# Recognizing Unnecessary Inference

Dan Benanav
Rensselaer Polytechnic Institute
Computer Science Department
Troy, N.Y. 12180

## Abstract

Intelligent reasoners sometimes draw conclusions that lack new or relevant information. Similarly, automated reasoning systems can produce formulas that are not necessary for the problem at hand. We concentrate on the problem of unnecessary inference in the context of resolution based systems. In such systems several strategies have been developed that allow for the deletion of clauses without sacrificing completeness. Unfortunately these strategies fail to recognize other frequently generated unnecessary formulas. We will present a generalized subsumption theorem that can be used to recognize such formulas and to develop new deletion methods which retain completeness.

## 1 Introduction

Intelligent reasoners expend much effort deciding what information is necessary for the problem at hand. When a conclusion is drawn we decide if it contains new and relevant information. It is well known that the performance of automated reasoning systems can be enhanced by eliminating unnecessary formulas. In such systems conclusions drawn from unnecessary formulas are also unnecessary. Failure to prevent the generation of these formulas can lead to rapid combinatorial explosion [Wos, 1988].

In this paper we concentrate on the problem of recognition of unnecessary formulas in the context of resolution based systems. In such systems several strategies have been developed that allow for the deletion of clauses without sacrificing completeness. Common strategies include subsumption, tautology elimination, and demodulation. The subsumption strategy eliminates clauses that are instances of other clauses. The demodulation strategy pertains to clauses containing the equality predicate. Using this strategy demodulators are used to rewrite clauses that are subsequently deleted.

Unfortunately these strategies fail to recognize other frequently generated unnecessary formulas. A less obvious way in which resolution based systems produce unnecessary formulas is related to skolemization. Skolem functions have the effect of automatically creating names for new objects, but sometimes too many names are created, leading to the generation of unnecessary formulas. In section 2 we will discuss some specific examples to clarify why and how this happens.

Another cause of unnecessary formula generation is due to the presence of permutative predicates. By a permutative predicate we mean one that is logically equivalent under any permutation of some subset, $s$, of its arguments where $|s| > 1$. In the case of a binary predicate this implies the relation is reflexive. A simple example is the *Equal* predicate where

$$Equal(x, y) \Leftrightarrow Equal(y, x)$$

holds. In clause form this axiom becomes

$$\neg Equal(x, y) \vee Equal(y, x). \qquad (c_1)$$

Now suppose that the unit clause $Equal(a, b)$ is generated. It resolves with $c_1$ to produce the clause $Equal(b, a)$. In general if $P$ is permutative and $P(a_1, a_2, \ldots a_n)$ is generated then resolution based systems tend to generate all formulas of the form $P(a_{j_1}, a_{j_2}, \ldots a_{j_n})$, where $j_1, j_2, \ldots j_n$ is a permutation of $1, 2, \ldots, n$. Thus we can end up with $n!$ such formulas. Often many of these formulas are not required for the proof.

The main result discussed here can be viewed as a generalization of subsumption. This result can be used to develop new deletion methods and to show that these methods do not sacrifice completeness. Using deletion methods we have developed, we were able to prove certain theorems from Hilbert's axioms for geometry. These theorems present a significant challenge to automated systems due to the constructive nature of the proofs. For a discusion of these theorems see [Benanav, 1988].

In section 2 we discuss an example that illustrates a subtle way in which resolution systems generate unnecessary formulas and present informal arguments as to why these formulas are unnecessary. In later sections we define more precisely what we mean by unnecessary clauses and how to recognize them.

## 2 The Naming Problem

In mathematical arguments one often sees inferences that assert the existence of some object and give it a name. Subsequently, other inferences are made that refer to the object by this name. This process is so natural

that we pay little attention to it and perhaps underestimate its importance. In this section we show how resolution creates more names than is necessary and in the process generates far too many formulas.

The usual way to express constructive statements in predicate logic makes use of existential quantifiers. In resolution based systems existential quantifiers are removed and replaced with functions. For example, consider the axiom,

A1. Any line contains at least two distinct points.

This can be expressed in predicate logic as

$$\forall y(Line(y) \rightarrow \exists x_1 x_2(Point(x_1) \wedge Point(x_2) \wedge \neg Eq(x_1, x_2) \wedge On(x_1, y) \wedge On(x_2, y)))$$

To skolemize this formula the variables $x_1$ and $x_2$ are replaced with functions $p_1(y_1)$ and $p_2(y_1)$ respectively, and the existential quantifiers are removed. After these replacements we get the formula,

$$\forall y(Line(y) \rightarrow (Point(p_1(y)) \wedge Point(p_2(y)) \wedge \neg Eq(p_1(y), p_2(y)) \wedge On(p_1(y), y) \wedge On(p_2(y), y)))$$

Proofs often contain statements that begin with the phrase, "Let $A$ be the ...". This phrase has the effect of generating a name for an entity that can be asserted to exist. Skolemization has this same effect of generating names for things. For instance, consider the axioms consisting of A1 and

A2. $a$ is a line.

which are represented in clause form by the formulas,

$$\begin{array}{l} \neg Line(y_1) \vee On(p_1(y_1), y_1) \\ \neg Line(y_1) \vee On(p_2(y_1), y_1) \\ \neg Line(y_1) \vee Point(p_1(y_1)) \\ \neg Line(y_1) \vee Point(p_2(y_1)) \\ \neg Line(y_1) \vee \neg Eq(p_1(y_1), p_2(y_1)) \\ Line(a) \end{array} \qquad (G_1)$$

The clause $Line(a)$ resolves with each of the other clauses, yielding

$$\begin{array}{l} On(p_1(a), a) \\ On(p_2(a), a) \\ Point(p_1(a)) \\ Point(p_2(a)) \\ \neg Eq(p_1(a), p_2(a)) \end{array} \qquad (G_2)$$

The new clauses produced contain two new constants, $p_1(a)$ and $p_2(a)$. The corresponding human proof might contain the following argument: "Since $a$ is a line, there must be two distinct points on it. Let's call those points $p_1(a)$ and $p_2(a)$."

Suppose that we add to the two previous axioms, the axiom,

A3. There exists two distinct points on the line $a$.

This is represented in clause form as,

$$\begin{array}{l} On(q_1, a) \\ On(q_2, a) \\ Point(q_1) \\ Point(q_2) \\ \neg Eq(q_1, q_2) \end{array} \qquad (G_3)$$

After resolving $Line(a)$ with the other clauses, the database would contain the clauses $G_1$, $G_2$, and $G_3$. We claim that the clauses in $G_2$ are unnecessary. To see this consider how an intelligent reasoner might argue. From axiom A3 we know there exists two distinct points on the line $a$. Let's call those points $q_1$ and $q_2$. From axioms A1 and A2 we know that $a$ must contain two distinct points. But this is just what axiom A3 says. Therefore no new names are created.

The redundant nature of this inference becomes clear by considering the corresponding deduction without first converting the axioms to clause form. They are represented in non-clausal form by the formulas,

$$\forall y(Line(y) \rightarrow \exists x z(Point(x) \wedge Point(z) \wedge \neg Eq(x, z) \wedge On(x, y) \wedge On(z, y))) \qquad (F_1)$$

$$Line(a) \qquad (F_2)$$

$$\exists x_1 x_2(Point(x_1) \wedge Point(x_2) \wedge \neg Eq(x_1, x_2) \wedge On(x_1, a) \wedge On(x_2, a)) \qquad (F_3)$$

Note that by substitution and modus ponens we can infer $F_3$ from $F_1$ and $F_2$. If the initial set of formulas is $F_1$, $F_2$, and $F_3$, then deducing $F_3$ from $F_1$ and $F_2$ is redundant. However, the clauses $G_2$, generated by the resolution inference rule, are not the same as the clauses $G_3$.

It should be remarked that the fact that the clauses $G_2$ are redundant has nothing to do with the equality predicate, although the clauses $G_1$ contain the $Eq$ symbol. We can easily construct a similar example where the equality predicate is not present. For example, we encounter the same problem if we begin with the axioms

A4. For any line there is a point not on the line.

A5. The point $Pt$ is not on the line $a$.

To illustrate the proliferation of such unnecessary clauses, consider in addition to those in $G_1$, the clauses,

$$\begin{array}{l} \neg Point(x) \vee \neg Point(y) \vee Eq(x, y) \vee Line(L(x, y)) \\ \neg Point(x) \vee \neg Point(y) \vee Eq(x, y) \vee On(x, L(x, y)) \\ \neg Point(x) \vee \neg Point(y) \vee Eq(x, y) \vee On(y, L(x, y)) \end{array}$$

These clauses represent the axiom,

A6. Through any two distinct points there is a line.

As before, the clauses $G_2$ can be produced. Then the clauses

$$\begin{array}{l} \neg Point(x) \vee \neg Point(y) \vee Eq(x, y) \vee Line(L(x, y)) \\ Point(p_1(a)) \\ Point(p_2(a)) \\ \neg Eq(p_1(a), p_2(a)) \end{array}$$

resolve to form the clause,

$$Line(L(p_1(a), p_2(a))$$

Since the clauses in $G_2$ are unnecessary this clause is also unnecessary. However note that it resolves with the clauses in $G_1$ to produce the unnecessary clauses,

$$\begin{array}{l} On(p_1(L(p_1(a), p_2(a))), L(p_1(a), p_2(a))) \\ On(p_2(L(p_1(a), p_2(a))), L(p_1(a), p_2(a))) \\ Point(p_1(L(p_1(a), p_2(a)))) \\ Point(p_2(L(p_1(a), p_2(a)))) \\ \neg Eq(p_1(L(p_1(a), p_2(a))), p_2(L(p_1(a), p_2(a)))) \end{array}$$

and the process continues ad infinitum.

## 3 What is an Unnecessary Clause?

There are several different ways in which a clause can be unnecessary. For example, consider the clauses,

$$S = \{P,\ P \vee Q,\ P \vee \neg Q,\ \neg P,\ R\}.$$

Suppose we are using the hyperresolution inference rule to deduce the empty clause. Recall that to deduce a clause from a set of clauses by hyperresolution, at most one clause, $C$, in the set can contain both positive and negative literals. The other clauses can contain only positive literals. All negative literals in $C$ should be resolved away. For this example it will be important to note that $P \vee \neg Q$ and $\neg P$ do not resolve by hyperresolution.

Note that we can obtain a proof by resolving $P$ with $\neg P$. The other clauses, $S_1 = \{P \vee Q,\ P \vee \neg Q,\ R\}$ are not needed for a proof. Given an inference rule $I$ and a set of clauses $S$, we say that a set of clauses $S' \subseteq S$ is *simply unnecessary* if whenever there is a proof of the empty clause from $S$ using $I$, there is also a proof from $S - S'$ using $I$. In this sense the clause $P$ is simply unnecessary since we can obtain the empty clause from $S - \{P\}$ using hyperresolution.

Not only is it possible to obtain a proof from $S - S_1$, but in fact there is a hyperresolution proof that does not use any clause from $S_1$. However using hyperresolution there is no way to obtain the empty clause without using the clause $P$ somewhere in the proof. (By binary resolution we can obtain such a proof). We say that a set of clauses $S'$ are *absolutely unnecessary* with respect to $I$ and a set of clauses $S$ if whenever there is a proof of the empty clause there is also a proof that does not contain any clause from $S'$.

There is an important difference between absolutely unnecessary clauses and simply unnecessary clauses. If a clause is found to be simply unnecessary it means that it can be deleted from the original set of clauses. However, if it gets generated again it cannot necessarily be removed. On the other hand if a clause is absolutely unnecessary then whenever it is generated it can be deleted.

One further sense of unnecessary is illustrated by the clause $R$. In fact there is no proof that uses $R$ to derive the empty clause. In such a case we say $R$ is *completely irrelevant* .

Progress in development of methods to recognize any of these classes of unnecessary formulas could lead to significant improvements in the performance of automated reasoning systems. In this paper we will examine the question of how to determine whether a set of clauses are absolutely unnecessary. In the remainder of this paper we will use the term unnecessary to mean absolutely unnecessary.

## 4 Recognizing Absolutely Unnecessary Clauses

To see how one might show a set of clauses to be absolutely unnecessary consider the clauses,

$$S_1 = \{P_1(a) \vee Q(b,b),\ P_1(a) \vee \neg Q(b,b),\ \neg P_1(a)\}$$

$$S_2 = \{P_2(c) \vee Q(d,d),\ P_2(c) \vee \neg Q(d,d),\ \neg P_2(c)\}$$

and let $S = S_1 \cup S_2 \cup S_3$ where $S_3$ is a set of clauses which do not contain $P_2(c)$ and $d$. Since $S_1$ is unsatisfiable, $S_2$ is unnecessary for deriving the empty clause from $S$ by binary resolution. The question is whether there is a method for determining that $S_2$ are unnecessary without having to first obtain a proof.

Note, if all occurrences of $P_2(c)$ in $S_2$ are replaced by $P_1(a)$ and all occurrences of $d$ are replaced with $b$ then $S_2$ becomes the same as $S_1$. Based on this observation it is easy to show, by induction, if $C$ follows from $S$ then we can also derive a clause $C'$ from $S - S_2$ where $C'$ is $C$ with all occurrences of $P_2(c)$ replaced by $P_1(a)$ and all occurrences of $d$ replaced with $b$. For example, note that we can derive $P_2(c)$ by resolving $P_2(c) \vee Q(d,d)$ with $P_2(c) \vee \neg Q(d,d)$. Also note that by replacing $P_2(c)$ by $P_1(a)$ in the clause $P_2(c)$ gives the clause $P_1(a)$. We can also derive $P_1(a)$ by resolving $P_1(a) \vee Q(b,b)$ with $P_1(a) \vee \neg Q(b,b)$. From these observations it follows that if we can derive the empty clause from $S$ we can also derive it from $S - S_2$ since the empty clause replaced with anything just gives the empty clause.

A convenient way to express replacement is with the use of rewrite rules. A rule $l \rightarrow r$ applies to to term $t$ if a subterm $s$ of $t$ matches the left-hand side $l$ with some substitution $\sigma$ (i.e. $s = l\sigma$). To apply the rule the occurrence of the subterm $s$ in $t\sigma$ is replaced with $r\sigma$. For any set of rules $R$ we write $t \Rightarrow_R u$ to indicate a term $t$ rewrites to $u$ by a single application of a rule in $R$. A *derivation* is a sequence of rewrites; if $t \Rightarrow \cdots \Rightarrow u$ in zero or more steps we say $u$ is *derivable* from $t$; if no rule can be applied to $t$ we say that $t$ is *irreducible*. A set of rules $R$ is *terminating* if no infinite derivations are possible. We extend these definitions to clauses in the appropriate way.

For example let $r_1$ be the rule $P_2(c) \rightarrow P_1(a)$ and let $r_2$ be the rule $d \rightarrow b$. Using this terminology we can say that the clause $P_2(c) \vee Q(d,d)$ rewrites to $P_1(a) \vee Q(b,b)$ using the rules $r_1$ and $r_2$. We can also say that $P_1(a) \vee Q(b,b)$ is irreducible by $r_1$ and $r_2$. We can now state a conjecture as follows:

**Conjecture** *Let $S$ be a set of clauses and $R$ a set of rewrite rules such that the set of clauses $S'$, obtained by rewriting all clauses in $S$ to irreducible forms, is contained in $S$. Then all clauses reducible using $R$ are unnecessary.*

In the example just shown, we take $R = \{r_1, r_2\}$. Rewriting the clauses in $S$ using $R$ gives the irreducible clauses $S_1$ which are contained in $S$. All the clauses in $S_2$ are reducible using $R$ and are therefore unnecessary.

Except for certain subtleties the above conjecture illustrates the basic approach to showing clauses to be unnecessary. We are given a set of clauses $S$. We attempt to find a set of rewrite rules $R$ and use these rules to form a set $S'$ obtained from $S$. If everything in $S'$ is *subsumed* by something in $S$ then any clause reducible by $R$ is unnecessary. In order to clarify the conjecture we must answer the questions,

1. What properties do the rewrite rules have to have?
2. How do we form the set $S'$ from $S$ and $R$?

It turns out that the answer to the first question is that the rewrite rules have to be terminating and no rule in $R$ should contain the negation symbol. The termination condition assures us that we do not have rules such as $a \rightarrow b$ and $b \rightarrow a$ both in $R$. The second condition assures us that we do not have rules such as $\neg P \rightarrow Q$ and $P \rightarrow Q$.

The answer to the second question is that we form a set which we call the *Terminal Overlap Closure* of $S$ and $R$. In the example we have been discussing all the rules and clauses were ground. In the case where the rules and clauses are not ground we must take into consideration clauses that may *overlap* with rules in $R$. For example, we may have a rule $P(x,a) \rightarrow Q(x)$ in $R$ and a clause in $S$ of the form $P(b,y) \vee R(y)$. Although $P(b,y) \vee R(y)$ does not rewrite using $P(x,a) \rightarrow Q(x)$, the left hand side of the rule, $P(x,a)$, unifies with the literal $P(b,y)$ and must be taken into consideration. In the next section we show how to compute the terminal overlap closure and indicate why this is the appropriate set.

## 5 Terminal Overlap Closure

We begin by defining the *clause overlap* of a clause $C$ and a rule, $l \rightarrow r$. We assume that $C$ and $l \rightarrow r$ have no variables in common. Let $\Gamma$ be a nonempty set of nonvariable subterms in $C$ that unify with $l$ and let $\lambda$ be the most general unifier (m.g.u.). Let $C'$ be the clause formed by replacing every occurrence of $l\lambda$ in $C\lambda$ by $r\lambda$. Then $C'$ is a clause overlap of $C$ and $l \rightarrow r$.

**Example 1** If $C$ is a ground clause and $l \rightarrow r$ is a ground rule then there is only one clause overlap, formed by applying $l \rightarrow r$ at every possible position in $C$. For example, the overlap of $f(a) \rightarrow a$ and $P(f(a), f(a)) \vee Q(f(a))$ is $P(a,a) \vee Q(a)$.

**Example 2** When several subterms unify with the left hand side of a rule then there may be several clause overlaps of the clause and the rule. For example, consider the clause

$$P(f(x,b), f(x,b)) \vee Q(f(a,b), f(a,b), x) \qquad (c_2)$$

and the rule

$$f(a,b) \rightarrow a. \qquad (r_1)$$

Since the identity substitution is an m.g.u. of $\{f(a,b)\}$ then

$$P(f(x,b), f(x,b)) \vee Q(a,a,x) \qquad (c_3)$$

is a clause overlap of $c_2$ and $r_1$. Likewise $\{x \leftarrow a\}$ is an m.g.u. of $\{f(a,b), f(x,b)\}$ so

$$P(a,a) \vee Q(a,a,a)$$

is also a clause overlap of $c_2$ and $r_1$.

**Example 3** Another example of multiple overlaps can be illustrated by the clause

$$C = P(f(x,y,c)) \vee Q(f(x,b,z))$$

and the rule

$$r = f(a, y_1, z_1) \rightarrow g(y_1, z_1).$$

In this case

$$\{f(x,y,c), f(a,y_1,z_1)\},$$

$$\{f(x,b,z), f(a,y_1,z_1)\},$$

$$\{f(x,y,c), f(x,b,z), f(a,y_1,z_1)\},$$

are all unifiable. Therefore there are three overlaps of $C$ and $r$,

1. $P(f(a,y,c)) \vee Q(g(b,z))$
2. $P(g(y,c)) \vee Q(f(a,b,z))$
3. $P(g(b,c)) \vee Q(g(b,c))$

Next we define the *Clause Overlap Closure*[1] of a set of clauses $S$ and a set of rules $R$ to be the smallest set $\mathcal{CC}$ such that $S \subseteq \mathcal{CC}$ and if $C \in \mathcal{CC}$ and $C'$ is a clause overlap of $C$ and some rule in $R$ then $C' \in \mathcal{CC}$.

The *Terminal Overlap Closure* of a set of clauses $S$ and a set of rules $R$ is the set of all irreducible clauses that are in the clause overlap closure of $S$ and $R$.

**Example 4** Let $S$ consist of the clauses,

$$P_1(x,y) \vee P(g(x,y)) \qquad (c_4)$$
$$\neg P_1(h(a), h(b)) \qquad (c_5)$$
$$P(g(h(a), h(b))) \qquad (c_6)$$

and let $R$ consists of the rules

$$g(h(x_1), y_1) \rightarrow f(g_1(x_1), y_1) \qquad (r_2)$$
$$f(z_1, h(z_2)) \rightarrow l(g_1(z_1), z_2) \qquad (r_3)$$

Notice that both $c_4$ and $c_5$ are irreducible in $R$ and thus are in the terminal overlap closure. However $c_6$ rewrites to

$$P(f(g_1(a), h(b))) \qquad (c_7)$$

by $r_2$ and thus is in the clause overlap closure but not in the terminal overlap closure. Likewise $c_7$ is not in the terminal overlap closure since it rewrites by $r_3$ to

$$P(l(g_1(g_1(a)), b)). \qquad (c_8)$$

This clause is irreducible and is therefore in the terminal overlap closure. Since the subterm $g(x,y)$ of $c_4$ unifies with the left hand side of $r_2$,

$$P_1(h(x), y) \vee P(f(g_1(x), y)) \qquad (c_9)$$

is in the clause overlap closure. Since it is irreducible it is also in the terminal overlap closure. Now the subterm $f(g_1(x), y)$ of $c_9$ unifies with the left hand side of $r_3$.

$$P_1(h(x), h(z_2)) \vee P(l(g_1(g_1(x)), z_2)) \qquad (c_{10})$$

is in the clause overlap closure. Again, since it is irreducible it is also in the terminal overlap closure. Up to renaming of variables these are the only clauses in the terminal overlap closure.

[1] The term "overlap closure" is used in [Guttag *et al.*, 1983] to denote a similar operation, but on a set of rewrite rules.

## 6 The Generalized Subsumption Theorem

We now state the generalized subsumption theorem and some important lemmas. We assume throughout that the set of rules R contain no negation symbols.

Theorem (Generalized Subsumption Theorem) Let R be a terminating set of rules. Let $S_1$ and S? $b_e$ sets of clauses such that every clause in the terminal overlap closure of $S_1$ and R is subsumed by a clause in $S_2$. Then if there is a resolution refutation from $S_1$ there is a refutation from $S_2$ which contains only terms that are irreducible with respect to R. In other words, any clause that reduces by R is unnecessary.

This theorem applies to most common resolution strategies with factoring. For example, the theorem holds if we are using UR-resolution with factoring. We will elaborate further how this theorem can be applied. First we will briefly indicate why the theorem holds.

The proof of this theorem is based on the following crucial lemma:

Lemma (Clause Overlap Closure Lemma) Let R be a set of rules. If C is a resolvent of C\ and C2 and C' is in the clause overlap closure of C and R then we can construct a $C_1$ in the clause overlap closure of C\ and R, a $C_2$ tn the clause overlap closure of $C_2$ and R, and a resolvent C" of $C_1$ and $C_2$ such that C" subsumes C.

For example let S and R be the clauses and rules from example 4. Note that $c_6$ is a resolvent of $c_4$ and $c_5$. Also as shown, $C_7$ is a clause overlap of $C_6$ and R. Note also that $C_9$ is a clause overlap of $C_4$ and R and that it resolves with $c_5$ giving $C_7$. Since $c_5$ is in the clause overlap closure of $C_5$ and R, the lemma holds for this example.

Based on this lemma we can show the following:

Lemma (Terminal Overlap Closure Lemma) Let R be a terminating set of rules. If C is a resolvent of $C_1$ and $C_2$ and C is in the terminal overlap closure of C and R then we can construct a $C_1$ tn the terminal overlap closure of $C_1$ and R, a $C_2$ in the terminal overlap closure 0/C2 and R, and a resolvent C" of C\ and $C_2$ such that C" subsumes C".

To prove this lemma we first use the previous lemma to construct clauses $C_1$ in the clause overlap closure of C\ and R and a clause $C_2$ is the clause overlap closure of C2 and R which resolve to a clause C" which subsumes C". Next it can be shown that either $C_1$ and $C'_2$ are irreducible or we can construct a clause overlap $C_1$" of $C_1$ and R and a clause overlap $C_2$ of $C_2$ and R that resolve to a clause C" that subsumes C". We continue to construct clause overlaps in this manner. Since R is terminating, it can be shown that this construction must eventually terminate with clauses that are irreducible.

Now the generalized subsumption theorem follows by a simple induction argument based on this lemma. For full details of these proofs see [Benanav, 1989] .

## 7 Deletion Methods

Consider the way in which subsumption strategy works. Each time a new clause is generated a check is made to determine if the clause is subsumed by a previously generated clause. If it is, then the clause is deleted. Deleting clauses in this way is called forward subsumption. Likewise is is also possible that the newly generated clause will subsume previously generated clauses. Deleting these clauses is called backward subsumpiion.

More generally, we need to check whether a newly generated clause is unnecessary and whether it's appearance causes other clauses to become unnecessary. Deleting a new clause will be called forward deletion and deleting previous clauses will be called backward deletion.

In this section we discuss how the generalized subsumption theorem can be used to develop methods for forward and backward deletion of clauses. There are several methods which can be developed that do not sacrifice completeness, ranging from general methods that apply to any set of clauses to more specific methods that apply to clauses with certain special properties. The most general methods are probably too expensive to use for many clause sets.

First, suppose that $S_1$ is the starting set of clauses, $S_2$ is the set of clauses which have been derived from $S_1$, and R is a set of terminating rules such that everything in the terminal overlap closure of $S_1$ and R is subsumed by something in $S_2$. (When we begin R is empty). If C is a clause generated from $S_2$ which is reducible by R then C is unnecessary and can be deleted. If C is irreducible then we can attempt to find a set of rules R' such that C is reducible by R\ R U R! is terminating, and everything in the terminal overlap closure of $S_1$ and RU $R^f$ is subsumed by something in $S_2$- Subsequently, any rule reducible by RUR' can be deleted.

To illustrate this we consider the example discussed in section 2. We assume that the starting set of clauses, $S_1$, consists of the clauses G\, $G_3$ and some other clauses that do not contain the symbols $p_1$ or $p_2$- Since these symbols are skolem functions no other clauses will contain them except possibly some equality substitution axioms. We show, however, in [Benanav, 1989] that the equality substitution axioms for skolem functions are simply unnecessary.

As before, we can derive the clauses, $G_2$ Let $S_2$ = S\ U $G_2$ and let R consists of the rules

$$p_1(a) \rightarrow q_1 \qquad (r_4)$$
$$p_2(a) \rightarrow q_2. \qquad ('\cdot 5)$$

By the generalized subsumption theorem, if it can be shown that everything in the terminal overlap closure of S\ and R is subsumed by something in $S_2$, then anything reducible by R is unnecessary. In particular the clauses $G_2$ are unnecessary. Computing the terminal overlap closure of $S_1$ and R gives the clauses $S_1$ and the clauses,

$$\neg Line(a) \vee On(q_1, a)$$
$$\neg Line(a) \vee On(q_2, a)$$
$$\neg Line(a) \vee Point(q_1)$$
$$\neg Line(a) \vee Point(q_2)$$
$$\neg Line(a) \vee \neg Eq(q_1, q_2)$$

Note that $G_4$ are subsumed by G3. Therefore we have shown that everything in the terminal overlap closure of $S_1$ and R is subsumed by something in $S_2$.

Although axiom A3 is not an axiom of geometry, this example illustrates what actually happens when when resolution-based systems are run on a set of clauses representing Hilbert's axioms for geometry. The axioms $A_1$, A2, and A6 are three such axioms and are represented by the clauses $G\backslash$ and 2. These clauses produce the clauses 2 which can be shown to be unnecessary by the generalized subsumption theorem.

We have shown how to use the generalized subsumption theorem to show that a set of clauses is unnecessary however, we have not indicated a method for finding the appropriate set of rules, $R'$. There are two difficulties associated with finding these rules. First, it may be difficult to show that $R \cup R'$ is terminating. In fact, the termination problem has been shown to be undecidable. Nevertheless, much research has been devoted to this problem and several excellent methods for proving termination are available (see [Dershowitz, 1987] for a survey). Another difficulty is testing that everything in the terminal overlap closure of $S\backslash$ and $R \cup R'$ is subsumed by something in $S_2$- In cases where the terminal overlap closure of $S_1$ and $R \cup R'$ is finite we can simply check that each clause is subsumed by a clause in $S_2$- But if the terminal overlap closure of $S\backslash$ and $RUR'$ is very large this can be computationally expensive. In fact, checking whether a term is subsumed by another term is NP-hard.

There are several ways to circumvent these difficulties. For example, we can require that the right hand sides of all rules be ground terms that are not reducible by other rules. In such cases it is easy to show that the rules will be terminating and the terminal overlap closure of $S\backslash$ and $R$ will be finite. In addition, we can require the left hand side of the rules to be ground terms not reducible by other rules. Then to check that everything in the terminal overlap closure of $S\backslash$ and $R \cup R$ is subsumed by a clause in $S_2$, one need only compute the terminal overlap closure of $S\backslash$ and $R'$.

It is not always necessary to construct rules in the actual implementation of deletion strategies based on the generalized subsumption theorem. For example in the case of permutative predicates it may be possible to determine that only one permutation of each formula need be retained. For a discussion on efficient methods that partially solve the naming problem and the permutative predicate problem see [Benanav, 1988].

## 8 Conclusions

In this paper we have shown how resolution systems tend to generate unnecessary clauses and have presented a theorem which can be used to show certain clauses are unnecessary. To apply this theorem one must be able to construct a terminating set of rules that have a special relationship to the starting set of clauses and the derived set of clauses. Namely, everything in the terminal overlap closure of the starting set of clauses and the set of rules must be subsumed by a clause in the derived set of clauses. In general these rules can be difficult to find, however by placing appropriate restrictions on the kinds of rules used, efficient methods can be developed.

Unfortunately, this theorem alone does not allow for the deletion of other kinds of unnecessary clauses that can be generated. In order to determine the full range of applicability of this theorem more experimentation needs to be done. In addition, we believe that further investigation of methods for deletion of other kinds of unnecessary clauses, such as those that are irrelevant, can signifigantly enhance resolution based systems. In fact, any reasoning system will need to devote computational resources to these matters in order to be effective.

## Acknowledgments

I would like to thank my advisor David Musser for his helpful suggestions on drafts of this paper and for several useful discussions I had with him regarding the ideas in this paper. I would also like to thank the developers of LMA and ITP for providing a powerful automated reasoning program which has been an invaluable tool for this research.

## References

[Benanav, 1988] D. Benanav. Recognizing unnecessary inference. Technical report, Rensselaer Polytechnic Institute, December 1988.

[Benanav, 1989] D. Benanav. *Recognizing Unnecessary Inference.* PhD thesis, Rensselaer Polytechnic Institute, 1989.

[Boyer and Moore, 1979] R. Boyer and J. Moore. *A computational logic.* Academic Press, Inc., 1979.

[Bundy, 1983] A. Bundy. *The Computer Modelling of Mathematical Reasoning.* Academic Press, Inc., 1983.

[Chang and Lee, 1970] C. Chang and R. Lee. *Symbolic Logic and Mechanical Theorem Proving.* Academic Press, Inc., New York, 1970.

[Dershowitz, 1987] N. Dershowitz. Termination of rewriting. In J.-P. Jouannaud, editor, *Rewriting Techniques and Applications.* Academic Press, 1987.

[Guttag *et al.*, 1983] J. V. Guttag, D. Kapur, and D. R. Musser. On proving uniform termination and restricted, termination of rewriting systems. *Siam J. Comput,* 12:189-214, 1983.

[Hilbert, 1938] D. Hilbert. *Foundations of Geometry.* The Open Court Publishing Company, 1938.

[Lankford, 1975] D. Lankford. Canonical inference. Report ATP- 32, Dept. of Mathematics and Computer Science, Univ. cf, Texas, Austin, Texas, 1975.

[Loveland, 1978] D. W. Loveland. *Automated Theorem Proving: A Logical Basis.* North-Holland, 1978.

[Robinson, 1965] J. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the Association for Computing Machinery,* 12(1):23—41, January 1965.

[Vaisman, 1980] I. Vaisman. *Foundations of Three-Dimensional Geometry.* Marcel Decker, Inc., New York, 1980.

[Wos, 1988] L. Wos. *33 Basic Research Problems.* Prentice Hall, Englewood Cliffs, 1988.