A Transformation System for Definite Programs Based on Termination Analysis

J. Cook and J.P. Gallagher*

Department of Computer Science, University of Bristol, Bristol BS8 1TR, U.K.

Abstract. We present a goal replacement rule whose main applicability condition is based on termination properties of the resulting transformed program. The goal replacement rule together with a multi-step unfolding rule forms a powerful and elegant transformation system for definite programs. It also sheds new light on the relationship between folding and goal replacement, and between different folding rules. Our explicit termination condition contrasts with other transformation systems in the literature, which contain conditions on folding and goal replacement, often rather complex, in order to avoid "introducing a loop" into a program. We prove that the goal replacement rule preserves the success set of a definite program. We define an extended version of goal replacement that also preserves the finite failure set. A powerful folding rule can be constructed as a special case of goal replacement, allowing folding with recursive rules, with no distinction between *old* and *new* predicates. A proof that Seki's transformation system preserves recurrence, an important termination property, is outlined.

1 Introduction

In this paper we define a goal replacement rule (for definite programs) based on termination analysis. Under the conditions of the rule, a replacement can only be made if the resulting program terminates for all ground calls to the head of the transformed clause. We show that application of this rule to a program will preserve its success set. The finite failure set can also be preserved by adding a similar termination condition on the program being transformed. We also give a definition of a more general unfolding rule which is based on a partial evaluation rule discussed in [LS91] and [GS91].

On reviewing the literature regarding folding and goal replacement, it was apparent that the underlying issue (when considering the correctness of such rules) was the termination of the resulting program. This was particularly evident in [Sek89] and [BCE92].

In the first of these papers, Seki, discussing the unfold/fold transformations of Tamaki and Sato [TS84], noted that a goal may not terminate in a program resulting from a Tamaki-Sato folding rule, even though it finitely failed in the original program. To combat this, Seki proposed a reformulated folding rule.

* Correspondence to jpg@compsci.bristol.ac.uk

Under the conditions of [BCE92], a goal replacement can be made provided that 'a loop is not introduced' into the program. This is clearly a reference to some termination properties of the transformed program.

It should be noted that the original unfold/fold transformations for functional programs [BD77] preserves only partial correctness. Total correctness, that is, termination, has to be established separately. [2] In logic programming, starting with the work of Tamaki and Sato [TS84], the tradition has been to devise conditions on unfolding and folding that implicitly guarantee total correctness results. Recently, conditions on goal replacement requiring 'reversibility' of goal replacement have been proposed by Proietti and Pettorossi [PP93], stressing the preservation of partial and total correctness as separate concerns. This work is in the same spirit as ours though the approach differs. Our goal replacement is not reversible, but still preserves total correctness (in the sense of the program's success set).

The link between transformation rules and termination analysis provided us with the motivation for the system we present. The main difference of approach between our work and the works just cited (and also [GS91], [Mah87], [Sek89] and others) is that in our goal replacement rule termination of the transformed program is a condition of application of the rule, and we leave open the means by which termination is checked. Although the problem of checking goal replacement conditions is thereby shifted to checking termination properties, the advantage gained is that the link between folding and goal replacement is clarified. More flexible folding rules can be invented, as we show below. Also, research on termination analysis can be brought to bear on the problem.

We note that Boulanger and Bruynooghe [BB93] also developed an approach to goal replacement that is based on generation of replacement lemmata during an abstract interpretation of the program. This approach also seems capable of performing fold-like operations that are less restrictive than usual, allowing folding with recursive rules, for instance. [3]

There are two alternative strategies for implementing our transformation rules. One possibility is to use termination analysis techniques, to check termination of a transformed program directly. Secondly, special cases of the goal replacement can be derived, in which syntactic conditions ensuring termination are checked. This suggest a reconstruction of 'fold' transformations. In general the first approach seem the more promising, since it seems very difficult to find syntactic conditions that guarantee useful termination properties (such as acceptability [AP90]).

Section 2 contains a review of logic program transformation systems. In Section 3 we introduce our transformation rules and prove the correctness results. In Section 4 termination analysis is reviewed. An example of our system is given in Section 5. We show that our replacement rule allows 'foldings' to be based on

[2] We are grateful to A. Pettorossi for this pertinent comment

[3] We have been informed that folding using recursive rules was also considered in unpublished technical reports by Tamaki and Sato (1986), and by Kanamori and Fujita (1986).

equivalences that depend on recursive definitions of predicates. This is not allowed in other unfold/fold transformation systems. An informal proof that Seki's transformation system preserves recurrence can be found in Section 6. Finally, Section 7 is a short concluding section and some thoughts on future research directions.

The terminology used throughout this paper is consistent with that of [Llo87].

2 Background

2.1 Unfold/Fold of Definite Programs

In [TS84] the notion of a transformation sequence for definite programs was introduced.

Definition 1. An initial program, P_0, is a definite program satisfying the following conditions:

- P_0 is divided into two sets of clauses, P_{new} and P_{old}. The predicates defined in P_{new} are called new predicates, those defined in P_{old} are called old predicates.
- Predicates defined in P_{new} do not appear in P_{old} nor in the body of any clause in P_{new}.

The clauses defining new predicates are considered as *new definitions*; we note that they cannot be recursive.

Definition 2. Let P_0 be an initial program, and P_{i+1} a program obtained from P_i by applying either unfolding or folding for $i \geq 0$. The sequence of programs $P_0, \ldots, P_n$ is called a transformation sequence starting from P_0.

We do not repeat the definition of the unfold and fold rules from [TS84] here, but note that folding depends on the distinction between *old* and *new* predicates in the initial program. In a folding transformation applied in P_i in a transformation sequence, the clause used for folding is one of the clauses defining a new predicate. Thus folding in [TS84], and other systems derived from it, is not defined as a transformation on a program, but rather as a transformation on a sequence of programs.

Denoting the least Herbrand model of a program P by $M[P]$, the main result of [TS84] is the following.

Proposition 3. *Let $P_0, \ldots, P_n$ be a transformation sequence starting from P_0 which uses Tamaki-Sato unfolding and folding. Then $M[P_0] = M[P_n]$.*

Although the unfolding and folding rules of [TS84] preserve the success set of a program, they do not preserve its finite failure set. Some queries that finitely failed in an original program may not terminate after the transformation has been applied. [Sek89] proposes a modified folding rule based on the notion of an inherited atom. (Although Seki's transformations are defined for normal programs, we restrict our attention to definite programs here).

The fold-unfold system using Seki's fold rule preserves the finite failure set of a program as well as its minimal model. Let $FF[P]$ denote the finite failure set of P.

Proposition 4. *Let $P_0, \ldots, P_n$ be a transformation sequence starting from P_0 which uses unfolding as in [TS84], and the folding rule in [Sek89]. Then $FF[P_0] = FF[P_n]$.*

A property of Seki's transformation system will be considered in Section 6. For the rest of the paper we refer to the success set $SS[P]$ rather than $M[P]$ for a program P, for symmetry with $FF[P]$.

Transformation rules (for normal programs) are also considered by Gardner and Shepherdson in [GS91]. The unfolding rule (restricted to definite programs) is the same as that of [TS84]. The unfolding rule is used in conjunction with a reversible folding rule, similar to that of [Mah87], in which the folding clause is taken from the current program rather than from the initial program. Folding can be reversed by unfolding using these rules, and thus the correctness of folding follows from the correctness of unfolding. This elegant framework is independent of any transformation sequence. However, as a consequence of this, the framework appears to be less useful for practical purposes than the transformation system proposed in [TS84], in which the equivalences on which the foldings are based come from a set of clauses in the fixed initial program, and so are not altered during a transformation sequence.

By proving that their folding rule is the inverse of unfolding, Gardner and Shepherdson are able to show that their system preserves procedural equivalence based on SLDNF-resolution. For definite programs, therefore, both the minimal model and the finite failure set are preserved.

2.2 Goal Replacement

The goal replacement operation was defined in [TS84] to increase the power of the unfold/fold transformation system. In general, the rule is the replacement of a clause in P of the form

$$C : A \leftarrow A_1, \ldots, A_k, A_{k+1}, \ldots, A_n$$

by the clause

$$C' : A \leftarrow B_1, \ldots, B_m, A_{k+1}, \ldots, A_n$$

to form the program $P' = P - \{C\} \cup \{C'\}$, where $A_1, \ldots, A_k$ and $B_1, \ldots, B_m$ are in some sense equivalent in P and P'. (We consider replacing only some left conjunct of the body, without loss of generality).

Tamaki and Sato claimed that their goal replacement rule preserved the least Herbrand model of P. However, [GS91] gives a counter-example to this claim and defined a goal replacement rule with stronger applicability conditions. Gardner and Shepherdson show that the least Herbrand model is preserved by their goal replacement transformation.

One motivation of our work is to provide a goal replacement rule so that folding *is* a special case, and more flexible folding rules can be derived.

This was also one of the aims of Bossi *et al.* [BCE92]. A replacement operation (for normal programs) is defined in [BCE92]. Applicability conditions are given in order to ensure the correctness of the operation with respect to Fitting's semantics for normal programs. These semantics are based on Kleene's three-valued logic, the truth values being true, false, and undefined.

The conditions for goal replacement there are based on the rather complex notions of 'semantic delay' and 'dependency degree'. It is explained in [BCE92] that the applicability conditions are to ensure that there is 'no room for introducing a loop'. When the program P is transformed to P', under the applicability conditions:

- the dependency degree is an indication of how big a loop would be.
- the semantic delay is an indication of the space in which the loop would be introduced.

The transformation rules in [BCE92], though complex, provided us with the inspiration to search for more general conditions for goal replacement by investigating the problem of termination, since avoiding the 'introduction of a loop' seems to be the main intuition behind the work outlined above.

3 A Transformation System Based on Termination Analysis

In this section, we apply some of the ideas from the survey in Section 2 to develop a new transformation system. The main idea is that a goal replacement operation should depend on the notion of termination. Furthermore, we claim that this approach sheds light on other transformation systems.

We present two transformation rules for definite programs and prove that if the program P' results from the transformation of P under these rules then P and P' have the same least Herbrand model. The rules are, the unfold rule (Definition 5) based on partial evaluation, and the goal replacement rule (Definition 8) based on termination analysis. We do not consider a rule to introduce new definitions since we may consider all such definitions to have been introduced in the initial program of a transformation sequence.

3.1 Unfolding

Definition 5. Let P be a definite program and C a clause in P of the form

$$A \leftarrow Q$$

where Q is a conjunction of atoms (possibly empty).

Build an (incomplete) SLD-tree for $P \cup \{\leftarrow Q\}$ via some computation rule.

Choose goals $\leftarrow G_1, \ldots, \leftarrow G_r$ such that every non-failing branch of the SLD-tree contains precisely one of them. Let θ_i be the computed answer for the derivation from $\leftarrow Q$ to $\leftarrow G_i$, $(i = 1, \ldots, r)$.

The result of unfolding C on B in P is the definite program P' formed from P by replacing C by the set of clauses $\{C_1, \ldots, C_r\}$, where $C_i = A\theta_i \leftarrow G_i$.

Theorem 6. *Let the definite program P' be obtained by applying an unfold transformation to a clause in P. Then $SS[P] = SS[P']$ and $FF[P] = FF[P']$.*

The proof is drawn from a special case of partial evaluation which is discussed in [LS91] and [GS91]. Note that the extension of the usual 'one-step' unfold rule to the 'multi-step' rule above is more than just convenience; there are unfoldings using Definition 5 that cannot be reproduced by a series of one-step unfoldings, since a clause being used for unfolding may itself be modified as the result of a single unfolding step.

3.2 Goal Replacement

Let $Q_1 \equiv_V Q_2$ in P denote a notion of computational equivalence of formulas, in the following sense.

Definition 7. Let Q_1 and Q_2 be conjunctions of atoms and V a set of variables. We write $Q_1 \equiv_V Q_2$ in P if

– $\exists\sigma$ such that $P \cup \{\leftarrow Q_1\}$ has an SLD-refutation with computed answer σ, where $\sigma|_V = \theta \quad \Leftrightarrow$
$\exists\rho$ such that $P \cup \{\leftarrow Q_2\}$ has an SLD-refutation with computed answer ρ, where $\rho|_V = \theta$

In other words the computed answers for goals $\leftarrow Q_1$ and $\leftarrow Q_2$ agree on the variables in V.

Definition 8. goal replacement

Let P be a definite program and C a (non-unit) clause in P of the form

$$A \leftarrow Q_1, Q$$

where A is an atom and Q_1,Q conjunctions of atoms. Let R be a computation rule, and let vars(A, Q) denote the set of variables occurring in either A or Q. Form P' from P by replacing C by the clause

$$A \leftarrow Q_2, Q$$

where both of the following conditions are satisfied:

– $Q_1 \equiv_{\text{vars}(A,Q)} Q_2$ in P.
– For all ground instances A' of A, the SLD-tree for $P' \cup \{\leftarrow A'\}$ via R is finite.

Note that the definition refers to termination (finiteness of an SLD-tree) via a particular computation rule. One can use whatever computation rule one chooses in order to check termination. In practice one is probably interested in 'standard' computation rules such as the Prolog leftmost selection rule. The order of atoms in clause bodies and the computation rule are interdependent. For simplicity in the definition we have assumed that goal replacement takes place on the left part of the body, but with a fixed computation rule such as Prolog's this restriction needs to be relaxed. The definition is also asymmetric, in that we check the termination property in P' only. This reflects the directed nature of the transformation sequence, and the concern not to introduce loops during transformations. The second condition could also be weakened by checking termination of derivations starting with the transformed clause only (rather than all derivations of atomic goals unifying with its head).

The main correctness result for the goal replacement transformation is the preservation of the success set (Theorem 10). We shall see later that a symmetric version of the goal replacement rule, with an extended notion of goal equivalence, preserves the finite failure set as well (Theorem 15).

We start by proving a lemma stating that goals which succeed in P do not finitely fail in P'. Note that this lemma does not require the termination condition in the goal replacement rule, but simply uses the properties of goal equivalence. In a sense this lemma establishes a "partial correctness" result, and the termination condition will be used to establish total correctness.

Lemma 9. *Let the definite program P' be obtained by applying a goal replacement transformation to a clause in P. Then for all definite goals $\leftarrow B$,*

> $P' \cup \{\leftarrow B\}$ *has a finitely failed SLD-tree implies that* $P \cup \{\leftarrow B\}$ *has no SLD-refutation.*

The proof of this lemma is in the appendix.

Theorem 10. *Let the definite program P' be obtained by applying a goal replacement transformation to a clause in P. Then $SS[P] = SS[P']$.*

Proof. The proof is in two stages. We show that:

1. $SS[P'] \subseteq SS[P]$
2. $SS[P] \subseteq SS[P']$

Stage 1: let $A' \in SS[P']$; show by induction on the length of the shortest SLD-refutation of $P' \cup \{\leftarrow A'\}$, that there is an SLD-refutation of $P \cup \{\leftarrow A'\}$. Let $SS^k[P']$ be the set of atoms A such that $P' \cup \{\leftarrow A\}$ has an SLD-refutation of length at most k.

For the base case, suppose $A' \in SS^1[P']$. This means that A' is a ground instance of the head of a unit clause D in P'. Goal replacement does not affect unit clauses, so D is also in P. Therefore $P \cup \{\leftarrow A'\}$ has an SLD-refutation of depth 1.

For the induction step, assume that $P \cup \{\leftarrow A'\}$ has an SLD-refutation for all $A \in SS^k[P']$. Suppose that $A' \in SS^{k+1}[P']$ and let T' be an SLD-refutation of $P' \cup \{\leftarrow A'\}$ of length $k+1$. Suppose the first clause (variant) used in T' is $C_r : H \leftarrow S$.

If C_r is not the result of the goal replacement, then C_r is also in P. Let $mgu(A', H) = \rho$; then $P' \cup \{\leftarrow S\rho\}$ has an SLD-refutation of length k with computed answer σ, say. Let $S\rho\sigma\gamma$ be any ground instance of $S\rho\sigma$. Then for each ground atom $\bar{A}$ in $S\rho\sigma\gamma$, $P' \cup \{\leftarrow \bar{A}\}$ has an SLD-refutation of length at most k and so by the induction hypothesis, $P \cup \{\leftarrow \bar{A}\}$ has an SLD-refutation. It follows that $P \cup \{\leftarrow S\rho\sigma\gamma\}$ has an SLD-refutation and hence so has $P \cup \{\leftarrow S\rho\}$. Therefore $P \cup \{\leftarrow A'\}$ has an SLD-refutation using C_r on the first step.

If C_r is the clause produced by goal replacement we have

$$C_r : A \leftarrow Q_2, Q \text{ in } P'$$

which replaced

$$C : A \leftarrow Q_1, Q \text{ in } P$$

where A' is unifiable with A with mgu θ. Then $P' \cup \{\leftarrow (Q_2, Q)\theta\}$ has an SLD-refutation and hence $P' \cup \{\leftarrow Q_2\theta\}$ has an SLD-refutation with computed answer α, say, and $P' \cup \{\leftarrow Q\theta\alpha\}$ has a refutation with computed answer σ, say.

Using the inductive hypothesis applied to the atoms in any ground instance of $Q_2\theta\alpha$ we can show that $P \cup \{\leftarrow Q_2\theta\}$ also has an SLD-refutation with computed answer α. Since $Q_1 \equiv_{\text{vars}(A,Q)} Q_2$ in P, and since θ acts on variables in A, $P \cup \{\leftarrow Q_1\theta\}$ has an SLD-refutation with computed answer $\hat{\alpha}$, where α agrees with $\hat{\alpha}$ in variables of $A\theta$ and $Q\theta$. Also by the inductive hypothesis applied to any ground instance of $Q\theta\alpha\sigma$ it follows that $P \cup \{\leftarrow Q\theta\alpha\}$ has a refutation with computed answer σ. Hence $P \cup \{\leftarrow Q\theta\hat{\alpha}\}$ has a refutation with computed answer σ. Putting the goals together it follows that $P \cup \{\leftarrow (Q_1, Q)\theta\}$ has an SLD-refutation. Therefore, $P \cup \{\leftarrow A'\}$ has an SLD-refutation using clause C on the first step.

Stage 2: let $A' \in SS[P]$ and show that $A \in SS[P']$. Proof is by induction on the length of an SLD-refutation of $P \cup \{\leftarrow A'\}$.

The base case is similar to the base case in Stage 1. For the inductive step, consider an SLD-refutation of $P \cup \{\leftarrow A'\}$ of length $k+1$, using C_r on the first step. If C_r is not the clause used in goal replacement the argument is identical to the corresponding step in Stage 1.

If C_r is the clause used in goal replacement, then A' is a ground instance of the head of C_r, and thus also of the head of the clause that replaces C_r. By the termination condition on goal replacement, $P' \cup \{\leftarrow A'\}$ has a finite SLD-tree, so it is either a finitely-failed tree, or contains a refutation. Suppose $P' \cup \{\leftarrow A'\}$ has a finitely-failed SLD-tree. Then by Lemma 9 $P \cup \{\leftarrow A'\}$ has no SLD-refutation. But this contradicts the inductive hypothesis. Hence $P' \cup \{\leftarrow A'\}$ has an SLD-refutation. □

We conjecture that the set of computed answers is also preserved, since goal equivalence is based on equivalence of computed answers.

There is an obvious corollary to Theorems 6 and 10, which is the basis for reconstructing folding as a special case of goal replacement.

Corollary 11. *Let $P_0, \ldots, P_k$ be a transformation sequence using unfold and goal replacement rules. Let $Q_1 \equiv_V Q_2$ in P_i for some $0 \leq i < k$. Then $Q_1 \equiv_V Q_2$ in P_j, $0 \leq j \leq k$.*

In particular, if there is a clause $A \leftarrow Q$ in some P_i, where A has no common instance with any other clause head in P_i, then $A \equiv_{\mathrm{vars}(A)} Q$ holds in P_i. Therefore it holds in subsequent programs in the sequence and the clause can be used for goal replacement in subsequent programs. In other words, if a clause of form $H \leftarrow Q\theta, R$ occurs in some P_j, where $j > i$. it may be replaced by $H \leftarrow A\theta, R$ in P_{j+1} provided that

1. Variables in $\mathrm{vars}(Q)/\mathrm{vars}(A)$ are mapped by θ into distinct variables not occurring in H, $A\theta$ or R. (This is sufficient to establish the first goal replacement condition that $Q\theta \equiv_{\mathrm{vars}(H,R)} A\theta$).
2. $P_{j+1} \cup \{\leftarrow H'\}$ has a finite SLD-tree (via some computation rule) for all ground instances H' of H.

This avoids the restrictions on folding present in [TS84], [Sek89], [GS91] and so on, that folding clauses define "new" predicates and are non-recursive. In Section 5 there is an example of a "fold" transformation using a recursive definition, which would not be possible with other fold rule based on [TS84]. Our goal replacement condition also avoids the rather complex syntactic conditions that are typical of folding rules.

In the literature there are two distinct flavours of folding. In [TS84] and related systems, folding is defined on a transformation sequence, and the clause used for folding might not be in the program to which folding is applied. By contrast, [GS91] and [Mah87] give a more elegant folding rule in which the folding clause comes from the program being transformed. These two folding rules are quite different in power. Both of them can be viewed as special cases of our goal replacement rule.

3.3 Preserving the Finite Failure Set

The goal replacement rule does not preserve the finite failure set. Preserving finite failure is desirable in some contexts, and so we strengthen the definition of goal equivalence in Definition 7, by insisting that the equivalent goals have the same failing behaviour as well as computed answers.

Definition 12. Let Q_1 and Q_2 be conjunctions of atoms and V a set of variables. We write $Q_1 \equiv_V Q_2$ in P if

– $\exists\sigma$ such that $P \cup \{\leftarrow Q_1\}$ has an SLD-refutation with computed answer σ, where $\sigma|_V = \theta \quad \Leftrightarrow$
  $\exists\rho$ such that $P \cup \{\leftarrow Q_2\}$ has an SLD-refutation with computed answer ρ, where $\rho|_V = \theta$, and

– for all substitutions θ acting only on variables in V,
$P \cup \{\leftarrow Q_1\theta\}$ has a finitely failed SLD-tree $\quad\Leftrightarrow$
$P \cup \{\leftarrow Q_2\theta\}$ has a finitely failed SLD-tree.

Assume now that the goal replacement transformation (Definition 8) refers to equivalence in the sense of Definition 12. We can then prove a stronger version of Lemma 9.

Lemma 13. *Let the definite program P' be obtained by applying a goal replacement transformation to a clause in P (using Definition 12 and Definition 8). Then for all definite goals $\leftarrow B$,*

$P' \cup \{\leftarrow B\}$ has a finitely failed SLD-tree
$\Rightarrow P \cup \{\leftarrow B\}$ has a finitely failed SLD-tree.

Proof. (outline)
The proof is similar in structure to the proof of Lemma 9, and does not use the termination condition on goal replacement. The key step is the construction of a finitely-failed SLD tree for a computation of form $P \cup \{\leftarrow (Q_1, R)\theta\}$ given a finitely-failed SLD tree for $P \cup \{\leftarrow (Q_2, R)\theta\}$, where $Q_1 \equiv_V Q_2$, θ acts only on V, and R is some conjunction. □

Note that the converse of Lemma 13 does not hold. A goal may fail finitely in P but loop in P', as the following example shows.

Example 1. Let $P = \{p \leftarrow q(X),\ \ q(f(X)) \leftarrow r(X)\}$. Then $q(X) \equiv_{\{X\}} r(X)$ and we can replace $r(X)$ by $q(X)$ in the second clause, since the termination condition is satisfied. We obtain $P' = \{p \leftarrow q(X),\ \ q(f(X)) \leftarrow q(X)\}$. $P \cup \{\leftarrow p\}$ has a finitely failed SLD tree but $P' \cup \{\leftarrow p\}$ loops.

As noted above, the goal replacement rule is asymmetric in that termination is checked only in the program after transformation. Clearly by virtue of Lemma 13 a symmetric version preserves the finite failure set as well as the success set.

Definition 14. (symmetric goal replacement) Let P be transformed to P' by goal replacement as defined in Definition 8 with Definition 12. If P can also be obtained from P' by goal replacement, then we say P' is obtained from P by symmetric goal replacement.

Theorem 15. *Let P' be obtained from P by symmetric goal replacement. Then $SS[P] = SS[P']$ and $FF[P] = FF[P']$.*

Proof. Using Theorem 10 and Lemma 13.□

Corollary 11 holds with respect to the symmetric goal replacement transformation and the extended definition of goal equivalence. Thus folding transformations that preserve the finite failure set can be constructed as goal replacements.

4 Termination of Definite Programs

We now turn our attention to the topic of termination analysis, which is needed in order to verify the conditions for our goal replacement rule. There has been a great deal of research effort addressing many aspects of termination analysis. This research has branched into many different directions because of the variety of definitions that exist for termination. The question of what is meant by a terminating program depends principally on such matters as the procedural semantics employed, the different modes of use of a program, and the fact that logic programs can be nondeterministic. This summary relies to a great extent on the survey in [DV92].

We may wish to consider either existential or universal termination, the former being termination because of finite failure, or because at least one solution is found (even though the program could have entered an infinite computation after finding such a solution), the latter being termination of the entire computation (all solutions having been found).

There are two different approaches in the research. The approach taken in, for example, [UV88] and [Plü92] looks to the provision of sufficient conditions for the termination of a logic program with respect to certain classes of queries that can be automatically verified. An alternative approach is taken in such papers as [VP86], [Bez89], [AP90] and [DVB92], where theoretical frameworks to solve questions about termination are offered.

Both [UV88] and [Plü92] use reasoning with linear predicate inequalities to generate termination proofs. The termination dealt with in [Plü92] is *left-termination* (i.e. termination under the left-to-right computation rule) of pure Prolog programs with respect to goals with input terms of known mode.

The concept of a level mapping is introduced by Bezem in [Bez89]. Denoting the Herbrand base of a program P by B_P:

Definition 16. A level mapping for a definite program P is a mapping $|.| : B_P \to N$.

The necessary decrease in measure for establishing termination through recursive clauses is ensured by the restriction of Bezem's work to a particular class of programs - those which are recurrent.

Definition 17. A definite program P is recurrent if there is a level mapping $|.|$ such that for each ground instance $A \leftarrow B_1, \ldots, B_n$ of a clause in P, we have $|A| > |B_i|$ for each $i = 1, \ldots, n$. A program is recurrent if it is recurrent with respect to some level mapping.

Note that, in the above definition, there is a decrease in measure between the head and body of *every* clause of the program. The main result of [Bez89] is that a logic program is terminating (with respect to ground goals and an arbitrary computation rule) if and only if it is recurrent.

Recurrency with respect to $|.|$ can be used to infer termination of non-ground goals whose ground instances have a maximum value with respect to $|.|$.

The ideas of [Bez89] are addressed with respect to termination in the left-to-right computation rule by Apt and Pedreschi in [AP90]. Many practical programs terminate under the Prolog selection rule (when given the correct class of inputs) even though they are not recurrent. As a result, in [AP90] the notion of a recurrent program is replaced by that of an *acceptable* one. Apt and Pedreschi prove that the notions of left-termination and acceptability coincide.

Some limitations of these approaches, with respect to automating the analysis of termination, are discussed by De Schreye *et al.* in [DVB92]. where extended notions of level mappings and recurrency are introduced. They introduce the notion of recurrency with respect to a set of atoms S. The key results in [DVB92] are that: P is recurrent with respect to S if and only if it is terminating with respect to S, and that P is recurrent if and only if it is recurrent with respect to B_P.

5 Example

This section contains an example of a program transformation under our rules in which the goal replacement stages cannot be made under the folding rule of [TS84], [Sek89] or [GS91]. This suggests that our system could be used to achieve more flexible folding rules, using, say, recursive definitions of new predicates. The example also illustrates the use of both recurrence and acceptability to establish applicability of goal replacements.

Example 2. Let $P^0 = \{C_1, \ldots, C_4\}$, where

$C_1 : \texttt{append}([], xs, xs).$
$C_2 : \texttt{append}([x|xs], ys, [x|zs]) \leftarrow \texttt{append}(xs, ys, zs).$
$C_3 : \texttt{leaves}(\texttt{leaf}(x), [x]).$
$C_4 : \texttt{leaves}(\texttt{tree}(x, y), zs) \leftarrow$
$\quad \texttt{leaves}(x, xs), \texttt{leaves}(y, ys), \texttt{append}(xs, ys, zs).$

Unfold C_4 on the first atom to obtain $P^1 = (P^0/\{C_4\}) \cup \{C_5, C_6\}$, where

$C_5 : \texttt{leaves}(\texttt{tree}(\texttt{leaf}(x'), y), [x'|zs]) \leftarrow \texttt{leaves}(y, zs),$
$C_6 : \texttt{leaves}(\texttt{tree}(\texttt{tree}(x', y'), y), zs) \leftarrow$
$\quad \texttt{leaves}(x', xs'), \texttt{leaves}(y', ys'), \texttt{append}(xs', ys', zs'),$
$\quad \texttt{leaves}(y, ys), \texttt{append}(zs', ys, zs).$

Goal Replacement 1
Replacement of $\texttt{append}(xs', ys', zs'), \texttt{append}(zs', ys, zs)$ by
$\texttt{append}(ys', ys, vs), \texttt{append}(xs', vs, zs)$ in C_6 to give P^2 :

$C_1 : \texttt{append}([], xs, xs).$
$C_2 : \texttt{append}([x|xs], ys, [x|zs]) \leftarrow \texttt{append}(xs, ys, zs).$
$C_3 : \texttt{leaves}(\texttt{leaf}(x), [x]).$
$C_5 : \texttt{leaves}(\texttt{tree}(\texttt{leaf}(x'), y), [x'|zs]) \leftarrow \texttt{leaves}(y, zs),$
$C_7 : \texttt{leaves}(\texttt{tree}(\texttt{tree}(x', y'), y), zs) \leftarrow$
$\quad \texttt{leaves}(x', xs'), \texttt{leaves}(y', ys'), \texttt{leaves}(y, ys),$
$\quad \texttt{append}(ys', ys, vs), \texttt{append}(xs', vs, zs).$

Goal Replacement 2
Goal replace $\texttt{leaves}(y', ys'), \texttt{leaves}(y, ys), \texttt{append}(ys', ys, vs)$ by $\texttt{leaves}(\texttt{tree}(y', y), vs)$ in C_7 to obtain P^3 :

$C_1 : \texttt{append}([], xs, xs).$
$C_2 : \texttt{append}([x|xs], ys, [x|zs]) \leftarrow \texttt{append}(xs, ys, zs).$
$C_3 : \texttt{leaves}(\texttt{leaf}(x), [x]).$
$C_5 : \texttt{leaves}(\texttt{tree}(\texttt{leaf}(x'), y), [x'|zs]) \leftarrow \texttt{leaves}(y, zs),$
$C_8 : \texttt{leaves}(\texttt{tree}(\texttt{tree}(x', y'), y), zs) \leftarrow$
$\quad \texttt{leaves}(x', xs'), \texttt{leaves}(\texttt{tree}(y', y), vs), \texttt{append}(xs', vs, zs).$

Goal Replacement 3
Replace $\texttt{leaves}(x', xs'), \texttt{leaves}(\texttt{tree}(y', y), vs), \texttt{append}(xs', vs, zs)$ by $\texttt{leaves}(\texttt{tree}(x', \texttt{tree}(y', y)), zs)$ in C_8 to obtain the final program P^4 :

$C_1 : \texttt{append}([], xs, xs).$
$C_2 : \texttt{append}([x|xs], ys, [x|zs]) \leftarrow \texttt{append}(xs, ys, zs).$
$C_3 : \texttt{leaves}(\texttt{leaf}(x), [x]).$
$C_5 : \texttt{leaves}(\texttt{tree}(\texttt{leaf}(x'), y), [x'|zs]) \leftarrow \texttt{leaves}(y, zs),$
$C_9 : \texttt{leaves}(\texttt{tree}(\texttt{tree}(x', y'), y), zs) \leftarrow \texttt{leaves}(\texttt{tree}(x', \texttt{tree}(y', y)), zs).$

We must show that the conditions of goal replacement (Definition 8) are satisfied for the three replacements steps in this transformation sequence. The recurrence of P^4 can easily be shown, which establishes the condition for the third goal replacement. For the first and second goal replacements, P^2 and P^3 are not recurrent, so we need a more refined notion of termination than recurrency. The notion of acceptability from [DVB92] is appropriate for these cases, since then we can prove termination for ground goals via a left-to-right computation rule. In [Coo92] the detailed demonstrations of the acceptability properties are given.

The applicability of the transformations does not actually require the notions of recurrence or applicability, which concern termination of all ground atoms, not just instances of the head of the transformed clause. There is clearly scope for using still more limited notions of termination (such as those in [DV92]) in cases where neither applicability nor recurrence can be shown.

6 Termination Preservation

It is interesting to examine current logic program transformation systems with regard to termination properties, since our work is based on the premise that termination properties are an essential ingredient in correct transformation systems. In this section we consider termination properties of the Seki transformation system [Sek89] (restricted to definite programs). We show that Seki's transformation system preserves recurrence. This shows that, given a recurrent initial program, a Seki transformation sequence is a special case of our transformations, since the Seki folding conditions are sufficient to establish recurrence and hence termination of the folded program.

Recently, Bossi and Etalle [BE94] have proved the stronger result that the Tamaki-Sato transformation system preserves another termination property, namely acyclicity. Although many typical terminating programs are neither recurrent nor acyclic these results are further evidence of the importance of termination properties in program transformation.

6.1 Preservation of Recurrence in Seki's System

We first recall that a transformation sequence is a sequence $P_0, \ldots, P_n$ where $P_0 = P_{new} \cup P_{old}$. The predicates in the heads of clauses in P_{new} are *new* predicates and the other predicates are *old* predicates.

The unfolding rule is standard and we do not repeat it here. The folding rule depends on the notion of an *inherited* atom. We do not give the full definition of an inherited atom here. Intuitively, an atom A in the body of a clause in P_i is inherited from P_0 if

– A was in the body of a clause in P_{new}, and
– for each transformation in the sequence up to P_i, A was neither the unfolded atom nor one of the folded literals.

Seki's folding rule can now be defined.

Definition 18. Let P_i be a program in a transformation sequence and C a clause in P_i of form $A \leftarrow Q_1\theta, Q$, where A is an atom and Q_1, Q conjunctions of atoms. Let D be a clause in P_{new} of form $B \leftarrow Q_1$. Then the result of folding C using D is

$$C' : A \leftarrow B\theta, Q$$

provided that all the following conditions hold.

– For each variable occurring in Q_1 but not in B, θ substitutes a distinct variable not appearing in C'.
– D is the only clause in P_{new} whose head is unifiable with $B\theta$.
– Either the predicate in the head of C is an old predicate, or there is no atom in $Q_1\theta$ which is inherited from P_0.

The resulting program is $P_{i+1} = (P_i/\{C\}) \cup \{C'\}$.

We first state a lemma (whose proof is in the appendix), and then the final result that recurrence is preserved by Seki's transformations.

Lemma 19. *Let $P_0, \ldots, P_n$ be a Seki transformation sequence and let P_0 be recurrent. Then for all clauses $A \leftarrow B_1, \ldots, B_m$ in P_i:*

(i) If A has an old predicate, none of the B_j is inherited from P_0.
(ii) If B_j has an old predicate and is not inherited from P_0, then for all ground instances $(A \leftarrow B_1, \ldots, B_m)\theta$ we have $|A\theta| - |B_j\theta| \geq 2$ for some level mapping $|.|$.

An outline proof of the lemma is in the appendix.

Theorem 20. *Suppose $P_0, \ldots, P_n$ is a Seki transformation sequence and P_0 is recurrent. Then P_n is recurrent.*

Proof. (Outline:) Suppose P_0 is recurrent by some level mapping$|.|'$. Construct a level mapping $|.|$ in the same way as in Lemma 19. The proof is by induction on the length of the transformation sequence. The case for P_0 is established by the construction of $|.|$. The inductive case for unfolding is a straightforward case analysis of the unfolding rule. For the folding case suppose P_{i+1} is obtained from P_i by folding $C : A \leftarrow Q_1\theta, Q$ using the folding clause $C_0 : B \leftarrow Q_1$ in P_{new} to obtain $C' : A \leftarrow B\theta, Q$ in P_{i+1}. By Lemma 19 and using the definition of folding, the 'distance' between A and each atom in $Q_1\theta$ is at least 2 (with respect to level mapping $|.|$). Also, since $C_0 \in P_{new}$ the distance between B and Q_1 is at least 1. Therefore the 'distance' between the head of C' and each atom in its body is at least 1. Therefore P_{i+1} is recurrent with respect to $|.|$.□

7 Conclusion

The results of this work can be summarised as follows: we have introduced a goal replacement rule based on termination analysis. By making the requirement that all ground calls to the head of the transformed clause terminate, we have shown that the replacement rule preserves the success set of a definite program. Finite failure is preserved when a symmetric goal replacement rule is used, with an extended notion of goal equivalence. We have shown that a transformation system based on our rules can perform 'foldings' based on equivalences that depend on recursive definitions of predicates in the original program. Tamaki and Sato's folding rule and other similar versions do not allow this. Finally we have shown that Seki's unfold/fold rules (restricted to definite programs) preserve recurrence.

The identification of termination as the key property could be said merely to shift the problem of checking goal replacement conditions to that of establishing termination properties. In practice this is so, but the advantages of doing this are that promising research on termination analysis can be brought to bear, and links between folding and goal replacement are clarified and strengthened.

Two approaches to implementation can be envisaged. Firstly, automated termination analysis tools could provide direct checking of the goal replacement conditions (e.g. checking the transformed program for recurrence or acceptability). Secondly, we could search for some syntactic conditions that guaranteed applicability of the replacement rule, as in forms of 'folding' that are special cases of our replacement rule.

Future research possibilities include the following. The replacement rule could be extended to normal programs. This is obviously dependent on advances in the termination analysis of such programs, since finite failure is closely bound up with termination analysis. The relationship between the goal replacement rule we have introduced and other existing goal replacement/folding rules, such as [GS91] could be further investigated. The termination properties preserved by folding/goal replacement conditions in systems other than Seki's can be investigated. Application of the goal replacement rule can be extended by using weaker notions of termination (such as in [DVB92]).

Acknowledgements

We thank M. Proietti for pointing out a serious error in a previous version. We also thank A. Pettorossi and the LOPSTR'94 referees for comments and corrections.

References

[AP90] K.R. Apt and D. Pedreschi. Studies in Pure Prolog: Termination. In J.W. Lloyd, editor, *Proceedings of the Esprit symposium on computational logic*, pages 150–176, 1990.

[BB93] D. Boulanger and M. Bruynooghe. Using abstract interpretation for goal replacement. In Y. Deville, editor, *Logic Program Synthesis and Tranformation (LOPSTR'93)*, Louvain-la-Neuve, 1993.

[BCE92] A. Bossi, N. Cocco, and S. Etalle. Transforming Normal Programs by Replacement. In *Third Workshop on Metaprogramming in Logic*, Uppsala, 1992. META92.

[BD77] R.M. Burstall and J. Darlington. A transformation system for developing recursive programs. *Journal of the ACM*, 24:44–67, 1977.

[BE94] A. Bossi and S. Etalle. Transforming Acyclic Programs. *ACM Transactions on Programming Languages and Systems*, (to appear); also available as CWI Technical Report CS-R9369 December 1993, CWI, Amsterdam, 1994.

[Bez89] M. Bezem. Characterising Termination of Logic Programs with Level Mappings. In E.L. Lusk and R.A. Overbeek, editors, *Proceedings NACLP89*, pages 69–80, 1989.

[Coo92] J. Cook. A transformation system for definite logic programs based on termination analysis. Master's thesis, School of Mathematics, University of Bristol, 1992.

[DV92] D. De Schreye and K. Verschaetse. Termination of Logic Programs:Tutorial Notes. In *Third Workshop on Metaprogramming in Logic*, Uppsala, 1992. META92.

[DVB92] D. De Schreye, K. Verschaetse, and M. Bruynooghe. A Framework for Analysing the Termination of Definite Logic Programs with respect to call patterns. In ICOT, editor, *Proceedings of the International Conference on Fifth Generation Computer Systems*, 1992.

[GS91] P.A. Gardner and J.C. Shepherdson. Unfold/fold transformations of logic programs. In J.L Lassez and G. Plotkin, editors, *Computational Logic: Essays in Honour of Alan Robinson.* MIT Press, 1991.

[Llo87] J.W. Lloyd. *Foundations of Logic Programming, 2nd Edition.* Springer-Verlag, 1987.

[LS91] J.W. Lloyd and J.C. Shepherdson. Partial Evaluation in Logic Programming. *Journal of Logic Programming*, 11(3 & 4):217–242, 1991.

[Mah87] M.J. Maher. Correctness of a Logic Program Transformation System. Research Report RC13496, IBM, T.J. Watson Research Center, 1987.

[Plü92] L. Plümer. Automatic Termination Proofs for Prolog Programs Operating on Nonground Terms. In *Proceedings ILPS'91,San Diego*, pages 503–517. MIT Press, 1992.

[PP93] M. Proietti and A. Pettorossi. Synthesis of programs from unfold/fold proofs. In Y. Deville, editor, *Logic Program Synthesis and Tranformation (LOPSTR'93)*, Louvain-la-Neuve, 1993.

[Sek89] H. Seki. Unfold/Fold Transformation of Stratified Programs. In G. Levi and M. Martelli, editors, *Sixth Conference on Logic Programming,Lisboa,Portugal.* The MIT Press, 1989.

[TS84] H. Tamaki and T. Sato. Unfold/Fold Transformation of Logic Programs. In *Proceedings of the Second international Logic Programming Conference*, pages 127–138, Uppsala, 1984.

[UV88] J.D. Ullman and A. Van Gelder. Efficient tests for top-down termination of logical rules. *Journal of the ACM, 35(2)*, pages 345–373, 1988.

[VP86] T. Vasak and J. Potter. Characterisation of Terminating Logic Programs. In *Proceedings 1986 Symposium on Logic Programming, Salt Lake City*, pages 140–147, 1986.

Appendix: Proofs of Lemma 9 and Lemma 19

Lemma 9: Let the definite program P' be obtained by applying a goal replacement transformation to a clause in P. Then for all definite goals $\leftarrow B$,

– $P' \cup \{\leftarrow B\}$ has a finitely failed SLD-tree implies that $P \cup \{\leftarrow B\}$ has no SLD-refutation.

Proof. Proof is by induction on the depth of the smallest finitely failed SLD-tree for $P' \cup \{\leftarrow B\}$. Let $B = B_1, \ldots, B_m$.

For the base case, suppose that $P' \cup \{\leftarrow B\}$ has a finitely-failed SLD-tree of depth 0. This means that some atom B_j $(1 \leq j \leq m)$ fails to unify with the head of any clause in P'. Goal replacement has no effect on clause heads, so B_j also fails to unify with the head of any clause in P. Hence $P \cup \{\leftarrow B\}$ has no SLD-refutation.

For the induction step, assume that the property holds for goals G such that $P' \cup \{G\}$ has a finitely-failed SLD-tree of depth at most k, and that $P' \cup \{\leftarrow B\}$ has a finitely-failed SLD-tree F' of depth $k+1$. Let $\leftarrow B$ have n immediate successor nodes, $\leftarrow S_1, \ldots, \leftarrow S_n$, in F', and let $D_1, \ldots, D_n$ be the corresponding clauses used. Each $\leftarrow S_i$ $(i = 1 \ldots n)$ is at the root of a finitely failed SLD-tree of depth at most k. By the induction hypothesis $P \cup \{\leftarrow S_i\}$ (for $i = 1 \ldots n$) has no SLD-refutation.

If none of the clauses $D_1, \ldots, D_n$ is the clause produced by the goal replacement, then clearly $P \cup \{\leftarrow B\}$ has no SLD-refutation, since otherwise at least one $P \cup \{\leftarrow S_i\}$ would have a refutation, violating the induction hypothesis.

If D_i, say, is the clause produced by the goal replacement, then we have

$$D_i : A \leftarrow Q_2, Q \text{ in } P'$$

which replaced

$$C : A \leftarrow Q_1, Q \text{ in } P$$

where for some j $(1 \leq j \leq m)$ B_j is unifiable with A with mgu θ. Assume without loss of generality that $j = 1$.

Then $S_i = (Q_2, Q, B_2, \ldots, B_m)\theta$ and $P \cup \{\leftarrow S_i\}$ has no SLD-refutation, by the induction hypothesis. Consider the computation of $P \cup \{\leftarrow S_i\}$. There are two cases to consider.

(1) Suppose $P \cup \{\leftarrow Q_2\theta\}$ succeeds with computed answer α. Then $P \cup \{\leftarrow (Q, B_2, \ldots, B_m)\theta\alpha\}$ has no SLD-refutation. By the definition of goal replacement, $Q_1 \equiv_{\text{vars}(A,Q)} Q_2$ in P. The mgu θ acts on the variables of A so $P \cup \{\leftarrow Q_1\theta\}$ succeeds with computed answer $\hat{\alpha}$ where α and $\hat{\alpha}$ agree on variables in $A\theta$ and $Q\theta$. It follows that $P \cup \{\leftarrow (Q, B_2, \ldots, B_m)\theta\hat{\alpha}\}$ has no SLD-refutation, and therefore $P \cup \{\leftarrow (Q_1, Q, B_2, \ldots, B_m)\theta\}$ has no SLD-refutation.

(2) Suppose $P \cup \{\leftarrow Q_2\theta\}$ has no SLD-refutation. Again, since $Q_1 \equiv_{\text{vars}(A,Q)} Q_2$ in P and θ acts on the variables of A, it follows that $P \cup \{\leftarrow Q_1\theta\}$ has no SLD-refutation. Therefore $P \cup \{\leftarrow (Q_1, Q, B_2, \ldots, B_m)\theta\}$ has no SLD-refutation.

It now follows that $P \cup \{\leftarrow B\}$ has no SLD-refutation, since otherwise at least one of the goals $\leftarrow S_1, \ldots, \leftarrow S_{i-1}, \leftarrow S_{i+1}, \ldots, \leftarrow S_n$ or $\leftarrow (Q_1, Q, B_2, \ldots, B_m)\theta$ would have an SLD-refutation, violating the induction hypothesis.□

Lemma 19: Let $P_0, \ldots, P_n$ be a Seki transformation sequence and let P_0 be recurrent. Then for all clauses $A \leftarrow B_1, \ldots, B_m$ in P_i:

(i) If A has an old predicate, none of the B_j is inherited from P_0.
(ii) If B_j has an old predicate and is not inherited from P_0, then for all ground instances $(A \leftarrow B_1, \ldots, B_m)\theta$ we have $|A\theta| - |B_j\theta| \geq 2$ for some level mapping $|.|$.

Proof. (Outline): Suppose P_0 is recurrent by level mapping $|.|'$. Construct a level mapping $|.|$ as follows:

- $|A\alpha| = 2 * |A\alpha|'$ if A has an old predicate and $A\alpha$ is a ground atom.
- $|A\alpha| = \max\{|A_1\alpha|, \ldots, |A_p\alpha|\} + 1$ if $(A \leftarrow A_1, \ldots, A_p)\alpha$ is a ground instance of a clause in P_{new}.

Note that (i) holds trivially from the definition of inherited atom. We prove (ii) by induction on the length of the transformation sequence. The base case (that is, for P_0) is established by the construction of the level mapping $|.|$. The inductive case is a straightforward case analysis using the definitions of unfold and fold.□