Making "Clausal"Theorem Provers "Non-Clausal"

David L Poole

Research Report CS-85-52

December 1985

Making "Clausal" Theorem Provers "Non-Clausal"

David L Poole

Logic Programming and Artificial Intelligence Group
Department of Computer Science
University of Waterloo
Waterloo, Ontario
Canada N2L 3G1

ABSTRACT

There has recently been a number of papers which extend "clausal" theorem proving systems into "non-clausal" theorem proving systems. Most of these use the justification that the non clausal form eliminates redundancy by not multiplying out subterms. This paper presents the fallacy of such justification by presenting a way to convert to clause form without multiplying out subterms. It also shows how to generate a non-clausal extension to your favourite clausal theorem prover.

This paper originally appeared in *Proceedings of the fifth conference of the Canadian Society for Computational Studies of Intelligence (CSCSI/SCEIO)*, University of Western Ontario, London, Ontario, May 1984, pp. 124-125.

1. Introduction

Recently there has been a number of papers which give non-clausal extensions to clausal deduction systems. For example, the extension of resolution to non-clausal form (Manna and Waldinger[80], Murray[82]); the extension of Kowalski[75]'s connection graph proof procedure to non-clausal form (Stickel[82]); and the extension of Andrews[76]' matings to non-clausal form (Andrews[81]).

One of the major disadvantages attributed to clausal form is the need to multiply out subterms. This paper shows that this disadvantage can be overcome by choosing a different algorithm to convert to clause form.

2. Converting to Clausal form - Propositional Case

In this section we will show how to transform a wff in negation normal form, NNF (Andrews[81]) into conjunctive normal form, CNF† (or equivalently use Murray[82]'s notion of *polarity* and ignore all explicit negations).

Assume f is a formula in negation normal form (Skolemised, with equivalence and implication expanded out and negations moved in).

If f contains something of the form

$$(\alpha \vee (\beta \wedge \gamma))$$

then it is not in conjunctive normal form and the usual way to convert to conjunctive normal form is to multiply out subterms, viz:

$$((\alpha \vee \beta) \wedge (\alpha \vee \gamma))$$

Thus forming two copies of the subformula α. This causes a problem in theorem proving systems, as a large number of such transformations produces an exponential growth of subterms.

We instead form f^0 which is f with $(\alpha \vee (\beta \wedge \gamma))$ replaced by $(\alpha \vee p)$ where p is a unique atom (not appearing in f). Then form $f' = f^0 \wedge (\neg p \vee \beta) \wedge (\neg p \vee \gamma)$

Theorem: Repeated use of this transformation from f to f' (assuming associativity and distributivity of $\wedge$ and $\vee$) will convert a formula from NNF to CNF.

Proof: (1) the only way a NNF formula will not be in CNF is if it has a subexpression of the form $(\alpha \vee (\beta \wedge \gamma))$ in which case the transformation can be repeated.

(2) the number of $\wedge$'s within the scope of $\vee$'s is reduced by (at least) one each time, thus repeated use of the transformation will terminate.

Theorem: There is no multiplication of subterms in this conversion.

Proof: This is shown by noting that there is only one occurrence of each of α, β and γ in the resulting formula. The repeated p is only an atom and has no structure.

---

† If you would rather convert to disjunctive normal form then read the dual of the paper (swap $\wedge$ and $\vee$; swap *true* and *false*; and read *valid* for *unsatisfiable*).

Theorem (Correctness): This transformation preserves the unsatisfiability of the resulting formula.

Proof follows directly from the following Lemma.

Lemma: f is satisfiable if and only if f' is satisfiable.

Proof:

1. (Only if Case) - Suppose f is satisfied by interpretation I. We can assume, without loss of generality, that the denotation of p does not occur in the domain of I. If it does then it can be removed, creating an interpretation still satisfying f.

There are two cases to consider:

a) $(\beta \wedge \gamma)$ is true in I. In this case make $I'=I \cup \{p\}$. I' satisfies f^0 as none of the truth values have changed. (We have substituted a true value for a true value). I' satisfies $(\beta \wedge \gamma)$ so it satisfies $((\neg p \vee \beta) \wedge (\neg p \vee \gamma)$ so I' satisfies f'

b) $(\beta \wedge \gamma)$ is false in I. In this case make $I'=I \cup \{\neg p\}$. Then I' satisfies f^0 as none of the truth values has changed. $\neg p$ is true in I' so $((\neg p \vee \beta) \wedge (\neg p \vee \gamma))$ is true in I', so I' satisfies f'.

2. (If Case) - Suppose f' is satisfied by interpretation I'. Then in particular both f^0 and $((\neg p \vee \beta) \wedge (\neg p \vee \gamma))$ are true in I'. There are two cases to consider:

a) p is true in I'. Then as $((\neg p \vee \beta) \wedge (\neg p \vee \gamma))$ is true in I', (β and γ) must be true in I', so f is true in I' as none of the truth values have changed.

b) p is false in I'. In this case f must be true in I' as replacing something that was false in a conjunction or a disjunction cannot make the conjunction or disjunction false when it was previously true. Note that there are no negations to be considered, as all negations are moved in, and p does not involve a negation in f.

Q.E.D.

3. The Predicate Calculus Case

The predicate calculus conversion is like the propositional calculus case except that the new atomic formula introduced is of the form $P(z_1, \cdots, z_n)$ where $z_1, \cdots, z_n$ are the free variables in $(\beta \wedge \gamma)$ and P is a unique n-place predicate symbol. Let f' be created from f in the same way as for the ground (propositional) case.

Theorem: For the predicate calculus case, f is unsatisfiable if and only is f' is unsatisfiable.

Proof:

Case 1: Suppose f is satisfied by I. Define $P(z_1, \cdots, z_n)$ to be true in I' in exactly those cases for which $(\beta \wedge \gamma)$ is true in I. Then for each of the values for the variables $z_1, \cdots, z_n$ the same argument as for the ground case holds. So f' is satisfied by I' for all values of $z_1, \cdots, z_n$

Case 2: Suppose f' is satisfied by I' Then for each value of $z_1, \cdots, z_n$ the ground argument holds. So f is satisfied by I'.

Q.E.D.

4. Using the Transformation

4.1. What Has Been Gained?

The gain that occured is that there is only one copy of α in the resulting formula. If α is a large structure, then once α has been proven false (or resolved away) once then both β and γ can be used. In the distributive form, α must be resolved away for both β and γ.

If n is the number of $\wedge$'s in the scope of $\vee$'s, then in the transformation here there are $2n+1$ clauses produced. In the traditional transformation there may be 2^n clauses produced. (Consider the case of converting something in disjunctive normal form (two literals per conjunct) into conjunctive normal form.)

4.2. Making the Transformation Implicit

The disadvantage of such a transformation may be in the cost of creating the new literals. In this section we show how to avoid creating new literals, and how to avoid doing the explicit transformation at all. The first approach is to change the deduction system to make the transformation implicit as a special case. The second is to modify the preprocessing that has to be carried out before the (unchanged) deduction system runs.

As an example of the former, consider a resolution-type theorem prover. In the transformed system, the only atoms that p can unify with are the instances of p explicitly created in the transformation. In particular, only three instances of the atom p appear. If p is ever successfully resolved away then both α and either β or γ is resolved away. If α is resolved away, then instead of leaving the residual literal p, and letting it resolve with one of the clauses $(\neg p \vee \beta)$ or $(\neg p \vee \gamma)$, and producing β or γ to be resolved away, the deduction system can be modified to recognise this case and produce β or γ one step earlier. The other case of having resolved away one of β or γ, leaves $(\alpha \vee p)$ as the only choice to resolve away p. Instead of having p explicit, the theorem prover can try immediately to resolve away α.

In connection graph proof procedures, the connection graph contains all of the information about unifications. In particular, after the connection graph is built, the internal form of the literals is irrelevant. A connection graph builder, instead of creating the p's and then adding the two connections, and forgetting about the internal forms of the p's, can build the connections without creating the p's at all.

5. Conclusion

This paper demonstrates that one of the advantages that "non clausal" theorem proving has over "clausal" theorem proving is **not** that converting to clause form multiplies out subterms.

In any theorem proving method the only unifications with the introduced atomic symbol will be those given by the procedure above. Therefore the effect of the connection can be calculated before any actual deduction, so the above procedure may not need to be carried out at all.

If you like the idea of non-clausal theorem proving then find your favorite clausal theorem prover; allow input in non-clausal form; find a way to do the transformation above implicitly; and you have an extension of the theorem prover to the non-clausal case.

6. References

Andrews,P.B.[76], "Refutation by Matings", *IEEE Trans. Comput. C-25.* pp 801-807.

Andrews,P.B.[81], "Theorem Proving via General Matings", *Journal A.C.M.* Vol 28, No 2, pp 193-214.

Kowalski,R.[75], "A Proof Procedure Using Connection Graphs", *J.A.C.M.* Vol 22, No 4, pp 572-595.

Manna,Z. and Waldinger,R.[80], "A Deductive Approach to Program Synthesis", *A.C.M.T.O.P.L.A.S.* Vol 2, No 1 pp 90-121.

Murray,N.V.[82], "Completely non-clausal theorem proving", *Artificial Intelligence,* Vol 18, No 1, pp 67-85.

Stickel,M.E.[82], "A nonclausal connection-graph resolution theorem-proving program", *AAAI-82,* pp 229-233.