We assume that we have access to a uniform distribution on $\{0,1\}^n$, and wish to obtain a uniform distribution on some other sets. Some probability distributions of interest are:

1. Uniform distribution on random walks on the one-dimensional integer lattice. (This is equivalent to the uniform distribution on $\{0,1\}^n$ under the transformation $0 \to Left, 1 \to Right$.)
2. A uniform distribution on random walks with a fixed bias $(p, 1-p)$ on the one-dimensional integer lattice. (For p which can be written as $a/2^b$ for some integer a and b, this can be done exactly in the following manner. For each $Left/Right$ move examine the sequence of random variables $\{0,1\}^b$ and move $Left$ if $\{0,1\}^b$ is one of a set of a different pre-chosen sequences, else move $Right$. For other p, it can be approximated to arbitrary accuracy by similar methods.)
3. Uniform distribution on permutations on the set $\{1, 2, ..., n\}$. (These can be obtained by shuffling techniques. This and some of the next examples will be of particular interest to us in the rest of this course).
4. Uniform distribution on the spanning trees of an undirected graph.
5. Uniform distribution on solutions to a knapsack problem, where a knapsack problem is defined as: Given $a_1, a_2, ..., a_n, b \in \mathbb{N}$, a solution is a vector $v = (v_1, v_2, ..., v_n) \in \{0,1\}^n$ s.t. $\sum_i v_i a_i \leq b$
6. Uniform distribution on matchings, or perfect matchings, of a graph.
7. Uniform distribution on self-avoiding walks in $\mathbb{Z}^k$.

Approximate counting vs. approximate sampling

Suppose we have a polynomial-time predicate ϕ , where $\phi : \{0,1\}^n \to \{0,1\}$, and we want to either count the number of solutions to ϕ (i.e., $|\phi^{-1}(1)|$) or sample uniformly from the set of solutions to ϕ.

Doing either task "perfectly", i.e. without any error, is often computationally "hard". For example if ϕ is a Boolean formula, then we are asking for the number of satisfying assignments, which is at least as hard as asking whether the formula is satisfiable. However, as we will see later, in many situations where exact solutions are hard, an approximate solution can be obtained in polynomial time.

This raises the question of whether we should strive for an additive or a multiplicative approximation. Specifically, suppose that Domain$(\phi) = 2^n$, so that the number we wish to compute, θ is between 0 and 2^n. Then the question is whether we are trying to produce an approximation between $\theta \pm 2^n\varepsilon$, or between $\theta(1 \pm \varepsilon)$. The latter is more demanding, and generally more meaningful; this is especially true if the solution-space that we are interested in represents an exponentially small fraction of the total sample space, where a multiplicative approximation gives us an upper limit on our percent error, while an additive approximation does not accomplish that.

Note: Since a poly-time algorithm for ϕ exists, one possibility is to do rejection sampling - i.e., generate a random vector x in $\{0,1\}^n$, check if $\phi(x) = 1$ and if so return x, and otherwise repeat. While this will give us a uniform distribution on the set of $\phi^{-1}(1)$, the number of samples required may become exponentially large if the relative proportion of solutions to the sample space is very small. Thus rejection sampling is not a viable alternative in many cases. Frequently, in fact, we will be interested in the case that θ is exponential

in n, yet an exponentially small fraction of 2^n. Then rejection sampling does not run in poly-time – and on the other hand even if we could suppose that we had an algorithm to list the elements of $\phi^{-1}(1)$, in order, and each in polynomial time, even that would not give us a method of sampling from $\phi^{-1}(1)$ uniformly (or of counting it's elements).

Unless we say otherwise, we'll henceforth be interested in multiplicative approximation. There's a term for this: a *fully polynomial randomized approximation scheme* (FPRAS) for ϕ is a randomized algorithm which, in time polynomial in n and $1/\varepsilon$, produces with probability at least $3/4$ a multiplicative $1 \pm \varepsilon$ approximation to θ.

For example, we've seen a FPRAS for DNF formulas, but we note not to expect one for CNF formulas since the existence problem ("does this formula have a satisfying assignment") is NP-complete for CNF formulas.

We're now interested in relating the counting task to the task of approximate sampling. A perfect sampling procedure would enable us to sample from the distribution p that is exactly uniform on $\phi^{-1}(1)$; in approximate sampling we're willing to sample from a distribution q that is, in L_1 distance, within ε of p (so $||p-q||_1 = \sum_x |p(x) - q(x)| \leq \epsilon$). So, we have two computational tasks, counting and sampling, in both of which we've allowed some "distortion"; we're interested in this lecture in relating the complexities of these two tasks.

Definition: A search/sampling/counting problem is self-reducible if $\exists$ a representation of n bits for the objects defined by problem P of size m s.t. for $i = 0, 1$, the set of objects whose first bit is i are the objects of another problem P_i of size $< m$ **and** P_i is polynomial-time computable from P.

Examples of self-reducible problems:

1. SAT: P is the boolean formula, objects are the satisfying assignments, m is the formula size, n is the number of variables.

2. Set of matchings of an undirected graph: P is the graph, objects are the matchings, n is proportional to the number of edges, restriction is to choose an edge e between vertices u and v, P_0 is $P - e$, P_1 is $P - \{u, v\}$.

Potential counterexamples (non-self-reducible problems) discussed in class are problems whose structure is not preserved under restriction (at least not under any obvious restriction), such as the set of prime numbers less than N, or the set of 3-colorings of a graph.

Theorem (Jerrum, Valiant, V. Vazirani): TFAE for a self-reducible problem:
1. $\forall k$ $\exists$ a polynomial-time algorithm that samples from a distribution p on the "objects" (the elements of $\phi^{-1}(1)$ s.t. if u is the uniform distribution on these objects, then $||p-u||_1 \leqslant n^{-k}$.
2. $\forall k$ $\exists$ a polynomial-time algorithm that estimates the number of objects to within a multiplicative factor $1 + n^{-k}$, except for an error that occurs with probability n^{-k}.

Proof: In this proof we'll ignore the distinction between n and m.

First we will show that $1 \to 2$.
We assume sampling to within $O(n^{-(k+1)})$ L_1 distance of u. Now consider any root to leaf path. After $i-1$ steps, we will be at an internal node, designated a, and it will have two children - b and c. Let r_i be the actual fraction of solutions to problem a that are in the subtree c. Notice that the total number of leaves in the tree equal to $|t| = \prod_{i=1}^{k-1} \frac{1}{r_i}$, which leads us to the following algorithm: Sample from the problem at node a until the fractions of leaves in nodes b and c can be estimated to within additive n^{-k-1} with probability of error n^{-k-1}. Then proceed to the child (b or c) in which this estimate q_i is at least $1/2$ (recall that the tree is binary). After finally reaching a leaf, set Estimate$= \prod_{i=1}^{k-1} \frac{1}{q_i}$. Then if t is the true number of leaves, then unless an error event has happened somewhere (which by a union bound is at most n^{-k}), $t/Estimate = \prod_1^{k-1} \frac{q_i}{r_i} \leq (1 + O(n^{-k-1}))^n \leq 1 + n^{-k}$. And for the same reason, and with the same bound on the probability of error, $Estimate/t \leq 1 + n^{-k}$.

Now we will show that $2 \to 1$.
At node P, estimate the number of leaves of P, P_0 and P_1 to within a multiplicative factor of $1 + O(n^{-k-1})$

with an error probability of $O(n^{-k-1})$. We'll call these estimates q, q_0 and q_1. Note that in the self-reducibility tree the root is located at level 0 and the leaves are located at levels $\leq n$. To make the rest of the proof technically simpler to state suppose that any leaves at depth $< n$ are replaced by a chain of nodes all the way down to a single leaf at depth n; so now all leaves are at depth n. Consider an internal node of the tree, and let p, p_0 and p_1 be the true fractions of leaves in the corresponding subtrees. We'll show by induction that at level l, $||p - q||_1 \leq ln^{-k-1}$. The case $l = 0$ is obvious. Induction:
Let the internal node be labeled v and let its children be v_1 and v_2. Let $p_v - q_v = r$. Then in the worst case $\frac{p_{v_1}}{p_v} = \frac{q_{v_1}}{q_v} - O(n^{-k-1})$ and $\frac{p_{v_2}}{p_v} = \frac{q_{v_2}}{q_v} - O(n^{-k-1})$. So $p_{v_2} - q_{v_2} \leq (q_v + r)(\frac{q_{v_2}}{q_v} + O(n^{-k-1})) - q_{v_2} = q_v O(n^{-k-1}) + r(\frac{q_{v_2}}{q_v} + O(n^{-k-1}))$. By combining v_1 and v_2 we obtain $|(p_{v_2}, p_{v_1}) - (q_{v_2}, q_{v_1})|_1 \leq qO(n^{-k-1}) + r\frac{q_{v_2}+q_{v_1}}{q_v} + rO(n^{-k-1})$. Summing over the entire level l results in $||p - q||_1^l \leq ||p - q||_1^{l-1}(1 + O(n^{-k-1})) + O(n^{-k-1}) \leq ln^{-k-1}$. This completes the proof of the induction. Setting $l = n$ gives us the second part of the theorem.
QED