Simple Improvements of A Simple Solution for Inverting Resolution

Jan C. Bioch
Patrick R. J. van der Laag[1]
Department of Computer Science
Erasmus University of Rotterdam
P.O. Box 1738, 3000 DR Rotterdam, the Netherlands
bioch@cs.eur.nl, patrick@cs.eur.nl

Abstract

In this paper we address some simple improvements of the algorithm of Rouveirol and Puget [11] for inverting resolution. Their approach is based on automatic change of representation called flattening and unflattening of clauses in a logic program. This enables a simple implementation of operators, such as Absorption, presented in Muggleton and Buntine [6]. Unfortunately both the algorithms of MB and RP are incomplete. We analyze the reasons of the incompleteness of the RP algorithm and present an improved Absorption operator. It appears that flat tree representations of clauses and predicate calculus with equality provide an appropriate context for these matters.

1 Introduction

1.1 Motivation

In [6] Muggleton and Buntine have described a well-known algorithm called CIGOL which is able to learn first order Horn clause theories on the basis of unit clause examples. In this paper "V" operators are introduced to simplify and compact theories during the learning process. These operators invert a single resolution step, see figure 1.

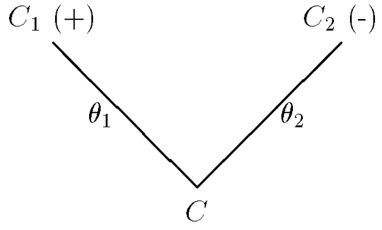


Figure 1: A single resolution step.

A "V" operator derives one of the clauses of the arm of the "V" given the clause on the other arm and the clause C at the base. In figure 1 the literal resolved on is positive (+) in C_1 and negative (-) in C_2. The *Absorption* operator constructs C_2 given C_1 and C. Conversely, the construction of C_1 from C_2 and C is called the *Identification* operator.

---

[1] supported by the Tinbergen Institute

In [6] a non-deterministic algorithm for the Absorption operator is given. Nienhuys-Cheng and Flach [8] have shown that this algorithm is not complete and not sound. They also give a non-deterministic algorithm for Absorption that is complete and satisfies an alternative soundness condition. However, both algorithms are more complex than the simple solution given by Rouveirol and Puget in [11]. We denote the aforementioned algorithms by MB, NCF and RP respectively. Unfortunately the last algorithm is also incomplete. There are two main reasons for this shortcoming. The first reason is that the operator they use itself is too restrictive. The second reason is the way the operator is used on a subset of the clauses that emerge from the representation change called flattening. In fact, we will show that this representation change can only be understood and formally described when we choose predicate calculus with identity (PC(=)) as our formal system rather then PC itself. We will discuss these matters in the next sections and we will present an improved version of the RP-algorithm. The algorithm is provably complete.

1.2 Notations and conventions

The notations are the ones of Lloyd [3] and Rouveirol and Puget [11]. A clause will be denoted by: $H \leftarrow B$. Although we will assume in the sequel that our theory consists of Horn clauses (H is a positive literal), the results can be easily extended to general clauses (H is a disjunction of positive literals), by assuming that H is a positive literal and allowing B also to contain negative literals.

The "V" operators in [6] are derived as a set of constraints from the following equation of resolution:

$$C = (C_1 - \{L_1\})\theta_1 \cup (C_2 - \{L_2\}\theta_2)$$

where $domain(\theta_1) \subseteq variables(C_1)$, $domain(\theta_2) \subseteq variables(C_2)$ and where L_1 is a positive literal in C_1, L_2 is a negative literal in C_2 and $\theta_1\theta_2$ is the mgu of $\neg L_1$ and L_2. C is called the *resolvent* of C_1 and C_2 and is denoted by $C = C_1 \cdot C_2$.

The relationship of generality between formulae is defined as follows:

Definition. Formula A is more general than formula B if and only if $A \vdash B$ and $B \not\vdash A$, where $\vdash$ means deduction in first order logic (PC). □

The following definitions of completeness and soundness are used in [8].

Definition. The absorption operator *Abs* is *complete* if it constructs all possible C_2's from C_1 and C such that $C = C_1 \cdot C_2$, and *sound* if only C_2's are constructed that satisfy this equation. □

Definition. The absorption operator *Abs* is called *weakly sound* if $\{C_1, C_2\} \vdash C$ for all C_2's constructed by the algorithm. □

All the operators mentioned in this paper satisfy the weakly soundness condition.

1.3 Predicate Calculus with Identity: PC(=)

It will appear from our discussion that Predicate Calculus with Identity: PC(=) rather than PC itself is the proper framework for understanding and formalizing the change in the description language of clauses, called flattening and unflattening. We therefore briefly recapitulate some basic results (cf. [4]). The formal system PC is extended to PC(=) by adding a two-place predicate E (also denoted by =) and the following axioms to the set of logical axioms:

1. $\forall x E(x, x)$, reflexivity
2. $\forall x \forall y (E(x, y) \rightarrow E(y, x)$, symmetry
3. $\forall x \forall y \forall z (E(x, y) \wedge E(y, z) \rightarrow E(x, z))$, transitivity
4. For any formula φ, $\forall x \forall y (E(x, y) \rightarrow (\varphi(x, x) \rightarrow \varphi(x, y)))$, Leibniz' law

Writing {Logical} and {Equality} for the sets of the logical and equality axioms respectively, it is known that

{Logical} + {Equality} $\vdash \varphi$ if and only if $\mathcal{M} \models \varphi$ for all *normal* models $\mathcal{M}$

An interpretation is called *normal* if the identity predicate is interpreted as the identity relation on the domain of objects.

Example. The formula $P(x) \wedge y = x \rightarrow P(y)$ is clearly true in all normal models, but is not necessarily true if we do not interpret the identity predicate as the identity relation. Thus $\vdash P(y) \leftarrow P(x) \wedge y = x$. According to the deduction theorem this is equivalent to $P(x) \wedge y = x \vdash P(y)$. □

2 Absorption according to Rouveirol and Puget

In [11] the following algorithm is used.

Absorption [11]. Given two clauses $C_1 : H_1 \leftarrow B_1$ and $C : H \leftarrow B$

1. Find a substitution θ_1 s.t.$B = a \wedge B_1\theta_1$.
2. Build the intermediate clause $C_{2a} : H \leftarrow a \wedge H_1\theta_1$.
3. Find a substitution θ_2 s.t. $H = H_2\theta_2$, $a = B_2\theta_2$ and $H_1\theta_1 = L\theta_2$.
4. $C_2 = Abs(C_1, C) : H_2 \leftarrow B_2 \wedge L$

Absorption on two clauses C_1 and C is only applicable if the body of C_1 matches a subpart of C. The central step is replacement of $B_1\theta_1$ by $H_1\theta_1$.

If the head of the clauses is interpreted as a class or concept identifier and the body of the clause is the conjunction of sufficient conditions for belonging to that class, then absorption can be used to rewrite an example (given as a ground clause) in terms of *higher level* concepts available in the domain theory, see [11].

Since $\theta_1\theta_2$ is not necessarily a *most general unifier*, the algorithm is not sound.

Example. Let $C_1 : P(x) \leftarrow$ and $C : Q(a) \leftarrow$ be given. Then $\theta_1 = \{x/a\}$ yields the intermediate clause $C_{2a} : Q(a) \leftarrow P(a)$. This may be generalized to $C_2 : Q(y) \leftarrow P(y)$. Thus $C_1 \cdot C_2 = Q(y) \leftarrow$ is more general than C. □

However, RP satisfies the weak soundness condition. It is easy to see that the algorithm is not complete. For, the central step, the replacement of the *whole* of $B_1\theta_1$ in B by $H_1\theta_1$, is too restrictive. We call this step absorption by replacement for the moment. A weaker form of replacement would be to split B_1 in two parts: $B_1 = D_1 \wedge E_1$, where D_1 and E_1 may be empty. Now replace $E_1\theta_1$ by $H_1\theta_1$. This yields an intermediate clause $C_{2a} : H \leftarrow H_1\theta_1 \wedge a \wedge D_1\theta_1$. We call this *absorption by partial replacement.* It is easy to see that C_{2a} and its generalizations satisfy $\{C_1, C_{2a}\} \vdash C$ and cannot be obtained by the RP-algorithm if D_1 is not empty. On the other hand, we have to be aware of the fact that partial replacement yields clauses that are difficult to interpret: both higher and lower level descriptions of a concept will occur in the same clause.

The extreme cases occur when D_1 is empty or $D_1 = B_1$. If we start with

$C_{2a} : H \leftarrow H_1\theta_1 \wedge a \wedge B_1\theta_1$, i.e.

$C_{2a} : H \leftarrow H_1\theta_1 \wedge B$, (absorption without replacement)

then all the possible intermediate clauses are obtained by dropping as many literals in $B_1\theta_1$ as is needed.

3 Absorption by Partial Replacement

In this section we prove that absorption by partial replacement is complete.

Absorption algorithm RP*.
Given two clauses $C_1 : H_1 \leftarrow B_1$ and $C : H \leftarrow B$

1. Find a substitution θ_1 s.t. $B = a \wedge B_1\theta_1$, where a and $B_1\theta_1$ are disjoint.
2. Split $B_1 : B_1 = D_1 \wedge E_1$, where D_1 and E_1 may be empty.
3. Build the intermediate clause $C_{2a} : H \leftarrow a \wedge H_1\theta_1 \wedge D_1\theta_1$.
4. Find L, D_2 and θ_2 s.t. $H_1\theta_1 = L\theta_2$ and $D_1\theta_1 = D_2\theta_2$.
5. Find H_2 and E_2 s.t. $H_2\theta_2 = H$, $B_2 = E_2 \wedge D_2$ and $E_2\theta_2 = a$.
6. $C_2 = Abs(C_1, C) : H_2 \leftarrow B_2 \wedge L$.

From step 3 of this algorithm we see that $E_1\theta_1$ in the body of C is replaced by $H_1\theta_1$ from C_1. Note also that $B_1\theta_1$ and $B_2\theta_2$ have a common subpart: $D = D_1\theta_1 = D_2\theta_2$.

Lemma. The algorithm RP* is weakly sound.
Proof. RP* is not sound since RP is not sound. However, RP* is weakly sound for we have

$\{C_1, C_2\} \vdash \{C_1, C_2\theta_2\} = \{C_1, C_{2a}\} \vdash (H \leftarrow a \wedge B_1\theta_1 \wedge D_1\theta_1) = C.$ □

Theorem. The algorithm RP* is complete.
Proof. Given $C_1 : H_1 \leftarrow B_1$, $C_2 : H_2 \leftarrow B_2 \wedge L$ and $C : H \leftarrow B$ such that $C_1 \cdot C_2 = C$. We have to show that the algorithm is able to construct C_2 from C_1 and C. Let $\theta = \theta_1\theta_2$ be a mgu of H_1 and L and let D be the common subpart of $B_1\theta_1$ and $B_2\theta_2$. Then we can construct D_1 and D_2 such that $D_1\theta_1 = D_2\theta_2 = D$. Construct E_1 such that $B_1\theta_1 = D_1\theta_1 \wedge E_1\theta_1$. Since $C_1 \cdot C_2 = C$ we know that $H = H_2\theta_2$ and $B = B_1\theta_1 \wedge B_2\theta_2 = B_1\theta_1 \wedge E_2\theta_2$. Thus $C_2 : H_2 \leftarrow D_2 \wedge E_2 \wedge L$ will be constructed by the algorithm if we split B_1 in step 1 according to $B_1 = D_1 \wedge E_1$. □

Example.

$$C_1 : P(x,y) \leftarrow Q(f(x),y) \wedge R(x,y)$$

$$C : S(x,y) \leftarrow Q(f(x),g(x)) \wedge R(x,g(x)) \wedge T(x,y)$$

$$C_{2a} : S(x,y) \leftarrow P(x,g(x)) \wedge T(x,y) \wedge [Q(f(x),g(x)) \wedge R(x,g(x))]$$ □

Here both literals in [ ] are optional.

The difficult step in RP* is of course step 4. In this step we have to generate all possible inverse substitutions in order to find all the C_2's such that $C_1 \cdot C_2 = C$. In [6] Muggleton and Buntine provided an indeterministic algorithm to solve this problem by using term partitions. This algorithm has been improved by Nienhuys-Cheng [8]. However, in both papers C_1 is a literal. Here C_1 is a Horn clause. But as we stated before we can easily extend the algorithm to general clauses $H \leftarrow B$ by assuming that the body may also contain negative literals whereas H remains a single literal. Furthermore, one can use the term partition algorithm of [8] to do the inverse substitution step.

In the next section we will discuss how Rouveirol and Puget try to simplify the complexity of the algorithm (i.e. step 4) by changing the description language by eliminating constants and function symbols.

4 Changing the Description Language

The simple solution for inverting resolution is based on the idea that inverse resolution is complex if the description language may contain arbitrary terms. Indeed, the critical step in the absorption algorithms is to find inverse substitutions (step 4). In [11] it is shown that turning a term into a new variable amounts to drop literals and unify variables in the flat version of a clause. Therefore the description language is changed by removing function symbols and constants. This transformation called *flattening* is well-known in mathematical logic [4] and used by Sammut in [12]. In this transformation each constant and function symbol is replaced by a predicate symbol. According to Rouveirol and Puget the algorithm is as follows:

Flattening [11]. For each function f of arity n $f(t_1, \ldots, t_n)$ that appears in the program

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 $f_p(t_1, \ldots, t_n, x)$ to the body of C.
3. Add the fact $f_p(t_1, \ldots, t_n, f(t_1, \ldots, t_n)) \leftarrow$ in order to define the new predicate f_p. These facts will *not* be considered for inverting resolution.

It has been argued by Nienhuys-Cheng (personal communication) that this algorithm has a subtle shortcoming: if a variable occurs more than once, then we have to introduce new variables. Otherwise we cannot obtain all the generalizations. For example, the literal $P(x,x) \leftarrow$ has to be replaced by the clauses $P(x,y) \leftarrow X(x,y)$, and $X(x,x) \leftarrow$.

Example [11]. Flattening the clause

$C : inf(x, s(s(x))) \leftarrow$ yields the clause

$C_f : inf(x,y) \leftarrow s_p(z,y) \wedge s_p(x,z)$, and the extra literal

$U : s_p(x, s(x)) \leftarrow$ □

Unflattening C_f is performed by resolution using the extra literal U. It is not clear from the algorithm whether all the arguments but the last of the added facts are variables (as they always should be in our opinion) or compound terms. Furthermore, we already notice that *not* considering the added facts for inverting resolution is a source for the incompleteness of the absorption operator! For C is logically equivalent with $C_f \wedge U$, but not with C_f. This is easy to prove, especially in PC(=).

Finally, generalizations of a flattened clause are obtained by removing the added literals of a term, and by unifying the result variables, see [11]. In the next section we show that predicate calculus with equality and so-called flat tree representations provide a natural framework for replacing function symbols by result variables. We also show in the next section that dropping literals to obtain generalizations should be done consistently.

4.1 Flattening in PC(=)

A rigorous approach to flattening and absorption will require that we extend our language $\mathcal{L}$ to a language $\mathcal{L}_f$ which contains the new predicate symbols f_p, and the addition of new axioms to the formal system PC. Furthermore, we have to study the semantics of this new system. Although this would not be very difficult we prefer to use the well known formal system PC(=), see section 1.3. Thus we need only one extra symbol, namely the predicate = (equality, identity). Notice that $\mathcal{L} \subset \mathcal{L}_= \subset \mathcal{L}_f$.

However, we will use symbols like f_p as meta-symbols to enable comparisons between the various representations of a clause. It is well known that a literal can be viewed as a *labeled tree*. For a precise definition of this concept we refer to Gallier [2]. The labels are symbols that denote predicates, functions, variables and constants.

Example. The literal $P(x, f(x))$ can be represented by a labeled tree, see figure 2a. □

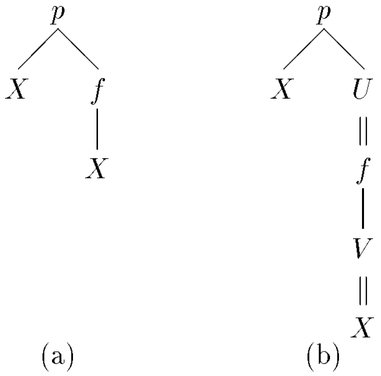


Figure 2: Labeled trees

We will now define the concept of what we call a *flat tree representation of a literal.* Let T be a tree representation of the literal L. Then we can relabel this tree by replacing the label of f, an internal node, by a label $u = f$, where u is a new variable: a result variable. Similarly, a leave representing a constant c is relabeled as $v = c$. Leaves representing variables of the literal will be represented in the same way, with at most one exception. That is, we permit that one occurrence of a variable is not relabeled. Thus, each label is either a variable or a string of the form $u = f$ or $v = a$, where f is a function, a is a constant and u and v are result variables. We assume that all result variables in the new representation are different.

We notice that the structure of the tree is not changed, only the names of the nodes are changed.

A tree obtained in this way is called a FTR (flat tree representation) of L. Such a representation can be obtained by any algorithm that visits all the nodes of a tree and changes the labels according to the constraints mentioned above.

We call a FTR *minimal* if for each variable x of L there exists exactly one label equal to x.

Example. Consider $L : P(f(x), g(x, c))$, where c is a constant, see figure 3. By abuse of language we will use the following meta-description of the FTR of

$L : P(f(x), g(x, c))$,

$FTR(L) = P(u_1 = f(x), u_2 = g(v_1 = x, v_2 = c))$ □

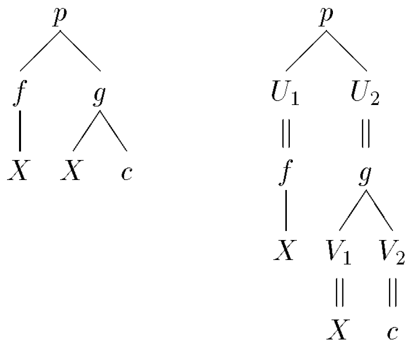
A flat tree representation of a literal L can be directly converted to a Horn clause L_f that is logically equivalent to the literal L. Thus

$L_f : P(u_1, u_2) \leftarrow u_1 = f(x) \wedge u_2 = g(v_1, v_2) \wedge v_1 = x \wedge v_2 = c$

is logically equivalent with $P(f(x), g(x, c))$. This follows from the fact that the equivalence $L \equiv L_f$ is a tautology. For we have the following possible sequence of steps:

$L : P(f(x), g(x, c))$

$L_1 : P(u_1, g(x, c)) \leftarrow u_1 = f(x)$



L $FTR(L)$

Figure 3: A flat tree representation.

$L_2 : P(u_1, u_2) \leftarrow u_1 = f(x) \wedge u_2 = g(x, c)$
$L_3 : P(u_1, u_2) \leftarrow u_1 = f(x) \wedge u_2 = g(v_1, c) \wedge v_1 = x$
$L_f : P(u_1, u_2) \leftarrow u_1 = f(x) \wedge u_2 = g(v_1, v_2) \wedge v_1 = x \wedge v_2 = c$

For example $\vdash L \equiv L_1$. To give an idea how to prove this we will consider the inference $L \vdash L_1$. Any *normal* model of L is also a model of L_1. Thus we have $L \vdash L_1$, see 1.3.

Example. $L = P(x, f(x))$ is logically equivalent with
$P(x, u) \leftarrow u = f(v) \wedge v = x$, and also with
$P(x, u) \leftarrow u = f(x)$. □

4.2 Generalization

Since all the arguments of a function (or predicate) in a FTR are different, it is easy to see that all possible generalizations of a literal can be obtained by replacing subtrees with root-labels of the form $u = f$ by the result variable u, and by (optionally) unifying result variables, if the subtrees represent terms that are equal.

Example. $P(x, f(x))$ has two generalizations, $P(x, u)$ and $P(x, f(v))$, obtained by "cutting" the labels $u = f$ and $v = x$ respectively, see figure 4(a). This is equivalent with dropping literals in $P(x, u) \leftarrow u = f(v) \wedge v = x$ *consistently*, i.e., we either delete $v = x$ or *both* $u = f(v)$ and $v = x$. □

Example. The ground literal $P(a, a)$ has four generalizations: $P(u, v)$, $P(u, a)$, $P(a, v)$ and $P(u, u)$. The last generalization is obtained by cutting the labels $u = a$ and $v = a$, and unifying the result variables u and v, see figure 4(b). □

4.3 Unflattening

We rewrite the clause $P(x, u) \leftarrow u = f(v) \wedge v = x$ as

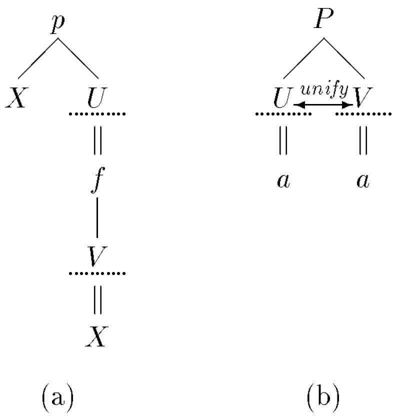


Figure 4: Where to “cut”, and where to unify.

$P(x,u) \leftarrow f_p(v,u) \wedge E(x,v)$.
Then, we can unflatten this clause by resolution with the additional literals $f_p(v, f(v)))$ and $E(x,x)$ respectively. However, in our previous notation, unflattening is the same as replacing each label $u = f$ (or $v = a$) by f (or a). Thus we simply *forget* all the result variables:

$P(x, u = f(v = x)) \overset{unflattening}{\longrightarrow} P(x, f(x))$.

4.4 Substitutions

It is also easy to carry out substitutions in a FTR of a literal.

Example. $L = P(x, f(x)), \theta = \{x/g(y)\}$. Then $FTR(L\theta)$ can be obtained from $FTR(L)$ by replacing the leaves with label x and $v = x$ by $x = g$ — y and $v = g$ — y. In general if y already occurs in L we have to introduce a new result variable as usual. Thus if
$L_f = P(x,u) \leftarrow u = f(v) \wedge v = x$, then
$L_f\theta = P(x,u) \leftarrow x = g(y) \wedge u = f(v) \wedge v = g(z) \wedge z = y$. □

Note, that after the substitution x is to be interpreted as a result variable. Thus in $L_f\theta$ we have only one ordinary variable: y.

Obviously, result variables will never be used in a substitution θ, unless θ is a renaming of the result variables. However, if θ is a renaming, then we will interpret L_f and $L_f\theta$ as equivalent expressions. Consequently, if L is a ground literal, then all variables are result variables, so that only renamings are allowed. In section 5.2 we will introduce so-called consistent substitutions of result variables in order to partially unflatten a clause.

4.5 Flat tree representation of general clauses

Up to now we have only explicitly discussed and defined the concept of flat tree representation of a literal. However, the definitions and results hold for general clauses as well, as the following example demonstrates:

Example. The FTR of the clause $P(x, f(y)) \leftarrow Q(y, g(x))$ is in clausal form

$P(x, z) \leftarrow z = f(y) \wedge Q(u, v) \wedge u = y \wedge v = g(w) \wedge w = x$, see figure 5.

This clause has eight possible generalizations. □

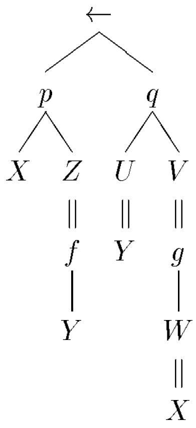


Figure 5: A minimal FTR of a clause.

5 Resolution and Absorption using Flattening

It is important to keep in mind that after flattening a clause C to C_f, the clause C_f is more general than C: $C_f \models C$, unless the equality predicate is interpreted as the identity relation.

5.1 Resolution

Let C_{1f} and C_{2f} represent the clauses C_1 and C_2 after flattening. Note that C_{1f} and C_{2f} are not necessarily flat tree representations in the strict sense. We do not introduce new result variables if the variables in $C_1 \cdot C_2$ are not distinct when we are not interested in generating all possible generalizations. Then the question is now how the resolvents $C_1 \cdot C_2$ and $C_{1f} \cdot C_{2f}$ are related.

Example.

$C_1 : P(f(x)) \leftarrow$ $\qquad$ $C_{1f} : P(u) \leftarrow u = f(x)$

$C_2 : Q(f(y)) \leftarrow P(g(y))$ $\qquad$ $C_{2f} : Q(v) \leftarrow v = f(y) \wedge P(w) \wedge w = g(y)$

Thus, although $C_1 \cdot C_2$ is not defined, we have

$C_{1f} \cdot C_{2f} : Q(v) \leftarrow v = f(y) \wedge w = f(x) \wedge w = g(y)$

Unflattening and combining the literals with w yields

$C' : Q(f(y)) \leftarrow f(x) = g(y)$.

We conclude that $\{C_1, C_2\} \vdash C'$. However, C' is not the resolvent of C_1 and C_2. □

The example shows that flattening inevitably introduces the equality predicate, and that we cannot always get rid of this predicate by means of unflattening. It also shows the effect of applying the resolution operator on *finer grained* descriptions of clauses. It can yield clauses that are logical consequences of the parent clauses C_1 and C_2 that cannot always be obtained in *one* resolution step using the "old" descriptions. The following example shows that the results of resolution after flattening have to interpreted carefully.

Example.

$$\begin{array}{ll} C_1 : P(f(x)) \leftarrow & C_{1f} : P(u) \leftarrow u = f(x) \\ C_2 : Q(g(y)) \leftarrow P(f(y)) & C_{2f} : Q(v) \leftarrow v = g(y) \wedge P(w) \wedge w = f(y) \end{array}$$

Thus, $C_1 \cdot C_2$ is $Q(g(y)) \leftarrow$, and $C_{1f} \cdot C_{2f}$ after flattening yields the clause

$$C' : Q(g(y)) \leftarrow w = f(x) \wedge w = f(y) \text{ (or } Q(g(y)) \leftarrow f(x) = f(y)\text{)}.$$

In this case C' and $C_1 \cdot C_2$ are logically equivalent, and we can drop the literals $w = f(x)$ and $w = f(y)$ in C'. In general we have to respect the tree structure of C'. Thus if we delete a literal containing a result variable, then possibly we have to drop other literals as well. □

Thus if we do not consider a flattened clause as a tree structure, then applying operators like resolution and absorption on the nodes (rather then on the subtrees) will have the effect of "disassembling the tree structure". It clearly shows that the representation change has to be interpreted carefully. Therefore, in the next subsection we will consider the representation change (flattening/unflattening) from a more abstract point of view.

5.2 Absorption

The representation change carried out by Rouveirol and Puget can be symbolized by the commutative diagram of figure 6.

$$\begin{tikzcd}
T_f \arrow[r, "\mathcal{A}_f"] & T'_f \arrow[d, "\psi"] & \mathcal{L}_f \arrow[d, leftrightarrow] \\
T \arrow[u, "\varphi"] \arrow[r, "\mathcal{A}"'] & T' & \mathcal{L}
\end{tikzcd}$$

Figure 6: Een commutative diagram.

Here $\mathcal{A}$ is e.g. the absorption operator, φ and ψ represent the operations of flattening and unflattening respectively and the T's represent theories. We assume that $\mathcal{A}_f$ is such that $\mathcal{A} = \psi \cdot \mathcal{A}_f \cdot \varphi$. A priori it is not obvious that $\mathcal{A}_f$ only performs absorption on the flattened theory T_f. Analogously, if $\mathcal{A}_f$ is the operation of resolution, then we have seen that $\mathcal{A}_f$ yields resolvents that are too specific. The reason is simple: applying φ and then $\mathcal{A}_f$ has the effect of destroying the tree structures of the literals. This can be repaired by defining the operator $\mathcal{A}_f$ such that the tree structure is preserved. However, in that case $\mathcal{A}_f$ is no longer a pure absorption (resolution) operator. It is also possible to let $\mathcal{A}_f$ apply absorption on T_f and carry out some correction operations afterwards.

Thus, one problem of the representation change is simply the preservation of the tree structure of clauses. Another problem is that T_f is more general than T. Even if we use a complete flattening procedure (instead of the procedure in [11]), to obtain generalizations, we can certainly not forget the extra literals as suggested in [11].

Thus if $T = \{C_1, C\}$, then T and $T_f = \{C_{1f}, C_f\}$ are not equivalent. In fact we have to apply $\mathcal{A}_f$ not on T_f but on $T_f \wedge U$, where U is the conjunction of the extra literals added by the flattening algorithm. Actually, we can show that adding enough of these literals to C_f will solve this problem of incompleteness. Note that the operation of adding literals has the effect of *specializing* C_f and can also be seen as a preliminary absorption step. On the other hand, if we try to find a suitable instantiation $C_{1f}\theta_1$ in order to find the intermediate clause C_{2a}, then in fact we specialize C_{1f} by adding "substitution"-literals.

To be more specific we will discuss the following example:

Example.

C_1: $P(f(0)) \leftarrow$	C_{1f}: $P(x) \leftarrow x = f(y) \wedge y = 0$
C : $Q(g(0)) \leftarrow$	C_f : $Q(u) \leftarrow u = g(v) \wedge v = 0$

Let $\mathcal{A}$ be the RP*-algorithm and let $\mathcal{A}_f$ denote RP* on flattened clauses. It is easy to see that $C_{2a} : Q(g(0)) \leftarrow P(f(0))$. However, it seems that we cannot apply absorption on C_{1f} and C_f at all, since the absorption requirement is not satisfied! For it seems that no instantiation of the body of C_{1f} matches a subpart of C_f. In our approach this *is* amazing, since C_{1f} and C_f are flat tree representations of C_1 and C respectively. Thus if we use only normal interpretations, then C_{1f} and C_f are logically equivalent with C_1 and C. Fortunately, we can trivially extend the body of C_f:

$Q(u) \leftarrow u = g(v) \wedge v = 0 \wedge f(y) = f(y) \wedge 0 = 0$

Note, that if we do not use normal interpretations, then this extension would be a proper specialization of C_f.

Now there are two substitutions θ_1 of the result variables in C_{1f} such that the body of $C_{1f}\theta_1$ matches a subpart of the body of C_f:

$\theta_1 = \{x/f(y), y/0\}$ and $\theta'_1 = \{x/f(v), y/v\}$

Both substitutions have the effect of *(partially) unflattening* the clause C_{1f}:

$C_{1f}\theta_1 : P(f(0)) \leftarrow$ and $C_{1f}\theta'_1 : P(f(v)) \leftarrow v = 0$

Thus $C_{1f}\theta_1$ and $C_{1f}\theta'_1$ are logically equivalent with C_{1f}.

Definition. Let C_f be the flattened representation of a clause C. Then a substitution θ of result variables is called *consistent* if $C\theta$ is a partial unflattening of C_f. □

We conclude that a consistent θ preserves the tree structure of C_{1f}. Furthermore, if C_1 is a literal, then we can always find a consistent substitution θ_1 such that the body of $C_{1f}\theta_1$ matches a subpart of the body of C_f. Hence, we can always satisfy the absorption requirement. For example, using θ_1 we find the intermediate clause

$C_{2af} : Q(u) \leftarrow u = g(v) \wedge v = 0 \wedge P(f(0))$, or equivalently

$C_{2af} : Q(u) \leftarrow u = g(v) \wedge v = 0 \wedge P(x) \wedge x = f(y) \wedge y = 0$

Both representations are equivalent with the intermediate clause

$C_{2a} : Q(g(0)) \leftarrow P(f(0))$

Using θ_1' we find the clause $C_2 : Q(g(v)) \leftarrow P(f(v))$. We also find C_{2a} if we apply absorption *without* replacement.

To avoid name clashes of the result variables in the intermediate clause, we will always assume that C_1 and C have no variables in common. Note here that *partial* unflattening yields the intermediate clause C_{2a} but also a generalization of C_{2a}.

We conclude that we can use consistent substitutions of the result variables to satisfy the absorption requirement. However, we will *not* carry out these substitutions directly! We only need them to explain how we can find the flattened versions of the intermediate clauses C_{2a}.

Lemma. Let $C_1 : H_1 \leftarrow$ and $C : H \leftarrow B$ be given, and let θ_1 be an arbitrary substitution of variables occurring in H_1. Let $H_{1f} \leftarrow H_e$ and $H_f \leftarrow B_f$ be the flat representations of $C_1\theta_1$ and C respectively. Then $C_{2af} : H_f \leftarrow H_e \wedge B_f$. Consequently, each C_{2f} is a generalization of this clause.

Proof. See the RP*-algorithm and the idea of consistent substitutions of result variables. □

Note that this lemma can be generalized. For we can split H_e in two parts $H_e = H_{e1} \wedge H_{e2}$, assuming that H_{e2} is that part of H_e that already matches a part of B_f. In that case we consider the clause $C'_{2f} : H_f \leftarrow H_{e1} \wedge B_f$. This has the same effect as *partially* unflattening C_1. However, as we have seen in the previous example, C'_{2f} has to be interpreted as C_{2af} or as a *generalization* thereof.

Using this lemma, it is easy to solve the incompleteness of the hammer-example discussed in [11]. In the general case in which the body of C_1 is not empty: $H_1 \leftarrow B_1$ we have to find a substitution θ_1 such that $B_1\theta_1$ matches a subpart of B. This can be done by *unifying* the *trees* B_{1f} and B_f. Obviously, one can *not* expect that such a substitution only substitutes variables for variables, as suggested by the approach of Rouveirol and Puget, see also section 5.3. Indeed, every step of the RP*-algorithm has its counterpart in the world of flat representations, and every form of incompleteness can be eliminated. Avoiding incompleteness means that we have to accept that a flattened clause is a tree structure. This structure has to be respected in our operations on the clause.

Thus it becomes more and more clear that incompleteness is not really a problem, but rather the gain of changing the description language! As we have noticed before, the RP*-algorithm can be split into two parts. In the first part (steps 1–3) the intermediate clauses are constructed. Then, in the second part (steps 4–6), we have to find generalizations of these clauses. We have shown that flattening is not appropriate in the first steps. It is not conceptually easier or computationally more efficient to carry out the first part in a flat world. In our opinion, the representation change is more useful in the second part of the algorithm. Therefore we propose the following algorithm for absorption:

Absorption algorithm RP**.
Given two clauses $C_1 : H_1 \leftarrow B_1$ and $C : H \leftarrow B$

1. Find a substitution θ_1 s.t. $B = a \wedge B_1\theta_1$, where a and $B_1\theta_1$ are disjoint.
2. Split $B_1 : B_1 = D_1 \wedge E_1$, where D_1 and E_1 may be empty.
3. Build the intermediate clause $C_{2a} : H \leftarrow a \wedge H_1\theta_1 \wedge D_1\theta_1$.
4. Construct a flat tree representation of C_{2a}: C_{2af} by relabeling C_{2a}.
5. Generate all generalizations (inverse substitutions) from C_{2af} to C_{2f}.
6. Unflatten C_{2f} to C_2.

The second part of this algorithm (step 4–6) is essentially the same as in RP*. However, in this case we use flattening to find the inverse substitutions. Since we know that RP* is complete it follows that RP** is also a complete algorithm given that in step 4,5 and 6 all the generalizations from C_{2a} to C_2 are obtained. Here we use minimal flat tree representations of C_{2a} to fulfill this task. But any other correct procedure that generates these inverse resolutions, e.g. [8] is sufficient. In fact it may turn out that our approach is theoretically equivalent with term partitioning. We currently investigate the relationship between the two approaches.

Remark. A similar algorithm has been obtained by Nienhuys-Cheng by another approach (personal communication, see also [7]).

We do not want to suggest that the steps in the RP** algorithm have to be taken in the given order. Obviously, the splitting-step 2 can be combined with step 4 by choosing E_1 empty. It is even not necessary to do the substitution step before flattening. Indeed, as we showed in our previous analysis in 5.2 it is also possible to use flattening also in the first step of the algorithm. This remains an interesting alternative at least in some cases, see 5.3.

5.3 Control heuristics

An implementation of any absorption algorithm needs a good set of control heuristics. Some of the heuristics are discussed in [5], [6], and [11]. Concerning the RP**-algorithm, heuristics are required in the following steps.

a) Step 1: Finding a substitution θ_1.
b) Step 2(3): Generating splittings of B_1 (intermediate clauses).
c) Step 5: Generating generalizations.

a) We already noticed that in order to be complete we cannot only use substitutions θ_1 from variables to variables as the approach of Rouveirol and Puget suggests. Therefore, we think that flattening in the first steps of the algorithm is not very helpful. On the other hand, in some cases, particularly those in which the body of C_1 is empty, flattening can be interesting because it generates automatically non-trivial substitutions.

Example.

$C_1 : P(x, f(x)) \leftarrow$ $\qquad$ $C_{1f} : P(x, u) \leftarrow u = f(x)$
$C : P(y, f(g(y))) \leftarrow$ $\qquad$ $C_f : P(y, v) \leftarrow v = f(w) \wedge w = g(y)$

Using the substitution $\theta_{var} = \{u/v, x/w\}$, in order to satisfy the absorption requirement, yields:

$P(y,v) \leftarrow P(w,v) \wedge w = g(y) \wedge [v = f(w)]$,

where the literal $v = f(w)$ obtained by absorption *without* replacement is optional. Unflattening this clause gives:

$P(y,v) \leftarrow P(g(y),v) \wedge [v = f(w)]$,

The clause $P(y,v) \leftarrow P(g(y),v)$ can also be obtained directly from C_1 and C by using the substitution $\theta_1 = \{x/g(y)\}$ followed by a generalization step. Thus the substitution θ_{var} induces both the substitution θ_1 and a generalization step. The reason that θ_{var} induces θ_1 is of course that θ_{var} substitutes the *result* variable w (occurring in C_f) for the ordinary variable x in C_{1f}. Since $w = g(y)$, this has the same effect as the substitution $\theta_1 = \{x/g(y)\}$. Therefore substitutions like θ_{var} can only induce substitutions that are "implicit" in the bodies of C_{1f} and C_f. Note, that θ_1 (θ_{var}) can also be found by comparing the tree structures of C_1 and C respectively. We conclude that flattening can be useful if we are only interested in substitutions that are "implicit". For example, we obviously cannot induce a substitution of the form $\theta_1 = \{x/h(y)\}$, since the function symbol does not occur in C_1 and C. In our opinion, such a substitution is indeed not of practical interest. For absorption is used to rewrite an example (*given* as a ground clause) in terms of higher level concepts *available* in the domain theory, see [11].

Therefore, we think that finding substitutions θ_1 that are "implicit" are of primary interest. The question whether these substitutions are the only interesting ones is left to the reader. However, the following example can give an idea of other substitutions that cannot be induced by θ_{var}.

Example.

$C_1 : P(f(x)) \leftarrow$ $\qquad$ $C_{1f} : P(u) \leftarrow u = f(x)$

$C : P(g(f(y))) \leftarrow$ $\qquad$ $C_f : P(v) \leftarrow v = g(w) \wedge w = f(y)$

Using C_{1f} and C_f the reader can verify that the RP-algorithm yields the clause $P(g(w)) \leftarrow P(w)$ (although not $P(g(f(y))) \leftarrow P(f(y))$ as found by the RP*-algorithm). However, using only variable substitutions, we cannot find the clauses $P(g(f(y))) \leftarrow P(f(g(y)))$ and $P(g(w)) \leftarrow P(f(w))$ generated by the RP**-algorithm. □

b) Generating all possible splittings, i.e., applying absorption with all possible partial replacements, including all generalizations can be very time-consuming. We also noticed that there are problems with the interpretation of the resulting clauses, since both higher and lower level descriptions of a concept will occur in the same clause.

c) Generating all generalizations of the intermediate clause(s) is clearly a formidable task. Moreover, many generalizations C_2' such that $C_1 \cdot C_2'$ is more general than C are too general or contain literals with unlinked variables. Therefore, heuristics that prevent such generalizations are necessary. Unfortunately, all known absorption algorithms are weakly sound and not sound. Although weakly soundness seems to be an inevitable property of absorption operators, we do not think that weakly soundness is harmless or even desirable as suggested in [8].

6 Conclusion

The idea of simplifying inverse resolution by changing the description language, as proposed by Rouveirol and Puget, remains useful. It is conceptually simpler than the idea of term partitions used in [6] and [8]. We have shown that absorption using flattening yields *complete* results if the representation change is carried out carefully, and if one uses absorption with partial replacement.

To understand this representation change we have argued that it is most convenient (and necessary) to work in the context of predicate calculus with identity (PC(=)), rather than in PC. Important issues as: how to interpret the requirement for application of the absorption operator, flattening, substitutions and last but not least proving algorithms are easier to understand in the context of PC(=) and flat tree representations. Disassembling this tree structure by applying operators like absorption on *nodes* rather than on subtrees will result in incompleteness. In our opinion, the advantage of the representation change is mainly that the inverse substitution step in the absorption algorithm becomes easier. Whether or not flattening is used to find the intermediate clause is more a pragmatic issue. It is not computationally less efficient. In section 5.2 we proposed a complete and weakly sound algorithm: RP**-absorption, that is easy to understand and implement. Although we have shown how to make the absorption operator complete even for general clauses, the question remains whether completeness is always desirable. For example, why should an absorption operator perform substitutions of the form $\{x/h(y)\}$ if the symbol h does not already occur in the existing theory or in the set of examples. Therefore an implementation of the RP**-absorption algorithm we have given will depend on the induction system that includes the absorption operator.

7 Related Work

After finishing an earlier draft version of this paper in februari this year [1] Céline Rouveirol sent us two papers [10] and [9]; with some similar results. In particular it appears that absorption with partial replacement is in fact already known. Absorption without deletion is called elementary saturation in [9]. The saturation operator introduced in [10] can be seen as a repeated application of absorption by partial replacement on a *set* of n (≥ 2) clauses. Thus, absorption with partial replacement (if we only allow sound solutions) is a single inverse resolution step, and the saturation operator can be compared with the immediate consequence operator in logic programming. Therefore, inversion completeness can also be proved by using the completeness of the RP*-algorithm which itself can be directly proved very easily (see 3). From this it follows that the use of the saturation operator in combination with flattening, but without compensating for the restriction due to variable-variable substitutions, is *not* complete.

In a recent report [9] equality theories are used as the proper framework for inverse resolution and flattening. We (independently) have argued (in 4.1) that using just PC(=) is sufficient and, in our opinion, even easier than equality theories.

Finally, we have already referred to the related work of Nienhuys-Cheng [7].

Acknowledgement. We like to thank Peter Flach for some useful comments on an earlier version of this paper [1].

References

[1] J.C. Bioch and P.R.J. van der Laag. Simple improvements of a simple solution for inverting resolution. Technical Report EUR-CS-91-03, Erasmus University Rotterdam, March 1991 (Preprint Feb 1991).

[2] J.H. Gallier. *Logic for Computer Science.* Harper & Row, 1986.

[3] J.W. Lloyd. *Foundations of Logic Programming.* Springer, 2nd edition, 1987.

[4] E. Mendelson. *Introduction to Mathematical Logic.* Van Nostrand, 1964.

[5] S.H. Muggleton. Inductive logic programming. In *First Conference on Algorithmic Learning Theory*, Ohmsha, Tokyo, 1990.

[6] S.H. Muggleton and W. Buntine. Machine invention of first-order predicates by inverting resolution. In *Proceedings of the Fifth International Conference on Machine Learning*, pages 339–352. Kaufman, 1988.

[7] S.H. Nienhuys-Cheng. Flattening, Generalizations of Clauses and Absorption Algorithms. These proceedings.

[8] S.H. Nienhuys-Cheng and P. Flach. Consistent term mappings, term partitions and inverse resolution. In *EWSL-91.* To appear.

[9] C. Rouveirol. Extensions of inversion of resolution applied to theory completion. Technical Report 624, Université de Paris-Sud, Orsay France, jan 1991.

[10] C. Rouveirol and J-F.Puget. Beyond inversion of resolution. In *proceedings of the fifth International Conference on Machine Learning.* Kaufman, 1990.

[11] C. Rouveirol and J-F. Puget. A simple solution for inverting resolution. In *EWSL-89*, pages 201–210, Pitman, London, 1989.

[12] C. Sammut. *Learning Concepts by Performing Experiments.* PhD thesis, University of New South Wales, Kingston, 1981.