Flattening, Generalizations of Clauses and Absorption Algorithms

Shan-Hwei Nienhuys-Cheng
Department of Computer Science
Erasmus University
Rotterdam, the Netherlands

1 Introduction

In this article we will compare three algorithms for absorption. We call them MB-absorption[2], NCF-absorption[3,5] and RP-absorption[7]. Let two clauses $C_1=L_1\vee C_1'$ and $C_2=L_2\vee C_2'$, where L_1 and L_2 are literals, be given. If there exists a pair of substitutions (θ_1,θ_2) which is a most general unifier (mgu) of L_1 and $\sim L_2$, then $C=C_1'\theta_1\vee C_2'\theta_2$ is called the *resolvent* of C_1 and C_2, and L_1 and L_2 are the literals to be *resolved on*. Given clauses $C_1=L_1\vee C_1'$ and C, a V-operator is an algorithm to construct a $C_2=L_2\vee C_2'$ such that C becomes the resolvent of C_1 and C_2, and L_1, L_2 are literals to be resolved on. If L_1 is a positive literal, then we say that this V-operator is an *absorption*. All three absorptions yield a $C_2=L_2\vee C_2'$ such that there is a substitution (θ_1,θ_2) from C_1 and C_2, respectively, instead of a mgu, with the property $L_1\theta_1=\sim L_2\theta_2$. On the other hand, C can indeed be concluded as a result of C_1 and C_2 and if the resolvent of C_1 and C_2 is C', there is always a substitution μ such that $C'\mu=C$. An absorption algorithm which finds a C_2 by considering a unifier instead of the mgu is said to satisfy the *alternative soundness*. An absorption algorithm which can find all such C_2's is said to be *complete*. This article concerns in the first place the completeness of absorption that satisfies alternative soundness. We use here the usual clausal form instead of Horn clauses because we can then forget which one is the body and which one is the head and we can also easier generalize the whole concept to V-operators.

In section 2 we are going to give a short introduction to MB-absorption and NCF-absorption. NCF-absorption is an improvement of MB-absorption and it is complete, so we shall pay more attention to the question how to construct C_2 when C_1' is not empty. This question is mostly ignored in [3,5] because there C_1 is assumed to be a literal. NCF-absorption is divided in two stages. There is a *bridge clause* which connect these two stages. The second stage has only to do with finding all generalizations from this brige clause with term partitions. It can be done without considering resolutions or inverse resolutions [3,5,6]. The first stage has to do with how to construct the brige clause from C and a substitution from C_1.

In section 4 we investigate RP-absorption. RP-absorption uses the concept of *flattening* in absorption. Flattening changes all terms in a clause to variables in a special way. RP-absorption states the algorithm with the assumption that the clauses are already flattened. That RP-absorption does not work well is caused mostly by forgetting the original unflattened clauses. By investigating the absorption of RP we find some other interesting properties about flattening and flattened clauses. These properties are worth investigating by themselves and they help also to understand the problems of RP-absorption. Thus we discuss flattening and generalizations in section 3 as an separate topic. We summarize sections 3, 4 as follows:

(1) What is wrong with the RP's flattening algorithm?

(2) How to use the flattened clause of C to get all the generalizations of C? This algorithm can be compared with finding all generalizations with term partitions on C in [3,5,6]. If we have a *partially flattened* clause of C, then the number of generalizations induced by this clause is less than the number of generalizations induced by the flattened clause of C.

(3) Let C_f and D_f be flattened clauses of C and D, respectively. If μ is a substitution from C to D, what kind of substitution ∂ from the variables of C_f does μ induce? Is D_f the result of substitution ∂ from C_f ?

(4) The RP-absorption can be divided into two stages. The bridge between these two is a clause M which is called the intermediate clause. M is actually a partially flattened clause but not flattened as RP has suggested. We shall see this more clearly if we know more about (1) and (2). The first stage has also to do with a substitution θ_1 from C_{1f}, which is the flattened clause of C_1. What kind of substitution should θ_1 be is not told in RP and it is investigated in this article. The problems, that such a substitution can give, becomes clearer if we know more about (3).

(5) The second stage is a direct application of (2). The intermediate clause M can be compared with the corresponding bridge clause in NCF-absorption. This comparison helps us to know more about the completeness of RP.

(6) There are three important reasons which make RP not complete. Firstly, the flattening algorithm of RP is not all correct. Secondly, M is mostly not a flattened clause but a result of unification of some variables in a flattened clause. Thirdly, there is a limited choice for the part of M which has to do with C_1.

(7) We can thus improve RP-absorption in two ways. In the first way we keep essentially the algorithm and improve things which cause problems. Thus we use good flattening algorithm; construct the intermediate clause in a more reasonable way and give restriction to θ_1. This improved algorithm gives interesting C_2 but still not complete. The second way looks quite

different as the original RP but it is complete. Both ways have disadvantages. We compare these disadvantages with NCF-absorption.

2 MB and NCF absorptions

MB-absorption. Let C_1 be a positive literal. MB considers a subset TP' of the set of all term occurrences in $C \vee \sim C_1$. TP' is partitioned in blocks of B

$B=\{(r,p_1),\ldots,(r,p_n),(s,q_1),\ldots,(s,q_m)\}$

such that there is a substitution θ_1 from C_1 which satisfies $r\theta_1=s$. Let every such block correspond with a new variable and let C_2 be the result of replacing terms in $C \vee \sim C_1$ by this variable. This algorithm is incomplete and it can be seen from the following examples.

Example 1. Let $C_1=P(x)$, $C=Q(v,g(v))$. Then $C_2=Q(v,g(v)) \vee \sim P(h(v))$ is not to be found with MB. A block which contains x in C_1 should be changed to a variable and it can never be changed to h(v).

Example 2. Let $C_1=P(x,y)$ and $C=Q(u,f(w))$. Then $C_2=Q(u,f(w)) \vee \sim P(u,u)$ can not be found by MB. A block can never contain x,y from C_1 at the same time by definition. Different blocks go to different variables and $\sim P(u,u)$ of C_2 can not be constructed.

NCF-absorption. It is known that if the literal C_1 and the clause C_2 have resolvent C w.r.t. (θ_1,θ_2), then C_2 satisfies $C_2\theta_2=C \vee \sim C_1\theta_1$. We call $C \vee \sim C_1\theta_1$ a *bridge clause.* This terminology shall be used later when we want to compare it with the intermediate clause of RP-absorption. It is observed in [3,5] for constructing C_2, we should take partitions based on terms in $C \vee \sim C_1\theta_1$ instead of $C \vee \sim C_1$. To find such term partitions on $C \vee \sim C_1\theta_1$ and corresponding generalizations of $C \vee \sim C_1\theta_1$ has nothing to do with resolutions or inverse resolutions and thus NCF approachs this problem by defining term partitions on an arbitrary clause. The concept of term partitions is motivated by inverse substitution. For example, given $C=P(f(g(x)),g(x),g(x))$ and $C_1=P(f(u),g(u),v)$. The variable u in C_1 corresponds with the set B_u containing the first and the second g(x) in C and the variable v corresponds with the set B_v containing the third g(x) in C. Thus the two disjoint sets B_u, B_v form a partition on $B_u \cup B_v$. Term partitions can also be defined without inverse substitution. That means a term partition on a clause C can be constructed by finding suitable terms at suitable positions. The *positions* of term occurrences should firstly be defined formally by sequences of natural numbers and a term occurrence is determined by its position. For example, the first g(x) in the example C above has position <1,1> and the first x in C has position <1,1,1>. A term occurrence is a proper subterm occurrence of the other if the position of the first is longer than the position of the second. The *term partitions* can then be defined and determined unambigueusely. If we apply the theory of term partitions on a general

clause to the clause $C \vee \sim C_1\theta_1$ for inverse resolution, then we find all C_2 's which has to do with this θ_1. If we let θ_1 change, then all C_2's are found. Hence this algorithm is complete.
In [3,5] more is done than just finding all C_2's. Given any clause C, we can also define an partial order relation between term partitions on C. It can be proved being equivalent with the order relation between two clauses defined by substitution. The application of it on inverse resolutions is as follows. If we have two term partitions $\prod$, Ω on $C \vee \sim C_1\theta_1$ and if they induce clauses $C_2(\prod)$ and $C_2(\Omega)$, respectively, we can use the order relation in partitions to compare $\prod$ and Ω to know which one of their induced clauses is more general. That means we do not have to construct the clauses $C_2(\prod)$, $C_2(\Omega)$ and a substitution explicitly.
The theory about comparing term partitions and minimal higer partitions on an arbitray clause is investigated more in [4].

Improved NCF-algorithm. We can easily improve NCF-absorption by considering C_1 not necessarily to be a literal. Let $C_1=L_1 \vee C_1'$ and θ_1 be a substitution of C_1. Notice that if we are given C, then C usually does not contain the same literal more than once. If there is a $C_2=L_2 \vee C_2'$ and θ_2 from C_2 such that $L_1\theta_1=\sim L_2\theta_2$, then $C_1'\theta_1$, $C_2'\theta_2$ may both contain a maximal subclause C'. That means $C_2\theta_2= (C-C_1'\theta_1) \vee C' \vee \sim L_1\theta_1$. To find all such C_2 for a fixed θ_1 and C', we should find all term partitions of $(C-C_1'\theta_1) \vee C' \vee \sim L_1\theta_1$. Thus we have the new algorithm:
(1) Given θ_1 such that $C_1'\theta_1$ is a subclause of C, we consider a subclause C' of $C_1'\theta_1$.
(2) Find all term partitions on $(C-C_1'\theta_1) \vee C' \vee \sim L_1\theta_1$. Every partition induces a C_2 up to equivalence of variable names.
(3) Let C' change, then we find all C_2's which have to do with this particular θ_1.
(4) Let θ_1 change, we find all C_2's which satisfy the alternative soundness condition.
Thus this new algorithm is complete.

3 Flattening and generalizations

3.1 Flattening

Flattening is a way to change a clause to a clause with only variables. In [7] the authors present an algorithm for flattening. The steps they have presented are unclear and also not complete. With the steps they have presented, we would never get all the generalizations by unflattening. Let us look at their algorithm:
For each function f of arity n $f(t_1,\ldots,t_n)$ that appears in the program do 1,2, 3
1. Introduce a new predicate symbol f_p of arity n+1: $f_p(t_1,\ldots,t_n,x)$. The additional variable x represents the result of the function.

2 For each occurrence of f, that is for each term of the form $f(t_1,\ldots,t_n)$ that appears in a clause C of the program replace it by a new variable x and add $f_p(t_1,\ldots,t_n,x)$ in the body of C.
3 Add the fact $f_p(t_1,\ldots,t_n,f(t_1,\ldots,t_n))$ in order to define the predicate f_p. The clauses produced by this step will not be considered for inverse resolution.

In the same article it is added that for a constant, we should also define a predicate. For finding a generalization of a clause, the following comment is given:
With flattened clauses, turning a term into a variable is obtained by removing the literals corresponding to this term in the flat version of the clause, and by unifying the result variables.

With their flattening algorithm we have the following problems when we try to find all generalizations:
Problem. Let C=P(x,x). This is already made of variables so we can not use this algorithm to add new literals. How can we drop literals to find a generalization like P(x,y)? To be able to find P(x,y), we need a new predicate I as we have defined in the following new algorithm.

Flattening
Let a clause $C=C_0$ be given and let T be the empty clause.
(1) If a is a constant in C, then define a new predicate A such that A(a) is true.
(2) Define a predicate I (identity) which satisfies I(t,t) is true for every term t.
(3) If f is a function in C, then define a predicate F such that $F(t_1,\ldots,t_n,f(t_1,\ldots,t_n))$ is true.
(4) Let $p_1,\ldots,p_n$ be the positions of constants and variables in C. We process all of them in the following way. Let p_j be the first of these positions in C_i which is not yet processed before. If at p_i is a constant a, replace it by a new variable x not in C_i. The new clause is called C_{i+1} and let $T:=T\vee\sim A(x)$. Let $C_{i+1}':=C_{i+1}\vee T$. A(a) is called a *supplementary literal* and C_i' is a resolvent of C_{i+1}' and the literal A(a). The clause C_i can also be achieved by applying the substitution {x/a} to C_{i+1}. If at p_j is a variable x in C_i, replace it by z where z is a new variable. The new clause is called C_{i+1} and let $T:=T\vee\sim I(x,z)$. For both situations we define $C_{i+1}'=C_{i+1}\vee T$. Then C_i' is a resolvent of C_{i+1}' and I(x,x). I(x,x) is called a *supplementary literal.* C_i can also be achieved by applying the substitution {z/x} to C_{i+1}. We repeat step (4) until all constants and variables in C are processed.
(5) Find the first compound term $t=f(x_1,\ldots,x_n)$ in C_i where x_i are variables. We replace this $f(x_1,\ldots,x_n)$ in C_i by a new variable z and we call it C_{i+1}. Let $T:=T\vee\sim F(x_1,\ldots.,x_n,z)$. Define $C_{i+1}'=C_{i+1}\vee T$. Let $F(x_1,\ldots,x_n,f(x_1,\ldots,x_n))$ be also called a *supplementary literal.* Then C_i is a resolvent of C_{i+1} and the supplementary literal. C_i can also be achieved by applying $\{z/f(x_1,\ldots,x_n)\}$ to C_{i+1}. We repeat this step until we do not find any more compound terms in C_k for

certain k. Then C_k' is called the *flattened clause* of C. If S is the conjunction of all supplementary literals so far, then C is equivalent with $C_k'\wedge S$. S is called the *supplementary clause* of C.

Example. Let $C=C_0=P(f(x,g(y)),h(x))$.

C_1': $P(f(x_0,g(y_0)),h(x_1))\vee{\sim}I(x,x_0)\vee{\sim}I(y,y_0)\vee{\sim}I(x,x_1)$,

$C_1=P(f(x_0,g(y_0)),h(x_1))$, $S_1=I(x,x_0)\wedge I(y,y_0)\wedge I(x,x_1)$

C_2': $P(f(x_0,z),h(x_1))\vee{\sim}G(y_0,z)\vee{\sim}I(x,x_0)\vee{\sim}I(y,y_0)\vee{\sim}I(x,x_1)$,

$C_2=P(f(x_0,z),h(x_1))$, $S_2=G(y_0,g(y_0))\wedge I(x,x_0)\wedge I(y,y_0)\wedge I(x,x_1)$

C_3': $P(u,h(x_1))\vee{\sim}F(x_0,z,u)\vee{\sim}G(y_0,z)\vee{\sim}I(x,x_0)\vee{\sim}I(y,y_0)\vee{\sim}I(x,x_1)$, $C_3=P(u,h(x_1))$,

$S_3=F(x_0,z,f(x_0,z))\wedge G(y_0,g(y_0))\wedge I(x,x_0)\wedge I(y,y_0)\wedge I(x,x_1)$

C_4': $P(u,w)\vee{\sim}H(x_1,w)\vee{\sim}F(x_0,z,u)\vee{\sim}G(y_0,z)\vee{\sim}I(x,x_0)\vee{\sim}I(y,y_0)\vee{\sim}I(x,x_1)$, $C_4=P(u,w)$,

$S=S_4=H(x_1,h(x_1))\wedge F(x_0,z,f(x_0,z))\wedge G(y_0,g(y_0))\wedge I(x,x_0)\wedge I(y,y_0)\wedge I(x,x_1)$.

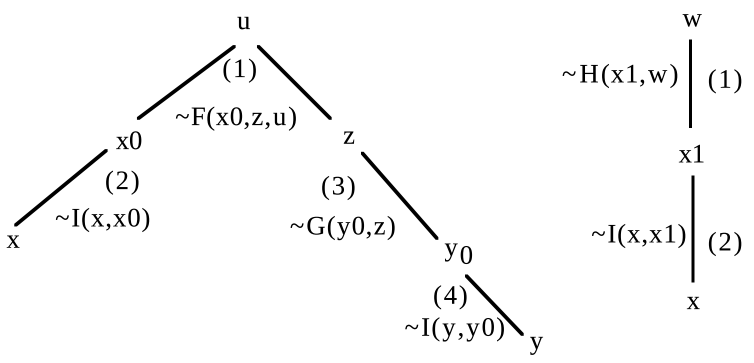
C_4 ' is the flattened clause of C and S is the supplementary clause. Notice that all the rightmost argument in the extra literals defined by terms are different. Notice also that every varible in C is replaced by a variable in the rightmost argument of literals begining with ~I. We have changed all the names of variables because we can then easier define the algorithm of tree substitution (see remark 1) and step 3 of the algorithm for generalizations.

Remark 1. Let C be a given clause and C_f be the flattened clause of C. Let T be the set of all extra literals in C_f defined by term occurrences in C. Notice that every variable in C_f beside the original variables in C appears precisely once as the rightmost argument of a literal in T. If u is such a variable in C_f then we use V_u to represent the set of variables in C_f which are related to u, i.e.

1) Initialize $V_u:=\{u\}$.

2) if there is an $\sim H(x_1,\ldots,x_n,u)$ in T and $H\neq I$, define $V_u:=\{u\}\cup V_{x1}\cup\ldots\cup V_{xn}$. where $n\geq 0$. If H=I, then define $V_u=\{u\}$.

We can find the original term occurrence back by substituting the variable by using the supplementary literals. We take the same example as above. The number in the tree tells the order of substituting:



Thus $u \rightarrow f(x_0,z) \rightarrow f(x,z) \rightarrow f(x,g(y_0)) \rightarrow f(x,g(y))$ and $w \rightarrow h(x_1) \rightarrow h(x)$. Let us call such way of finding the original term back the *tree substitutions.*

Definition. Let C_f be the flattened clause of C. A *term literal* is a literal in C_f which is defined by a term in C. An *important variable* of C_f is defined as a variable which is arightmost argument of a term literal.

3.2 Finding generalizations with unflattening

According to [7], *turning a term into a new variable is obtained by removing the literals corresponding to this term in the flat version of the clause, and by unifying the result variables.* What they mean can be seen with the following example:

Example. The flatted version of $C=P(f(x),f(x))$ is

C': $P(u,w) \vee \sim F(x_1,u) \vee \sim F(x_0,w) \vee \sim I(x,x_1) \vee \sim I(x,x_0)$,

If we drop $\sim F(x,u) \vee \sim F(x_1,w)$ and unify u, w we get $P(u,u) \vee \sim I(x,x_1) \vee \sim I(x,x_0)$. By resolution (unflattening) we get $P(u,u)$. Notice that if we would delete $\sim I(x,x_1) \vee \sim I(x,x_0)$ as well, it would save us the work of resolution. We shall discuss this and other problems of their description below.

Problem 1. What they say sounds as if all the term occurrences of the same term should be turned to only one variable. The example they give indicates also this special situation. This is of course not true. For example, P(a,a) can be generalized to P(x,x), P(x,a) and P(a,y) and P(x,y). In fact, we can find generalizations by changing a subset of all term occurrences of the same term into a new variable.

Problem 2. Suppose we want to drop a few literals which correspond to terms and unify the corresponding variables, we have first to know if these variables represent the same term in the original clause. In [7] this condition is not mentioned. We intend also to forget this problem when we look at the simple examples they have given. Now we give a new example.

Example. Given $C=P(f(g(a)),f(g(x)))$. Then the flattened clause is

$$C_f=P(u,v) \vee \sim F(y,u) \vee \sim F(z,v) \vee \sim G(x_0,z) \vee \sim G(w,y) \vee \sim I(x,x_0) \vee \sim A(w).$$

How do we know u can not be unified with v if we do not apply the tree substitutions which we have defined above to u, v? This is still a simple example. If u and v are terms with complicated structure, it is even more difficult. There is another way to remedy it than tree substitutions: we keep track of terms and variables representing the terms in every stage of flattening. This shall save the trouble of tree substitutions. This is not directly applicable in the RP-algorithm because their intermediate clause is a combination of two flattened clauses and it is in general not flat.

Problem 3. It is not clear what the order of actions in dropping literals, unifying the variables and using resolution is.

Problem 4. Let us consider the example in problem 2 again. According to RP changing y and u to variables means to drop ~G(w,y) and ~F(y,u). This does not turn y into a new variable because y represents actually a subterm occurrence of u in the original clause.

From all these problems stated above, we know that the RP-method of unflatting to generalize a clause needs to be modified.

Algorithm 1 for finding generalizations from a flattened clause

Given a clause C, we can find all generalizations of it by the following algorithm.

(1) Firstly flatten the clause C to C_f. Let $C_f=D\vee T$ where T is the part in C_f which contains the extra literals defined by terms.

(2) We choose $u_1,u_2,\ldots,u_n$ from the variables which are the rightmost arguments in T and which satisfiy the condition $u_i\notin V_{uj}$ for $i\neq j$. Remove the literals which appear in the substitution trees of all u_i's.

(3) Divide these u_i 's in a partition of blocks which satisfies the following condition: if $\{u_{i1},\ldots,u_{ik}\}$ is a block then the tree substitutions of all u_{ij} for j=1,…,k give the same term. Change these $u_{i1},\ldots,u_{ik}$ to u_{i1} (or u_{i2},etc.).

(4) Apply resolutions by considering other literals in T and their corresponding supplementary literals. The order of resolutions is determined by the following condition: if $\sim H(x_1,\ldots,x_n,u)$ is a literal such that u is not an argument in another literal, then we may apply resolution to this literal and its corresponding supplementary literal.

Example. Given a clause C=P(f(g(x)),g(x)). The flattened clause of C is

$$C_f=P(u,v)\vee\sim I(x,x_0)\vee\sim I(x,x_1)\vee\sim G(x_0,y)\vee\sim G(x_1,v)\vee\sim F(y,u).$$

1) If we drop nothing and we apply direct resolutions with the following order: $\sim G(x_1,v)$, $\sim I(x,x_1)$, $\sim F(y,u)$, $\sim G(x_0,y)$, $\sim I(x,x_0)$, then we get C back.

2) Choose x_1 for step (2). x_1 form its own block. Drop thus $\sim I(x,x_1)$. We have

$$P(u,v)\vee\sim I(x,x_0)\vee\sim G(x_0,y)\vee\sim F(y,u)\vee\sim G(x_1,v).$$

With resolutions we get $P(f(g(x)),g(x_1))$.

3) Choose y, v for step (2). By tree substitutions we can see y=g(x) and $v=g(x_1)=g(x)$. Thus we can consider them to be in the same block. After dropping literals and unifying v and y

$$P(u,y)\vee\sim F(y,u).$$

By applying resolution we get P(f(y),y)

4) Choose y,v again for step (2). This time {y} and {v} are different blocks. After dropping literals we have

$P(u,v) \vee \sim F(y,u)$

By applying resolutions we get

$P(f(y),v)$

5) Choose u,v for stap (2). Let {u}, {v} be two blocks. By dropping all literals we get

$P(u,v)$

For every term partition on C and a generalization defined by it, we can apply the method defined above to find the same generalization. Thus this method is complete (i.e. it can find all generalizations) because the method with term partitions is comlete.

Example. Consider the same clauses $C=P(f(g(x)),g(x))$ and

$C_f=P(u,v) \vee \sim I(x,x_0) \vee \sim I(x,x_1) \vee \sim G(x_0,y) \vee \sim G(x_1,v) \vee \sim F(y,u)$.

The term partition with only one block $B=\{(g(x),<1,1>),\ g(x),<2>)\}$ corresponds with the variables y and v in 3) and this partition generalizes the same clause as $P(f(y),y)$.

The term partition with two blocks $B_1=\{(x,<1,1,1>)\}$ and $B_2=\{(x,<2,1>\}$ corresponds with the the variable x_1 and the variable x in 2).

Remark. Notice that there are two ways to get a generalization like $P(f(g(x)),g(x_1))$. One is illustrated as 2). The other way is to choose x_0 and x_1 for step (2) and drop both $\sim I(x,x_0)$ and $\sim I(x,x_1)$. Let $\{x_0\}$ and $\{x_1\}$ be blocks, then we get $P(f(g(x_0)),g(x_1))$ as the generalization. Let us use another example. If we consider $\{x_0,x_1\}$ as a block and we use x_0 for both variables, then we get $P(f(g(x_0)),g(x_0))$. This is equivalent with what we find in 1). Thus we can change the algorithm in the following way:

Algorithm 2 for finding generalizations from a flattened clause

Given a clause C, we can find all generalizations of it by the following algorithm.

(1) Firstly flatten the clause C to C_f. Let $C_f=D \vee T$ where T is the part in C_f which contains the extra literals defined by terms.

(2) We choose $u_1,u_2,\ldots,u_n$ from the variables which are the rightmost arguments in T and which satisfiy two conditions: firstly, for every rightmost variable x_i in a literal beging with ~I, there is a u_j such that $x_i \in V_{uj}$; secondly, $u_i \notin V_{uj}$ for $i \neq j$. Remove the literals which appear in the substitution trees of all u_i's. Notice that this step makes all ~I literals disappear.

(3) Divide these u_i 's in a partition of blocks which satisfies the following condition: if $\{u_{i1},\ldots,u_{ik}\}$ is a block then the tree substitutions of all u_{ij} for $j=1,\ldots,k$ give the same term. Change these $\{u_{i1},\ldots,u_{ik}\}$ to a u_{i1} .

(4) Apply resolutions by considering other literals in T and their corresponding supplementary literals. The order of resolutions is determined by the following condition: if $\sim H(x_1,\ldots,x_n,u)$ is a literal such that u is not an argument in another literal, then we may apply resolution to this literal and its corresponding supplementary literal.

Notice that algorithm 2 gives a more uniform way in finding new variables for a generalization. In fact, step (3) of this algorithm gives all variables of the generalization. Step (4) finds the term occurrences of the generalization which have either some of $u_1,\ldots,u_n$ as arguments or they are constants. With algorithm 1 we get variables in the generalization in two ways, namely, (3) and part of (4). Thus with algorithm 1 we can sometimes still get the variables which are in the original clause C, for example, the x of $P(f(g(x)),g(x_1))$ in the remark. With algorithm 2 we can only get u_i's for variables of the generalization, for example, $P(f(g(x_0)),g(x_1))$ in the same remark. In fact, with algorithm 2 we can also easier formulate the proof of the correspondence between term partitions on C and such generalizations from the flattened clause of C.

Comparison between finding generalizations with unflattening and term partitions.

We have seen how to find generalizations of a clause C with its flattened clause. If we have a clause C, we should first find the flattened clause C_f of C. This means we have to change every term in C by taking a number of steps such that it becomes a variable in the end. For every such step we have to still add a literal to the clause. To get the generalization we should first find a set of a variables which satisfies certain conditions and we can then make a partition of these variables. Two variables are allowed in the same block only if they can become the same term after tree substitution. The next step is to drop the literals which correspond to these variables and also to drop the literals which come in the tree substitution. Hence the whole process involves so many changing terms to variables and then changing them back. This process is very complicated compared to the term partitions. For term partitions we need only to change every term occurrence in a block to a variable.

The generalization with term partitions [3,5] gives for every term occurrence in the clause C a position. This saves the effort of turning first the term occurrences into variables and adding literals. We choose then a subset of term occurrences where no two positions of the occurrences have a subsequence relationship, i.e. no one occurrence is a subterm occurrence of the other. This is equivalent to $u_j \notin V_{ui}$ in step (2) in the above algorithm, but finding V_{ui} gives more trouble and it is not so neat as comparing the sequences which represent positions.

3.3 What kind of transformations between flattened clauses are induced by substitutions?

Let C_f and D_f be flattened clauses of C and D, respectively. If μ is a substitution from C to D, what kind of substitution ∂ from the variables of C_f does μ induce? Is D_f the result of this substitution ∂ from C_f ?

Example. Let us consider $\mu=\{x/g(y))$ from $C=P(f(x))$ to $D=P(f(g(y)))$. The flattened clauses are thus $C_f=P(u)\vee\sim F(x_0,u)\vee\sim I(x,x_0)$ and $D_f=P(v)\vee\sim F(z,v)\vee\sim G(y_0,z)\vee\sim I(y,y_0)$. The g(y) in D is represented by z. If we consider the substitution $\{x_0/z\}$ from C_f we have $P(u)\vee\sim F(z,u)\vee\sim I(x,z)$ as the result which is quite different from D_f. We can improve $\{x/z\}$ by using $\{u/v,x_0/z\}$. By applying this substitution to C_f we get $P(v)\vee\sim F(z,v)$ and $\sim I(x,z)$. The first part is actually a subclause of D. That means a substitution from C to D induces a substitution which is not necessary a substitution from C_f to D_f.

Algorithm for finding induced substitution. Let μ be a substitution from C to D where C, D are assumed to be atoms for simplicity. Let C_f and D_f be the flattened clauses of C and D, respectively. The substitution ∂ induced by μ can be described as follows:

(1) Compare the variables in $C_f=P(u_1,\ldots,u_n)$ and $D_f=P(v_1,\ldots,v_n)$. Initialize $\partial=\{u_1/v_1,\ldots,u_n/v_n\}$

(2) If $\sim H(x_1,\ldots,x_i,u_j)$ is an extra literal in C_f defined by a term and $H\neq I$, then there is also $\sim H(y_1,\ldots,y_i,v_j)$ in D_f. We extend ∂ by $\{x_1/y_1,\ldots,x_i/y_i\}$.

(3) We repeat the process in (2) until we have found a substitution for every variable which is rightmost argument of extra literals in C_f.

If we divide C_f in two parts, one part is the subclause T_1 of all literals in C_f which does not begin with ~I. The other part T_2 is the subclause which contains literals begining with ~I. We can prove that the induced ∂ brings T_1 to a subclause of D_f. Furthermore, if $\sim I(u,u_1),\ldots,\sim I(u,u_n)$ are in the clause C_f then the $u_1\partial,\ldots,u_n\partial$ have the same tree substitutions.

Example. Let $C=P(f(x),f(x))$ and $D=P(f(g(y,a)),f(g(y,a)))$. Then

$C_f=P(u,v)\vee\sim I(x,x_0)\vee\sim I(x,x_1)\vee\sim F(x_0,u)\vee\sim F(x_1,v)$.

$D_f=P(u,v)\vee\sim I(y,y_0)\vee\sim I(y,y_1)\vee\sim A(z)\vee\sim A(z_1)\vee\sim G(y_0,z,w)\vee\sim G(y_1,z_1,w_1)\vee\sim F(w,u)\vee\sim F(w_1,v)$

Notice that $\partial=\{u/u,v/v,x_0/w,x_1/w_1\}$ brings the subclause T_1 of C_f to a subclause of D_f. Consider $x_1\partial=w_1$ and $x\partial=w$. The tree substitutions of w and w_1 both give the same term g(y,a).

Definition. Let C_f and D_f be the flattened clauses of C and D, respectively. Let C_f be divided into two subclauses T_1 and T_2 as above. If there is a substitution ∂ from the variables of C_f such that $T_1\partial$ is a subclause of D_f and for every $\sim I(u,u_1),\ldots,\sim I(u,u_n)$ in T_2, the tree substitutions of $u_1\partial,\ldots,u_n\partial$ are the same, then we call ∂ a *transformation* from C_f to D_f.

Theorem. Let C_f and D_f be the flattened clauses of C and D, respectively. Let ∂ be a transformation from the clause C_f to D_f. Then there is a substitution μ from C to a subclause of D which induces ∂. The converse is also true.

The following example, theorem and corollary perhaps do not make much sense now. We want actually to use them in the comparisof NCF- and RP-algorithms later. They can be used to show that all the generalizations of the intermediate clause of RP-absorption are also generalizations of the corresponding bridge clause of NCF-absorption.

Example. Let $C=\sim P(x)\vee\sim Q(f(x))$ and let $D=R(a)\vee\sim Q(f(a))$. Then

$C_f=\sim P(x_0)\vee\sim Q(z)\vee\sim F(x_1,z)\vee\sim I(x,x_0)\vee\sim I(x,x_1)$,

$D_f=R(y_0)\vee\sim Q(u)\vee\sim F(y_1,u)\vee\sim A(y_0)\vee\sim A(y_1)$.

We notice the following:

1) Consider a substitution $\mu=\{x/a\}$ from $\sim P(x)\vee\sim Q(f(x))$ to a subclause of $E=\sim P(a)\vee\sim Q(f(a))\vee R(a)$. This induces a transformation $\theta=\{x_0/y_0,x_1/y_1,z/u\}$ from C_f to

$E_f=\sim P(y_0)\vee\sim Q(u)\vee\sim F(y_1,u)\vee\sim A(y_0))\vee\sim A(y_1)\vee R(y_2)\vee\sim A(y_2)$.

(We shall see that this clause corresponds with the flattened clause of a bridge clause in NCF.)

2) Notice that the subclause $\sim Q(z)\vee\sim F(x_1,z)\vee\sim I(x,x_1)$ of C_f is transformed to D_f by θ, or preciser, $\{x_1/y_1,z/u\}$

3) Notice that the following (not flattened) clause which is induced by D_f and the transformation θ from $\sim P(x_0)\vee\sim I(x,x_0)$,

$\sim P(y_0)\vee\sim Q(u)\vee\sim F(y_1,u)\vee\sim A(y_0)\vee R(y_0)\vee\sim A(y_1)$

is a partially flattened clause of E and it can also be used for finding some general[illegible]ons.

Theorem. Let C and D be given [illegible]ses. [illegible] C_f and D_f the flattened claus[illegible]f C [illegible] D respectively. Let K be the set of al[illegible]rals [illegible]f which begin with ~I and let [illegible]t of all rightmost arguments of literals in [illegible] Let [illegible] a substitution from the variables to variables which brings the variables in [illegible]bles in D_f and suppose θ satisfies the following conditions: if $u\theta=v$ where v is a variable in D_f, then every literal which does not begin with ~I in the substitution tree of u is brougt to a literal in D_f; if $\sim I(x,x_1),\ldots,\sim I(x,x_n)$ are in the C_f, then $x_1\theta,\ldots,x_n\theta$ give the same term. Then there is a substitution μ from C to D such that $(C_f\text{-}K)\theta\vee D_f$ is a partially flattened clause of $C\mu\vee D$.

Corollary. Let C=L$\vee$C' where L is a literal and D be given clauses. Let C_f and D_f the flattened clause of C and D respectively. Let $C_f=U\vee V$ where U is the part of C_f which has to do with L and V is the part of C_f which has to do with C'. Let K be the subclause of U which contains the literals begining with ~I. Let θ be a substitution from the variables in C_f (rightmost arguments) which satisfies the following two conditions:

1) if $u\theta=v$ where v is a variable in D_f, then every literal in the substitution tree of uwhich does not begin with ~I is brougt to a literal in D_f and furthermore, if ~I(x,x_1),…,~I(x,x_n) are in K, then $x_1\theta,\dots,x_n\theta$ give the same term.

2) θ induces a transformation from V to D_f.

Then there is a substitution μ from C such that C'μ is a subclause of D and $(U\text{-}K)\theta\vee D_f$ is a partially flattened clause ofof L$\mu\vee$D

3.4 Partially flattened clauses defined by a substitution

Definition. Let C and D be two cluases and μ be a substitution from C. If ∂ is the transformation defined by μ, then $C_f\partial\vee D_f$ is called a *partially flattened clause* of C$\mu\vee$D. We can define generalizations based on $C_f\partial\vee D_f$ the same way as we do with $(C\mu\vee D)_f$. In fact, the generalizations based on $C_f\partial\vee D_f$ is a generalization of C$\mu\vee$D.

(1) We choose important $u_1,u_2,\dots,u_n$ from the partially flattened clause $C_f\partial\vee D_f$. which satisfiy two conditions: firstly, for every important x_i in a literal begining with ~I, there is a u_j such that $x_i\in V_{uj}$; secondly, $u_i\notin V_{uj}$ for $i\neq j$. Remove the literals which appear in the substitution trees of all u_i's. Notice that this step makes all ~I literals disappear.

(2) Divide these u_i 's in a partition of blocks which satisfies the following condition: if $\{u_{i1},\dots,u_{ik}\}$ is a block then the tree substitutions of all u_{ij} for j=1,…,k give the same term. Change these $\{u_{i1},\dots,u_{ik}\}$ to a u_{i1} .

(3) Apply resolutions by considering other term literals and their corresponding supplementary literals. The order of resolutions is determined by the following condition: if ~H$(x_1,\dots,x_n,u)$ is a literal such that u is not an argument in another literal, then we may apply resolution to this literal and its corresponding supplementary literal.

Example. Let C=~P(x)$\vee$~Q(f(x)) and let D=R(a)$\vee$~Q(f(a)). Then

C_f=~P$(x_0)\vee$~Q(z)$\vee$~F$(x_1,z)\vee$~I$(x,x_0)\vee$~I(x,x_1),

D_f=R$(y_0)\vee$~Q(u)$\vee$~F$(y_1,u)\vee$~A$(y_0)\vee$~A(y_1).

We notice the following:

1) Consider a substitution $\mu=\{x/a\}$ from $\sim P(x)\vee\sim Q(f(x))$ to $\sim P(a)\vee\sim Q(f(a))$. We have then a clause $C\mu\vee D=\sim P(a)\vee\sim Q(f(a))\vee R(a)$.

2) $(C\mu\vee D)_f=\sim P(y_0)\vee\sim Q(u)\vee\sim F(y_1,u)\vee\sim A(y_0))\vee\sim A(y_1)\vee R(y_2)\vee\sim A(y_2)$.

3) The substitution μ induces a substitution $\partial=\{x0/y0,x1/y1,u/u\}$. This is a transformation from Cf to Df. Consider

$$C_f\partial\vee D_f =\sim P(y_0)\vee\sim Q(u)\vee R(y_0)\vee\sim A(y_0)\vee\sim A(y_1)\vee\sim F(y_1,u).$$

This clause gives generalizations:

1) By dropping $\sim A(y_0)$ and $\sim A(y_1)$ and let $\{y_0,y_1\}$ be in one block we have

$$\sim P(y_0)\vee\sim Q(u)\vee R(y_0)\vee\sim F(y_0,u).$$

The resolutions give $\sim P(y_0)\vee\sim Q(f(y_0))\vee R(y_0)$.

2) By dropping $\sim A(y_0)$ and $\sim A(y_1)$ and let y_0,y_1 be in apart blocks we have

$$\sim P(y_0)\vee\sim Q(u)\vee R(y_0)\vee\sim F(y_1,u).$$

The resolutions give $\sim P(y_0)\vee\sim Q(f(y_1))\vee R(y_0)$.

3) The other three possible generalizations of $C\mu\vee D$ based on $C_f\partial\vee D_f$ are

$$\sim P(y_0)\vee\sim Q(u)\vee R(y_0),\ \sim P(a)\vee\sim Q(f(y_1))\vee R(a),\ \sim P(y_0)\vee\sim Q(f(a))\vee R(y_0).$$

The generalizations like $\sim P(y_0)\vee\sim Q(f(y_1))\vee R(y_1)$ or $\sim P(y_0)\vee\sim Q(f(y_1))\vee R(y_2)$, etc. of $C\mu\vee D$ can not be found by using $C_f\partial\vee D_f$. With this partially flattened clause we can only find generalizations which yields the argument of ~P the same as the argument of R.

Remark. Let C and D be given clauses and μ be a substitution. Let ∂ be the substitution induced by μ. The clause $(C\mu\vee D)_f$ gives all the generalizations of $C\mu\vee D$ but the clause $C_f\partial\vee D_f$ not. Because $C_f\partial$ is a subclause of $C\mu$, thus the difference between $C_f\partial$ and $(C\mu)_f$ are in two respects:

1) There are only corresponding variables in $C_f\partial$ for the terms in C or in D.

2) The variables in $C_f\partial$ can be in the part of $(C\mu\vee D)_f$ which is induced by D. Thus if a variable u is in both parts $C_f\partial$, D_f and if they correspond to u_1, u_2 in $(C\mu\vee D)_f$, then the generalizations which are constructed by the first clause shall never have two different terms in u_1 and u_2.

Example. Given a clause $C=P(f(g(x)),g(x))$. The flattened clause of C is

$$C_f=P(u,v)\vee\sim I(x,x_0)\vee\sim I(x,x_1)\vee\sim G(x_0,y)\vee\sim G(x_1,v)\vee\sim F(y,u).$$

Define C1 by unifying y and v, then we have

$$C1=P(u,u)\vee\sim I(x,x_0)\vee\sim G(x_0,y)\vee\sim F(y,u)$$

Algorithm for finding induced substitution. Let μ be a substitution from C to D where C, D are assumed to be atoms for simplicity. Let C_f and D_f be the flattened clauses of C and D, respectively. The substitution ∂ induced by μ can be described as follows:

(1) Compare the variables in $C_f=P(u_1,\ldots,u_n)$ and $D_f=P(v_1,\ldots,v_n)$. Initialize $\partial=\{u_1/v_1,\ldots,u_n/v_n\}$

(2) If $\sim H(x_1,\ldots,x_i,u_j)$ is an extra literal in C_f defined by a term and $H \neq I$, then there is also $\sim H(y_1,\ldots,y_i,v_j)$ in D_f. We extend ∂ by $\{x_1/y_1,\ldots,x_i/y_i\}$.

(3) We repeat the process in (2) until we have found a substitution for every variable which is rightmost argument of extra literals in C_f.

4 RP-absorption

4.1 RP-absorption

The first stage of RP-absorption does the following. For given C and C_1, it first flattens the C and C_1 to C_{1f} and C_f. RP considers then a substitution θ_1 from C_{1f}. Combining $C_{1f}\theta_1$ and C they construct an intermediate clause M which looks likes the bridge clause in step (2) in the NCF-algorithm. The second stage does generalizations of M. As we know that NCF is complete and the the algorithm for finding generalizations from flattened clauses is also complete, the problem is M. How far is M from a flattened clause? How far is M from the bridge clause in step (2) in NCF? We are going to discuss the following problems of RP-absorption.

Problem 1. Because the flattening algorithm of RP is not complete so C_f and C_{1f} are also not good enough. From such M we can not find all nice C_2 ‘s.

Problem 2. The conditions for θ_1 stated by RP cause two kinds of problems. One kind shall be explained later and one kind has to do with the difference between transformation and substitution from one clause to other clause.

Problem 3. M looks like a flattened clause but it is often not a flattened because it contains the same variables as the rightmost argument in the extra literals sometimes more than once. Thus we can not get all generalizations we want from such M's in the second stage. Moreover, even we do not require M to be flattened, we have still limited choice. The reason is that the part of M which has to do with C_{1f} has to satisfies some restrictions.

Problem 4. Let $C_1=C_1'\vee L_1$ and $C_2=C_2'\vee L_2$ and let (μ_1,μ_2) be an unifier of L_1 and L_2, then $L_1\mu_1$ and $\sim L_2\mu_2$ can have a subclause in common. This problem has also been mentioned in NCF and we can also improve RP in the same way as we do with NCF.

The *algorithm given by RP* supposed that the clauses are flattened. Here is their formulation.

Given two clauses

C_1: $H_1 \leftarrow \alpha$ and R: $H \leftarrow \gamma$, do 1, 2, 3

1. Find θ_1 such that $\gamma = a \wedge \alpha\theta_1$

2. *Build the intermediate clause*

   $M: H \leftarrow a \wedge H_1\theta_1$

3. *Find a substitution* θ_2 *such that*

   $H=H_2\theta_2$ *and* $a=\beta\theta_2$ *and* $H_1\theta_1 = A\theta_2$.

4 $Abs(C_1,R)$: $H_2 \leftarrow A \wedge \beta$

We can formulate their algorithm with clausal form and with consideration of flattening and unflattening as follows:

RP-algorithm

Let C and $C_1=L_1 \vee C_1'$ be given. We do the following steps:

(1) Flatten the clause C to C_f and C_1 to $C_{1f}=D \vee E$ where $D=L_1' \vee D'$ is the part of C_{1f} which has to do with L_1, i.e. the literal L_1' is found by changing terms in L_1 to variables and D' is the subclause of C_{1f} consisting of extra literals from the terms in L_1. Furthermore, E is the part of C_{1f} induced by C_1'. Let S_1 be the supplementary clause of C_{1f} and S be the supplementary clause of C_f.

(2) Let θ_1 be a substitution from C_{1f}. If $(D' \vee E)\theta_1$ is a subclause of C_f, then we can construct an intermediate clause $M=(C_f-(D' \vee E)\theta_1) \vee \sim L_1'\theta_1$.

(3) Choose one way to generalize the clause M, by using $S_1\theta_1 \wedge S$. We get one C_2.

Suppose $C_1=L_1 \vee C_1'$ and $C_2=L_2 \vee C_2'$. It is possible for given unifier (μ_1,μ_2) of L_1 and L_2, $C_1'\mu_1$ and $C_2'\mu_2$ have subclause in common. Thus we can understand the following example which has to do with problem 4 above.

Example. Let $C_1=P(x,y) \vee \sim R(x,y)$ and $C=Q(x,y) \vee \sim R(f(x),f(y))$. We get (with their way of flattening)

$C_{1f}=P(x,y) \vee \sim R(x,y)$

$C_f=Q(x,y) \vee \sim R(u,v) \vee \sim F(x,u) \vee \sim F(y,v)$

Let $\theta_1=\{x/u,y/v\}$, then $M=Q(x,y) \vee \sim F(x,u) \vee \sim F(y,v) \vee \sim P(u,v)$.

We can never get $C_2=Q(x,y) \vee \sim R(f(x),f(y)) \vee \sim P(f(x),f(y))$ by unflattening.

For simplicity we are going to consider only the C_2 such that $C_2\mu_2$ contains all C in the rest of the article beside the part of new RP-algorithm.

The following example is given by RP[7]. Although with their algorithm we can find some interesting C_2, but not all interesting C_2. This has to do with their flattening algorithm not complete and also other problems mentioned above.

Example of RP-absorption. In arithmetic we know x is less than its successor and x is also less than the successor of successor of x. We want to use absorption to find new concepts such as: if x is less than y, then x is less than the successor of y. In our notation L means *less*, s(x) means x+1. The way RP has used to solve this problem is first given below ((1)-(4)'). We shall then investigate their steps.

(1) Given C_1=L(v,s(v)) and C=L(u,s(s(u))).

(2) C_{1f}=L(v,x)∨~S_p(v,x), with S_p induced by function s and supplementary S_p(v,s(v)).
C_f=L(u,x)∨~S_p(u,y)∨~S_p(y,x), supplementary S=S_p(u,s(u))∧S_p(y,s(y))

(3) Consider θ_1={v / u,x / y}.
Then M=L(u,x)∨~S_p(y,x)∨~L(u,y), supplementary S_p(u,s(u))∧S_p(y,s(y)).

(4) Unflatten with S_p(y,s(y)): C_2=L(u,s(y))∨~L(u,y). It means if u is less than y, then it is less than s(y)=y+1.

(3)' Consider θ_1'={v / y}. We have then M=L(u,x)∨~S_p(u,y)∨~L(y,x)

(4)' Unflatten by using S_p(u,s(u)): C_2=L(u,x)∨~L(s(u),x). It means if s(u) is less than x, then u is less than x.

Discussion of the example. For simplicity we use D_{1f} or D_f to name the flattened clauses of C_1 and C according to our algorithm.

Stap (2): According to our definition of flattening (for simplicity we do not use ~I(v,v_0)) C_{1f} and C_f should be

D_{1f}=L(v,x)∨~S_p(v_1,x)∨~I(v,v_1)

D_f=L(u,x)∨~S_p(u_1,y)∨~S_p(y,x)∨~I(u,u_1)

From their clause C_{1f} we can never get L(v,s(v_1)) as generalization of C_1=L(v,s(v)). From their C_f we can never get L(u,s(s(u_1))) as a generalization of C.

Stap (3), (4): We define a substitution θ_1={v / u, v_1 / u_1, x / y} and an intermediate clause N in their way but by using D_f and D_{1f} as basis. We have

N=L(u,x)∨~S_p(u_1,y)∨~S_p(y,x)∨~L(u,y)

By resolution with S_p(y,s(y)), S_p(u_1,s(u_1)), we get L(u,s(s(u_1))∨~L(u,s(u_1)). This means: if u is less than the successor of u_1, then u is less than the successor of succesor of u_1. This clause can not be found by the corresponding M in RP.

Stap (3), (4) *based on* N: Suppose we take N instead of M as the intermediate clause, we still have problems to find all the C_2 which should be found. All right most arguments in literals of a flattened clause should be different. In N we have y in both ~S_p(u_1,y) and ~L(u,y). Furthermore, every variable in a flattened clause should appear precisely once as the rightmost argument of literals defined by terms. This is not true for u_1 and u. If we consider the corresponding substitution μ={v / u} from the unflattened C_1 and the clause C∨~$C_1\mu_1$=L(u,s(s(u))∨~L(u,s(u)) then we can have L(u,s(y))∨~L(u_1,y), L(u,s(y))∨~L(u_1,y_1) as generalizations (these two clauses are

mathematically uninteresting). These two are not constructable from N. As we have mentioned in the end of section 3, this N is only partially flattened.
On the other hand, if we use the flattened clause of $C\vee{\sim}C_1\mu_1$, then we find all generalizations of this clause just as we can find all generalizations by term partitions on $C\vee{\sim}C_1\mu_1$ with NCF.

4.2 Improvements of RP

To begin with we need the good C_f and C_{1f} by using the flattening defined in this article. In other words, we assume that the absorptions, be they the old absorptions or the improvements, all begin with good C_f and C_{1f}. We can then improve the algorithm in two ways.

1 First way to improve RP. This change can not give all the generalizations, but it would give the interesting generalizations. To understand the improvements we should first understand the shortcomings of RP. We will explain them by giving examples.
Looking at the RP-algorithm, we notice that RP states actually the algorithm for general clauses and it can not be directly applied to the flattened clauses. For example, L_1 after flattening has more than one literal and these extra literals do not have to do with the flattened clause of C. In the RP-algorithm it is required that these extra literals shall be absorbed into C_f after the substitution θ_1. This is an unreasonable requirement, as the following shows:

Example. Given $C_1=P(f(x))$, $C=Q(g(x))$. To find $C_2=Q(g(x))\vee{\sim}P(f(x))$ is a problem because

(1) Flatten C_1 and C, we get $C_{1f}=P(z)\vee{\sim}I(x,x_0)\vee{\sim}F(x_0,z)$, $C_f=Q(u)\vee{\sim}I(x,x_0)\vee{\sim}G(x_0,u)$.

(2) There is no substitution θ_1 from C_1 such that ${\sim}F(x_0,z)\theta_1$ is a subclause of C_f. Thus this C_2 can not be constructed.

A similar example is first thought of by P. v. d. Laag (personal communication). He suggests to carry ${\sim}F(x_0,z)$ to the intermediate clause. By the explanation above we can also see why the intermediate clause should carry the literals induced by L_1. In fact if we consider $\theta_1=\{x/x,z/u\}$ then we have the following clause:

$M=Q(u)\vee{\sim}I(x,x_0)\vee{\sim}G(x_0,u)\vee P(u)\vee{\sim}I(x,x_0)\vee{\sim}F(x_0,u)$.

Then the resolutions with $F(x,f(x))$, $G(x,g(x))$ and $I(x,x)$ give $P(f(x))\vee{\sim}Q(g(x))$

Example. An example in [7] brought problems as it is told in that article. A suggestion of Muggeleton to solve this problem is also given in [7]. Although the suggestion is good, it is not certain if the reason given is correct. Let us find the solution in our way.

E_1: denser(hammer,feather).

E: heavier(hammer,feather).

C: larger(hammer,feather).

Using RP we can not find the following clause by absorptions

heavier(X,Y) $\leftarrow$ denser(X,Y)$\wedge$larger(X,Y).

Let us change them to the clausal forms and flattened forms in the same time. The constants a, b represent hammer and feather, respectively. P means heavier, D means denser and L means larger.

E_1: $D(x,y)\vee\sim H(x)\vee\sim F(y)$, $S_1=H(a)\wedge F(b)$

E: $P(x,y)\vee\sim H(x)\vee\sim F(y)$, $S=H(a)\wedge F(b)$

C_1: $L(x,y)\vee\sim H(x)\vee\sim F(y)$, $S_{c1}=H(a)\wedge F(b)$

The intermediate clause of E_1 and E with identity substitution is

$M=P(x,y)\vee\sim H(x)\vee\sim F(y)\vee\sim D(x,y)$

after we omit some double literals. Let $C=P(x,y)\vee\sim D(x,y)$ be a generalization of M. According to old RP we have problem to construct new intermediate clause because the literal with ~F can not be absorbed by C. With this improvement we can find $C_2=P(x,y)\vee\sim D(x,y)\vee\sim L(x,y)$ from C_f and C_{1f}.

Another problem which RP has not noticed is the choice of θ_1. It is not true that every substitution from variables in $\sim C_1$ is good substitution. We give a few examples.

Example. Let us consider $C_1=P(f(x))$ and $C=Q(g(x))$ again. For simplicity we do not change x to x_0. Then

$C_{1f}=P(z)\vee\sim F(x,z)$, $S_1=F(x,f(x))$

$C_f=Q(u)\vee\sim G(x,u)$, $S_2=G(x,g(x))$

Take $\theta_1=\{z/u, x/x\}$, we have

$M=Q(u)\vee\sim G(x,u)\vee\sim P(u)\vee\sim F(x,u)$.

If we use $F(x,f(x))$ to resolve with M, we get $Q(f(x))\vee\sim G(x,f(x))\vee\sim P(f(x))$. If we use $G(x,g(x))$ to resolve with I, then we get $Q(g(x))\vee\sim F(x,g(x))\vee\sim P(g(x))$. We can not eliminate all extra literals by resolution. If we look at the original clause, $\{x/x,z/u\}$ means that f(x) is replaced by g(x), such a substitution for the original clauses is not allowed.

On the other hand, if $C_1=P(f(x))$ and $C=Q(f(x))$, then

$C_{1f}=P(z)\vee\sim F(x,z)\vee\sim I(x,x)$

$C_f=Q(u)\vee\sim F(x,u)\vee\sim I(x,x)$.

Take the substitution $\theta_1=\{z/u\}$, we have

$M=Q(u)\vee\sim F(x,u)\vee\sim P(u)\vee\sim F(x,u)\vee\sim I(x,x)$.

The resolvent of M and $F(x,f(x))$ is $Q(g(x))\vee\sim P(g(x))$. This is a good C_2.

Example. Let C_1=P(x)∨~Q(x,x) and let C=R(a)∨~Q(a,a). This is a case where C_1' is not empty. According to RP, the part of C_{1f} which has to do with C_1' should be brought to C_f as a subclause. Let μ={x/a} be the substitution from C_1. A good C_2 is R(y)∨~P(y) which is a generalization of P(a)∨~R(a) (see NCF). This can not be found with old RP-algorithm because if we consider

C_{1f}=P(x)∨~Q(x_1,x_2)∨~I(x,x_1)∨~I(x,x_2),

C_f=R(y)∨~Q(y_1,y_2)∨~A(y1)∨~A(y_2).

The subclause ~Q(x_1,x_2)∨~I(x,x_1)∨~I(x,x_2) of C_{1f} can not be brought to C_f because ~I is another predicate than those of the literals in C_f.

What is a good θ_1? A good θ_1 should map the variables in C_{1f} in a consistent way. Let θ_1 be a substitution from variables in C_{1f}. Then such θ_1 should satisfy the following conditions :
Let C_1=L_1∨C_1'. We use the corollary in the last section to apply to D_1=~L_1∨C_1'. Let D_{1f}=U∨V where U is the part of D_{1f} which has to do with ~L_1 and V is the part of D_{1f} which has to do with C'. Let K be the subset of U which contains literals beginning with ~I. Then a good θ_1 is a substitution from the variables in C_f which satisfies the following conditions:

1) if uθ=v where v is a variable in C_f, then every literal in the substitution tree of u which does not begin with ~I is brougt to a literal in C_f and furthermore, if ~I(x,x_1),…,~I(x,x_n) are in K, then $x_1\theta_1$,…,$x_n\theta_1$ give the same term.

2) θ_1 induces a transformation from V to D_f.

Such a θ_1 can give an intermediate clause (U-K)θ_1∨C_f which is a partially flattened clause of some ~$L_1\mu_1$∨C. This μ_1 is a substitution from C_1 which brings C_1' to C.

Improved RP-absorption 1

(1) Choose a θ_1 which is a good substitution from C_{1f} defined as above.

(2) Define the intermediate clause M=(U-K)θ_1∨C_f where U and K are defined as above.

(3) Find all generalizations of M.

Remark. Looking at the μ_1 which is induced by a good θ_1, we notice that μ_1 brings variables in C_1 to terms in C. In fact we can give θ_1 some more freedom, especially when C_1' is empty or the variables in L_1 are different from the variables in C_1'. Some variables in K can then be brought to variables which are not in C_f. But a good θ_1 is for us often enough to find a good C_2. Thus a non-empty C_1' gives more restrictions to θ_1.
If a C_2 constructed by NCF-algorithm satisfies the condition: if a block of term occurrences in C∨~$L_1\mu_1$ for constructing C_2 contains occurrences from ~$L_1\mu_1$, that block contains also occurrences from C, then such C_2 can be constructed by RP-algorithm which we have modified above.

2 Change the intermediate clause to $(C\vee\sim L_1\mu_1)_f$ for some μ_1. In principle we can make RP-algorithm complete by considering other intermediate clauses which are flattened versions of $C\vee\sim L_1\mu_1$ for some μ_1. In this situation the part of $(C\vee\sim L_1\mu_1)_f$ which has to do with $L_1\mu_1$ can have more literals than the literals in C_{1f} which have to do with L_1 (we use $A\vee B$ in the discussion above). If we still want to consider a substitution from C_{1f} then it should be a transformation from C_{1f} to $(C\vee\sim L_1\mu_1)_f$.

Example. Let $C_1=P(x)$, $C=Q(y)$. Then $C_{1f}=P(x_0)\vee\sim I(x,x_0)$ and $C_f=Q(y_0)\vee\sim I(y,y_0)$.
We get only $Q(y)\vee\sim P(x)$ and $Q(y)\vee\sim P(y)$ as result if we consider θ_1 only variables to variables. We can for example never get $C\vee\sim C_1\mu_1=Q(y)\vee\sim P(f(y))$. On the other hand, if we consider the flattened clause of $Q(y)\vee\sim P(f(y))$, we have

$$Q(y_0)\vee\sim I(y,y_0)\vee\sim I(y,y_1)\vee\sim P(z)\vee\sim F(y_1,z)$$

as intermediate clause. From this clause we get the generalization we want. Notice that a literal beginning with ~F does not exist in C_{1f}.
It is possible to make the a modified RP-algorithm if we allow every $(C\vee\sim L_1\mu_1)_f$ as an intermediate clauses and consider the transformation induced by μ_1 from C_{1f} to $(C\vee\sim L_1\mu_1)_f$ instead of a θ_1 which induces no new literals than the literals from L_1 after transformation. This is rather artificial. We can just as well build a μ_1 and a $C\vee\sim L_1\mu_1$ and then build $(C\vee\sim L_1\mu_1)_f$. With this flattened clause we can construct all generalizations and we can forget what for transformations μ_1 shall induce.
The following algorithm gives actually a complete algorithm which we consider as the second kind of improvement.

Improvd RP-absorption 2. Let C and $C_1=L_1\vee C_1'$ be given. We follow the following steps to construct C_2.

(1) Let μ_1 be a substitution from variables of L_1 which satisfies that $C_1'\mu_1$ is a subclause of C.
(2) For a given C' which is a subclause of $C_1'\mu_1$ we consider the clause
$(C-C_1'\mu_1)\vee C'\vee\sim L_1\mu_1$.
(3) Let M be the flattened clause of $(C-C_1'\mu_1)\vee C'\vee\sim L_1\mu_1$.
(4) Find generalizations of M to get C_2's.

With this new algorithm we have two questions to ask:
Question 1. We compare the first stage of NCF with the first stage of the improvement 1. For a μ_1 which brings variables in L_1 to terms in C, improvement 1 is good enough for constructing C_2

which satisfies some conditions as stated in the remark above. The question is whether it is worth of using it. We have to go through the process of flattening and find good θ_1. In this stage finding μ_1 and then consider $C\vee\sim L_1\mu_1$ look easier. The first stage of improvement 2 should also begin with $C\vee\sim L_1\mu_1$. We have still to flatten it. Thus it is more work.

Question 2. We have already discussed the advantages of term partitions compared with generalizations from flattened (or partially flattened) clause. We can use the same discussion to compare the second stage of NCF and the second stage of improved RP-absorption 1, 2.

Acknowledgements. I express here my gratitute to the discussions I have had with Drs. P. v. d. Laag and Dr. J. C. Bioch.

References

1 Stephen Muggleton. Inductive Logic Programming. First Conference on Algorithmic Learning Theory, Ohmsha, Tokyo, October 1990.

2 Stephen Muggleton & Wray Buntine. Machine Invention of First-order Predicates by Inverting Resolution. Proceedings of the 5th International Conference on Machine Learning, Morgan Kaufmann, pp. 339-351, 1988.

3 Shan-Hwei Nienhuys-Cheng. Consequent Functions and Inverse Resolutions. Report Eur-CS-90-03, Erasmus University, Rotterdam, Netherlands, May 1990.

4 Shan-Hwei Nienhuys-Cheng. Term Partitions and Minimal Generalizations of Clauses. Report, Eur-CS-91-01, Erasmus University, Rotterdam, Netherlands, January, 1991,.

5 Shan-Hwei Nienhuys-Cheng & Peter FLach. Consistent Term Mappings, Term partitions and Inverse Resolutions, to appear in Proceedings of European Work Session on Learning, Portugal, Porto, March, 1991

6 Shan-Hwei Nienhuys-Cheng. Flattening, Generalizations of Clauses and Absorption Algorithms. Eur-CS-91-02, Erasmus University, Rotterdam, Netherlands, February 1991,.

7 Céline Rouveirol & Jean-Francois Puget. A Simple Solution for Inverting Resolution. EWSL-89, Pitman, London, pp. 201-210, 1989.

8 Sammut. Learning Concepts by Performing Experiments, Ph.D. dissertation, University of New South Wales, Kingston, 1981