TITLE : **A Prolog Implementation of an Analytic Tableau Theorem Prover for the Propositional Calculus**

REFERENCE : 89024

DATE : 3 May 1989

AUTHOR : André Vellino

ABSTRACT

This paper describes an improvement on the analytic tableau method which simulates SL-resolution. It is implemented by a short and elegant Prolog program which is relatively efficient.

Computing Research Laboratory, Bell-Northern Research
P.O. Box 3511, Station C, Ottawa Canada K1Y 4H7
Telephone 613-763-3841, Fax 613-763-4222

1. INTRODUCTION

A proof method for a system of logic is more powerful than another to the degree that it simplifies the task of producing derivations for theorems. One possible measure of the relative complexity of proof systems for the propositional calculus is offered by the notion *polynomial simulation* [Cook 1971]. Intuitively speaking, if any proof α of a tautology T in proof system A can be transformed into a proof β of T in system B is such that the length of β is a polynomial function of the length of α then system B p-simulates system A. Conversely, one can show that proof system A *cannot* p-simulate system B by showing that there are tautologies that are hard-to-prove for A are but not hard-to-prove for B, i.e. B is strictly more powerful than A.The *polynomial simulation* relation imposes a partial ordering among proof systems, and, although many simulation relations are known [Cook and Reckhow 1979], there are still a number of gaps in the literature. It has recently been shown [Vellino 1989] that tree resolution simulates a varient of the analytic tableau tableau procedure [Smullyan 1968] and that both SL-resolution [Kowalski and Kuehner 1971] and the connection method [Bibel 1982] are equivalent to restrictions of this improved tableau method.

In this note we describe the improved analytic tableau method and present a short and elegant Prolog program that implements it efficiently. Applied to examples that are hard for resolution such as the Pigeon Hole problem [Haken 1985, Urquhart 1987], experimental results show that this program is superior to implementations of resolution methods.

2. ANALYTIC TABLEAUX METHODS

An *analytic tableau* q for a set of propositional formulae (in Disjunctive Normal Form) Σ, is a tree such that all the nodes in q other than the root node are labelled by literals occurring in Σ and for each interior node k in the tree, the set of literals labelling the children of k is a formula in Σ. The root node of any analytic tableau for formulae will be designated by the special symbol ϑ.

A branch in a tableau is *closed* if it contains both a literal and its complement. An analytic tableau is *closed* if all its branches close but *open* if at least one branch is not closed and all the formulae in Σ have been decomposed in that branch.

It is easy to see that if Σ is a consistent set of formulae, an assignment of truth values to the literals which makes Σ true can be read off an open branch of its analytic tableau. Conversely, all the branches of an analytic tableau for a set Σ of formulae are closed if and only if Σ is inconsistent.

2.1 CLASH RESTRICTED ANALYTIC TABLEAU

The search for the smallest tableaux refutation (closed tableau), particularly for sets of non-minimally inconsistent formulae (a set of formulae all of which are necessary to prove inconsistency), can be quite inefficient if the tableau method has no built-in heuristic. One restriction of the general tableaux method that can reduce the search space is to impose the rule that each formula decomposition closes a branch in the tableau. This is equivalent to the condition that the formula decomposed at each node *k* in the tableau contain at least one literal that clashes with (i.e. is the complement of) a literal labelling *k* or an ancestor of *k*. We will call such a tableau an *ancestor clash restricted* (*ACR*) analytic tableau.

A more stringent clash restriction is to specify that some literal in every decomposed formula clash with the literal labelling the *parent* node, i.e. that every non-leaf node *k* in the tableau is labelled with a literal that clashes with a literal labelling a child of *k*. We say that such a tableau is *parent clash restricted* (*PCR*). Both *ACR* and *PCR* analytic tableaux methods are complete: a set Σ of formulae is inconsistent if and only if there exists both a *ACR* and a *PCR* analytic tableau for Σ.

Contrast the following analytic tableaux for the refutation of the set {*ab*, ~*ab*, ~*b*, ~*ac*, *ef* }.

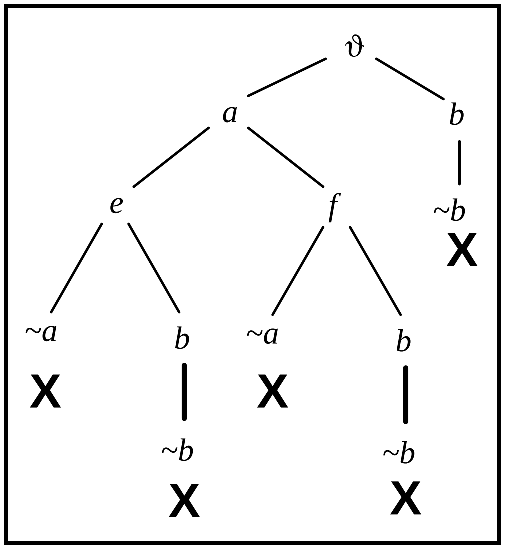

Figure 2.1.i
Tableau with no restrictions for {*ab*, *~ab*, *~b*, *~ac*, *ef* }

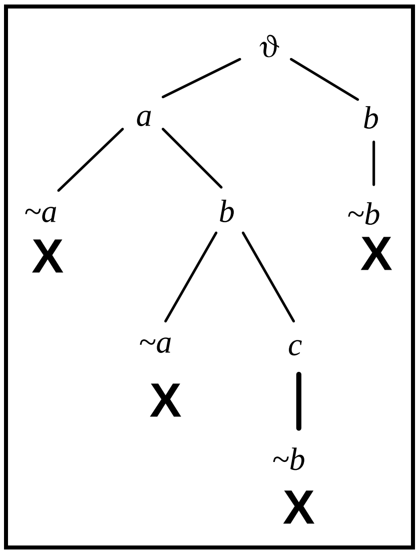

Figure 2.1.ii
Tableau with ancestor clash restriction for {*ab*, *~ab*, *~b*, *~ac*, *ef* }

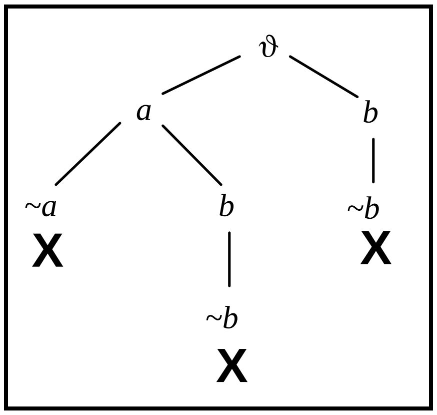

Figure 2.1.iii

Tableau with parent clash restriction for{*ab*, *~ab*, *~b*, *~ac*, *ef* }

These show that *PCR* and *ACR* tableaux can be shorter than unrestricted analytic tableaux. Thus, as a heuristic for searching, this restriction appears to minimize the decomposition of irrelevant formulae.

Whether or not *PCR* or *ACR* tableaux polynomially-simulate unrestricted analytic tableaux is an open question. However, we know that some minimal unrestricted tableau are smaller than either *PCR* or *ACR* tableaux. The examples that show this are constructed as follows.

Consider a set S of $2(2^n - 1)$ distinct positive literals from which the set C of $2^n - 1$ formulae containing exactly two distinct literals each, such that each literal in S occurs only once in C. Then construct an open analytic tableau containing only one decomposition of each formula in C. This tableau can be closed by the set of 2^n formulae each containing *n* negative occurrences of literals in S. The resulting tableau T_n has $(n + 2)2^n - 2$ nodes.

For instance, the tableau on figure 2.1.iv are minimal and all the *PCR* or *ACR* tableaux for those sets of formulae are larger.

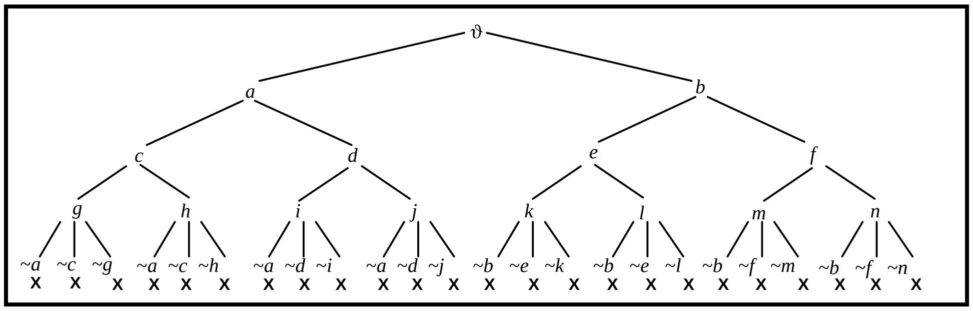
ϑ
a
b
c
d
e
f
g
h
i
j
k
l
m
n
~a ~c ~g ~a ~c ~h ~a ~d ~i ~a ~d ~j ~b ~e ~k ~b ~e ~l ~b ~f ~m ~b ~f ~n
x x x x x x x x x x x x x x x x x x x x x x x x

Figure 2.1.iv

Minimal tableau for {ab, cd, ef, gh, ij, kl, mn, ~a~c~g, ~a~c~h, ~a~d~i, ~a~d~j, ~b~e~k, ~b~e~l, ~b~f~m, ~b~f~n}

Since each formula in these minimally inconsistent sets is decomposed exactly once, by construction, the class of tableau T_n is minimal for *n*   3. If either the *PCR* or *ACR* restriction is placed on the construction of such a tableau then the same formula must be decomposed more than once because the symmetry of the literal clashes in the tree is broken. This is illustrated by observing the start of an *ACR* tableau for T_3.

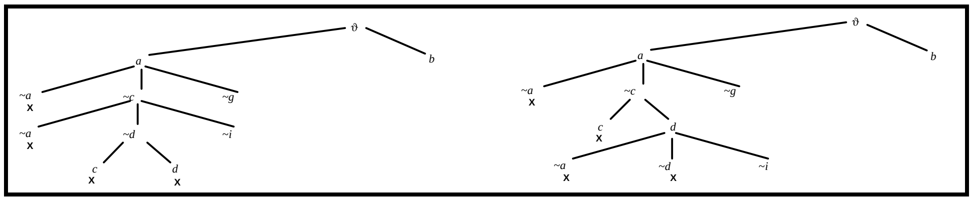

Figure 2.1.v

Partial *ACR* tableaux for T_3

Given the start formula *ab*, the next formula beneath *a* must be one of the four formulae containing *~a*, say *~a~c~g*. The branch containing *~c* can be closed by *cd* in a parent clash decomposition (right hand side of figure 2.1.v) or after some ancestor clash (left hand side of the figure). In either case there are still two more formulae containing *~c* and *~d* which have yet to be decomposed (since the set is minimally inconsistent) and whose branches must be closed by another decomposition of *cd*.

2.2 IMPROVED ANALYTIC TABLEAU

There is a simple extension to the analytic tableau method that increases its efficiency as a method for proving theorems and produces a considerable improvement in the complexity of its proofs. A short and elegant Prolog program that implements this method is described in section 3.

We will say that an *improved* (*I*) analytic tableau for a set of formulae Σ if it is *completed* or *checked* which we define simultaneously by induction as follows:

(i) A subtableau is completed if it is closed.

(ii) If a branch of a subtableau ends in a literal *l* and there is an ancestor of this node that has a child also labelled with *l* which is at the top of a completed subtableau then the branch ending in *l* is checked.

(iii) A subtableau is completed if all its branches are closed or completed.

For example, a completed *I*-analytic tableau for the formulae { *ab*, *~ab*, *~b* } is given in figure 2.2.i. Compare this with the tableau in figure 2.1.ii.

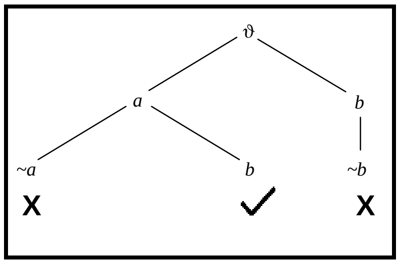

Figure 2.2.i
Completed *I*-analytic tableau for {*ab*, *~ab*, *~b*, *~ac*, *ef* }

To show the soundness of this method it is sufficient to observe that any completed *I*-analytic tableau can be transformed into a closed tableau by replacing every check mark by a closed sub-tableau containing no check marks (check-marks cannot justify each other cyclically). Without loss of generality, we will assume that *I*-analytic tableau are constructed so that the check marks occur to the left of the nodes by which they are justified. This is always possible since the literals in each formula are not order sensitive.

Notice that checking a branch that ends in a literal *l* simply has the effect of reporting (or delaying) the justification for closing that branch to the decomposition of formulae on another branch ending in *l*, provided that both occurrences of the literal have the same ancestor. The checking of a literal always reports its decomposition to a literal belonging to an ancestor formula and effectively merges nodes in the tableau, allowing them to share the closure of a sub-tableau.

3. PROLOG IMPLEMENTATION

The following Prolog program is an short and elegant implementation of the improved analytic tableau method.

We represent a conjunction of formulae in DNF as a list of lists: each disjunction of a formula is a list of literals and the conjunction of formulae as a list of disjunctions. For each disjunctive formula an attempt is made to provide a satisfying assignment to at least one propositional variable in that formula. If we represent each propositional variable by a (first-order) logic variable in Prolog, this truth-value assignment is propagated to all the occurrences of this propositional variable in every formula simply by unification.

A formula is disjunction (list) of literals of the form

```
              literal(NegOrPos,Variable)
```

where NegOrPos indicates the presence or absence of a negation sign in front of the Variable. For example the formula ((a v ~b) & (b v ~c) is represented as `[[literal(true,A),literal(false,B)],[literal(true,B),literal(false,C)]]`. A formula is satisfied if a variable can be unified with its prefix (`true` or `false`).

The plain analytic tableau method is a simple program. The predicate definition for `satisfiable/1` says that a conjunction (list) of formulae is satisfiable if each formula can be satisfied and the predicate definition for `satisfied/1` satisfies a formula by assigning or verifying a satisfying truth value assignment for some literal.

```
satisfiable([]).
satisfiable([F|Formulae]) :- satisfied(F), satisfiable(Formulae).

satisfied([literal(T,T)|_]).         % the first disjunct is satisfied
satisfied([_|R]) :- satisfied(R).    % or one of the rest are
```

(This program, incidentally, is equivalent in complexity to L. Pereira's theorem prover in "Prolog by Example" by Coelho and Cotta (1988).)

There are two major problems with this program:

(1) it will explore unnecessary truth-value assignments for clauses that are already true in virtue of a previous truth-value assignment to another literal (i.e. it will decompose clauses unnecessarily).

(2) it does not take advantage of the fact that once it has shown that a truth-value assignment for a literal in a clause cannot be used to satisfy the other clauses then only its negation can (the *checking* rule described above).

The beauty of the improved tableau method is that *both* these optimizations are implemented immediately by the following modifications:

```
satisfied([literal(T,T)|_]).
satisfied([literal(TV,V) | Literals]):-
                              negation(TV,V),satisfied(Literals).
negation(true,false).
negation(false,true).
```

If the attempt to satisfy a disjunctive formula (the first clause for `satisfied/1`) fails then the literal is falsified (the truth-value reversed) and another literal in that formula must be satisfied. Thus, if the second clause to `satisfied/1` is called with a variable already bound to a satisfying assignment, then looking at the others is unnecessary, which is guaranteed by the failure of the guard `negation/2` (optimization 1). And if `satisfied/1` was originally called with an unbound variable and backtracking causes the second clause for `satisfied/1` to be called, then the literal can have its truth-value reversed by the call to `negation/2` (optimization 2).

Prolog systems with indexing may benefit further by folding in `negation/2` into the head of `satisfied/1`,

```
satisfied([literal(T,T)|_]).
satisfied([literal(true,false) | Literals]):- satisfied(Literals).
satisfied([literal(false,true) | Literals]):- satisfied(Literals).
```

making it a 5-line program.

CONCLUSION

The built-in depth-first search and first order unification mechanisms of Prolog are ideal for searching and propagating satisfying truth-value assignments to literals in formulae. The improved tableau rule (the checking rule) described in section 2.2 provides tableau proofs whose minimal size is of the order of resolution proofs. As with resolution proofs minimal sized tableau proofs can be acheived only by chosing a stragic order in which to decompose the formulae but these strategies are not discussed here.

BIBLIOGRAPHY

Bibel, W. (1982). *Automated Theorem Proving*. Vieweg Verlag, Braunschweig; Wiesbaden: Vieweg.

Cook, S. A. (1971). "The' Complexity of Theorem Proving Procedures" *Proc. 3rd ACM STOC* pp.151-158.

Cook, S. A., and Reckhow, R. A. (1974). "On the Length of Proofs in the Propositional Calculus" *Proc. 6th ACM STOC*, pp.135-148.

Cook, S. A., and Reckhow, R. A. (1979). "The Relative Efficiency of Propositional Proof Systems" *Journal of Symbolic Logic*, **44**, pp.36-50.

Haken, A. (1985). "The Intractability of Resolution" *Theoretical Computer Science* **39** pp.297-308.

Kowalski, R., and Kuehner, D. (1971). "Linear Resolution with Selection Function" *Artificial Intelligence* **2** p. 227-260, Reprinted in Siekmann and Wrightson (1983) Vol. 2.

Smullyan, R., (1968). First Order Logic Springer-Verlag, New York.

Urquhart, A. I. F. (1987). "Hard Examples for Resolution" *J. ACM*. **34** No.1, pp. 209-219.

Vellino, A., (1989). "*The Complexity of Automated Reasoning*" draft of Ph.D. thesis, Department of Philosophy, University of Toronto.

PROLOG PROGRAM

```
satisfiable( [] ).
satisfiable([F|Formulae]) :- satisfied(F), satisfiable(Formulae).


satisfied([literal(T,T)|_]).
satisfied([literal(TV,V) | Literals]):-
            negation(TV,V),satisfied(Literals).


negation(true,false).
negation(false,true).
```

```
%%%    5 objects can't fit into 4 holes !

formulae( [
  [literal(true,_a),literal(true,_b),literal(true,_c),literal(true,_d)],
  [literal(true,_e),literal(true,_f),literal(true,_g),literal(true,_h)],
  [literal(true,_i),literal(true,_j),literal(true,_k),literal(true,_l)],
  [literal(true,_m),literal(true,_n),literal(true,_o),literal(true,_p)],
  [literal(true,_q),literal(true,_r),literal(true,_s),literal(true,_t)],
  [literal(false, _a), literal(false, _e)],
  [literal(false, _a), literal(false, _i)],
  [literal(false, _a), literal(false, _m)],
  [literal(false, _a), literal(false, _q)],
  [literal(false, _b), literal(false, _f)],
  [literal(false, _b), literal(false, _j)],
  [literal(false, _b), literal(false, _n)],
  [literal(false, _b), literal(false, _r)],
  [literal(false, _c), literal(false, _g)],
  [literal(false, _c), literal(false, _k)],
  [literal(false, _c), literal(false, _o)],
  [literal(false, _c), literal(false, _s)],
  [literal(false, _d), literal(false, _h)],
  [literal(false, _d), literal(false, _l)],
  [literal(false, _d), literal(false, _p)],
  [literal(false, _d), literal(false, _t)],
  [literal(false, _e), literal(false, _i)],
  [literal(false, _e), literal(false, _m)],
  [literal(false, _e), literal(false, _q)],
  [literal(false, _f), literal(false, _j)],
  [literal(false, _f), literal(false, _n)],
  [literal(false, _f), literal(false, _r)],
  [literal(false, _g), literal(false, _k)],
  [literal(false, _g), literal(false, _o)],
  [literal(false, _g), literal(false, _s)],
  [literal(false, _h), literal(false, _l)],
  [literal(false, _h), literal(false, _p)],
  [literal(false, _h), literal(false, _t)],
  [literal(false, _i), literal(false, _m)],
  [literal(false, _i), literal(false, _q)],
  [literal(false, _j), literal(false, _n)],
  [literal(false, _j), literal(false, _r)],
  [literal(false, _k), literal(false, _o)],
  [literal(false, _k), literal(false, _s)],
  [literal(false, _l), literal(false, _p)],
  [literal(false, _l), literal(false, _t)],
  [literal(false, _m), literal(false, _q)],
  [literal(false, _n), literal(false, _r)],
  [literal(false, _o), literal(false, _s)],
  [literal(false, _p), literal(false, _t)]   ] ).
```