UNFOLD/FOLD TRANSFORMATION OF LOGIC PROGRAMS

Hisao TAMAKI
Ibaraki University
Dept. of Information Science
Hitachi, 316, Japan

Taisuke SATO
Electrotechnical Laboratory
Machine Inference Section
Umezono, Sakura-mura, 305
Japan

ABSTRACT

The unfold/fold transformation method is formulated for logic programs in such a way that the transformation always preserves the equivalence of programs as defined by the least model semantics. A detailed proof for the basic system is presented first. Then some augmenting rules are introduced and the conditions of their safe application within the unfold/fold system are clarified. There are useful special cases of those rules whose application is always safe.

1 INTRODUCTION

The unfold/fold program transformation method was developed by Burstall and Darlington (Burstall & Darlington 1977) in the context of their recursive equation language. The idea was generalized and applied to logic program synthesis(Clark & Sickel 1977)(Hogger 1981), where the authors naturally formulated the unfold and fold transformations as just special cases of logical deduction. Thus each clause in the synthesized program is a theorem deduced from the specification axioms. This ensures the partial correctness of the synthesized program because every result of computation (atomic theorem deduced from the program) is derivable directly from the specification as well. Total correctness, however, is not guaranteed in general and should be proved separately(Clark 1979).

They applied this deductive approach to logic program transformation taking the initial program, viewed as if-and-only-if definitions, to be the specification. But what is ensured in general is again just partial correctness: the relations computed by the transformed program are narrower or equal to those computed by the original one. In other words the least Herbrand model(Van Emden & Kowalski 1976) of the transformed program is included in that of the initial one. If we want exact equivalence, the inverse inclusion should be proved for individual cases.

As an alternative to the deductive approach, we have formulated an unfold/fold transformation system for logic programs(Tamaki & Sato 1983) in such a way that the transformation always preserves the equivalence of programs as defined by the least model semantics. Though we have to sacrifice the generality of the deductive approach, the guaranteed equivalence should worth the cost.

This paper augments the basic unfold/fold system with

some other transformation rules. Though the rules themselves are obviously equivalence preserving, their interaction with unfold/fold transformation needs careful study. The condition for the application of the rules to be safe will be clarified.

Section 2 describes the basic unfold/fold system and proves that it preserves the equivalence of programs. The proof is simpler than the one given in (Tamaki & Sato 1983) and more suitable for our purpose. Section 3 and 4 introduce and study augmenting rules.

The readers are assumed to be familiar with the standard notions and notations of logic programs (Kowalski 1974). Note that our target language is a pure one rather than a practical implementation such as existing Prologs. Thus a *program* is a set (not an ordered list) of definite clauses. A *definite clause* is a pair of a goal(atomic formula), called a *head*, and a multi-set (again not an ordered list) of goals called a *body*.

2 BASIC UNFOLD/FOLD SYSTEM

2.1 Description of the system

The transformation process proceeds as follows.

Transformation process

```
begin P0 := the initial program;
      D0 := {}; /* the set of
                   definitions of
                   new predicates */
      mark every clause in P0
      'foldable';
      for i := 1 to arbitrary N
          apply any of the trans-
          formation rules to ob-
          tain Pi and Di from
          Di-1
end
```

In this section we are only concerned with the three basic rules, namely, *definition*, *unfolding* and *folding*, each of which are described in the sequel.

Example (initial program)

P_0 :
C1. subseq([],X)
C2. subseq([A|X],[A|Y]) ← subseq(X,Y)
C3. subseq(X,[A|Y]) ← subseq(X,Y)

We use this example to illustrate the process and rules of transformation. The upper case letters are variables, [] denotes an empty list and [A|X] a list with head A and tail X. Thus the predicate subseq(X,Y) is intended to mean that X is a subsequence of Y.

Rule 1. definition

Let C be a clause of the form $p(x_1,\ldots,x_n) \leftarrow A_1,\ldots,A_m$, where

1. p is an arbitrary predicate not appearing in P_{i-1} or D_{i-1},
2. $x_1,\ldots,x_n$ are distinct variables, and
3. $A_1,\ldots,A_m$ are goals whose predicates all appear in P_0.

Then let P_i be $P_{i-1} \cup \{C\}$ and D_i be $D_{i-1} \cup \{C\}$.
Do not mark C 'foldable'.

The predicates introduced by the definition rule are called *new* predicates while those in P_0 are called *old*. Those variables occurring in $A_1,\ldots,A_m$ other than $x_1,\ldots,x_n$ are called *internal variables* of C.

Example (continued)

We define C4, motivated by some need for a common subsequence relation.

C4. csub(X,Y,Z) ← subseq(X,Y), subseq(X,Z)

Then $P_1 = \{\underline{C1},\underline{C2},\underline{C3},C4\}$, $D_1 = \{C4\}$. Underline indicates 'foldable' clauses. We are going to optimize this predicate 'csub'.

Rule 2. Unfolding

Let C be a clause in P_{i-1}, A a goal in its body and $C_1,\ldots,C_n$ be all the clauses in P_{i-1} 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 let P_i be $(P_{i-1} - \{C\}) \cup \{C_1',\ldots,C_n'\}$ and D_i be D_{i-1}. Mark each C_i' 'foldable' unless it is already in P_{i-1}.

Example (continued)

We unfold C4 at its first goal to obtain $P_2 = \{\underline{C1},\underline{C2},\underline{C3},\underline{C5},\underline{C6},\underline{C7}\}$, $D_2 = \{C4\}$ where the clauses C5,C6 and C7 are listed below.

C5. csub([],Y,Z) ← subseq([],Z)

C6. csub([A|X],[A|Y],Z) ← subseq(X,Y),subseq([A|X],Z)

C7. csub(X,[A|Y],Z) ← subseq(X,Y), subseq(X,Z)

Then C5 is unfolded into

C5'. csub([],Y,Z)

and we get $P_3 = \{\underline{C1},\underline{C2},\underline{C3},\underline{C5}',\underline{C6},\underline{C7}\}$ and $D_3 = \{C4\}$.

The folding rule in our system is not just the inverse of the unfolding rule as it is in the Burstall and Darlington's system. To fold a goal set into a goal, we allow only a clause in D_{i-1} to be used as the folder.

Rule 3. folding

Let C be a clause in P_{i-1} of the form $A \leftarrow A_1,\ldots,A_n$ and C_1 be a clause in D_{i-1} of the form $B \leftarrow B_1,\ldots,B_m$. Suppose there is a substitution θ and a subset $\{A_{i_1},\ldots,A_{i_m}\}$ of the body of C such that the following conditions hold.

1. $A_{i_j} = B_j\theta$ for $j = 1,\ldots,m$,
2. θ substitutes distinct variables for the internal variables of C_1, and moreover those variables do not occur in A or $\{A_1,\ldots,A_n\} - \{A_{i_1},\ldots,A_{i_m}\}$, and
3. C is marked 'foldable' or $m < n$.

Then let P_i be $(P_{i-1} - \{C\}) \cup \{C'\}$ and D_i be D_{i-1} where C' is a clause with head A and body $(\{A_1,\ldots,A_n\} - \{A_{i_1},\ldots,A_{i_m}\}) \cup \{B\theta\}$. Let C' inherit the mark of C.

Example (continued)

Folding the whole body of C7 by C4, we obtain $P_4 = \{\underline{C1},\underline{C2},\underline{C3},\underline{C5}',\underline{C6},\underline{C8}\}$ and $D_4 = \{C4\}$ where C8 is

C8. csub(X,[A|Y],Z) ← csub(X,Y,Z).

To see the need for the condition 2, suppose we fold the clause p(X) ← q(X,Y),r(Y) using a definition s(U) ← q(U,V) into the clause p(X) ← s(X),r(Y). Then the equivalence is destroyed because

the result clause would correspond to a clause $p(X) \leftarrow q(X,Y1),r(Y)$ but not to the original one.

The condition 3 prevents for example immediate folding of a definition by itself. Without the condition we fold C4, in P_0 of our example, by itself to end in P_1' $=\{C1,C2,C3,C4'\}$ where C4' is

$$csub(X,Y,Z) \leftarrow csub(X,Y,Z).$$

To complete our example, we need one more new predicate.

Example (continued)

Motivated by the failure to fold the body of C6, we introduce an auxiliary predicate 'csub1' and define

C9. $csub1(A,X,Y,Z) \leftarrow subseq(X,Y), subseq([A|X],Z)$

to obtain $P_5 = \{C1,C2,C3,C5',C6,C8,C9\}$, $D_5 = \{C4,C9\}$.

By unfolding C9 at its second goal, we get $P_6 = \{C1,C2,C3,C5',C6,C8,C10,C11\}$ and $D_6 = \{C4,C9\}$ where C10 and C11 are

C10. $csub1(A,X,Y,[A|Z]) \leftarrow subseq(X,Y),subseq(X,Z)$

C11. $csub1(A,X,Y,[B|Z]) \leftarrow subseq(X,Y),subseq([A|X],Z).$

Folding C6, C10 and C11, we obtain the final result $P_9 = \{C1,C2,C3,C5',C6',C8,C10',C11'\}$ and $D_9 = \{C4,C9\}$. Note that C5',...,C11' listed below define the new predicates independently from $P_0 = \{C1,C2,C3\}$.

C5'. $csub([],Y,Z)$

C6'. $csub([A|X],[A|Y],Z) \leftarrow csub1(A,X,Y,Z)$

C8. $csub(X,[A|Y],Z) \leftarrow csub(X,Y,Z)$

C10'. $csub1(A,X,Y,[A|Z]) \leftarrow csub(X,Y,Z)$

C11'. $csub1(A,X,Y,[B|Z]) \leftarrow csub1(A,X,Y,Z)$

When used for generating common subsequences of two given lists, the final program is far more deterministic than the original one because a selection of an element in the first list is immediately checked against the second one. (Of course we are assuming here the fixed order control under which the original program behaves as a typical generare-and-test program.)

The point is that P_9 is equivalent (in the least model semantics) to $P_0 \cup D_9$ and that this is generally true for any transformation sequence obeying the rules. The rest of this section is devoted to the proof of this fact.

2.2 Correctness of the Basic System

First we characterize the least model semantics by means of proof trees. We assume a fixed Herbrand universe and a fixed set of predicates so that the set of ground goals is fixed.

Definition. proof tree

Let S be a program. A tree T, whose nodes are labelled with ground goals, is called a *proof tree*, or simply a *proof*, in S if the following conditions hold.

1. Let A be the root label of T, $T_1,\ldots,T_n$ $(n \geq 0)$ its immediate subtrees and $A_1,\ldots,A_n$ their root labels. Then $A \leftarrow A_1,\ldots,A_n$ is an instance of some clause C in S.

2. Each immediate subtree T_i ($1 \leq i \leq n$) is a proof in S.

We say that T is a proof of A in S and that A is provable (by T) in S. We also say that the clause C is used at the root of the proof T and that $T_1,..,T_n$ are immediate subproofs of T.

In the following, we often argue by induction on the structure of proofs and omit the base case, which is usually subsumed by the induction step as the special case $n=0$, as in the above definition.

The meaning, $M(S)$, of the program S is now defined as the set of all ground goals provable in S. This $M(S)$ is nothing but the least Herbrand model of S (Van Emden 76).

For a transformation sequence $(P_0,D_0),\ldots,(P_N,D_N)$, we define a sequence $S_0,\ldots,S_N$ called *virtual transformation sequence*, by

$$S_i = P_i \cup (D_N - D_i).$$

In particular $S_0 = P_0 \cup D_N$ and $S_N = P_N$. In the following discussion we will always deal with virtual transformation sequences. This amounts to pretending that the definitions of all new predicates are given at the beginning. The set of definitions D_N will be fixed and referred to as D throughout. Since the definition transformation is an identity transformation in the virtual transformation sequence, it will be ignored.

THEOREM

Let $S_0,\ldots,S_N$ be the transformation sequence. Then $M(S_N) = M(S_0)$.

To prove the theorem we need some definitions.

Definition. rank of a ground goal

Let A be a goal in $M(S_0)$ and $r'(A)$ be the size of the smallest proof of A in S_0. Then $r(A)$, the *rank* of A, is $r'(A)$ if A has an old predicate and $r'(A)-1$ if A has a new predicate.

Definition. rank consistent proof

Let S_i be a program in the transformation sequence. Let T be a proof in S_i, C the clause used at its root, $T_1,\ldots,T_n$ ($n \geq 0$) its immediate subproofs, and A, $A_1,\ldots,A_n$ their root labels. Then T is said to be *rank-consistent* if

1. $r(A) \geq r(A_1)+..+r(A_n)$ with equality holding only when C is not marked 'foldable', and
2. $T_1,..,T_n$ are rank consistent.

Now the proof of the theorem consists of showing that the following invariants hold for each i ($0 \leq i \leq N$).

I1. $M(S_i) = M(S_0)$

I2. For each goal A in $M(S_i)$, there is a rank-consistent proof of A in S_i.

The first invariant I1 trivially

holds for $i=0$. As for I2, for any goal A in $M(S_0)$, the smallest proof of A is obviously rank-consistent. (Remember $S_0 = P_0 \cup D$ and the clauses in P_0 are marked 'foldable' while those in D are not.)

The preservation of the invariants is proved in the three lemmas below.

LEMMA 1

If I1 holds for S_i, then $M(S_{i+1}) \subset M(S_i)$.

Proof.

Let A be a ground goal in $M(S_{i+1})$ and T its proof in S_{i+1}. We construct a proof T' of A in S_i by induction on the structure of T.

Let C be the clause used at the root of T, and $T_1,..,T_n$ $(n \geq 0)$ the immediate subproofs of T, By the induction hypothesis we can construct proofs $T_1',..,T_n'$ in S_{i+1} with each T_j' corresponding to T_j. If C is in S_i we can immediately construct T' from C and the proofs $T_1',..,T_n'$. If C is the result of unfolding, we can construct T' from $T_1',..,T_n'$ using the two clauses in S_i of which C is the resolvent.

Now suppose C is the result of folding. Then for some j $(1 \leq j \leq n)$, the root label A_j of T_j is an instance of the folded goal in the body of C. We assume $j=1$. Because A_1 is provable in S_i by T_1', it is also provable in S_0 by the invariant I1. So there should be a ground instance $A_1 \leftarrow B_1,..,B_m$ of some clause in D such that $B_1,..,B_m$ are provable in P_0. Again by I1, $B_1,..,B_m$ are provable in S_i. Let C' be the clause in S_i of which C is the folded result. Owing to the condition 2 of folding, we can combine the proofs of $B_1,..,B_m$ and proofs $T_2',..,T_n'$ with C' to obtain T', the proof of A in S_i. []

LEMMA 2

If the invariants I1 and I2 hold for S_i, then $M(S_i) \subset M(S_{i+1})$.

Proof.

Let A be a ground goal in $M(S_i)$. Then by the invariant there is a rank-consistent proof T of A in S_i. We construct a proof T' of A in S_{i+1} by induction on the well-founded ordering $>>$ defined on $M(S_0)$ $(=M(S_i))$ as

$A >> B$ iff

$r(A) > r(B)$ or

$r(A) = r(B)$ and A has a new and B has an old predicate.

The base case where $r(A)=1$ and A has an old predicate obviously holds because then A should be a ground instance of some unit clause in P_0 which should be in both S_i and S_{i+1}.

Let C be the clause in S_i used at the top of T, and $T_1,..T_n$ $(n \geq 0)$ the immediate subproofs of T. By the invariant I2, for each root label A_i of T_i, $A >> A_i$ holds. So by the induction hypothesis there are proofs

$T_1', .., T_n'$ of $A_1, .., A_n$ in S_{i+1}. If C is in S_{i+1} the construction of T' is immediate.

Suppose C is unfolded into $C_1, .., C_m$ in S_{i+1} and assume that the root label A_1 of T_1 is the instance of the goal at which C is unfolded. Let $T_{11}, .. T_{1p}$ be the immediate subproofs of T_1, and $A_{11}, .., A_{1p}$ their root labels. Then again by I2 and the induction hypothesis, there are proofs $T_{11}', .., T_{1p}'$ of $A_{11}, .., A_{1p}$ in S_{i+1}. Combining the proofs $T_{11}', .., T_{1p}', T_2', .., T_n'$ with some C_k $(1 \leq k \leq m)$ we get a proof T' of A in S_{i+1}.

Now suppose C is folded into C' in S_{i+1}. Assume that the root labels $A_1, .., A_k$ of $T_1, .., T_k$ $(k \leq n)$ are the instances of the folded goals in C. Let B be a goal such that $B \leftarrow A_1, .., A_k$ is a ground instance of the clause in D used in the folding. By definition, $r(A_1)+...+r(A_k) \geq r(B)$. By the condition 3 of folding, either C is marked 'foldable', which means $r(A) > r(A_1)+...+r(A_k)$, or $k < n$. In either cases, $r(A) > r(B)$ holds. Moreover, by the equivalence of S_i to S_0, B is provable in S_i. Therefore by the induction hypothesis, B has a proof T_B in S_{i+1}. Combining the proofs T_B, $T_{k+1}', .., T_n'$ with the clause C', we obtain the proof T' of A in S_{i+1}.

LEMMA 3

If the invariant I1 and I2 holds for S_i, I2 holds for S_{i+1}.

Proof.

We first note that in the proof of lemma 2, T' is constructed in such a way that it is rank-consistent. Thus every goal in $M(S_i)$ has a rank-consistent proof in S_{i+1}. Because $M(S_{i+1}) \subset M(S_i)$ by lemma 1, I2 holds for S_{i+1}. []

This completes the proof of the theorem.

3 GOAL REPLACEMENT

The unfold/fold system becomes more powerful when combined with goal replacement rules.

3.1 General Principle

Let S be a program and $\exists_x B_1 \& .. \& B_n$ be an existentially quantified conjunction of goals without free variables. (By x we represent a vector of variables.) We say the formula is provable in S and write $S \vdash \exists_x B_1 \& .. \& B_n$ if there is some ground instantiation θ of x such that every $B_i\theta$ $(1 \leq i \leq n)$ is provable in S.

Now let C be a clause in S of the form

$$A \leftarrow A_1, .., A_k, B_1, .., B_m$$

and C' be a clause (not in S) of the form

$$A \leftarrow A_1, .., A_k, B_1', .., B_n'$$

Let $x[y]$ be variables occurring in $B_1, .., B_m$ $[B_1', .., B_n']$ and not

in A, $A_1,..,A_k$ and $B_1',..,B_n'$ [B_1, $..,B_m$].

Suppose for every ground instantiation θ of A, $A_1,..,A_k$ it holds that

$$S-\{C\} \mid\!- \exists x(B_1\&..\&B_m)\theta$$
$$\text{iff } S-\{C\} \mid\!- \exists y(B_1'\&..\&B_n')\theta.$$

Then we can transform S into $S' = (S-\{C\}) \cup \{C'\}$.

It is rather obvious that the transformation itself preserves the least model. But when we use this rule within the unfold/fold system, we must be careful so that the second invariant I2 of the transformation process is preserved. Consider the following transformation sequence.

P_0: q(s(X)) ← q(X) (1)
q(0) (2)
r(s(X)) ← r(X) (3)
r(0) (4)
Define.
p1(X,Y) ← q(X),r(Y) (5)
p2(X,Y) ← q(X),r(y) (6)
Unfold q in (5).
p1(0,y) ← r(Y) (7)
p1(s(X),Y) ← q(X),r(Y) (8)
Replace r(Y) by r(s(Y)).
p1(s(X),Y) ← q(X),r(s(y)) (9)
Unfold r in (6).
p2(X,0) ← q(X) (10)
p2(X,s(Y)) ← q(X),r(Y) (11)
Replace q(X) by q(s(X)).
p2(X,s(Y)) ← q(s(X)),r(Y) (12)
Fold (9).
p1(s(X),Y) ← p2(X,s(Y)) (13)
Fold (12).
p2(X,s(Y)) ← p1(s(X),Y) (14)

Though each step of goal replacement is valid by itself, the resulting program contains infinite recursion and is not equivalent to the original one. This is because the goal replacement steps destroyed the invariant I2.

The general condition to preserve the invariant I2 is that for every ground instantiation θ of A, $A_1,..,A_k$,

$$r(\exists x(B_1\&..\&B_m)\theta)$$
$$\geq r(\exists y(B_1'\&..\&B_n')\theta) \quad (*)$$

holds, where by $r(\exists zB_1\&..\&B_n)$ we represent the minimum of $r(B_1\sigma)+..r(B_n\sigma)$ for every ground instantiation σ of z.

Under this condition, a rank-consistent proof in S can be converted into a rank-consistent proof in S'. There are many special cases where this condition unconditionally holds.

3.2 Special Cases

goal deletion

Let C be a clause of the form $A \leftarrow B_1,..,B_n$. If for every ground instantiation θ of the clause, $S-\{C\} \mid\!- (B_1\&..\&B_{n-1})\theta$ implies $S-\{C\} \mid\!- B_n\theta$, then B_n can be deleted.

Considering this as the replacement of $B_1,..,B_n$ by $B_1,..,B_{n-1}$, the condition (*) is obviously satisfied.

goal merging

We can merge identical goals in a body into one goal. The condition (*) is also satisfied.

function merging

Suppose there are two goals $p(t_1,..,t_{n-1},x)$ and $p(t_1,..,t_{n-1},y)$ in the body of the clause. Assume further that a ground goal $p(s_1,..,s_n)$ in $M(S-\{C\})$ is unique up to $s_1,..,s_{n-1}$. (The relation

denoted by p is actually a function.) Then we can merge the two goals applying the substitution $\{y/x\}$ to the rest of the clause. The condition (*) is satisfied.

goal addition

This is the inverse of goal deletion. The following example shows the utility of this seemingly pessimising transformation.

Example (sorting by permutation and order check)

```
P0: perm([],[])
    perm([A|X],Y) ← perm(X,Z),
                    ins(A,Z,Y)
    ins(A,X,[A|X])
    ins(A,[B|X],[B|Y]) ← ins(A,X,Y)
    ord([])
    ord([A])
    ord([A,B|X]) ← A ≤ B,ord([B|X])
Define .
    sort(X,Y) ← perm(X,Y),ord(Y)
Unfold perm .
    sort([],Y) ← ord([])
    sort([A|X],Y) ←
       perm(X,Z),ins(A,Z,Y),ord(Y)
```

Add ord(Z) in the body because for any ground terms t_1,t_2 and t_3, $P_0 \mid\!- \mathrm{ins}(t_1,t_2,t_3)\&\mathrm{ord}(t_3)$ implies $P_0 \mid\!- \mathrm{ord}(t_2)$.

```
    sort([A|X],Y) ← perm(X,Z),
          ord(Z),ins(A,Z,Y),ord(Y)
```

Fold the first two goals.

```
    sort([A|X],Y) ←
       sort(X,Y),ins(A,Z,Y),ord(Y)
```

Thus this technique is a vital step from the O(n!) sorting program to an $O(n^3)$ insertion sort program. To obtain an $O(n^2)$ program, however, we need the idea of context (Wegbreit 76), which is beyond the scope of this paper.

Though goal insertion clearly violates condition (*), the above transformation sequence does preserve equivalence. A technique to get around the difficulty will be presented in section 3.3.

laws of primitives

There are various laws for primitive predicates, such as associativity of the predicate 'append' defined by

```
Pap: append([],X,X)
     append([A|X],Y,[A,Z])
       ← append(X,Y,Z).
```

We can prove by induction that

$$P_{ap} \mid\!- \exists_X \mathrm{append}(t_1,t_2,X)\ \&\ \mathrm{append}(X,t_3,t_4)$$

iff

$$P_{ap} \mid\!- \exists_Y \mathrm{append}(t_1,Y,t_4)\ \&\ \mathrm{append}(t_2,t_3,Y)$$

for any ground terms $t_1,..,t_4$. So we can apply the associativity of append in any program incorporating P_{ap}.

The condition (*) holds if we use the associativity in one direction (the left hand side of iff to the right hand side), but does not hold in the other direction.

3.3 Weakening the Condition

We have seen that in many cases the goal replacement rule can be used with the unfold/fold transformation unconditionally. But we have also seen interesting cases where the condition (*) does not hold. For such cases we can weaken the condition (*) into the following, (though at the cost of additional bookkeeping of folding conditions.)

For every such θ as in (*), there is a partition of $\{B_1',..,B_n'\}$ such that for each part

$\{B_1'',..,B_j''\}$ of the partition, $r(\exists x(B_1\&..\&B_m)\theta) \geq r(\exists y(B_1''\&..\&B_j'')\theta)$ holds.

In the sorting example, we replaced the goals {ins(A,Z,Y), ord(Y)} by {ord(Z),ins(A,Z,Y), ord(Y)}. This is now justified because $r(\text{ord}(t_2)) < r(\text{ins}(t_1,t_2,t_3)) + r(\text{ord}(t_3))$ for any ground terms t_1,t_2 and t_3 for which the goals are provable. But we have to put labels on the introduced goals as

$$\text{sort}([A|X],Y) \text{ - perm}(X,Z), \underset{1.1}{\underline{\text{ord}(Z)}}, \underset{1.2}{\underline{\text{ins}(A,Z,Y)}}, \underset{1.2}{\underline{\text{ord}(Y)}}$$

Inheriting these labels through transformation and prohibiting folding of goals of label 1.1 and of label 1.2 together, we can make the induction in the proof of lemma 2 valid.

To prove the correctness of this technique, the condition 1 in the definition of rank-consistency should be changed:

1'. $r(A) \geq r(A_{i_1})+..+r(A_{i_m})$ for any subset $\{A_{i_1},..,A_{i_m}\}$ of $\{A_1,..,A_n\}$ such that no two goals in the subset have imcompatible labels, with equality holding only when C is marked 'foldable'.

The detail of the modified proof is omitted.

Note that the folding of the first two goals in the above clause does not violate the label constraint, so that the whole example of sorting is justified. The reverse direction of the associativity of 'append' can also be handled in this manner.

4 CLAUSE ADDITION/DELETION

clause addition

Let C be a clause not in S. If for every ground instance $A \leftarrow A_1,..,A_n$ of C, $S \mid\!- A_1\&..\&A_n$ implies $S \mid\!- A$, we can add C to S.

clause deletion

Let C be a clause in S. If for every ground instance $A \leftarrow A_1,..,A_n$ of C, $S-\{C\} \mid\!- A_1\&..\&A_n$ implies $S-\{C\} \mid\!- A$, we can delete C from S.

The correctness of these transformations themselves is again obvious. When combined with the unfold/fold transformation, clause addition causes no problem. Clause deletion can in general destroy the invariant I2 of the transformation process. As in the case of goal replacement, there are important special cases.

Let C and C' be clauses in S of the forms $A \leftarrow A_1,..,A_n$ and $B \leftarrow B_1,..,B_m$ such that A is an instance $B\sigma$ of B. Let $x[y]$ be the sequence of variables in $A_1,..,A_n$ $[B_1,..,B_m]$ but not in $A[B]$. If for every ground instantiation θ of A, $S-\{C\} \mid\!- \exists x(A_1\&..\&A_n)\theta$ implies $S-\{C\} \mid\!- \exists y(B_1\&..\&B_m)\sigma\theta$, then C can be deleted. In this special case of goal deletion, the condition

$$r(\exists x(A_1\&..\&A_n)\theta) \geq r(\exists x(B_1\&..\&B_m)\sigma\theta) \text{ for every } \theta$$

guarantees the preservation of the invariant I2. In particular, if $\{B_1\sigma,..,B_m\sigma\} \subset \{A_1,..,A_n\}$,

which means syntactic subsumption, the condition is trivially satisfied.

Finally, it should be remarked that clause addition/deletion, unlike goal replacement, are often used apart from the unfold/fold system. In such cases we need not worry about the invariant.

5 CONCLUDING REMARKS

We have proved the correctness of the basic unfold/fold system and then examined the interaction of the augmenting transformation rules with the correctness property. We have stated a sufficient condition for their application to be safe. To ensure the equivalence of the result of some transformation sequence with the initial program, we need only to check the condition for each application of those rules. As we have seen, in many useful special cases this involves only a simple syntactic checking. In other cases, proving the condition can be a difficult task. However, we can still claim the advantage over the usual separate equivalence proof approach because we have the choice of either keeping the conditions through transformation sequence or proving separately the equivalence of the result with the original program.

Though one might expect that the unfold/fold system preserves stronger properties like completion or finite failure(Clark 1978) (Apt and Van Emden 1982), this is not the case for these properties. There are easy counter examples.

The practical power of the system depends on the heuristics we employ: we have a large search space generated by the choice of applicable transformation rules. We are currently investigating this strategic aspect with some experimental implementations.

ACKNOWLEDGEMENTS

We would like to thank Yoshihiko Futamura for initiating our interest in this area, Hozumi Tanaka, Toshio Yokoi and Kouichi Furukawa for encouragement, Koukichi Futatsugi and other members of ETL for helpful discussions, Rodney Harries for correcting the English of the first manuscript, and Kazuko Hata for typing the final manuscript.

REFERENCES

Apt, K.R. and Van Emden, M.H. Contributions to the theory of logic programming. Journal of the ACM 29, No. 3, 1982.

Burstall, R.M. and Darlington, J. A transformation system for developing recursive programs. Journal of the ACM 24, No. 1, 1977.

Clark, K.L. and Sickel, S. Predicate logic: a calculas for deriving programs. Procs. IJCAI-77, Boston, 1977.

Clark, K.L. Negation as Failure, in H. Gallaire and J. Minker(eds), Logic and Databases, Plenum Press, 1978.

Clark, K.L. Predicate logic as a computational formalism. Imperial College research monograph 79/59 TOC, December 1979.

Hogger, C.J. Derivation of logic programs. Journal of the ACM 23, No. 4, 1976.

Kowalski, R.A. Predicate logic as a programming language. IFIP 74, North Holland Publishing Co., 1974.

Tamaki, H. and Sato, T. A transformation system for logic programs which preserves equivalence. ICOT TR-018, July 1983.

Van Emden, M.H. and Kowalski, R.A. The semantics of predicate logic as a programming languages. Journal of the ACM 23, No. 4, 1976.

Wegbreit, B. Goal-directed program transformation. 3rd POPL symposium, Tucson, January 1976.