A QUESTION WITHIN A QUESTION

William Bricken

March 1998

The thousands of mappings of mathematical structures onto 3SAT provide a particularly rich structural variety of forms. All of these forms can be cross-mapped in polynomial time. That is, all can be expressed as decision questions, and in particular, as expressions in the shallow canonical form of CNF (Conjunctive Normal Form). 3SAT is a subset of CNF, both representations consist of a conjunction of clauses.

Most CNF tautological forms are very large. Due to the shallowness of the representation the requisite cross-connectivity forces an exponentially growing number of variable references for full propositional expressability. It is this dense cross-connectivity that makes identification of CNF tautologies difficult. The line of difficulty is of course between polynomial and non-polynomial algorithms for TAUT.

The most deeply nested CNF form is INF (Implicate Normal Form), it is commonly represented by nested if-then-else forms. BDDs are a type of deeply nested representation, but they lose contact with the semantics of CNF, and consequently are difficult to transform. That multiplication is exponential in BDDs suggests that there may be a better way.

I propose to begin with a leap of faith, I wish to believe that P=NP. Therefore there is a polynomial algorithm for TAUT. I will assume that BM is a good place to look cause of its minimality and algorithmic ease.

Lets continue, I will believe that the INSERT algorithm is polynomial. The purpose of this belief is to define an investigative posture. When it fails, there is an engineering problem to identify. In fact the engineering itself is in place, the challenge is to ask the right question of the tool.

Leaping-into-faith makes proofs short for this reason: polynomial algorithms can be expressed as well-founded recursions. Since the primary BM tool of void-substitution is well-founded recursively, the INSERT algorithm has a well defined halting condition, the failure to erase something during a cycle.

This attitude also provides a meta-strategy: when in doubt, recur. The Question-Within-A-Question strategy.

TAUT consists of illusions. Tangles of variables with no independent meaning. Since each variable must show itself in each part of the tangle in which it participates, it seems reasonable to assume that following the ends to untangle shouldn't be abnormally (i.e. NP) difficult.

[Note that the positive approach to having tangled knowledge and meta-knowledge is built on the leap of P.]

Of course, the less ends the better, the deeper the tangle the more symmetrical it will be.

A Friendly Example

Let's select a very friendly representation of P=?=NP. A symmetry of the three-coloring-a-tetrahedron, 3TET, form of TAUT has nice properties. Here it is in a simple example with 34 clauses:

```
    (B1 B2) (B1 B3) (B1 B4) (B1 G1) (B1 R1) (B2 B3)
    (B2 B4) (B2 G2) (B2 R2) (B3 B4) (B3 G3) (B3 R3)
    (B4 G4) (B4 R4) (G1 G2) (G1 G3) (G1 G4) (G1 R1)
    (G2 G3) (G2 G4) (G2 R2) (G3 G4) (G3 R3) (G4 R4)
    (R1 R2) (R1 R3) (R1 R4) (R2 R3) (R2 R4) (R3 R4)
    ((B1)(G1)(R1)) ((B2)(G2)(R2)) ((B3)(G3)(R3)) ((B4)(G4)(R4))
```

What is nice is that there are a very few negated literals.

The INSERT algorithm has predictable properties when dealing with CNF forms. The reduction action is almost always associated with the negated literals. The lack of negated literals in the predominance of other clauses keeps them from interacting and thus keeps reduction well constrained

Now the engineering question: what kind of tangle is holding up this form?

We know that the eventual collapse will be caused by one of the boundary forms (disjunctive clauses) becoming a mark, ( ), the mark of TAUT.

Simple set-insertion yields the following reduction:

Forms of the symmetry (<letter.number> <different-letter.same-number>) vanish. An example:

```
  (G4 R4
     ^(B1 B2) (B1 B3) (B1 B4) (B1 G1) (B1 R1) (B2 B3)
      (B2 B4) (B2 G2) (B2 R2) (B3 B4) (B3 G3) (B3 R3)
      (B4 G4) (B4 R4) (G1 G2) (G1 G3) (G1 G4) (G1 R1)
      (G2 G3) (G2 G4) (G2 R2) (G3 G4) (G3 R3)
      (R1 R2) (R1 R3) (R1 R4) (R2 R3) (R2 R4) (R3 R4)
      ((B1)(G1)(R1)) ((B2)(G2)(R2)) ((B3)(G3)(R3)) ((B4)(G4)(R4))^)

  (G4 R4
     ^(B1 B2) (B1 B3) (B1 B4) (B1 G1) (B1 R1) (B2 B3)
      (B2 B4) (B2 G2) (B2 R2) (B3 B4) (B3 G3) (B3 R3)
      (B4   ) (B4   ) (G1 G2) (G1 G3) (G1   ) (G1 R1)
      (G2 G3) (G2   ) (G2 R2) (G3   ) (G3 R3)
      (R1 R2) (R1 R3) (R1   ) (R2 R3) (R2   ) (R3   )
      ((B1)(G1)(R1)) ((B2)(G2)(R2)) ((B3)(G3)(R3)) ((B4)(  )(  ))^)
```

```
(G4 R4
   ^(B1 B2) (B1 B3) (B1 B4) (B1 G1) (B1 R1) (B2 B3)
    (B2 B4) (B2 G2) (B2 R2) (B3 B4) (B3 G3) (B3 R3)
    (B4   )         (G1 G2) (G1 G3) (G1   ) (G1 R1)
    (G2 G3) (G2   ) (G2 R2) (G3   ) (G3 R3)
    (R1 R2) (R1 R3) (R1   ) (R2 R3) (R2   ) (R3   )
    ((B1)        ) ((B2)        ) ((B3)        )              ^)

(G4 R4
   ^(     ) (     ) (   B4) (   G1) (   R1) (     )
    (   B4) (   G2) (   R2) (   B4) (   G3) (   R3)
    (B4   )         (G1 G2) (G1 G3) (G1   ) (G1 R1)
    (G2 G3) (G2   ) (G2 R2) (G3   ) (G3 R3)
    (R1 R2) (R1 R3) (R1   ) (R2 R3) (R2   ) (R3   )
    B1 B2 B3^)

( ^(  )^ )  ==>  <void>
```

For our example, this leaves the following form, which is also TAUT but simpler:

```
(B1 B2) (B1 B3) (B1 B4)                 (B2 B3)
(B2 B4)                 (B3 B4)
                (G1 G2) (G1 G3) (G1 G4)
(G2 G3) (G2 G4)         (G3 G4)
(R1 R2) (R1 R3) (R1 R4) (R2 R3) (R2 R4) (R3 R4)
((B1)(G1)(R1)) ((B2)(G2)(R2)) ((B3)(G3)(R3)) ((B4)(G4)(R4))


(B1 B2) (B1 B3) (B1 B4) (B2 B3) (B2 B4) (B3 B4)
(G1 G2) (G1 G3) (G1 G4) (G2 G3) (G2 G4) (G3 G4)
(R1 R2) (R1 R3) (R1 R4) (R2 R3) (R2 R4) (R3 R4)
((B1)(G1)(R1)) ((B2)(G2)(R2)) ((B3)(G3)(R3)) ((B4)(G4)(R4))
```

[Incidentally, this may map onto a different NP-complete problem.]

Now it is time to recur. We know that the first pass eliminated all other reductions of positive clauses. We know that the action is at the next level. Here is the first (of 12) negated literals.

```
((B1
 ^(B1 B2) (B1 B3) (B1 B4) (B2 B3) (B2 B4) (B3 B4)
  (G1 G2) (G1 G3) (G1 G4) (G2 G3) (G2 G4) (G3 G4)
  (R1 R2) (R1 R3) (R1 R4) (R2 R3) (R2 R4) (R3 R4)
   ((B2)(G2)(R2)) ((B3)(G3)(R3)) ((B4)(G4)(R4))^ )(G1)(R1))
```

```
((B1
 ^(   B2) (   B3) (   B4) (B2 B3) (B2 B4) (B3 B4)
  (G1 G2) (G1 G3) (G1 G4) (G2 G3) (G2 G4) (G3 G4)
  (R1 R2) (R1 R3) (R1 R4) (R2 R3) (R2 R4) (R3 R4)
   ((B2)(G2)(R2)) ((B3)(G3)(R3)) ((B4)(G4)(R4))^ )(G1)(R1))

((B1
 ^(   B2) (   B3) (   B4) (B2 B3) (B2 B4) (B3 B4)
  (G1 G2) (G1 G3) (G1 G4) (G2 G3) (G2 G4) (G3 G4)
  (R1 R2) (R1 R3) (R1 R4) (R2 R3) (R2 R4) (R3 R4)
   (    (G2)(R2)) (    (G3)(R3)) (    (G4)(R4))^ )(G1)(R1))

((B1
 ^(   B2) (   B3) (   B4)
  (G1 G2) (G1 G3) (G1 G4) (G2 G3) (G2 G4) (G3 G4)
  (R1 R2) (R1 R3) (R1 R4) (R2 R3) (R2 R4) (R3 R4)
   (    (G2)(R2)) (    (G3)(R3)) (    (G4)(R4))^ )(G1)(R1))
```

At this point it looks as though the form will not further reduce. The objective is to get the virtual form to go to mark, so that this subexpression will vanish. If all the negated variable insertions vanish, then we succeed.

Name the virtual insertion the **put-form** and the form into which the virtual form is put, the **into-form**.

Algorithm

The dynamic is

1. Expose a new, possibly empty, set-of atoms ATS by inserting the put-form into the first level of the into-form. If no atoms, descend by inserting the appropriately reduced put-form into each bound in the current context. That is, descend the parens-tree breadth first with optional parallelism.

At init, the put form is all other forms in the context of the into-form. Each positive-clause requires exactly one put-form, since it has depth =1.

2. Erase the current ATS from the put-form. If there are changes, submit the new put-form to the algorithm.

```
      If the resulting put-form is a ground,
            empty = return (nil) =  <void>
            mark  = return ( )   =  nil
      else recur on inserting the current put-form
            into the next level of the local into-form.
```

3. When the put-form reaches bottom, it is in only one state, that of an indeterminate collection of variables and boundaries.

      If it were <void>, it would not reach bottom
      If it were mark, it would obliterate bottom before reaching it

If the current and last extraction does not produce a mark, then the put-form is abandoned. (Some advanced techniques use the scraps)

4. The entire assemblage of top-level bound Insertions (with useless virtual forms erased), and their recursive descent down each bound, is collected and compared to the initial into-form. If the new into-form is smaller, recur on it. Otherwise return into-form.

Is the virtual insertion reduced? That is, is

```
^(   B2) (   B3) (   B4)
     (G1 G2) (G1 G3) (G1 G4) (G2 G3) (G2 G4) (G3 G4)
     (R1 R2) (R1 R3) (R1 R4) (R2 R3) (R2 R4) (R3 R4)
     (    (G2)(R2)) (    (G3)(R3)) (    (G4)(R4))^
```

in TAUT? If so, certainly the B variables are irrelevant, which seems reasonable given that the selected literal for insertion was B. We have again recurred, again with a simpler question. This time, however, the question: is

```
      (G1 G2) (G1 G3) (G1 G4) (G2 G3) (G2 G4) (G3 G4)
      (R1 R2) (R1 R3) (R1 R4) (R2 R3) (R2 R4) (R3 R4)
      ((G2)(R2)) ((G3)(R3)) ((G4)(R4))
```

in TAUT is know, since the form is 2SAT.

The Central Idea

CNF forms are maximally tangled general forms.

Non-polarized CNF forms are hypercube rotations of the polarized case, therefore assume a polarized CNF form (all clauses are completely positive or negative).

Divide the N clauses into X positive and Y negative: N = X + Y

Positive case:

Each insertion of (N-1) context clauses into a positive clause asks a question.

```
(a b ... ^(a c)...^)
```

The context of the positive-clause is disjunctive. For the form to be in TAUT, one positive-question must be ( ). But there is no way for an inserted form to convert a positive-clause to ( ). Should the virtual form vanish, the positive variable will remain. Should the virtual form become ( ), it will erase the clause, not make it ( ). Finally, should the virtual form not reduce, then this clause does not simplify.

We carry out the insertions on the positive-clauses to see if any are redundant. If any vanish, we have a simpler TAUT form to insert into the negative-clauses.

Negative case:

Each insertion of at most (N-1) context clauses into a negative clause asks a sub-question.

```
((a ^(a c)...^)
 (b ^(a c)...^)
 ...
)
```

For the negative clause to become ( ), every negated literal must vanish. That is,

```
((a ^...( )...^)
 (b ^...( )...^)
 ...
)
```

Should any one of these not vanish, the form will not collapse.

DISCUSSION

We have answered a question within a question, by seeing if the virtual form itself was tautological. Rather than simply removing context from the into-form, we must ask if it itself is a tautology, fortunately a smaller tautology than the one in the outer context. This technique is "virtual proof", since it take a proof excursion in a virtual form rather than the form in question. We can feel better knowing that the virtual form itself is a fragment of original.

The technique is proof factoring. We view the original form as a proof tree, the output of a series of constructive tanglings beginning with ( ).

Insertion of the (N-1) clause context into each of N clauses asks N virtual questions.

In the case of 3COLOR, we have two types of clauses. Given K nodes to color, there are these clauses:

ternary-negative: K

binary-positive: 3*choose[K 2] + K*choose[3 2]

and a total of 3*K variables (i.e. edge ends on the graph)

3* comes from 3 colors, permutations of same color-letter over nodes

K* comes from K nodes, permutations of different node-numbers over colors

Total clauses in K-node 3COLOR:

3*choose[K 2] + K*choose[3 2] + K

The K*choose[3 2] = 4*K factor vanishes as redundant on the first INSERT pass.

Total context clauses remaining for negative-clause insertion

3*choose[K 2] + K - 1

We can now partition the insertion into 3*K subproblems, each of which is also in TAUT. We stopped here at the first of 12 negative-literals in the negative-clauses.

```
(B1
 ^(   B2) (   B3) (   B4)
  (G1 G2) (G1 G3) (G1 G4) (G2 G3) (G2 G4) (G3 G4)
  (R1 R2) (R1 R3) (R1 R4) (R2 R3) (R2 R4) (R3 R4)
  (    (G2)(R2)) (    (G3)(R3)) (    (G4)(R4))^ )
```

We know that

```
(   B2) (   B3) (   B4)
(G1 G2) (G1 G3) (G1 G4) (G2 G3) (G2 G4) (G3 G4)
(R1 R2) (R1 R3) (R1 R4) (R2 R3) (R2 R4) (R3 R4)
(    (G2)(R2)) (    (G3)(R3)) (    (G4)(R4))
```

must be in TAUT for the original form to be in TAUT.

But this is an easy question , since the remaining form is in 2SAT.

Of the

(3*choose[K 2] + K - 1)

clauses inserted into each negative-literal, there are choose[K 2] which erase, all those associated with the color of the negative-literal.

The remaining (2*choose[K 2] + K - 1) clauses are a 2SAT TAUT.

Counting K-node 3COLOR

clauses: 3*choose[K 2] + K*choose[3 2] + K

pos-clauses: 3*choose[K 2] + K*choose[3 2]

after insertion, each with

(3*choose[K 2] + K*choose[3 2] + K - 1)

interior virtual clauses. These

(3*choose[K 2] + K*choose[3 2])

insert-reductions result in

3*choose[K 2]

remaining pos-clauses.

neg-clauses: K with 3*K neg-literals

after insertion, each with

(3*choose[K 2] + K - 1)

interior virtual clauses

This requires 3*K insert-reductions, resulting in 3*K 2SAT problems, each having

(2*choose[K 2] + K - 1)

2-clauses. Note that each recursive pass reduces the number of nodes by one.

DISCUSSION TWO

Using the insertion proof technique on a virtual form at some point recurs into putting virtual forms into other virtual forms. This is *virtual proof*, using a proof step on an imaginary form to come to a real conclusion. Extending to four-coloring is easy. Seems like the representation and its reduction should generate the coloring as well.

APPENDIX

Here is what the Losp engine reductions give. They do not reduce all the way to TRUE, but applying case-analysis completes the reduction.

Three-color-a-tetrahedron

Original problem in conjunctive form:

```
(((B1) (B2)) ((B1) (B3)) ((B1) (B4)) ((B1) (G1)) ((B1) (R1)) ((B2) (B3))
 ((B2) (B4)) ((B2) (G2)) ((B2) (R2)) ((B3) (B4)) ((B3) (G3)) ((B3) (R3))
 ((B4) (G4)) ((G1) (G2)) ((G1) (G3)) ((G1) (G4)) ((G1) (R1)) ((G2) (G3))
 ((G2) (G4)) ((G2) (R2)) ((G3) (G4)) ((G3) (R3)) ((G4) (R4)) ((R1) (R2))
 ((R1) (R3)) ((R1) (R4)) ((R2) (R3)) ((R2) (R4)) ((R3) (R4)) ((R4) (R4))
 (B1 G1 R1) (B2 G2 R2) (B3 G3 R3) (B4 G4 R4))
```

Losp reduction yields:

```
(R4
 (B4 G4)
 ((R1) (R2))
 (B1 G1 R1)
 (B2 G2 R2)
 (B3 G3 R3)
 ((G2) (G3 R2))
 ((B3) (B4 G3 R3))
 ((G1) (G2 G3 R1))
 ((R3) (G3 R1 R2))
 ((B2) (B3 B4 G2 R2))
 ((G4) (B4 G1 G2 G3))
 ((B1) (B2 B3 B4 G1 R1)))
```

Case-analysis yields:

```
(nil)
```

```
Original problem in disjunctive form:

(((B1 B2) (B1 B3) (B1 B4) (B1 G1) (B1 R1) (B2 B3)
  (B2 B4) (B2 G2) (B2 R2) (B3 B4) (B3 G3) (B3 R3)
  (B4 G4) (B4 R4) (G1 G2) (G1 G3) (G1 G4) (G1 R1)
  (G2 G3) (G2 G4) (G2 R2) (G3 G4) (G3 R3) (G4 R4)
  (R1 R2) (R1 R3) (R1 R4) (R2 R3) (R2 R4) (R3 R4)
  ((B1)(G1)(R1)) ((B2)(G2)(R2)) ((B3)(G3)(R3)) ((B4)(G4)(R4))))


Losp reduction yields:

        (R2 R3)
        ((B1) (G1) (R1))
        ((B2) (G2) (R2))
        ((B3) (G3) (R3))
        ((B4) (G4) (R4))
        (B4 ((G4) (R4)))
        (G3 ((G4) (R3)))
        (R1 ((R2) (R3)))
        (B3 ((B4) (G3) (R3)))
        (G2 ((G3) (G4) (R2)))
        (B2 ((B3) (B4) (G2) (R2)))
        (G1 ((G2) (G3) (G4) (R1)))
        (R4 ((G4) (R1) (R2) (R3)))
        (B1 ((B2) (B3) (B4) (G1) (R1)))


Case-analysis yields:

        (nil)
```