## 3.4 #DNF

A formula $\phi(x_1, \ldots, x_n)$ in DNF is of the form

$$\phi(x_1, \ldots, x_n) = \bigvee_{i=1}^{k} \left( \bigwedge_{j=1}^{r_i} l_{ij} \right)$$

where every "literal" $l_{ij}$ is one of $x_1, \ldots, x_n, \bar{x}_1, \ldots, \bar{x}_n$, (each $x_i$ is a Boolean variable). The computational complexity of satisfiability for DNF is trivial. This is because unless all the individual conjunctive clauses are individually non-satisfiable (which occurs for clause $C_i$ only if it contains both $x_j$ and $\bar{x}_j$ for some $j$), one can select values for the $x_i$'s such that at least one clause evaluates to 1. However, the problem of determining the existence of $\{x_i\}_{i=1}^n$ such that $\phi(x_1, \ldots, x_n) = 0$ is difficult.

Compare with Conjunctive Normal Form (CNF), the format we use for 3SAT: here the satisfiability problem is difficult, but determining the existence of $(x_i)_{i=1}^n$ such that $\phi(x_1, \ldots, x_n) = 0$ is trivial. DNF and CNF can be related to each other by the distributive law. How does this square with the disparity in computational hardness? Distributing a CNF to convert it to a DNF can blow up its representation exponentially. Verify that both sides of the following equation are minimal, and that written out in full, the LHS has size polynomial in $n$, and the RHS exponential in $n$.

$$\wedge_{i=1}^{n}(x_{1,i} \vee x_{2,i}) = \vee_{\varepsilon \in \{1,2\}^n} \wedge_{i=1}^{n} x_{\varepsilon_i, i}$$

The function $\#DNF : \phi \to \mathbb{N}$ is defined for formulas $\phi$ in DNF to be the number of inputs $x = (x_i)_{i=1}^n$ satisfying $\phi$. This is a *counting problem*, not a decision problem.

$\#DNF \in \#P$, where $\#P$ is the set of functions $\{0,1\}^n \to \mathbb{N}$ which which give the number of satisfying leaves in the tree of a nondeterministic poly-time Turing machine. Or more formally:

**Definition 28.** *A function $f : \{0,1\}^n \to \mathbb{N}$ is in #P if there is a deterministic-polytime predicate $\phi$ and a natural number $k$ such that*

$$f(x) = \sum_{r \in \{0,1\}^{n^k}} \phi(x, r)$$

(Comment: it doesn't matter for the definition whether the range of $\phi$ is $\{0,1\}$ or the nonnegative integers expressed in binary.)

$\#DNF$ can be shown to be $\#P$-complete with respect to deterministic poly-time reductions.

A counting class is not formally comparable to a decision class but roughly speaking, #P is more powerful than NP, since a #P machine can count accepting paths of a nondeterministic machine rather than just determining whether the number is nonzero; this comparison can be formalized by considering the class of languages decidable by a poly-time machine with access to a subroutine for a #P-complete language. This class is called $P^{\#P}$, and it contains NP (and a great deal more). See Fig. 3.1.

Other examples:

- #linear extensions (linear extensions to a partial order) is in #P
- Permanent of a nonnegative integer matrix is in #P

These are also complete (with respect to deterministic-polytime reductions) for #P. We'll settle for an approximate counting algorithm for #DNF.

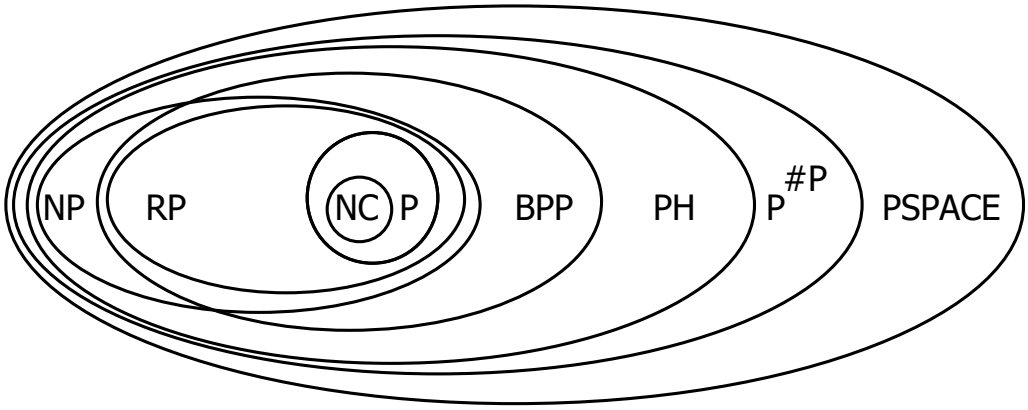


Figure 3.1: Some commonly used complexity classes

## Additive vs. Multiplicative Approximation

Let $\theta = \theta(\phi)$ be the fraction of assignments satisfying $\phi$. Therefore $\theta = |\{x : \phi(x) = 1\}|/2^n = \#DNF(\phi)/2^n$.

An *additive approximation* $T$ satisfies $|T - \theta| < \epsilon$.

A *multiplicative approximation* $T$ satisfies $\theta(1-\epsilon) < T < \theta(1+\epsilon)$.

When our unknown quantity is in the interval $[0, 1]$, then a multiplicative approximation implies an additive approximation, but not vice versa.

Example: Let $\phi(x) = (x_1 \wedge \ldots \wedge x_{n/3}) \vee (x_{n/3+1} \wedge \ldots \wedge x_{2n/3}) \vee (x_{2n/3+1} \wedge \ldots \wedge x_n)$. Then $\theta \approx 2^{-n/3}$. An additive approximation of $1/\text{poly}(n)$ is useless as a multiplicative approximation.

Here is a simple estimator $S$ for $\theta$: sample an assignment $\tau$ of $x$ uniformly and check for satisfiability, scoring $S = 1$ if yes and $S = 0$ otherwise. $S$ is unbiased (which means that $E(S) = \theta$), but its variance is huge, namely, $\theta(1-\theta)$. For $\theta \ll 1$ the standard deviation of $S$, $\sqrt{\theta(1-\theta)}$, is far larger than $\theta$. To get an estimate of $\theta$ to within constant factor by repetition, we would need (remember Chebyshev inequality) about $\frac{\text{Var}(S)}{E(S)^2} = \frac{\theta(1-\theta)}{\theta^2} \cong \frac{1}{\theta}$ repetitions.

## Multiplicative approximation for $\theta$

This algorithm will later be useful for network reliability problems, in particular the probability that a network remains connected when links fail, independently, with some probability.

With that and other applications in mind, we will address a slightly more general problem than stated above. Now for an arbitrary $0 \leq p \leq 1$, each variable is independently 1 with probability $p$. $\theta$ is still defined as the probability that the formula equals 1, but with respect to the "$p$ distribution".

However, for simplicity of presentation, we will address only the case that the formula is monotone. This would be a severe restriction if it was a genuine requirement. But we are only making this assumption in order to simplify some expressions, and you can work out the variant of the algorithm that makes sense when monotonicity is not required, and in fact also when each variable is distributed with its own private $p_i$ (but still independently of one another!).

Let $\phi = C_1 \vee C_2 \vee \ldots \vee C_k$, where the $C_i$'s are the clauses.

Let $r_i$ be the number of literals in clause $C_i$. Define $q = \sum_i p^{r_i}$. Notice that $q$ is the union bound estimate for $\theta$. So it is always an upper bound, but generally isn't very tight. What we can say is that for $k$ clauses, if we write $r = \min_i r_i$ then

$$p^r \leq \theta \leq q \leq kp^r \tag{3.16}$$

For an assignment $\tau$ of $x$, let $\mu(\tau)$ be the number of clauses that $\tau$ satisfies.

Let $P(\tau) = p^{|\tau|}(1-p)^{n-|\tau|}$, i.e., the probability of $\tau$ in the distribution with 1's sampled independently with probability $p$.

**Algorithm 4:** Karp, Luby, Madras #DNF approximation (for monotone case) [33]:

**Input:** DNF formula with $k$ "$\wedge$ clauses" $C_i$ of sizes $r_i$.
**Output:** Estimator $T$
1. Choose a clause $C_i$ with probability $p^{r_i}/q$.
2. Choose $\tau$ from the distribution $P(\tau)$ subject to the condition of satisfying $C_i$.
3. Let $T$, the estimator of $\theta$, be $T = q/\mu(\tau)$

**Lemma 29.** *$T$ is an unbiased estimator of $\theta$.*

*Proof.*

$$\begin{aligned}
E\left(\frac{q}{\mu(\tau)}\right) &= \sum_i \frac{p^{r_i}}{q} E\left(\frac{q}{\mu(\tau)} \mid \text{chose } C_i\right) \\
&= \sum_i \frac{p^{r_i}}{q} \sum_{\tau: C_i(\tau)=1} \frac{P(\tau)}{p^{r_i}} \frac{q}{\mu(\tau)} \quad \text{cond. prob. depends only on bits outside } C_i \\
&= \sum_i \sum_{\tau: C_i(\tau)=1} \frac{P(\tau)}{\mu(\tau)} \\
&= \sum_{\tau:\phi(\tau)=1} \mu(\tau) \frac{P(\tau)}{\mu(\tau)} \\
&= \theta.
\end{aligned}$$

□

Let's see why this unbiased estimator $T$ is better than its predecessor $S$.

**Lemma 30.** $\mathrm{Var}(T) \leq (k-1)\theta^2$.

*Proof.* Recall $\mathrm{Var}(T) = E(T-\theta)^2 = E(T^2) - \theta^2$.

$$\begin{aligned}
E(T^2) &= \sum_i \frac{p^{r_i}}{q} \sum_{\tau: C_i(\tau)=1} \frac{P(\tau)}{p^{r_i}} \frac{q^2}{\mu^2(\tau)} \\
&= q \sum_i \sum_{\tau: C_i(\tau)=1} \frac{P(\tau)}{\mu^2(\tau)} \\
&= q \sum_{\tau:\phi(\tau)=1} \frac{P(\tau)}{\mu(\tau)}
\end{aligned} \tag{3.17}$$

We have from (3.16) that $q \leq k\theta$. So

$$\begin{aligned} E(T^2) &\leq k\theta \sum_{\tau:\phi(\tau)=1} \frac{P(\tau)}{\mu(\tau)} \\ &\leq k\theta \sum_{\tau:\phi(\tau)=1} P(\tau) \\ &= k\theta^2 \end{aligned}$$

Therefore

$$\mathrm{Var}(T) \leq (k-1)\theta^2. \tag{3.18}$$

□

**Amplification:**
The variance is reduced in two steps, (a) averaging, (b) median; the latter is shown in the next section (and was also on a homework).

1. Repeat the K-L-M procedure $m = \frac{k-1}{\epsilon^2\delta}$ times to get estimates $T_1, \ldots, T_m$ of $\theta$. Evaluate $\bar{T} = \frac{1}{m}\sum T_i$. Since the $T_i$ are independent, $\mathrm{Var}(\bar{T}) = \mathrm{Var}(T_i)/m = \frac{\epsilon^2\delta}{k-1}(k-1)\theta^2 = \epsilon^2\delta\theta^2$.

**Reminder:**
*Chebyshev Inequality:* If $A$ is a finite-mean real-valued rv, then $\Pr(|A - E(A)| \geq c\sqrt{\mathrm{Var}(A)}) \leq \frac{1}{c^2}$.

Consequence for the #DNF approximation algorithm: $P(|\bar{T} - \theta| \geq \epsilon\theta) \leq \delta$.

This means that with a little more than $k$ samples, we can already get a decent multiplicative estimate of $\theta$ with positive probability $1 - \delta$. The dependence on $\delta$ can still use improvement though. Next we'll show how to turn this into a "fully polynomial randomized approximation scheme" (FPRAS) for #DNF.

## 3.5 FPRAS

We've already given an algorithm for estimating #DNF which runs in time $\mathrm{poly}(n, \frac{1}{\epsilon}, \frac{1}{\delta})$ and that produces an unbiased estimator $T$ of $\theta$ satisfying:

$$\Pr[(1-\epsilon)\theta \leq T \leq (1+\epsilon)\theta] \geq 1 - \delta$$

We can do a little better in terms of the dependence on $\delta$. First, a general definition.

**Definition 31.** *Randomized algorithm $A$ is a* FPRAS *(fully polynomial randomized approximation scheme) for nonnegative quantity $\theta$ if it produces output $T$ and, given a parameter $\varepsilon > 0$ and an input of length $n$,*

- *$A$ runs in time $\mathrm{poly}(n, \frac{1}{\epsilon})$.*
- $\Pr[(1-\epsilon)\theta \leq T \leq (1+\epsilon)\theta] \geq \frac{2}{3}$.

**Lemma 32.** *Having a FPRAS implies that in time $\mathrm{poly}(n, \frac{1}{\epsilon}, \log\frac{1}{\delta})$ we can produce $T$ satisfying:*

$$\Pr[(1-\epsilon)\theta \leq T \leq (1+\epsilon)\theta] \geq 1 - \delta$$

In our algorithm from last time, we started with an algorithm to approximate #DNF, and amplified it using the Chebyshev inequality to shrink the variance below $\epsilon$, and then continued to shrink it below $\epsilon\delta$. The above lemma shows us that there is a way of avoiding going as far in the variance-reduction as we did last time, since we only need $2/3$ of the probability mass inside the $\theta(1 \pm \epsilon)$ range to apply the lemma.

This lemma was assigned as part of ps1-2 (the second "sheep problem"). Here anyway is the proof.

*Proof.* By assumption, we have a random variable $X$ which we can produce in time $\text{poly}(n, \frac{1}{\epsilon})$ with $\frac{2}{3}$ of the probability mass inside the range $\theta(1 \pm \epsilon)$. Collect $m = \frac{\log 1/\delta}{D(1/2\|1/3)}$ inependent samples $X_1, \ldots, X_m$, from this distribution.

Here it is useful to remember (see **??** for our earlier discussion) that $D(p\|q)$ is the "two-point Kullback-Lieber divergence" of the distribution $(p, 1-p)$ from $(q, 1-q)$, specifically $D(p\|q) = p\log(p/q) + (1-p)\log((1-p)/(1-q))$, and that it is the main term in the Chernoff bound which we saw in Theorem **??**: *If $X_1, \ldots, X_n$ are iid coins each with probability $q$ of being heads, the probability that the number of heads, $X = \sum X_i$, is $> pn$ (for $p \geq q$) or $< pn$ (for $p \leq q$), is $< \exp(-nD(p\|q))$.*

Select the median of $X_1, \ldots, X_m$ as the output. Each sample with probability $\frac{2}{3}$ is in the $\theta(1 \pm \epsilon)$ range. So

$$\Pr[|\text{Median}(X_i) - \theta| > \theta\epsilon] < e^{-mD(1/2\|1/3)} = \delta$$

□

Now our overall algorithm consists of $m$ applications of a variance-reduction step, which averages the samples, and one median calculation on the $m$ averages.

Next time we will discuss a randomized min-cut algorithm, and then put it together with the #DNF approximation algorithm, to solve the network reliability problem.