EXPLORING BOUNDARY LOGIC 3SAT ALGORITHMS

William Bricken
February 1998

What follows is an exploration of P=NP.

APPROACH

The task is to identify whether or not an arbitrary 3-CNF form is satisfiable. 3-DNF is used here for notational convenience.

The strategy is to count all of the 3-DNF forms that are satisfiable (alternatively count all forms that are either tautological or contradictory and subtract that from the total number of possible forms). The number of forms is parameterized by the number of variables available for a particular N-DNF form. This count reveals structural constraints on forms which are not-satisfiable, and these structures are identifiable by polynomial counting algorithms.

To illustrate the principles, we begin with the simpler cases of 0-DNF, 1-DNF and 2-DNF to illustrate counting principles.

Degenerate example of 0-DNF

Consider the degenerate case of no variables:

```
     <void>   = False
not(<void>)  = True
```

We have no combinatorics, only complementation.

Preliminary example of 1-DNF with 1 variable

Consider the simple case of all possible DNF Boolean forms with one literal and one variable:

```
a  (a)
```

There are four (2^2^1) disjunctive combinations of these two (2^1) forms:

```
<void>      =  False
 a
(a)
 a (a)      =  True
```

These four forms define the entire space of Boolean expressions of one variable.

The Boolean value of an expression is determined (i.e. not-satisfiable) in only two cases:

1) when then are no expressions = False

2) when the two complemented forms of a variable share the same expression, i.e. A (A) = True

RELEVANCE

The forms of a Boolean expression occur at four levels, each a proper subset of the next:

value
functions
combinations
infinite combinations

The value of forms is relevant to the SAT/TAUT question. The functional form of a Boolean expression is relevant to Boolean minimization and circuit synthesis. The combinations of forms is relevant to counting rules and characterizations of Boolean spaces. The infinite combination approach is relevant to language generation approaches.

My attitude is that infinite forms and other highly redundant expressions (eg: False or False or False or ...) are conceptually misleading, never encountered, and without theoretical value. That is, the language generator approach (upon which much of complexity theory is based) is decrepit.

Combinational, group theoretic, and counting approaches are proposed to replace infinite generators as foundational.

Functional invariance is the pragmatic goal, and all analytic efforts are constrained to eliminating combinational diversity in favor of functional invariance.

Within this framework, the space of values of Boolean expressions is comprehensible. One immediate advantage may be a proof that P=NP.

COUNTING

We propose four distinct ways of counting Boolean forms:

Power-of-2: The number of possible conjunctive unit forms of n variables is (2^n). This can be understood as the vertices of a Boolean hypercube in n-space. The lattice join defines presence or absence of a particular variable in the unit form, where
presence is determined by the Boolean truth under conjunction. This structure for 3 variables:

```
                     ( a  b  c )
      ( a  b (c))    ( a (b) c )    ((a) b  c )
      ( a (b)(c))    ((a) b (c))    ((a)(b) c )
                     ((a)(b)(c))
```

Bounded atoms are present, unbounded atoms are not (that is, their negation is present). The lattice has lub of the easiest form to evaluate as True, glb as hardest form, i.e. all atoms must be present. Meet is defined as positive presence, join as negative presence.

The functionally invariant forms of n variables are defined by the number of possible disjunctive combinations of conjunctive unit forms (thus the N-DNF formalism). For n variables, the count of discrete Boolean functions is 2^(2^n). The formula 2^N, where N=(2^n) defines a Boolean hypercube of unit forms. This meta-hypercube is the space of Boolean functions. It can be interpreted as a distributive, complemented (Boolean) lattice of conjunctive units. The meta-hypercube has a portion of the unit hyper-cube at each node. Again, the join defines the presence or absence of components of the embedded unit hypercube.

In the following, the logical box convention is used: each node is a box, empty boxes are True. Embedded hypercubes at each node of the meta-hypercube are fractional. Join is of voids, meet is of existants.

This structure for 0 variables:

unit 0-cube as 2^0 = 1

```
              (     )
```

meta-hypercube as 2^(2^0) = 2, with void members marked in <>

```
              (<( )>)
              ( ( ) )
```

meta-hypercube in condensed notation

```
              (     )
              ( ( ) )
```

For 1 variable, the unit 1-cube as 2^1 = 2

```
                    ( (a) )
                    (  a  )
```

meta-hypercube as 2^(2^1) = 4, with void members marked in <>

```
                  ( <((a))>
                    <( a )> )
      ( <((a))>               (  ((a))
         ( a )  )               <( a )> )
                  (  ((a))
                     ( a )  )
```

meta-hypercube in condensed notation, void components omitted

```
                    (     )
         ((a))                    (a)
                    (a (a))
```

meta-hypercube in logically condensed notation

```
                    (     )
         ((a))                    (a)
                    ((   ))
```

For 2 variables, the unit hypercube as 2^2 = 4

```
                    ((a)(b))
             ((a) b )     ( a (b))
                    ( a  b )
```

The meta-hypercube as 2^(2^2) = 16, with void members marked as <>

```
                              (<((a)(b))>
                               <((a) b )>
                               <( a (b))>
                               <( a  b )>)

       (<((a)(b))>      (<((a)(b))>      (<((a)(b))>      ( ((a)(b))
        <((a) b )>       <((a) b )>        ((a) b )        <((a) b )>
        <( a (b))>        ( a (b))        <( a (b))>       <( a (b))>
         ( a  b ) )      <( a  b )>)      <( a  b )>)      <( a  b )>)

(<((a)(b))>  (<((a)(b))>  ( ((a)(b))   (<((a)(b))>  ( ((a)(b))   ( ((a)(b))
 <((a) b )>    ((a) b )    <((a) b )>    ((a) b )    <((a) b )>    ((a) b )
  ( a (b))    <( a (b))>   <( a (b))>    ( a (b))     ( a (b))    <( a (b))>
  ( a  b ) )   ( a  b ) )   ( a  b ) )  <( a  b )>)  <( a  b )>)  <( a  b )>)

       (<((a)(b))>      ( ((a)(b))       ( ((a)(b))       ( ((a)(b))
         ((a) b )        <((a) b )>        ((a) b )         ((a) b )
         ( a (b))         ( a (b))        <( a (b))>        ( a (b))
         ( a  b ) )       ( a  b ) )       ( a  b ) )      <( a  b )>)

                              ( ((a)(b))
                                ((a) b )
                                ( a (b))
                                ( a  b ) )
```

The meta-hypercube in condensed notation, voids omitted

```
                               (


                                          )

        (                (                (                ( ((a)(b))
                                            ((a) b )
                           ( a (b))
          ( a  b ) )                )                )                )

 (            (            ( ((a)(b))   (            ( ((a)(b))   ( ((a)(b))
                ((a) b )                  ((a) b )                  ((a) b )
   ( a (b))                               ( a (b))     ( a (b))
   ( a  b ) )   ( a  b ) )   ( a  b ) )            )            )            )

        (                ( ((a)(b))       ( ((a)(b))       ( ((a)(b))
          ((a) b )                          ((a) b )         ((a) b )
          ( a (b))         ( a (b))                          ( a (b))
          ( a  b ) )       ( a  b ) )       ( a  b ) )                )

                               ( ((a)(b))
                                 ((a) b )
                                 ( a (b))
                                 ( a  b ) )
```

The meta-hypercube in logically condensed notation

```
                               (          )

        ( ( a  b ) )     ( ( a (b)) )     ( ((a) b ) )     ( ((a)(b)) )

 ( ( a    ) ) ( (    b ) ) ( ((a)(b))   ( ((a) b )   (      b   ) (   a      )
                             ( a  b ) )   ( a (b)) )

        (  (a)(b)  )     (  (a) b   )     (   a (b)  )     (   a  b   )

                               ( (      ) )
```

The meta-hypercube in standard logical notation

```
                                   TRUE

            a OR b         FI a b          IF a b            a NAND b

        a            b          a ≠ b          a = b      NOT b      NOT a

            a AND b       NIF a b         NFI a b            a NOR b

                                   FALSE
```

The above has illustrated that the Boolean meta-hypercube is composed of existence-meet operations, first at the variable level to form the unit conjunction n-cube, then at the unit conjunction level to form the functional N-cube.

Void-power-of-2: An alternative way to understand the power rule is to imagine 2n spaces. Each space can either contain an object or not. We then count the number of ways to fill these spaces, with the peculiar void-rule that a void space is a particular state. This generates the following tableaux:

```
a  (a)        b  (b)    ...
1   2         3   4     ...2n
_   _          _   _
o   _     X   o   _
_   o         _   o
o   o         o   o
```

Since each space is binary, we have 2^2 possible configurations for each complemented variable. When a second variable is added, the product space is (2^2)*(2^2) = (2^2)^2 or 2^(2^2). A third variable creates a product space of size (2^2)^3 = 2^4 which is considerably smaller than 2^(2^3) = 2^8. We have counted the discrete forms, but not all possible combinations of forms.

The column labels in this interpretation are not solely variables and their complements, since the combinatorial product of void spaces represents presence or absence of the conjunction of each component. That is, the column label actually identifies the meaning of the entire column, with regard to other columns. Thus each row signifies the conjunction of present elements from each column.

One way to visualize this is to see the entire column to be the 1-space projection of the variable from N space. Thus, for a 3-cube,a variable is the plane (2-cube) of elements containing that variable. We need to account for all projections from 3-space, including 3-space onto 3-space, 3-space onto 2-

space, 3-space onto 1-space, and 3-space onto 0-space. This accounts for all the possible cross-product terms. There is one way to project 3-onto-3, and one way to project 3-onto-0. 3-onto-2 and 3-onto-1 each come in three varieties, for possible variables and for possible variable pairings. The total is 2^3 = 8 projections, and 2^8 possible combinations generated by all projections. Again in general we have 2^(2^n) possible states.

Binomial Expansion: Imagine a box with a ball for each conjunction of variables and their complements. In the case of 2 variables, we form the following units:

```
    NOR= (a b)     NIF= ((a) b)     NFI= (a (b))     AND= ((a)(b))
```

We place them in a box and select all possible combinations:

```
      |                     |
      | NOR  NIF  NFI  AND  |
     ----------------------

           choose[0,4] =  1
           choose[1,4] =  4
           choose[2,4] =  6
           choose[3,4] =  4
           choose[4,4] =  1
                 total = 16
```

This is analogous to the binomial expansion of (x + 1)^4, with powers of x being listings of the presence of each unit form. More properly, it would be written as

```
      (NOR +1)(NIF + 1)(NFI + 1)(AND +1)
```

with the precaution that addition and multiplication and the unit 1 are not Boolean (thus this is an analogy). For example, there are four "x^3" expressions, representing the four choose[4,3] unit form combinations

(NOR+NIF+NFI, NOR+NIF+AND, NOR+NFI+AND, NIF+NFI+AND).

In the 2-DNF case, the combinational forms each reduce to a named binary Boolean function.

In general, for n variables, we have (2^n) conjunctive forms, and (x + 1)^(2^n) binomial choices, for a total of 2^(2^n) binomial elements.

Literal Selection: This approach is useful for dealing with N-DNF forms having i variables, and is used later.

Just like the binomial expansion is an alternative expression of the power-of-2 Boolean hypercube, literal selection is the counting analogy to void-power-of-2. Imagine that there are n boxes with two objects each:

```
|     |
| o x |  ...
-------
   A
```

We select either 0, 1, or 2 or the objects.
Let ch2 be the function choose-from-2 and note that

```
ch2[0] = 1
ch2[1] = 2
ch2[2] = 1
```

Thus, the 4 elements of 1-DNF are the ways to choose a positive or negative literal from the box, k at a time. They can be counted as a binomial expansion: (1 + 2 + 1) = 4

which is summation[i, ch2[i]].

In general, the ways to choose from n literals is

summation[i, lim[k, i], binomial[n, ch2[k]]   ]

where k = {0, 1, 2}
n = number of variables {a, b, ...}
i = {0..2n}

ch2[k] = choose k objects from 2 (positive and negative variable cases)

binomial[n, ch2[k]] = form the specialized binomial expansion
(x + 1)^n where powers of x are the products of the ch2[k] terms for each variable.

lim[k, i] = each product term in the binomial over n variables is limited so that the sum of the ks in the choose terms is equal to i

I'm in unfamiliar notational territory here, but here's what the above means: form the binomial expansion term for i, with each variable being represented by the choose[2,k] for that variable. Build 2n of these terms, one for each i in {0..2n}, while limiting the sum of the k components to i. Examples follow in the sections on DNF forms with several available variables.

The reason for formulating this selection function is that it clearly indicates the degenerate (i.e., tautological) classes of possible Boolean unit combinations in the presence of extra variables.

One additional abbreviation: the product of ch2 terms over i variables looks like the following example of four variables with i = 5

```
ch2[2]ch2[2]ch2[1]ch2[0]
```

Here the sum of indices is 5, the four entries are selections from variable boxes {a,b,c,d}. This function will be written

```
chi[2,2,1,0]
```

There are only two unique ways to divide 5 selections over four variables, when the maximum of any variable is 2: chi[2,2,1,0] and chi[2,1,1,1]. Since variables are symmetric, we can count the permutations of the two chi functions to count the total possible ways to construct unique Boolean functions with five variables.

```
perm[ chi[2,2,1,0] ] = 12
perm[ chi[2,1,1,1] ] =  4
```

Facing the Actual Complexity

Thus far, we have selected only from pure variables, the difference between 2n choices and 2^n choices addressed under void-power-of-2. Recall that the literal names are just abbreviations for particular pairs of conjunctive units. chi[1,1] for instance refers to selecting one item from A from one from B, which can be done in four different ways:

```
 a  b     =  ((a) b ) ((a)(b)) ( a (b)) ((a)(b))
 a (b)    =  ((a) b ) ((a)(b)) ( a  b ) ((a) b )
(a) b     =  ( a  b ) ( a (b)) ( a (b)) ((a)(b))
(a)(b)    =  ( a  b ) ( a (b)) ( a  b ) ((a) b )
```

As is indicated, each of these literal configurations on the left is shorthand for a collection of four conjunctive units. None are tautologies because 1) it takes four units to cover the 2-space, and 2) each contains a duplicate which thus makes coverage impossible. The actual selection from A and B boxes is choose[4,2], in terms of conjunctive units:

```
| ((a) b )   ( a  b ) |   | ( a (b))   ( a  b ) |
| ((a)(b))   ( a (b)) |   | ((a)(b))   ((a) b ) |
-----------------------   -----------------------
          AasB                      BasA
```

But we can see another way of selection when there are two or more variables, taking "half" of a and "half" of (a). These are the diagonal selections. We cannot take solely the top or bottom halves from either, since this is tantamount to taking from the other variable box. Note that A and B have identical diagonal half-forms. This is the source of complexity in Boolean forms, we can take either diagonal from A, but must take the alternative diagonal from B. The half components of variables interact.

Aside: This analysis is built deeply into the Kauffman axiomatization, which is a single axiom formulation of boundary logic. Kauffman uses the following tableau:

```
              ((a) b )   ( a  b )    = (b)
              ((a)(b))   ( a (b))    =  b
                 =          =           =
                 a         (a)    =    True
```

The three variable case builds quarter components for each variable box:

```
      | ((a) b  c )   ( a  b  c ) |
      | ((a) b (c))   ( a  b (c)) |
      | ((a)(b) c )   ( a (b) c ) |  ...     ...
      | ((a)(b)(c))   ( a (b)(c)) |
      -----------------------------
                  AasBC                 BasAC      CasAB   ...
```

and a more complex diagonal interaction. The projection approach is to eliminate the specific variable from each box that matches the label and recur. Specific variable elimination makes the two columns in the box identical. Duplicates are removed so as to leave two columns which retain the literal distinction for the remaining variable. For instance:

```
  | (    b  c )   (    b  c ) |     |               (    b  c ) |
  | (    b (c))   (    b (c)) |     |               (    b (c)) |
  | (   (b) c )   (   (b) c ) |     | (   (b) c )                |
  | (   (b)(c))   (   (b)(c)) |     | (   (b)(c))                |
  -----------------------------  =  -----------------------------
        A-reduced-to-BC                         BasC

  | (    b  c )   (    b  c ) |     |               (    b  c ) |
  | (    b (c))   (    b (c)) |     | (    b (c))                |
  | (   (b) c )   (   (b) c ) |     |               (   (b) c ) |
  | (   (b)(c))   (   (b)(c)) |     | (   (b)(c))                |
  -----------------------------  =  -----------------------------
        A-reduced-to-BC                         CasB
```

Diagonals as Complexity

The objective is to isolate all complexities so that they can be identified by polynomial identification algorithms as non-tautological.

In the case of 2-space, there is one diagonal complexity:

```
      (a b) ((a)(b))  and its complement  (a (b)) ((a) b)
```

The two diagonals combine to cover the 2-space and thus form a tautology. However, they require disassembly of variables to be identified.

In 3-space, four paired diagonals are needed to form the tautology. Two types of diagonals occur (edge-diagonals and point-diagonals. The edge-diagonals are the 2-space diagonal pair for each variable (i.e. 3 permutations of four conjunctive units). For C, as an example

```
      ( a  b  c ) ((a)(b) c ) ( a  b (c)) ((a)(b)(c))
```

and complement

```
      ( a (b) c ) ((a) b  c ) ( a (b)(c)) ((a) b (c))
```

These are the diagonals taken as unique pairings from the 3-space literal box of C.

As well there is the point-diagonal, the unique diagonal introduced by the third dimension which does not reduce to any single variable:

```
      (a b c) ((a)(b)(c))
```

Four of these pairs are required to form the tautology. They are the four single unit selections for each variable box, in which the polarity of every variable differs (matched by numbering below):

```
      1 ((a) b  c )  ( a  b  c ) 4
      2 ((a) b (c))  ( a  b (c)) 3
      3 ((a)(b) c )  ( a (b) c ) 2
      4 ((a)(b)(c))  ( a (b)(c)) 1
```

Other complex forms in 3-space are derivable as projections, with the exception of the forms (x y z) ((x) y (z)) ((x)(y) z). These three unit forms cannot be combined with other *non-projective* forms to form a tautology. That is, they can be identified by the projective components of any other forms they can degenerately combine with.

Comment: this technique would be difficult to manage for N dimensions. However 3-SAT places a convenient ceiling on the dimensionalities of concern.

Counting by Reed-Muller Decomposition

The Reed-Muller normal form uses what we have been calling interactions as the basis set (at the cost of simple conjunctive forms). The DNF equivalent is the **equivalence polynomial** form.
For 3-space, the unit cube is the conjunctive powerset of the variables:

```
                   (         )
           ((a))      ((b))      ((c))
          ((a)(b))   ((a)(c))   ((b)(c))
                   ((a)(b)(c))
```

The price paid for the simpler unit cube is that the meet operation in the meta-hypercube is XOR, the most complex Boolean function. In the following example of the IFF-meta-hypercube, we pay this price in the logical simplification step.

```
      XOR-unit hypercube:
                       (      )
                   ((a))      ((b))
                       ((a)(b))
```

```
      4-space IFF-meta-hypercube, with voids marked as <>

                               (<(      )>
                                <   a    >
                                <   b    >
                                <((a)(b))>)

        (<(      )>      (<(      )>      (<(      )>      ( (      )
         <   a    >       <   a    >           a            <   a    >
         <   b    >           b            <   b    >       <   b    >
          ((a)(b)) )      <((a)(b))>)      <((a)(b))>)      <((a)(b))>)

 (<(      )>  (<(      )>  ( (      )   (<(      )>  ( (      )   ( (      )
  <   a    >      a         <   a    >      a         <   a    >      a
      b         <   b    >  <   b    >      b             b         <   b    >
   ((a)(b)) )   ((a)(b)) )   ((a)(b)) )  <((a)(b))>)  <((a)(b))>)  <((a)(b))>)

        (<(      )>      ( (      )       ( (      )       ( (      )
             a            <   a    >           a                a
             b                b            <   b    >           b
          ((a)(b)) )       ((a)(b)) )       ((a)(b)) )      <((a)(b))>)

                               ( (      )
                                    a
                                    b
                                 ((a)(b)) )
```

4-space IFF-meta-hypercube, with voids deleted

```
                               (


                                          )

        (                (                (                ( (      )
                                            a
                              b
          ((a)(b)) )                )                )                )

 (            (            ( (      )   (            ( (      )   ( (      )
                   a                         a                         a
      b                                      b            b
   ((a)(b)) )   ((a)(b)) )   ((a)(b)) )            )            )            )

        (                ( (      )       ( (      )       ( (      )
             a                                 a                a
             b                b                                 b
          ((a)(b)) )       ((a)(b)) )       ((a)(b)) )                )

                               ( (      )
                                    a
                                    b
                                 ((a)(b)) )
```

4-space IFF-meta-hypercube, with partial logical simplification

```
                               (          )

        ( ((a)(b)) )     (    b     )     (    a     )     ( (      ) )

 (    b       (    a       ((           (    a       ((           ((
   ((a)(b)) )   ((a)(b)) )   ((a)(b))))      b     )      b    ))      a    ))

        (    a           ((               ((               ((
             b                b                a                a
          ((a)(b)) )       ((a)(b))))       ((a)(b))))          b    ))

                               ((   a
                                    b
                                 ((a)(b))))
```

Above, the presence of a bare mark, when XORed with the other units in the node, has the effect of negating the node.

4-space IFF-meta-hypercube, with complete logical simplification

```
                         (          )

   ( ((a)(b)) )     (     b     )     (     a     )     ( (       ) )

(   a (b)  ) (  (a) b   ) (  (a)(b)  ) ( a IFF b  ) ((   b    )) ((   a    ))

   (   a  b   )     ((  a (b) ))     (( (a) b  ))     (( a IFF b ))

                         ((  a  b  ))
```

[NOTE: I haven't checked the integrity of meet-join for this. Meet is IFF.]

We now examine N-DNF forms without excess variables.

Counting 2-DNF without extra variables

Consider the 2-DNF basis with two variables and four literals (2^2):

```
a  (a)  b  (b)
```

To build the 16 Boolean functions of two variables, we form basis units by combining these four literals by using a conjunctive table:

```
           (a)            a

(b)     ( a  b )      ((a) b )

 b      ( a (b))      ((a)(b))
```

These basis cubes are combined by disjunction in all possible ways to generate the 16 Boolean functions of two variables.

Counting: (2^2^2) = (1 + 4 + 6 + 4 + 1)

```
<void>                                         =  False
( a  b )                                                 NOR
( a (b))                                                 NFI
((a) b )                                                 NIF
((a)(b))                                                 AND
( a  b ) ( a (b))                        =  (a)
( a  b ) ((a) b )                        =  (b)
( a  b ) ((a)(b))                                    XOR
( a (b)) ((a) b )                                    IFF
( a (b)) ((a)(b))                        =   b
((a) b ) ((a)(b))                        =   a
( a  b ) ( a (b)) ((a) b )               = (a)(b)  NAND
( a  b ) ( a (b)) ((a)(b))               = (a) b    IF
( a  b ) ((a) b ) ((a)(b))               =  a (b)   FI
( a (b)) ((a) b ) ((a)(b))               =  a  b    OR
( a  b ) ( a (b)) ((a) b ) ((a)(b))      =  True
```

Entries in this clumsy construction can be reduced to more succinct forms. The reduction symmetries form the Boolean distributive lattice, a 4-space hypercube with a specific greatest-lower-bound and least-upper-bound.

Using the literal selection model for counting, we have the following picture:

```
      |     |   |     |
      | o x |   | o x |
      -------   -------
         a         b
```

To count all the ways of selecting 0 or 1 objects from each box, with the total number of objects adding to i=0..4, we are summing five (i=0..4) terms, each of which expresses all the ways of choosing i objects. This choice is itself the combinations of ways of choosing k=0..2 objects from box A and k=0..2 objects from box B, with the important restriction that we cannot choose more than i objects total.

We will write the first component of each product as the choice from box A, the second from box B:

```
i=0     chi[0,0]                                    = 1
i=1     chi[1,0] + chi[0,1]                   = 4
i=2     chi[2,0] + chi[1,1] + chi[0,2]        = 6
i=3     chi[2,1] + chi[1,2]                   = 4
i=4     chi[2,2]                                    = 1
```

Symmetry arguments allow this to be condensed. We ignore the distinction between variables.

```
i=0     1*chi[0,0]                        = 1
i=1     2*chi[0,1]                        = 4
i=2     2*chi[0,2] + chi[1,1]             = 6
i=3     2*chi[1,2]                        = 4
i=4     1*chi[2,2]                        = 1
```

Note that the two occurrences of chi[0,2] are the diagonal terms, otherwise the two cases of the variable would undermine the unique Boolean function (XOR and its complement IFF).

Counting 3-DNF without extra variables

Consider 3 variable forms drawn from three different variables. The basis cube is now three dimensional, having (2^3) members

```
          (c)       (a)                 a

      (b)       ( a  b  c )       ((a) b  c )

       b        ( a (b) c )       ((a)(b) c )


           c        (a)                 a

      (b)       ( a  b (c))       ((a) b (c))

       b        ( a (b)(c))       ((a)(b)(c))
```

We have 256 combinations of these eight unit cubes,
(2^2^3) = (1 + 8 + 28 + 56 + 70 + 56 + 28 + 8 +1)

These cluster into 14 discrete and identifiable symmetry categories.

Literal selection creates the following picture:

```
         forms    choices

i=0        1       1*chi[0,0,0]
i=1        8       8*chi[1,0,0]
i=2       28       8*chi[2,0,0] + 12*chi[1,1,0] + ( 8 cross)
i=3      56       12*chi[2,1,0] +  8*chi[1,1,1] + (36 cross)
i=4       70       3*chi[2,2,0] + 12*chi[2,1,1] + (55 cross)
i=5       56       6*chi[2,2,1]                 + (50 cross)
i=6       28       1*chi[2,2,2]                 + (27 cross)
i=7        8                                      ( 8 cross)
i=8        1                                      ( 1 cross)
sum      256                                        185 cross
```

The cross terms are obviously the source of complexity. We can use the projection argument to note that all but one of them are satisfiable.

Counting N-DNF without extra variables

In general, using this construction procedure for n variables, there are (2^n) basis units and (2^2^n) functional forms, only two of which are not satisfiable. In counting notation:

    2^2^n = summation[i={0..2^n}, choose[(2^n), i]]

Now we ask about TAUT, CONTRA, and SAT. Of the constructions, only one, that of no forms is False, and only one, that of all forms, is True. This must be correct since we are counting functionally invariant forms. Thus the SAT question is trivial, we just count the number of distinct conjunctive units.

Note that we do not need to go beyond 3-DNF to address 3-SAT. The above equation with n=3 represents a particular subset of 3-SAT forms, those with only three unique variables. The next level of complexity occurs in N-DNF forms contain more than N variables. Let's step back to the degenerate cases.

1-DNF with two available variables

A counting of satisfiable forms needs to exclude the unique case of choose[m, 0] (the CONTRADICTION case) and any case which includes both the positive and negative case of the variable.

To count satisfiable expressions (and to count TAUT+CONTRA, since SAT = Universe - TAUT+CONTRA, with CONTRA = 1 always), we will use the range of counting methods.

Consider 1-DNF with two available variables:

    a (a) b (b)

The observation which organizes our approach is that two variables form a simple Boolean 2-space, and the valid 1-space forms are simply a projection from 2-space onto 1-space. That is, literals are 2-space edges and values are 1-space coverings by all combinations of edges. Should the 1-space be completely covered, it is a tautology. Should no elements exist, it is a contradiction. Finally, should any uncovered vertices exist in the combination of 2-space edges, then the expression is satisfiable.

The type of space formed by extra variables is called simple because it excludes all cross-product terms. For instance, the 2-space of Boolean functions includes the cross-product (a b) ((a)(b)), which can be visualized as diagonal elements, and is called the IFF function. The functional complement (and the opposite diagonal) is XOR. Conveniently, these are exactly the terms which are difficult (computationally expensive) to identify in the minimization of Boolean functions. Thus our approach automatically excludes satisfiable but pathological forms. To say this differently, *all structurally difficult forms for minimization are satisfiable*. Structurally difficult tautologies do not exist because they must contain extreme symmetries in order to cover the Boolean space they are in.

From 2-DNF, we see that the literals can be identified as unique combinations of each other. In particular

```
(a)   =    ( a  b ) ( a (b)) =    proj[ b in  a  ]
(b)   =    ( a  b ) ((a) b ) =    proj[ a in  b  ]
 b    =    ( a (b)) ((a)(b)) =    proj[ a in (b) ]
 a    =    ((a) b ) ((a)(b)) =    proj[ b in (a) ]
```

From the perspective of the Boolean hypercube in 2-space, each is simply a projection of the other variable along the other literal of the resultant variable. Each variable is an edge in a 2D cube with vertices formed by discrete conjunctive units. This is also a spatial analogy to a DeMorgan transform: the resultant is the negation of the projection of the other variable on the other literal form of the resultant.

We now form all of the disjunctive combinations of the four forms. This is equivalent to looking at all possible coverings of a Boolean plane by combinations of edges. This yields 16 forms, analogous to the two variable functional forms, but they are not functionally distinct. Ten of the original 16 Boolean functions survive, while all that do not survive, degrade to TAUT. Of these ten, eight are the satisfiable terms. These represent the overlaps of the projections from 2-space into the 1-space of values, which do not entirely cover the space

```
<void>                  =  False
 a
(a)
 b
(b)
 a (a)                  =  True
 a  b                               OR
 a (b)                              FI
(a) b                               IF
(a)(b)                              NAND
 b (b)                  =  True
 a (a)  b               =  True
 a (a) (b)              =  True
 a  b  (b)              =  True
(a) b  (b)              =  True
 a (a)  b  (b)          =  True
```

That is, the resultant forms are reductions of the same disjunctive composition operation, but the units are *adjacent pairs* of the original discrete unit cubes, instead of single cubes. The construction is identical, but the "units" are edges rather than points. In Boolean 2-space, when we combine two particular opposite edges, we cover the space. Likewise, any selection of three edges covers the space.

Counting the covering (tautology) cases,

number of: planes	forms	coverings=tautologies	
0	1	CONTRA	
1	4	0	
2	6	2	opposite pair
3	4	4	all
4	1	TAUT	
sum	16	7	

Since TAUT+CONTRA = 8, there are 8 satisfiable expressions.

Now consider the Literal Selection method. We can select either 0 or 1 from each variable box, but not 2, since this generates a tautology. In all cases, this simply excludes ch2[2].

```
|     |   |     |
| o x |   | o x |
-------   -------
   A         B
```

i=0	1*chi[0,0]	= 1	CONTRA
i=1	2*chi[0,1]	= 4	
i=2	2*chi[0,2] + chi[1,1]	= 6	
i=3	2*chi[1,2]	= 4	
i=4	1*chi[2,2]	= 1	TAUT

Eliminating the ch2[2] terms and CONTRA, we get the satisfiable terms:

i=0		= 0
i=1	2*chi[0,1]	= 4
i=2	chi[1,1]	= 4
i=3		= 0
i=4		= 0
	satisfiable sum	= 8

1-DNF with three available variables

Projective method: We have a 3-space of 256 possible disjunctive forms, and 2^3=8 conjunctive units. Variables are the 2-space planes:

```
 a  = ((a) b  c ) ((a) b (c)) ((a)(b) c ) ((a)(b)(c))
(a) = ( a  b  c ) ( a  b (c)) ( a (b) c ) ( a (b)(c))
 b  = ( a (b) c ) ( a (b)(c)) ((a)(b) c ) ((a)(b)(c))
(b) = ( a  b  c ) ( a  b (c)) ((a) b  c ) ((a) b (c))
 c  = ( a  b (c)) ( a (b)(c)) ((a) b (c)) ((a)(b)(c))
(c) = ( a  b  c ) ( a (b) c ) ((a) b  c ) ((a)(b) c )
```

Without exhaustive listing it is easy to see the distribution of tautologies:

number of: planes	forms	coverings=tautologies+CONTRA	
0	1	1	CONTRA
1	8	0	
2	28	3	opposite pair
3	56	48	openings
4	70	70	all
5	56	56	all
6	28	28	all
7	8	8	all
8	1	1	TAUT
sum	256	215	

There are three pairs of opposite planes when taken two at a time, and eight ways to leave an uncovered vertex using three planes. All other cases cover. This gives 8+25+8 = 41 satisfiable forms.

a b c (a) (b) (c) 6

Using Literal Selection:

```
i=0   1*chi[0,0,0]
i=1   8*chi[1,0,0]
i=2  8*chi[2,0,0] + 12*chi[1,1,0]
i=3  chi[2,1,0] + chi[1,1,1]
i=4  chi[2,2,0] + chi[2,1,1]
i=5  chi[2,2,1]
i=6  chi[2,2,2]
i=7
i=8
```

To provide an example of chi counting: chi[1,1,0] means we take one object from each of two variable boxes. Recall the ch2[1] contributes two cases, we can select one object or the other. with two 1s, we have four selections. We have abstracted across variables, so we count the ways to take something from two of three sources. This is choose[3,2] = 3. In total then there are 4*3=12 count objects for chi[1,1,0].

1-DNF with n available variables

In general, for 1-DNF with n variables, there is 1 CONTRA and

(2^n + (choose[2^n, 2] - n) + 2^n)

non-tautologies.

Simplifying: (2^(n+1) - n)

All are easy to identify structurally by counting the N-1 projection objects in the arbitrary 1-DNF form and looking for opposing N-1 objects to determine non-TAUT cases.

The exploration prematurely ends here, prior to developing 2-DNF forms with n variables. The same principles hold.