Reducing the Derivation of Redundant Clauses in Reasoning Systems

Rolf Socher-Ambrosius
Fachbereich Informatik, Universitat Kaiserslautern
Postfach 3049, D-6750 Kaiserslautern, W.-Germany

Abstract

This paper addresses two problems concerning the issue of redundant information in resolution based reasoning systems. The first one deals with the question, how the derivation of redundant clauses can be substantially reduced a priori. The second one asks for a criterion to decide, which clauses need not be tested for reduncancy. In this paper we consider a particular kind of reduncancy, which we call *ancestor subsumption,* that is the subsumption of a resolvent by one of its ancestors. We give a complete syntactic characterization of clause sets producing ancestor subsumed clauses. Given this characterization, a solution to the problem with generating redundant clauses is proposed. Moreover, a suitable restriction of the (usually very expensive) subsumption test is derived from this result. SAM's lemma will serve as an example for demonstrating various possibilities how to avoid the derivation of redundant clauses.

1 Introduction

The derivation of redundant information is one of the greatest obstacles to the efficiency of reasoning programs. Wos (1988) reports an attempt to prove SAM's lemma (see Guard 1969) using hyperrcsolution, where 6000 clauses identical to retained clauses and 5000 clauses being proper instances of retained clauses were generated. Even if these redundant clauses can be removed after their generation, they must be processed with demodulation, subsumption, and other procedures. Moreover, the test on subsumption, being a very useful means for removing reduncancies, is rather expensive as Chang & Lee (1973), Eisinger (1981), and some others remark. A strategy to prevent the generation of redundant clauses, or at least to reduce the amount of newly generated unneeded clauses, would thus prove very useful for increasing the power of reasoning systems.

On closer inspection of the proof of SAM's lemma it turns out that many of the 6000 duplicates are generated by double resolution with a clause of the form *Pxy* v ¬*Pyx.* Such a duplicate is identical to its own "grandfather" in this resolution derivation. More general, we will deal with the situation that a resolvent is subsumed by one of its own ancestors, which we will call *ancestor subsumption.* Ancestor subsumption is a particular kind of *forward subsumption* (Ovcrbcek 1975), that is the subsumption of a newly deduced clause by a given clause. One of the paper's objectives is to characterize clause sets that admit ancestor subsumption. This approach is based on the following observation: A resolvent of two clauses cannot be subsumed by one of its parent clauses, unless the other parent is self-resolving. This can be easily seen for the ground case: let $C=\{L_1,L2,..L_n\}$ and $D=\{-L_1K_2,..,K_n)$ be ground clauses and assume, C subsumes the resolvent $R=[K_2,..,K_{n,}L_2$ $-L_n\}$. Then L eR must hold and from L_1 $\{L_2,..J_n\}$ now follows L_1 $[K_2,..,K_n\}$- Hence D is a tautology. We will generalize this observation in the following way: A resolvent R cannot be subsumed by an ancestor C, unless the set of ancestors of R contains a cycle (the notion of a cycle was introduced by Shostak (1976)). Noncyclic clause sets thus have the nice property of excluding ancestor subsumption. A prominent example for this class of clause sets is *Schubert[1]s Steamroller* (see Stickel 1986). Linear resolution has the property that the later a clause is derived in the linear deduction, the more ancestors it possesses. Provided that a given clause set is known to disallow ancestor subsumption, a linear strategy thus seems most preferable in order to reduce the generation of subsumed clauses.

Another question addressed by this paper is closely related to the first problem. The characterization mentioned above provides a means to avoid the generation of subsumed clauses only for noncyclic clause sets. But for other clause sets we can at least give a criterion to decide, which clauses have to be considered as potential subsumers in the subsumption test. Being not self-resolving, for example, a clause can be excluded from being subsumer of its own "child".

As cycles in clause sets are heavily responsible for the generation of redundant information, a technique to remove such cycles would prove very useful. This paper considers two approaches. A certain class of cycles (cycles representing the logical equivalence of some literals) can be made harmless by using them only as demodulators. Consider for example a clause set containing the two clauses $\neg P \vee Q$ and $\neg Q \vee P$, which form a cycle and give rise to an endless resolution derivation. These two clauses express that P and Q are equivalent literals, which therefore can be substituted for each other without changing the truth value of the whole clause set. Replacing for instance each occurrence of P in the clause set by Q yields an equivalent clause set, where the two cycle clauses have become tautologies. This approach is in fact a demodulation on literals instead of terms. The well-known problems with term rewriting, however, arise with literal demodulation, too. Directing equations to rewrite rules requires the existence of a well founded ordering on terms. Thus, according to the same reason, why the equation $fxy = fyx$ cannot be directed to a rewrite rule, the equivalence $Pxy = Pyx$ cannot be used as a demodulator. The second approach, which overcomes the problem with directing equivalences, consists in using cycles as the basic theory for performing *theory resolution.* The cyclic clauses, for instance the symmetry clause $Pxy \vee \neg Pyx$, disappear in a *theory box,* enabling in this case resolution between the clauses Pab and $\neg Pba$.

2 Basic Notions

In the following we assume the reader to be familiar with the standard terminology of First Order Logic. The few basic notions of clause graphs used in this paper can be found for instance in Eisinger's (1988) thesis. Clauses are always considered as sets of literals, but written without set braces. The empty clause is denoted by $\emptyset$.

Let $\mathcal{V}$ be a denumerable set of variables. A *substitution* a is an endomorphism on the term algebra, which is identical almost everywhere on Vand thus can be represented as a finite set $\sigma = \{x_1 \rightarrow t_1, \ldots, x_n \rightarrow t_n\}$. A substitution p is called a *renaming* substitution, iff p is injective on its domain and $\mathcal{V}\rho \subseteq \mathcal{V}$. A literal or clause L is called a *variant* (or a *copy)* of the literal or clause K, if there exists a renaming substitution p, such that $L\rho = K$. Two substitutions σ, τ are *compatible,* if there exists some substitution X with $\sigma\lambda = \tau\lambda$.

3 Cycles in Clause Sets

This chapter provides a characterization of clause sets admitting ancestor subsumption. Our main result is as follows: Clause sets admitting ancestor subsumption possess *cycles,* whose elements are *the far parents* of the subsumed clause. First, we define the two basic notions of ancestor subsumption in terms of *circular derivations* and of cycles. The proofs of the following lemmata and theorems can be found in the full version, Socher (1988), of this paper.

3,1 Definition;

A clause is self-resolving, if it resolves with a copy of itself.

The following lemma determines those clauses that possibly produce subsumed resolvents. For any clause D we define the deduction relation ->D between clauses C and R by C —>D R> iff R is a resolvent of C and D.

<u>3.2 Lemma:</u>

Let C be a unit clause and D , R be clauses with $C \rightarrow_D R$.

a) If R is a variant of C, then D is self-resolving.

b) If R is an instance of C, that is, $C\mu = R$ for some substitution μ, then $D = LK_1 \ldots K_n$, and L, K_i have the same predicate symbol, but different polarity for all $i \in \{1..n\}$.

c) If C subsumes R, that is, $C\mu \subseteq R$ for some substitution μ, then $D = LK_1 \ldots K_m M_1 \ldots M_n$, and L, K_i have the same predicate symbol, but different polarity for all $i \in \{1..n\}$. ■

The next lemma shows that clauses, which only produce subsumed clauses, are tautologies.

<u>3.3 Lemma:</u>

Let D be a clause.

a) Suppose for all clauses C the following holds: $C \rightarrow_D R$ implies that C and R are variants. Then D is a tautology, $|D| = 2$, and D is function and constant free.

b) Suppose for all clauses C the following holds: $C \rightarrow_D R$ implies that R is an instance of C. Then D is a tautology and $|D| = 2$.

c) Suppose for all clauses C the following holds: $C \rightarrow_D R$ implies that C subsumes R. Then D is a tautology. ■

In the following we will generalize lemmata 3.2 and 3.3, that is, we want to determine those clause sets $\mathcal{D}$, which - possibly or only - produce subsumed clauses. It will turn out that the appropriate generalization of self-resolving and tautologous clauses are *cyclic* clause sets.

We generalize the deduction relation to clause sets $\mathcal{D}$, by $C \rightarrow_{\mathcal{D}} R$, iff there is a sequence $C \rightarrow_{E_1} C_1 \rightarrow_{E_2} \ldots \rightarrow_{E_n} R$ such that $\{E_1, \ldots E_n\} = \mathcal{D}$.

A **path** in directed graph is an alternating sequence (N_1, l_1, ..., N_{n-1}, l_{n-1}, N_n) of nodes and links, such that l_i goes from N_i to N_{i+1}. It is called cyclic, if $N_1 = N_n$. A directed graph is called cyclic, if it contains a cyclic path.

A semicycle *D* is a directed graph, whose nodes are labelled with clauses and whose links are R-links, labelled with substitutions and which satisfies the following conditions a) to c). (Note that we do not distinguish between a node and its label.)

a) Each node of D has the following form (figure 1):

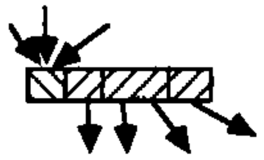

fig.I

with m incoming R-links and n outgoing R-links, and m>0, n>0 holds.

b) There is a node C_o such that each cyclic path of *D* passes C_0 There is a substitution T, which is a common instance of all substitutions of the R-links going into Co and there is a common unifier a for all other links of D.

c) a and T are compatible, with common instance *X*- The substitution *%* is called the cycle substitution of *D.*

The semicycle *D* is called complete, if a is an instance of T, that is La = Ka for all literals L and K joined by a link going into C_0. Let *D* be a semicycle consisting of only one clause C. Then *a* is the identity substitution andC is self-resolving. If, moreover, *CD* is complete, then *C* is a tautology. *D* is called a cycle, if each node has exactly one successor.

A cycle is just what Shostak (1976) and (1979) calls a *loop.* This notion also corresponds to the notion of *recursive predicates* in the terminology of deductive databases (Vieille 1987, Ohlbach 1988) and logic programming.

3,4 Example;

The clause sets D_1 D_2, and D_3, which are shown in figure 2, are semicycles; D_1 is a cycle. It can be seen that each node of a cycle may be chosen to be the special clause C_o. As to semicycles, still several, but in general not all nodes have this property. For instance the clauses ¬*rw* and *sw* cannot be chosen to be C_0.

The previous examples illustrated that in general there are several possibilities to choose a clause of a semicycle to be the special clause C_0. The cycle substitution, however, is independent of this choice, according to the definition of semicycles.

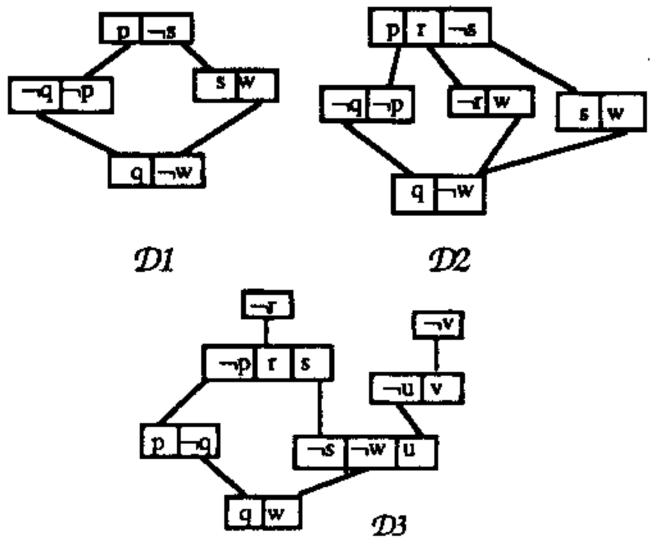

fig.2

The following two theorems provide a characterization of clause sets producing subsumed clauses. It turns out that those clause sets have a "cyclic" structure, varying from the weakest form for clause sets producing some ancestor subsumed resolvent to the strongest form of a cycle for clause sets producing only copies as resolvents.

3.5 Theorem;

Let *C* be a unit clause, let *D* be a set of clauses and let *C->DR.*

a) If *R* is a variant of C, then *D* is a semicycle.

b) If R is subsumed by C, then *D* contains clauses $D_1..D_n$ such that for each D_i, i€ {I,..,n-I} the following hold:

(i) there exist literals K_i and L_i; and an R-link between L_i and k_{i+1}.

(ii) L_n and *K* are literals with the same predicate symbol, but different polarity.

3.6 Theorem;

Let D be a set of clauses.

a) Suppose for all clauses *C* the following holds: *C->R* implies that *C* and *R* are variants. Then *D* is a complete cycle, all clauses of which are function and constant free.

b) Suppose for all clauses *C* the following holds: $C\text{->}D^R$ implies that *R* is an instance of C. Then *D* is a complete cycle.

c) Suppose for all clauses C the following holds: $C\text{->}D^R$ implies that C subsumes R. Then *D* is a complete semicycle.

Note that the definition of the deduction relation ->D does not capture ancestor resolution, which is part of complete linear strategics.

3,7 Example;

Let D be the clause set, which is shown in fig. 3.

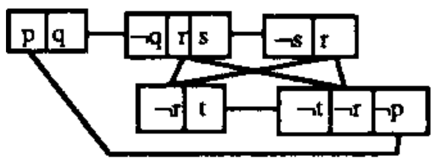

fig 3

D allows a linear deduction with ancestor resolution as follows:

$$\neg p \rightarrow q \rightarrow rs \rightarrow r \rightarrow t \rightarrow \neg r \neg p \rightarrow \neg p$$

where the last step is an ancestor resolution step. But *D* is not a semicycle.

The following lemma shows that it is sufficient to look for cycles in an initial clause set during a resolution deduction, as "new" cycles cannot be generated by resolution. For any clause set 5,R(5) denotes the resolution closure of 5, that is the smallest clause set containing *S* and closed under the resolution operation.

3.8 Lemma;

Let *S* be a set of clauses. Then *S* contains a semicycle, iff R(S) contains a semicycle. ■

4 Removing Cycles From Clause Sets

Besides being the very reason for the undecidability of first order logic, cycles in clause sets also turned out to be a main source of reduncancy for clause set resolution. Thus the question arises as to what extent cycles can be excluded from producing lots of unneeded clauses. A first approach to this question is due to Bibel (1981). He showed that under certain conditions, similar to those allowing the deletion of tautologies in clause graphs, cycles can be removed from clause sets. We will consider here two approaches, both restricted to proper cycles. However, we will also show by means of SAM's lemma, that in particular cases there exists also an appropriate treatment of semicycles.

The first idea in order to shut down this source of reduncancy is to use cycles not directly for resolving. Instead, they are taken as *literal demodulators* (Wos 1967): Cycles, expressing the equivalence of literals, can be transformed into rules, in the same way as equations can be directed yielding rules. Consider for example a clause set containing among others the two clauses ¬*PvQ* and *-QvP.* These two clauses represent the formula *P=Q>* which can be directed yielding for instance the rule *P->Q.* Application of this rule means substituting *Q* for each occurrence of *P* and ¬Q for each occurrence of ¬P. (Note that this effect can be achieved also with an ordering restriction (Loveland 1978), which precludes resolution with the Q and ¬Q literal in the two clauses.) However, the method of literal demodulators applies only for cycle clauses that can be directed, such as ground clauses or clauses with different predicate symbols. Thus, a lot of relevant clause sets does not allow this approach. For instance, the (cyclic) clause ¬*PxyvPyx,* expressing the symmetry of P, cannot be directed.

Another approach, which overcomes the problem with directing equivalences, is based on the idea of "compiling" cycles into a theory serving as the basis for *theory resolution.* Theory resolution, first proposed by Stickel (1985), is a generalization of ordinary resolution. The literals resolved upon need not be syntactically complementary, it is sufficient for them to be complementary under some theory. In particular, each clause *C* = *{L_1.L_n)* defines a theory Tc, in which, for instance, the set of literals {L_1,..,-.L_n} is complementary. Taking clauses (or clause sets) as a particular theory to perform theory resolution is essentially the same idea as Ohlbach's (1988) *link resolution.* The set Tc in general is not computable. Yet, in order to perform theory resolution on a clause set 5, it is sufficient to compute the resolution closure R(*S)* of 5.

4.1 Lemma;

Let *S* be a set of clauses and D E Tsbe a clause, which is not a tautology. Then there is some clause *Ce* R(5), such that C subsumes *D.* ■

In order to find the substitutions , which make a set *[L_1,..,L_n)* of literals complementary under the theory TS, or, equivalently, which map the clause D:={-L_1,..,¬L_n} onto an element of Ts, it is sufficient, according to the previous lemma, to find some and a clause C E R(5), such that C subsumes *Da.* Thus, having a finite resolution closure R(S), the set *S* guarantees the number of Ts-resolvcnts of two clauses to be finite. In particular, this is the case for cycles that produce only copies as resolvents, since these cycles are function free. The next lemma shows that the theory of a complete, function free cycle *CD* with cycle substitution *a* can be completely described by the equivalence of all literals of *(Da.*

4.2 Lemma;

Let *CD* be a complete, function free cycle with cycle substitution a. Then R(*D)* = *[-TLGKG* IL and *K* are literals occurring in *CD).* ■

4.3 Examples:

a) Let $\mathcal{D} = \{\neg Pxy\ Qxy, \neg Quv\ Puv\}$. $\mathcal{D}$ is a complete and function free cycle with cycle substitution $\sigma=\{x\to u, y\to v\}$. The theory of $\mathcal{D}$ consists of the formula $Puv \equiv Quv$.

b) Let $\mathcal{D}=\{\neg Pxy\ Pyx\}$. $\mathcal{D}$ is function free, but not complete. By adding a copy of the clause $\neg Pxy\ Pyx$, a complete cycle $\mathcal{D}$ is obtained.

Then $\Re(\mathcal{D}) = \Re(\mathcal{D}) = \{\neg Pxy\ Pyx, \neg Pxy\ Pxy\}$ is finite and thus the number of $\mathcal{T}_{\mathcal{D}}$-resolvents of any two clauses is finite, too.

c) Let $\mathcal{D}=\{\neg Pf(f(x,y),z)\ Pf(x,f(y,z))\}$.

Then $\Re(\mathcal{D})= \mathcal{D} \cup \{\neg Pf(f(f(x,y),z),u)\ Pf(x,f(y,f(z,u))), ...\}$ is infinite and the number of $\mathcal{T}_{\mathcal{D}}$-resolvents of two clauses is infinite, too.

Taking into account the tight connection between £-unification (see Biirckert, Herold & Schmidt-SchauB 1987) and theory resolution for equational theories, the previous examples' behaviour is not surprising, as the number of most general unifiers under the theory of commutativity is finite, whereas it is infinite under associativity (see Herold & Siekmann 1985).

5 Conclusion

In this chapter we want to show by means of an example, how the information about cycles in clause graphs can be used to reduce the search space in resolution theorem proving. The predicate logic formulation of SAM's lemma (without equality) leads to a clause set S, consisting of a set of the 11 units $\{U_1,...,U_{11}\}$

$\{min(0\ x\ 0),\ max\ (x\ 0\ x),\ max(a\ b\ d_1),\ min(d_1\ c_1\ 0),\ min(a\ b\ d_2),\ min(d_2\ c_2\ 0),\ min(c_2\ b\ e),\ min(c_2\ a\ f),\ max(c_1\ e\ g),\ max(c_1\ f\ h),\ \neg min(g\ h\ c_1)\}$

and the following 6 non unit clauses:

(C_1) $\neg min(x\ y\ z)\ min(y\ x\ z)$

(C_2) $\neg max(x\ y\ z)\ max(y\ x\ z)$

(C_3) $\neg min(x\ y\ u)\ \neg min(y\ z\ v)\ \neg min(x\ v\ w)\ min(u\ z\ w)$

(C_4) $\neg min(x\ y\ u)\ \neg min(y\ z\ v)\ \neg min(u\ z\ w)\ min(x\ v\ w)$

(C_5) $\neg max(x\ y\ z)\ min(x\ z\ x)$

(C_6) $\neg min(x\ z\ x)\ \neg max(x\ y\ x_1)\ \neg min(y\ z\ y_1)$
$\neg max(x\ y_1\ z_1)\ min(z\ x_1\ z_1)$

In the following we describe a refutation of this clause set using positive hyperresolution together with theory resolution, as in chapter 4. The cyclic clauses C_1 and C_2, describing the symmetry of the *min* and *max* predicates, are used for theory resolution, as in example 4.3.b). The same holds for the clause C5, which is neither self-resolving nor a member of a cycle. The clauses C3 and C4, describing the associativity of the *min* predicate, form a semicycle, which produces copies in the following way: Let (D_1, D_2, D_3) resolve with C_3 to the unit clause D_4. Then (D_1, D_2, D_4) resolves with C_4 to a copy of D_3. The resolution closure of the semicycle $\mathcal{D}=\{C_3, C_4\}$ being not finite, this cycle cannot be used directly for theory resolution. But C_3, C_4 can be used to produce instances of $\mathcal{T}_{\mathcal{D}}$ in the folllowing way:

from C_3, U_6, U_7: $\neg min(d_2\ e\ w)\ min(0\ b\ w)$
from C_4, U_6, U_7: $\neg min(0\ b\ w)\ min(d_2\ e\ w)$

In the following, we will abbreviate these two lines by:
from C_3, C_4, U_6, U_7: $min(d_2\ e\ w) \equiv min(0\ b\ w)$
If the theory of some clause C_n or E_m is used, then "with $T(C_n)$" or "with $T(E_m)$" is added to the from-line.

The result of resolving {c3,c4} with the two units is a complete cycle, which is added to the theory box.

Proceeding this way, only the clause c6 is needed to produce "ordinary" resolvents. Of course, there is also the possibility to produce copies of already retained cycles. Taking into account these reduncancies, a total 660 copies or instances of already present clauses are generated in the proof.

The proof of SAM's lemma consists of 8 steps:

from C_3, C_4, U_6, U_7, with $T(C_2, C_5)$:
E_1: $min(c_2\ b\ w) \equiv min(c\ d_1\ w)$
from C_6, U_7, U_9, U_4, U_2, with $T(E_1, C_1, C_2)$:
U_{12}: $min(d_1\ e\ g)$
from C_3, C_4, U_8, U_3, with $T(C_5)$:
E_2: $min(c_2\ a\ w) \equiv min(f\ d_1\ w)$
from C_3, C_4, U_8, U_5:
E_3: $min(c_2\ d_2\ w) \equiv min(f\ b\ w)$
from C_3, C_4, U_6, U_7, with $T(C_1, E_3)$:
E_4: $min(f\ e\ w) \equiv min(0\ c_1\ w)$
from C_3, C_4, U_8, U_{12}, with $T(E_3)$:
E_5: $min(f\ e\ w) \equiv min(f\ g\ w)$
from $C_6, U_9, U_{10}, U_1, U_2$, with $T(E_4, E_5, C_5)$:
U_{13}: $min(g\ h\ c_1)$

Clause U_{13} contradicts clause U_{11}, and the proof is complete.

Acknowledgement

I would like to thank my colleague Norbert Eisinger for thoroughly reading this manuscript. His comprehensive knowledge of the field, particularly about many examples and counterexamples that arise in theorem proving, resulted in improvements of the paper.

References

Bibel, W. (1981). On Matrices with connections. *Journal of the ACM* 28/4, 633 - 645.

Burckert, H.-J., Herold, A. & Schmidt-SchauB, M. (1987). On Equational Theories, Unification and Decidability, in:

Lescanne, P. (ed): Proc. of 2nd Conference on Rewriting Techniques and Applications, Bordeaux, France. Springer LNCS 256, 204 - 215.

Chang, C.L. & Lee, R.C. (1973). Symbolic Logic and Mechanical Theorem Proving. Academic Press. New York.

Eisinger, N. (1981). Subsumption and Connection Graphs. Proceedings of the 7th IJCAI, Vancouver, 480 - 486.

Eisinger, N. (1988). Completeness, Confluence, and Related Properties of Clause Graph Resolution. PhD thesis and SEKI-Report SR-88-07, Universitftt Kaiserslautern.

Guard, J. et al (1969). Semi-Automated Mathematics. Journal of the ACM. 16, 49 - 62.

Herold, A. (1983). Some Basic Notions of First Order Unification Theory. Internal Report, Universitftt Kaiserslautern.

Herold, A. & Siekmann, J. (1985). Unification in Abelian Semigroups. Memo-SEKI-85-III. Universitat Kaiserslautern.

Loveland, D.W. (1978). Automated Theorem Proving: A Logical Basis. North-Holland.

Ohlbach, H.J. (1988). Using Automated Reasoning Techniques for Deductive Databases. SEKI-Report SR-88-06, Universitat Kaiserslautern.

Overbeek, R. (1975). An Implementation of Hyper-Resolution. Computational Mathematics with Applications 1, 201 - 214.

Shostak, R.E. (1976). Refutation Graphs. Artificial Intelligence. III, 51 - 64.

Shostak, R.E. (1979). A Graph-Theoretic View of Resolution Theorem-Proving. Report SRI International, Menlo Park.

Socher, R. (1988). Reducing the Derivation of Redundant Clauses in Reasoning Systems. SEKI-Report SR-88-20, University of Kaiserslautern.

Stickel, M. E. (1985). Automated Deduction by Theory Resolution. Journal of Automated Reasoning. 1/4, 333 - 356.

Stickel, M. E. (1986). Schubert's steamroller problem: Formulations and Solutions. Journal of Automated Reasoning. 2/1, 89 - 102.

Vieille, L. (1987). Recursive Query Processing: The Power of Logic. ECRC Munich, Technical Report TR-KB-17.

Wos, L. et.al. (1967). The Concept of Demodulation in Theorem Proving. Journal of the ACM, 14, 698 - 709.

Wos, L. (1988). Automated Reasoning: 33 Basic Research Problems. Prentice Hall, Englewood Cliffs.