PROCEEDINGS OF THE INTERNATIONAL CONFERENCE
ON FIFTH GENERATION COMPUTER SYSTEMS 1992,
edited by ICOT. © ICOT, 1992

Logic Program Synthesis from First Order Logic Specifications

Tadashi KAWAMURA
Institute for New Generation Computer Technology
1-4-28 Mita, Minato-ku, Tokyo 108, Japan
tkawamur@icot.or.jp

Abstract
In this paper, a logic program synthesis method from first order logic specifications is described. The specifications are described by Horn clauses extended by universally quantified implicational formulae. Those formulae are transformed into definite clause programs by meaning-preserving unfold/fold transformation. We show some classes of first order formulae which can be successfully transformed into definite clauses automatically by unfold/fold transformation.

1 Introduction

Logic program synthesis based on unfold/fold transformation [1] is a standard method and has been investigated by many researchers [2, 3, 5, 6, 11, 12, 19]. As for the correctness of unfold/fold rules in logic programming, Tamaki and Sato proposed meaning-preserving unfold/fold rules for definite clause programs [20]. Then, Kanamori and Horiuchi proposed unfold/fold rules for a class of first order formulae [7]. Recently, Sato proposed unfold/fold rules for full first order formulae [18].

In the studies of program synthesis, unfold/fold rules are used to eliminate quantifiers by folding to obtain definite clause programs from first order formulae. However, in most of those studies, unfold/fold rules were applied nondeterministically and general methods to derive definite clauses were not known. Recently, Dayantis [3] showed a deterministic method to derive logic programs from a class of first order formulae. Sato and Tamaki [19] also showed a deterministic method by incorporating the concept of continuation.

This paper shows another characterization of classes of first order formulae from which definite clause programs can be derived automatically. Those formulae are described by Horn clauses extended by universally quantified implicational formulae. As for transformation rules, Kanamori and Horiuchi's unfold/fold rules are adopted. A synthesis procedure based on unfold/fold rules is given, and with some syntactic restrictions, those formulae are successfully transformed into equivalent definite clause programs. This study is also an extension of those by Pettorossi and Proietti [14, 15, 16] on logic program transformations.

The rest of this paper is organized as follows. Section 2 describes unfold/fold rules and formalizes the synthesis process. Section 3 describes a program synthesis procedure and proves that definite clause programs can be successfully derived from some classes of first order formulae using this procedure. Section 4 discusses the relations to other works and Section 5 gives a conclusion.

In the following, familiarity with the basic terminologies of logic programming is assumed[13]. As syntactical variables, X, Y, Z, U, V are used for variables, A, B, H for atoms and F, G for formulae, possibly with primes and subscripts. In addition, θ is used for a substitution, $F\theta$ for the formula obtained from formula F by applying substitution θ, $\overline{X}$ for a vector of variables and $F_G[G']$ for replacement of an occurrence of subformula G of formula F with formula G'.

2 Unfold/Fold Transformation for Logic Program Synthesis

In this section, preliminary notions of our logic program synthesis are shown.

2.1 Preliminaries

Preliminary notions are described first.

A formula is called an *implicational goal* when it is of the form $F_1 \rightarrow F_2$, where F_1 and F_2 are conjunctions of atoms.

Definition 2.1 Definite Formula
Formula C is called a *definite formula* when C is of the form

$$A \leftarrow G_1 \wedge G_2 \wedge \cdots \wedge G_n (n \geq 0),$$

where G_i is a (possibly universally quantified) conjunction of implicational goals for $i = 1, 2, \ldots, n$. A is called the *head of* C, $G_1 \wedge G_2 \wedge \ldots \wedge G_n$ is called the *body of* C and each G_i is called a *goal in the body of* C.

Note that the notion of a definite formula is a restricted form of that in [7].

A set of definite formulae is called a *definite formula program*, while a set of definite clauses is called a *definite clause program*. We may simply say *programs* instead of definite formula (or clause) programs when it is obvious to which we are referring.

Definition 2.2 Definition Formula

Let P be a definite formula program. A definite formula D is called a *definition formula for P* when all the predicates appearing in D's body are defined by definite clauses in P and the predicate of D's head does not appear in P. The predicate of D's head is called a *new predicate*, while those defined by definite clauses in P are *old predicates*. A set of formulae $\mathcal{D}$ is called a *definition formula set for P* when every element D of $\mathcal{D}$ is a definition formula for P and the predicate of D's head appears only once in $\mathcal{D}$.

Atoms with new predicates are called *new atoms*, while those with old predicates are called *old atoms*.

2.2 Unfold/Fold Transformation

In this subsection, unfold/fold transformation rules are shown following [7]. Below, we assume that the logical constant *true* implicitly appears in the body of every unit clause. Further, we assume that a goal is always deleted from the body of a definite formula when it is the logical constant *true*, and a definite formula is always deleted when some goal in its body is the logical constant *false*.

Further, we introduce the reduction of implicational goals with logical constant *true* and *false*, such as $\neg true \Rightarrow false, true \wedge F \Rightarrow F$, and so on. (See [7] for details.) Let G be an implicational goal. The reduced form of G, denoted by $G\downarrow$, is the normal form in the above reduction system.

Variables not quantified in formula F are called *global variables of F*. Atoms appearing positively (negatively) in formula F are called *positive (negative)* atoms of F.

Definition 2.3 Positive Unfolding

Let P_i be a program, C be a definite formula in P_i, G be a goal in the body of C and A be a positive old atom of G containing no universally quantified variable. Then, let G_0 be $G_A[false]\downarrow$ and C'_0 be the definite formula obtained from C by replacing G with G_0. Further, let $C_1, C_2, \ldots, C_k$ be all the definite clauses in P_i whose heads are unifiable with A, say by mgu's $\theta_1, \theta_2, \ldots, \theta_k$. Let G_j be the reduced form of $G\theta_j$ after replacing $A\theta_j$ in $G\theta_j$ with the body of $C_j\theta_j$, and C'_j be the definite formula obtained from $C\theta_j$ by replacing $G\theta_j$ in the body with G_j. (New variables introduced from C_j are global variables of G_j.) Then, $P_{i+1} = (P_i - \{C\}) \cup \{C'_0, C'_1, C'_2, \ldots, C'_k\}$. $C'_0, C'_1, C'_2, \ldots, C'_k$ are called the *results of positive unfolding C at A* (or G).

Example 2.1 Let P be a definite clause program as follows :

C_1 : list([]).
C_2 : list([X|L]) $\leftarrow$ list(L).
C_3 : 0 < suc(Y).
C_4 : suc(X) < suc(Y) $\leftarrow$ X < Y.
C_5 : member(U,[U|L]).
C_6 : member(U,[V|L]) $\leftarrow$ member(U,L).

Let C_7 be a definition formula for P as follows :

C_7 : less-than-all(X,L) $\leftarrow$
list(L) $\wedge$ $\forall$ Y(member(Y,L) $\rightarrow$ X<Y).

Suppose that $P_0 = P \cup \{C_7\}$. Then, by unfolding C_7 at list(L), program $P_1 = P \cup \{C_8, C_9\}$ is obtained, where

C_8 : less-than-all(X,[]) $\leftarrow$ $\forall$ Y(member(Y,[]) $\rightarrow$ X<Y).
C_9 : less-than-all(X,[Z|L]) $\leftarrow$
list(L) $\wedge$ $\forall$ Y(member(Y,[Z|L]) $\rightarrow$ X<Y).

Before showing the negative unfolding rule, we introduce the notion of *terminating atoms*. Intuitively, atom A is terminating when every derivation path of A is finite. See [7] for the precise definition.

Definition 2.4 Negative Unfolding

Let P_i be a program, C be a definite formula in P_i, G be a goal in the body of C and A be a negative old atom of G such that every atom obtained from A by instantiating all global variables in A to ground is terminating. Let $C_1, C_2, \ldots, C_k$ be all the definite clauses in P_i whose heads are unifiable with A, say by mgu's $\theta_1, \theta_2, \ldots, \theta_k$, where θ_j instantiates no global variable in G. Let G_0 be $G_A[false]\downarrow$ and G_j be the reduced form of $G\theta_j$ after replacing $A\theta_j$ in $G\theta_j$ with the body of $C_j\theta_j$. (New variables introduced from C_i are universally quantified variables in G_i.) Let C' be the definite formula obtained from C by replacing G in the body of C with $G_0 \wedge G_1 \wedge \ldots \wedge G_k$. Then, $P_{i+1} = (P_i - \{C\}) \cup \{C'\}$. C' is called the *results of negative unfolding C at A* (or G).

Example 2.2 Let P and P_1 be programs in Example 2.1. By unfolding C_8 at member(X,[]), $P_2 = P \cup \{C_9, C_{10}\}$ is obtained, where

C_{10} : less-than-all(X,[]) $\leftarrow$ $\forall$ Y ($false$ $\rightarrow$ X < Y) $\downarrow$.

that is,

C_{10} : less-than-all(X,[]).

Further, by unfolding C_9 at member(X,[Z|L]), $P_3 = P \cup \{C_{10}, C_{11}\}$ is obtained, where

C_{11} : less-than-all(X,[Z|L]) $\leftarrow$ list(L) $\wedge$
$\forall$ Y($false$ $\rightarrow$ X<Y)$\downarrow$ $\wedge$
$\forall$ Y($true$ $\rightarrow$ X<Z)$\downarrow$ $\wedge$
$\forall$ Y (member(Y,L) $\rightarrow$ X<Y)$\downarrow$.

that is,

C_{11} : less-than-all(X,[Z|L]) $\leftarrow$ list(L) $\wedge$
X < Z $\wedge$ $\forall$ Y (member(Y,L) $\rightarrow$ X < Y).

Definition 2.5 Folding

Let P_i be a definite formula program, C be a definite formula in P_i of the form $A \leftarrow K \wedge L$ and D be a definite formula of the form $B \leftarrow K'$, where K,K' and L are conjunctions of goals. Suppose that there exists a substitution θ such that $K'\theta = K$ holds. Let C' be a clause of the form $A \leftarrow B\theta, L$. Then $P_{i+1} = (P_i - \{C\}) \cup \{C'\}$.

Note that when applying folding, some conditions have to be satisfied to preserve the meanings of programs. See [7] for details.

Example 2.3 Let P and P_3 be programs in Example 2.2. By folding C_{11} by C_7, $P_4 = P \cup \{C_{10}, C_{12}\}$ is obtained, where

C_{12} : less-than-all(X,[Y|L]) $\leftarrow$
X $<$ Y $\wedge$ less-than-all(X,L)

2.3 Program Synthesis by Unfold/Fold Transformation

In this subsection, our program synthesis problem is formalized. Firstly, several notions are defined to formalize the program synthesis processes.

Definition 2.6 Descendant and Ancestor Formula

Let P be a definite formula program, C be a definite formula in P and P' be a definite formula program obtained from P by successively applying positive or negative unfolding to P. A definite formula C' in P' is called a *descendant formula* of C when

(a) C' is identical to C, or

(b) C' is the result of positive or negative unfolding of a descendant formula of C.

Conversely, C is called an *ancestor formula* of C'.

Example 2.4 In Examples 2.1 – 2.3, definite formulae $C_7, C_8, \ldots, C_{11}$ are descendant formulae of C_7.

Definition 2.7 U-selection Rule

A rule that determines what transformation should be applied to a definite formula program is called a *selection rule*. Let P be a definite formula program and C be a definite formula in P. A selection rule R is called a *U-selection rule for P rooted on C* when R always selects positive or negative unfolding applied to a descendant formula of C. C is called the *root formula for R* (or *of the transformation*.) A definite formula program obtained from P by successively applying transformation rules according to R is called a definite formula program *obtained from P via R*.

Definition 2.8 Closed Program

Let P be a definite clause program, C be a definition formula for P, $\mathcal{D}$ be a definition formula set for P and R be a U-selection rule for $P \cup \{C\}$ rooted on C. Let P' be a definite formula program obtained from $P \cup \{C\}$ via R. P' is said to be *closed with respect to* triple $< P, C, \mathcal{D} >$ when every descendant formula C' of C in P' satisfies one of the following:

(a) C' is a definite clause.

(b) There exists a goal G consisting of positive atoms only in the body of C' such that an old atom in G is not unifiable with the head of any definite clause in P'.

(c) By successively folding C' by clauses in $\{C\} \cup \mathcal{D}$, a definite clause can be obtained.

$P \cup \{C\}$ is said to be *closed with respect to* $\mathcal{D}$ when there exists a closed program with respect to $< P, C, \mathcal{D} >$ and for every definition formula D in $\mathcal{D}$ there exists a closed program with respect to $< P, D, \mathcal{D} \cup \{C\} >$.

Example 2.5 Let P and P_3 be programs in Example 2.2. Then, P_3 is closed w.r.t. $< P, C_7, \emptyset >$. Further, $P \cup \{C_7\}$ is closed w.r.t. $\emptyset$.

The above framework is an extension of the one shown in [8], and also a modification of the one Pettorossi and Proietti proposed [14, 15, 16] in their studies of program transformation.

Now, our problem can be formalized as follows: for given definite clause program P and definition formula C for P, find a finite definition formula set $\mathcal{D}$ for P such that $P \cup \{C\}$ is closed with respect to $\mathcal{D}$.

3 Some Classes of First Order Formulae from Which Logic Programs Can Be Derived

In this section, we specify some classes of first order formulae from which definite clause programs can be derived by unfold/fold transformation.

3.1 A Program Synthesis Procedure

In this subsection, we show a naive program synthesis procedure. In the following, we borrow some notions about programs in [15, 16]. We consider definite formula (clause) programs with predicate =, which have no explicit definition in the programs. Predicate = is called a *base predicate*, while other predicates are called *defined predicates*. Atoms with base predicates are called *base atoms*, while those with defined predicates are called *defined atoms*. Transformation rules can be applied to defined atoms only.

A formula containing base atoms can be reduced by unifying arguments of =. When a universally quantified variable and a global variable are unified, the global variable is substituted for the universal one. The above reduction is called the *reduction with respect to* =. We assume that no formulae are reduced w.r.t. = unless this is explicitly mentioned.

Further, we assume that the following operations are always applied implicitly to the results of positive or negative unfolding. Goals G is said to be *connected* when at most one universally quantified implicational goal G'

appears in G and each atom in G' has common universally quantified variables with at least one another atom in G'. Let C be a definite formula such that all the goals in its body are connected. Let C' be one of the results of positive or negative unfolding C at some goal. By logical deduction, definite formulae $C'_1, C'_2, \ldots, C'_m (m \geq 1)$ are obtained from C' such that all the goals in the body of C'_i are connected. (Note that some goal G in the body of C' is of the form $F_1 \rightarrow F_2$ or $F_1 \vee F_2$ and no universally quantified variables appear in both F_1 and F_2, C' can be split into two formulae by replacing G in C' with $\neg F_1$ (or F_1) and F_2.)

Before showing our program synthesis procedure, a notion is defined.

Definition 3.1 Sound Unfolding
Suppose that positive or negative unfolding is applied to a definite formula at atom A. Then, the application of unfolding is said to be *sound* when no two distinct universally quantified variables in A are unified when reducing the result of unfolding with respect to $=$.

Some syntactic restrictions on programs ensure the soundness of all possible applications of unfolding. In fact, the restriction shown in [3] ensures the soundness. However, in the following, we assume that every application of unfolding is sound, without giving any syntactic restriction, for simplicity.

Now, we show our program synthesis procedure, which is similar to partial evaluation procedures(cf.[9, 10]). First, a procedure to synthesize new predicates is shown.

Procedure 3.1 *Synthesis of New Predicates*
Suppose that definite formula program P and definite formula C in P of the form $A \leftarrow G_1, G_2, \ldots, G_n$ are given. Let G'_i be the reduced formula obtained from G_i by removing all base atoms and by replacing all universally quantified variables appearing in every base atom with distinct fresh global variables if global variables are substituted for them when reducing G_i w.r.t. $=$. Let D_i be of the form $H_i \leftarrow G'_i$ for $i = 1, 2, \ldots, n$, where H_i is an atom whose predicate does not appear in P or H_j for $i \neq j$ and whose arguments are all global variables of C appearing in G'_i. Then, $D_1, D_2, \ldots, D_n$ are returned.

Note that in Procedure 3.1, C can be folded by $D_1, D_2, \ldots, D_n$ after reducing it w.r.t. $=$ when C is the result of sound unfolding, and the result of the folding is a definite clause.

Example 3.1 Let P be a program as follows.

C_1 : all-less-than(L,M) $\leftarrow$ list(L) $\wedge$ list(M) $\wedge$
    $\forall$ U,V (member(U,L) $\wedge$ member(V,M) $\rightarrow$ U $<$ V).
C_2 : member(U,[V|X]) $\leftarrow$ U = V.
C_3 : member(U,[V|X]) $\leftarrow$ member(U,X).

The definition of '$<$' is given in Example 2.1. Suppose that C's body consists of only one goal. By applying positive unfolding and negative unfolding to C successively, the following formulae are obtained. (The reduction w.r.t. $=$ is done when no universally quantified variable appears as an argument of $=$.)

C_4 : all-less-than([],M) $\leftarrow$ list(M).
C_5 : all-less-than([X|L],M) $\leftarrow$ (list(L) $\wedge$ list(M)) $\wedge$
    (list(L) $\wedge$ list(M) $\wedge$
    $\forall$ U,V (U = X $\wedge$ member(V,M) $\rightarrow$ U $<$ V)) $\wedge$
    (list(L) $\wedge$ list(M) $\wedge$
    $\forall$ U,V (member(U,L)$\wedge$member(V,M) $\rightarrow$ U $<$ V)).

Then, by Procedure 3.1, the following new predicates are defined from C_5.

D_1 : new1(X,L,M) $\leftarrow$ list(L) $\wedge$ list(M) $\wedge$
    $\forall$ V (member(V,M) $\rightarrow$ X $<$ V).
D_2 : new2(L,M) $\leftarrow$ list(L) $\wedge$ list(M) $\wedge$
    $\forall$ U,V (member(U,L) $\wedge$ member(V,M) $\rightarrow$ U $<$ V).

Next, the whole procedure for program synthesis is shown.

Procedure 3.2 *A Program Synthesis Procedure*
Suppose that definite clause program P and definition formula C for P are given. Let $\mathcal{D}$ be the set $\{C\}$.

(a) If there exist no unmarked formulae in $\mathcal{D}$, then return P and stop.
(b) Select an unmarked definition formula D from $\mathcal{D}$. Mark D 'selected.' Let P' be the set $\{D\}$.
(c) If there exist no formulae in P' which do not satisfy conditions (a) and (b) in Definition 2.8, then $P := P \cup P'$ and go to (a).
(d) Select a definite formula C' from P'. Apply positive or negative unfolding to C'. Let $C_1, \ldots, C_n$ be the results. Remove C' from P'.
(e) Apply Procedure 3.1 to $C_1, \ldots, C_n$. Let $D_1, \ldots, D_m$ be the outputs. Add D_i to $\mathcal{D}$ if it is not a definite clause and there exists no formula in $\mathcal{D}$ which is identical to D_i except for the predicate of the head. Fold $C_1, \ldots, C_n$ by the formulae in $\mathcal{D}$ and add the results to P'.
(f) Go to (c).

Example 3.2 Consider the program in Example 3.1 again. We see that D_2 is identical to C except for the predicate of the head. C_5 can be folded by D_1 and C after reduction w.r.t. $=$. The result is as follows.

C_6 : all-less-than([X|L],M) $\leftarrow$ list(L) $\wedge$ list(M) $\wedge$
    new1(X,L,M) $\wedge$ all-less-than(L,M).

Similar operations are applied to D_1, and finally, the following clauses are obtained.

D_3 : new1(X,L,[]) $\leftarrow$ list(L).
D_4 : new1(X,L,[Y|M]) $\leftarrow$ X $<$ Y $\wedge$ new1(X,L,M).

Note that Procedure 3.2 does not necessarily derive a definite clause program from a definite formula program. For example, when the following program is given as input, Procedure 3.2 does not halt.

C_1 : p(X,Y) $\leftarrow$ p(X,Z) $\wedge$ p(Z,Y)
C_2 : h(X,Y) $\leftarrow$ $\forall$ Z (p(X,Z) $\rightarrow$ p(Y,Z))

3.2 Classes of First Order Formulae

In this section, we show some classes of definite formula programs which can be transformed into equivalent definite clause programs by Procedure 3.2.

Throughout this subsection, we assume that unfolding is always applicable to every definite formula at an atom when there exist definite clauses whose heads are unifiable with the atom. Note that the above assumption does not always hold. This problem will be discussed in 3.3.

After giving a notion, we show a theorem which is an extension of the results shown in [15]. A *simple expression* is either a term or an atom.

Definition 3.2 Depth of Symbol in Simple Expression

Let X be a variable or a constant and E be a simple expression in which X appears. The depth of X in E, denoted by depth(X,E), is defined as follows.

(a) depth(X,X) = 1.

(b) depth(X,E) = $\max\{\text{depth}(X,t_i)|X$ appears in t_i for $i = 1,\ldots,n\} + 1$, if E is either $f(t_1,\ldots,t_n)$ or $p(t_1,\ldots,t_n)$, for any function symbol f or any predicate symbol p.

The deepest variable or constant in E is denoted by maxdepth(E).

Theorem 3.1 Let P be a definite clause program. Suppose that for any definition formula C for P, there exists a U-selection rule R for $P \cup \{C\}$ rooted on C such that R is defined for all descendant clauses of C in which at least one defined atom appears. Suppose also that there exist two positive integers H and W such that every descendant clause C' of C in every program P' obtained from $P \cup \{C\}$ via R satisfies the following two conditions.

(a) The depth of every term appearing in every goal in the body of C' is less than H.

(b) Let $G_1, G_2, \ldots, G_n$ be connected goals in the body of C'. Then, the number of atoms appearing in G_i is less than W, for $i = 1, 2, \ldots, n$.

Then, there exists a finite definition formula set $\mathcal{D}$ for P such that $P \cup \{C\}$ is closed with respect to $\mathcal{D}$.

Proof. From hypothesis (a), only a finite number of distinct atoms (modulo renaming of variables) can appear in the goals of all the descendant formulae of C. Then, apply Procedure 3.2 to P and C. Note that every goal in the body of every descendant formula of C is connected. Then, for every goal of every descendant formula of C, the number of atoms appearing in the goal is less than W, from hypothesis (b). Hence, only a finite number of distinct goals can appear in all the descendant formulae of C. Thus, we can obtain a finite definition formula set $\mathcal{D}_0$ for P such that there exists a closed program P' w.r.t. $< P, C, \mathcal{D}_0 >$.

The above discussion holds for all the definition formulae in $\mathcal{D}_0$, since those formulae are constructed from bodies of the descendant formulae of C. Evidently, only a finite number of distinct definition formulae can be defined. Thus, there exists a finite definition formula set $\mathcal{D}$ for P such that $P \cup \{C\}$ is closed w.r.t. $\mathcal{D}$. □

Theorem 3.1 shows that Procedure 3.2 can derive a definite clause program when (a) a term of infinite depth can not appear, or (b) an infinite number of atoms can not appear in a connected goal during a transformation process. In the following, we show some syntactic restrictions on programs which satisfy the above conditions.

Proietti and Pettorossi showed some classes of definite clause programs which satisfy the conditions in Theorem 3.1 in their studies of program transformation [15]. We show that some extensions of their results are applicable to our problem.

The following definitions are according to [15]. The set of variables occurring in simple expression E is denoted by var(E).

Definition 3.3 Linear Term Formula and Program

A simple expression or a formula is said to be *linear* when no variable appears in it more than once. A definite formula (clause) is called a *linear term formula (clause)* when every atom appearing in it is linear. A definite formula (clause) program is called a *linear term program* when it consists of linear term formulae (clauses) only.

A linear term formula (clause) is called a *strongly linear term formula (clause)* when its body is linear. A definite formula (clause) program is called a *strongly linear term program* when it consists of strongly linear term formulae (clauses) only.

Note that the following definite clause is not a linear term clause.

member(X,[X|L]).

However, it is easy to obtain an equivalent linear term clause as follows :

member(X,[Y|L])← X=Y.

Definition 3.4 A Relation $\leq$ between Linear Simple Expressions

Let E_1 and E_2 be linear simple expressions. When depth(X,E_1)≤depth(X,E_2) holds for every variable X in var(E_1)∩var(E_2), we write $E_1 \leq E_2$. (Both $E_1 \leq E_2$ and $E_2 \leq E_1$ hold when var(E_1)∩var(E_2)= $\emptyset$.)

Definition 3.5 Non-Ascending Formula and Program

Let C be a linear term formula and H be the head of C. C is said to be *non-ascending* when $A \leq H$ holds for every defined atom A appearing in the body of C. A linear term program is said to be *non-ascending* when it consists of non-ascending formulae only.

A definite formula (clause) is said to be *strongly non-ascending* when it is a strongly linear term formula (clause) and non-ascending. A definite formula (clause) program is said to be *strongly non-ascending* when it consists of strongly non-ascending formulae (clauses) only.

Definition 3.6 Synchronized Descent Rule
Let P be a linear term program, R be a U-selection rule for P and C be any descendant formula of the root formula for R. Let $A_1, A_2, \ldots, A_n$ be all the atoms appearing in the body of C. Then, R is called a *synchronized descent rule* when

(a) R selects the application of positive or negative unfolding to C at A_i if and only if $A_j \leq A_i$ holds for $j = 1, \ldots, n$, and
(b) R is not defined for C, otherwise.

Note that synchronized descent rules are not necessarily defined uniquely for given programs and definition formulae.

The following theorem is an extension of the one shown in [15, 16].

Lemma 3.2 Let P be a non-ascending definite clause program, C be a linear term definition formula for P, and R be a synchronized descent rule rooted on C. Let P' be a program obtained from $P \cup \{C\}$ via R. For each defined atom A appearing in the body of every descendant clause of C in P', the following holds :

$$\text{maxdepth}(A) \leq \max\{\text{maxdepth}(B) \mid B \text{ is a defined atom in } P \cup \{C\}\}$$

Proof. By induction on the number of applications of unfolding. □

Now we show some classes of definite formula programs which satisfy the hypotheses of Theorem 3.1. In the following, for simplicity, we deal with definition formulae with only one universally quantified implicational goal in the body. The results are easily extended to the definite formulae with a conjunction of universally quantified implicational goals.

The following results are also extensions of those shown in [15].

Theorem 3.3 Let P be a strongly non-ascending definite clause program and C be a linear term definition formula for P of the form $H \leftarrow A_1 \wedge \forall\overline{X}(A_2 \rightarrow A_3)$, such that the following hold.

(a) For every clause D in P of the form $H_D \leftarrow B_1 \wedge \ldots \wedge B_n \wedge B'_1 \wedge \ldots \wedge B'_m$, where $B_1, \ldots, B_n$ are defined atoms and $B'_1, \ldots, B'_m$ are base atoms, the following hold.
  (a-1) Let t_H be any argument of H_D. For every argument t_i of B_i, if t_H contains a common variable with t_i, then t_i is a subterm of t_H.
  (a-2) For every argument t_i of B_i, if t_i is a subterm of an argument t_H of H_D, then no other argument of B_i is a subterm of t_H.
(b) There exist two arguments t_i and s_i of some A_i ($t_i \neq s_i, i = 1, 2$ or 3) such that the following hold.
  (b-1) There exists an argument t_j of A_j ($i \neq j$) such that
   · vars(A_i)∩vars(A_j)=vars(t_i)∩vars(t_j), and
   · either t_i is a subterm of t_j, t_j is a subterm of t_i or vars(t_i)∩vars(t_j)=$\emptyset$.
  (b-2) There exists an argument s_k of A_k ($k \neq i, j$) such that the same relations as above hold for s_i and s_k.
  (b-3) A_j contains no common variable with A_k.

Then, there exists a definition formula set $\mathcal{D}$ for P such that $P \cup \{C\}$ is closed with respect to $\mathcal{D}$.

Proof. Note that there exists an atom A in the body of C s.t. an argument of A is a maximal term in the body of C w.r.t. subterm ordering relation. Let C' be any result of unfolding C at A and G be any connected goal in the body of C' of the form $F_1 \wedge \forall\overline{X}(F_2 \rightarrow F_3)$, where F_i is a conjunction of atoms. Then, from the hypothesis, it can be shown that a similar property to hypothesis (b) holds for G. Note that the number of implicational goals dose not increase by applying positive unfolding and no global variables are instantiated by applying negative unfolding. Then, again there exists an atom in the body of C' s.t. one of its arguments is a maximal term in the body of C' w.r.t. subterm ordering relation. By induction on the number of applications of unfolding, a synchronized descent rule can be defined for every descendant formula of C. Then, from Lemma 3.2, the depth of every term appearing in every descendant clause of C is bounded.

Note that the number of different subterms of a term is bounded. Then, from the hypothesis, the number of atoms appearing in every connected goal in the body of every descendant formula of C is bounded. Thus, P and C satisfy the hypotheses of Theorem 3.1. Hence, there exists a definition formula set $\mathcal{D}$ for P such that $P \cup \{C\}$ is closed with respect to $\mathcal{D}$. □

Note that Theorem 3.3 holds for any nondeterministic choice of synchronized descent rules in the above proof. Note also that any program can be modified to satisfy hypothesis (a) of Theorem 3.3 by introducing atoms with = in the body.

Corollary 3.4 Let P be a strongly non-ascending definite clause program and P' be a definite clause program such that no predicate appears in both P and P'. Let C be a linear term definition formula for $P \cup P'$ of the form $H \leftarrow A_1 \wedge \forall\overline{X}(A_2 \rightarrow A_3)$, where the predicates of A_1 and A_2 are defined in P and that of A_3 is defined in P'. Suppose that the following hold.

(a) Hypothesis (a) of Theorem 3.3 holds for every clause D in P.
(b) There exist arguments t_1 of A_1 and t_2 of A_2 such that the following hold.
  (b-1) vars(A_1)∩vars(A_2)=vars(t_1)∩vars(t_2).

(b-2) Either t_1 is a subterm of t_2, t_2 is a subterm of t_1 or vars(t_1)$\cap$vars(t_2)=$\emptyset$.

(c) No variable in A_3 is instantiated by applying positive or negative unfolding to C successively.

Then, there exists a definition formula set $\mathcal{D}$ for $P \cup P'$ such that $P \cup P' \cup \{C\}$ is closed with respect to $\mathcal{D}$.

Proof. Suppose that unfolding is never applied at A_3. A synchronized descent rule can be defined by neglecting A_3. Since variables in A_3 are never instantiated, no other atoms are derived from A_3. Thus, the corollary holds. □

In Corollary 3.4, no restrictions are required on the definition of A_3. This result corresponds to that in [3]. Note that any program can be modified to satisfy hypothesis (c) of Corollary 3.4 by introducing atoms with = in the body.

Example 3.3 The program and the definition formula in Example 2.1 satisfy the hypotheses of Theorem 3.3 and Corollary 3.4, if clause C_5 is replaced with the equivalent clause :

C_5' : member(U,[V|L]) $\leftarrow$ U=V.

In fact, a definite clause program can be obtained, as shown in subsection 2.2.

Next, we show an extension of the results shown in Theorem 3.3. Let P be a non-ascending definite clause program and C be a definition formula for P of the form $H \leftarrow A \wedge \forall \overline{X}(F_1 \rightarrow F_2)$, where A is an atom, and F_1 and F_2 are conjunctions of atoms. Let D_i be the definition clause for P of the form $H_i \leftarrow F_i$ for $i = 1, 2$. If D_i can be transformed into a set of definite clauses which satisfies the hypotheses of Theorem 3.3, by replacing F_i with H_i, we can show that $P \cup \{C\}$ can be transformed into an equivalent definite clause program.

The above problem is related to the foldability problem in [16]. The foldability problem is described informally as follows. Let P be a definite clause program and C be a definition clause for P. Then, find program P' obtained from $P \cup \{C\}$ which satisfies the following : for every descendant clause C' of C in P', there exists an ancestor clause D of C' such that C''s body is an instance of D's.

Proietti and Pettorossi showed some classes of definite clause programs such that the foldability problem can be solved [16]. We show that their results are also available to our problem.

A definite clause program P is said to be *linear recursive* when at most one defined atom appears in the body of each clause in P. Note that a linear recursive and linear term program (clause) is a strongly linear term program (clause).

Lemma 3.5 Let P be a linear recursive non-ascending program and C be a non-ascending definition clause for P of the form $H \leftarrow A_1 \wedge A_2 \wedge B_1 \wedge \ldots \wedge B_n$, where A_1 and A_2 are defined atoms and $B_1, \ldots, B_n$ are base atoms. Suppose that the following hold.

(a) For every clause D in P of the form $H_D \leftarrow A_D \wedge B_1' \wedge \ldots \wedge B_n'$, where A_D is the only defined atom in the body of D, the following hold.

(a-1) Let t_H be any argument of H_D. For every argument t_A of A_D, if t_H contains a common variable with t_A, then t_A is a subterm of t_H.

(a-2) For every argument t_A of A_D, if t_A is a subterm of an argument t_H of H_D, then no other argument of A_D is a subterm of t_H.

(b) There exist arguments t_1 of A_1 and t_2 of A_2 such that the following hold.

(b-1) vars(A_1)$\cap$vars(A_2)=vars(t_1)$\cap$vars(t_2).

(b-2) Either t_1 is a subterm of t_2, t_2 is a subterm of t_1 or vars(t_1)$\cap$vars(t_2)=$\emptyset$.

Then, from $P \cup \{C\}$, we can obtain a linear recursive non-ascending program which define the predicate of H by unfold/fold transformation.

Proof. As shown in [16], we can get a solution of the foldability problem for P and C. Then, obviously, a linear recursive program is obtained. □

Example 3.4 Let P be a linear recursive non-ascending program as follows.

C_1 : subseq([],L).
C_2 : subseq([X|L],[Y|M]) $\leftarrow$ X = Y $\wedge$ subseq(L,M).
C_3 : subseq([X|L],[Y|M]) $\leftarrow$ subseq([X|L],M).

Let C be a non-ascending definition clause for P as follows.

C : csub(X,Y,Z) $\leftarrow$ subseq(X,Y), subseq(X,Z).

Then, $P \cup \{C\}$ can be transformed into a linear recursive non-ascending program as follows.

csub([],Y,Z).
csub([A|X],[B|Y],Z) $\leftarrow$ A = B $\wedge$ cs(A,X,Y.Z).
csub([A|X],[B|Y],Z) $\leftarrow$ csub([A|X],Y,Z).
cs(A,X,Y,[B|Z]) $\leftarrow$ A = B $\wedge$ csub(X,Y,Z).
cs(A,X,Y,[B|Z]) $\leftarrow$ cs(A,X,Y,Z).

Though Proietti and Pettrossi showed one more class [16], we will not discuss this here.

Now, we get the following theorem.

Theorem 3.6 Let P be a linear recursive non-ascending program and C be a linear term definition formula for P of the form $H \leftarrow A_1 \wedge \forall \overline{X}(A_2 \wedge B_2 \rightarrow A_3 \wedge B_3)$, such that the following hold.

(a) Hypothesis (a) of Lemma 3.5 holds for P.

(b) Let S_1 be the set of all the arguments of A_1, and S_i be the set of all the arguments of A_i and B_i for $i = 2, 3$. Then, there exist two terms t_j and s_j in some S_j ($t_j \neq s_j, j = 1, 2$ or 3) such that the following hold.

(b-1) there exists a term t_k in S_k ($j \neq k$) such that

· vars(S_j)$\cap$vars(S_k)=vars(t_j)$\cap$vars(t_k), and

- either t_j is a subterm of t_k, t_k is a subterm of t_j or $\mathrm{vars}(t_j) \cap \mathrm{vars}(t_k) = \emptyset$.

(b-2) There exists a term s_l of S_l ($l \neq j, k$) such that the same relations as above hold for s_j and s_l.

(b-3) S_k contains no common variable with S_l.

Then, there exists a definition formula set $\mathcal{D}$ for P such that $P \cup \{C\}$ is closed with respect to $\mathcal{D}$.

Proof. Obvious from Theorem 3.3 and Lemma 3.5. □

Note that it is easy to extend the result of Theorem 3.6 to allow the conjunction of an arbitrary number of atoms to appear in the body of the definition formula. Note also that it is possible to extend the result to allow arbitrary definition of A_3 and B_3, in a similar way to Corollary 3.4.

3.3 Further Consideration about Syntactic Restrictions

As described in 3.2, the application of unfolding may be prohibited in Kanamori and Horiuchi's framework. In this subsection, we discuss some methods to avoid prohibition, though we do not necessarily give the precise syntactic restriction. (Due to space limitations, we do not refer to the terminating property, though several sufficient conditions are known to guarantee it.)

(1) Universally Quantified Variables Appearing in Positive Atoms

Positive unfolding can not be applied to definite formulae at positive atoms with universally quantified variables. Thus, we have the following two problems.

(a) Synchronized descent rules can not be defined when universally quantified variables are instantiated by negative unfolding.

(b) We can not unfold formulae of the form $\forall \overline{X} A$ when A is an atom and some variables in $\overline{X}$ appear in A.

To avoid case (a), the following restriction is sufficient. When applying negative unfolding, no universally quantified variable is instantiated. Though the restriction seems to be strong, most of significant examples of program synthesis can be dealt with under the restriction.

Case (b) corresponds to the compilation failure in Sato and Tamaki's first order compiler [19]. They restricted their language as follows. For every implicational goal $F_1 \rightarrow F_2$ appearing in a formula, $\mathrm{uvar}(F_1) \supseteq \mathrm{uvar}(F_2)$ holds, where $\mathrm{uvar}(F_i)$ means the set of universally quantified variables appearing in F_i.

The above condition is available for our problem. Note that the application of positive unfolding does not affect the condition. When applying negative unfolding at atom A in universally quantified implicational goal G, the following restrictions are also required. All the universally quantified variables appearing in A also appear in some negative defined atom in each result of negative unfolding G, or they are unified with terms consisting of constants and global variables by reduction w.r.t. $=$.

We believe that techniques such as mode analysis are available to guarantee that every applicable negative unfolding satisfies the above conditions.

(2) Global Variables Appearing in Negative Atoms

Negative unfolding should be applied without instantiating global variables. In some cases, this restriction may be critical. However, we can deal with most of those cases by adding positive atoms to the formula such that the global variables can be instantiated by applying positive unfolding at those atoms. Atoms with predicates which specify data types (cf. list) are available. For example, with the definitions of 'member' and '<' in Example 2.1, negative unfolding can not be applied to the definite formula below.

$$\text{less-than-all}(X,L) \leftarrow \forall Y(\text{member}(Y,L) \rightarrow X<Y).$$

However, we can apply negative unfolding to the formula below, after positive unfolding list(L).

$$\text{less-than-all}(X,L) \leftarrow \text{list}(L) \wedge \forall Y(\text{member}(Y,L) \rightarrow X<Y).$$

(3) Sato's Unfold/Fold Transformation

Recently, Sato proposed unfold/fold transformation rules for full first order programs [18]. Their unfolding operation does not require conditions like Kanamori and Horiuchi's. On the other hand, more complex conditions are required when applying folding. Thus, when we adopt Sato's rules in place of Kanamori and Horiuchi's, we need not consider the restrictions discussed in (1) and (2) above, while some other difficulties are introduced to satisfy the folding conditions.

4 Discussion

The work described here is an extension of Pettorossi and Proietti's work on program transformation [14, 15, 16]. They formalized the successful unfold/fold transformation in three ways, and showed that the problem of whether a given program can be transformed successfully or not is unsolvable. They also showed some classes of definite clause programs which can be transformed successfully. Our results owe much to their work, though currently we do not know whether our problem is decidable.

Proietti and Pettorossi also showed that any definite clause program can be transformed successfully by performing suitable generalization of the atoms to be folded [15, 16]. However, the generalization technique is not available for our problem. Folding by a definition formula obtained by generalizing atoms with universally quantified variables may not satisfy the conditions for folding [7], since universally quantified variables can not appear in the head of the formula.

Proietti and Pettorossi also showed a transformation procedure called loop absorption [15, 16]. In this procedure, they found clause C and its descendant clause C' such that C''s body is an instance of C's (or a subset of C''s body is identical to C's body). Then, a new definition clause whose body is identical to that of C is constructed. They also showed a procedure to eliminate unnecessary variables [17]. We can modify our naive procedure described in 3.1 by incorporating the loop absorption and the elimination of unnecessary variables. Programs obtained by the modified procedure are expected to be more efficient and have less code than those obtained by the naive procedure.

There have been several studies on logic program synthesis from universally quantified implicational formulae [3, 4, 19]. Our work is closely related to that of Dayantis [3]. There, program synthesis was also considered from formulae of the form $H \leftarrow \forall \overline{X}(A \rightarrow B)$. They showed that a class of those formulae can be transformed into definite clauses by deductive derivation. They also discussed the generality of the class using several examples. Their deductive method is analogous to unfold/fold transformation and the derivation processes almost correspond to those by our procedure when our procedure does not apply positive unfolding. They also mechanized their derivation processes. Our notion of the soundness of the application of unfolding is ensured by part of their syntactic restrictions on the arguments of formulae, though we have not discussed how this is ensured. However, the classes we have shown are still wider than those they showed after we incorporate those restrictions.

Sato and Tamaki showed a deterministic algorithm to transform logic programs with universally quantified implicational formulae into definite clause programs [19]. In their method, unfold/fold transformation is applied to universal continuation forms. Their method can be applied to a wider class of first order formulas than ours, while the results of the compilation are not necessarily efficient and the code sizes of those results increase generally.

5 Conclusion

A logic program synthesis method from some classes of first order logic specifications have been shown. The method is based on unfold/fold transformation. Some classes of first order formulae which can be transformed into definite clause programs by unfold/fold transformation have been shown.

Acknowledgments

I would like to thank Tadashi Kanamori and anonymous referees for helpful comments. I would also like to thank Koichi Furukawa and Ryuzo Hasegawa for their advice, and Kazuhiro Fuchi for giving me the opportunity to do this research.

References

[1] Burstall, R.M. and J.Darlington, "A Transformation System for Developing Recursive Programs", J.ACM, Vol.24, No.1, pp.44–67, 1977.

[2] Clark, K.L. and S. Sickel, "Predicate Logic: A Calculus for Deriving Programs", Proc. of 5th International Joint Conference on Artificial Intelligence, pp.419–420, 1977.

[3] Dayantis, G., "Logic Program Derivation for a Class of First Order Logic Relations", Proc. of 10th International Joint Conference on Artificial Intelligence, pp.9–14, Italy, 1987.

[4] Fribourg, L., "Extracting Logic Programs from Proofs that Use Extended Prolog Execution and Induction", Proc. of 7th International Conference on Logic Programming, pp.685–699, Jerusalem, 1990.

[5] Hansson, A. and Tarnlund, S.A., "A Natural Programming Calculus", Proc. of 6th International Joint Conference on Artificial Intelligence, pp.348–355, 1979.

[6] Hogger, C.J., "Derivation of Logic Programs", J.ACM, Vol.28, pp.372–392, 1981.

[7] Kanamori, T. and K. Horiuchi, "Construction of Logic Programs Based on Generalized Unfold/Fold Rules", Proc. of 4th International Conference on Logic Programming, pp.744–768, Melbourne, 1987.

[8] Kawamura, T., "Derivation of Efficient Logic Programs by Synthesizing New Predicates", Proc. of 1991 International Logic Programming Symposium, pp.611–625, San Diego, 1991.

[9] Komorowski, J., "Partial Evaluation As A Means for Inferencing Data Structures in An Applicative Language : A Theory And Implementation in The Case of Prolog", Proc. of the ACM Symposium on Principles of Programming Languages, pp.255–267, 1982.

[10] Komorowski, J., "Towards a Programming Methodology Founded on Partial Deduction", Proc. of the European Conference on Artificial Intelligence, pp.404–409,1990.

[11] Lau, K.K. and S. D. Prestwich, "Top-down Synthesis of Recursive Logic Procedures from First-order Logic Specifications", Proc. of 7th International Conference on Logic Programming, pp.667–684, Jerusalem, 1990.

[12] Lau, K.K. and S. D. Prestwich, "Synthesis of a Family of Recursive Sorting Procedures", Proc. of 1991 International Logic Programming Symposium, pp.641–658, San Diego, 1991.

[13] Lloyd, J. W., "Foundations of Logic Programming", Springer-Verlag, 2nd Edition, Berlin, Heidelberg, New York, 1987.

[14] Pettorossi, A. and M. Proietti, "Decidability Results and Characterization of Strategies for the Development of Logic Programs", Proc. of 6th International Conference on Logic Programming, pp.539–553, Lisboa, 1989.

[15] Proietti, M. and A. Pettorossi, "Construction of Efficient Logic Programs by Loop Absorption and Generalization", Proc. of the Second Workshop on Meta-programming in Logic, pp.57–81, Leuven, 1990.

[16] Proietti, M. and A. Pettorossi, "Synthesis of Eureka Predicates for Developing Logic Programs", Proc. of 3rd European Symposium on Programming, Copenhagen, LNCS 432, Springer-Verlag, pp.307–325,1990.

[17] Proietti, M. and A. Pettorossi, "Unfolding - Definition - Folding, In This Order, For Avoiding Unnecessary Variables In Logic Programs", Proc. of 3rd International Symposium on Programming Language Implementation and Logic Programming, Passau, LNCS 528, Springer-Verlag, pp.347–358,1991.

[18] Sato, T., "An Equivalence Preserving First Order Unfold/fold Transformation System", Algebraic and Logic Programming, Proceedings, LNCS 463, Springer-Verlag, pp.173–188, 1990.

[19] Sato, T. and H. Tamaki, "First Order Compiler : A Deterministic Logic Program Synthesis Algorithm", J.Symbolic Computation, Vol.8, pp.605–627, 1989.

[20] Tamaki, H. and T. Sato, "Unfold/Fold Transformation of Logic Programs", Proc. of 2nd International Logic Programming Conference, pp.127–138, Uppsala, 1984.