Enumerating the Propositions Equivalent to a Boolean Function

Nachum Dershowitz and Mitchell A. Harris

School of Computer Science, Tel Aviv University, Ramat Aviv, Tel Aviv 69978 Israel,
nachumd@tau.ac.il
and
Department of Computer Science, University of Illinois at Urbana-Champaign, 1304 W. Springfield Avenue, Urbana, Illinois 61801, USA, maharri@cs.uiuc.edu

Abstract. A general method is presented for counting the propositional formulas of a given size that are equal to an arbitrary boolean function. The number of formulas equivalent to each boolean function on v variables is calculated using a system of non-linear recurrence relations. For particular boolean functions are and for small values of v, simplification or exact computation is feasible. As a corollary, it is shown that the probability that a formula over one variable is satisfiable is asymptotically $1 - 1/2\sqrt{3} = 0.711325$.

1 Introduction

In exploring the NP-complete problem of testing satisfiability of propositional formulas, researchers have looked at upper bounds given by algorithms, reductions to convert these to others, asymptotics and thresholds. Here we take the strategy of exploring the data directly. How many formulas of a given size are satisfiable? Instead of trying to find bounds on probabilities, we take the approach of exact counting, where probabilities, asymptotics and thresholds can then be computed. Dershowitz and Lindenstrauss [1] use generating function techniques for counting boolean formulas of certain kinds. Harris [2] uses the counting method on a linear system of equations to count k-CNF formulas. Those results are applied here to the more general nonlinear system. Larson and Lindenstrauss [3] use another method to arrive at similar results.

First, from a description of the operations of "$\wedge$" and "$\vee$", we find a context free grammar that generates formulas for each boolean function. From this we can construct a system of nonlinear equations whose solution is a set of power series generating function that counts the number of formulas of a given size. We then use combinatorial techniques to solve the system and extract the counting function.

2 Syntax and Semantics

The counting problem we address corresponds to the decision problem SAT. The data is described as follows: we have a set V of v independent propositional *variables*, a set $\overline{V}$ of v of their negations. The variables and negations are called *literals*. A *propositional formula* of size n is a full, ordered, binary tree with $n+1$ literals at the leaves, and a conjunction '$\wedge$' or disjunction '$\vee$' at each of n internal nodes. Notice that this syntax is not the most general; internal negations are not permitted. Pushing them down to the leaves using DeMorgan's law preserves the semantics and the structure, and makes the analysis cleaner. There are a total of

$$\mathrm{Tot}(n, v) = 2^n (2v)^{n+1} \frac{1}{n+1} \binom{2n}{n} \tag{1}$$

distinct formulas of size n on v variables, the first factor for the boolean operations at the internal nodes, the second for the literals at the leaves, and the last for the distinct trees (counted by the Catalan numbers).

A formula is *satisfiable* iff there exists an assignment of true and false to the variables such that the formula evaluates to true upon application of the operations at the internal nodes. There are a total of

$$2^{2^v} \tag{2}$$

boolean functions on v variables. So no matter how large the formula, the set of possible functions is dependent only on the number of variables. This discrepancy leads to the basic result that there are just not enough formulas of polynomial size for every function: for n to be large enough to realize all the boolean functions on v, we need to equate formulas 1 and 2

$$2^{2^v} = 2 \cdot v^{n+1} \frac{16^n}{n^{3/2}\sqrt{\pi}}$$

and solve for n as a function of v, which is roughly

$$n \in O(2^v / v).$$

For a given number of variables, we notate a function by its truth table of length 2^n. For $v = 1$, there are 4 functions, indexed as f_{00}, f_{01}, f_{10}, and f_{11}, which we will label for convenience $\mathbf{F}$, $\overline{\mathbf{P}}$, $\mathbf{P}$, and $\mathbf{T}$, respectively.

For every one of the 2^{2^v} boolean functions, we will count the number of formulas that evaluates to each function.

For the disjunction operator '$\vee$', the function produced depends on the functions represented by its operands. For example, '$\vee$' has the behavior for boolean functions in Equation 3. The binary function '$\wedge$' has the obvious dual behavior.

$$\begin{array}{l} f_{00} = \mathbf{F} \\ f_{01} = \overline{\mathbf{P}} \\ f_{10} = \mathbf{P} \\ f_{11} = \mathbf{T} \end{array} \qquad \begin{array}{c|cccc} \vee & \mathbf{F} & \overline{\mathbf{P}} & \mathbf{P} & \mathbf{T} \\ \hline \mathbf{F} & \mathbf{F} & \overline{\mathbf{P}} & \mathbf{P} & \mathbf{T} \\ \overline{\mathbf{P}} & \overline{\mathbf{P}} & \overline{\mathbf{P}} & \mathbf{T} & \mathbf{T} \\ \mathbf{P} & \mathbf{P} & \mathbf{T} & \mathbf{P} & \mathbf{T} \\ \mathbf{T} & \mathbf{T} & \mathbf{T} & \mathbf{T} & \mathbf{T} \end{array} \tag{3}$$

3 A System of Equations

Because of the syntax, the set of formulas is straightforward to specify as a context free language. What is more useful is that a context free grammar can be given that preserves the semantics. From the table of a logical operator acting on boolean functions such as Equation 3, one can construct this grammar as in Table 4.

$$\begin{aligned}
&\mathbf{F} = \mathbf{F} \vee \mathbf{F}\\
&\overline{\mathbf{P}} = \mathbf{F} \vee \overline{\mathbf{P}} + \overline{\mathbf{P}} \vee \mathbf{F} + \overline{\mathbf{P}} \vee \overline{\mathbf{P}} + \text{"}\neg p\text{"}\\
&\mathbf{P} = \mathbf{F} \vee \mathbf{P} + \mathbf{P} \vee \mathbf{F} + \mathbf{P} \vee \mathbf{P} + \text{"}p\text{"}\\
&\mathbf{T} = \mathbf{F} \vee \mathbf{T} +\\
&\quad\ \ \overline{\mathbf{P}} \vee \mathbf{P} + \overline{\mathbf{P}} \vee \mathbf{T} +\\
&\quad\ \ \mathbf{P} \vee \overline{\mathbf{P}} + \mathbf{P} \vee \mathbf{T} +\\
&\quad\ \ \mathbf{T} \vee \mathbf{F} + \mathbf{T} \vee \overline{\mathbf{P}} + \mathbf{T} \vee \mathbf{P} + \mathbf{T} \vee \mathbf{T}
\end{aligned} \tag{4}$$

This translates immediately, by the Chomsky-Schutzenberger correspondence between context free grammars and formal power series (as generating functions), to a system of equations. This system implicitly defines the set of power series where the coefficient of z^n is the number of words in the language of length n, that is the number of propositional formulas of size n. The system of power series, including the terms corresponding to productions involving both "$\wedge$" and"$\vee$", is given in Equation 5.

$$\begin{aligned}
\mathbf{F}(z) &= z(2\mathbf{F}(z)(\mathbf{T}(z) + \mathbf{P}(z) + \overline{\mathbf{P}}(z) + \mathbf{F}(z)) +\\
&\qquad 2\overline{\mathbf{P}}(z)\mathbf{P}(z))\\
\overline{\mathbf{P}}(z) &= z2\overline{\mathbf{P}}(\mathbf{F}(z) + \overline{\mathbf{P}}(z) + \mathbf{T}(z)) + 1\\
\mathbf{P}(z) &= z2\mathbf{P}(\mathbf{F}(z) + \mathbf{P}(z) + \mathbf{T}(z)) + 1\\
\mathbf{T}(z) &= z(2\mathbf{T}(z)(\mathbf{T}(z) + \mathbf{P}(z) + \overline{\mathbf{P}}(z) + \mathbf{F}(z)) +\\
&\qquad 2\overline{\mathbf{P}}(z)\mathbf{P}(z))
\end{aligned} \tag{5}$$

Note that a solution for this system is a solution for a system based on '$\wedge$' but with functions ordered in reverse (the complement of the bitwise representation).

The grammar and corresponding system of equations can be generated recursively for arbitrary v (see [2] for details of the construction).

From this system of equations we would like to extract the generating function $\mathbf{F}(z)$ and find a closed form for its coefficients.

4 The Groebner basis method

Systems of equations arising from context free grammars, such as the one above, are inherently non-linear. The system is still amenable to quantifier elimination. Since the system is over a multivariate polynomial ring (the variables being the counting power series for the boolean functions), we can use Groebner basis completion as a mechanical procedure to attempt to eliminate all but one variable.

This results in the following single fourth degree equation in $\mathbf{F}(z)$:

$$\begin{aligned}\mathbf{F}(z) = 32\mathbf{F}(z)^4 z^3 - 32\mathbf{F}(z)^3 z^2 - 16\mathbf{F}(z)^2 z^2 +\\ 10\mathbf{F}(z)^2 z + 8\mathbf{F}(z) z + 2z\end{aligned}$$

The monomial ordering that allows such variable elimination is the lexicographic ordering, i.e. the function symbols (as variables) are given an ordering, and then two monomials are compared in order of these symbols, along with the degree.

The generating function for $\mathbf{F}(z)$ is one of the roots of Equation 6. The root chosen is that which gives nonegative integer coefficients, namely

$$\mathbf{F}(z) = \frac{1 - \sqrt{\dfrac{1 + 8z + \sqrt{1 - 16z}}{2}}}{4z}$$

From this we need to extract a closed form for the coefficients. The generating function for the inner square root $\sqrt{1 - 16z}$ can be found either in analogy with that for some binomials, or by solving the recurrence implied by successive derivatives from the Taylor expansion:

$$\sqrt{1 - 16z}^{(n)} = \sqrt{1 - 16z}^{(n-1)} 16(n - \frac{1}{2})$$

Along with the other terms inside the outermost square root, this gives:

$$\frac{1 + 8z + \sqrt{1 - 16z}}{2} = 1 - \sum_{k \geq 2} z^k 2^{2k} \binom{2k - 2}{k - 1} / k.$$

So,

$$\begin{aligned}4z\mathbf{F}(z) &= 1 - \sqrt{1 - \sum_{k \geq 2} z^k 2^{2k} \binom{2k - 2}{k - 1} / k}\\ &= 1 - \sum_{n \geq 0} \binom{1/2}{n} \left(- \sum_{k \geq 2} z^k 2^{2k} \binom{2k - 2}{k - 1} / k \right)^n\\ &= 1 + \sum_{n \geq 0} \frac{2^{-2n}}{2n - 1} \binom{2n}{n} \left(\sum_{k \geq 2} z^k 2^{2k} \binom{2k - 2}{k - 1} / k \right)^n\end{aligned}$$

The inner Catalan summation raised to an arbitrary power, has a convenient identity removing the exponent:

$$\left(\sum_{k \geq 2} 2^{2k} \binom{2k - 2}{k - 1} \frac{z^k}{k} \right)^n = \sum_{k \geq 0} \frac{2^{2k} n \binom{2k + 2n - 1}{k}}{(k + 2n)} z^k$$

Then the derivation follows in the numerator:

$$\begin{aligned}4z\mathbf{F}(z) &= 1+\sum_{n\geq 0} 2^{-2n}\binom{2n}{n}\frac{1}{2n-1}\sum_{k\geq 0} z^k 2^{2k}\frac{n}{k+2n}\binom{2k+2n-1}{k}\\ &= \sum_{n\geq 0}\sum_{k\geq 0} z^k\binom{2n}{n}2^{2k-2n-1}\frac{n}{k+2n}\frac{1}{2n-1}\binom{2k+2n-1}{k}\\ &= \sum_{n\geq 1} z^n\sum_{k\geq 0}\binom{2k}{k}\frac{1}{2k-1}2^{2n-2k-1}\frac{k}{n}\binom{2n-2k-1}{n-2k}\end{aligned}$$

Let $\mathbf{F}(n)$ be the coefficient for z^n, $n \geq 1$:

$$\mathbf{F}(n) = \sum_{k\geq 0}\binom{2k}{k}\frac{1}{2k-1}2^{2n-2k+1}\frac{k}{n+1}\binom{2n-2k+1}{n-2k+1} \tag{6}$$

This is a closed form formula for the number of propositional formulas of size n over one variable that are tautologies. From this formula for $\mathbf{F}(n)$, we would like to examine its behavior with respect to the total number of formulas, $\text{Tot}(n, v)$:

n	0	1	2	3	4	5	6
$\mathbf{F}(n)$	0	2	16	168	1920	23360	296448
$\text{Tot}(n,1)$	2	8	64	640	7168	86016	1081344

The ratio of successive terms $k+1$ to k in $\lim_{\mathbf{F}}(n)/\text{Tot}(n,1)$ (from equation 1) and is $\frac{1}{4}\frac{k+1/2}{k+1}$ from which we can read directly the hypergeometric formula

$$\lim_{n\to\infty}\mathbf{F}(n) = \frac{1}{4}F\left(\left.{1/2 \atop -}\right|\frac{1}{4}\right) = \frac{1}{4}\sum_{k\geq 0}\frac{\frac{1}{2}^{\overline{k}}}{k!}\left(\frac{1}{4}\right)^k$$

A basic hypergeometric and generating function identity says that

$$F\left(\left.{a \atop -}\right| z\right) = \sum_{k\geq 0}\frac{a^{\overline{k}}}{k!}z^k = \sum_{k\geq 0}\binom{a+k-1}{k}z^k = \frac{1}{(1-z)^a}$$

which gives

$$\begin{aligned}\lim_{n\to\infty}\mathbf{F}(n) &= \frac{1}{4}\frac{1}{(1-\frac{1}{4})^{\frac{1}{2}}}\\ &= \frac{1}{2\sqrt{3}} \approx 0.288675\end{aligned}$$

Since $F(n)$ is the same, the ratio of satisfiable to all formulas as n goes to infinity converges to the constant

$$1-\frac{1}{2\sqrt{3}} \approx 0.711325.$$

5 Larger number of variables

When there are two variables, there are sixteen equations in sixteen unknowns. The number of terms in each equation corresponding to each operator is 256, for a total of 512 (for both $\wedge$ and $\vee$) but some simplification occurs because of commutativity, resulting in (the general case):

$$\begin{aligned}
&\sum_{k=0}^{2^v} \binom{2^v}{k} \left(\frac{3^k - 1}{2} + \frac{3^{2^v - k} - 1}{2} + 1 \right) \\
&= \sum_{k=0}^{2^v} \binom{2^v}{k} \left(\frac{3^k + 3^{2^v - k}}{2} \right) \\
&= 1/2(4^{2^v} + 4^{2^v}) = 4^{2^v}
\end{aligned}$$

Of course, by symmetry (the complements are counted equally), we can reduce the total number of equations and unknowns by half. Groebner basis completion or any kind of elimination procedure has a doubly exponential lower bound in the number of variables (with the nontrivial number of terms computed above). Supposing that despite this infeasibility, we can eliminate all but one unknown, then we would have to solve for that one variable. Since the degree of that equation is certainly higher than four, extracting roots is not automatic and would require difficult ad hoc methods. And even knowing which root is appropriate and in an elementary form (not hypergeometrics), extracting the coefficients from that is again another intractable process.

6 Comments

We are not limited to the operators "$\wedge$" and "$\vee$". We can determine the system of equations from any set of boolean operators, from their actions on the boolean functions. For example, "$|$" (nand) or even the unary "$\neg$" can be used in the grammar. In principle, more variables can be used. For $v = 2$, there are 16 equations to solve for; this is infeasible for Groebner basis elimination. For $v = 3$ there are 256 equations with $< 256^2 = 65536$ terms; even successive approximation quickly exhausts space resources. But the system is highly structured and so then should all the reductions that follow despite their apparent intractability for machines.

Exacting counting of formulas for all boolean functions will make it easier to find properties of the probabilities: asymptotics, lower bounds, and thresholds. Extending these techniques to handle nontrivial numbers of variables is the next thing to explore.

References

[1] N. Dershowitz and N. Lindenstrauss. Average time analyses related to logic programming. In *Logic programming (Lisbon, 1989)*, pages 369–381. MIT Press, Cambridge, MA, 1989.

[2] M. A. Harris. Counting satisfiable k-cnf formulas. In T. Walsh, editor, *Principles and Practice of Constraint Programming- CP 2001*, volume 2239 of *LNCS*, page 765. Springer, 2001.

[3] M. J. Larson and N. Lindenstrauss. personal communication.