is exactly the question we capture by defining the complexity class $\#\mathcal{P}$. We first define:

Definition 3 *For every binary relation $R \subseteq \Sigma^\star \times \Sigma^\star$, the counting function $f_R : \Sigma^\star \to \mathsf{N}$, is defined by:*

$$f_R(x) \stackrel{\text{def}}{=} |\{y \mid (x, y) \in R\}|$$

* Lecture Notes for a course given by Oded Goldreich at the Weizmann Institute of Science, Israel.

The function f_R captures our notion of counting witnesses in the most natural way. So, we define $\#\mathcal{P}$ as a class of functions. Specifically, functions that count the number of witnesses in an $\mathcal{NP}$-relation.

Definition 4 $\#\mathcal{P} = \{f_R : R \textit{ is an } \mathcal{NP} \textit{ relation}\}$

We encounter some problems when trying to relate $\#\mathcal{P}$ to other complexity classes, since it is a class of functions while all classes we discussed so far are classes of languages. To solve this, we are forced to give a less natural definition of $\#\mathcal{P}$, using languages. For each $\mathcal{NP}$-relation R, we associate a language $\#_R$. How do we define $\#_R$? Our first attempt would be:

Definition 5 (Counting language — first attempt) *:* $\#_R = \{(x,k) : |\{y : (x,y) \in R\}| = k\}$

First observe that given an oracle to f_R, it is easy to decide $\#_R$. This is a nice property of $\#_R$, since we would like it to closely represent our other formalism using functions. For the same reason we also want the other direction: Given an oracle to $\#_R$, we would like to be able to calculate f_R efficiently (in polynomial time). This is not as trivial as the other direction, and in fact, is not even necessarily true. So instead of tackling this problem, we alter our definition:

Definition 6 (Counting language — actual definition) *:* $\#_R = \{(x,k) : |\{y : (x,y) \in R\}| \geq k\}$. *In other words,* $(x,k) \in \#_R$ *iff* $k \leq f_R(x)$.

We choose the last definition, because now we can prove the following:

Proposition 1.1 *For each $\mathcal{NP}$-relation R :*

1. *$\#_R$ is Cook reducible to f_R*
2. *f_R is Cook reducible to $\#_R$*

We denote the fact that problem P Cook reduces to problem Q by $P \ \alpha_c \ Q$.
Proof:

1. *($\#_R$ is Cook reducible to f_R)* : Given (x,k), we want to decide whether $(x,k) \in \#_R$. We use our oracle for f_R, by calling it with parameter x. As an answer we get : $l = |\{y : (x,y) \in R\}|$. If $l \geq k$ then we accept, otherwise reject.

2. *(f_R is Cook reducible to $\#_R$)* : Given x, we want to find $f_R(x) = |\{y : (x,y) \in R\}|$ using our oracle. We know $f_R(x)$ is in the range $\{0, \ldots, 2^{|p(x)|}\}$, where $p(\cdot)$ is the polynomial bounding the size of the witnesses in the definition of an NP-relation. The oracle given is exactly what we need to implement binary search.

   $BINARY(x, Lower, Upper)$:

   - if $(Lower = Upper)$ output $Lower$.
   - $Middle = \frac{Lower+Upper}{2}$
   - if $(x, Middle) \in \#_R$ output $BINARY(x, Middle, Upper)$
   - else output $BINARY(x, Lower, Middle)$

   Where the branching in the third line is because if $(x, Middle) \in \#_R$, then $f_R(x) \geq Middle$, so we need only search for the result in the range $[Middle, Upper]$. A symmetric argument explains the *else* clause.

   The output is: $f_R(x) = BINARY(x, 0, 2^{p(|x|)})$. Binary search in general, runs in time logarithmic in interval it searches in. In our case : $O(\log(2^{p(|x|)})) = O(p(|x|))$. We conclude, that the algorithm runs in polynomial time in $|x|$.

■

Notice that we could have changed our definition of $\#_R$, to be:

$$\#_R = \{(x,k) : |\{y : (x,y) \in R)\}| \leq k\}$$

The proposition would still hold. We could have also changed it to a strict inequality, and gotten the same result.

¿From now on we will use the more natural definition of $\#\mathcal{P}$: as a class of functions. This doesn't really matter, since we showed that in terms of cook-reducibility, the two definitions are equivalent.

It seems that the counting problem related to a relation R should be harder than the corresponding decision problem. It is unknown whether it is strictly harder, but it is certainly not weaker. That is,

Proposition 1.2 *For every $\mathcal{NP}$-relation R, the corresponding language L_R Cook reduces to f_R.*

Proof: Given $x \in \Sigma^\star$, use the oracle to calculate $f_R(x)$. Now, $x \in L_R$ if and only if $f_R(x) \geq 1$. ■

Corollary 7 *$\mathcal{NP}$ Cook reduces to $\#\mathcal{P}$*

On the other hand we can bound the complexity of $\#\mathcal{P}$ from above:

Claim 1.3 *$\#\mathcal{P}$ Cook reduces to $\mathcal{P}$SPACE*

Proof: Given x, we want to calculate $f_R(x)$ using polynomial space. Let $p(\cdot)$ be the polynomial bounding the length of the witnesses of R. We run over all possible witnesses of length $\leq p(|x|)$. For each one, we check in polynomial time whether it is a witness for x, and sum the number of witnesses. All this can be done in space $O(p(|x|) + q(|x|))$, where $q(\cdot)$ is the polynomial bounding the running time (and therefore space) of the witness checking algorithm. Such a polynomial exists since R is an $\mathcal{NP}$-relation. ■

2 Completeness in #P

When one talks about complexity classes, proving the existence, and finding complete problems in the complexity class, is of great importance. It helps reason about the whole class using only one specific problem. Therefore, we are looking for an $\mathcal{NP}$-relation R, s.t. for every other $\mathcal{NP}$-relation Q, there is a Cook reduction from f_Q to f_R. Formally:

Definition 8 *($\#\mathcal{P}$-complete) : f is $\#\mathcal{P}$-complete if*

1. *f is in $\#\mathcal{P}$.*
2. *For every g in $\#\mathcal{P}$, g Cook reduces to f.*

With Occam's Razor in mind, we'll try to find a complete problem, such that all other problems are reducible to it using a very simple form of reduction. Note that by restricting the kind of reductions we allow, we may rule out candidates for $\#\mathcal{P}$-complete problems. We take a restricted form of a Levin reduction ϕ from f_Q to f_R:

$$\forall x \in \Sigma^\star : f_Q(x) = f_R(\phi(x))$$

By allowing only this kind of reduction, we can find out several things about our candidates for $\#\mathcal{P}$-complete problems. For example:

$$f_Q(x) \geq 1 \Leftrightarrow f_R(\phi(x)) \geq 1$$

In other words :

$$x \in L_Q \Leftrightarrow \phi(x) \in L_R$$

Which means that ϕ is a Karp reduction from L_Q to L_R. This implies that the decision problem related to R must be $\mathcal{NP}$-complete. Moreover, we require that the reduction preserves the number of witnesses for every input x. We capture this notion in the following definition:

Definition 9 *(*Parsimonious*) : A reduction $\phi : \Sigma^\star \to \Sigma^\star$, is* Parsimonious *w.r.t. $\mathcal{NP}$-relations Q and R if for every x : $|\{y : (x, y) \in Q\}| = |\{y : (\phi(x), y) \in R\}|$.*

Corollary 10 *if R is an NP-relation, and for every $\mathcal{NP}$-relation Q there exists $\phi_Q : \Sigma^\star \to \Sigma^\star$ s.t. ϕ_Q is parsimonious w.r.t. Q and R then f_R is $\#\mathcal{P}$-complete.*

As we've said, a parsimonious reduction from f_Q to f_R must be a Karp reduction from L_Q to L_R. Therefore, we'll try to prove that the Karp reductions we used to prove SAT is $\mathcal{NP}$-complete, are also parsimonious, and thereby $\#SAT$ is $\#\mathcal{P}$-complete.

Definition 11

$$R_{SAT} = \left\{ (\psi, \tau) \;\middle|\; \begin{array}{l} \psi \textit{ is a boolean formula on variables } V(\psi) \\ \tau \textit{ is a truth assignment for } V(\psi) : \psi(\tau) = 1 \end{array} \right\}$$

We have proved $SAT \stackrel{\text{def}}{=} L_{R_{SAT}}$ is $\mathcal{NP}$-complete by a series of Karp reductions. All we need to show is that each step is in fact a parsimonious reduction.

Theorem 12 *$\#SAT \stackrel{\text{def}}{=} f_{R_{SAT}}$ is $\#\mathcal{P}$-complete.*

Proof: (outline)

1. Obviously $\#SAT$ is in $\#\mathcal{P}$, since R_{SAT} is an NP-relation.

2.
   - The reduction from a generic $\mathcal{NP}$-relation R to Bounded-Halting, is parsimonious because the correspondence between the witnesses is not only one-to-one, it is in fact the identity.
   - The reduction from Bounded-Halting to Circuit-SAT consists of creating for each time unit a set of variables that can describe each possible configuration uniquely. Since a successful run is a specific list of configurations, and corresponds to one witness of Bounded-Halting, we get the same witness translated into one unique representation in binary variables.
   - In the reduction from Circuit-SAT to SAT we add extra variables for each internal gate in the circuit. Each satisfying assignment to the original circuit uniquely determines all the values in the internal gates, and therefore gives us exactly one satisfying assignment to the formula.

■

Notice that we actually proved that the counting problems associated with Bounded-Halting, Circuit-SAT, and SAT are $\#\mathcal{P}$-complete. Not only did we prove $\#SAT$ to be $\#\mathcal{P}$-complete, we also showed that for every f in $\#\mathcal{P}$, there exists a parsimonious reduction from f to $\#SAT$.

The reader might have gotten the impression that every NP-relation R, such that f_R is $\#\mathcal{P}$-complete implies L_R is $\mathcal{NP}$-complete. But the following theorem shows the contrary:

Theorem 13 *There exists an $\mathcal{NP}$-relation R s.t. f_R is $\#\mathcal{P}$-complete, and L_R is polynomial time decidable.*

Notice that such a $\#\mathcal{P}$-complete function, does not have the property that we showed $\#SAT$ has: Not all other functions in $\#\mathcal{P}$ have a parsimonious reduction to it. In fact it cannot be that every $\#\mathcal{P}$ problem has a Karp reduction to f_R, since otherwise L_R would be $\mathcal{NP}$-complete.

The idea in the proof is to modify a hard to calculate relation by adding easy to recognize witnesses to every input, so that the question of existence of a witness becomes trivial, yet the counting problem remains just as hard. Clearly, the $\#\mathcal{P}$-hardness will have to be proven by a non-parsimonious reduction (actually even a non-Karp reduction).

Proof: We define :

$$R'_{SAT} = \left\{ (\phi, (\tau, \sigma)) \;\middle|\; \begin{array}{c} (\phi(\tau) = 1) \wedge (\sigma = 1) \\ \vee \\ \sigma = 0 \end{array} \right\}$$

Obviously $L_{R'_{SAT}} = \Sigma^\star$, so it is in $\mathcal{P}$. But $f_{R'_{SAT}}$ is $\#\mathcal{P}$-complete, since for every ϕ : ϕ's witnesses in R'_{SAT} are:

$$\{(\tau, 1) \;:\; \phi(\tau) = 1\} \cup \{(\tau, 0)\}$$

Which means:

$$\#SAT(\phi) + 2^{|\text{Variables}(\phi)|} = f_{R'_{SAT}}(\phi)$$

So given an oracle to $f_{R'_{SAT}}$ we can easily calculate $\#SAT$, meaning that $f_{R'_{SAT}}$ is $\#\mathcal{P}$-complete.
■

We proved the above theorem by constructing a somewhat unnatural NP-relation. We will now find a more natural problem that gives the same result (i.e., which is also $\#\mathcal{P}$-complete).

Definition 14 *(*Bipartite Graph*) : $G = (V_1 \cup V_2, E)$ is a* Bipartite Graph *if*

- $V_1 \cap V_2 = \emptyset$
- $E \subseteq V_1 \times V_2$

Definition 15 *(*Perfect Matching*) : Let $G = (V_1 \cup V_2, E)$ be a bipartite graph. A* Perfect Matching *is a set of edges $M \subseteq E$, that satisfies:*

1. *every v_1 in V_1 appears in exactly one edge of M.*
2. *every v_2 in V_2 appears in exactly one edge of M.*

Definition 16 *(*Perfect Matching — equivalent definition*) : Let $G = (V_1 \cup V_2, E)$ be a bipartite graph. A* Perfect Matching *is a one-to-one and onto function $f : V_1 \rightarrow V_2$ s.t. for every v in V_1, $(v, f(v)) \in E$.*

Proof: *(equivalence of definitions) :*

- Assume we have a subset of edges $M \subseteq E$ that satisfies the first definition. Define a function $f : V_1 \to V2$:
$$f(v_1) = v_2 \iff (v_1, v_2) \in M$$
f is well defined, because each v_1 in V_1 appears in exactly one edge of M. It is one-to-one and onto because each v_2 in V_2 appears in exactly one edge of M. Since $M \subseteq E$, f satisfies the condition that for all v_1 in V_1 : $(v_1, f(v_1))$ is in E.

- Assume we have a one-to-one and onto function $f : V_1 \to V_2$ that satisfies the above condition. We construct a set $M \subseteq E$:
$$M = \{(v_1, f(v_1)) : v_1 \in V_1\}$$
$M \subseteq E$ because for every v_1 in V_1 we know that $(v_1, f(v_1))$ is in E. The two conditions are also satisfied:

  1. Since f is a function, every v_1 in V_1 appears in exactly one edge of M.
  2. Since f is one-to-one and onto, every v_2 in V_2 appears in exactly one edge of M.

■

Definition 17 $R_{PM} = \{(G, f) : G$ *is a bipartite graph and* f *is a perfect matching of* $G\}$

Fact 2.1 $L_{R_{PM}}$ *is polynomial time decidable.*

The idea of the algorithm is to reduce the problem to a network-flow problem which is known to have a polynomial time algorithm. Given a bipartite graph $G = (V_1 \cup V_2, E)$, we construct a directed graph $G' = (V_1 \cup V_2 \cup \{s, t\}, E')$, so that:

$$E' = E \cup \{(s, v_1) : v_1 \in V_1\} \cup \{(v_2, t) : v_2 \in V_2\}$$

where E is viewed as directed edges from V_1 to V_2. What we did, is add a source s and connect it to one side of the graph, and a sink t connected to the other side. We transform it to a flow problem by setting a weight of 1 to every edge in the graph. There is a one to one correspondence between partial matchings and integer flows in the graph: Edges in the matching correspond to edges in E having a flow of 1. Therefore, there exists a perfect matching iff there is a flow of size $|V_1| = |V_2|$.

Theorem 18 $f_{R_{PM}}$ *is* $\#\mathcal{P}$*-complete.*

This result is proved by showing that the problem of computing the permanent of a $\{0, 1\}$ matrix is $\#\mathcal{P}$-complete. We will show the reduction from counting the number of perfect matchings to computing the permanent of such matrices. In fact, the two problems are computationally equivalent.

Definition 19 *(*Permanent*) : The* permanent *of an* $n \times n$ *matrix* $A = (a_{i,j})_{i,j=1}^n$ *is:*

$$Perm(A) = \sum_{\pi \in S_n} \prod_{i=1}^{n} a_{i,\pi(i)}$$

Where $S_n = \{\pi : \pi$ *is a permutation of* $\{1, \ldots, n\}\}$.

Note that the definition of the permanent of a matrix closely resembles that of the determinant of a matrix. In the definition of determinant, we have the same sum and product, except that each element in the sum is multiplied by the $sign \in \{-1, 1\}$ of the permutation π. Yet, computing the determinant is in $\mathcal{P}$, while computing the permanent is $\#\mathcal{P}$-complete, and therefore is believed not to be in $\mathcal{P}$. The main result in this section is the (unproven here) theorem:

Theorem 20 *$Perm$ is $\#\mathcal{P}$-complete.*

To show the equivalence of computing $f_{R_{PM}}$ and $Perm$, we use:

Definition 21 *(*Bipartite Adjacency Matrix*) : Given a bipartite graph $G = (V_1 \cup V_2, E)$, where $V_1 = \{1, \ldots, n\}$, and $V_2 = \{1, \ldots, m\}$, we define the* Bipartite Adjacency Matrix *of the graph G, as an $n \times m$ matrix $B(G)$, where :*

$$B(G)_{i,j} = \begin{cases} 1 & (i,j) \in E \\ 0 & otherwise \end{cases}$$

Proposition 2.2 *Given a bipartite graph $G = (V_1 \cup V_2, E)$ where $|V_1| = |V_2|$,*

$$f_{R_{PM}}(G) = Perm(B(G))$$

Proof:

$$\begin{aligned} Perm(B(G)) &= |\{\pi \in S_n : \textstyle\prod_{i=1}^{n} b_{i,\pi(i)} = 1\}| \\ &= |\{\pi \in S_n : \forall i \in \{1, \ldots, n\}, b_{i,\pi(i)} = 1\}| \\ &= |\{\pi \in S_n : \forall i \in \{1, \ldots, n\}, (i, \pi(i)) \in E\}| \\ &= |\{\pi \in S_n : \pi \text{ is a perfect matching in } G\}| \\ &= f_{R_{PM}}(G) \end{aligned}$$

■

We just showed that the problem of counting the number of perfect matchings in a bipartite graph Cook reduces to the problem of calculating the permanent of a $\{0, 1\}$ matrix. Notice that the other direction is also true by the same proof: Given a $\{0, 1\}$ matrix, create the bipartite graph that corresponds to it.

Now we will show another graph counting problem, that is equivalent to both of these:

Definition 22 *(*Cycle Cover*) : A* Cycle Cover *of a directed graph G, is a set of vertex disjoint simple cycles that cover all the vertices of G. More formally: $C \subseteq E$ is a* cycle cover *of G if for every connected component V_1 of $G' = (V, C)$, there is an ordering $V_1 = \{v_0, \ldots v_{d-1}\}$ s.t. $(v_i, v_j) \in C \Leftrightarrow j = i + 1 (mod\ d)$*

Notice that there is no problem with connected components of size 1, because we allow self loops.

Definition 23 *$\#Cycle(G)$ = number of cycle covers of G.*

Definition 24 *(*Adjacency Matrix*) : The* Adjacency Matrix *of a directed graph $G = (\{1, \ldots, n\}, E)$ is an $n \times n$ matrix $A(G)$:*

$$A(G)_{i,j} = \begin{cases} 1 & (i,j) \in E \\ 0 & otherwise \end{cases}$$

Proposition 2.3 *For every directed graph G, $Perm(A(G)) = \#Cycle(G)$*

In proving this proposition we use the following:

Claim 2.4 *C is a cycle cover of G if and only if every $v \in V$ has an out-degree and in-degree of 1 in $G' = (V, C)$.*

Proof: (Claim)

- ($\Longrightarrow$) Every vertex appears in exactly one cycle of C, because the cycles are disjoint. Also, since the cycles are simple, every vertex has an in-degree and out-degree of 1.

- ($\Longleftarrow$) For every connected component V_0 of G', take a vertex $v_0 \in V_0$, and create the directed path : $v_0, v_1, \ldots,$, where for every i, $(v_i, v_{i+1}) \in C$. Since the out-degree of every vertex is 1 in G', this path is uniquely determined, and:

  1. There must exist a vertex v_i that appears twice : $v_i = v_j$. Because V is finite.
  2. We claim that the least such i is 0. Otherwise, the in-degree of v_i is greater than 1.
  3. All the vertices of V_0 appear in our path because it is a connected component of G'.

  Thus each V_0 induces a directed cycle, and so G' is a collection of disjoint directed cycles which cover all V.

■

Proof: (Proposition) We'll define:

$$\Omega \stackrel{\text{def}}{=} \{\pi \in S_n : \forall i \in \{1, \ldots, n\}, (i, \pi(i)) \in E\}$$

It is easy to see that $Perm(A(G)) = |\Omega|$. Every $\pi \in \Omega$ defines

$$C_\pi = \{(i, \pi(i)) : i \in \{1, \ldots, n\}\} \subseteq E$$

and since π is a 1-1 and onto $\{1, \ldots, n\}$, the out-degree and in-degree of each vertex in C_π is 1. So C_π is a cycle cover of G. On the other hand, every cycle cover $C \subseteq G$ defines a mapping: $\pi_C(i) = j$ s.t. $(i, j) \in C$, and by the above claim, this is a permutation. ■

3 How close is $\#\mathcal{P}$ to $\mathcal{NP}$?

The main purpose of this lecture, is to study the class $\#\mathcal{P}$, and classify it as best as we can among other complexity classes we've studied. We've seen some examples of $\#\mathcal{P}$ complete problems. We also gave upper and lower complexity bounds on $\#\mathcal{P}$:

$$\mathcal{NP} \ \alpha_c \ \#\mathcal{P} \ \alpha_c \ \mathcal{P}\text{SPACE}$$

We will now try to refine these bounds by showing that $\#\mathcal{P}$ is not as far from $\mathcal{NP}$ as one might suspect. In fact, a counting problem in $\#\mathcal{P}$ can be probabilistically approximated in polynomial time using an NP oracle.

3.1 Various Levels of Approximation

We will start by introducing the notion of a range problem. A range problem is a relaxation of the problem of calculating a function. Instead of requiring one value for each input, we allow a full range of answers for each input.

Definition 25 *(*Range Problem*) : A* Range Problem Π *is defined by two functions* $\Pi = (\Pi_l, \Pi_u)$. $\Pi_l, \Pi_u : \Sigma^\star \to \mathsf{N}$. *s.t. on input* $x \in \Sigma^\star$*, the problem is to find* $t \in (\Pi_l(x), \Pi_u(x))$*, or in other words, return an integer* t*, s.t.* $\Pi_l(x) < t < \Pi_u(x)$.

Note that there is no restriction on the functions Π_l and Π_u, they can even be non-recursive. Since we are going to use range problems to denote an approximation to a function, we define a specific kind of range problems that are based on a function:

Definition 26 *(*Strong Range*) : For* $f : \Sigma^\star \to \mathsf{N}$*, and a polynomial* $p(\cdot)$*, we define the range problem* $StrongRange_p(f) = (l, u)$ *where:*

$$l(x) = f(x) \cdot (1 - \frac{1}{p(|x|)})$$
$$u(x) = f(x) \cdot (1 + \frac{1}{p(|x|)})$$

Strong range captures our notion of a good approximation. We will proceed in a series of reductions that will eventually give us the desired result. The first result we prove, is that it is enough to strongly approximate $\#SAT$.

Proposition 3.1 *If we can approximate* $\#SAT$ *strongly we can just as strongly approximate any* f *in* $\#\mathcal{P}$*. In other words : For every* f *in* $\#\mathcal{P}$*, and every polynomial* $p(\cdot)$*,*

$$StrongRange_p(f) \ \alpha_c \ StrongRange_p(\#SAT)$$

Proof: As we've seen, for every f in $\#\mathcal{P}$, there is a parsimonious reduction ϕ_f w.r.t. f and $\#SAT$. Meaning, for all x : $f(x) = \#SAT(\phi_f(x))$. We may assume that $|\phi_f(x)| > |x|$, because we can always pad $\phi_f(x)$ with something that will not change the number of witnesses:

$$\phi_f(x) \wedge z_1 \wedge z_2 \wedge \ldots \wedge z_{|x|}$$

We now use our oracle to $StrongRange_p(\#SAT)$ on $\phi_f(x)$, and get a result t, that satisfies :

$$\begin{aligned} t &\in (1 \pm \frac{1}{p(|\phi(x)|)}) \cdot \#SAT(\phi_f(x)) \\ &\subseteq (1 \pm \frac{1}{p(|x|)}) \cdot f(x) \end{aligned}$$

■

We now wish to define a weaker form of approximation:

Definition 27 *(*Constant Range*) : For* $f : \Sigma^\star \to \mathsf{N}$*, and a constant* $c > 0$*, we define the range problem* $ConstantRange_c(f) = (l, u)$ *where:*

$$l(x) = \frac{1}{c} \cdot f(x)$$
$$u(x) = c \cdot f(x)$$

We want to show that approximating $\#SAT$ up to a constant suffices to approximate $\#SAT$ strongly. We'll in fact prove a stronger result: that an even weaker form of approximation is enough to approximate $\#SAT$ strongly.

Definition 28 (Weak Range) *: For $f : \Sigma^\star \to \mathsf{N}$, and a constant $\epsilon > 0$, we define the range problem $WeakRange_\epsilon(f) = (l, u)$ where:*

$$l(x) = (\tfrac{1}{2})^{|x|^{1-\epsilon}} \cdot f(x)$$
$$u(x) = 2^{|x|^{1-\epsilon}} \cdot f(x)$$

It is quite clear that $ConstantRange$ is a stronger form of approximation than $WeakRange$:

Claim 3.2 *For every $0 < \epsilon < 1$ and $c > 0$:*

$$WeakRange_\epsilon(\#SAT) \ \alpha_c \ ConstantRange_c(\#SAT)$$

Proof: Simply because for large enough n:

$$(\frac{1}{2}, 2)^{n^{1-\epsilon}} \subseteq (\frac{1}{c}, c)$$

where we use $(\frac{1}{2}, 2)^{n^{1-\epsilon}}$ to denote the range $((\frac{1}{2})^{n^{1-\epsilon}}, 2^{n^{1-\epsilon}})$. ■

Now we prove the main result:

Proposition 3.3 *For every polynomial $p(\cdot)$, and constant $0 < \epsilon < 1$,*

$$StrongRange_p(\#SAT) \ \alpha_c \ WeakRange_\epsilon(\#SAT)$$

Proof: Given ϕ, a boolean formula on variables $\vec{x}$. Define a polynomial $q(n) \geq (2n \cdot p(n))^{\frac{1}{\epsilon}}$, and build ϕ' :

$$\phi' = \bigwedge_{i=1}^{q(|\phi|)} \phi(\vec{x^i})$$

Where each $\vec{x^i}$ is a distinct copy of the variables $\vec{x}$. Obviously $\#SAT(\phi') = (\#SAT(\phi))^{q(|\phi|)}$. Notice $|\phi'| \leq 2|\phi| \cdot q(|\phi|)$. Now, assuming we have an oracle to $WeakRange_\epsilon(\#SAT)$, we call it on ϕ' to get:

$$\begin{aligned} t &\in (\tfrac{1}{2}, 2)^{|\phi'|^{1-\epsilon}} \cdot \#SAT(\phi') \\ &\subseteq (\tfrac{1}{2}, 2)^{2|\phi| \cdot q(|\phi|)^{1-\epsilon}} \cdot (\#SAT(\phi))^{q(|\phi|)} \end{aligned}$$

Our result would be $s = t^{\frac{1}{q(|\phi|)}}$. And we have :

$$\begin{aligned} s &\in (\tfrac{1}{2}, 2)^{\frac{2|\phi|}{q(|\phi|)^\epsilon}} \cdot \#SAT(\phi) \\ &\subseteq (\tfrac{1}{2}, 2)^{\frac{2|\phi|}{2|\phi| p(|\phi|)}} \cdot \#SAT(\phi) \\ &= (\tfrac{1}{2}, 2)^{\frac{1}{p(|\phi|)}} \cdot \#SAT(\phi) \\ &\subseteq (1 \pm \tfrac{1}{p(|\phi|)}) \cdot \#SAT(\phi) \end{aligned}$$

where the last containment follows from :

$$\forall x \geq 1 : (1 - \frac{1}{x})^x \leq \frac{1}{2} \ \text{ and } \ 2 \leq (1 + \frac{1}{x})^x$$

■

After a small diversion into proving a stronger result than needed, we conclude that all we have to do is to find a constant $c > 0$, such that we can solve $ConstantRange_c(\#SAT)$.

We still have a hard time solving the problem directly, so we'll do yet another reduction into a relaxed form of decision problems, called promise problems. While machines that solve decision problems are required to give an exact answer for every input, promise problems are only required to do so on a predefined 'promise' set.

Definition 29 *(*Promise Problem*) : A* Promise Problem $\Pi = (\Pi_Y, \Pi_N)$*, where* $\Pi_Y, \Pi_N \subseteq \Sigma^\star$*, and* $\Pi_Y \cap \Pi_N = \emptyset$*, is the question: Given* $x \in Promise(\Pi) \stackrel{\text{def}}{=} \Pi_Y \cup \Pi_N$*, decide whether* $x \in \Pi_Y$*.*

Notice that if $x \notin Promise(\Pi)$, there is no requirement. Also, promise problems are a generalization of decision problems, where in decision problems $Promise(\Pi) = \Sigma^\star$, so no promise is made.

Definition 30 *(*$Gap_8\#SAT$*) The promise problem* $Gap_8\#SAT = (Gap_8\#SAT_Y, Gap_8\#SAT_N)$*, where:*

$$\begin{array}{rcl} Gap_8\#SAT_Y & = & \{(\phi, K) : \#SAT(\phi) > 8K\} \\ Gap_8\#SAT_N & = & \{(\phi, K) : \#SAT(\phi) < \frac{1}{8}K\} \end{array}$$

We now continue in our reductions. For this we choose $c = 64$, and show we can solve $ConstantRange_{64}(\#SAT)$ using an oracle to $Gap_8\#SAT$.

Proposition 3.4 *$ConstantRange_{64}(\#SAT)$ Cook reduces to $Gap_8\#SAT$*

Proof: We run the following algorithm on input ϕ:

- $i = 0$
- While ($Gap_8\#SAT$ answers YES on $(\phi, 8^i)$) do $i = i + 1$
- return $8^{i-\frac{1}{2}}$

Denote $\alpha = \log_8(\#SAT(\phi))$. The result $8^{k-\frac{1}{2}}$, satisfies : $\alpha - 2 < k - \frac{1}{2} < \alpha + 2$, because :

- For all $i < \alpha - 1$, $\#SAT(\phi) > 8 \cdot 8^i$, so $(\phi, 8^i) \in Gap_8\#SAT_Y$. Therefore, we are promised that in such a case the algorithm will increment such an i, and not stop. So, $k \geq \alpha - 1$ follows.
- For all $i > \alpha + 1$, $\#SAT(\phi) < \frac{1}{8} \cdot 8^i$, so $(\phi, 8^i) \in Gap_8\#SAT_N$. Meaning that the algorithm must stop at the first such i or before. The first such i that satisfies $i > \alpha + 1$ also satisfies $i \leq \alpha + 2$. Therefore $k \leq \alpha + 2$.

Now :

$$\begin{array}{ccccc} \alpha - 1 & \leq & k & \leq & \alpha + 2 \\ & & \Downarrow & & \\ \alpha - 2 & < & k - \frac{1}{2} & < & \alpha + 2 \end{array}$$

We conclude:

$$8^{k-\frac{1}{2}} \in (\frac{1}{64}, 64) \cdot \#SAT(\phi)$$

■

So far we've shown the following reductions:

$$StrongRange_{\text{poly}}(\#\mathcal{P}) \; \alpha_c \; StrongRange_{\text{poly}}(\#SAT) \; \alpha_c \; WeakRange_{\epsilon}(\#SAT)$$
$$\alpha_c \; ConstantRange_{64}(\#SAT) \; \alpha_c \; Gap_8\#SAT$$

Since Cook reductions are transitive, we get :

$$StrongRange_{\text{poly}}(\#\mathcal{P}) \; \alpha_c \; Gap_8\#SAT$$

We will show how to solve $Gap_8\#SAT$ using an oracle to SAT, but with a small probability of error. So we will show, that in general, if we can solve a problem P using an oracle to a promise problem Q, then if we have an oracle to Q that makes little mistakes, we can solve P with high probability.

Comment : *(Amplification)* : For every promise problem P, and machine M that satisfies:

$$\text{for every } x \in Promise(P) : Prob[M(x) = P(x)] > \frac{2}{3}$$

If on input x that is in $Promise(P)$, we run M on x, $O(n)$ times, then the majority of the results will equal $P(x)$ with probability greater than $1 - 2^{-n}$.

This we proved when we talked about $\mathcal{BPP}$, and the proof stays exactly the same, using Chernoff's bound. Note that we do not care if machine M has an oracle or not, and if so how this oracle operates, as long as different runs of M are independent.

Proposition 3.5 *Given a problem P and a promise problem Q, such that P Cook reduces to Q, if we have a probabilistic machine Q' that satisfies:*

$$\textit{for every } x \in Promise(Q) : Prob[Q'(x) = Q(x)] > \frac{2}{3}$$

then for every polynomial $p(\cdot)$, we have a probabilistic polynomial time machine M that uses an oracle to Q', and satisfies:

$$Prob[M^{Q'}(y) \textit{ is a solution of } P \textit{ on input } y] > 1 - 2^{-p(|y|)}$$

Proof: We start by noticing that since the reduction from P to Q is polynomial, there exists a polynomial $q(\cdot)$, such that the oracle Q is called less than $q(|y|)$ times. Since we use Q' and not Q as an oracle, we have a probability of error. If each one of these calls had a probability of error less than : $\frac{1}{q(|y|)} \cdot 2^{-p(|y|)}$, then by using the union bound we would get that the probability that at least one of the oracle calls was incorrect is less than $2^{-p(|y|)}$. The probability of M being correct, is at least the probability that all oracle calls are correct, therefore in this case it is greater than $1 - 2^{-p(|y|)}$.

Using the comment about amplification, we can amplify the probability of success of each oracle call to $1 - \frac{1}{q(|y|)} 2^{-p(|y|)}$, by calling it $O(p(|y|) \cdot \log q(|y|))$ number of times, which is polynomial in the size of the input. ■

In conclusion, all we have to do is show that we can solve $Gap_8\#SAT$ with a probability of error $< \frac{1}{3}$. Then, we showed that we can find a solution to $\#SAT$, that is very close to the real solution ($StrongRange_p(\#SAT)$), with a very high probability of success.

3.2 Probabilistic Cook Reduction

In the next sections, we extensively use the notion of probabilistic reduction. Therefore, we'll define it formally, and prove some of it's properties.

Definition 31 (Probabilistic Cook Reduction) *: Given promise problems P and Q, we say that there is a* Probabilistic Cook Reduction *from P to Q denoted $P \ \alpha_R \ Q$, if there is a probabilistic polynomial time oracle machine M that uses Q as an oracle, and satisfies:*

$$\textit{for every } x \in Promise(P) : Prob[M^Q(x) = P(x)] > \frac{2}{3}$$

where $M^Q(x)$ denotes then operation of machine M on input x when given oracle access to Q. Whenever a query to Q satisfies the promise of Q, the answer is correct, but when the query violates the promise the answer may be arbitrary.

Notice that in the definition, the oracle has no probability of error. We now show that this restriction does not matter, and we can do the same even if the oracle is implemented with bounded probability of error.

Proposition 3.6 *If P probabilistically Cook reduces to Q, and we have a probabilistic machine Q' that satisfies*

$$\text{for every } x \in Promise(Q) : Prob[Q'(x) = Q(x)] > \frac{2}{3}$$

then we have a probabilistic polynomial time oracle machine M that uses Q' as an oracle, and satisfies :

$$\text{for every } y \in Promise(P) : Prob[M^{Q'}(y) = P(y)] > \frac{2}{3}$$

Proof: By the definition of a probabilistic Cook reduction, we have a probabilistic polynomial time oracle machine N that satisfies:

$$\text{for every } y \in Promise(P) : Prob[N^{Q}(y) = P(y)] > \frac{3}{4}$$

Where we changed $\frac{2}{3}$ to $\frac{3}{4}$, using the comment about amplification. Machine N runs in polynomial time, therefore it calls the oracle a polynomial $p(|y|)$ number of times. We can assume Q' to be correct with a probability $> \frac{1}{9} \cdot \frac{1}{p(|y|)}$, by calling it each time instead of just once, $O(\log(p(|y|)))$ times, and taking the majority. Using the union bound, the probability that all oracle calls (to this modified Q') are correct is greater than $\frac{8}{9}$.

When all oracle calls are correct, machine N returns the correct result. Therefore with probability greater than $\frac{3}{4} \cdot \frac{8}{9} = \frac{2}{3}$ we get the correct result. ■

We list some properties of probabilistic cook reductions:

- Deterministic Cook reduction is a special case (i.e., $P \ \alpha_c \ Q \implies P \ \alpha_R \ Q$).
- *Transitivity* : $P \ \alpha_R \ Q \ \alpha_R \ R \implies P \ \alpha_R \ R$

3.3 $Gap_8\#SAT$ Reduces to SAT

Our goal, is to show that we can approximate any problem in $\#\mathcal{P}$ using an oracle to SAT. So far we've reduced the problem several times, and got:

$$StrongRange_{\text{poly}}(\#\mathcal{P}) \ \alpha_c \ Gap_8\#SAT$$

Now we'll show:

$$Gap_8\#SAT \ \alpha_R \ SAT$$

And using the above properties of probabilistic Cook reductions, this will mean that we can approximate $\#\mathcal{P}$ very closely, with an exponentially small probability of error.

Reminder: $Gap_8\#SAT$ is the promise problem on input pairs (ϕ, k), where ϕ is a boolean formula, and k is a natural number. $Gap_8\#SAT = (Gap_8\#SAT_Y, Gap_8\#SAT_N)$, where:

$$\begin{aligned} Gap_8\#SAT_Y &= \{(\phi,k) : \#SAT(\phi) > 8k\} \\ Gap_8\#SAT_N &= \{(\phi,k) : \#SAT(\phi) < \tfrac{1}{8}k\} \end{aligned}$$

How do we approach the problem? We know, that there is either a very large or a very small number of truth assignment in comparison to the input parameter k. So if we take a random $\frac{1}{k}$

fraction of the assignments, with high probability in the first case at least one of them is satisfying, and in the second, none are. Assume that we have a way of restricting our formula to a random fraction of the assignments S that satisfies : each assignment τ is in the set with probability $\frac{1}{k}$ independently of all other assignments. We set $\phi'(\tau) = \phi(\tau) \wedge (\tau \in S)$. Then we simply check satisfiability of ϕ'. First notice:

$$Prob_S[\phi' \in SAT] \quad = \quad 1 - Prob_S[\forall\tau \text{ s.t. } \phi(\tau) = 1 \ : \ \tau \notin S] \quad = \quad 1 - (\tfrac{k-1}{k})^{\#SAT(\phi)}$$

Therefore:

$$\begin{array}{llllllll} \text{If } \#SAT(\phi) > 8k \text{ then} & Prob_S[\phi' \in SAT] & > & 1 - (\frac{k-1}{k})^{8k} & \approx & 1 - \frac{1}{e^8} & > & \frac{2}{3} \\ \text{If } \#SAT(\phi) < \frac{1}{8}k \text{ then} & Prob_S[\phi' \in SAT] & < & 1 - (\frac{k-1}{k})^{\frac{1}{8}k} & \approx & 1 - \frac{1}{e^{\frac{1}{8}}} & < & \frac{1}{3} \end{array}$$

The problem is, we don't have an efficient procedure to choose such a random S. So we weaken our requirements, instead of total independence, we require only pairwise independence. Specifically, we use the following tool:

Definition 32 (Universal$_2$ Hashing) *: A family of functions, $H_{n,m}$, mapping $\{0,1\}^n$ to $\{0,1\}^m$ is called* Universal$_2$ *if for a uniformly selected h in $H_{n,m}$, the random variables $\{h(e)\}_{e\in\{0,1\}^n}$ are pairwise independent and uniformly distributed over $\{0,1\}^m$. That is, for every $x \neq y \in \{0,1\}^n$, and $a, b \in \{0,1\}^m$,*

$$Prob_{h\in H_{n,m}}[h(x) = a \ \& \ h(y) = b] \ = \ (2^{-m})^2$$

An efficient construction of such families is required to have algorithms for selecting and evaluating functions in the family. That is,

1. *selecting:* There exists a probabilistic polynomial-time algorithm that on input $(1^n, 1^m)$, outputs a description of a uniformly selected function in $H_{n,m}$.

2. *evaluating:* There exists a polynomial-time algorithm that on input: a description of a function $h \in H_{n,m}$ and a domain element $x \in \{0,1\}^m$ outputs the value $h(x)$.

A popular example is the family of all affine transformations from $\{0,1\}^n$ to $\{0,1\}^m$. That is, all functions of the form $h_{A,b}(x) = Ax + b$, where A is an m-by-n 0-1 matrix, b is an m-dimensional 0-1 vector, and arithmetic is modulo 2. Clearly, this family has an efficient construction. In Appendix A, we will show that this family is Universal$_2$.

Lemma 3.7 (Leftover Hash Lemma): *Let $H_{n,m}$ be a family of Universal$_2$ Hash functions mapping $\{0,1\}^n$ to $\{0,1\}^m$, and let $\epsilon > 0$. Let $S \subseteq \{0,1\}^n$ be arbitrary provided that $|S| \geq \epsilon^{-3} \cdot 2^m$. Then:*

$$Prob_h[|\{e \in S : h(e) = 0^m\}| \in (1 \pm \epsilon) \cdot \frac{|S|}{2^m}] > 1 - \epsilon$$

The proof of this lemma appears in Appendix B.

We are now ready to construct a probabilistic Cook reduction from $Gap_8\#SAT$ to SAT, using a Universal$_2$ family of functions. Specifically we will use the family of affine transformations.

Theorem 33 $Gap_8\#SAT \ \alpha_R \ SAT$

Proof: We construct a probabilistic polynomial time machine M which is given oracle access to SAT. On input $(\phi, 2^m)$, where ϕ has n variables, M operates as follows:

1. Select uniformly $h \in H_{n,m} = \{\text{Affine transformations from } \{0,1\}^n \text{ to } \{0,1\}^m\}$. The function h is represented by a $\{0,1\}$ matrix $A_{m\times n} = (a_{i,j})_{\substack{i=1,\ldots,m \\ j=1,\ldots,n}}$ and a $\{0,1\}$ vector $b = (b_i)_{i=1,\ldots,m}$.

2. We construct a formula ψ_h, on variables $x_1, ..., x_n, y_1, ..., y_t$, so that for every $x \in \{0,1\}^n$ $h(x) = 0^m$ iff there exists an assignment to the y_i's so that $\psi_h(x_1, ..., x_n, y_1, ..., y_t)$ is true. Furthermore, in case $h(x) = 0^m$, there is a unique assignment to the y_i's so that $\psi_h(x_1, ..., x_n, y_1, ..., y_t)$ is true.

   The construction of ψ_h can be presented in two ways. In the abstract way, we just observe that applying the standard Cook-reduction to the assertion $h(x) = 0^m$, results in the desired formula. (The claimed properties have to be verified indeed.) A more concrete way is to start by the following observations

$$h(x_1, \ldots x_n) = 0^m$$
$$\Updownarrow$$
$$\bigwedge_{i=1}^m \left(\sum_{j=1}^n a_{i,j}x_j \equiv b_i \pmod 2\right)$$
$$\Updownarrow$$
$$\bigwedge_{i=1}^m \left((b_i \oplus 1) \oplus \bigoplus_{j=1}^n (a_{i,j} \wedge x_j)\right)$$

   Introducing auxiliary variables, as in the construction of the standard reduction from Circuit–Satisfiability to 3SAT, we obtain the desired formula ψ_h. For example, introducing variables $y_1, ..., y_n, y_{1,1}, ..., y_{m,n}$, the above formula is satisfied for a particular setting of the x_i's iff the following formula is satisfyiable for these x_i's (and furthermore for a unique setting of the y_i's):

$$\bigwedge_{i=1}^m (b_i \oplus 1 \oplus y_i) \;\wedge\; \bigwedge_{i=1}^m \left(y_i = \bigoplus_{j=1}^n y_{i,j}\right) \;\wedge\; \bigwedge_{i=1}^m \bigwedge_{j=1}^n (y_{i,j} = a_{i,j} \wedge x_j)$$

   So all that is left is to write a CNF for $\bigoplus_{j=1}^n y_{i,j}$, by using additional auxiliary variables. To write a CNF for $\bigoplus_{j=1}^n z_j$, we look at a binary tree of depth $\ell \stackrel{\text{def}}{=} \log_2 n$ which computes the XOR in the natrual way. We introduce an auxiliary variable for each internal node, and obtain

$$w_{0,1} \;\wedge\; \bigwedge_{i=0}^{\ell-1} \bigwedge_{j=1}^{2^i} (w_{i,j} = w_{i+1,2j-1} \oplus w_{i+1,2j}) \;\wedge\; \bigwedge_{j=1}^n (w_{\ell,j} = z_j)$$

3. Define $\phi' = \phi \wedge \psi_h$. Use our oracle to SAT on ϕ', and return the result.

We claim :

- *claim 1* : If $(\phi, 2^m) \in Gap_8\#SAT_Y$ then $\phi' \in SAT$ with probability $> \frac{1}{2}$.
- *claim 2* : If $(\phi, 2^m) \in Gap_8\#SAT_N$ then $\phi' \in SAT$ with probability $< \frac{1}{8}$.

Before the proof of these claims, notice that we can amplify these results in the standard way, to get the desired probabilities. Now we prove the above claims:

1. We define : $S_\phi \stackrel{\text{def}}{=} \{x : \phi(x) = 1\}$. Because $(\phi, 2^m) \in Gap_8\#SAT_Y$, we know that $|S_\phi| > 8 \cdot 2^m$ Now:

$$\begin{aligned} Prob_h[\phi' \in SAT] &= Prob_h[\{x : \phi(x) = 1 \;\&\; h(x) = 0^m\} \neq \emptyset] \\ &= Prob_h[\{x \in S_\phi : h(x) = 0^m\} \neq \emptyset] \\ &\geq Prob_h[|\{x \in S_\phi : h(x) = 0^m\}| \in (1 \pm \tfrac{1}{2})\tfrac{|S_\phi|}{2^m}] > \tfrac{1}{2} \end{aligned}$$

   The last inequality is an application of the Leftover Hash lemma, setting $\epsilon = \frac{1}{2}$.

2. As $(\phi, 2^m) \in Gap_8\#SAT_N$, we have $|S_\phi| < \frac{1}{8} \cdot 2^m$.

$$\begin{aligned} Prob_h[\phi' \in SAT] &= Prob_h[\{x \in S_\phi : h(x) = 0^m\} \neq \emptyset] \\ &= Prob_h[(\bigcup_{x \in S_\phi} \{x : h(x) = 0^m\}) \neq \emptyset] \\ &\leq \sum_{x \in S_\phi} Prob_h[h(x) = 0^m] \\ &< \frac{1}{8} \cdot 2^m \cdot 2^{-m} = \frac{1}{8} \end{aligned}$$

The last inequality uses the union bound.

■

In conclusion, we have shown:

$$StrongApprox_{\text{poly}}(\#\mathcal{P}) \ \alpha_c \ Gap_8\#SAT \ \alpha_R \ SAT$$

Which is what we wanted.

4 Reducing to $uniqueSAT$

We've introduced the notion of *promise problems* as a means to prove that we can approximate $\#SAT$ using SAT. But promise problems are interesting by their own right, so we will try to investigate them a bit more. We've shown that using an oracle to SAT we can solve $Gap_8\#SAT$. The converse is also true, because we've shown we can approximate (deterministically) $\#SAT$ using $Gap_8\#SAT$, so all we have to do is approximate well enough, to differentiate 0 from positive results, and thus, solve SAT. We will try to refine this result, by showing that a more restricted version of $Gap_8\#SAT$ is enough to solve SAT (and even approximate $\#SAT$).

Definition 34 *$Gap_8\#SAT'$ is the promise problem on input pairs (ϕ, k) defined by:*

$$\begin{aligned} Gap_8\#SAT'_Y &= \{(\phi, k) : 8k < \#SAT(\phi) < 32k\} \\ Gap_8\#SAT'_N &= \{(\phi, k) : \#SAT(\phi) < \tfrac{1}{8}k\} \end{aligned}$$

Claim 4.1 *SAT Cook reduces to $Gap_8\#SAT'$*

Proof: Given ϕ, first we will create formula ϕ', s.t. $\#SAT(\phi') = 15 \cdot \#SAT(\phi)$. Take 4 variables $\{x_1, x_2, x_3, x_4\}$ not appearing in ϕ. and define:

$$\begin{aligned} \psi &= (x_1 \vee x_2 \vee x_3 \vee x_4) \\ \phi' &= \phi \wedge \psi \end{aligned}$$

Observe that $\#SAT(\psi) = 15$, and since the variables of ψ do not appear in ϕ, the above equality holds. So we know that :

$$\begin{aligned} \#SAT(\phi') \geq 15 &\iff \phi \in SAT \\ \#SAT(\phi') = 0 &\iff \phi \notin SAT \end{aligned}$$

For every $0 \leq i \leq |Variables(\phi')|$, we call our oracle: $Gap_8\#SAT'(\phi', 2^i)$. We claim : One of the answers is YES iff $\phi \in SAT$.

- Suppose that $\phi \notin SAT$. Then $\#SAT(\phi') = 0 < \frac{1}{8}k$ for all $k > 0$, therefore for all i, $(\phi', 2^i) \in Gap_8\#SAT'_N$, so we are promised to always get a NO answer.

- Suppose $\phi \in SAT$, so as we showed, $\#SAT(\phi') \geq 15$. Therefore, $\log_2(\#SAT(\phi')) \geq \log_2(15) > 3$. There exists an integer $i \geq 0$ s.t.

$$\begin{gathered} i < \log_2(\#SAT(\phi')) - 3 < i+2 \\ \Downarrow \\ 2^{i+3} < \#SAT(\phi') < 2^{i+5} \\ \Downarrow \\ 8 \cdot 2^i < \#SAT(\phi') < 32 \cdot 2^i \end{gathered}$$

  And for that i, we are guaranteed to get a YES answer.

■

The reader may wonder why we imposed this extra restriction on $Gap_8\#SAT$. We want to show that we can solve SAT using weak oracles. For example $Gap_8\#SAT'$ is a weak oracle. But we wish to continue in our reductions, and our next step is:

Definition 35 *$fewSAT$ is the promise problem defined by:*

$$\begin{array}{rcll} fewSAT_Y & = & \{\phi : 1 \leq \#SAT(\phi) < 100\} & \subset \ SAT \\ fewSAT_N & = & \{\phi : \#SAT(\phi) = 0\} & = \ \overline{SAT} \end{array}$$

Proposition 4.2 *$Gap_8\#SAT'$ probabilistically Cook reduces to $fewSAT$*

Proof: We will use the same reduction we used when proving $Gap_8\#SAT \ \alpha_R \ SAT$, except we now have $Gap_8\#SAT'$. Recall, we uniformly select $h \in H_{n,m}$, and construct $\phi'(x) = \phi(x) \wedge (h(x) = 0^m)$.

We make analogous claims to the ones stated in the former proof:

- *claim 1* : If $(\phi, 2^m) \in Gap_8\#SAT'_Y$ then $\phi' \in fewSAT$ with probability $> \frac{1}{2}$.
- *claim 2* : If $(\phi, 2^m) \in Gap_8\#SAT'_N$ then $\phi' \in fewSAT$ with probability $< \frac{1}{8}$.

1. Since $(\phi, 2^m) \in Gap_8\#SAT'_Y$, we have:

$$8 \cdot 2^m < |S_\phi \stackrel{\text{def}}{=} \{x : \phi'(x) = 1\}| < 32 \cdot 2^m$$

   So now:

$$\begin{array}{rcl} Prob_h[\phi' \in fewSAT] & = & Prob_h[0 < |\{x : \phi(x) = 1 \ \& \ h(x) = 0^m\}| < 100] \\ & = & Prob_h[0 < |\{x \in S_\phi : h(x) = 0^m\}| < 100] \\ & \geq & Prob_h[(1 - \frac{1}{2}) \cdot 8 < |\{x \in S_\phi : h(x) = 0^m\}| < (1 + \frac{1}{2}) \cdot 32] \\ & \geq & Prob_h[|\{x \in S_\phi : h(x) = 0^m\}| \in (1 \pm \frac{1}{2})\frac{|S_\phi|}{2^m}] > \frac{1}{2} \end{array}$$

2. In the original proof we showed: if $(\phi, 2^m) \in Gap_8\#SAT_N$ then ϕ' is not satisfiable with probability greater than $\frac{7}{8}$. Notice :$Gap_8\#SAT_N = Gap_8\#SAT'_N$, so if $(\phi, 2^m) \in Gap_8\#SAT'_N$ then ϕ' is not satisfiable with probability greater than $\frac{7}{8}$, and in that case, it's in $fewSAT_N$, so we are guaranteed to get a NO answer.

■

As a last step in this endless crusade to understand the complexity of SAT promise problems, we will show that the weakest SAT related promise problem, is in fact as strong as the others.

Definition 36 *uniqueSAT is the promise problem on input ϕ defined by:*

$$\begin{array}{rclcl} uniqueSAT_Y & = & \{\phi : \#SAT(\phi) = 1\} & \subset & SAT \\ uniqueSAT_N & = & \{\phi : \#SAT(\phi) = 0\} & = & \overline{SAT} \end{array}$$

Proposition 4.3 *$fewSAT$ Cook reduces to uniqueSAT*

Proof: Given a formula ϕ, we want to solve $fewSAT$. For each $1 \leq i < 100$ we construct a formula ϕ_i, s.t. :

- $\phi \notin SAT \Rightarrow \phi_i \notin SAT$.
- ϕ_i has a unique satisfying assignment if ϕ has exactly i satisfying assignments.

If we can do this, we can check all these ϕ_i's, with our oracle to $uniqueSAT$. If all of them are NO, then we return NO, otherwise we answer YES. This is correct because if $0 < k \stackrel{\text{def}}{=} \#SAT(\phi) < 100$, then ϕ_k has exactly one satisfying assignment, and therefore $uniqueSAT$ returns YES on ϕ_k. Also, if $\phi \notin SAT$, then all for all $i\ : \phi_i \in uniqueSAT_N$, so all the answers must be NO.

All that is left is to construct ϕ_i: We create i copies of ϕ, each on a separate set of variables:

$$\psi_i = \wedge_{j=1}^{i} \phi(x_1^j, \ldots, x_n^j)$$

First notice, that if $\phi \notin SAT$, then so is ψ_i. Now assume $\#SAT(\phi) = i$. Every satisfying assignment of ψ_i, corresponds to i satisfying assignments of ϕ. But we want to force them to be different, so we would require that the assignments are different and add this requirement to ψ_i. But then, we will have exactly $i!$ satisfying assignments to the new ψ_i. To solve this, instead of just requiring that they are different, we will impose a lexicographical ordering of the solutions, which will fix one satisfying assignment from the $i!$ possible.

$$\phi_i = \psi_i \wedge \bigwedge_{j=1}^{i-1} (\vec{x^j} <_{lex} \vec{x^{j+1}})$$

■

Just for the heck of it, we'll list all the reductions in order:

$$\begin{array}{c} StrongRange_{\text{poly}}(\#\mathcal{P})\ \alpha_c \\ StrongRange_{\text{poly}}(\#SAT)\ \alpha_c \\ WeakRange_{\epsilon}(\#SAT)\ \alpha_c \\ ConstantRange_{64}(\#SAT)\ \alpha_c \\ Gap_8\#SAT\ \alpha_R \\ SAT\ \alpha_c \\ Gap_8\#SAT'\ \alpha_R \\ fewSAT\ \alpha_c \\ uniqueSAT \end{array}$$

Some collapsing gives us:

$$StrongRange_{\text{poly}}(\#\mathcal{P})\ \alpha_c\ Gap_8\#SAT\ \alpha_R\ uniqueSAT$$

Appendix A: A Family of Universal_2 Hash Functions

In this appendix we show that the family of affine transformations from $\{0,1\}^n$ to $\{0,1\}^m$ is efficiently constructible and is Universal_2.

1. *selecting:* Simply selecting uniformly and independently each bit of A and b, will output a uniformly selected affine transformation. This runs in $O(nm+m)$ time, which is polynomial in the length of the input.

2. *evaluating:* Calculating Ax takes $O(mn)$ time, and the addition of b adds $O(m)$ time. All in all, polynomial in the size of the input.

Proposition: *The family of affine transformations from $\{0,1\}^n$ to $\{0,1\}^m$ is $\textit{Universal}_2$.*

Proof: Given $x_1 \neq x_2 \in \{0,1\}^n$, and $y_1, y_2 \in \{0,1\}^m$. If $x_1 = 0^n$, then

$$\begin{aligned} Prob_{A,b}[h(x_1) = y_1 \ \& \ h(x_2) = y_2] &= Prob_{A,b}[b = y_1 \ \& \ Ax_2 + b = y_2] \\ &= Prob_{A,b}[b = y_1 \ \& \ Ax_2 = y_2 - y_1] \\ &= Prob_A[Ax_2 = y_2 - y_1] \cdot Prob_b[b = y_1] \\ &= 2^{-m} \cdot 2^{-m} = (2^{-m})^2 \end{aligned}$$

Where $Prob[Ax_2 = y_2 - y_1] = 2^{-m}$, because for a given vector $x_2 \neq 0^n$, a uniformly chosen linear transformation A, maps x_2 uniformly into $\{0,1\}^m$. If $x_2 = 0$ the same argument holds. Assume both are different than 0^m. Since we choose among the linear transformations uniformly, it does not matter in what base we represent them. Since $x_1, x_2 \neq 0$, and they are both in $\{0,1\}^n$, they must be linearly independent. So we may assume they are both base vectors in the representation of A, meaning one column in A : column a_1 in A represents the image of x_1, and a different column a_2 represents the image of x_2.

$$\begin{aligned} Prob_{A,b}[h(x_1) = y_1 \ \& \ h(x_2) = y_2] &= Prob_{A,b}[Ax_1 + b = y_1 \ \& \ Ax_2 + b = y_2] \\ &= Prob_{a_1,a_2,b}[a_1 + b = y_1 \ \& \ a_2 + b = y_2] \\ &= Prob_{a_1,a_2}[a_1 = y_1 - b \ \& \ a_2 = y_2 - b] \qquad \text{(for every } b\text{)} \\ &= Prob_{a_1}[a_1 = y_1 - b] \cdot Prob_{a_2}[a_2 = y_2 - b] \quad \text{(for every } b\text{)} \\ &= 2^{-m} \cdot 2^{-m} = (2^{-m})^2 \end{aligned}$$

■

Appendix B: Proof of Leftover Hash Lemma

In this appendix, we prove the Leftover Hash Lemma (Lemma 3.7). We first restate the lemma.

The Leftover Hash Lemma: Let $H_{n,m}$ be a family of Universal$_2$ Hash functions mapping $\{0,1\}^n$ to $\{0,1\}^m$, and let $\epsilon > 0$. Let $S \subseteq \{0,1\}^n$ be arbitrary provided that $|S| \geq \epsilon^{-3} \cdot 2^m$. Then:

$$Prob_h[|\{e \in S : h(e) = 0^m\}| \in (1 \pm \epsilon) \cdot \frac{|S|}{2^m}] > 1 - \epsilon$$

Proof: We define for each $e \in \{0,1\}^n$ a random variable X_e :

$$X_e = \begin{cases} 1 & h(e) = 0^m \\ 0 & \text{otherwise} \end{cases}$$

For each $e_1 \neq e_2 \in \{0,1\}^n$, we claim that X_{e_1}, X_{e_2} are stochastically independent, because they are functions of the independent random variables $h(e_1)$ and $h(e_2)$ respectively. That is, we use the known fact by which if X and Y are independent random variables then, for every function f, $f(X)$ and $f(Y)$ are also independent random variables.

We compute :

$$\begin{aligned} E(x_e) &= Prob[X_e = 1] = \tfrac{1}{2^m} \\ VAR(X_e) &= Prob[X_e = 1] \cdot (1 - Prob[X_e = 1]) = \tfrac{1}{2^m}(1 - \tfrac{1}{2^m}) \end{aligned}$$

We define a new random variable $Y = \sum_{e \in S} X_e$. In other words : $Y = |\{e \in S : h(e) = 0^m\}|$. Since the X_e's are pairwise independent we get:

$$\begin{aligned} E(Y) &= \textstyle\sum_{e \in S} E(X_e) = \tfrac{|S|}{2^m} \\ VAR(Y) &= \textstyle\sum_{e \in S} VAR(X_e) = \tfrac{|S|}{2^m}(1 - \tfrac{1}{2^m}) = (1 - \tfrac{1}{2^m}) \cdot E(Y) \end{aligned}$$

We will now use the Chebychev inequality to prove:

$$\begin{aligned} Prob[|\{e \in S : h(e) = 0^m\}| \in (1 \pm \epsilon) \cdot \tfrac{|S|}{2^m}] &= Prob[Y \in (1 \pm \epsilon) \cdot E(Y)] \\ &= Prob[|Y - EY| \leq \epsilon \cdot E(Y)] \\ &\geq 1 - \tfrac{VAR(Y)}{(\epsilon \cdot E(Y))^2} = 1 - \tfrac{(1 - \frac{1}{2^m}) \cdot E(Y)}{\epsilon^2 (E(Y))^2} \\ &= 1 - \tfrac{(1 - \frac{1}{2^m}) 2^m}{\epsilon^2 \cdot |S|} \geq 1 - \epsilon \cdot (1 - \tfrac{1}{2^m}) > 1 - \epsilon \end{aligned}$$

■