In memory of my mother

LOCKING:
A RESTRICTION OF RESOLUTION

by

Robert S. Boyer, B.A.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

In Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN
August 1971

Preface

For a first reading, may we suggest the following.

1) Read the Introduction for a general view of the subject matter.

2) Follow carefully the example at the beginning of Chapter 1 for a basic idea of what resolution is.

3) Read Chapter 2 (which marks the beginning of the material original to this thesis) for an explanation of the locking restriction on resolution.

4) Muse over a few of the examples of locking proofs in Chapter 6.

5) Read Chapters 3 and 4, the core of the thesis, after a cursory reading of the definitions in Chapter 1. Refer to these definitions (by means of the Index) when necessary.

Acknowledgements

W.W. Bledsoe made several substantial suggestions that are incorporated in this thesis. The concept of locking is derivative from a proposal he made.

Robert B. Anderson made several valuable criticisms and section 5.7 is his discovery.

Marvin Minsky made available the facilities of the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology during 1970-1971.

W.W. Bledsoe, Anne Boyer, and Vesko Marinov made many textual and stylistic corrections.

NASA gave me financial support from 1967 to 1970.

NIH gave me support in 1970-71 through grant GM 15769-03 to W.W. Bledsoe.

ARPA supports the AI Lab at MIT.

I am most deeply indebted to Prof. Bledsoe and my wife Anne for inspiration and encouragement. Dr. Bledsoe, with his enthusiasm and profound zeal for mathematical thinking, has been my principal guide throughout my graduate education. My wife's love and devotion have held me together.

July 1971

Abstract

A restriction, called "locking", of the resolution deductive system of J.A. Robinson is presented. Locking involves arbitrarily indexing with integers the literals in the clauses to be resolved. Different occurrences of the same literal may be indexed differently. Resolution is then permitted only on literals of lowest index in each clause. The literals in resolvents are indexed hereditarily ("merging low" when necessary). This restriction is shown to be complete. Locking results in a significant reduction in the number of clauses generated. Locking is compared to other restrictions of resolution and is shown to be incompatible with some. Several examples of locking derivations are given. Finally, a special application of locking to a troublesome axiom is described which reduces the irrelevant clauses generated by that axiom.

Table of Contents

Introduction

It is an old wish that we might have rules by which to think. To some degree mathematics gives us rules for thinking about the world. It is only natural that logicians have tried to produce mathematical rules for thinking about mathematics. The invention of computers has inspired an enthusiasm for a practical mechanization of mathematical inference. Though the important results so far produced in this endeavor have been about, rather than by means of, computation, this field is known as "automatic theorem proving."

The foundation or "ground" of automatic theorem proving lies in the concept of the tautology. The propositional logic is essentially the calculus of tautologies; and any question in the propositional logic can be decided by the method of "truth tables." As anyone knows who has computed the truth tables for a few propositions, questions about tautologies are essentially boring. They would not be of real interest to anyone, perhaps, were it not for the remarkable theorem of J. Herbrand.

While trying to reduce mathematical logic to a mechanizable system in 1930, Herbrand discovered that any question in the first order predicate logic can "almost" be reduced to a question of tautologies. It is generally believed that most of mathematics can be done in the first order

predicate calculus.

Roughly speaking, Herbrand showed that if P is a formula in the predicate logic and H is the collection of all names of objects,(H is commonly known a Herbrand's Universe), then P is a theorem if and only if some finite disjunction of instances of P by terms in H is a tautology. The "some" in his theorem is the reason that we must say he "almost" reduced logic to tautologies. It is Church's theorem that tells us the "almost" can never be erased; for if we could find a procedure that would give us that "some" for any P (or tell us if such did not exist), we would have a decision procedure for the predicate calculus. But there is none.

Despite the impossibility of deciding whether there is some finite disjunction of instances for any P, we can still always find a proof for any P, if P is a theorem. For example, we might simply proceed to examine all the possible instances. This procedure was actually tried on computers in 1960 by Wang(20),Prawitz(13), and Davis(7). It was discovered, however, to be a task too demanding for the computers in existence then or now.

In 1960 D. Prawitz(14) and in 1963 J.A. Robinson(16) independently discovered a method for greatly increasing the search for the right instantiations. They showed that one may search in a perspicuous fashion, high above the ground level of the proposition logic, considering infinitely many instances "in

a single bound." Robinson's discovery he named the process of "unification"; he embedded it in a inference system called "resolution," which system is the basic framework for most work done since then in "automatic theorem proving."

Even though resolution provides a vast improvement over the method of instantiation, it is still plagued by an exponential explosion in the number of "clauses" it generates; this explosion is unmanageable on hard problems. Many researchers have found strategies that restrict this growth while still preserving resolution's completeness. In this thesis we present such a restriction which we call "locking." In Chapter 1 we present a grammar of resolution. In Chapter 2 we introduce the concept of locking. Chapters 3 and 4 contain the proofs of the completeness of locking in the "ground" and "general" cases. In Chapter 5 we describe the relationship of locking to other restricted forms of resolution. Chapter 6 contains some examples of locking refutations. Chapter 7 describes a special application of locking to a common but troublesome axiom.

1. A Grammar of Resolution

Before precisely defining the concepts of resolution we offer a brief example of a resolution proof.

For any binary predicate G, (such as "=" or "<") the following is a theorem.

$$\forall x \exists y (((G\ y\ x) \rightarrow \exists w\ (G\ w\ y)) \wedge ((\exists z\ (G\ z\ y) \wedge (G\ y\ z)) \rightarrow (G\ y\ x))))$$

To prove this theorem by resolution we first transform it into the set of five clauses:

1 [[(G y (a)) (G (f y) y)]

2 [(G y (a)) (G y (f y))]

3 [¬(G w y) (G (f y) y)]

4 [¬(G w y) (G y (f y))]

5 [¬(G w y) ¬(G y (a))]]

The rules for this transformation are quite straightforward(cf. p.18). The members of the clauses are called "literals", e.g. ¬ (G y (a)) in the fifth clause. The literals without the "not" sign (¬) at the front are called "atoms." The symbol "f" is called a "function symbol" and was introduced during the transformation. The symbol "a" is called a "constant," and it also appeared during the transformation. "(f x)" and "(a)" are called "terms." The symbols "w" and "y" are, of course, variables, and we think of them as being replaceable.

Once we have these original clauses, we try to "resolve" them together. Resolving involves taking any two clauses, picking any two of their literals (one from each clause), and checking for a "match" of a certain kind. (This matching is called unification[1]). If this match is found, we then create a new clause, called a "resolvent," from the literals (other than the ones we matched) in the two clauses we are resolving.

For example, we take clause 1 and clause 3, and try to "match" their first literals

(G y (a)) ¬(G w y).

To match them, we first check to see that one is an atom and one has the "not" sign. We then look for a way of replacing some of the variables in these two literals so that the two resulting literals will be identical (except for the "not" sign). If we replace y by w in (G y (a)) and y by (a) in ¬(G w y) we shall have

(G w (a)) and ¬(G w (a)).

Therefore, a "match" exists. And therefore, we may produce a resolvent.

The resolvent produced is the set consisting of the other literal in clause 1 (with y replaced by w) and the other literal in clause 3 (with y replaced by a), namely

6 [(G (f w) w) (G (f (a)) (a))].

In a similar way we resolve the first literal in

clause 2 and the first literal in clause 4 to get the resolvent

7 [(G w (f w)) (G (a) (f (a)))]

In fact, it is possible to resolve the first five clauses in twenty ways. After making all these resolvents, we then proceed to resolve the resolvents with one another and with the original five clauses. Then we resolve the new resolvents with one another, with the previous resolvents, and with the original five; and so on.

The purpose of all this resolving is to generate the empty set as a resolvent (we denote the empty set by "□"). For once we have generated □, we shall have "proved" the theorem. But the only way to generate □ as a resolvent is to resolve two clauses which have only one literal apiece (since only then would there be no "other" literals in either clause to appear in the resolvent.) The reader may guess, therefore, that we have one more process in store.

This other process is called "factoring" (1.18). To factor a clause is to replace some of the variables in the literals of the clause in such a way that the clause "shrinks." For example, if we replace w by (a) in clause 6, we obtain the clause

8 [(G (f (a)) (a)))]

This "shrinkage" occurs simply because a clause is a set and no set can "contain the same element twice."

Similarly,

9 [(G (a) (f (a)))]

is a factor of 7.

In resolution, we admit the factors of clauses to full standing, i.e., in addition to resolving our original clauses, their resolvents, etc. we also resolve on the factors of every clause in sight.

In particular, we can resolve clause 9 with clause 5. For if we replace y by (f (a)) and w by (a) in the literal $\neg$(G w y), we get $\neg$(G (a) (f (a))), which is identical to the only literal in clause 9 (except for the "not" sign). Hence 9 and 5 resolve to produce

10 [$\neg$ (G (f (a)) (a))]

i.e. the set consisting of the other literal in clause 5 (with y replaced by (f (a)) and w replaced by (a).)

Now we can resolve 8 and 10 to obtain the desired $\square$. Thus we have proved the theorem by means of resolution.

The main result of this chapter is the Resolution Theorem 1.26. Those familiar with resolution may well skip the entire chapter after a cursory glance at 1.15, 1.16, 1.17, 1.19, and 1.20. Virtually all of the definitions and theorems may be found in (16).

Those who tire of the definitions and theorems before 1.26 may well begin reading after 1.26 and refer to the earlier material when necessary.

We now describe the language in which resolution is the rule of inference.

1.1 The Symbols

The language is determined by three disjoint sets of symbols, viz., the variables, the function symbols, and the predicate symbols.

The variables are infinite in number and include:

w,x,y,z,w',x',y',z',w'',x'',...

There are a countable number of function symbols. With each function symbol is associated exactly one non-negative integer called "the number of arguments" of that symbol. A function symbol of 0 arguments is called a "constant." The language has at least one constant.

With each predicate symbol is associated exactly one non-negative integer called "the number of arguments" of that symbol.

There are four additional symbols,

( ) ¬ /

1.2 Terms

From the function symbols and variables are built the "terms" of the language. To be precise, let H be the smallest set such that

1) H contains the variables of the language, and

2) if t1, t2, t3, ... tn are members of H and f is a function symbol of n arguments, then H contains the string (f t1 t2 ... tn).

H is the set of terms.

1.3 Atoms

If P is a predicate letter of 0 arguments, then P is an atom. If P is a predicate letter of N arguments and t1, t2, t3, ... tn are terms, then the string

(P t1 t2 t3 ... tn)

is an atom.

1.4 Literals, Complements, and Signs

If A is an atom then

1) A is a "literal",

2) the string

$\neg$ A

is a "literal",

3) A and $\neg$ A are "complements" of one another,

4) the "sign" of A is "T" and the "sign" of $\neg$ A is "F",

and

5) A is called "the atom" of both A and $\neg$ A.

1.5 Clauses

A clause is a finite set of literals. The empty clause is the empty set and is denoted by "$\square$". A clause with exactly one member is called a "unit" clause.

1.6 Groundness

A clause, a literal, or an atom is called "ground" if and only if no variable occurs in it.

1.7 Substitution Components

If x is a variable and t is a term and x is not t, then the string

t/x

is called a "substitution component." x is called the "second part" of the component and t is called the "first" part of the component. (We think of the second part as about to be replaced and the first part as the expression that will take the place of the second part.)

1.8 Substitutions

A finite set of substitution components is called a "substitution" provided no two distinct members have the same

second part. (The reason a substitution cannot have two components, say t/x and t'/x , with the same second part is that we could not know whether to replace x with t or with t'.)

1.9 Instances

Suppose that L is a term, a literal, or a clause. The instance of L under a substitution σ is the result of simultaneously replacing in L each variable x that is the second part of a component t/x of σ with the t.

"L_σ" denotes the instance of L under σ. If L_σ is ground(i.e. has no variables), it is called a ground instance of L. If σ is a substitution, L is a set of literals, and L_σ has as many members as L, then L_σ is called a "direct instance" of L.

1.10 Variants

If E is an instance of F and F is an instance of E, then E and F are called "variants" of each other.

For any E there exists an F such that E and F have no variables in common and E is a variant of F.

If σ and τ are substitutions, we would like to have a substitution γ such that $(E_\sigma)_\tau$ is E_γ, i.e. the composition of σ and τ.

1.11 Composition of Substitutions

Suppose σ and τ are substitutions. Let γ be the substitution to which the component t/x belongs if and only if either

1) for some t', t'/x is a component of σ and $t = t'_{\tau}$,

or

2) x is not a second part of a component of σ and t/x is a component of τ.

γ is called the "composition" of σ and τ and is denoted by " $\sigma\tau$ ".

1.12 Unification

If S is a set of literals, σ is a substitution, and S_{σ} is a singleton, then σ is said to "unify" S.

Consider the set C

[(G y x) (G (f x) z)].

Each of the following substitutions unifies C.

[(f(f(f(f(a))))/y (f(f(f(a))))/x
(f(f(f(a))))/z]

[(f (a))/y (a)/x (a)/z]

[(f (x))/y x/z].

1.13 Most General Unifiers

If σ unifies S and for each τ that unifies S there exists a

λ such that $\sigma\lambda = \tau$, then σ is said to be a "most general unifier" of S.

The substitution

$$[\ (f(x))/y \quad x/z\ ]$$

is a most general unifier of C above.

1.14 Unification Theorem

If any substitution unifies S, then there exists a most general unifier of S.

1.15 Simultaneous Unification

If S1, S2, ... Sn is a sequence of sets of literals, σ is a substitution, and for each i, Si_σ is a singleton, then σ is said to "simultaneously unify" S1, S2, ... Sn.

1.16 Most General Simultaneous Unifiers

If σ simultaneously unifies S1, S2, ... Sn and for each τ that simultaneously unifies S1, S2, ... Sn there exists a λ such that $\sigma\lambda = \tau$, then σ is said to be a "most general simultaneous unifier" of S1, S2, ... Sn.

1.17 Simultaneous Unification Theorem

If any substitution simultaneously unifies S1, S2, ... Sn, then there exists a most general simultaneous unifier of S1, S2, ...

Sn.

1.18 Factors

If C is a clause, T is a subset of C, and σ most generally unifies T, then C_σ is called a "factor" of C. (Since a ground clause has no non-trivial instances, it has no non-trivial factors.)

1.19 Fully-factored Sets

A set S of clauses is said to be "fully-factored" if and only if some variant of every factor of every member of S is a member of S.

1.20 Fully-factored Theorem

If C is a member of a fully-factored set S of clauses and σ is a substitution, then there exist a member C' of S and a substitution τ such that C_σ is a direct instance of C' under τ.

1.21 Interpretations

A set of ground literals I is called an "interpretation" provided that for each ground atom A, either A is a member of I or $\neg A$ is a member of I, and not both are.

1.22 Models and Satisfiability

If S is a set of clauses and I is an interpretation, then I is called a "model" of S if and only if each ground instance of each member of S contains some member of I. S is called "satisfiable" if and only if some interpretation is a model of S.

1.23 The Model Theorem

If S is a finite set of clauses, there exists a model of S if and only if for each sequence of substitutions $\sigma_1, \sigma_2, \ldots, \sigma_n$ such that for all i, S_{σ_i} is a set of ground clauses, $S_{\sigma_1} \cup S_{\sigma_2} \cup \ldots \cup S_{\sigma_n}$ admits a model.

We now prove the Model Theorem. By the definition of model, if S admits a model M, then M is a model for any $S_{\sigma_1} \cup S_{\sigma_2} \cup \ldots \cup S_{\sigma_n}$. Hence, one half of the Model Theorem is trivial.

The other half is only a variation on König's Lemma. One might view the Model Theorem as a compactness theorem. Suppose that for each sequence $\sigma_1, \sigma_2, \ldots, \sigma_n$ of substitutions such that for all i, S_{σ_i} is a set of ground clauses, $S_{\sigma_1} \cup S_{\sigma_2} \cup \ldots \cup S_{\sigma_n}$ admits a model. There are only countably many substitutions. Let $\tau_1, \tau_2, \tau_3, \ldots$ be an enumeration of those substitutions τ such that S_τ is a set of ground clauses. Let us denote by I(m) $S_{\tau_1} \cup S_{\tau_2} \ldots \cup S_{\tau_m}$.

Let A1, A2, A3, ... be an enumeration of the ground atoms.

For each i, we shall recursively define an M_i and it will turn out that $\bigcup_{i>0} M_i$ is a model of S.

There is a literal L such that

1) either L is A_1 or L is $\neg A_1$ and,

2) for each m, L is a member of some model of I(m).

For suppose that for some j_0 A_1 is not a member of some model of $I(j_0)$. Then for all $k > 0$, A is not a member of any model of $I(j_0+k)$, since any model of $I(j_0+k)$ is a model of $I(j_0)$. Similarly, if for some j_1 $\neg A_1$ is not a member of a model of $I(j_1)$, then $\neg A_1$ is not a member of a model of $I(j_1+k)$. But then neither of A_1, $\neg A_1$ is a member of a model of $I(j_0+j_1)$. This, however, contradicts the assumption that for every $\sigma_1, \sigma_2, \cdots, \sigma_n$ such that S_{σ_i} is a set of ground clauses $S_{\sigma_1} \cup S_{\sigma_2} \cup \ldots \cup S_{\sigma_n}$ admits a model.

We define M_1 to be [L] for such an L. Suppose that for some $I > 0$ M_i has been defined and that for each m, M_i is a subset of some model of I(m). By an argument similar to the one immediately above:

There is a literal L' such that

1) either L' is A_{i+1} or L' is $\neg A_{i+1}$ and

2) for each m, $M_i \cup [L']$ is a subset of some model of I(m).

Let $M_{i+1} = M_i \cup [L']$. Then for each m, M_{i+1} is a subset of some model of I(m).

Now let $M = \bigcup_{i>0} M_i$. Then for each m, M is a model

of I(m). For, let J be a positive integer such that if L is a literal in I(m) and A_n is the atom of L in the ordering A1, A2 , A3, ... of the atoms, then n < J. Then if C is a clause in I(m), some member of M_J is a member of C. Therefore, M is a model of S.QED.

1.24 Resolvents

If C and D are clauses, L1 and L2 are literals in C and D respectively, the sign of L1 is not the sign of L2, and there exists a most general unifier σ of the atoms of L1 and L2, then

$$((C - [L1]) \cup (D - [L2]))_\sigma$$

is called a "resolvent" of C and D. [2]

1.25 Resolution

If S is a set of clauses, then let R(S,0) be S and for each positive integer i+1 let R(S,i+1) be the union of R(S,i) and the set of all resolvents of factors of members of R(S,i). Let $R(S, \omega)$ be $\bigcup_{i \in \omega} R(S,i)$.

1.26 Resolution Theorem

If S is a finite set of clauses, S is unsatisfiable if and only if $\square \in R(S, \omega)$.

The Clausal Transformation and Herbrand's Theorem.

Before proving the Resolution Theorem (1.26) we do two things. First, we show how to transform any question in the first order predicate calculus [3] into a set of clauses. Second, we state Herbrand's Theorem, which tells us why resolution is of interest.

Suppose we wish to prove in the first order predicate calculus that C follows from A1, A2, ..., An. Without loss of generality we assume there are no free variables in any of C, A1, A2, ... , An. Let TH be the formula $A1 \wedge A2 \wedge \ldots \wedge An \wedge \neg C$. (Thus in our example at the beginning of this chapter TH is just

$$(\neg \ \forall x \exists y(((G\ y\ x) \rightarrow \exists w\ (G\ w\ y)) \wedge ((\exists z\ (G\ z\ y) \wedge (G\ y\ z)) \rightarrow (G\ y\ x)))).$$

Now perform the following three operations on TH.

1) Transform TH into prenex normal form. (For our example, this is

$$\exists x \forall y \exists z \forall w \ \neg (((G\ y\ x) \rightarrow (G\ w\ y)) \wedge ((G\ z\ y) \wedge (G\ y\ z) \rightarrow (G\ y\ x))).$$

2) If TH has the form $\forall x_1 \forall x_2 \ldots \forall x_n \exists y B$ (i.e., y is the first existentially quantified variable in TH) then let f be a new function symbol of n arguments and transform TH into the result of replacing y by $(f\ x_1\ x_2 \ldots x_n)$ in $\forall x_1 \forall x_2 \ldots \forall x_n B$. If TH still

has any existentially quantified variables, perform step 2 again.

(Step 2 is referred to as "skolemization" and the new function symbols introduced are called "skolem functions." In our example, step 2 is performed twice and the result is

$$(\forall y\ \forall w\ \neg(((G\ y\ (a)) \rightarrow (G\ w\ y)) \wedge ((G\ (f\ y)\ y) \wedge (G\ y\ (f\ y)) \rightarrow (G\ y\ (a))))).$$

3) Transform TH into conjunctive normal form. In our example, this is

$$\begin{aligned}&\forall y\ \forall w\ (((G\ y\ (a)) \vee (G\ (f\ y)\ y))\\ &\quad \wedge((G\ y\ (a)) \vee (G\ y\ (f\ y)))\\ &\quad \wedge(\neg(G\ w\ y) \vee (G\ (f\ y)\ y))\\ &\quad \wedge(\neg(G\ w\ y) \vee (G\ y\ (f\ y)))\\ &\quad \wedge(\neg(G\ w\ y) \vee \neg(G\ y\ (a)))\ )\end{aligned}$$ [4]

After these three steps have been performed, TH is of the form

$$\begin{aligned}&\forall x_1 \ldots \forall x_p(\ (\ L_{11} \vee L_{12} \vee \ldots \vee L_{1k_1}\ )\\ &\quad \wedge(\ L_{21} \vee L_{22} \vee \ldots \vee L_{2k_2}\ )\\ &\quad \wedge \ \ldots\\ &\quad \wedge(\ L_{m1} \vee L_{m2} \vee \ldots \vee L_{mk_m}\ )\ ).\end{aligned}$$

Each L_{ij} is a literal as defined at (1.4). TH is inconsistent if and only if C follows from A1, A2, ..., An.

Let us call

$$((L_{11} \vee L_{12} \vee \ldots \vee L_{1k_1})$$
$$\wedge (L_{21} \vee L_{22} \vee \ldots \vee L_{2k_2})$$
$$\ldots$$
$$\wedge (L_{m1} \vee L_{m2} \vee \ldots \vee L_{mk_m}))$$

the negated-skolemized-conjunctive-normal-matrix of (C, A1, A2, ..., An) or NSCNM(C, A1, A2, ... An) for short.

Let S be the set of clauses

$$[[L_{11} \; L_{12} \ldots L_{1k_1}]$$
$$[L2_1 \; L_{22} \ldots L_{2k_2}]$$
$$\ldots$$
$$[L_{m1} \; L_{m2} \ldots L_{mk_m}]]$$

S is precisely the set of clauses we give to resolution to decide whether C follows from A1, A2, ..., An. Let us call S the clausal form of C, A1, A2, ..., An or CL(C,A1, A2, ..., An) for short.

Now that we have shown how to obtain clauses from questions in the first order predicate calculus, we state Herbrand's Theorem and show its connection with resolution.

Herbrand's Theorem

If D is NSCNM(C, A1, A2, ..., An), then C follows from A1, A2, ..., An if and only if there exist substitutions $\sigma_1, \sigma_2, \ldots, \sigma_k$ such that for each i, D_{σ_i} has no variables and $(D_{\sigma_1} \wedge D_{\sigma_2} \wedge \ldots \wedge D_{\sigma_k})$ is inconsistent. [5]

What is the connection with resolution? Suppose that $\sigma_1, \sigma_2, \ldots, \sigma_k$ are substitutions and for each i, D_{σ_i} has no variables. Then by the method of truth tables, $(D_{\sigma_1} \wedge D_{\sigma_2} \wedge \ldots \wedge D_{\sigma_k})$ is inconsistent if and only if there exists no way of assigning the value T to some literal in each disjunction of each D_{σ_i} without assigning T to some literal and its complement. It immediately follows that $(D_{\sigma_1} \wedge D_{\sigma_2} \wedge \ldots \wedge D_{\sigma_k})$ is inconsistent if and only if $(S_{\sigma_1} \cup S_{\sigma_2} \cup \ldots \cup S_{\sigma_k})$ does not admit a model (recall that we let S = CL(C,A1, A2, ... , An).) Now by the Model Theorem (1.23), C follows from A1, A2, ... , An if and only if S does not admit a model. Therefore, once we have established the Resolution Theorem (1.26) we shall have

The Completeness and Soundness of Resolution

If S is CL(C, A1, A2, ..., An), then C follows from A1, A2, ..., An if and only if $\square \in R(S, \omega)$.

We thus complete the digression on the clausal transformation and Herbrand's Theorem and begin proving the Resolution Theorem (1.26).

One half of (1.26) is easy, namely:

1.26.1 If S has a model, then $\square \notin R(S, \omega)$.

We first prove by induction on n,

1.26.2 Lemma

If S is a set of clauses and M is a model of S, then for each $C \in R(S,n)$ and for each λ such that C_λ is a ground clause, some member of M is a member of C_λ.

1.26.1 is an immediate consequence of this lemma, for if every ground instance of every member of $R(S,\omega)$ contains a member of M, then surely $\square \notin R(S,\omega)$.

By the definition of model, every ground instance of every member of S contains a member of M. Since $R(S,0) = S$, the lemma holds for the case $n = 0$.

Suppose 1.26.2 holds for $l < n+1$, and C1 and C2 are members of R(S,n). By the induction hypothesis, every ground instance of C1 or C2 contains a member of M. Let C1' and C2' be factors of C1 and C2. Trivially, any ground instance of C1' or C2' contains a member of M. Suppose that $L1 \in C1'$, $L2 \in C2'$, the atoms of L1 and L2 have most general unifier σ, and L1 and L2 have opposite signs. Let $R = ((C1' - [L1]) \cup (C2' - [L2]))_\sigma$. Let λ be any substitution such that R_λ is a ground clause. Then $(C1' - [L1])_{\sigma\lambda}$ or $(C2' - [L2])_{\sigma\lambda}$, hence R_λ, contains a member of M. QED.

We now suppose that S does not admit a model and proceed to show that for some l, $\square \in R(S,l)$. The proof is based upon the Model Theorem and

1.26.3 The Ground Case

If S is a finite, unsatisfiable set of ground clauses, then $\square \in R(S, \omega)$.

In proving the Ground Case we make use of a particularly transparent form of induction first presented in Anderson & Bledsoe (3). We shall again make use of it in Chapter 3 to prove the ground case for locking. We need the definition of

1.26.4 The K Parameter

If S is a finite set of clauses, then K(S) is the difference between the sum of the cardinalities of the members of S and the cardinality of S. (More simply, K(S) is the number of literals minus the number of clauses. But note we count distinct occurrences of literals!!!)

We now prove 1.26.3.

Suppose that S is a finite, unsatisfiable collection of ground clauses. If $K(S) < 1$, then either $\square \in S$ or each member of S is a singleton. If $\square \in S$, then $\square \in R(S,0)$.

So suppose that each member of S is a singleton. Since S is unsatisfiable, there must exist some atom L such that $[L] \in S$ and $[\neg L] \in S$. But then $\square \in R(S,1)$.

Now assume that k is an integer and for all j such that $j < k+1$, if S' is an unsatisfiable collection of ground clauses and $K(S') = j$ then for some n, $\square \in R(S',n)$. Suppose

further that K(S) = k+1. Then either $\square \in S$ or S contains a clause that is not a singleton. If $\square \in S$, $\square \in R(S,0)$. So suppose there is a clause C in S with at least two members, one of which is the literal L. Let A = C - [L]. Let S* = (S -[C]) U[A]. Clearly, then, K(S*) = k. Furthermore, S* is unsatisfiable since any model of S* is a model of S. Hence for some integer J0, $\square \in R(S^*, J0)$. But this implies that either $\square \in R(S, J0)$ or $[L] \in R(S, J0)$. Let S** = (S -[C]) U [[L]]. K(S**) < k+1. Furthermore, S** is unsatisfiable. Therefore, for some integer J1, $\square \in R(S^{**}, J1)$. Therefore, $\square \in R(S, J0 + J1)$. This establishes 1.26.3.

Digression on Lifting and Instantiating

After the following digression, the proof of 1.26 continues at 1.26.5

By the Model Theorem, a collection S of clauses is unsatisfiable if and only if there exists a sequence $\sigma 1, \sigma 2, \ldots, \sigma n$ of substitutions such that for each i, S_{σ_i} is a collection of ground clauses and

$S_{\sigma_1} \cup S_{\sigma_2} \cup \ldots \cup S_{\sigma_n}$

is unsatisfiable.

If G is a finite collection of ground clauses, then for some integer n, R(G,n) = R(G, ω). Hence to decide whether S is satisfiable, we might take some enumeration of all substitutions σ such that S_σ is a collection of ground

clauses, and for each i compute $R(S_{\sigma_1} \cup S_{\sigma_2} \cup \ldots \cup S_{\sigma_i}, \omega)$ until $\square$ is generated. Although Davis, Prawitz, and Wang used methods more efficient than resolution for checking inconsistencies, their programs attempted essentially this computation.

It is, however, an unsatisfactory approach to proving difficult theorems.

It was J. A. Robinson(16) who discovered that it is possible to avoid taking resolvents of ground instances of S by taking "general" resolvents of S. He describes resolution as a "demon" that, if possible, finds just the substitutions $\sigma 1, \sigma 2, \ldots, \sigma n$ for which we are looking. In fact, if $\sigma 1, \sigma 2, \ldots, \sigma n$ is a shortest sequence of substitutions such that $S_{\sigma_1} \cup S_{\sigma_2} \cup \ldots \cup S_{\sigma_n}$ is unsatisfiable and J0 is the smallest integer such that $\square \in R(S_{\sigma_1} \cup S_{\sigma_2} \cup \ldots \cup S_{\sigma_n}, J0)$, then $\square \in R(S, J0)$.

The kernel of Robinson's proof is

1.26.5 The Lifting Lemma

If C1 and C2 are clauses and σ is a substitution and R is a resolvent of $C1_\sigma$ and $C2_\sigma$, then there exist factors C1' and C2' of C1 and C2, a resolvent R' of C1' and C2', and a substitution λ such that $R = R'_\lambda$.

We now conclude the proof of the other half of the Resolution Theorem.

1.26.6 If S is a finite, unsatisfiable collection of clauses, then $\square \in R(S, \omega)$.

Suppose S is a finite, unsatisfiable collection of clauses. By the Model Theorem, there exist $\sigma_1, \sigma_2, \ldots, \sigma_n$ such that S_{σ_i} is a set of ground clauses and $S_{\sigma_1} \cup S_{\sigma_2} \cup \ldots S_{\sigma_n}$ is unsatisfiable.

By induction on i, it is clear that if $C \in R(S_{\sigma_1} \cup S_{\sigma_2} \cup \ldots \cup S_{\sigma_n}, i)$, then there exist σ and $C' \in R(S,i)$ such that $C = C'_{\sigma}$. If $i = 0$, this is trivial. By induction, if C and D are in $R(S_{\sigma_1} \cup S_{\sigma_2} \cup \ldots \cup S_{\sigma_n}, i)$ there exist $C' \in R(S,i)$, $D' \in R(S,i)$, and λ such that $C = C'_{\lambda}$ and $D = D'_{\lambda}$. Hence any resolvent R of C and D is an instance of a resolvent R' of factors of C' and D' by the Lifting Lemma.

Since $S_{\sigma_1} \cup S_{\sigma_2} \cup \ldots S_{\sigma_n}$ is unsatisfiable, for some J, $\square \in R(S_{\sigma_1} \cup S_{\sigma_2} \cup \ldots \cup S_{\sigma_n}, J)$. Therefore, $\square$ is an instance of some member of R(S,J). Therefore $\square \in R(S,J)$.

2. The Concept of Locking

The use of unification permits a vast increase in the efficiency of proof procedures based on Herbrand's Theorem. By avoiding the individual examination of ground instantiations of the original clauses, a proof procedure needs less time and storage. Nevertheless, there is still no proof procedure that solves many and varied hard problems in mathematics on computers. The explosion in the number of clauses generated b resolution, in particular, and the corresponding increase in the time necessary to examine the clauses still provide insuperable problems for any present computing facility. One of the most obvious causes[6] of this problem is that any unsatisfiable collection of clauses is likely to admit a large number of refutations, and resolution will proceed to find them all. Any technique that provides a method for significantly reducing the number of resolvents, without significantly increasing the depth of the shortest refutation, is of interest.

We present in this thesis a restriction of resolution that does significantly reduce the number of resolvents generated. This restriction is based on the simple idea that if one were to resolve on only one literal in each clause, the number of resolvents would certainly be reduced. Ideas in this general direction have been pursued in the works of Reynolds, Slagle, Kowalski, Hayes, Loveland, and Reiter[7].

One wishes he might choose any literal in each of the original clauses, resolve upon just those literals, choose one literal in each of the resolvents, resolve again, choose again, and so on. However, it is easy to show that this procedure is incomplete. For consider the following unsatisfiable set of ground clauses:

$$[[\ A\quad B\ ]\ [\neg B\quad A\ ]\ [B\quad \neg A\ ]\ [\neg A\quad \neg B]]\ .$$

Suppose we choose to resolve only upon the first literal in each clause. We get the following resolvents:

$$[\ B\ \neg B\ ]\ [\ A\ \neg A]\ .$$

Again suppose we choose to resolve upon the first literals only. We then obtain the resolvents:

$$[\neg B\ A\ ]\ [\neg A\ \neg B\ ]\ .$$

If we again decide to resolve only upon the first literals, we can only produce duplicates of clauses we already have. If we continue to choose literals from these clauses as we chose before, we will never get any more new clauses, and certainly not $\square$.

A restricted version of this procedure is complete, however. In particular, in the ground case, it is true that one need only resolve on a particular literal in the clause (and it is easy to determine which one it is). The general case permits slightly less freedom.

The technique for deciding which literal to resolve upon is roughly as follows. Before beginning resolution, we

assign to each occurrence of each literal in each clause a positive integer. This may be done arbitrarily. Let us call the integer the "index" of the literal. Note that different occurrences of the same literal may be assigned different indices. Having made this assignation, we resolve clause against clause as in unrestricted resolution, but only upon the lowest indexed literal in each clause. The literals in the resolvents have "hereditary" indices.

For example, suppose we have the two clauses

$$[\underset{10}{(G\ y\ (a))}\quad \underset{1}{(G\ (f\ y)\ y)}]$$
$$[\neg \underset{11}{(G\ w\ y)}\ \neg \underset{6}{(G\ y\ (a))}].$$

(The number printed below a literal is its index.) The only literal in the first clause that may be resolved upon is (G (f y) y) (because 1 < 10). The only literal in the second clause that may be resolved upon is $\neg \underset{6}{(G\ y\ (a))}$ (because 6 < 11). There is only one lock-resolvent of these two clauses, viz.

$$[\underset{10}{(G\ (a)(a))}\quad \neg \underset{11}{(G\ w\ (f\ (a)))}]$$

as opposed to four ordinary resolvents.

There are two matters we need to clarify. Strictly speaking, we may not refer to "the" lowest indexed literal of a clause. Even if we index different occurrences of literals differently at the beginning, two distinct literals, descendent from the same original literal, may appear in a resolvent. Therefore, we resolve on all the literals of lowest index in a

clause. Thus if the clause

$$[\ (G_3\ x\ y)\quad (G_3\ x\ (f\ x))\quad (G_{10}\ (f\ x)\ (f\ x))\ ]$$

should arise, we would resolve on both literals of index 3.

The second matter is that a literal may appear in a resolvent as a descendent from two literals, one in each of the clauses resolved. If these two literals have different indices, we assign to the descendent the lower of the two indices. Thus, the resolvent of

$$[\ (G_3\ (f\ x)\ (f\ x))\quad (G_2\ x\ x)\ ]$$
$$[\ (G_4\ (f\ x)\ (f\ x))\ \neg\ (G_1\ x\ x)\ ]$$

is

$$[\ (G_3\ (f\ x)\ (f\ x))].$$

Because we think of the literals in a clause that are not of lowest index as "locked-out" of the resolving process, we refer to our method of resolution as "locking."

2.1 Locked

If S is a set of clauses, then S is said to be "locked" if and only if for each clause C in S and for each literal L in C, there exists a positive integer i such that i is the index of L.

2.2 Lowest Index

If C is a clause and L is a literal in C and the index of L is less than or equal to the index of every literal in C, then L is said to be "of lowest index" in C.

In Chapters 3 and 4 we precisely define the concept of "lock-resolvent" and prove the completeness of lock-resolution.

3. The Completeness of Ground Lock-resolution

The usual method for proving the completeness of a version of resolution has two parts. First, one proves the "ground case", the case in which no variables (and consequently no substitutions) occur. Then one "lifts" to the general case. We follow the method here. This chapter contains the definition of a "ground lock-resolvent" and the completeness proof for ground lock-resolution. Chapter 4 treats the general case.

3.1 Ground lock-resolvents

Suppose C1 and C2 are members of a locked set of ground clauses.

Suppose further that L1 is a literal of lowest index in C1 and L2 is a literal of lowest index in C2.

Finally, suppose that L1 is an atom and L2 is $\neg$ L1.

Now let R = (C1 - [L1]) U (C2 -[L2]). If a literal L in R is a member of both (C1 -[L1]) and (C2 -[L2]), let the index of L in R be the least of the index of L in (C1 -[L1]) and the index of L in (C2 -[L2]). Otherwise, let the index of a literal in R be the same as it is in (C1 - [L1]) or (C2 - [L2]).

Under these suppositions and allowances, R is said to be a "ground lock-resolvent" of C1 and C2.

3.2 Ground Lock-resolution

Suppose S is a locked set of ground clauses. Let L(S,0) be S, and for each positive integer i+1 let L(S,i+1) be the union of L(S,i) and the set of all ground lock-resolvents of members of L(S,i). Let L(S,ω) be $\bigcup_{i \in \omega} L(S,i)$.

The kernel of this chapter is the following lemma:

3.3 Lemma

Suppose S is a finite, locked set of ground clauses.

Suppose also that m is the least integer such that some literal in some clause of S has index m and every literal in every non-unit clause of S has index less than or equal to m.

Finally, suppose C is a clause in S, L is a literal in C, and the index of L is m.

Let S* = (S - [C]) U [(C - [L])].

Then, for each non-negative integer n and for each clause D in L(S*,n) there exists a clause D' in L(S,n) such that either

1) D and D' are identical and have the same locking, or

2) D = D' - [L], L has index m in D', and each member of D has the same index in D and D'.

Proof:

We prove this lemma by induction on n. If n = 0, the lemma is trivial. We associate (C - [L]) in S* with C in S; and we associate every other member of S* with itself.

Suppose the lemma holds for some non-negative integer j.

There are two ways a clause D can be a member of L(S*,j+1). If D ε L(S*,j), then D ε L(S*,j+1). But in this case, by the induction hypothesis, there exists a D' ε L(S,j) with the right properties and D' ε L(S,j+1).

On the other hand, D may be a ground lock-resolvent of two clauses D1 and D2 in L(S*,j). By the induction hypothesis, there exist D1' and D2' in L(S,j) with the right properties. Suppose that D is the result of resolving on L1 in D1 and L2 in D2. Then L1 ε D1' and L2 ε D2'. Furthermore, L1 is of lowest index in D1' and L2 is of lowest index in D2'; this follows from the induction hypothesis and the fact that m is greater than or equal to the index of any literal in any non-unit clause in L(S, ω). Therefore there exists a ground lock-resolvent D' of D1' and D2' on L1 and L2; and thus D' ε L(S,j+1).

It remains to be shown that D and D' have the correct relationship. There are four cases to consider:

1) If D1 and D1' are identical and have the same locking and D2 and D2' are identical and have the same locking, then D and D' are identical and have the same locking.

2) If D1 = D1' - [L], L has index m in D' and each member of D1 has the same index in D1 and D1', and D2 = D2' - [L] (etc.), then D = D' -[L], L has index m in D', and each member of D has the same index in D and D'. The main point is that since L has index m in both D1' and D2', it will have index m in D'.

3) If D1 and D1' are identical and have the same locking, but D2 = D2' - [L], L has index m in D2', and each member of D2 has the same index in D2 and D2', then

a) D and D' are identical and have the same locking if and only if L occurs in D1 and is not L1, whereas

b) D = D' - [L], L has index m in D', and each member of D has the same index in D and D' if and only if L does not occur in D1 or L is L1.

The main point here is that if L is in D1 and is not resolved upon, then L will appear in D' and D with the same index it had in D1.

4) case 3, mutatis mutandis.

This concludes the proof of the lemma.

Let the reader recall the definitions of "satisfiable" (1.22) and the "K-parameter"(1.26.4).

3.4 The Completeness of Ground Lock-resolution

If S is an unsatisfiable, finite, and locked collection of ground clauses, then $\square \in L(S, \omega)$.

The proof is by induction on K(S). Let S be an unsatisfiable, finite, and locked collection of ground clauses.

If K(S) < 1, then either $\square \in S$ (in which case $\square \in L(S,0)$), or each member of S is a unit clause. In the latter case, there exists an atom A such that $[A] \in S$ and $[\neg A] \in S$. For otherwise, S would be satisfiable. Clearly,

$\square \in L(S,1)$.

Suppose for some non-negative integer, i, that if S' is an unsatisfiable, finite, and locked set of clauses and $K(S') < i+1$, then $\square \in L(S',\omega)$. Suppose further that $K(S) = i+1$. Either $\square \in S$ or there is a clause in S with at least two literals.

We now apply Lemma 3.3. Let m be the least integer such that some literal in some clause of S has index m and every literal in every non-unit clause of S has index less than or equal to m. There exists a clause C in S that is not a unit clause and a literal L in C whose index is m. Let $S^* = (S - [C]) \cup [(C - [L])]$. Clearly, then, $K(S^*) = i$. Furthermore, S* is unsatisfiable since any model of S* is a model of S. By the induction hypothesis, for some j_0, $\square \in L(S,j_0)$. By Lemma 3.3 there exists a clause D' in $L(S,j_0)$ such that either

$\square = D'$, or

$\square = D' - [L]$ and L has index m in D'.

If $\square \in L(S,j_0)$ the proof is complete. So suppose that $[L] \in L(S,j_0)$. Let $S^{**} = (S - [C]) \cup [[L]]$. Clearly, then, $K(S^{**}) < i+1$. Furthermore, S** is unsatisfiable since any model of S** is a model of S. By the induction hypothesis, for some j_1, $\square \in L(S^{**},j_1)$. Therefore, $\square \in L(S,j_0+j_1)$. QED.

4. The Completeness of Lock-resolution

In this chapter we define "lock-resolvent" and prove the completeness of lock-resolution. The definition and proof are fundamentally based on the seminal concept of unification. Because we pay close attention to the details of indices, our definition and proof differ widely in the letter (but hardly in the spirit) from the definition and proof of the completeness of resolution given in Chapter 1.

Except for the restriction on literals to be resolved upon, the major difference between lock-resolution and unrestricted resolution is that in lock-resolution we are interested in factors only at the beginning. Instead of factoring each resolvent, we have built the essence of factoring into the definition of lock-resolvent. This technique is not new[8] and has been called "merge-resolution." Although the definition of lock-resolvent is more complicated than it would be if we relied on factoring, we believe it results in a clearer completeness proof.

The idea behind the definition of the lock-resolvent of two clauses C and D is simple. Basically, we take a literal A1 of lowest index in C and a literal B1 of lowest index in D and look for a most general substitution σ_1 such that $\neg A1_{\sigma_1} = B1_{\sigma_1}$. To avoid factoring, we also look for literals Ai in C and literals Bi in D and an extension σ of σ_1 that unifies each

Ai with Bi. That is, we try to merge Ai with Bi. We then take $R = ((C - [A1]) \cup (D - [B1]))_\sigma$ as a resolvent of C and D after assigning appropriate indices to the literals in R.

The precise definition is:

4.1 Lock Resolvent

Suppose that

1) C and D are members of a locked collection of clauses,

2) n is a positive integer, (we intend that n may be 1)

3) A1, A2, ..., An is a sequence of distinct literals in C, A1 is a literal of lowest index in C, and A1 is an atom,

4) B1, B2, ..., Bn is a sequence of distinct literals in D, B1 is a literal of lowest index in D, B1 is not an atom (i.e. B1 is the negative of an atom), and finally

5) σ is a most general simultaneous unifier of $[[\neg A1\ B1][A2\ B2]..[An\ Bn]]$, σ does not unify two distinct literals of $(C - [A1])$, and σ does not unify two distinct literals of $(D - [B1])$.

Let R be $((C - [A1]) \cup (D - [B1]))_\sigma$.

If there exist L1 in $(C - [A1])$ and L2 in $(D - [B1])$ such that $L1_\sigma = L2_\sigma$, then let the index of $L1_\sigma$ in R be the least of the index of L1 in $(C - [A1])$ and the index of L2 in $(D - [B1])$. Otherwise, if L is a literal in $(C - [A1])$ or $(D - [B1])$, let the index of L_σ in R be the index of L in $(C - [A1])$ or $(D - [B1])$.

Under these suppositions and allowances, R is said to be a "lock-resolvent" of C and D. [2]

We now prove the basis of the completeness proof:

4.2 The Lifting Lemma for Locking

Suppose that

1) C and D are members of a locked collection of clauses,

2) τ is a substitution, C_τ and D_τ are ground clauses, and τ does not unify two distinct members of C nor does it unify two distinct members of D,

3) if $L \in C$, then the index of L_τ in C_τ is the index of L in C; and if $L \in D$, then the index of L_τ in D_τ is the index of L in D, and finally suppose

4) R is a ground lock-resolvent of C_τ and D_τ.

Then there exist a lock-resolvent R' of C and D and a substitution λ such that

(i) $R'_\lambda = R$,

(ii) λ does not unify two distinct members of R', and

(iii) if $L \in R'$, then the index of L_λ in R is the index of L in R'.

Proof:

Let us make the suppositions of the Lifting Lemma for Locking. There exist a sequence a1, a2, ..., an of distinct

members of C_τ and a sequence b1, b2,..., bn of distinct members of D_τ such that

1) a1 is an atom and a literal of lowest index in C_τ,

2) b1 is not an atom but is a literal of lowest index in D_τ,

3) $\neg$ a1 = b1,

4) R = $(C_\tau - [a1]) \cup (D_\tau - [b1])$,

5) for i > 1, $a_i = b_i$, and

6) if L $\in (C_\tau - [a1])$ and L $\in (D_\tau - [b1])$, then for some i > 1, L = $a_i = b_i$.

These points are all easy consequences of the fact that R is a resolvent of C_τ and D_τ.

Since τ does not unify distinct members of C or D, there exist a sequence A1, A2, ..., An of distinct members of C and a sequence B1, B2, ..., Bn of distinct members of D such that for each i, $Ai_\tau = a_i$, $Bi_\tau = b_i$, the index of Ai in C is the index of a_i in C_τ, and the index of Bi in D is the index of b_i in D_τ.

Since τ is a simultaneous unifier of [[$\neg$A1 B1][A2 B2]...[An Bn]], there exists a most general simultaneous unifier σ of [[$\neg$A1 B1][A2 B2]...[An Bn]], and a substitution λ such that $\sigma\lambda = \tau$. Because τ does not unify two members of C or of D, neither does σ. Therefore, $R' = ((C - [A1]) \cup (D - [B1]))_\sigma$ is a lock-resolvent of C and D. But

$$R = (C_\tau - [a1]) \cup (D_\tau - [b1])$$
$$= (C - [A1])_\tau \cup (D - [B1])_\tau$$
$$= ((C - [A1]) \cup (D - [B1]))_\tau$$
$$= ((C - [A1]) \cup (D - [B1]))_{\sigma\lambda}$$
$$= R'_\lambda .$$

This establishes the first part of the conclusion of the Lifting Lemma.

We now show that λ does not unify two members of R'. Suppose $k1 \in R'$, $k2 \in R'$, $k1 \neq k2$, but $k1_\lambda = k2_\lambda$. From these suppositions we shall derive the contradiction $k1 = k2$. There exist $K1 \in (C - [A1]) \cup (D - [B1])$ and $K2 \in (C - [A1]) \cup (D - [B1])$ such that $K1_\sigma = k1$ and $K2_\sigma = k2$. Since σ does not unify two members of $(C - [A1])$ or of $(D - [B1])$, we may assume without loss of generality that $K1 \in (C - [A1])$ and $K2 \in (D - [B1])$. By (6) above, for some $i > 1$, $k1_\lambda = a_i = b_i = k2_\lambda$. But we have $Ai_\tau = a_i$ and $Bi_\tau = b_i$. Since $Ai_\tau = K1_\tau$, $Ai = K1$. Furthermore, $Bi = K2$. But $k1 = Ai_\sigma = Bi_\sigma = k2$, contradicting the assumption that $k1 \neq k2$. Thus we have established the second part of the conclusion of the Lifting Lemma.

Finally, we must show that if $L \in R'$, then the index of L in R' is the index of L_λ in R. There are two cases to consider.

1) If for some $i > 1$, $L = Ai_\sigma = Bi_\sigma$, then the index of L in R' is the least of the index of Ai in C and the index of

Bi in D. For since σ does not unify members of $(C - [A1])$, if $L' \in (C - [A1])$ and $L'_\sigma = L_\sigma$, then $L' = Ai$. And similarly, if $L' \in (D - [B1])$ and $L'_\sigma = L_\sigma$, then $L' = Bi$. In this case, however, the index of L in R' is the least of the index of a_i in $(C_\tau - [a1])$ and the index of b_i in $(D_\tau - [b1])$.

2) If for no $i > 1$ is $L = Ai_\sigma = Bi_\sigma$, then there do not exist L1 in $(C - [A1])$ and L2 in $(D - [B1])$ such that $L = L1_\sigma = L2_\sigma$. Hence, there exists precisely one L' in $(C - [A1])$ or $(D - [B1])$ such that $L'_\sigma = L_\sigma$. If $L' \in (C - [A1])$, then the index of L' in $(C - [A1])$ is the index of L in R', the index of L'_τ in C, and the index of L_λ in R. And similarly, if $L' \in (D - [B1])$.

This concludes the proof of the Lifting Lemma for Locking. Before proving the completeness of lock-resolution, we need

4.3 Lemma

Suppose S is a finite, locked, fully-factored, unsatisfiable collection of clauses. Then there exists a finite, locked, unsatisfiable collection I of ground clauses with the following property:

For each member D of I, there exists a member C of S and a substitution λ such that

1) $D = C_\lambda$

2) λ does not unify two distinct members of C,

3) if $L \varepsilon C$, then L has the same index in C as L_λ has in D.

Proof:

Suppose S is a finite, fully-factored, unsatisfiable collection of clauses. By the Model Theorem, there exists a sequence $\sigma_1, \sigma_2, \ldots, \sigma_n$ of substitutions such that for each i S_{σ_i} is a collection of ground clauses and $S_{\sigma_1} \cup S_{\sigma_2} \ldots \cup S_{\sigma_n}$ is unsatisfiable. Let I be $S_{\sigma_1} \cup S_{\sigma_2} \ldots S_{\sigma_n}$. By the fully-factored theorem, for each D in I there exists a C in S and a substitution λ such that $D = C_\lambda$ and λ does not unify two members of D. We now only need to "rig" the indices of the literals in the members of I. For each clause D in I, let D' be a member of S and λ a substitution such that $D'_\lambda = D$ and λ does not unify two members of D'. If $L \varepsilon D'$, let the index of L_λ in D be the index of L in D'. QED.

We are now prepared for the completeness theorem, so we define:

4.5 Lock Resolution

If S is a set of locked clauses, let L(S,0) be S, and for each positive integer i+1, let L(S,i+1) be the union of L(S,i) with the set of all lock-resolvent of members of L(S,i). Let $L(S,\omega)$ be $\bigcup_{i \varepsilon \omega} L(S,i)$.

4.6 The Completeness Theorem for Lock-resolution

If S is a finite, fully-factored, unsatisfiable collection of clauses, then $\square \in L(S, \omega)$.

Suppose that S is a finite, fully-factored, unsatisfiable, and locked collection of clauses. Let I be a finite, locked, unsatisfiable collection of ground clauses with the following property:

For each member D of I, there exists a member C of S and a substitution λ such that

1) $D = C_\lambda$

2) λ does not unify two distinct members of C

3) if $L \in C$, then L has the same index in C as L_λ has in D.

There exists such an I by Lemma 4.3. We now prove by induction on n

4.7 Lemma

If n is a non-negative integer and $R \in L(I,n)$, there exists an $R' \in L(S,n)$ and a substitution λ such that $R = R'_\lambda$, λ does not unify two member of R', and if $L \in R'$, then the index of L in R' is the index of L_λ in R.

In the case n = 0, this follows immediately from the defining properties of I.

Suppose that Lemma 4.7 holds for n and that $R \in L(I,n+1)$. There are two ways R can be a member of L(I,n+1). If

$R \in L(I,n)$, then $R \in L(I,n+1)$. In this case, however, there exists by the induction hypothesis an $R' \in L(S,n)$ and a λ with the right properties, and $R' \in L(S,n+1)$.

On the other hand, R may be a ground lock-resolvent of two clauses C and D in $L(I,n)$. By the induction hypothesis there exist C' and D' in $L(S,n)$ and substitutions λ_1 and λ_2 such that $C'_{\lambda_1} = C$, $D'_{\lambda_2} = D$, λ_1 does not unify two distinct members of C', and λ_2 does not unify two distinct members of D'. Furthermore, if $L \in C'$ then the index of L in C' is the index of L_{λ_1} in C, and similarly if $L \in D'$. Since we provide that no two clauses being resolved have variables in common, there exists a substitution τ, namely $\lambda_1\lambda_2$, such that $C'_{\tau} = C$, $D'_{\tau} = D$, and τ does not unify two distinct members of C' or of D'. By the Lifting Lemma for Locking, there exists a resolvent R' of C' and D' and a substitution λ such that $R'_{\lambda} = R$, λ does not unify two distinct members of R', and if $L \in R'$, then the index of L_{λ} in R is the index of L in R'. This completes the induction step, and consequently the proof of Lemma 4.7.

Since I is unsatisfiable, by the Ground Completeness of Lock-resolution, for some n, $\square \in L(I,n)$. Therefore, $\square \in L(S,n)$. QED.

5.The Incompatibilities and Idiosyncracies of Locking

5.1 Locking is incompatible with the elimination of tautologies. In the literature on resolution, a "tautology" is defined to be a clause that contains two complementary literals. In every other form of resolution of which we are aware, □ may be derived from an unsatisfiable set even if one never resolves a tautology against any clause. The following locked set of clauses fairly approximates the restrictions which we wished to apply in the first round of the example in Chapter 2.

1 [[A_1 B_2]

2 [¬ B_3 A_4]

3 [B_5 ¬ A_6]

4 [¬ A_7 ¬ B_8]

Notice that every possible lock-resolvent in the first round is a tautology. Clearly, eliminating them would prevent □ from arising. We present a derivation of □ for this set of clauses.

==========

5 [B_2 ¬ B_8] 1,4

6 [A_4 ¬ A_6] 2,3

==========

7 [A_4 ¬ B_8] 5,2

8 [¬ A_6 ¬ B_8] 4,6

==========

9 [¬ B] 7,8

10 [¬ A_6 ¬ B_8] 6,8

==========

11 [¬ A] 9,3

==========

12 [B] 11,1

==========

13 □ 9,12

The key difference between lock-resolution and the incomplete strategy suggested for the example in Chapter 2 comes to light in clause 7. Notice that lock-resolution forces clause 7 to be locked differently from clause 2. (cf. example 7.2 for another locking refutation of the same set of clauses.)

5.2 Locking is incompatible with linear format. There is a powerful restriction of resolution discovered by Loveland(10) among others, called "linear format". If S is a set of clauses, C1, C2, C3, ... Cn is a sequence of clauses, C1 ε S, and for each i > 0, Ci+1 is a resolvent of Ci and a member of S or some Cj where j < i, then C1, C2, ... Cn is called a linear derivation of Cn from S. There is a linear derivation of □ from any unsatisfiable collection of clauses. This is not the case for locking. Consider the following set of clauses:

[[A_1 P_2]

[¬ A]

$$[\quad \underset{3}{B} \quad \neg \underset{4}{P} \quad]$$
$$[\ \neg B \quad]]$$

It is possible to derive [P]on one branch, and [$\neg$ P]on another branch, but these two clauses do not admit lock-resolvents with either "ancestors" or "input parents".

5.3 Locking is incompatible with the ancestor restriction of Loveland(10). If S is an unsatisfiable collection of clauses, [] may be deduced from S if one accepts only those resolvents R of two clauses C and D such that one of C, D is in S or some instance of R is a subset of some instance of C or D. This is not the case for locking, however; consider the following unsatisfiable collection of clauses:

1 $[[\ \underset{1}{R}\ \underset{2}{S}\ \neg \underset{3}{P}\ ]$

2 $[\ \neg R\ ]$

3 $[\ P\ ]$

4 $[\ \underset{4}{A}\ \neg \underset{5}{S}\ \underset{6}{F}\ ]$

5 $[\ \neg A\ ]$

6 $[\ \neg F\ ]$

==========

In the first round we obtain

7 $[\ \underset{2}{S}\ \neg \underset{3}{P}\ ]$ 1,2

8 $[\ \neg \underset{5}{S}\ \underset{6}{F}\ ]$ 4,5

But neither 7 nor 8 lock-resolve with any of the original clauses. The lock-resolvent $[\ \neg \underset{3}{P}\ \underset{6}{F}\ ]$of 7 and 8 is not a subset

of 7 or 8. Hence, if we accept the ancestor restriction, no resolvents will appear in round 2.

5.4 Locking is incompatible with the merging restriction of Andrews(4). If S is an unsatisfiable collection of clauses, $\square$ may be be deduced from S if one accepts only those resolvents R of two clauses C and D such that one of C,D is in S or one of the literals resolved upon is a merged literal. This is not the case for locking, however, as the counter-example to linear-format above reveals.

5.5 Locking is incompatible with a set of support restriction. If S is an unsatisfiable collection of clauses but every proper subset of S is satisfiable, and $C \in S$, it is possible to generate $\square$ by resolving just C against S, then those resolvents against themselves and S, then the new resolvents against themselves, S, and the previous resolvents, and so on. But consider the following clauses:

$$[\ \neg Q_1\ \neg P_2\ ]$$
$$[\ Q\ ]$$
$$[\ P\ ]$$

Notice that the third clause admits no resolvent on the first round.

5.6 The method of P1 resolution invented by Robinson(17) is an easy consequence of the completeness of lock-resolution. A P1 resolvent is a clause formed by resolving two clauses together, one of which contains only negative literals. This effect can be achieved in lock-resolution by simply assigning the index 1 to all positive literals and the index 2 to all negative literals. No negative literal can be lock-resolved upon unless all the literals in the clause containing the literal are negative.

5.7 Locking is compatible with Anderson's restriction on merging [9]. Recall that when we resolve two "general" clauses C and D we not only try to unify $\neg$ A1 and B1 (for some A1 ε C and B1 ε D), but also try to simultaneously unify some other Ai's ε C with Bi's ε D. This has the beneficial effect of sometimes "shrinking" the resolvent by the merging of literals; but it also requires a significant amount of computation. Surprisingly, it is possible to restrict the search to just those Ai's and Bi's that have the same index. To be precise we may:

> Add to supposition 4 of definition 4.1 the condition "and for each i > 1, the index of Ai in C is the index of Bi in D"

without the loss of completeness.

If we make sure before beginning resolution that

literals in different clauses and in the same clause have different indices, this result tells us that we need only merge literals whose origin is the same. To some extent this strikes us as intuitively reasonable; it means that the only essential function of merging is to keep a literal from getting in its own way.

5.8 Several authors have discovered restrictions of resolution based on resolving only some of the literals in a clause.

Reynolds [10] orders the predicate symbols before beginning resolution; then in one of the clauses being resolved he resolves only on literals with highest predicate letter. Slagle (18) combines this restriction with maximal clash resolution.

Kowalski and Hays (8) redefine an A-ordering (a concept introduced by Slagle) to be a total ordering of some of the literals. Their A-ordering includes the requirements that (1) if $A < A'$, then for all substitutions σ $A_\sigma < A'_\sigma$, (2) if A is a variant of A', then $A \leq A'$, and (3) $\neg A \leq A \leq \neg A$. They then demand that neither of the literals resolved upon be less than any other literal in either of the clauses being resolved.

Reiter (15) follows Kowalski and Hays in defining an A-ordering as a partial ordering on all the literals. This

ordering is induced by a total ordering of all the ground atoms and by instantiation. Using a linear format, he requires that in one of the clauses being resolved the literal resolved upon be maximal.

Kowalski & Kuehner(9), Loveland(11), and Reiter(15) have found linear format and other restrictions compatible with resolving on exactly one literal in a clause being resolved with an original clause. (They do not restrict the literal to be resolved upon in original clauses.)

The essential difference between locking and all these versions of resolution is that in lock-resolution "most" of the time one need only resolve on one literal in each clause. If one simply simply assigns different indices to the literals in the original clauses, one never resolves on more than one literal in any clause unless the clause contains two distinct descendents of the same original literal, and these literals have lowest index in the clause. In particular, one never resolves on more than one literal in an original clause. Thus lock resolution differs from the versions mentioned in the last two paragraphs because in those versions resolution is always permitted on any of the literals in an original clause. In comparison to the ordering of predicate letters or A-ordering, locking permits a greater discrimination between literals (and their complements, variants, and instances.) This comes to light in examples such as example 4 in the next chapter.

Ordering the predicate letters there has no effect whatsoever since there is only one. If one A-orders the literals and admits to the ordering the literal $(x < z)$ from the transitivity axiom, the A-ordering becomes completely trivial. We first became interested in locking clauses precisely because of examples like this, which arise in proving elementary theorems in analysis. Finally, locking may be distinguished from all versions of resolution by the number of known restrictions with which it is incompatible.

6. Some Examples of Lock-resolution

Example 1

Our first example of a lock-resolution proof is based on the example at the beginning of Chapter 1. Consider the following collection of locked clauses.

1 [[$(G_6\ y\ (a))$ $(G_1\ (f\ y)\ y)$]

2 [$(G_7\ y\ (a))$ $(G_2\ y\ (f\ y))$]

3 [$\neg (G_9\ w\ y)$ $(G_3\ (f\ y)\ y)$]

4 [$\neg (G_{10}\ w\ y)$ $(G_4\ y\ (f\ y))$]

5 [$\neg (G_{11}\ w\ y)$ $\neg (G_8\ y\ (a))$]]

==========

In the first round there are only two possible matches (as opposed to twenty for unrestricted resolution).

6 [$(G_6\ (a)(a))$ $\neg (G_{11}\ w\ (f(a)))$] 1,5

7 [$\neg (G_9\ w\ (a))$ $\neg (G_{11}\ w'\ (f\ (a)))$] 3,5

==========

In the second round we have

8 [$\neg (G_{11}\ w\ (a))$ $\neg (G_{11}\ w'\ (f\ (a)))$] 6,5

9 [$\neg (G_{11}\ w\ (f\ (a)))$ $\neg (G_{11}\ w'\ (f\ (a)))$] 6,7

10 [$\neg (G_{11}\ w\ (f\ a)))$] 6,7

11 [$(G_6\ (a)\ (a))$ $\neg (G_{11}\ w'\ (f\ (a)))$] 7,1

12 [$\neg (G_9\ w\ (a))$ $\neg (G_{11}\ w'\ (f\ (a)))$] 7,3

Since clause 10 subsumes every other resolvent, we may delete them all [11].

==========

The only resolvents of round 3 are those of clause 10 with the original 5 clauses, viz.

13 [$(G_6\ (f\ (a))\ (a))$] 10,1

14 [$(G_7\ (a)\ (a))$] 10,2

15 [$\neg(G_9\ w\ (f\ (a)))$] 10,3

16 [$\neg(G_{10}\ w\ (a))$] 10,4

==========

Among the resolvents of the next round is

17 □ 14,16

==========

We know of a program using linear-format, set of support, merging, splitting, and subsumption (but not locking) that generated more than 100 resolvents in proving this theorem.

Example 2

In Section 5.1 we presented one locking refutation for the "full set on two atoms." We present here another locking that leads to a less pathological derivation of □.

1 [A_1 B_4]

2 [A_2 $\neg B_6$]

3 [$\neg A_7$ B_3]

4 [$\neg A_8 \neg B_5$]

==========

5 [$\neg A$] 3,4

==========

6 [B] 1,5

7 [¬ B] 2,5

==========

□ 6,7

Notice there is only one possible match in the first round. This compares with the eight possible matches of unrestricted resolution.

Example 3

The following collection of clauses may not appear at first glance to be the clausal representation of a theorem of the form A -> A. [12]

1 [$0 \not< x$ $0 \not< y$ $0 <_1 (f\ x\ y)$]

2 [$0 \not< x$ $0 \not< y$ $(f\ x\ y) <_2 x$]

3 [$0 \not< x$ $0 \not< y$ $(f\ x\ y) <_3 y$]

4 [$0 < (a)$]

5 [$0 < (b)$]

6 [$0 \not< (f(a)(b))$ $(f(a)(b)) \not<_5 (a)$ $(f(a)(b)) \not< (b)$]

In the five rounds that resolution takes to derive □, the first four clauses lead to many spurious resolvents. But notice the locking proof.

7 [$0 \not< (a)$ $0 \not< (b)$ $(f(a)(b)) \not<_5 (a)$ $(f\ (a)(b)) <_6 (b)$]

6,1

8 [$0 \not< (a)$ $0 \not< (b)$ $(f\ (a)\ (b)) \not< (b)$]

7,2

9 [0 $\not<_{7}$ (a) 0 $\not<_{8}$ (b)] 8,3

10 [0 $\not<$ (b)] 9,4

11 □

Notice that not one spurious clause was generated.

Example 4

Our next example is more interesting and more difficult.[13]

1 [0 $\not<_{12}$ x 0 $\not<_{13}$ y 0 $<_{4}$ (f x y)]

2 [0 $\not<_{14}$ x 0 $\not<_{15}$ y (f_{5} x y) < x]

3 [0 $\not<_{16}$ x 0 $\not<_{17}$ y (f x y) $<_{6}$ y]

4 [x $\not<_{3}$ y y $\not<_{2}$ z x $<_{1}$ z]

5 [0 < (a)]

6 [0 < (b)]

7 [0 $\not<_{18}$ z (c) $<_{7}$ z (d) $<_{10}$ z]

8 [0 $\not<_{19}$ z (c) $<_{8}$ z (d) $\not<_{20}$ (b)]

9 [0 $\not<_{21}$ z (c) $\not<_{22}$ (a) (d) $<_{9}$ z]

10 [0 $\not<_{23}$ z (c) $\not<_{24}$ (a) (d) $\not<_{11}$ (b)]

==========

The lock-resolvents of round 1 are:

11 [(d) $\not<_{3}$ y y $\not<_{2}$ (b) (c) $\not<_{24}$ (a) 0 $\not<_{23}$ z] 10,4

12 [(d) $\not<_{3}$ 0 0 $\not<_{2}$ (b) (c) $\not<_{24}$ (a)] 10,4

13 [0 $\not<_{21}$ (b) (c) $\not<_{22}$ (a) 0 $\not<_{23}$ z] 10,9

14 [0 $\not<_{21}$ (b) (c) $\not<_{22}$ (a)] 10,9

==========

Among the resolvents of round 2 is:

15 [(c) $\not<$ (a)] 14,6

This clause subsumes all the other resolvents of round 1. Hence we may delete them and their descendents. We also delete clause 9 and clause 10.

==========

In round three the resolvents are:

16 [(c) $\not<_{3}$ y y $\not<_{2}$ (a)] 4,15

17 [0 $\not<_{18}$ (a) (d) $<_{10}$ (a)] 7,15

18 [0 $\not<_{19}$ (a) (d) $\not<_{20}$ (b)] 8,15

==========

Among the resolvents in round four are

19 [0 $\not<_{14}$ (a) 0 $\not<_{15}$ y (c) $\not<_{3}$ (f (a) y)] 16,2

20 [0 $\not<_{14}$ (a) (c) $\not<_{3}$ (f (a)(a))] 16,2

21 [0 $\not<_{16}$ x 0 $\not<_{17}$ (a) (c)$\not<_{3}$(f x (a))] 16,3

22 [0 $\not<_{16}$ (a) (c) $\not<_{3}$ (f (a) (a))] 16,3

23 [(c) $\not<$ 0] 16,5

24 [0 $\not<_{18}$ (a) (d) $<_{10}$ (a) (c) $\not<_{3}$ (c)] 16,7

25 [0 $\not<_{19}$ (a) (d) $\not<_{20}$ (b) (c) $\not<_{3}$ (c)] 16,8

26 [0 $\not<_{18}$ (a) (c) $\not<_{3}$ (d)] 16,17

27 [0 $\not<_{3}$ y y $\not<_{2}$ (a) (d) $\not<_{20}$ (b)] 18,4

28 [(d) $\not<$ (b)] 18,5

Because of 28 we can delete clauses 8, 18, 25, and 27. Thus after four rounds, we are concerned only with the clauses 1, 2, 3, 4, 5, 6, 7, 15, 16, 17, 19, 20, 21, 22, 23, 24, 26,

and 28. Let the number 18 be roughly compared to the 61 possible matches of unrestricted resolution in the first round alone. Having praised lock-resolution, we now admit that writing all lock-resolvents of the future rounds becomes tedious. We simply list the interesting clauses produced in each round.

==========

29 [(d) $\not<_3$ y y $\not<_2$ (b)] 28,4

30 [0 $\not<_{18}$ (f (a) y) (d) $<_{10}$ (f (a) y) 0 $\not<_{15}$ y 0 $\not<_{14}$ (a)] 19,7

==========

31 [0 $\not<_{16}$ x 0 $\not<_{17}$ (b) (d) $\not<_3$ (f x (b))] 29,3

==========

32 [0 $\not<_{18}$ (f (a) (b)) 0 $\not<_{14}$ (a) 0 $\not<_{15}$ (b)] 30,31

==========

33 [0 $\not<_{18}$ (f (a) (b)) 0 $\not<_{15}$ (b)] 32,5

==========

34 [0 $\not<$ (f (a) (b))] 33,6

==========

35 [0 $\not<_{12}$ (a) 0 $\not<_{13}$ (b)] 34,1

==========

36 [0 $\not<$ (b)] 35,5

==========

37 □ 36,6

==========

(If we added to lock-resolution the helpful strategy

of always resolving immediately with unit clauses (regardless of indices) whenever the resolvent produced subsumes the other clause with which we resolved, then this proof would be shortened by five rounds. In particular, the last six rounds would be shortened to two rounds.)

Example 5

The following theorem is due to Joyce Friedman.

$$\exists x \exists y \forall z \quad (((P\ x\ y) \rightarrow ((P\ x\ z) \leftrightarrow (Q\ y\ z))) \wedge ((P\ x\ y) \leftrightarrow ((P\ z\ z) \rightarrow (Q\ z\ z))) \rightarrow ((Q\ x\ y) \leftrightarrow (Q\ z\ z)))$$

In clausal form this is

1 $[\neg(P_{20}\ x\ y)\ \ \neg(P_{10}\ x\ (z\ x\ y))\ \ (Q_{1}\ y\ (z\ x\ y))]$

2 $[\neg(P_{21}\ x\ y)\ \ (P_{2}\ x\ (z\ x\ y))\ \neg(Q_{11}\ y\ (z\ x\ y))]$

3 $[\neg(P_{22}\ x\ y)\ \neg(P_{12}\ (z\ x\ y)\ (z\ x\ y))\ (Q_{3}\ (z\ x\ y)\ (z\ x\ y))]$

4 $[\ (P_{7}\ x\ y)\ \ (P_{4}\ (z\ x\ y)\ (z\ x\ y))\ ]$

5 $[\ (P_{5}\ x\ y)\ \neg(Q_{13}\ (z\ x\ y)(z\ x\ y))\ ]$

6 $[\ (Q_{8}\ x\ y)\ \ (Q_{6}\ (z\ x\ y)\ (z\ x\ y))\ ]$

7 $[\neg(Q_{23}\ x\ y)\ \ \neg(Q_{14}\ (z\ x\ y)\ (z\ x\ y))\ ]$

Here is a locking derivation of $\square$ from these clauses.

==========

8 $[\neg(P_{22}\ x\ y)\ \ \neg(P_{12}\ (z\ x\ y)\ (z\ x\ y)\ \neg(Q_{23}\ x\ y)\ ]$ 7,3

9 $[\neg(P_{22}\ x\ y)\ \neg(Q_{13}\ (z\ (z\ x\ y)\ (z\ x\ y))\ (z\ (z\ (x\ y))\ (z\ x\ y))))$
$\neg(Q_{23}\ x\ y)\ ]$ 5,8

10 $[\neg(P_{22}\ x\ y)\ \ (Q_{8}(z\ x\ y)\ (z\ x\ y))\neg(Q_{23}\ x\ y)\ ]$ 6,9

11 $[\neg(P_{22}\ x\ y)\ \neg(Q_{23}\ x\ y)\ ]$ 7,10

12 $[\neg(P_{21}\ x\ y)\ \neg(Q_{23}\ x\ (z\ x\ y))\ \neg(Q_{11}\ y\ (z\ x\ y))\ ]$ 11,2

13 $[\neg(P_{21}\ x\ x)\neg(Q_{11}\ x\ (z\ x\ x)\ )\ ]$ 11,2

14 $[\neg(P_{20}\ x\ x)\neg(P_{10}\ x\ (z\ x\ x))\ ]$ 1,13

15 $[\neg(P_{20}\ x\ x)\neg(Q_{13}\ (z\ x\ (z\ x\ x))\ (z\ x\ (z\ x\ x)))]$ 5,14

16 $[\neg(P_{20}\ x\ x)\cdot\ (Q_{8}\ x\ (z\ x\ x))\ ]$ 6,15

17 $[\neg(P\ x\ x)\ ]$ 13,16

18 $[\quad(P\ x\ y)\ ]$ 4,17

19 $\square$ 17,18

==========

There are only two clauses generated in the first round. (There are 27 possible matches in the first round for unrestricted resolution.) The two are clause 8 and the tautology

$$[\quad(Q_{8}\ x\ y)\quad\neg(Q_{23}\ x\ y)\quad] \qquad 6,7$$

Let us for the moment call a clause "spurious" if it is a tautology or is a superset of a clause previously generated (ignoring indices). We pointed out in 5.1 that in general if spurious clauses are eliminated then completeness is lost. Let us for the moment, however, eliminate spurious clauses. Then in the twelve rounds needed to derive $\square$ for this theorem fewer than twenty resolvents besides those listed above are produced (and fifteen of them are spurious). This leads us to hope for a powerful improvement of locking based on the elimination of

spurious clauses which meet some special condition. (For example, one might place a condition on the original locking.)

7. Hammer Locking

In unrestricted resolution, a set of clauses that contains a transitivity axiom such as

$$[\ (x \not< y)\ \ (y \not< z)\ \ (x < z)\ ]$$

presents an annoying problem. Naturally, any clause whatsoever containing a literal of the form $(a < b)$ or $(a \not< b)$ can be resolved against the transitivity axiom. The problem is that resolvents so formed will contain at least two literals, descendent from the axiom, which again match literals in the axiom.

For example, if the clause

$$[\ (a) \not< (b)\ ]$$

should arise, then after four more rounds we should obtain such spurious resolvents as

$$[\ ((a) \not< y)\ (y \not< y')\ (y' \not< y'')\ (y'' \not< y''')\ (y''' \not< (b))\ ]$$

We believe such resolvents are unnatural; in essence they amount to an application of the transitivity axiom to itself. In this chapter we show that such resolvents are unnecessary; and that they can be avoided by a very simple use of locking.

If S is an unsatisfiable collection of clauses including the transitivity axiom

$$[\ (\ x \underset{3}{\not<}\ y)\ (\ y \underset{2}{\not<}\ z)\ \ (\ x \underset{1}{<}\ z)\ ]$$

and every literal in each member of S except this axiom has an

index greater than 3, then there is a l ing derivation of $\square$ from S such that no literal of index 2 i ever resolved against the transitivity axiom. This restriction prevents any resolvent that contains two literals of index 3 or two literals of index 2 from arising.

Before proving the completeness of this restriction we wish to point out a peculiar aspect of it. Consider the following collection of ground clauses:

$$[[\quad (a) \underset{3}{\nless} (b) \qquad (b) \underset{2}{\nless} (c) \qquad (a) \underset{1}{<} (c)\ ]$$
$$[\quad (c) \underset{3}{\nless} (d) \qquad (d) \underset{2}{\nless} (e) \qquad (c) \underset{1}{<} (e)\ ]$$
$$[\quad (a) \underset{3}{\nless} (c) \qquad (c) \underset{2}{\nless} (e) \qquad (a) \underset{1}{<} (e)\ ]$$
$$[\quad (a) \nless (e)\ ]$$
$$[\quad (a) < (b)\ ]$$
$$[\quad (b) < (c)\ ]$$
$$[\quad (c) < (d)\ ]$$
$$[\quad (d) < (e)\ ]\ ].^{14}$$

Note that it is unsatisfiable, but that if we observe the restriction of not resolving any literal of index 2 against the transitivity axiom, then we shall not obtain $\square$. However, using the uninstantiated transitivity axiom

$$[\quad x \underset{3}{\nless} y \qquad y \underset{2}{\nless} z \qquad x \underset{1}{<} z\ ]$$

we can obtain $\square$ under this restriction.

Thus to prove the completeness of the restriction we consider first a "semi" ground case. Suppose S is a collection of ground clauses, every literal in every clause of S has index

greater than 3, and

$S^* = S \cup [[x \not<_3 y \quad y \not<_2 z \quad x <_1 z]]$ is unsatisfiable. Then there is a locking derivation of $\square$ from S^* in which no literal of index 2 is resolved against the transitivity axiom.

We procede by induction on K(S). If K(S) = 0, then there exist ground terms a1, a2, ..., an such that for $0 < i < n$ $[a_i < a_{i+1}] \in S$ and $[a_1 \not< a_n] \in S$. This can be seen by considering $S \cup [[x \not<_1 y \quad y \not<_2 z \; x <_3 z]]$. From this it follows that there is a derivation of $\square$ from S^* in which no literal of index 2 is resolved against the transitivity axiom.

If $K(S) > 0$, then there is a literal L in a non-unit clause C of S such that the index of L is greater than the index of any literal in any non-unit clause in S and greater than 3. By the induction hypothesis $(S^* - [C]) \cup [C - [L]]$ admits a restricted lock derivation of $\square$. Hence S^* admits a restricted lock derivation of $\square$ or $[L]$ where L has the same index it had in C. But $(S^* - [C]) \cup [[L]]$ admits a restricted lock derivation of $\square$. Hence S^* admits a restricted derivation of $\square$. QED. The lifting to the full general case is immediate.

We hope that results similar to this can be derived for other axioms. We also look for a generalization of locking that will permit control over a set of axioms. In particular, we believe it is possible to lock equality axioms in such a way as to

simulate paramodulation(21) and E-resolution(12).

Appendix

Two complications, monumentally shallow, arise in the locking of literals.

The first problem is that the "same" literal may have different indices in different clauses. To be completely precise we might redefine a literal to be an ordered pair consisting of an index and a "literal" as we have defined it. Then we should redefine an interpretation to contain equivalence classes of literals. And then we should worry whether it is possible to unify literals with different indices. We believe this would only be a source of annoyance to the reader.

The second problem is the notion of "factor." In the main theorem of the thesis (4.6) we start with a fully-factored set of clauses. It is a minor point, but true, that when generating a fully factored set one need pay no attention to indices. That is, one may "fully-factor" first, and then "lock." But if literals have indices "built-in" as suggested above, then to achieve the desired factor-locking generality, one must engage in a gyration like the following:

Variations

If C and C' are clauses and there is a one-to-one correspondence between C and C' such that corresponding literals have the same sign and atom (but not necessarily the same

indices), then C and C' are called "variations" of one another.

Factor*

If C is a clause, T is a subset of C, the members of T all have the same sign, σ is a most general unifier of the atoms of the members of T, B is the set of all literals in C_σ for which there does not exist a literal in C_σ with the same sign and atom but a lower index, and D is a variation of B, then D is called a "factor*" of C.

Fully-factored* Sets

If S is a collection of clauses such that if C is in S and C' is a factor* of C, some variant of some variation of C' is in S, then S is said to be "fully-factored*."

Index

Footnotes

[1] The exact definition of resolvent may be found at (1.24). It is there explained that one looks for a "most general unifier". Before one resolves two clauses C and D, it is actually necessary first to change the variables in C and D so that C and D will have no variables is common; otherwise, one may not obtain the right unifier. We pass over this detail in our examples throughout the thesis.

[2] Before resolving C and D one must first create a variant C' of C and a variant D' of D such that C' and D' have no variables in common. Then one resolves C' and D' instead of C and D. Otherwise, one could not resolve [$\neg$ (G x)] with [(G (f x))], for example.

[3] A valuable reference on the first order predicate calculus is Schoenfield(19). He proves Herbrand's Theorem on p.54.

[4] We ignore tautologous disjunctions such as ((G y (a)) $\vee \neg$ (G y (a))).

[5] In the Introduction, we claimed that Herbrand's Theorem dealt with tautologies. $D_{\sigma_1} \wedge D_{\sigma_2} \wedge \ldots \wedge D_{\sigma_k}$ is inconsistent if and only if $\neg D_{\sigma_1} \vee \neg D_{\sigma_2} \vee \ldots \vee \neg D_{\sigma_k}$ is a tautology.

[6] There is a valuable examination of this problem in Kowalski & Kuehner (9). For a discussion of other inadequacies of resolution, see Bledsoe (5) and Bledsoe, Boyer, & Henneman (6).

[7] In section 5.8 there is a comparison of locking with the works of these authors.

[8] It is defined in Anderson(2), for example.

[9] This result was pointed out to me in a conversation with Robert Anderson.

[10] We have not seen the original source (viz. Unpublished seminar notes. Stanford University, fall 1965, cited by Slagle(18)). This technique is described in Allen and Luckham(1).

[11] In this chapter, if C is a unit clause, D is a clause different from C, and C is a subset of D, then we say that "C subsumes D" and delete D. This is a very special case of the general principle of subsumption.

[12] In the next two examples, we depart from the "prefix" notation for "$<$" and "$\neg$". Furthermore, 0 is a contant here and should really be written "(0)."

[13] This is example 4, section 6, of (6).

[14] This example was pointed out by Robert Anderson.

Bibliography

1 Allen,J., & Luckham,D., An interactive theorem-proving program., Machine Intelligence vol. 5, B.Meltzer & D. Michie eds., pp. 321-36 (Edinburgh University Press) 1969.

2 Anderson, R., Some theoretical aspects of automatic theorem proving., (Ph.D.Thesis, Mathematics Dept. Univ. Texas at Austin) 1970.

3 Anderson,R., & Bledsoe,W., A linear format for resolution with merging and a new technique for establishing completeness, J. ACM, 17, pp.525-534,(July 1970).

4 Andrews,P.,Resolution with merging.,J. ACM,15,p.720,(Oct 1968).

5 Bledsoe,W.,Splitting and reduction heuristics in automatic theorem proving., Artificial Intelligence, 3, pp. 55-77, (1970).

6 Bledsoe,W., Boyer,R., Henneman,W., Computer proofs of limit theorems., Proc. IJCAI 2, 1971(to appear).

7 Davis,M., & Putnam,H., A computing procedure for quantification theory., J. ACM, 7, pp. 201-15,(1960).

8 Kowalski,R., & Hayes,P., Semantic trees in automatic theorem-proving, Machine Intelligence vol.4, B. Meltzer & D. Michie, eds., pp. 87-101, (American Elsevier Publishing Co., Inc., New York) 1969.

9 Kowalski,D., & Kuehner,D., Linear resolution with

selection function., Memo 34, Metamathematics Unit (University of Edinburgh) 1970.

10 Loveland,D., A linear format for resolution. Symposium on automatic demonstration, Lecture Notes in Mathematics,vol. 125, (Springer-Verlag, Berlin, New York) 1970.

11 Loveland,D.,Some linear Herbrand proof procedures: an analysis., Department of Computer Science Memo (Carnegie-Mellon University) 1970.

12 Morris, J., E-resolution: extension of resolution to include the equality relation., Proc. IJCAI 1, (Washington, D.C.) 1969.

13 Prawitz,D., Prawitz,H., & Voghera,N., A mechanical proof procedure and its realization in an electronic computer., J. ACM, 7, pp. 102-28, (1960).

14 Prawitz, D., An improved proof procedure., Theoria, 26, pp. 102-39, (1960).

15 Reiter, R., Two results on ordering for resolution with merging and linear format., (to appear).

16 Robinson, J., A machine-oriented logic based on the resolution principle., J. ACM, 12, pp. 23-41 (January 1965)

17 Robinson, J., Automatic deduction with hyper-resolution., Int.J. Comput. Math.,1, 227-34, (July 1965).

18 Slagle,J., Automatic theorem-proving with renamable and sematic resolution., J. ACM, 14, pp. 687-697, (October 1967).

19 Schoenfield, J., Mathematical logic ., (Addison-Wesley, Reading, Mass.) 1967.

20 Wang, Hao., Towards mechanical mathematics., IBM J. Res. Dev., 4, pp. 2-22, (January 1960).

21 Wos,L., & Robinson, G.A., Paramodulation and theorem-proving in first-order theories with equality., Machine Intelligence, vol. 4, B. Meltzer & D. Michie, eds., (American Elsevier Publishing Co., Inc., New York) 1969.

Vita

Robert Stephen Boyer was born August 2, 1946 in Washington, D.C., to Cdr. and Mrs. Fred Y. Boyer, U.S.N.(ret.).

He was graduated from H.M. King High School in Kingsville, Texas in May 1964.

He attended Texas A&I University in the summer of 1964.

He matriculated at the University of Texas at Austin in September 1964 and was graduated Bachelor of Arts with High Honors in August 1967. He took a major in mathematics and a minor in philosophy.

Since September 1967 he has been enrolled as a graduate student in mathematics at the University of Texas at Austin. From 1967 to 1970 he was a teaching assistant in the Mathematics Department.

He is co-author with Dr. W.W. Bledsoe and Dr. William Henneman of a paper "Computer Proofs of Limit Theorems."

He is married to Anne Herrington Boyer, and they have a daughter Madeleine.

Permanent Address:

555 Elizabeth Avenue

Kingsville, Texas 78363

This thesis was typed by a PDP-10 computer.