Improving Example-Guided Unfolding

Henrik Boström

Dept. of Computer and Systems Sciences
Stockholm University
Electrum 230, 164 40 Kista, SWEDEN
henke@dsv.su.se

Abstract. It has been observed that the addition of clauses learned by explanation-based generalization may degrade, rather than improve, the efficiency of a logic program. There are three reasons for the degradation: i) increased unification cost ii) increased inter-clause repetition of goal calls iii) increased redundancy. There have been several approaches to solve (or reduce) these problems. However, previous techniques that solve the redundancy problem do in fact increase the two first problems. Hence, the benefit of avoiding redundancy might be outweighed by the cost associated with these techniques. A solution to this problem is presented: the algorithm EGU II, which is a reformulation of one of the previous techniques (Example-Guided Unfolding). The algorithm is based upon the application of program transformation rules (definition, unfolding and folding) and is shown to preserve the equivalence of the domain theory. Experimental results are presented showing that the cost of avoiding redundancy is significantly reduced by EGU II, and that even when the redundancy problem is not present, the technique can be superior to adding clauses redundantly.

1 Introduction

The benefits of adding clauses learned by explanation-based generalization (EBG) [13, 9] to a logic program come from reordering effects and decreased path costs when the clauses are successfully applied (cf. [12]). However, it has been observed that the addition of learned clauses may degrade the efficiency of a program. There are three reasons for the degradation. First, the total time spent on unifying a particular goal with heads of clauses may increase when the number of clauses defining a predicate increases (the problem of increased unification cost). Second, the same goals are called repeatedly in different clauses to a greater extent after learning than before (the problem of increased inter-clause repetition of goal calls). Third, for some goal calls, the number of ways to succeed increases when learned clauses are added redundantly, and hence the number of times subsequent goals may be called increases (the redundancy problem). There have been a number of approaches to solve (or reduce) the three problems. These include techniques for reducing the number of goals called in different clauses [11, 20, 15], indexing of learned clauses [17] and techniques that avoid redundancy [1, 3]. However, none of the previous approaches addresses all three problems,

and notably, the methods that avoid redundancy do in fact increase the two first problems.

In this work, we present EGU II, a reformulation of one of the algorithms that avoid redundancy, EGU (Example-Guided Unfolding) [1]. The new algorithm shows that redundancy can be avoided without increasing the two first problems. Moreover, the algorithm EGU II has been combined with a technique for organizing learned clauses efficiently [2], and this combination is the first approach to address all three problems.

In the next section, we give definitions of the three program transformation rules (definition, unfolding and folding) upon which the algorithm EGU II is based. In section three, we present the algorithm and show that it preserves the equivalence of the domain theory. In section four, we present experimental results from comparing the algorithm to both EGU and adding clauses redundantly. Finally, in section five we give concluding remarks and point out some future research directions.

2 Preliminaries

In the following we assume the reader to be familiar with the standard terminology in logic programming [10]. The following rules for transformation of a definite program (below referred to as P) are taken from [19], where formal definitions can be found as well as proofs of their equivalence preserving properties.

Rule 1. Definition
Add to P a clause C of the form $p(x_1, \ldots, x_n) \leftarrow A_1, \ldots, A_m$ where p is a predicate symbol not appearing in P, $x_1, \ldots, x_n$ are distinct variables and $A_1, \ldots, A_m$ are literals whose predicate symbols all appear in P.

Rule 2. Unfolding
Let C be a clause in P, A a goal in its body and $C_1, \ldots, C_n$ be all clauses in P whose heads are unifiable with A. Let $C'_i (1 \leq i \leq n)$ be the result of resolving C with C_i upon A. Then replace C with $C'_1, \ldots, C'_n$.

Rule 3. Folding
Let C be a clause in P of the form $A \leftarrow A_1, \ldots, A_{i+1}, \ldots, A_{i+m}, \ldots, A_n$ and C_1 be a clause that previously have been introduced by the rule of definition of the form $B \leftarrow B_1, \ldots, B_m$. If there is a substitution θ such that $A_{i+1}, \ldots, A_{i+m} = B_1, \ldots, B_m\theta$ where θ substitutes distinct variables for the internal variables of C_1 and moreover those variables do not occur in $A, A_1, \ldots, A_i$ or $A_{i+m+1}, \ldots, A_n$, then replace C by the clause $A \leftarrow A_1, \ldots, A_i, B\theta, A_{i+m+1}, \ldots, A_n$.

3 Example-Guided Unfolding Revisited

In this section, we first review how the algorithm EGU works using an example. We point out in what way the problems of increased unification cost and increased inter-clause repetition of goal calls become more significant when using EGU compared to adding learned clauses redundantly to the program. We then present a solution to this problem: the algorithm EGU II. The algorithm is illustrated using the same example and is shown to preserve the equivalence of the domain theory.

3.1 The Algorithm EGU

The algorithm EGU is a reformulation of PROLOG-EBG [9] in terms of definition, unfolding and folding [1]. In contrast to the previous formulation, a learned clause is not supposed to be added redundantly but is derived while the domain theory is transformed.

Example [1] Let the domain theory be the simple english grammar shown in Figure 1 and the target concept be `:-s(X,Y)`. Let the training example be represented by the training instance `:-s([sue,loves,a,man],[])` and the training clauses: {`(F1) nm([sue|X],X)`, `(F2) tv([loves|X],X)`, `(F3) d([a|X],X)`, `(F4) n([man|X],X)`}. Then there is a SLD-refutation of the training instance, given the domain theory and training clauses, for which the sequence of input clauses is $R1, R3, F1, R5, F2, R2, F3, F4$. Let the operationality criterion be defined by the following predicate symbols $\{d, n, nm, iv, tv\}$. Removing clauses defining operational predicates from the sequence results in $R1, R3, R5, R2$. This sequence is used by EGU to guide unfolding. It is done in the following way.[2]

The first clause in the sequence ($R1$) is selected and the first non-operational goal in its body is unfolded. Then $R1$ is replaced with the following clauses:

```
(R6)  s(X,Z):- d(X,Y),n(Y,Y2),vp(Y2,Z).
(R7)  s(X,Z):- nm(X,Y),vp(Y,Z).
```

The resolvent $R7$ of the selected clause $R1$ and the next clause in the sequence $R3$ is then selected. The first non-operational goal in $R7$ is unfolded, giving two clauses:

```
(R8)  s(X,Z):- nm(X,Y),iv(Y,Z).
(R9)  s(X,Z):- nm(X,Y),tv(Y,Y2),np(Y2,Z).
```

The resolvent $R9$ of the selected clause $R7$ and the next clause in the sequence $R5$ is then selected. The first non-operational goal in $R9$ is unfolded, giving two clauses:

---

[1] The standard Edinburgh syntax for logic programs is used [4].

[2] This example is somewhat simplified. In EGU the rules of definition and folding are used in addition to unfolding to overcome a problem associated with recursive domain theories. However, in non-recursive domain theories the application of these rules in the algorithm is superfluos.

```
(R10) s(X,Z):- nm(X,Y),tv(Y,Y2),d(Y2,Y3),n(Y3,Z).
(R11) s(X,Z):- nm(X,Y),tv(Y,Y2),nm(Y2,Z).
```

The resolvent $R10$ of the selected clause $R9$ and the last clause in the sequence $R2$ is then finally selected. This clause is placed first in the program. The resulting domain theory is shown in Figure 2.

It can be observed that the problems of increased unification cost and inter-clause repetition of goal calls have become more significant in comparison to adding the learned clause ($R10$) redundantly to the original domain theory. Instead of two clauses defining the target concept, there are four in the transformed domain theory. Moreover, the goal **nm(X,Y)** may in the worst case be called three times in the transformed theory (not including repetition within a clause), and only two times when adding the clause redundantly.

```
(R1) s(X,Z):- np(X,Y),vp(Y,Z).
(R2) np(X,Z):- d(X,Y),n(Y,Z).
(R3) np(X,Y):- nm(X,Y).
(R4) vp(X,Y):- iv(X,Y).
(R5) vp(X,Z):- tv(X,Y),np(Y,Z).
```

Fig. 1. Original domain theory.

```
(R10) s(X,Z):- nm(X,Y),tv(Y,Y2),d(Y2,Y3),n(Y3,Z).
(R6)  s(X,Z):- d(X,Y),n(Y,Y2),vp(Y2,Z).
(R8)  s(X,Z):- nm(X,Y),iv(Y,Z).
(R11) s(X,Z):- nm(X,Y),tv(Y,Y2),nm(Y2,Z).
(R2)  np(X,Z):- d(X,Y),n(Y,Z).
(R3)  np(X,Y):- nm(X,Y).
(R4)  vp(X,Y):- iv(X,Y).
(R5)  vp(X,Z):- tv(X,Y),np(Y,Z).
```

Fig. 2. Domain theory after applying EGU.

3.2 The Algorithm EGU II

We first informally describe the algorithm EGU II, and illustrate it using the grammar example. Then we formally describe the algorithm, and show that the algorithm produces a program that is equivalent to the original domain theory.

Informal description of EGU II The problem of increased unification cost when using EGU is due to the unfolding of a goal that unifies with the head of more than one clause, since the clause in which the goal appears is replaced with more than one clause. Moreover, all goals that precede the goal in the original clause are repeated in the clauses that replace the original one, thus increasing the inter-clause repetition of goal calls. Note that this is a potential problem for all techniques that are based on unfolding (e.g. partial evaluation [16] and lazy partial evaluation [3]). This problem is solved by EGU II in the following way.

Instead of unfolding the first non-operational goal in a selected clause, which is done in EGU, a new predicate is defined, that is equivalent to the conjunction consisting of the first non-operational goal and the subsequent goals in the selected clause (definition). The first goal in the clause defining the new predicate is then unfolded, yielding a new set of clauses. The input sequence is then used to select one of these clauses, that is processed in the same way as the first clause. This process continues until one clause is finally selected after having iterated through the input sequence.

The body of each previously selected clause (except the last one) is then folded using the new definitions.

The finally selected clause is then resolved with the only clause calling a goal that unifies with the head of the selected clause. The selected clause is removed and the resolvent is then treated in the same way as the first clause. This process is iterated until the final resolvent is selected, which is then placed first in the program.

Finally, goals that unify with one clause only are unfolded (since this is guaranteed to improve efficiency).

Example revisited Let the domain theory, operationality criterion and the input sequence $(R1, R3, R5, R2)$ from the previous example be the input to EGU II.

The first clause in the sequence $(R1)$ is selected, and the first non-operational goal in its body is `np(X,Y)`. A new predicate $(p1)$ is defined by the clause:

```
(R6) p1(X,Z):- np(X,Y),vp(Y,Z).
```

Unfolding upon the first goal in $R6$, gives two new clauses:

```
(R7) p1(X,Z):- d(X,Y2),n(Y2,Y),vp(Y,Z).
(R8) p1(X,Z):- nm(X,Y),vp(Y,Z).
```

Then $R8$ is selected since it is the resolvent of $R6$ and the next clause in the sequence $(R3)$. The first non-operational goal in $R8$ is `vp(Y,Z)` and a new

predicate ($p2$) is defined by the clause:

```
(R9) p2(Y,Z):- vp(Y,Z).
```

Unfolding upon the first goal in $R9$, gives two new clauses:

```
(R10) p2(Y,Z):- iv(Y,Z).
(R11) p2(Y,Z):- tv(Y,Y2),np(Y2,Z).
```

Then $R11$ is selected since it is the resolvent of $R9$ and the next clause in the sequence ($R5$). The first non-operational goal in $R11$ is `np(Y2,Z)` and a new predicate ($p3$) is defined by the clause:

```
(R12) p3(Y2,Z):- np(Y2,Z).
```

Unfolding upon the first goal in $R12$, gives two new clauses:

```
(R13) p3(Y2,Z):- d(Y2,Y3),n(Y3,Z).
(R14) p3(Y2,Z):- nm(Y2,Z).
```

Then $R13$ is the finally selected clause since it is the resolvent of $R12$ and the last clause in the sequence ($R2$). In the second step of the algorithm, the body of $R1$ is folded using $R6$, the body of $R8$ is folded using $R9$, and the body of $R11$ is folded using $R12$. The transformed domain theory is shown in Figure 3.

```
(R1')  s(X,Z):- p1(X,Z).
(R2)   np(X,Z):- d(X,Y),n(Y,Z).
(R3)   np(X,Y):- nm(X,Y).
(R4)   vp(X,Y):- iv(X,Y).
(R5)   vp(X,Z):- tv(X,Y),np(Y,Z).
(R7)   p1(X,Z):- d(X,Y2),n(Y2,Y),vp(Y,Z).
(R8')  p1(X,Z):- nm(X,Y),p2(Y,Z).
(R10)  P2(Y,Z):- iv(Y,Z).
(R11') p2(Y,Z):- tv(Y,Y2),p3(Y2,Z).
(R13)  p3(Y2,Z):- d(Y2,Y3),n(Y3,Z).
(R14)  p3(Y2,Z):- nm(Y2,Z).
```

Fig. 3. Domain theory after the two first steps in EGU II.

In the third step of the algorithm, the selected clause ($R13$) is used to resolve upon the only goal that unifies with the head of the clause. By resolving upon the goal `p3(Y2,Z)` in $R11'$, the following clause is obtained:

```
(R15) p2(Y,Z):- tv(Y,Y2),d(Y2,Y3),n(Y3,Z).
```

The clause $R13$ is now redundant and is removed. The clause $R15$ is then used to resolve upon the goal `p2(Y,Z)` in $R8'$, resulting in the clause:

```
(R16) p1(X,Z):- nm(X, Y),tv(Y,Y2),d(Y2,Y3),n(Y3,Z).
```

The clause $R15$ is then removed. The clause $R16$ is then, before being removed, used to resolve upon the only goal that unifies with the head, and that is the goal in $R1'$, resulting in the finally selected clause, which is placed first in the program:

```
(R17) s(X,Z):- nm(X,Y),tv(Y,Y2),d(Y2,Y3),n(Y3,Z).
```

In the fourth step of the algorithm, all goals that can be reduced by one

clause only are unfolded. The final domain theory (after removing dead code cf. [6]) is shown in Figure 4. It can be observed that the problem of increased unification cost and increased inter-clause repetition of goal calls are not more significant in comparison to adding the clause $R17$ redundantly.

```
(R17)  s(X,Z):- nm(X,Y),tv(Y,Y2),d(Y2,Y3),n(Y3,Z).
(R1')  s(X,Z):- p1(X,Z).
(R2)   np(X,Z):- d(X,Y),n(Y,Z).
(R3)   np(X,Y):- nm(X,Y).
(R4)   vp(X,Y):- iv(X,Y).
(R5)   vp(X,Z):- tv(X,Y),np(Y,Z).
(R7)   p1(X,Z):- d(X,Y2),n(Y2,Y),vp(Y,Z).
(R8')  p1(X,Z):- nm(X,Y),p2(Y,Z).
(R10)  P2(Y,Z):- iv(Y,Z).
(R11'')p2(Y,Z):- tv(Y,Y2),nm(Y2,Z).
```

Fig. 4. Final domain theory after applying EGU II.

Algorithm EGU II

Input: a definite program P (domain theory), a definite unit goal : $-T$ (target concept), an operationality criterion O and a sequence of clauses $C_1, \ldots, C_m$ (proof of training example).

Output: a definite program P

Let $S_1 = C_1$.

FOR $i = 2$ TO m DO

Let B_j be the first non-operational goal in the body of $S_{i-1} = H : -B_1, ..., B_n$, that unifies with the heads of the clauses $E_1, \ldots, E_p$.

Let B'_j be defined by the clause $D_i = B'_j : -B_j, \ldots, B_n$ where the arguments of B'_j are all variables in $B_j, ..., B_n$ that appear in $H, B_1, \ldots, B_{j-1}$ (definition).

Replace D_i with $R_1, \ldots, R_p$, where $R_k (1 \leq k \leq p)$ is the resolvent of D_i and E_k upon B_j (unfolding).

Let S_i be the resolvent R_k of D_i and E_k such that $E_k = C_i$.

FOR $i = 1$ TO $m - 1$ DO

Replace $S_i = H : -B_1, ..., B_n$ with a clause $S'_i = H : -B_1, \ldots, B_{j-1}, B'_j$ using $D_{i+1} = B'_j : -B_j, \ldots, B_n$ (folding).

Let $C = S_m$.

FOR $i = 1$ TO $m - 1$ DO

Let C' be the resolvent of C and the clause in P in which body there is a goal that unifies with the head of C.

Remove C and let $C = C'$.

Place C first in P.

Unfold all goals that can be reduced by one clause only.

Correctness of EGU II The algorithm EGU II produces a new domain theory that is equivalent to the original domain theory with respect to a target concept T, i.e. an instance $T\theta$ follows from the original domain theory if and only if $T\theta$ follows from the transformed domain theory. To see this, we look at each step in the algorithm.

According to a theorem in [19], any program that is obtained by unfolding and folding, from a program P and a set of clauses introduced by the rule of definition D, is equivalent to $P \cup D$. Thus the program P' that is obtained after the two first steps in EGU II is equivalent to the original domain theory P together with the set of new definitions D. Hence, an instance $T\theta$ follows from P' if and only if it follows from $P \cup D$. Moreover, an instance of the target concept $T\theta$ follows from P if and only if it follows from $P \cup D$, since the predicates defined in D do not appear in P. Thus P' is equivalent to P with respect to a target concept T.

The third step in the algorithm does also preserve the equivalence of the domain theory with respect to a target concept as is shown by the following. Let P be a definite program (domain theory), T a unit goal (target concept) and C_1 a clause in P, in which there is a goal G with a different predicate symbol from T and that unifies with the head of another clause C_2 in P. If there is no other goal in P that unifies with the head of C_2, then P with respect to T is equivalent to the program obtained by adding the resolvent of C_1 and C_2 upon G to P, and removing C_2. This is the case since the resolvent follows logically from P and thus can be added, and C_2 cannot be used to reduce any other goal in P and not any instance of T and thus can be removed. The above conditions hold in the third step of the algorithm since each goal that is resolved upon only appears in one clause and has a different predicate symbol than T.

The fourth step in the algorithm preserves equivalence of the domain theory since it only involves unfolding (see [19]).

4 Experimental Results

The significance of the three problems that may degrade the efficiency of a program due to learned clauses are domain dependent. For example, if no predicates are declared as operational (e.g. the theorem proving domain in [14]) then the problem of increased inter-clause repetition of goal calls does not occur, since this problem only involves operational goals. If learned clauses are not invoked on other goals than on top level unit goals, then the redundancy problem is of no importance. However, if some goals are specified as operational or learned clauses are allowed to be invoked from other clauses (e.g. recursively), then these problems can be of major importance. Note that the redundancy problem can increase repetition of non-operational as well as operational goal calls.

In this section we present results from experiments with a domain theory for theorem proving in the MIU system [8]. The algorithm EGU II is compared to EGU [1] and to adding learned clauses redundantly (cf. PROLOG-EBG [9]). All three algorithms have been extended by a technique for organizing learned

clauses efficiently, called CLORG [2]. The effect of disallowing learned clauses from being invoked recursively is also investigated. The algorithms have been implemented in SICStus PROLOG 2.1, and the experiments have been run on a Personal DECstation 5000 with 16 Mb PM.

The MIU System A theorem in the MIU system is a string $w \in m(u \cup i)*$, and there are four rules of inference: i) if wi is a theorem, then so is wiu ii) if mw is a theorem, then so is mww iii) if w_1iiiw_2 is a theorem then so is w_1uw_2 iv) if w_1uuw_2 is a theorem then so is w_1w_2. The problem of finding a sequence of rule applications from an axiom to a theorem can be solved by the program in the appendix.

The test examples in this experiment were produced by finding all strings that could be proved as theorems with not more than six rule applications given mi as an axiom (282 theorems). A subset of the test examples was used as a training set. In the experiment, the size of the subset varied from 10% to 100% of the test set. In Figure 5, it can be seen that the cost of using EGU is higher than the benefits of learning (when the size of the training set is less than 80%). The reason for this is the increased unification cost and inter-clause repetition of goal calls. In this domain it is certainly better to add clauses redundantly (organized efficiently) than to use EGU. However, redundancy can be eliminated at lower cost as is shown by EGU II. In fact, the resulting program is more efficient than the program obtained by adding clauses redundantly. This would not have been a surprise had the redundancy problem been significant (as in [1]), but this is not the case in this domain since the target concept is never backtracked into. This result can however be explained by the elimination of failure branches that is obtained when the domain theory is transformed. To show that the increase in efficiency was not caused by elimination of redundancy, we also present results from an experiment where learned clauses were not invoked recursively (Figure 6).

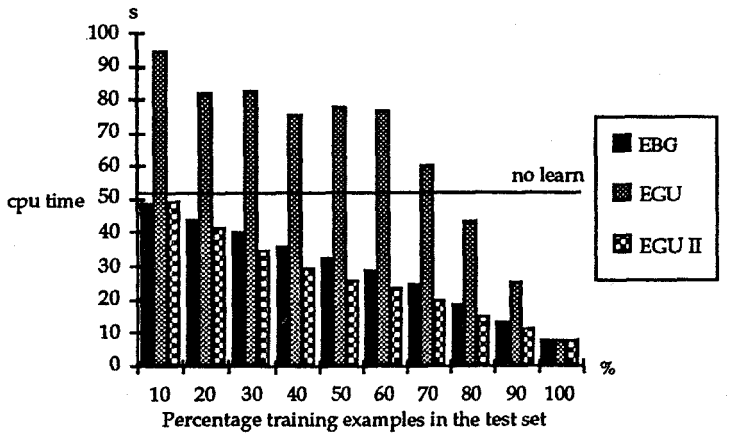

Fig. 5. Comparing three ways of transforming the MIU domain theory.

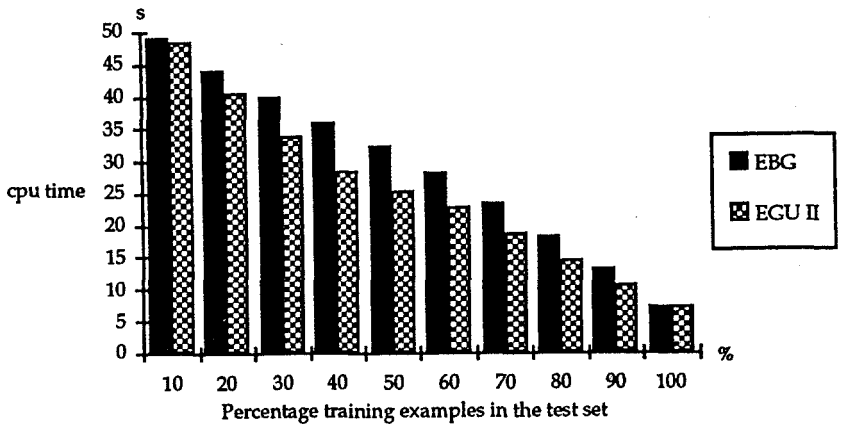

Fig. 6. Using EBG and EGU II when chaining of learned clauses is not allowed.

5 Concluding Remarks

There are three potential causes of the decreased efficiency of a logic program when adding clauses learned by EBG: i) increased unification cost ii) increased inter-clause repetition of goal calls iii) increased redundancy. None of the previous approaches for reducing these problems has addressed all three problems, and notably, the methods that avoid redundancy increase the two first problems.

We have presented a solution to this problem: EGU II, which is a reformulation of one of the previous algorithms that avoid redundancy (Example-Guided Unfolding). By the new algorithm it is shown how to avoid redundancy without increasing the two first problems. Moreover, the algorithm has been combined with a technique for organizing learned clauses efficiently, thus showing that all three problems can be addressed at a time.

Experimental results have been presented showing that the cost of avoiding redundancy by EGU may outweigh the benefits compared to adding clauses redundantly. However, this cost can be significantly reduced by the algorithm EGU II, as was shown by the experiments. The experiments also showed that even when the redundancy problem is not significant, it can be more beneficial to use EGU II than to add clauses redundantly.

There are several interesting directions for future research. One direction is to investigate under what conditions the algorithm is beneficial. These conditions include the distribution of problems, the cost of testing whether learned clauses are applicable, and the benefits when they are successfully applied.

Another question is the problem of generalizing number (e.g. [5, 18, 7]). A direction for future research is to investigate how clauses learned by methods for generalizing number can be incorporated without introducing redundancy.

Appendix

```
% The call solve(Axiom,Theorem,Length,[Axiom], Sequence) will find a
% Sequence of length less or equal to Length from Axiom to Theorem.
% E.g. ?- solve([m,i],[m,i,u,i,u],s(s(0)),[[m,i]],Sequence).
%              Sequence = [add_u, double]

solve(State,State,_,_,[]).
solve(State1,State2,s(Length),History,[Operator|Solution]):-
   successor(Operator,State1,State3),
   not_member(State3,History),
   solve(State3,State2,Length,[State3|History],Solution).

successor(add_u,L1,L2):-
   append(_,[i],L1),
   append(L1,[u],L2).
successor(double,[m|L1],[m|L2]):-
   append(L1,L1,L2).
successor(iii_to_u,L1,L2):-
   append(L3,[i,i,i|L4],L1),
   append(L3,[u|L4],L2).
successor(delete_uu,L1,L2):-
   append(L3,[u,u|L4],L1),
   append(L3,L4,L2).

append([],L,L).
append([X|L1],L2,[X|L3]):-
   append(L1,L2,L3).

not_member(_,[]).
not_member(X,[Y|L]):-
   X\==Y,
   not_member(X,L).

% Operationality criterion: operational(\==(_,_)).
```

References

1. Boström H., "Eliminating Redundancy in Explanation-Based Learning", *Machine Learning: Proceedings of the 9th International Conference*, Morgan Kaufmann, CA (1992) 37–42
2. Boström H., *Efficient Organization of Clauses Learned by Explanation-Based Generalization*, SYSLAB Report, Dept. of Computer and Systems Sciences, Stockholm University (1993)

3. Clark P. and Holte R., "Lazy Partial Evaluation: an Integration of Explanation-Based Generalization and Partial Evaluation", *Machine Learning: Proceedings of the 9th International Conference*, Morgan Kaufmann, CA (1992) 82–91
4. Clocksin W. F. and Mellish C. S., *Programming in Prolog*, Springer Verlag, Berlin Heidelberg (1981)
5. Cohen W. W., "Generalizing Number and Learning from Multiple Examples in Explanation-Based Learning", *Proceedings of the Fifth International Conference on Machine Learning*, Morgan Kaufmann, CA (1988) 256–269
6. Debray S. K., *Global Optimization of Logic Programs*, Ph.D. thesis, Stony Brook (1986)
7. Feldman R. and Subramanian D., "Example-Guided Optimization of Recursive Domain theories", *Proceedings of Conference on Artificial Intelligence Applications*, Miami Beach, Florida, IEEE (1991) 240–244
8. Hofstadter D. R., *Godel, Escher, Bach: an Eternal Golden Braid*, Penguin Books, New York (1980)
9. Kedar-Cabelli S. and McCarty L. T., "Explanation-based generalization as resolution theorem proving", *Proceedings of the Fourth International Machine Learning Workshop*, Morgan Kaufmann, CA (1987) 383–389
10. Lloyd J. W., *Foundations of Logic Programming*, Springer-Verlag (1987)
11. Minton S., *Learning Effective Search Control Knowledge: An Explanation-Based Approach* , Ph.D. thesis, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA (1988)
12. Minton S., "Issues in the Design of Operator Composition Systems", *Proceedings of the Seventh International Conference on Machine Learning*, Morgan Kaufmann, CA (1990) 304–312
13. Mitchell, T. M., Keller R. M. and Kedar-Cabelli S. T., "Explanation-Based Generalization: A Unifying View", *Machine Learning* **1**, (1986) 47–80
14. Mooney R., "The Effect of Rule Use on the Utility of Explanation-Based Learning", *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, CA (1989) 725–730
15. Sablon G., De Raedt L. and Bruynooghe M., "Generalizing Multiple Examples in Explanation Based Learning", *Proceedings of the International Workshop AII 2*, Reinhardsbrunn, GDR (1989) 177–183
16. Sahlin D., *An Automatic Partial Evaluator for Full Prolog*, Ph.D. thesis, Dept. of Tele-communication and Computer Systems, The Royal Institute of Technology, Stockholm (1991)
17. Samuelsson C. and Rayner M., "Quantitative Evaluation of Explanation–Based Learning as an Optimization Tool for a Large Scale Natural Language System", *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, CA (1992) 609–615
18. Shavlik J. W., "Acquiring Recursive Concepts with Explanation-Based Learning", *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, CA (1989) 688–693
19. Tamaki H. and Sato T., "Unfold/Fold Transformations of Logic Programs", *Proceedings of the Second International Logic Programming Conference*, Uppsala University, Uppsala, Sweden (1984) 127–138
20. Wogulis J. and Langley P., "Improving Efficiency by Learning Intermediate Concepts", *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, CA (1989) 657–662