TAUTOLOGY CHECKING

William Bricken
March 1987

The Boyer-Moore Theorem Prover uses a "waterfall" control model. If the problem can be solved by Method1, then Method2, which is more complex, is not called. The induction prover is called only on problems that have inductive support such as a recursive definition and a decreasing metric. Boyer-Moore provides switches for turning off some processes, which is so admirable that I'm copying that control structure. But some engines designed specifically for propositional work are faster than Boyer-Moore. Which creates a particular problem: very sophisticated propositional engines are used in conjunction with Boolean minimization of circuits. These are bit-coded and machine specialized. So we come right down to local resource constraints. Since this is a well researched area (which also lacks comparative benchmarking), the pulling-apart-other-work strategy looks like too much effort. Or maybe I'm resource limited in my ability to do that well.

Naturally, expressive power is more important than speed. Here are some Losp algorithms that do *only* tautology checking and don't try to return minimal contingent expressions. Since the ideas are of general interest (and illustrate awesome boundary computational power), I'll outline them here.

The idea is to generate a contradiction, thus demonstrating that a given expression is not a tautology. If a contradiction cannot be found, then we have a tautology.

Transcribe the logical expression into the Losp expression E. E is an arbitrary expression in parens notation.

```
CLEAN E (which means apply ABSORB and CLARIFY deeply) to get E'
  If E' is ground, that is E' = ( ) or E' = (( )), we're done.
  Otherwise, E' has no ground tokens and no double parens.

Examine the possible cases of E':
  E' is a literal, i.e. an atom a or a bounded atom, (a):
    Substitute a ground value to generate a contradiction

  E' is an unbounded collection, A ... :
    Recur on each subform in the collection.  If any one of them fails to
    disappear then we cannot avoid a contradiction.

  E' is a bounded collection, (A ...):
    Substitute a ground value for all literals at the top level -- and all
    their replicates throughout E' -- to make them disappear;
    CLEAN E' and recur on the new smaller E''
      If E' has no top-level literals, recur on each bounded subform;
        all must be void to avoid a contradiction.
```