ANDRÉ VELLINO

The Relative Complexity of Analytic Tableaux and SL-Resolution

Abstract. In this paper we describe an improvement of Smullyan's analytic tableau method for the propositional calculus–Improved Parent Clash Restricted (*IPCR*) tableau– and show that it is equivalent to SL–resolution in complexity.

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 measure of the relative complexity of proof systems is offered by the notion *polynomial simulation* [3]. 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 such that the length of β is a polynomial function of the length of α then system B *p-simulates* system A. Conversely, if 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 then B is strictly more powerful than A.

The *polynomial simulation* relation imposes a partial ordering among proof systems for the propositional calculus, and, although several simulation relations are known [4], there are still a number of gaps in the literature. In this paper we describe the *clash restricted improved analytic tableau* method for the propositional calculus (section 3) and show (section 4) that it is equivalent to (p- simulates and is p-simulated by) SL-resolution [7]. It has also be shown elsewhere [9] that SL-resolution is equivalent in complexity to the connection method [1]. We start, in section 2, by establishing some preliminary definitions that will be used in subsequent sections.

2. Definitions

Firstly we say that the symbols of the propositional calculus (PC) belong either to the set of *logical constants*, comprising the unary operator $\sim$ and the binary connectives: $\{\vee, \&, \equiv, \rightarrow\}$ or to the infinite set of atomic *propositional variables* and their complements: $\{a, b, c, \ldots, \sim a, \sim b, \sim c, \ldots\}$. A *literal* in PC

Studia Logica **52**: 323–337, 1993.

is either a propositional variable l or its complement $\sim l$. The *complement* of a literal $\sim l$ (l) is l ($\sim l$). A *formula* is either a literal or an expression of the form $\alpha \otimes \beta$, where α and β are sub-formulas and $\otimes$ is one of the binary connectives.

A *clause*, denoted by upper case letters $\{A, B, C, \ldots, A_1, B_1, \ldots\}$ is either a finite formula of the form $\alpha \vee \beta$, where α and β are either literals or clauses, or the formula containing no literals, $\emptyset$, referred to as the *empty clause*. A clause with just a single literal is called a *unit clause*.

Since the literals contained in a clause are all related by the same connective $\vee$, we will often represent the clause $a \vee b$ either in the abbreviated form ab. A set of clauses $\{A, B, C, \ldots\}$ is considered to be a conjunction of clauses, or equivalently, a formula in conjunctive normal form (CNF).

The variables S contained in a clause C may be given a truth value assignment (*tva*) by a map $\mathcal{T} : S \mapsto \{true, false\}$. Let V be a *tva* to the variables S in C. Then the truth value of C is a function (V) into the set $\{true, false\}$. The total number of *tvas* to C is 2^n where n is the number of variables in C, $\|S\|$. A complete listing of all the possible *tvas* to variables of a clause C and their transformation by the truth-functional connectives of C is the *truth-table* for C.

A clause C (or set of clauses Σ) is satisfied by a *tva* V to the variables in C (Σ) iff $\Im(V) = true$. C (Σ) is *satisfiable* if there is some *tva* that satisfies C (Σ). If all the possible *tvas* for C (Σ) satisfy C (Σ), then C (Σ) is a *tautology*. C (Σ) is *falsified* by a *tva* V to the variables S in C (Σ) iff $\Im(V) = false$. If all the *tvas* falsify C (Σ), then C (Σ) is *inconsistent* or *unsatisfiable*.

If Σ is an inconsistent set of clauses and $\Im(V)$ is a *tva* to the variables S in Σ then $\Im(V)$ is *critical* for a clause $C \in \Sigma$ if $\Im(V)$ satisfies all the clauses Σ in except C. Σ is *minimally inconsistent* if for every $C \in \Sigma$ there exists a *tva* that is critical for C.

3. Analytic Tableaux Methods

An *analytic tableau* θ for a set of propositional clauses Σ, is a tree such that all the nodes in θ 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 decom-

posed in that branch. For example, Fig. 1 shows a closed analytic tableau for the set of clauses $\Sigma_0 = \{ab, \sim ab, \sim b, \sim ac, ef\}$.

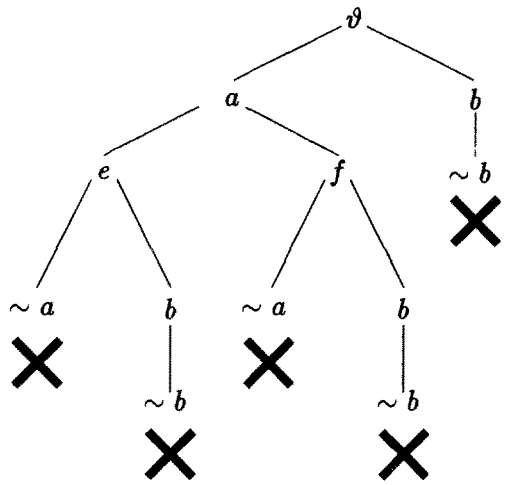

Fig. 1: Closed Tableau for Σ_0

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.

It is worth noting here that, with respect to worst case complexity, the analytic tableau method and the truth table method are incomparable. As D'Agostino points out in [5] there is indeed a class of problems that is more difficult to prove with tableaux than with truth-tables. Specifically, the class of 2^n clauses formed from the literals $\{p_1, p_2, \ldots, p_n\}$ such that every clause contains exactly one (positive or negative) occurence of each literal, produces minimal tableaux proofs whose size is on the order of $n!$, whereas the minimal truth tables for these example have only 2^{2n} entries (rows plus columns). For example, for the literals $\{p_1p_2p_3, \sim p_1p_2p_3, p_1 \sim p_2p_3, p_1p_2 \sim p_3, \sim p_1 \sim p_2p_3, \sim p_1p_2 \sim p_3, p_1 \sim p_2 \sim p_3, \sim p_1 \sim p_2 \sim p_3\}$ and the minimal tableau proof for them has 48 nodes. Conversely, there are classes of examples (such as those illustrated in Fig. 4 below) that have exponentially shorter minimal tableaux proofs compared to the size of their truth-tables. Thus the tableaux method is neither stronger than nor weaker than nor equivalent to the truth-table method.

3.1. Clash Restricted Analytic Tableau

The search for the smallest tableau 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. For instance, the tableau in Fig. 1 is clearly not minimal since a clause (ef) was chosen for decomposition which is logically independent of the remaining clauses.

The general tableaux method can be restricted to reduce the number of clauses from which to choose by imposing the rule that each formula chosen for 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. For example, Fig. 2 shows a clash restricted tableau for the same set of clauses as the unrestricted tableau in Fig. 1.

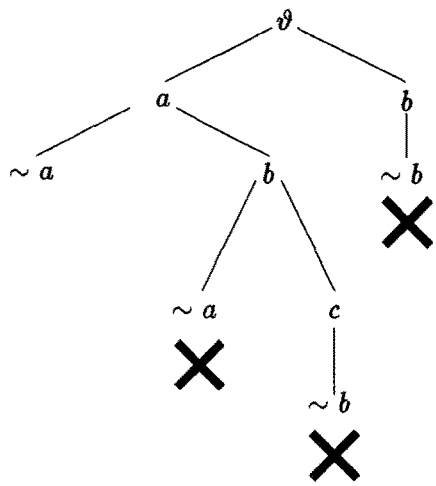

Fig. 2: Closed Tableau for Σ_0

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 an *ACR* and a *PCR* analytic tableau for Σ. Fig. 3 shows *PCR* analytic tableau for the set of clauses $\{ab, \sim ab, \sim b, \sim ac, ef\}$.

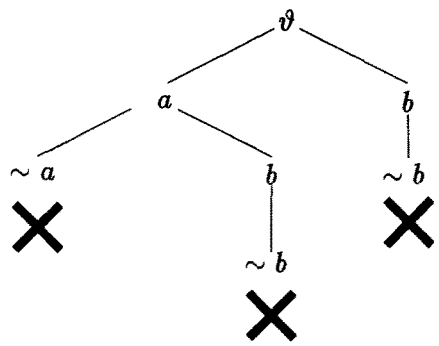

Fig. 3: Minimal Closed Tableau for Σ_0

Although it is not known whether either *PCR* or *ACR* tableaux p-simulate unrestricted analytic tableaux we do know that some minimal unrestricted tableau are smaller than both the minimal *PCR* and *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 in Fig. 4 are minimal and all the *PCR* or *ACR* tableaux for those sets of formulae are larger.

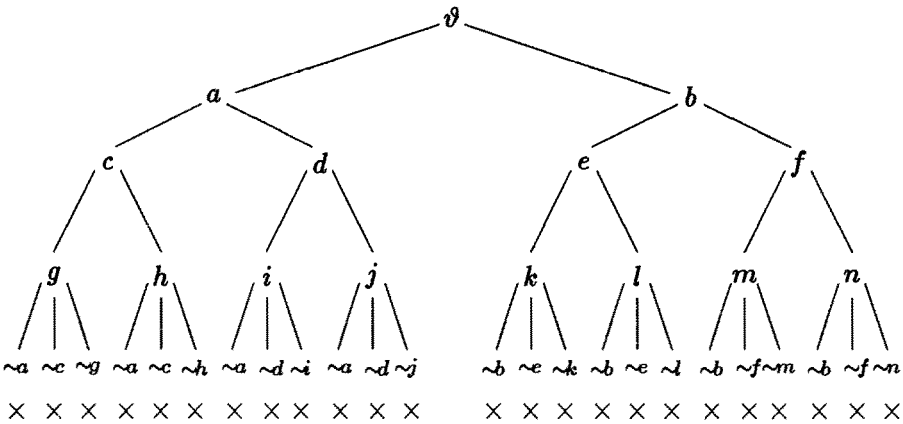

Fig. 4: Minimal Tableau for T_3

Since each formula in these minimally inconsistent sets is decomposed exactly once, by construction, the class of tableau T_n is minimal for $n \geq 3$. 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.

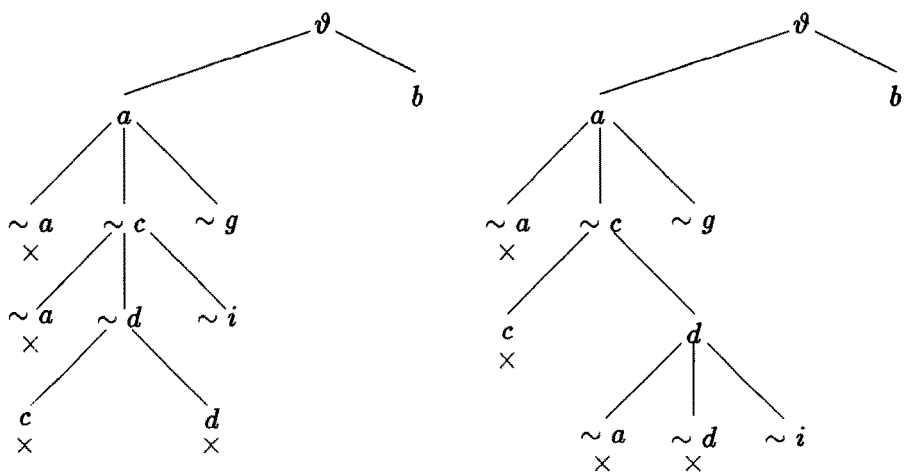

Fig. 5: Partial ACR tableaux for T_3

Given the start formula ab, the next formula beneath a must be one of the four formulae containing $\sim a$, say $\sim a \sim c \sim g$. The branch containing $\sim c$

can be closed by *cd* in a parent clash decomposition (right hand side of Fig. 5) or after some ancestor clash (left hand side of the figure). In either case there are still two more formulae containing $\sim c$ and $\sim d$ which have yet to be decomposed (since the set is minimally inconsistent) and whose branches must be closed by another decomposition of *cd*.

3.2. 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.

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

1. A subtableau is completed if it is closed.
2. 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.
3. A subtableau is completed if all its branches are closed or completed.

For example, a completed I-analytic tableau for the formulae $\{ab, \sim ab, \sim b\}$ is given in Fig. 6. Compare this with the tableau in Fig. 3.

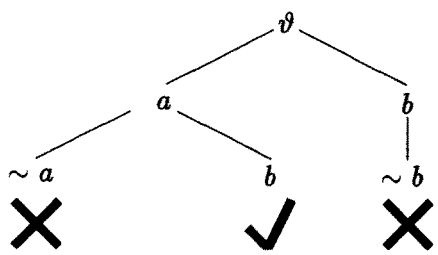

Fig. 6: Completed I-analytic tableau for Σ_0

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.

4. Linear Resolution

In this section we examine a refinement of resolution that has been studied extensively in the field of automatic theorem proving: *linear resolution.* We show that the SL refinement of resolution p-simulates and is p-simulated by the parent clash restricted improved analytic tableau.

4.1. SL-Resolution

A *linear resolution* proof is a resolution proof where each resolvent is one of the parents of the next resolvent and the other parent is either an input clause or a previous resolvent. Resolvents neither of whose parents are input clauses are said to follow by *reduction.* The SL refinement of linear resolution linear resolution with selection function was first introduced in [7]. SL– resolution is also a general form of OL (ordered linear) resolution ([2], p. 144-159).

The strategy underlying SL-resolution is derived from considerations common to *semantic resolution* and *set of support resolution* (see [2], Chapter 6). The idea with both of these restrictions is to force a choice of resolutions that will yield a contradiction quickly by imposing a strict order on the elimination of the literals. Since a partial *tva* that satisfies a group of clauses also satisfies all its resolvents it follows that a good strategy is to separate a (minimally) inconsistent set of clauses into two classes of self-consistent but mutually inconsistent clauses and resolve the clauses from each set against one another.

Applied to linear resolution, this insight means that the *tva* assigned to the literals of the set of input clauses S, must be *critical* for the start clause of the linear proof: that is, make the start clause false and each of the other input clauses true. The choice of start clause in a linear proof therefore imposes an order on the resolutions of clauses with one another. A further restriction on the progression of resolutions is to specify the order in which literals in a resolvent should be annihilated. In SL-resolution the

literal scheduled for the next resolution on the vine is the right-most literal of the current clause.

To avoid redundant sub-proofs, it is useful to keep track of the previous resolutions representing a clash of *tvas*. This means that a certain amount of syntactic overhead must be grafted onto each clause in a linear resolution in order to maintain a record of both the ordering of the literals in the clauses and the previously resolved literals, the proof then verifies that no *tva* can be consistently assigned to all the clauses in $S-\{\text{start clause}\}\cup\{\text{start clause}\}$. We ignore the method of choosing appropriate *tvas* and set of support. The results below are independent of such heuristic mechanisms.

To keep track of each literal which is resolved on from an input clause and to ensure that the next resolvent inherits the semantic information about the previously resolved literal until the information is no longer necessary, every resolved literal in an SL-resolution proof is kept track of by *framing* the resolved literal in the position in which it occurs in the previous resolvent. ([7] distinguish between A-literals, here called *framed* literals and B-literals, here called *unframed* literals, following [2].)

A sequence of sets of literals is called a *chain*. Each literal belonging to a chain is either framed or unframed depending on whether the literal has undergone a previous resolution. Input clauses then, are chains of unframed literals and SL resolvents, in general, are chains containing framed literals.

We call each sequence of contiguous unframed literals in a chain a *cell*. Thus a chain is a sequence of cells. Note that while the order of the elements in a cell is immaterial, the order of cells in the chain is significant since it partially determines the order in which literals are resolved.

Now, if any resolvent contains both a literal framed and its complement unframed, the *reduction* operation is trivial: it consists in simply deleting the unframed literal. On the other hand, a framed literal indicates that its resolution has already been performed. Therefore a resolution on a framed literal is equivalent to using the resolvent immediately prior to the framing of that literal, thus forming an arc in the vine (linear proof tree). If the right-most cell is empty, it may be discarded (retention of the information that previous resolutions on these literals has been performed is no longer necessary).

The choice of literal for resolution depends on a *selection function* which picks out a literal from the right-most cell containing a non-framed literal. The question of which literal is to be selected on for resolution does not affect the results below.

We can now define an SL-resolution [7] proof of C from a set of chains (clauses) Σ as a sequence of $C_1 \ldots C_n$ of chains such that:

1. $C_n = C$.
2. C_1 is an input clause
3. C_k, $1 < k \leq n$ is obtained from C_{k-1} by one of extension (with an input chain) or reduction (with a previous resolvent).
4. No two literals occurring at distinct positions in C_k have the same variable unless C_{k+1} is a reduction step (*admissibility* restriction).

A resolvent C_i is obtained by *extension* with an input chain B iff:

1. The right-most literal in C_{i-1} is an unframed literal.
2. The selected literal l in C_{i-1} (picked out by the selection function) is the complement of the left-most literal in B.
3. C_i is the chain obtained by concatenating C_{i-1} / l, $< l >$ and B / $\sim l$, in that order. The literal l in C_{i-1} is framed in C_i and every other literal has the same status as it had in its ancestor.

A resolvent C_i is obtained by *reduction* iff:

1. The right-most literal in C_{i-1} is an unframed literal.
2. The literal l occurs both framed and unframed in C_{i-1}.
3. C_i is the chain obtained by deleting the unframed occurrence of l in C_{i-1}.

C_i is obtained by *truncation* iff:

1. The right-most literal l inC_{i-1} is a framed literal.
2. C_i is obtained by deleting every framed literal to the right of the right most unframed literal in C_{i-1}.

C_i is obtained by *merging* iff:

1. C_{i-1} contains two or more occurrences of an unframed literal l.
2. C_i is obtained by deleting all the occurrences of l subsequent to the first.

For instance, an SL-resolution refutation for the set of clauses $\Sigma_1 = \{a{\sim}b, {\sim}ab, b{\sim}c, {\sim}bc, ac, {\sim}a{\sim}c\}$ is given in Fig. 7.

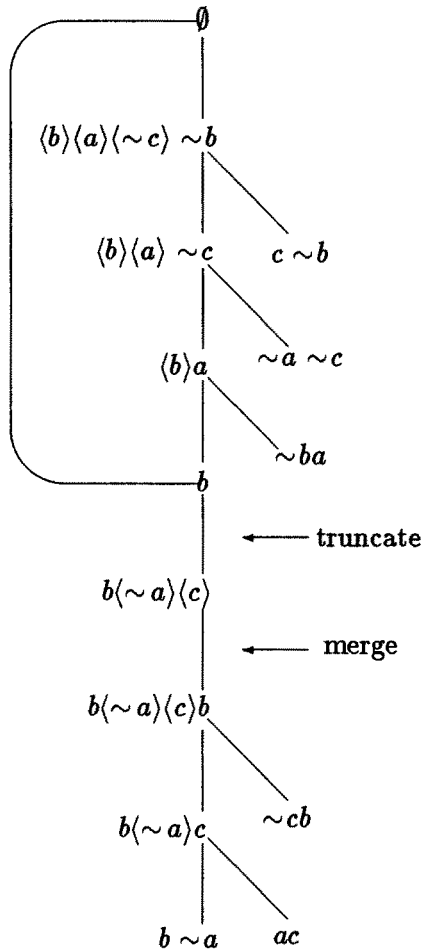

Fig. 7: SL-resolution vine for Σ_1

It should be noted from the definition of SL-resolution that, from the point of view of a worst case complexity measure, neither merging nor truncation nor reduction are crucial. The number of truncations will be a maximum in a refutation with no reductions. If the number of resolution steps is K, there will be at most K more truncation steps.

As for merging, it is evident that if L is the number of distinct literals in S then the merging procedure need be performed at most $L \times K$ times, where K is the number of extension steps in the proof only a linear increase. Similarly, the definition of reduction makes it clear that reduction steps can be made just by keeping track of previous resolutions (by framing literals). But since reduction steps add no new literals to the resolvent, performing the

reduction step is computationally equivalent to merging literals. One need only scan the resolvent chain to check for the presence of framed (unframed) literal and its unframed (framed) complement. Moreover, the number of reduction steps is also bounded by the number of extension steps. The worst case is when the number of merges in the proof is smallest, i.e. 0. Then each input resolution will add a maximum of $N-1$ literals to the chain, where N is the length of the longest input clause. Assuming that all the remaining literals are eliminated by reduction, the greatest number of reduction steps is bounded by $K \times (N-1)$ where K is the number of input clauses used in the proof times $N-1$.

4.2. Linear Resolution and Analytic Tableaux

In this section we show that SL-resolution and the improved parent clash restricted ($IPCR$) analytic tableau method exactly simulate each other.

Let θ be an $IPCR$ analytic tableau for the set of clauses Σ. Since the decomposition of the literals in a clause can be performed in any order, we can assume that the tableau is constructed in such a way that, at each decomposition stage, the literals are ordered so that the parent clashed literal is always right-most in the extension step. However, the tableaux are constructed depth-first from left to right.

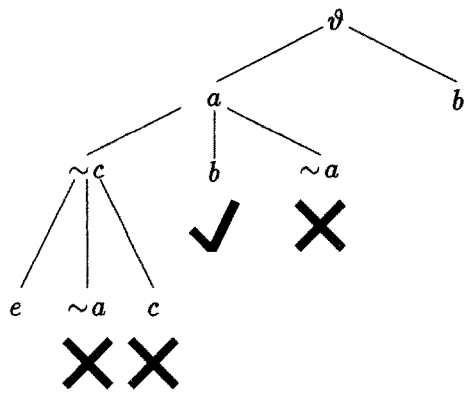

Fig. 8: Partial $IPCR$ tableau for Σ_1

Now let C' be the clause formed by traversing θ in reverse postorder [1] gathering all and only the *unchecked* literals at the leaves of θ up to and including the literal on the left-most open branch. Let C'' be the result of

[1] A postorder traversal of a tree traverses the subtrees of the first (left-most) subtree, visits the root and then traverses the remaining subtrees ([6], p.316)

replacing every literal p in C' that corresponds to a literal whose branch is closed by a parent literal, by the framed literal $<\sim p>$. Then let C be the result of deleting from C'' all the literals q whose branch is checked or closed by a complementary *ancestor* in θ.

For example, in the tableau shown in Fig. 8, $C' = b\sim ac\sim ae$, $C'' = b < a > <\sim c > \sim ae$ and $C = b < a > <\sim c > e$.

Now to prove that resolution and $IPCR$ analytic tableau exactly simulate each other we must show that a sequence of tableaux $\theta_1, \theta_2, \ldots \theta_n$ constructed in the stages described in section 3.2, corresponds to a sequence of chains in an SL-resolution proof. For the case of the singleton clause, $\Sigma = \{C\}$, the theorem is obvious. There are three cases to consider:

CASE 1: EXTENSION

If p labels a node in θ_n and $C_n = a_1 a_2 \ldots a_j p$ is the chain obtained from θ_n by the construction above, then θ_{n+1} is obtained by extending the tableau θ_n with a clause $C = \sim p q_1 q_2 \ldots q_k$.

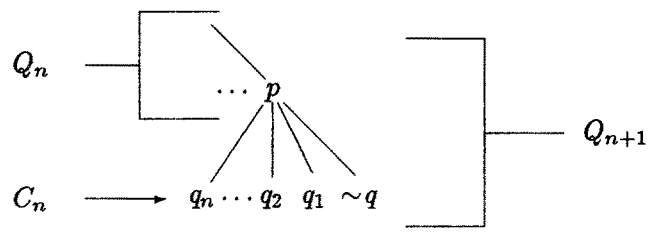

Fig. 9: Partial $IPCR$ tableau

The chain C_{n+1} obtained by construction on θ_{n+1} is then $a_1 a_2 \ldots a_j < p > q_1 q_2 \ldots q_k$ which corresponds exactly to an input resolution (extension) in the corresponding SL proof as shown in Fig. 10.

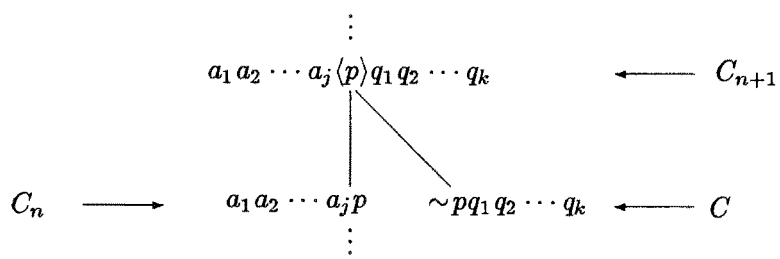

Fig. 10: Partial SL-resolution vine

CASE 2: CHECKING

Checking a literal in a tableau θ corresponds exactly to the merging step in an SL proof. If there are two open branches containing the literal p in θ then the SL clause obtained by construction has the form $a_1 a_2 \ldots p q_1 q_2 \ldots p \ldots$. After checking the left-most open branch the clause has the literal p merged to the left.

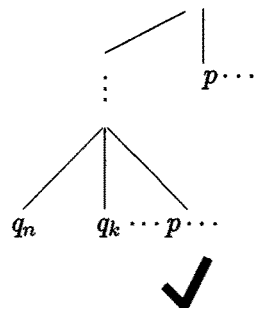

Fig. 11: Checked partial tableau

CASE 3: CROSSING

Closing a branch in θ whose endnode is labelled by a literal that is not complementary to a parent ancestor is the equivalent of the reduction operation in SL-resolution. QED.

Thus, an SL-resolution refutation can be interpreted as instructions for the construction of a $IPCR$ analytic tableau and visa versa. This result is quite interesting and somewhat unexpected given the semantic nature of analytic tableaux methods and the syntactic nature of resolution methods.

5. Conclusion

The PCR improved tableau method described in section 3.2 provides tableau proofs whose minimal size is of the order of SL-resolution proofs. However, just as resolution proofs methods require strategies to search for the smallest trees, minimal sized tableau proofs can be obtained only by choosing a strategic order in which to decompose the formulae. Although this paper does not discuss this problem it should be easy to adapt resolution-style optimization strategies to this semantic method.

References

[1] W. BIBEL, **Automated Theorem Proving,** Vieweg Verlag, Braunschweig; Wiesbaden: Vieweg 1982.

[2] C. L. CHANG and R. T. C. LEE, **Symbolic Logic and Mechanical Theorem Proving,** New York 1973.

[3] S. A. COOK, *The Complexity of Theorem Proving Procedures,* **Proc. 3rd ACM STOC, 1971, pp. 151 – 158.**

[4] S. A. COOK and R. A. RECKHOW, *The Relative Efficiency of Propositional Proof Systems,* **Journal of Symbolic Logic, 44** (1979), pp. 36 – 50.

[5] M. D'AGOSTINO, *Investigations into the Complexity of some Propositional Calculi,* Ph.D. Thesis, Oxford University Computing Laboratory, Technical Monograph PRG-88, 1990.

[6] D. E. KNUTH, **The Art of Computer Programming,** Vol.1, Addison-Wesley, 1968.

[7] R. KOWALSKI and D. KUEHNER, *Linear Resolution with Selection Function,* **Artificial Intelligence 2** (1971), pp. 227 – 260.

[8] R. SMULLYAN, **First Order Logic,** Springer-Verlag, New York 1968.

[9] A. VELLINO, *The Complexity of Automated Reasoning,* Ph.D. Thesis, Department of Philosophy, University of Toronto, 1989.

COMPUTING RESEARCH LABORATORY
BELL-NORTHERN RESEARCH
P.O. BOX 3511, STATION C
K1Y 4H7, OTTAWA, ONTARIO, CANADA
e-mail: vellino @ bnr.ca

Received June 17, 1992