An Unfold/Fold Transformation Framework for Definite Logic Programs

Abhik Roychoudhury
National University of Singapore, Singapore
K. Narayan Kumar
Chennai Mathematical Institute, India
C.R. Ramakrishnan
State University of New York at Stony Brook, USA
and
I.V. Ramakrishnan
State University of New York at Stony Brook, USA

Given a program P, an unfold/fold program transformation system derives a sequence of programs $P = P_0$, P_1, ..., P_n, such that P_{i+1} is derived from P_i by application of either an unfolding or a folding step. Unfolding corresponds to a resolution step, while the folding step corresponds to replacing an occurrence of the r.h.s. of a clause with its head. In general the application of a folding step can result in a semantically different program (i.e. P_{i+1} and P_i may not have the same semantics.) Thus, there is a need to consider restricted varieties of folding. These restrictions represent sufficient conditions that ensure preservation of semantics under the application of a folding step. Existing unfold/fold transformation systems for definite logic programs differ from one another mainly in the restrictions imposed on folding. One of the restrictions is is usually a condition on the syntax of the clauses that participate in the folding step.

In this paper we develop a *parameterized* framework for unfold/fold transformations that do not restrict folding by the syntax of the participating clauses. The framework is parameterized by a "measure space" and some associated measure functions. Any measure space and measure function immediately yields a semantics preserving unfold/fold transformation system. The power of the system is determined by the choice of the measure space and functions, and the correctness of the transformations follows from the correctness of the framework. The unfold/fold transformation framework is extended with a *goal replacement* rule which allows semantically equivalent conjunctions of atoms to be interchanged. We show that various existing transformation systems can be obtained by instantiating the parameters of our framework. More importantly, we use our framework to construct a new unfold/fold transformation system. This transformation system is provably more powerful (in terms of transformation sequences allowed) than existing transformation systems.

Our transformation framework is useful for reasoning about programs. In particular, we have used the transformation rules developed in this paper to inductively prove temporal properties of transition systems (whose transition relation is encoded as a logic program). In this paper, we show the issues in automating each application of the transformation rules (including goal replacement). We also provide an illustrative example to demonstrate how the additional power of our folding rule is used in constructing induction proofs.

Categories and Subject Descriptors: D.3.2 [**Language Classifications**]: Constraint and Logic Languges; F.3.2 [**Logics and Meanings of Programs**]: Semantics of Programming Languages; I.2.2 [**Automatic Programming**]: Program transformation

General Terms: Logic Programming, Program Transformation

Additional Key Words and Phrases: Unfolding, Folding

Contact Author's Address: Abhik Roychoudhury, School of Computing, National University of Singapore, S16 Level 5, 3 Science Drive 2, Singapore 117543. E-mail: `abhik@comp.nus.edu.sg`

1. INTRODUCTION

Some of the most extensively studied transformation systems for definite logic programs are the so called *unfold/fold* transformation systems. At a high level unfold and fold transformations can be viewed as follows. Definite logic programs consist of definitions of the form $A{:-}\ \phi$ where A is an atom and ϕ is a positive boolean formula over atoms. Unfolding replaces an occurrence of A in a program with ϕ while folding replaces an occurrence of ϕ with A. Folding is called *reversible* if its effects can be undone by an unfolding, and *irreversible* otherwise. An unfold/fold transformation system for definite logic programs was first described in a seminal paper by Tamaki and Sato [1984]. In the flurry of research activity that followed, a number of unfold/fold transformation systems were developed. Kanamori and Fujita [1987] proposed a transformation system that was based on maintaining counters to guide folding. Maher [Maher 1987; 1993] described a transformation system that permits only reversible folding. The basic Tamaki-Sato system itself was extended in several directions (e.g., to handle folding with multiple clauses [Gergatsoulis and Katzouraki 1994], negation [Aravindan and Dung 1995; Seki 1991; 1993]) and applied to program optimization problems (e.g., [Bossi et al. 1990; Boulanger and Bruynooghe 1993; Pettorossi et al. 1997]). (See Pettorossi and Proietti [1998] for an excellent survey of research on this topic over the past decade).

Correctness of Unfold/Fold Transformations. Correctness proofs for unfold/fold transformations consider *transformation sequences* of the form $P_0, P_1, \ldots$, where P_0 is an initial program and P_{i+1} is obtained from P_i by applying an unfolding or folding transformation. The proofs usually show that all programs in the transformation sequence have the same least Herbrand model. It is easy to verify that transforming P_i to P_{i+1} using unfolding or folding is *partially correct*, i.e., the least model of P_{i+1} is a subset of that of P_i. It is also easy to show, by induction on the structure of the proof trees, that unfolding transformation is *totally* correct, i.e., it *preserves* the least model. However, as illustrated below, indiscriminate folding may introduce circularity in definitions, thereby replacing finite proof paths with infinite ones.

Consider the sequence of programs in Figure 1. In the figure, P_1 is derived by unfolding the occurrence of `q(X)` in the first clause of P_0. P_2 is derived from P_1 by folding the literal `q(X)` in the body of the second clause of predicate `p` into `p(X)` using the clause `p(X) :- q(X)` in P_0. Alternatively, consider the transformation

```
                      p(a).                 p(a).
p(X):-q(X).           p(f(X)):-q(X).        p(f(X)):-p(X).
q(a).                 q(a).                 q(a).
q(f(X)):-q(X).        q(f(X)):-q(X).        q(f(X)):-q(X).
```

Program P_0 Program P_1 Program P_2

Fig. 1. An example of correct unfold/fold transformation sequence

sequence in figure 2. By folding `q(X)` in the second clause of `p` in P_1 (using the second clause defining `q` in P_1), we obtain program P'_2. Now folding `q(X)` in the second clause of `q` in P'_2 (using second clause of `p` in P_1), we get program P'_3, whose least model differs from that of P_0.

Transformation Systems with Irreversible Folding. If the folding transformation is reversible, then since its effect can be undone by an unfolding, any partially correct unfold/fold transformation sequence is also totally correct. However, for reversibility, folding at step i of the transformation can *only* use the clauses in P_i. Therefore reversibility is a *restrictive* condition that seriously limits the power of unfold/fold systems by disallowing many correct folding transformations, such as the one used to derive P_2 from P_1. Hence almost all research on unfold/fold transformations have focused on constructing systems that permit irreversible folding. In such systems, folding at step i can use clauses that are not in P_i. For example, in the original and extended Tamaki-Sato systems [1984; 1986a] folding always uses clauses in P_0 whereas in the Kanamori-Fujita system [1987] the clauses can come from any P_j $(j \leq i)$. But ensuring total correctness of irreversible transformation sequences is difficult. In order to ensure that folding is still totally correct, these systems permit folding using only clauses with certain (syntactic) properties. For instance, the original Tamaki-Sato system permits folding using a single clause only (*conjunctive* folding) and this clause is required to be non-recursive. In [Gergatsoulis and Katzouraki 1994] the above system was extended to allow folding with multiple clauses (*disjunctive* folding) but all the clauses are required to be be non-recursive. Kanamori and Fujita [1987] as well Tamaki and Sato in a later paper [1986a] gave two different approaches for conjunctive folding using recursive clauses. But the design of a transformation system that allows folding in the presence of both disjunction *and* recursion has remained open so far. We will describe such a system in this paper.

To generalize in this direction one needs to first understand the strengths and limitations of the above systems. The key observation is that, although the book-keeping needed to determine permissible foldings appear radically different in the different systems, there is a striking similarity in how the transformations are proved correct. Essentially, these systems associate some *measure* with different program elements, namely, atoms and clauses to determine whether folding is permissible in that step (e.g., "foldable" flag in [Tamaki and Sato 1984], descent levels/strata numbers in [Tamaki and Sato 1986a], and counters in [Kanamori and Fujita 1987]). Moreover, they ensure that each transformation step maintains an invariant relating proofs in the derived program to the various measures (e.g., the notions of

Program P_0	Program P_1	Program P'_2	Program P'_3
`p(X):-q(X).`	`p(a).`	`p(a).`	`p(a).`
`q(a).`	`p(f(X)):-q(X).`	`p(f(X)):-q(f(X)).`	`p(f(X)):-q(f(X)).`
`q(f(X)):-q(X).`	`q(a).`	`q(a).`	`q(a).`
	`q(f(X)):-q(X).`	`q(f(X)):-q(X).`	`q(f(X)):-p(f(X)).`

Fig. 2. An example of incorrect unfold/fold transformation sequence

rank-consistency in [Kanamori and Fujita 1987; Tamaki and Sato 1984], weight-consistency in [Gergatsoulis and Katzouraki 1994] and μ-completeness in [Tamaki and Sato 1986a]). This raises another interesting question: can we exploit the similarities in the correctness proofs of irreversible unfold/fold systems to develop an abstract framework. Such a framework will specify the obligations that must be satisfied to ensure total correctness and hence can simplify construction of unfold/fold systems to the extent that one is relieved of the burden of giving correctness proofs. We propose such a framework in this paper.

Summary of Results. In this paper, we develop a general transformation framework for definite logic programs parameterized by certain abstract measures. These abstract measures are obtained by suitably abstracting and extending the measures used in [Gergatsoulis and Katzouraki 1994; Kanamori and Fujita 1987; Tamaki and Sato 1984; 1986a] (see Section 2). We relax the invariants needed in the proofs to permit *approximation* of measure values. This is the key idea that enables us to fold using multiple recursive clauses. We prove the correctness of transformations in the framework based only on the properties of the abstract measures. We show that various existing unfold/fold transformation systems can be derived from the framework by instantiating these abstract measures (see Section 4). We also show how the framework can be extended to include the Goal Replacement transformation (see Section 3).

The parameterized framework presented in this paper is useful for understanding the strengths and limitations of existing transformation systems. It also enables the construction of new unfold/fold systems. As evidence we obtain SCOUT (Strata and COunter based Unfold/fold Transformations), a transformation system that permits disjunctive folding using recursive clauses. The development of SCOUT was based on two crucial observations made possible by the framework. First, when instantiating the framework to obtain the Kanamori-Fujita system, it is easy to see that the counters (the measure used in their system) may come from any linearly ordered set; this permits us to incorporate stratification into the counters to obtain a system that generalizes the extended Tamaki-Sato system [1986a] as well as the Kanamori-Fujita system. Secondly, the framework enables us to maintain approximate counters; we can hence generalize the combination of the Kanamori-Fujita and the extended Tamaki-Sato systems to fold using multiple recursive clauses.

The motivation behind the development of our parameterized transformation framework is its applicability in inductive reasoning. Unfold/fold transformations have traditionally been used for program efficiency improvement. However, there has been a parallel line of work in using unfold/fold transformations for constructing proofs [Hsiang and Srivas 1987; Kanamori and Fujita 1986; Pettorossi and Proietti 1999; Roychoudhury and Ramakrishnan 2001]. Roughly speaking, these works prove predicate equivalences of the form $\mathtt{p} \equiv \mathtt{q}$ by transforming `p` and `q` such that their equivalence can be inferrred from syntax. Our generalized folding rule is useful for constructing such proofs. In particular, when `p` , `q` are defined using predicates with multiple clauses (some of which may be recursive) we may need a more general folding rule to transform `p`, `q`. An interesting application where such a situation crops up is in the verification of temporal properties (predicates describing temporal properties are encoded using multiple recursive clauses). We show the application of our more general transformations with a detailed example in Section 5.

2. A PARAMETERIZED TRANSFORMATION FRAMEWORK

We now describe our parameterized unfold/fold transformation framework and illustrate the abstractions by drawing analogies to the Kanamori-Fujita system.

We assume familiarity with the standard notions of terms, models, substitutions, unification, most general unifier (mgu), definite clauses, SLD resolution, and proof trees. For a background on these materials, the reader is referred to [Das 1992; Lloyd 1993]. We will use the following symbols (possibly with primes and subscripts): P to denote a definite logic program; $M(P)$ its least Herbrand model; C and D for clauses; A, B to denote atoms and literals and σ for most general unifier (mgu).

2.1 Unfolding and Folding

The unfolding and folding rules are defined as follows:

RULE 1. **Unfolding** Let C be a clause in P_i and A an atom in the body of C. Let $C_1, \ldots, C_m$ be the clauses in P_i whose heads are unifiable with A with most general unifier $\sigma_1, \ldots, \sigma_m$. Let C'_j be the clause that is obtained by replacing $A\sigma_j$ by the body of $C_j\sigma_j$ in $C\sigma_j$ $(1 \leq j \leq m)$. Assign $(P_i - \{C\}) \cup \{C'_1, \ldots, C'_m\}$ to P_{i+1}. □

RULE 2. **Folding** Let $\{C_1, \ldots, C_m\} \subseteq P_i$ where C_l denotes the clause

$$A{:\!-}\ A_{l,1}, \ldots, A_{l,n_l}, A'_1, \ldots, A'_n$$

and $\{D_1, \ldots, D_m\} \subseteq P_j$ $(j \leq i)$ where D_l is the clause $B_l{:\!-}\ B_{l,1}, \ldots, B_{l,n_l}$. Further, let:

(1) $\forall 1 \leq l \leq m\ \exists \sigma_l\ \forall 1 \leq k \leq n_l\ A_{l,k} = B_{l,k}\sigma_l$

(2) $B_1\sigma_1 = B_2\sigma_2 = \cdots = B_m\sigma_m = B$

(3) $D_1, \ldots, D_m$ are the only clauses in P_j whose heads are unifiable with B

(4) $\forall 1 \leq l \leq m$, σ_l substitutes the internal variables[1] of D_l to distinct variables which do not appear in $\{A, B, A'_1, \ldots A'_n\}$.

Then $P_{i+1} := (P_i - \{C_1, \ldots, C_m\}) \cup \{C'\}$ where $C' \equiv A{:\!-}\ B, A'_1, \ldots, A'_n$. □

$D_1, \ldots, D_m$ are the *folder* clauses, $C_1, \ldots, C_m$ are the *folded* clauses, and B is the *folder* atom. A folding step is *conjunctive* whenever both the folder and folded clauses are singleton sets and is *disjunctive* otherwise. Note that in the latter case a set of folded clauses is *simultaneously* replaced by a single clause using a set of folder clauses.

We say that $P_0, P_1, \ldots, P_n$ is an unfold/fold transformation sequence if the program P_{i+1} is obtained from P_i $(i \geq 0)$ by application of an unfold or a fold rule. Partial correctness of an unfold/fold transformation sequence (Theorem 1) now follows easily.

[1]Variables appearing in the body of a clause, but not its head

Theorem 1. **Partial Correctness** *Let $P_0, P_1, \ldots, P_i$ be a program transformation sequence where $M(P_j) = M(P_0)$ for all $0 \le j \le i$. If P_{i+1} is obtained from P_i by applying either unfolding or folding, then $M(P_{i+1}) \subseteq M(P_i)$.*

Proof. This is established by showing that a proof T of any ground atom $A \in M(P_{i+1})$, has a corresponding proof T' of A in P_i. This can be proved by induction on the structure of T. Let $C = (A{:-}\ A_1, \ldots, A_n)$ be the clause applied at the root of T. There are three cases:
Case 1: $C \in P_i$.
Then, the result follows by induction hypothesis.
Case 2: C is obtained by unfolding.
Let $C \in P_{i+1}$ be obtained by unfolding clause $C' \in P_i$ using clause $D \in P_i$. Without loss of generality, there exist ground instances of C' and D, in P_i, of the form $A{:-}\ B, A_{k+1}, \ldots, A_n$ and $B{:-}\ A_1, \ldots, A_k$. The proof T' of A can be then constructed by applying clause C' at the root, and then clause D. The existence of ground proofs of $A_1, \ldots, A_n$ in P_i follows by induction hypothesis.
Case 3: C is obtained by folding.
Let $C \in P_{i+1}$ be obtained by folding $C' \in P_i$ using $D \in P_j (j \le i)$ as folder. Let A_1 be the folder atom in clause C, *i.e.* the atom introduced by folding. Since $M(P_j) = M(P_i)$ and $A_1 \in M(P_i)$ (by induction hypothesis) therefore $A_1 \in M(P_j)$. Thus, A_1 has a ground proof T_1 in P_j. By condition 3 of the folding transformation, the clause applied at the root of T_1 *must* be one of the folder clauses. Let this folder clause be D and let the corresponding folded clause be $C' \in P_i$. Then, without loss of generality, C' and D have ground instances of the form $A{:-}\ A_{1,1}, \ldots, A_{1,l}, A_2, \ldots, A_n$ and $A_1{:-}\ A_{1,1}, \ldots, A_{1,l}$ respectively. Since $A_{1,1}, \ldots, A_{1,l} \in M(P_j)$ therefore $A_{1,1}, \ldots, A_{1,l} \in M(P_i)$. Thus, $A_{1,1}, \ldots, A_{1,l}$ have ground proofs in P_i. Also, $A_2, \ldots, A_n$ have ground proofs in P_i by induction hypothesis. Thus, we can construct a ground proof of A in P_i by applying clause C' at the root. This completes the proof. □

2.2 Measures, Measure-Consistent Proofs and Total Correctness

Total correctness of an unfold/fold transformation sequence is established by induction over some well-founded order to construct a proof in P_{i+1} for any atom A in $M(P_i)$. To see the subtleties involved in proving total correctness, consider transforming P_i to P_{i+1} using a conjunctive folding step. To construct a proof of A (the head of the folded clause) in P_{i+1}, we need a proof of B (the folder atom) in P_{i+1}. But the existence of such a proof can be established (by induction hypothesis) only if B is less than A in the well-founded order on which the inductive argument is presented. Note that if the folder clause is picked from P_j, $j < i$, we cannot use simple well-founded orders like size of proof trees in P_i, as the proof of B in P_i can be larger in size than the proof of A in P_i.

It is worth noting that we do not attempt to translate every proof of A in P_i to a proof of A in P_{i+1}. Instead, following [Kanamori and Fujita 1987; Tamaki and Sato 1984; 1986a] we consider a "special proof" called *strongly measure consistent proof* (see Definition 6) of A in P_i and construct a proof of A in P_{i+1}. The induction proof for establishing total correctness is completed by showing that the proof of A in P_{i+1} thus constructed is itself strongly measure consistent.

Recall that irreversible folding steps need to be constrained in order to preserve the semantics. In order to enforce these constraints, we maintain some book-keeping information as we perform the transformations, formalized using the following notions of *Measure structure*, *Atom measure*, and *Clause measure*.

DEFINITION 1. **Measure Structure** *A Measure Structure is a 4-tuple* $\mu = \langle \mathcal{M}, \oplus, \prec, \mathcal{W} \rangle$ *where* $\langle \mathcal{M}, \oplus \rangle$ *is a commutative group with* $\mathbf{0} \in \mathcal{M}$ *as its identity element,* $\prec$ *is a linear order on* $\mathcal{M}$*,* $\oplus$ *is monotone w.r.t.* $\prec$*, and* $\mathcal{W}$ *is a subset of* $\{x \in \mathcal{M} \mid \mathbf{0} \preceq x\}$*, over which* $\prec$ *is well-founded.*

We will refer to $\mathcal{M}$, the first component of the measure structure, as the *measure space*. We let $\preceq$ denote $\prec$ or $=$. Moreover, we use $\ominus$ to denote the inverse operation of the group $\langle \mathcal{M}, \oplus \rangle$. We also use $\ominus$ as a binary operator, $a \ominus b$ meaning $a \oplus (\ominus b)$ (where $(\ominus b)$ is the inverse of b). The Kanamori-Fujita system [1987] keeps track of integer counters. Thus the measure structure is $\langle \mathbb{Z}, +, <, \mathbb{N} \rangle$, where $\mathbb{Z}$ and $\mathbb{N}$ are the set of integers and natural numbers respectively, $+$ denotes integer addition, and $<$ is the arithmetic comparison operator.

DEFINITION 2. **Atom Measure** *An atom measure* α *of a program* P *w.r.t. a measure structure* μ *is a partial function from the Herbrand base of* P *to* $\mathcal{W}$ *such that it is total on the least Herbrand model of* P*. For our purposes, it suffices to use the same atom measure for each program in a transformation sequence.*

In the Kanamori-Fujita system, the atom measure of any P_i in the transformation sequence is the number of nodes in the shortest proof tree of A in the initial program P_0. The proof of total correctness for folding will induct on the atom measure, relating the atom measure of A (the head of the folded clauses) with the atom measure of B (the folder atom).

DEFINITION 3. **Clause Measure** *A clause measure* $(\gamma_{lo}, \gamma_{hi})$ *of a program* P *w.r.t. a measure structure* μ *is a pair of total functions from clauses of* P *to* $\mathcal{M}$ *such that* $\forall C \in P \;\gamma_{lo}(C) \preceq \gamma_{hi}(C)$*.*

In the Kanamori-Fujita system, γ_{lo} and γ_{hi} are the same and map each clause to its corresponding counter value. However, as we will see later, to allow disjunctive folding we will need the two distinct functions γ_{lo} and γ_{hi}. Henceforth, we denote the clause measure of a program P_i by $(\gamma_{lo}^i, \gamma_{hi}^i)$. We will now develop the idea of "special proofs" mentioned earlier. For that purpose, we need the definition:

DEFINITION 4. **Ground Proof of an Atom** *Let* T *be a tree, each of whose nodes is labeled with a ground atom. Then* T *is a ground proof in program* P*, if every node* A *in* T *satisfies the condition :* A:– $A_1, ..., A_n$ *is a ground instance of a clause in* P*, where* $A_1, ..., A_n$ $(n \geq 0)$ *are the children of* A *in* T*.*

Consider transforming P_i to P_{i+1} by a folding step (see figure below). C and D are the folded and folder clauses respectively and $j < i$.

.....	.....	.....
D : q:– $q_1, ..., q_k$	C : p:– $q_1, ..., q_k, q_{k+1}, ..., q_n$	C' : p:– $q, q_{k+1}, ..., q_n$
.....	.....	.....
Program P_j	Program P_i	Program P_{i+1}

In order to show that $\mathtt{p} \in M(P_i) \Rightarrow \mathtt{p} \in M(P_{i+1})$ by induction on $\prec$, we would like to show that $\alpha(\mathtt{q}) \prec \alpha(\mathtt{p})$. The atoms $\mathtt{p}$ and $\mathtt{q}$ are related by what is shared between the bodies of the clauses C and D. Hence we attempt to relate their measures via the measures of bodies of C and D. Suppose D satisfies

$$\alpha(\mathtt{q}) \preceq \sum_{1 \le i \le k} \alpha(\mathtt{q_i}) \qquad \text{(i)}$$

then we can relate $\alpha(\mathtt{q})$ to the sum of the measures of the body atoms of the folded clause C (since $k \le n$). Further if C satisfies

$$\alpha(\mathtt{p}) \succeq \sum_{1 \le i \le n} \alpha(\mathtt{q_i}) \qquad \text{(ii)}$$

then we can establish that $\alpha(\mathtt{q}) \preceq \alpha(\mathtt{p})$. If either (i) or (ii) is a strict relationship then we can establish that $\alpha(\mathtt{q}) \prec \alpha(\mathtt{p})$. Relations (i) and (ii) form the basis for the notions of *weak* and *strong measure consistency*.

DEFINITION 5. **Weakly Measure Consistent Proof** *A ground proof T in program P_i is weakly measure consistent w.r.t. atom measure α and clause measure $(\gamma^i_{lo}, \gamma^i_{hi})$ if every ground instance A:– $A_1, ..., A_n$ of a clause $C \in P_i$ used in T satisfies $\alpha(A) \preceq \gamma^i_{hi}(C) \oplus \sum_{1\le l \le n} \alpha(A_l)$.*

DEFINITION 6. **Strongly Measure Consistent Proof** *A ground proof T in program P_i is strongly measure consistent w.r.t. atom measure α and clause measure $(\gamma^i_{lo}, \gamma^i_{hi})$ if every ground instance A:– $A_1, ..., A_n$ of a clause $C \in P_i$ used in T satisfies $\forall 1 \le l \le n$ $\alpha(A_l) \prec \alpha(A)$ and $\alpha(A) \succeq \gamma^i_{lo}(C) \oplus \sum_{1\le l \le n} \alpha(A_l)$*

DEFINITION 7. **Measure Consistent Proof** *A ground proof T in program P_i is said to be measure consistent w.r.t. atom measure α and clause measure $(\gamma^i_{lo}, \gamma^i_{hi})$, if it is strongly and weakly measure consistent w.r.t. α and $(\gamma^i_{lo}, \gamma^i_{hi})$.*

We point out that our abstract notion of measure consistency relaxes the concrete notion of rank consistency of [Kanamori and Fujita 1987]. While rank consistency of [Kanamori and Fujita 1987] imposes a strict equality constraint on $\alpha(A)$, measure consistency only *bounds it from above and below.* As we will show later, this facilitates maintenance of approximate information. This is the central idea that permits us to do disjunctive folding using recursive clauses. For proving total correctness, we need :

DEFINITION 8. **Measure consistent Program** *A program P is measure consistent w.r.t. atom measure α and clause measure $(\gamma_{lo}, \gamma_{hi})$, if for all $A \in M(P)$, we have*

(1) All ground proofs of A in P are weakly measure consistent w.r.t. α and $(\gamma_{lo}, \gamma_{hi})$

(2) A has a ground proof in P which is strongly measure consistent w.r.t. α and $(\gamma_{lo}, \gamma_{hi})$

We are now ready to define the abstract conditions on folding and constraints on how the clause measures are to be updated after an unfold/fold step. For each clause C obtained by applying an unfold/fold transformation on program P_i, we derive a lower bound on $\gamma^{i+1}_{hi}(C)$ and an upper bound on $\gamma^{i+1}_{lo}(C)$, denoted by

$GLB^{i+1}(C)$ and $LUB^{i+1}(C)$ respectively. We will see later that the conditions on when the rules become applicable, as well as these bounds are designed to ensure the correctness of the folding step.

We assume that for any atom A (not necessarily ground), $\alpha_{min}(A)$ denotes a lower bound on the measure of any provable ground instantiation of A *i.e.* $\forall\theta\ \alpha_{min}(A) \preceq \alpha(A\theta)$. We use α_{min} in the folding condition of rule 4 below.

RULE 3. **Measure Preserving Unfolding** Let P_{i+1} be obtained from P_i by an unfolding transformation as described in Rule 1. We say that the unfolding step is *measure preserving* if the associated clause measures satisfy the following inequalities: $\forall 1 \leq j \leq m$

$$\gamma_{lo}^{i+1}(C'_j) \ \preceq\ \gamma_{lo}^{i}(C) \oplus \gamma_{lo}^{i}(C_j) \quad (\stackrel{\text{def}}{=} GLB^{i+1}(C'_j) \ ) \tag{1}$$

$$\gamma_{hi}^{i+1}(C'_j) \ \succeq\ \gamma_{hi}^{i}(C) \oplus \gamma_{hi}^{i}(C_j) \quad (\stackrel{\text{def}}{=} LUB^{i+1}(C'_j) \ ) \tag{2}$$

and the clause measure of all other clauses in P_{i+1} are inherited from P_i. □

RULE 4. **Measure Preserving Folding** Let P_{i+1} be obtained from P_i by a folding transformation as described in Rule 2. We say that this folding step is measure preserving, if the associated clause measures satisfy the following: [2]

$$\forall 1 \leq l \leq m.\ \gamma_{hi}^{j}(D_l) \prec \gamma_{lo}^{i}(C_l) \oplus \sum_{1 \leq k \leq n} \alpha_{min}(A'_k)$$

and moreover,

$$\gamma_{lo}^{i+1}(C') \ \preceq\ \min_{1 \leq l \leq m} (\gamma_{lo}^{i}(C_l) \ominus \gamma_{hi}^{j}(D_l)) \quad (\stackrel{\text{def}}{=} GLB^{i+1}(C') \ ) \tag{3}$$

$$\gamma_{hi}^{i+1}(C') \ \succeq\ \max_{1 \leq l \leq m} (\gamma_{hi}^{i}(C_l) \ominus \gamma_{lo}^{j}(D_l)) \quad (\stackrel{\text{def}}{=} LUB^{i+1}(C') \ ) \tag{4}$$

and the clause measure of all other clauses in P_{i+1} are inherited from P_i. □

It should be noted that the above rules do not prescribe *unique* values for upper and lower clause measures for the clauses generated by the transformations. Instead, they only specify bounds of these values; the values themselves are chosen only when instantiating the framework to a concrete system.

Observe from the definition of atom measures that we can always assign $\mathbf{0}$ to α_{min}. However, by setting a more accurate estimate of α_{min}, we can allow more folding steps. As an example, consider any conjunctive folding step where the folded clause $C \in P_i$ has more body atoms than the folder clause $D \in P_j$, and $\gamma_{lo}^{i}(C) = \gamma_{hi}^{j}(D)$. Such a folding step will not be allowed if $\forall A\ \alpha_{min}(A) = \mathbf{0}$.

The Need for Approximate Clause Measures. In the Kanamori-Fujita system, a counter (corresponding to our clause measure) is associated with every clause. Roughly, the counter associated with a clause $C \in P_i$ where $C \equiv A{:-}\ A_1, \ldots, A_n$ indicates the number of interior nodes in the smallest proof tree in P_0 that derives $A_1, \ldots, A_n$ from A. Thus, it is the amount saved (in terms of proof tree size, compared to the smallest proof in P_0) whenever C is used in a proof in P_i. The folding rule is applicable provided the savings accrued in the folded clause is more than that in the folder clause.

To see why a single counter is inadequate for disjunctive folding, consider the following example:

Program P_i	Program P_{i+1}
C_1: `p :- r, t.` (x_1)	
C_2: `p :- s, t.` (x_2)	C': `p :- q, t.` $(?)$
C_3: `q :- r.` (x_3)	C_3: `q :- r.` (x_3)
C_4: `q :- s.` (x_4)	C_4: `q :- s.` (x_4)

P_{i+1} is obtained from P_i by folding $\{C_3, C_4\}$ into $\{C_1, C_2\}$. Now, the savings due to C' in a proof of P_{i+1} depends on whether C_3 or C_4 is used to resolve `q` in that proof. Since this information is unknown at transformation time, we can only keep approximate information about savings. In our framework we choose to approximate the savings by the closed interval $[\gamma_{lo}, \gamma_{hi}]$.

We now have the necessary machinery for establishing total correctness of a sequence of unfold/fold transformations.

Lemma 1. **Preserving Weak Measure Consistency** *Consider a transformation sequence of measure consistent programs $P_0, \ldots, P_i$ such that $M(P_0) = M(P_j)$ for all $0 \leq j \leq i$. Let P_{i+1} be obtained from P_i by applying measure-preserving unfolding or measure-preserving folding. Then, all ground proofs of P_{i+1} are weakly measure consistent.*

Proof. We will use $M(P_{i+1}) \subseteq M(P_i)$, a result which was independently proved in theorem 1. The proof proceeds by induction on size of ground proofs in P_{i+1}. Let T be a ground proof of some ground atom A in P_{i+1}, and let $A{:-}\ A_1, \ldots, A_n$ (where $n \geq 0$) be the ground instance of a clause $C \in P_{i+1}$ that is used at the root of the proof T. Then the proofs of $A_1, \ldots, A_n$ in T are weakly measure consistent by induction hypothesis. Hence, it suffices to show that, $\alpha(A) \preceq \gamma_{hi}^{i+1}(C) \oplus \sum_{1 \leq l \leq n} \alpha(A_l)$.

Case 1: C was inherited from P_i
Since $M(P_{i+1}) \subseteq M(P_i)$, hence $A_1, \ldots, A_n$ are provable in P_i. Therefore, the ground clause $A{:-}\ A_1, \ldots, A_n$ is used at the root of a ground proof in P_i. Since P_i is measure consistent, the result follows.

Case 2: C was obtained by *unfolding*
Let $A_1, \ldots, A_k$ be the instances of the body atoms of C which were introduced through unfolding. By the definition of the unfolding transformation, then there must be clauses C' and C'' in P_i with ground instances $A{:-}\ B, A_{k+1}, \ldots, A_n$ and $B{:-}\ A_1, \ldots, A_k$ respectively with $\gamma_{hi}^{i+1}(C) \succeq \gamma_{hi}^{i}(C') \oplus \gamma_{hi}^{i}(C'')$.

Again, $A_1, \ldots, A_k, A_{k+1}, \ldots, A_n$ are provable in P_i (as $M(P_{i+1}) \subseteq M(P_i)$). Hence, the above mentioned ground instances of C' and C'' are ground clauses used at the root of some proof in P_i. As P_i is a measure consistent program, we have :

$$\alpha(A) \preceq \gamma_{hi}^{i}(C') \oplus \alpha(B) \oplus \sum_{k+1 \leq l \leq n} \alpha(A_l)$$

$$\alpha(B) \preceq \gamma_{hi}^{i}(C'') \oplus \sum_{1 \leq l \leq k} \alpha(A_l)$$

The result now follows by combining these two inequations.

Case 3: C was obtained by *folding*
Let A_1 be the instance of the folder atom (*i.e.* the atom corresponding to the head of the folder clauses) in C, and let $P_j (j \leq i)$ be the program from which folder clauses were picked. We have $M(P_i) = M(P_j) = M(P_0)$, and hence $M(P_{i+1}) \subseteq M(P_j)$. Thus, $A_1 \in M(P_j)$. Since P_j is a measure consistent program, A_1 must have a strongly measure consistent proof T'_{A_1} in P_j. Let the clause used at the root of this proof be D' and let the ground instance of D' used at the root of T'_{A_1} be $A_1{:-}\, A_{1,1}, \ldots, A_{1,k}$. Then, by the strong measure consistency of T'_{A_1}

$$\alpha(A_1) \succeq \gamma^j_{lo}(D') \oplus \sum_{1 \leq l \leq k} \alpha(A_{1,l})$$

But, D' must be a folder clause by definition of folding. Hence, there must be a clause C' in P_i with a ground instance $A{:-}\, A_{1,1}, \ldots, A_{1,k}, A_2, \ldots, A_n$ (this is the folded clause corresponding to D'). Now, $A_2, \ldots, A_n$ are provable in P_i (since $M(P_{i+1}) \subseteq M(P_i)$), and also $A_{1,1}, \ldots, A_{1,k}$ are provable in P_i (since $M(P_j) = M(P_i)$). Therefore, the above mentioned ground instance of C' is used at the root of a weakly measure consistent proof of A in P_i (since program P_i is measure consistent). Hence

$$\begin{aligned}
\alpha(A) &\preceq \gamma^i_{hi}(C') \oplus \sum_{1 \leq l \leq k} \alpha(A_{1,l}) \oplus \sum_{2 \leq l \leq n} \alpha(A_l) \\
&\preceq \gamma^i_{hi}(C') \ominus \gamma^j_{lo}(D') \oplus \alpha(A_1) \oplus \sum_{2 \leq l \leq n} \alpha(A_l) \\
&\preceq \gamma^i_{hi}(C') \ominus \gamma^j_{lo}(D') \oplus \sum_{1 \leq l \leq n} \alpha(A_l)
\end{aligned}$$

Since D' and C' are folder and folded clauses and C is the clause obtained by folding therefore $\gamma^{i+1}_{hi}(C) \succeq \gamma^i_{hi}(C') \ominus \gamma^j_{lo}(D')$, and hence

$$\alpha(A) \preceq \gamma^{i+1}_{hi}(C) \oplus \sum_{1 \leq l \leq n} \alpha(A_l)$$

Thus, we have established that any arbitrary ground proof T in P_{i+1} is weakly measure consistent. □

We now formally state and prove the total correctness of any unfold/fold transformation sequence.

Theorem 2. **Total Correctness** *Let $P_0, P_1, \ldots, P_i$ be a transformation sequence of measure consistent programs such that $M(P_0) = M(P_j)$ for all $0 \leq j \leq i$. Let P_{i+1} be obtained from P_i by applying measure-preserving unfolding or measure-preserving folding. Then,* (i) *$M(P_{i+1}) = M(P_i)$ and* (ii) *P_{i+1} is a measure-consistent program.*

Proof. By theorem 1, we have $M(P_{i+1}) \subseteq M(P_i)$, and by lemma 1 we know that all ground proofs of P_{i+1} are weakly measure consistent. Hence it is sufficient to prove that (1) $M(P_i) \subseteq M(P_{i+1})$ and (2) $\forall A \in M(P_{i+1})$, A has a strongly measure consistent proof in P_{i+1}.

Consider any ground atom $A \in M(P_i)$. Since P_i is measure consistent, A has a strongly measure consistent proof T in P_i. We now construct a strongly measure consistent proof T' of A in P_{i+1}. Construction of T' proceeds by induction on atom measures. Let C be a clause used at the root of T. Let $A{:-}\ A_1, ..., A_n$ (where $n \geq 0$) be the ground instantiation of C at the root of T. Since T is strongly measure consistent $\alpha(A_i) \prec \alpha(A)$, for all $1 \leq i \leq n$. Hence, we have strongly measure consistent proofs $T'_1, ..., T'_n$ of $A_1, ..., A_n$ in P_{i+1}. We construct T' by considering the following cases:

Case 1: C is *inherited* from P_i into P_{i+1}

T' is constructed with $A{:-}\ A_1, ..., A_n$ at its root and $T'_1, ..., T'_n$ as its children. This proof T' is strongly measure consistent.

Case 2: C is *unfolded.*

Let A_1 be the atom in the body of C which is unfolded. Let the clause used to resolve A_1 in T be C_1 and the ground instance of C_1 used be $A_1{:-}\ A_{1,1}, ..., A_{1,l_1}$. By definition of unfolding, $A{:-}\ A_{1,1}, ..., A_{1,l_1}, A_2, ..., A_n$ is a ground instance of a clause C'_1 in P_{i+1} with $\gamma_{lo}^{i+1}(C'_1) \preceq \gamma_{lo}^i(C) \oplus \gamma_{lo}^i(C_1)$. Also, $\alpha(A_{1,j}) \prec \alpha(A_1)$ and $\alpha(A_1) \prec \alpha(A)$, for all $1 \leq j \leq l_1$. Thus, we have strongly measure consistent proofs $T'_{1,1}, ..., T'_{1,l_1}$ of $A_{1,1}, ..., A_{1,l_1}$ in P_{i+1}. The proof T' is now constructed by applying $A{:-}\ A_{1,1}, ..., A_{1,l_1}, A_2, ..., A_n$ at the root, and putting $T'_{1,1}, ..., T'_{1,l_1}, T'_2, ..., T'_n$ as the children. Since T is strongly measure consistent,

$$\begin{array}{ll} & \alpha(A) \succeq \gamma_{lo}^i(C) \oplus \sum_{1\leq j\leq n} \alpha(A_j) \text{ and } \alpha(A_1) \succeq \gamma_{lo}^i(C_1) \oplus \sum_{1\leq j\leq l_1} \alpha(A_{1,j}) \\ \Longrightarrow & (\alpha(A) \oplus \alpha(A_1)) \succeq \gamma_{lo}^i(C) \oplus \gamma_{lo}^i(C_1) \oplus \sum_{1\leq j\leq n} \alpha(A_j) \oplus \sum_{1\leq j\leq l_1} \alpha(A_{1,j}) \\ \Longrightarrow & \alpha(A) \succeq \gamma_{lo}^{i+1}(C'_1) \oplus \sum_{2\leq j\leq n} \alpha(A_j) \oplus \sum_{1\leq j\leq l_1} \alpha(A_{1,j}) \end{array}$$

Hence, T' is a strongly measure consistent proof in P_{i+1}.

Case 3: C is *folded.*

Let C (potentially with other clauses) be folded, using folder clauses from P_j, $j \leq i$, to clause C' in P_{i+1}. Assume that $A_1, ..., A_k$ are the instances of the folded atoms in C. Then, C' has a ground instance of the form $A{:-}\ B, A_{k+1}, ..., A_n$ where $B{:-}\ A_1, ..., A_k$ is a ground instance of a folder clause $D \in P_j$.[3] Since $M(P_i) = M(P_j)$ and $A_1, ..., A_k$ are provable in P_i they must also be provable in P_j. Moreover, since $D \in P_j$, $B \in M(P_j) = M(P_i)$. Since P_j is measure consistent,

$$\alpha(B) \preceq \gamma_{hi}^j(D) \oplus \sum_{1\leq l\leq k} \alpha(A_l).$$

Now, by the strong measure consistency of T,

$$\begin{array}{rl} \alpha(A) & \succeq \gamma_{lo}^i(C) \oplus \sum\limits_{1\leq l\leq k} \alpha(A_l) \oplus \sum\limits_{k+1\leq l\leq n} \alpha(A_l) \\ & \succeq \gamma_{lo}^i(C) \oplus (\alpha(B) \ominus \gamma_{hi}^j(D)) \oplus \sum\limits_{k+1\leq l\leq n} \alpha(A_l) \qquad (5) \\ & \succeq (\gamma_{lo}^i(C) \ominus \gamma_{hi}^j(D)) \oplus \alpha(B) \oplus \sum\limits_{k+1\leq l\leq n} \alpha_{min}(A_l) \\ & \succ \alpha(B) \text{ (by condition of measure preserving folding)} \end{array}$$

[3]Recall that in the folding transformation, all clauses in P_j whose head is unifiable with B are folder clauses.

Now, by induction hypothesis, B has a strongly measure consistent proof T'_B in P_{i+1}. We construct T', the proof of A in P_{i+1}, with $A{:}-\ B, A_{k+1}, ..., A_n$ at its root, and $T'_B, T'_{k+1}, ..., T'_n$ as its children. To show that T' is strongly measure consistent, note that $\gamma_{lo}^{i+1}(C') \preceq (\gamma_{lo}^{i}(C) \ominus \gamma_{hi}^{j}(D))$ according to the definition of measure preserving folding, as C and D are folded and folder clauses. Combining this with inequation (5) we get,

$$\alpha(A) \succeq \gamma_{lo}^{i+1}(C') \oplus \alpha(B) \oplus \textstyle\sum_{k+1 \le l \le n} \alpha(A_l)$$

This completes the proof. □

Assigning tighter clause measures. The measure preserving unfolding and folding transformations of Rules 3, 4 provide constraints on the clause measures in P_{i+1}. Note that by applying measure preserving unfolding/folding to program P_i we can generate a clause C which is already in P_i, but with new clause measures. Instead of assigning the clause measures as prescribed by Rules 3 and 4 (computed via addition/subtraction), we can assign tighter measures as follows. Formally, let *unfold(C')* be the set of clauses generated by measure preserving unfolding of $C' \in P_i$ and let there exist a clause C s.t. $C \in unfold(C') \wedge C \in P_i - \{C'\}$. Clearly, then $C \in P_{i+1}$. However, the question is how do we assign $(\gamma_{lo}^{i+1}(C), \gamma_{hi}^{i+1}(C))$, the clause measures of C in P_{i+1}. Similarly, by measure preserving folding of $\{C_1, \ldots, C_m\} \subseteq P_i$, we can generate a clause $C \in P_i - \{C_1, \ldots, C_m\}$. Again, we need to assign $(\gamma_{lo}^{i+1}(C), \gamma_{hi}^{i+1}(C))$. Let the clause measures of C computed by the unfold/fold transformation be $(\gamma'_{lo}, \gamma'_{hi})$. We can then set $\gamma_{lo}^{i+1}(C) = \min(\gamma'_{lo}, \gamma_{lo}^{i}(C))$ and $\gamma_{hi}^{i+1}(C) = \min(\gamma'_{hi}, \gamma_{hi}^{i}(C))$ without affecting the measure consistency of P_{i+1}. For the purposes of measure consistency, note that we could have chosen $\gamma_{hi}^{i+1}(C) = \max(\gamma'_{hi}, \gamma_{hi}^{i}(C))$. Taking the minimum, which also preserves measure consistency, gives us a tighter bound. This also ensures that when we restrict ourselves to conjunctive folding, the lower and higher measures of any clause in program P_i (appearing in some transformation sequence of measure consistent programs) are identical.

3. GOAL REPLACEMENT

Augmenting an unfold/fold transformation system with the goal replacement rule makes it more powerful. In this section we incorporate goal replacement to our parameterized framework. Goal replacement allows semantically equivalent conjunctions of atoms to be freely interchanged. We formally define it below. For a conjunction of atoms $A_1, ..., A_n$, we use the notation $vars(A_1, ..., A_n)$ to denote the set of variables in $A_1, ..., A_n$.

RULE 5. **Goal Replacement** Let C be a clause $A{:}-\ A_1, \ldots, A_k, G$ in P_i, and G' be an atom such that $vars(G) = vars(G') \subseteq vars(A, A_1, ..., A_k)$. Suppose for all ground instantiation θ of G, G' we have $P_i \vdash G\theta \Leftrightarrow P_i \vdash G'\theta$. Then $P_{i+1} := (P_i - \{C\}) \cup \{C'\}$ where $C' \equiv A{:}-\ A_1, \ldots, A_k, G'$. □

Note that although we replace a single atom G by another atom G' (where G and G' do not contain any internal variables), we can replace conjunctions of atoms using a sequence of folding, goal replacement and unfolding transformations.

The above transformation is partially correct. A formal proof of its partial correctness appears below.

THEOREM 3. *Let program P_{i+1} be obtained from program P_i by applying goal replacement as described in rule 5. Then, $M(P_{i+1}) \subseteq M(P_i)$.*

PROOF. We take any ground proof T of some $B \in M(P_{i+1})$ and construct a ground proof T' of B in P_i, thereby proving $M(P_{i+1}) \subseteq M(P_i)$. This proof proceeds by induction on size of ground proofs in P_{i+1}. The base case is obvious because unit clauses are not manipulated by goal replacement. For the induction step, if the clause used at the root of T is not the replacing clause C', then the proof follows from induction hypothesis. Let the clause used at the root of T be a ground instance of C' and let the ground instance used be $A\theta{:-}\ A_1\theta, \ldots A_k\theta, G'\theta$. Then, $A_1\theta,\ldots,A_k\theta, G'\theta$ have ground proofs $T'_1, \ldots, T'_k, T'_{G'\theta}$ in P_i by induction hypothesis. Then, by rule 5, there exists a ground proof $T'_{G\theta}$ of $G\theta$ in P_i. Now T', the ground proof of $A\theta$ in P_i, is constructed with the ground clause $A\theta{:-}\ A_1\theta, \ldots, A_k\theta, G\theta$ at the root and $T'_1, \ldots, T'_k, T'_{G\theta}$ as its children. □

However, if goal replacement is applied to a measure consistent program P_i it is totally correct. But then we also need to ensure that the resulting program P_{i+1} is measure consistent. If this is ensured, then even if goal replacement is interleaved with irreversible folding total correctness will be preserved. Formally,

RULE 6. **Measure Preserving Goal Replacement** Let program P_{i+1} is obtained from program P_i by applying the goal replacement transformation as described in Rule 5. We say that such a goal replacement is *measure preserving* if there exists $\delta, \delta' \in \mathcal{M}$ (where measure structure is $\mu = \langle \mathcal{M}, \oplus, \prec, \mathcal{W} \rangle$) such that for all ground instantiation θ of G, G':

(i) $\delta \preceq \alpha(G\theta) \ominus \alpha(G'\theta) \preceq \delta'$

(ii) $\gamma^i_{lo}(C) \oplus \delta \oplus \sum_{1 \le p \le k} \alpha_{min}(A_p) \succ \mathbf{0}$.

and further the associated clause measures satisfy,

$$\gamma^{i+1}_{lo}(C') \preceq \gamma^i_{lo}(C) \oplus \delta \tag{6}$$
$$\gamma^{i+1}_{hi}(C') \succeq \gamma^i_{hi}(C) \oplus \delta' \tag{7}$$

The clause measures of the other clauses of P_{i+1} are inherited from P_i. □

We now present a formal proof of total correctness and preservation of measure consistency of the above rule.

THEOREM 4. *Let P_{i+1} be derived from P_i by applying measure preserving goal replacement as described in rule 6. If P_i is measure consistent, then $M(P_i) = M(P_{i+1})$ and P_{i+1} is also measure consistent.*

PROOF. Since measure preserving goal replacement is a special case of the goal replacement transformation in rule 5, we have $M(P_{i+1}) \subseteq M(P_i)$ by partial correctness of rule 5. Therefore it is sufficient to prove that:

(1) All ground proofs of P_{i+1} are weakly measure consistent
(2) $M(P_i) \subseteq M(P_{i+1})$
(3) $\forall B \in M(P_{i+1})$ there exists a strongly measure consistent proof of B in P_{i+1}.

We prove the obligation (1) separately. Proof obligations (2) and (3) are proved by showing that: $\forall B \in M(P_i)$ there exists a strongly measure consistent proof of B in P_{i+1}. This is sufficient since we know $M(P_{i+1}) \subseteq M(P_i)$.

First, we prove that all ground proofs of P_{i+1} are weakly measure consistent. The proof proceeds by induction on the size of ground proofs in P_{i+1}. Let T be a ground proof of a ground atom B in P_{i+1}. If the clause used at the root of T is not the new clause C', then the proof follows by induction hypothesis and the measure consistency of P_i. If the clause used at the root of T is C', then let the ground instance of C' used at the root of T be $A\theta{:-}\ A_1\theta, \ldots, A_k\theta, G'\theta$. By induction hypothesis, the proofs of $A_1\theta, \ldots, A_k\theta, G'\theta$ in T are weakly measure consistent. It suffices to show that

$$\alpha(A) \preceq \gamma_{hi}^{i+1}(C') \oplus \sum_{1 \le l \le k} \alpha(A_l\theta) \oplus \alpha(G'\theta)$$

Now, $G'\theta \in M(P_{i+1}) \Rightarrow G'\theta \in M(P_i)$. Hence by rule 5 we have $G\theta \in M(P_i)$. Also, $\forall 1 \le l \le k.\ A_l\theta \in M(P_i)$ (as $M(P_{i+1} \subseteq M(P_i))$. Then, $A\theta{:-}\ A_1\theta, \ldots A_k\theta, G\theta$ is a ground instantiation of C which appears at the root of some ground proof in P_i. Since P_i is measure consistent we have

$$\begin{aligned} \alpha(A) &\preceq \gamma_{hi}^{i}(C) \oplus \textstyle\sum_{1 \le l \le k} \alpha(A_l\theta) \oplus \alpha(G\theta) \\ &\preceq \gamma_{hi}^{i}(C) \oplus \textstyle\sum_{1 \le l \le k} \alpha(A_l\theta) \oplus (\ \alpha(G'\theta) \oplus \delta'\ ) \\ &\preceq \gamma_{hi}^{i+1}(C') \oplus \textstyle\sum_{1 \le l \le k} \alpha(A_l\theta) \oplus \alpha(G'\theta) \end{aligned}$$

Now, we prove that $\forall B \in M(P_i)$ there is a strongly measure consistent proof of B in P_{i+1}. Since P_i is measure consistent, it suffices to translate a strongly measure consistent proof T of B in P_i to a strongly measure consistent proof T' of B in P_{i+1} for all $B \in M(P_i)$. We do this translation by induction on the atom measures. If the clause used at the root of T is not C (where C is the clause in P_i that is replaced) then the proof follows from the definition of strong measure consistency and induction hypothesis. Let C be the clause used at the root of T (a strongly measure consistent proof of A in P_i) and let $A\theta{:-}\ A_1\theta, \ldots, A_k\theta, G\theta$ be the ground instance of C used. Then, by strong measure consistency of T, $\alpha(A_l\theta) \prec \alpha(A\theta)$ for all $1 \le l \le k$. By induction hypothesis, we then have strongly measure consistent ground proofs $T'_1, \ldots, T'_k$ of $A_1\theta, \ldots, A_k\theta$ in P_{i+1}. Also, by strong measure consistency of T

$$\begin{aligned} \alpha(A) &\succeq \gamma_{lo}^{i}(C) \oplus \sum_{1 \le l \le k} \alpha(A_l\theta) \oplus \alpha(G\theta) \\ &\succeq \gamma_{lo}^{i}(C) \oplus \sum_{1 \le l \le k} \alpha(A_l\theta) \oplus (\ \alpha(G'\theta) \oplus \delta\ ) \qquad (8) \\ &\succeq (\ \gamma_{lo}^{i}(C) \oplus \sum_{1 \le l \le k} \alpha_{min}(A_l\theta) \oplus \delta\ ) \oplus \alpha(G'\theta) \\ &\succ \alpha(G'\theta) \text{ (By condition (ii) of rule 6)} \end{aligned}$$

Then, by induction hypothesis, $G'\theta$ has a proof $T'_{G'\theta}$ in P_{i+1}. The ground proof T' is constructed with $A\theta{:-}\ A_1\theta, \ldots, A_k\theta, G'\theta$ at the root (this is a ground instance of C', the new clause in P_{i+1}) and $T'_1, \ldots, T'_k, T'_{G'\theta}$ as its children. To show that this proof T' is measure consistent, note that $\gamma_{lo}^{i+1}(C') \preceq \gamma_{lo}^{i}(C) \oplus \delta$. Combining this with inequation (8), we get

$$\alpha(A) \succeq \gamma_{lo}^{i+1}(C') \oplus \sum_{1 \le l \le k} \alpha(A_l\theta) \oplus \alpha(G'\theta)$$

This completes the proof. □

Observe that, similar to the goal replacement transformation in [Kanamori and Fujita 1987; Tamaki and Sato 1984; 1986a] the conditions under which rule 6 may be applied are not testable at transformation time. For testability we need to (1) determine whether G and G' are semantically equivalent, and (2) estimate δ and δ' such that the clause measures of P_{i+1} can be computed. We have developed a testable goal replacement rule called Syntactic Goal Replacement. A description of this rule will appear in Section 5.

4. CONSTRUCTING CONCRETE UNFOLD/FOLD SYSTEMS BY INSTANTIATING THE FRAMEWORK

To construct a concrete unfold/fold transformation system from our abstract framework, the following parameters need to be instantiated :

(1) a measure structure μ;

(2) atom measure α and α_{min};

(3) clause measure $(\gamma_{lo}, \gamma_{hi})$ for clauses in the initial program P_0 such that P_0 is measure consistent; and

(4) functions to compute the clause measure of new clauses obtained by the transformations such that they satisfy the constraints imposed by equations (1) through (4) (refer Rules 3 and 4).

There are *no further* proof obligations. Once the above four elements are defined, total correctness of the transformation system is *guaranteed* by the framework. We now instantiate our farmework to obtain some existing transformation systems. *Note that the instantiations given below consider all three rules (unfolding, folding and goal replacement) of these existing transformation systems.*

4.1 Existing Unfold/fold Systems

We now show how our framework can be instantiated to obtain the Kanamori-Fujita and the extended Tamaki-Sato systems. To the best of our knowledge, these are the only two existing systems that allow folding using recursive clauses. However in both of these systems folding is conjunctive.

The Kanamori-Fujita System [1987]. This system can be obtained as an instance of our framework as follows:

(1) $\mu = \langle \mathbb{Z}, +, <, \mathbb{N} \rangle$. This measure structure corresponds to the use of integer counters in [Kanamori and Fujita 1987].

(2) $\alpha(A)$ = number of nodes in the smallest proof of A in P_0, and for any atom A, $\alpha_{min}(A) = 1$. Thus, $\alpha(A)$ denotes the *rank* of A described in [Kanamori and Fujita 1987].

(3) $\forall C \in P_0\ \gamma^0_{lo}(C) = \gamma^0_{hi}(C) = 1$. Since all clause measures are 1, it follows immediately from the definition of atom measures that the smallest proofs of any ground goal G are strongly measure consistent, and all proofs in P_0 are weakly measure consistent. Hence P_0 is measure consistent.

(4) $\forall C \in P_{i+1} - P_i$ we have $\gamma^{i+1}_{lo}(C) = \mathit{GLB}^{i+1}(C)$ and $\gamma^{i+1}_{hi}(C) = \mathit{LUB}^{i+1}(C)$. Under the given measure structure, it is immediate that the above definition is identical to the computation on counters in [Kanamori and Fujita 1987].

Furthermore, the measure preserving folding rule (Rule 4) is applied only when both folder and folded clauses are singleton sets. It is easy to see a one-to-one correspondence between the conditions on unfold/fold transformations of the above instantiation and the Kanamori-Fujita system.

The Extended Tamaki-Sato System [1986a]. In this system, all the predicate symbols are partitioned into n strata. In the initial program a predicate from stratum j is defined using only predicates from strata $\leq j$. We can obtain this system as an instance of our framework as follows:

(1) $\mu = \langle \mathbb{Z}^n, \oplus, \prec, \mathbb{N}^n \rangle$ where $\oplus$ denotes coordinate-wise integer addition of n-tuples of integers, and $\prec$ denotes the lexicographic $<$ order over n-tuples of integers. The n-tuples in the measure structure will correspond to the n strata of the original program.

(2) $\alpha(A) = \min(\{w(T) \mid T \text{ is a proof of } A \text{ in } P_0\})$, where $w(T)$ is the *weight* of the proof T defined as an n-tuple $\langle w_1, \ldots, w_n \rangle$ such that $\forall 1 \leq j \leq n$, w_j is the number of nodes of predicates from stratum j in T. $\alpha(A)$ corresponds to the notion of *weight-tuple measure* of A defined in [Tamaki and Sato 1986a].
For any atom A, $\alpha_{min}(A) = \mathbf{0} = \langle 0, \ldots, 0 \rangle$.

(3) $\forall C \in P_0$, $\gamma^0_{lo}(C) = \gamma^0_{hi}(C) = \langle w_1, \ldots, w_n \rangle$, where $C \equiv A\text{:-}\ A_1, \ldots, A_n$ and for $1 \leq j \leq n$, $w_j = 1$ if the predicate symbol of A is from stratum j, and 0 otherwise.
For any $A \in M(P_0)$, the proof T that defines $\alpha(A)$ (item 2 above) is strongly measure consistent. Weak measure consistency of ground proofs in P_0 is established by induction on their size.

(4) $\forall C \in P_{i+1} - P_i$, $\gamma^{i+1}_{hi}(C) = \mathit{LUB}^{i+1}(C)$ and $\gamma^{i+1}_{lo}(C) = \mathit{approx}(\mathit{GLB}^{i+1}(C))$. The function *approx* reduces a measure as follows. Let $u = \langle u_1, \ldots, u_n \rangle$ and k_{min} be the smallest index k such that $u_k > 0$. Then $\mathit{approx}(u) = \langle u'_1, \ldots, u'_n \rangle$ where $u'_{k_{min}} = 1$ and is 0 elsewhere.

As in the Kanamori-Fujita system, here also the measure preserving folding rule is applied only when both folder and folded clauses are singleton sets.

To establish the correspondence between the above instantiation and the extended Tamaki-Sato system, recall that the latter associates a descent level with each clause of every program in a transformation sequence. If a clause C in P_i has the descent level k, then with the above instantiation, $\gamma^i_{lo}(C) = \langle l_1, \ldots, l_n \rangle$ where $l_k = 1$ and 0 elsewhere; i.e. the only non-zero entry in its lower clause measure appears in the k^{th} position. Thus our lower clause measure precisely captures the information that is kept track of by the extended Tamaki-Sato system.

Assigning Measure Structures and Clause Measures. Observe that our framework does not prescribe exact values to the clause measures. Instead it bounds the clause measures from above and below. So an important aspect of our instantiation involves assigning values to the clause measures that satisfy these constraints. From an abstract point of view, the Kanamori-Fujita system uses a relatively coarse measure space ($\mathbb{Z}$) but within this space it maintains accurate clause measures (integer counters). Our instantiation reflects this by not relaxing the bounds while updating the clause measures (see step 4 of the instantiation). On the other hand, the extended Tamaki-Sato system uses a more fine-grained measure space ($\mathbb{Z}^n$). But this measure space is not completely utilized since clause measures are the descent level of clauses, which can be simply represented by an integer. Therefore in step 4 of our instantiation we accordingly loosened the bound.

As far as the Gergatsoulis-Katzouraki [1994] and original Tamaki-Sato systems [1984] are concerned, first note that they do not permit folding using recursive clauses. The main difference between these two systems is that [1994] allows disjunctive folding (folding where multiple clauses are replaced by one clause) whereas [1984] does not. However the book-keeping performed (clause measures) in these two systems is not different. These systems use coarse measure spaces. Moreover they do not even fully utilize these measure spaces as is evident from the lesser amount of book keeping performed by them. By choosing a coarse measure structure and relaxing the bounds along lines similar to the extended Tamaki-Sato system we can instantiate these two systems as well. Both these systems partition the program predicates into two strata, the so-called "old" and "new" predicates. Therefore, we set the measure structure to be $\mu = \langle \mathbb{Z}^2, \oplus_2, \prec_2, \mathbb{N}^2 \rangle$ where $\oplus_2$ denotes coordinate-wise integer addition of 2-tuples of integers, and $\prec_2$ denotes the lexicographic $<$ order over 2-tuples of integers. Since these systems partition the predicate symbols into "old" and "new" predicates, the choice of a measure structure with two strata is obvious.

4.2 SCOUT— A New Unfold/fold System

We now construct SCOUT, an unfold/fold transformation system for definite logic programs that allows disjunctive folding using recursive clauses. It incorporates the notion of strata from the extended Tamaki-Sato system into the counters of the Kanamori-Fujita system. Thus with every clause it maintains a *pair* of stratified counters as the clause measure. The instantiation is as follows. We assume that the predicate symbols appearing in the initial program P_0 are partitioned into n strata, as in the extended Tamaki-Sato system.

(1) $\mu = \langle \mathbb{Z}^n, \oplus, \prec, \mathbb{N}^n \rangle$ where $\oplus$ denotes coordinate-wise integer addition of n-tuples of integers, and $\prec$ denotes the lexicographic $<$ order over n-tuples of integers.

(2) $\alpha(A)$ is defined exactly as in the instantiation of the extended Tamaki-Sato system above. For any atom A we set $\alpha_{min}(A) = \langle w_1, \ldots, w_n \rangle$ where $w_j = 1$ if A is from stratum j and 0 elsewhere.

(3) Clause measure of clauses in P_0 is defined exactly as in the instantiation of the extended Tamaki-Sato system above. Therefore the proofs of measure consistency are also identical.

(4) $\forall C \in P_{i+1} - P_i$, $\gamma_{lo}^{i+1}(C) = GLB^{i+1}(C)$ and $\gamma_{hi}^{i+1}(C) = LUB^{i+1}(C)$.

SCOUT provides a solution to two important (and orthogonal) problems that have thus far remained open. First, it allows folding using clauses that have disjunctions as well as recursion. Secondly, SCOUT combines the stratification-based (extended) Tamaki-Sato system with the counter-based Kanamori-Fujita system thereby obtaining a single system that strictly subsumes either of them even when restricted to conjunctive folding. A formal proof of this claim appears in the appendix. Note that we prove that *any transformation sequence made out of unfold/fold/goal replacement rules which is allowed by the existing transformation systems is also allowed by SCOUT.*

It is interesting to note that by simple inspection of the instantiations, one can see that when the number of strata is 1 and only conjunctive folding is permitted, SCOUT collapses to the Kanamori-Fujita system. Collapsing SCOUT to other existing unfold/fold systems by varying the number of strata and extending the parameters (e.g. measure structure) remains an interesting open problem.

5. EMPLOYING TRANSFORMATIONS TO CONSTRUCT PROOFS

Our motivation in developing the transformation framework presented in this paper lies in its application to deduction *i.e.* constructing proofs. Thus, we ensure that our transformations preserve correctness w.r.t the model theoretic semantics of definite logic programs: the least Herbrand model semantics. Our transformation framework does not consider other operational aspects of the program, such as preserving termination properties (studied in [Amtoft 1992; Bossi and Cocco 1994]) and preserving computed answer substitutions ([Kawamura and Kanamori 1990] is an early reference on this subject).

Unfold/fold transformations have been used for inductive reasoning in the past [Hsiang and Srivas 1987; Kanamori and Fujita 1986; Pettorossi and Proietti 1999]. Since unfolding represents a resolution step, it can be used to prove the base case and finite part of the induction step. Folding can be used to remember the induction hypothesis and recognize its occurrence. We have used the SCOUT transformation system (developed in the last section) to construct inductive proofs of temporal properties of concurrent systems [Roychoudhury et al. 2000; Roychoudhury and Ramakrishnan 2001]. In this section, we present the key issues in using the SCOUT transformation system for deduction. We also present an example to show how additional power of our transformations (such as our more general folding rule) can be exploited for constructing proofs. A full-fledged discussion on the use of our transformations for concurrent system verification appears in [Roychoudhury 2000].

5.1 Automation of the Goal Replacement rule

In order to use SCOUT for automated deduction, a second look at the goal replacement transformation is necessary. Goal replacement, where semantically equivalent goals are interchanged, creates more opportunities for folding. There are two immediate problems with integrating goal replacement in an automated proof system. First, the identification of equivalent goals must be based on some syntactic (or analysis-based) criteria, since semantic equivalence is, in general, undecidable. Secondly, the conditions under which goal replacement is permitted by the transformation system are usually specified in terms of uncomputable measures such as the atom measure α. Recall that in the SCOUT system the atom measure $\alpha(A)$ is related to the "shortest" ground proof of atom A.

Thus, we need a nontrivial computational mechanism to check the semantic equivalence of two given atoms purely based on syntax. We must also identify testable conditions that imply the untestable restrictions on weights of atoms required by the general goal replacement rule. The notion of syntactic equivalence described below addresses the first issue, while the definition of the syntactic goal replacement rule resolves the second issue.

Syntactic Equivalence. Consider the following example program P

```
p(X) :- r(X).              q(X) :- s(X).
p(X) :- e(X,Y), p(Y).      q(X) :- e(X,Y), q(Y).
r(X) :- b(X).              s(X) :- b(X).
```

Fig. 3. A program fragment to illustrate syntactic equivalence

`r(X)` and `s(X)` are equivalent since the clauses defining them have identical right hand sides. We can now use this to infer that `q(X)` and `p(X)` are equivalent. Note that even though the clauses of `p(X)` and `q(X)` are not syntactically identical, the "recursive structure" of these clauses is the same. We formalize this notion in the definition given below.

Definition 9. **Syntactic Equivalence of Atoms** *Let $\cong^P$ be an equivalence relation on the set of predicates of a program P and let $A = p(t_1, \ldots, t_k)$ and $B = q(t'_1, \ldots, t'_k)$ be two atoms. Then atoms A and B are said to be syntactically equivalent w.r.t. to the relation $\cong^P$, denoted $A \cong^P_{atom} B$, if we have $p \cong^P q$ and $(t_1, \ldots, t_k)$ is a variant of $(t'_1, \ldots, t'_k)$*

Definition 10. **Syntactic Equivalence of Predicates** *An equivalence relation $\cong^P$ on the set of predicates of P is said to be a syntactic equivalence relation if whenever $p \cong^P q$ we have:*
1. The predicates p and q belong to the same stratum.
2. Let the clauses of p and q in program P be $\{C_1, \ldots, C_m\}$ and $\{D_1, \ldots, D_m\}$ respectively. Then, for all $1 \leq i \leq m$ we have :
(i) *C_i is a variant of D_i when all predicate symbols in C_i and D_i are replaced with the same predicate.*
(ii) *Let C_i and D_i be of the form $H{:-}\ B_1, ..., B_k$ and $H'{:-}\ B'_1, ..., B'_k$ respectively. Then for all $1 \leq l \leq k$ $B_l \cong^P_{atom} B'_l$.*

It is easy to see that the family of syntactic equivalence relations is closed under union. Thus there is a largest syntactic equivalence relation $\equiv^P$. The relation $\equiv^P$ can be computed by starting with all predicates in the same class, and repeatedly splitting the classes that violate properties (1) and (2) until a fixed point is reached. In the example program fragment P given in Figure 3, the largest syntactic equivalence relation $\equiv^P$ is $\{(\mathtt{p}, \mathtt{q}), (\mathtt{r}, \mathtt{s})\} \cup Id$, where Id is the identity relation. Therefore, $\mathtt{p(X)} \equiv^P_{atom} \mathtt{q(X)}$ where for two atoms A and B we say $A \equiv^P_{atom} B$ if and only if $A \cong^P_{atom} B$ for some syntactic equivalence relation $\cong^P$.

We show that all syntactically equivalent atoms are semantically equivalent.

Lemma 2. *Let $\cong^P$ be a syntactic equivalence relation of the predicates of a program P. For all predicates p, q, if $p \cong^P q$, then p and q are semantically equivalent in program P.*

Proof : Let $p \cong^P q$. We show that for any ground proof T of a ground atom $p(\overline{X})\theta$ in program P there is a ground proof T' of $q(\overline{X})\theta$ in program P and vice-versa.

For any ground proof T of $p(\overline{X})\theta$ we can show the existence of a ground proof T' of $q(\overline{X})\theta$ by induction on the size (number of nodes) of T. Let the clause used at the root of T be $C = (p(\ldots) \text{:-}\ B_1, \ldots, B_k)$. Since $p \cong^P q$ therefore q has a clause $C' = (q(\ldots) \text{:-}\ B'_1, \ldots, B'_k)$ and $p_l \cong^P p'_l$ where p_l (p'_l) is the predicate symbol in B_l (B'_l) for all $1 \leq l \leq k$. Let $p_l(\overline{Y})\sigma$ be the ground instantiation of B_l appearing in T. Now, the size of the subproof of $p_l(\overline{Y})\sigma$ in T is clearly less than the size of T. By induction hypothesis there exists a ground proof of $p'_l(\overline{Y})\sigma$. Also $p'_l(\overline{Y})\sigma$ is an instance of $B'_l\theta$ since clause C is an instance of clause C' when all predicates are replaced by their labels. By applying clause C' at the root we can construct a ground proof T' of $q(\overline{X})\theta$.

For any ground proof T' of $q(\overline{X})\theta$ we can show the existence of a ground proof T of $p(\overline{X})\theta$ in a similar fashion. $\square$

Note that we can straightforwardly *generalize* our definition of syntactic equivalence to define syntactic equivalence of subgoals. Thus, we can then make inferences like `p(f(X))` $\equiv$ `q(X)` based on the syntax[4]. We now introduce the notion of *relevant clause set* of an atom. Intuitively, it is a conservative estimate (*i.e.* a superset) of the set of clauses which are used in the proof of some ground instance of the atom.

Definition Relevant Clause Set. *Let A be an atom and P a program. Let $reach(A, P)$ denote the set of predicates which are reachable from the predicate of A in the predicate dependency graph[5] of P. Then, the relevant clause set of A in P (denoted $rel(A, P)$) is the set of clauses of the predicates in $reach(A, P)$.*

We now define the Syntactic Goal Replacement rule. For any clause C, $hd(C)$ denotes the head atom of C. Our definition is adapted to the SCOUT transformation system described in Section 4.2. Recall that the SCOUT system is an instance of our transformation framework where the predicate symbols appearing in the initial program P_0 are partitioned into n strata. Furthermore, for any atom A, the SCOUT system defines $\alpha_{min}(A) = \langle w_1, \ldots, w_n \rangle$ where $w_j = 1$ if A is from stratum j and 0 elsewhere.

Rule 7. **Syntactic Goal Replacement** Let C be a clause in P_i of the form:

$$C \equiv A \text{:-}\ A_1, ..., A_k, G$$

and consider another clause C' (not in P_i) of the form :

$$C' \equiv A \text{:-}\ A_1, ..., A_k, G'$$

such that

1. G and G' are syntactically equivalent *i.e.* $G \equiv^{P_i}_{atom} G'$, and $vars(G) =$

[4]With definitions 9, 10 we can only infer `p(f(X))` $\equiv$ `q(f(X))` if `p` $\equiv$ `q`.

[5]The predicate dependency graph of a program P has the predicate symbols of P as its vertices, and there is an edge from predicate p to predicate q if q occurs in the body of a clause of p in program P.

$vars(G') \subseteq vars(A, A_1, \ldots, A_k)$
2. The clauses in $rel(G', P_0)$ are never modified in the transformation sequence $P_0, P_1, \ldots, P_i$ *i.e.* $rel(G', P_0) = rel(G', P_i)$.
3. For each clause $D \in rel(G, P_i)$ $\gamma^i_{lo}(D) \geq \alpha_{min}(hd(D))$.
4. Let $Cl_i(G)$ be the clauses in P_i whose heads unify with the atom G. We define:

$$\delta \;=\; min_{D \in Cl_i(G)} \; (\gamma^i_{lo}(D) - \alpha_{min}(hd(D)))$$

We must have: $\gamma^i_{lo}(C) + \delta + \sum_{1 \leq j \leq k} \alpha_{min}(A_j) > \mathbf{0} = \langle 0, \ldots, 0 \rangle$
Then, assign $P_{i+1} := (P_i - \{C\}) \cup \{C'\}$ where C' is A:– $A_1, ..., A_k, G'$. Also, set $\gamma^{i+1}_{lo}(C') = \gamma^i_{lo}(C) + \delta$ and $\gamma^{i+1}_{hi}(C') = \gamma^i_{hi}(C) + \delta'$ where $\delta' = \langle \infty, 0, \ldots, 0 \rangle$. [6] □

Syntactic Goal Replacement can be proved to be a special case of the Goal Replacement transformation of the SCOUT system. First we define the notion of "weight of a ground proof" and use it to prove a property about syntactically equivalent atoms.

Definition Weight of a Ground Proof. *Let T be a ground proof of a ground atom $A \in M(P)$ for a program P. We assume that the predicate symbols of P are a-priori partitioned in n strata. Then the weight of T (denoted $w(T)$) is the the n-tuple $\langle w_1, \ldots, w_n \rangle$ where for all $1 \leq i \leq n$, w_i is the number of nodes of T whose predicate symbol is assigned to strata i.*

The following holds for syntactically equivalent atoms. Note that this is a stronger claim than Lemma 2.

Lemma 3. *Let P be a program and G, G' be atoms such that $G \equiv^P_{atom} G'$* i.e. *G and G' are syntactically equivalent in P. For any ground proof T of a ground instantiation $G\theta$ there exists a ground proof T' of $G'\theta$ such that $w(T) = w(T')$, and vice-versa.*

Proof. The proof proceeds by induction on the size of T, as in Lemma 2. □

We will now use the above lemma to prove that Syntactic Goal Replacement is a special case of the Measure preserving Goal Replacement rule.

Lemma 4. **Special Case of Goal Replacement** *Let $P_0 \rightarrow \ldots \rightarrow P_i$ be a sequence of measure consistent programs. Then, any syntactic goal replacement transformation applicable in P_i is also a Measure preserving goal replacement transformation (transformation 6) applicable in P_i.*

Proof. For any ground instantiation θ of G and G' (recall $vars(G) = vars(G')$), by using lemma 2, we have $P_i \vdash G\theta \Leftrightarrow P_i \vdash G'\theta$. To prove that the other conditions of Measure preserving goal replacement transformation are also true whenever syntactic goal replacement is applicable, we now just need to show that the inequalities in conditions *(i)* and *(ii)* of rule 6) are satisfied whenever syntactic goal replacement is applicable. We have $\gamma + \delta + \sum_{1 \leq j \leq k} \alpha_{min}(A_j) > \mathbf{0} = \langle 0, \ldots, 0 \rangle$. We also need to show that whenever Syntactic Goal Replacement is applied to

[6]Note that ∞ is only a notational convenience. It represents a value that exceeds the weights of all atoms. Formally, this can be achieved by extending the clause annotations by one extra stratum.

clause C to replace G by G' we have $\forall\theta\ \delta \leq \alpha(G\theta) - \alpha(G'\theta) \leq \delta'$. Since $\delta' = \langle\infty, 0, \ldots, 0\rangle$, therefore δ' is lexicographically greater than the weight of any ground atom; hence $\alpha(G\theta) - \alpha(G'\theta) \leq \delta'$. We now prove that $\delta \leq \alpha(G\theta) - \alpha(G'\theta)$ where $\delta = min_{D\in Cl_i(G)}\ (\gamma^i_{lo}(D) - \alpha_{min}(hd(D)))$ and $Cl_i(G)$ are the clause in P_i whose heads unify with G.

Since P_i is measure consistent, therefore if $G\theta \in M(P_i)$, then $G\theta$ has a strongly measure consistent proof T in P_i. Hence, by lemma 3, $G'\theta$ has a proof T' in P_i such that $w(T) = w(T')$. Let C_{root} be the clause used the root of T. Clearly $C_{root} \in Cl_i(G)$. Let the ground instance of C_{root} used at the root of ground proof T be $G\theta : -B_1, \ldots, B_m$. Then :

$$\alpha(G\theta) \geq \alpha_{min}(G\theta) + (\gamma^i_{lo}(C_{root}) - \alpha_{min}(G\theta)) + \sum_{1\leq j\leq m} \alpha(B_j)$$

Since each of the body atoms B_j also have strongly measure consistent proofs as subproofs of T, we can again use this condition to expand out the $\alpha(B_j)$ in the above inequality. Continuing in this way until we reach the leaves of T, we get the following inequality (where $hd(C)$ denotes the head of clause C).

$$\alpha(G\theta) \geq \sum_{C\ used\ in\ T} \alpha_{min}(hd(C)) + \sum_{C\ used\ in\ T} (\gamma^i_{lo}(C) - \alpha_{min}(hd(C)))$$

According to the definition of α_{min} in the SCOUT system

$$\sum_{C\ used\ in\ T} \alpha_{min}(hd(C)) = w(T)$$

The above inequality follows from that fact that each of the nodes of T are the head of some clause C used in T. Thus, we have:

$$\alpha(G\theta) \geq w(T) + \sum_{C\ used\ in\ T} (\gamma^i_{lo}(C) - \alpha_{min}(hd(C)))$$

Now, since $\forall C \in rel(G, P_i)$, we have $(\gamma^i_{lo}(C) - \alpha_{min}(hd(C)))$ to be non-negative:

$$\sum_{C\ used\ in\ T} (\gamma^i_{lo}(C) - \alpha_{min}(hd(C))) \ \geq\ (\gamma^i_{lo}(C_{root}) - \alpha_{min}(hd(C_{root}))) \geq\ \delta$$

Note that $\gamma^i_{lo}(C_{root}) - \alpha_{min}(hd(C_{root})) \geq\ \delta$ since $C_{root} \in Cl_i(G)$. Thus,

$$\begin{aligned} \alpha(G\theta) &\geq w(T) + \delta \\ &\geq w(T') + \delta \quad \text{(by Lemma 3)} \\ &\geq \text{Measure of lexicographically shortest proof of } G'\theta \text{ in } P_i \\ &= \alpha(G'\theta) \quad (\text{since } rel(G', P_0) = rel(G', P_i)) \end{aligned}$$

Hence, $\alpha(G\theta) \geq \alpha(G'\theta) + \delta$ for any ground substitution θ of G and G'. This completes the proof. □

Applicability of the Syntactic Goal Replacement rule is testable and the clause annotations of the new clause C' can be effectively computed since we have conservatively estimated the value of δ, δ'. Note that in the Syntactic Goal Replacement rule, we have set δ' to $\langle\infty, 0, \ldots, 0\rangle$. This will prevent the new clause C' from being used as a folder later in the transformation sequence. However, our choice of δ

satisfies $\delta \geq \langle 0, \ldots, 0 \rangle$ and therefore we will always have $\gamma_{lo}^{i+1}(C') \geq \gamma_{lo}^{i}(C)$. Thus, C' can participate in future folding transformations as one of the folded clauses. Also, note that a tighter value of δ' is hard to obtain. This is because we need to satisfy $\alpha(G\theta) - \alpha(G'\theta) \leq \delta'$ for any ground substitution θ. The proof sizes of $G\theta$ and $G'\theta$ could be monotonic on the instantiation of some variable of G, G' and θ could be constructed to instantiate that variable to larger and larger ground terms, thereby ruling out a tighter value of δ'.

Other standard transformations. In addition to unfolding/folding/goal replacement, a number of other standard transformations, such as deletion of subsumed clauses, deletion of duplicate goals [Pettorossi and Proietti 1998] can be readily adapted to the SCOUT system. These transformations can also be useful for constructing proofs.

Also, note that we do not explicitly consider a *Definition Introduction* transformation which allows us to define new predicates in terms of old predicates. This is because new predicates introduced in the course of constructing a transformation sequence $P_0, \ldots, P_n$ can be assumed to be present in the initial program [Tamaki and Sato 1984].

5.2 On the utility of Stratification

The SCOUT transformation system allows the predicate symbols of the initial program P_0 to be a-priori partitioned into $n \geq 1$ strata. This may give us additional flexibility in constructing a totally correct transformation sequence without exactly computing the clause annotations. To illustrate this point, consider the following goal replacement step $P_i \rightarrow P_{i+1}$. The predicates are partitioned into 2 strata : `p` is placed in the upper strata, and `q`,`r` are placed in the lower strata.

`p :- q.`	$(\langle 1,0\rangle, \langle 1,0\rangle)$	`p :- r.`	$(\langle 1,\_\rangle, \langle 1,\_\rangle)$
`q.`	$(\langle 0,1\rangle, \langle 0,1\rangle)$	`q.`	$(\langle 0,1\rangle, \langle 0,1\rangle)$
`r.`	$(\langle 0,1\rangle, \langle 0,1\rangle)$	`r.`	$(\langle 0,1\rangle, \langle 0,1\rangle)$
Program P_i		Program P_{i+1}	

Recall that the stratification of predicates is such that a predicate of stratum j is defined using predicates of stratum $\leq j$ in P_0. Therefore, in the above example since `q`,`r` are placed in the lower stratum we can conclude that $\alpha(\mathtt{q}) = \langle 0, \_\rangle$ and $\alpha(\mathtt{r}) = \langle 0, \_\rangle$. Thus, the annotations of the replaced clause `p :- r` are guaranteed to be of the form $\langle 1, \_\rangle$ irrespective of the exact value of $\alpha(\mathtt{q}) - \alpha(\mathtt{r})$.

The above observation could be successfully exploited while constructing a transformation sequence as follows. Recall that the Goal Replacement rule does not prescribe exact values of δ, δ' and hence its application is not automated.[7] Consider a goal replacement step $P_i \rightarrow P_{i+1}$ in which G is replaced by G' in clause $C \equiv A{:-}\, A_1, \ldots, A_k, G$. We can avoid computing $\alpha(G) - \alpha(G')$ and annotate the new clause $C' \equiv A{:-}\, A_1, \ldots, A_k, G'$ with only *approximate* annotations if the

[7]This problem is partially remedied in the Syntactic Goal Replacement rule which tells us how to compute δ, δ' provided certain extra conditions (such as conditions 2,3 of the Syntactic Goal Replacement rule) are satisfied.

predicates are partitioned into > 1 strata. In the above example, we observed that

$$\langle 0, -\infty \rangle \ \leq \alpha(\mathtt{q}) - \alpha(\mathtt{r}) \leq \ \langle 0, \infty \rangle$$

We then used these inequalities to compute the approximate annotations of the replaced clause p :− r as $(\langle 1, \_ \rangle, \langle 1, \_ \rangle)$ In this example, if all the predicates are placed in only one stratum, we must compute $\alpha(G) - \alpha(G')$. Otherwise, we will annotate the new clause C' with the counters $(-\infty, \infty)$. Clearly this will forbid C' from participating in *any* future folding step.

5.3 An Example Induction Proof

We now illustrate by an example how our program transformation rules can be used for constructing induction proofs. Consider the program P_0 given below. Any string consisting of only 0's is generated by gen while the test predicate checks whether a given list can be transformed (through finite number of applications of trans) into a string consisting of only 1's. The trans predicate transforms a string by converting the leftmost occurrence of 0 in the string to 1. The property that we would like to establish is $\forall$ X gen(X) $\Rightarrow$ test(X). A hand proof of this property will proceed by induction on the length of the strings generated by gen.

In P_0, all predicate symbols are assumed to be in the same stratum and the lower and upper clause measures are set to 1 for all clauses. In the following, the clause annotations $(\gamma^i_{lo}(C), \gamma^i_{hi}(C))$ for any clause $C \in P_i$ are shown in parentheses beside clause C.

```
thm(X) :- gen(X), test(X)            (1,1)
gen([]).                             (1,1)
gen([0|X]) :- gen(X).                (1,1)
test(X) :- canon(X).                 (1,1)
test(X) :- trans(X,Y), test(Y).      (1,1)
canon([]).                           (1,1)
canon([1|X]) :- canon(X).            (1,1)
trans([0|X], [1|X]).                 (1,1)
trans([1|T],[1|T1]) :- trans(T,T1).  (1,1)
```

Now, from the definition of thm in P_0 we see that $\forall$ X thm(X) $\Leftrightarrow$ gen(X) $\wedge$ test(X). Thus, if we can establish that $\forall$ X thm(X) $\Leftrightarrow$ gen(X) then we can conclude that the formula $\forall$ X gen(X) $\Rightarrow$ test(X) is true. One technique to establish $\forall$ X thm(X) $\Leftrightarrow$ gen(X) is to show that the above program P_0 is equivalent to some program P_{final} in which the thm(X) and gen(X) are syntactically equivalent.

We now construct such a transformation sequence $P_0, \ldots, P_{final}$. Unfolding the only clause of thm/1 several times we obtain:

```
thm([]).                                        (4,4)
thm([0|X]) :- gen(X), canon(X).                 (6,6)
thm([0|X]) :- gen(X), trans(X,Y), test([1|Y]).  (6,6)
```

We now introduce the definition:

```
test1(Y) :- test([1|Y]).   (1,1)
```

and fold the occurrence of test([1|Y]) in the last clause of thm/1 to obtain:

```
thm([]).                                    (4,4)
thm([0|X]) :- gen(X), canon(X).             (6,6)
thm([0|X]) :- gen(X), trans(X,Y), test1(Y). (5,5)
```

Unfolding the definition clause of `test1/1` several times and then folding (using the definition clause of `test1/1` as the folder) we get:

```
test1(Y) :- canon(Y).              (3,3)
test1(Y) :- trans(Y,Z), test1(Z).  (2,2)
```

Note that `test1(Y)` and `test(Y)` are syntactically equivalent, since the clauses of `test/1` are :

```
test(X) :- canon(X).             (1,1)
test(X) :- trans(X,Y), test(Y).  (1,1)
```

We therefore apply *Syntactic Goal Replacement* in the last clause of `thm/1` to obtain the following :

```
thm([]).                                        (4,4)
thm([0|X]) :- gen(X), canon(X).                 (6,6)
thm([0|X]) :- gen(X), trans(X,Y), test(Y).      (6,∞)
```

We can now fold the above clauses of `thm/1` using the clauses of `test/1` as the folder. Note that we are folding using *multiple recursive clauses as the folder*. The additional power of our folding rule is exploited in this transformation step. We obtain:

```
thm([]).                          (4,4)
thm([0|X]) :- gen(X), test(X).    (5,∞)
```

Finally, we fold using the clause of `thm/1` in P_0 as folder to obtain the program P_{final}

```
thm([]).                                (4,4)
thm([0|X]) :- thm(X).                   (4,∞)
gen([]).                                (1,1)
gen([0|X]) :- gen(X).                   (1,1)
test(X) :- canon(X).                    (1,1)
test(X) :- trans(X,Y), test(Y).         (1,1)
canon([]).                              (1,1)
canon([1|X]) :- canon(X).               (1,1)
trans([0|T],[1|T]).                     (1,1)
trans([1|T], [1|T1]) :- trans(T,T1).    (1,1)
```

The atoms `thm(X)` and `gen(X)` are now syntactically equivalent (refer definition 10). Thus, $M(P_{final}) \models \forall$ `X thm(X)` $\Leftrightarrow$ `gen(X)`. Since $M(P_{final}) = M(P_0)$ therefore $M(P_0) \models \forall$ `X thm(X)` $\Leftrightarrow$ `gen(X)`. By definition of `thm(X)` in P_0, this means that $M(P_0) \models \forall$ `X gen(X)` $\Rightarrow$ `test(X)`. Thus, by using our transformation rules, we have constructed a nontrivial induction proof.

5.4 Verification of Parameterized Concurrent Systems

The above proof is illustrative since it is structurally similar to the proofs that arise in the verification of concurrent systems. Using the transformation rules of SCOUT and the Syntactic Goal Replacement rule in a similar fashion we verified properties like liveness of a m-bit shift register, correctness of a m-bit carry-lookahead adder etc. Thus, in the problem of verification of liveness of a m-bit shift register : the predicate `gen` represents the encoding of the m-bit shift register while the predicate `test` represents the encoding of the liveness property that we verify. To accomplish the proof of liveness for any m, we perform a folding step using `test` as the folder similar to the above example. Moreover, as in the above example, the proof of liveness also involves application of the Syntactic Goal Replacement rule to replace a specialized version of `test`. This corresponds to proving that the liveness property holds in a process Q iff the property holds in a sub-process of Q.

In particular, we have used the transformations developed in this paper to inductively prove temporal properties of *parameterized* concurrent systems [Roychoudhury 2000; Roychoudhury et al. 2000; Roychoudhury and Ramakrishnan 2001]. Verification of distributed algorithms with arbitrary number of constituent processes can be naturally cast as verifying parameterized systems. A parameterized concurrent system (such as a n-bit shift register for arbitrary n) represents an infinite family of finite state concurrent systems, parameterized by a recursively defined type. Therefore, it is natural to prove properties of parameterized concurrent systems by inducting over this type. We have automated the construction of these induction proofs by using the unfold/fold rules developed in this paper along with domain specific control strategies. In our approach, the parameterized system and the temporal property to be verified are encoded as a logic program. The verification problem is reduced to the problem of determining the equivalence of predicates in this program. The predicate equivalences are then established by employing unfold/fold transformations on the predicates. Finally the proof of semantic equivalence of the predicates is achieved by showing syntactic equivalence of their transformed definitions.

The additional power of our transformation rules is useful in our transformation based proofs of temporal properties. Note that temporal properties contain fixed point operators. These properties are typically encoded as a logic program predicate with multiple recursive clauses *e.g.* a least fixed point property containing disjunctions is encoded using multiple recursive clauses. Therefore, one cannot assume restrictions that are imposed by existing transformation systems [Tamaki and Sato 1984; 1986a; Kanamori and Fujita 1987; Gergatsoulis and Katzouraki 1994] on the syntax of clauses encoding a temporal property. As mentioned before, the applicability of our transformation rules is not restricted by program syntax. Instead, book-keeping is performed at every transformation step, and this book-keeping is used to restrict the applicability of the transformation rules. This makes these rules suitable for constructing proofs of temporal properties.

A full-fledged discussion of the use of our transformations for verification needs to discuss transformation strategies as well. This is outside the scope of this paper. The interested reader is referred to [Roychoudhury 2000].

6. RELATED WORK

In this section, we survey related work on unfold/fold transformations and their usage in deduction. In Section 6.1, we discuss work on developing totally correct irreversible unfold/fold transformation systems. In particular, we discuss the restrictions which need to be placed on the folding rule in order to make any interleaving of unfolding/folding preserve semantics. In Section 6.2, we discuss past work on using unfold/fold transformations of logic programs for inductive reasoning. Finally, we conclude by briefly discussing work on other (more traditional) usage of logic program transformations such as: use of transformations for partial deduction, and reversible transformations for deductive databases.

6.1 Restricting Transformations to ensure Total Correctness

Conditions to ensure total correctness of unfold/fold transformations have been extensively studied for logic programs. Most of these transformation rules impose two kinds of restrictions: (a) the syntax of the folder clauses is restricted, *and* (b) clauses are annotated with book-keeping (our clause measures) which is updated in each transformation step; conditions are imposed on this book-keeping to restrict applicable folding steps. In this paper, we have shown that the first kind of restrictions (syntactic restrictions on the folder clauses) are redundant. Only the second kind (restriction on clause measures) is necessary to show $A \succ B$ in any folding step, where A is the head of the clause produced by folding, B is the folder atom (the atom introduced by folding) and $\succ$ is a well-founded order. Preservation of such a well-founded order allows us to prove total correctness by induction.

Syntactic Restrictions on folder clauses. Among the previous works which imposed syntactic restrictions on the folder clauses:

—[Tamaki and Sato 1984; Gergatsoulis and Katzouraki 1994] required the folder clauses to be non-recursive.

—[Tamaki and Sato 1984; 1986a; Kanamori and Fujita 1987] required a single folder clause (conjunctive folding)

We have shown that our transformation framework subsumes each of these transformation systems. In other words, the syntactic restrictions imposed in these systems are not needed for ensuring total correctness. There is however, an interesting observation to make from the book-keeping/annotations maintained in these transformation systems.

Various kinds of clause annotations. The clause annotations maintained in the afore mentioned transformation systems are of roughly two types:

—[Tamaki and Sato 1984; 1986a; Gergatsoulis and Katzouraki 1994] partition the predicate symbols into $n > 1$ strata (among these, [Tamaki and Sato 1984] and [Gergatsoulis and Katzouraki 1994] set $n = 2$). A total order is assumed among the strata *i.e.* strata $1 \succ \ldots \succ$ strata n. Also, for each clause C in program P_i a *flag* is maintained. The flag is set if C was obtained via one or more unfoldings in the sequence $P_0, \ldots, P_i$. In a folding step, clauses C, D can be used as folded and folder clauses if
 —strata of predicate at head of C $\succ$ strata of predicate at head of D, or
 —strata of predicate at head of C $=$ strata of predicate at head of D and the flag of C is set.
 The above conditions allow the definition of a well-founded order among ground atoms on which we can induct. This idea has also been used to develop totally correct unfold/fold transformations for normal logic programs [Seki 1991; 1993].

—A different approach is taken in [Kanamori and Fujita 1987]. Here all the predicate symbols are placed in one stratum. Each clause C in program P_i is annotated with an integer counter which is incremented on unfolding and decremented on folding. In a folding step, clauses C, D can be used as folded and folder clauses if annotation of C $>$ annotation of D. This approach is similar to the work on functional program transformations by Kott [Kott 1985] and the seminal work of David Sands [Sands 1996]. Intuitively both Kott and Sands allow a folding step if the number of unfolds exceeds the number of folds (there are however important differences which we will outline in the following).

The two different kinds of annotations (strata and counters) have been combined in our SCOUT transformation system. This gives a transformation system which allows more folding steps even when the syntactic restrictions on the folder clauses hold. Thus, SCOUT can be proved to be more powerful than the Tamaki-Sato style transformation systems.

Ensuring Total Correctness without imposing Syntactic Restrictions. One of the key features of our transformation framework (as well as the SCOUT transformation system) is that the applicability of a transformation to program P_i is decided based on the clause measures (*i.e.* annotations) in P_i, and not on program syntax. This objective has previously been achieved in the work of Amtoft [Amtoft 1992]. Similar to Tamaki-Sato style transformations [Tamaki and Sato 1986a], Amtoft partitions the predicates into $n > 1$ strata. This is achieved by assigning "weights" to the predicates. In the initial program P_0, weights are assigned to a clause based on the weight of its head predicate. The weights of a clause get updated on unfolding/folding. By unfolding an atom of higher weight, more opportunities are created for future folding steps. The intuition presented by Amtoft is an important one and conceptually close to the extended Tamaki-Sato system of [Tamaki and Sato 1986a]. We believe that this similarity between the two works has not been noticed due to the scarce availability of the [Tamaki and Sato 1986a] technical report. Both these works show that by stratifying the predicates and annotating the clauses with strata number during unfolding and folding it is possible to ensure total correctness. There is also an additional restriction requiring every folding step to be conjunctive. However, this restriction can be showed to be unnecessary for ensuring total correctness.

The work of Amtoft gives us *one* way to ensure total correctness without restricting folder clause syntax. However, it is conceptually different from the work of Kanamori [Kanamori and Fujita 1987] which maintains integer counters with every clause. Like [Tamaki and Sato 1986a], Kanamori also restricts folding to be conjunctive; again this restriction is unnecessary. However, as observed in Section 4, there is an important difference between [Tamaki and Sato 1986a] and [Kanamori and Fujita 1987]. The measure space in [Kanamori and Fujita 1987] is coarser than [Tamaki and Sato 1986a] but [Kanamori and Fujita 1987] completely utilizes its measure space by maintaining accurate clause measures. Thus the book-keeping is more detailed. Intuitively we can argues that maintaining the number of unfoldings through which a clause C in program P_i was derived is more detailed than maintaining whether clause $C \in P_i$ was obtained by at least one unfolding. Thus, Kanamori's intuition can also be used to yield a transformation system with no syntactic restrictions. In fact, this system is identical to our SCOUT system where all predicates are placed in the same stratum.

In functional programming, similar ideas have been used to ensure correctness of unfold/fold transformations. Kott [Kott 1985] presents a restricted transformation system for a set of mutually recursive functions $f_1, \ldots, f_n$ in a first order functional language. In this work, a particular function f_i is transformed as follows: (a) unfold the body of f_i (b) apply certain "laws" (c) fold the body of f_i. A "law" corresponds to a rewriting of semantically equivalent expressions (similar to our goal replacement). Subsequent to the transformation, Kott's method checks the correctness of the transformation sequence. If all the functions are strict, then this check corresponds to checking that the number of unfolds exceeds the number of folds. Thus, Kanamori's method of checking counter values of folder and folded clauses is similar to Kott's check. However Kanamori checks the applicability of a folding step at the time of transformation and not post-mortem.

One of the most well-understood and comprehensive work on functional program transformations is by David Sands [Sands 1996]. Sands presents an elegant theory of improvement which clarifies the "number of unfolds exceeds number of folds" check proposed by Kott. Moreover, [Sands 1996] is applicable to higher order functional languages as well. In [Sands 1996], a transformation from $fx \triangleq e$ to $gx \triangleq e'$ using the equivalence $e \cong e'$ is totally correct if e' is an improvement over e. In other words, for any context if e terminates with n function calls then e' must terminate with at most n function calls. Sands then exploits this notion of improvement for unfold/fold transformations by requiring an unfold/fold transformation sequence to show overall improvement. Superficially, this might appear similar to Kanamori's check on counters. There are however, important differences. Since an unfold step reduces a function call, Sands records this by inserting a "*tick*". Similarly a fold step increases a function call, so it must be paid for by removing a tick (which was introduced earlier by an unfold). Thus, the total number of ticks in a function definition roughly correponds to Kanmori's counter annotation of a clause. However this correspondence holds only if ticks can be arbitrarily propagated in an expression, that is, *any* tick introduced by unfolding can be used to pay for a future folding. This is in general not true if an unfolding step produces a lazy context. The tick introduced by unfolding cannot be propagated across this lazy context and thus cannot be used to pay for a future folding. This issue does not arise in logic programs. Any unfolding step can "increase" the clause measures and can thus be used to pay for any future folding step. This is reflected in our generalized transformation framework, as well as in [Kanamori and Fujita 1987]. Furthermore, our work combines the notion of stratification with counters to maintain more fine-grained book-keeping per clause. This is evidenced in the SCOUT transformation system. Our generalized transformation framework abstractly specifies the conditions which the clause annotaions must satisfy in order to maintain total correctness. Our notion of *measure consistency* captures thes conditions. This parallels with Sands' theory of improvement for functional programs where he shows that a transformation sequence which leads to improvement (in terms of function calls) is guaranteed to preserve correctness. The tick algebra of Sands is a mechanism for ensuring this "improvement". Similarly the transformation rules in our SCOUT transformation are guaranteed to ensure the abstract notion of "measure consistency".

We conclude this section by discussing the work of Bossi, Cocco and Etalle on correctness of replacement operations in normal logic programs [Bossi et al. 1992; 1996]. Since unfolding and folding are restricted versions of goal replacement, their correctness theorem can also be used to derive a safe folding operation [Bossi et al. 1992]. In particular, their correctness condition depends on two notions:

—*dependency degree*: Intuitively, the dependency degree of an atom B on clause C is the shortest path from B to C in a proof of B. Thus, if a circularity is introduced by folding B into clause C then the length of the loop is the dependency degree of B on C.

—*semantic delay*: The semantic delay of a goal G w.r.t. another goal G' roughly denotes the minimum difference in the lengths of their derivations.

A typical sufficient condition for correct folding of clause C using folder clause B:− $B_1, \ldots, B_m$ is: *dependency degree* of B on $C \geq$ *semantic delay* of B w.r.t. $B_1, \ldots, B_m$. Intuitively, the semantic delay of B w.r.t. $B_1, \ldots, B_m$ is related to the transformation history: the unfold/fold steps taken so far from the folder clause. However, the idea of dependency degree does not only correspond to the transformation history. Instead it is also related to the stratification of the predicates used in Tamaki-Sato style systems. In particular, if B never uses clause C in its proof, then the dependency degree is the ordinal ω and folding of C using B:− $B_1, \ldots, B_m$ is always allowed. This roughly corresponds to the folding of "old" predicates w.r.t. "new" predicates in [Tamaki and Sato 1984; Gergatsoulis and Katzouraki 1994]. Thus, the notion of dependency degree is related to stratification of predicates as well as transformation history. It seems that our correctness condition is more uniform: it simply compares the clause measures of the folded and folder clauses. These clause measures can be instantiated to incorporate the notions of stratification and/or counting of past unfold/fold steps (the transformation history).

6.2 Logic Program Transformations to construct proofs

Unfold/fold logic program transformations have been primarily used for program synthesis, specialization and optimization (see [Bossi et al. 1990; Boulanger and Bruynooghe 1993; Schreye et al. 1999; Pettorossi et al. 1997]). These works use restricted versions of unfold/fold rules and concentrate on a different (and important) issue: automated strategies to guide the rules. For example, partial deduction or partial evaluation [Jones et al. 1993; Komorowski 1982] primarily allows unfolding. Folding is often restricted to a single atom, and is often used to replace a partially instantiated atom $\mathtt{p(t(\overline{X}))}$ to an open atom $\mathtt{q(\overline{X})}$ via the definition $\mathtt{q(\overline{X})} : -\mathtt{p(t(\overline{X}))}$. This is relaxed in conjunctive partial deduction [Schreye et al. 1999] which allows specialization w.r.t. conjunctions of atoms (instead of a single atom). Still folding of multiple clauses in one step is not allowed in the interests of automated control.

Relatively little work has been done on using these transformations for constructing *proofs*. As discussed in the previous section, unfold/fold transformations can be used to construct induction proofs of program properties. In such induction proofs, unfolding accomplishes the base case and the finite part of the induction step, and folding roughly corresponds to application of induction hypothesis. This observation has been exploited in [Hsiang and Srivas 1987; Kanamori and Fujita 1986; Pettorossi and Proietti 1999; 2000] to construct inductive proofs of program properties.

Hsiang and Srivas in [Hsiang and Srivas 1987] extended Prolog's evaluation with "limited forward chaining" to perform inductive theorem proving. This limited forward chaining step is in fact a very restricted form of folding: only the theorem statement (which is restricted to be conjunctive) can be used was a folder clause. The work of Kanamori and Fujita [Kanamori and Fujita 1986] is closer to ours. They proved certain first order theorems about the Least Herbrand Model of a definite logic program via induction. In particular, they observed that the least fixed point semantics of logic programs could be exploited to employ fixpoint induction. Our usage of the transformations is similar. Given a program P we intend to prove $p \equiv q$ in the Least Herbrand Model of P. To do this proof by induction, we transform p and q to obtain a program P'. If the transformed definitions of p and q in P' are "syntactically equivalent" (Definition 10) then our proof is finished. Note that this equivalence check is in fact an application of fixpoint induction. It allows us to show $p \equiv q$ in $M(P')$ (the least Herbrand model of P'). Furthermore, since $M(P') = M(P)$ this amounts to showing $p \equiv q$ in program P. Thus, in our work predicates are transformed to facilitate the construct of induction schemes (for proving predicate equivalence). [Kanamori and Fujita 1986] also exploits transformations for similar purposes. However, their method performs *conjunctive* folding using only a single non-recursive clause. Apart from the restriction in their folding rule, they also do not employ goal replacement in their induction proofs. Consequently, nested induction proofs cannot be constructed (the example worked out in Section 5 is a nested induction proof).

The idea of using logic program transformations for proving goal equivalences was explored in [Pettorossi and Proietti 1999; 2000]. These works employ more restricted Tamaki-Sato style unfold/fold transformations, which are not suitable in general for constructing induction proofs of temporal properties. This is because temporal properties employ fixed point operators, and are typically encoded using multiple recursive clauses. A simple reachability property EFp (which specifies that a state in which proposition p holds is reachable) [Clarke et al. 1999] will be encoded as a logic program as follows:

```
ef(X) :- p(X).
ef(X) :- trans(X,Y), ef(Y).
```

where the predicate `trans` captures the transtion relation of the system being verified, and `p(X)` is true if the proposition p holds in state `X`. This encoding contains two clauses one of which is recursive. Our work relaxes restrictions on the applicability of the transformation rules thereby enabling their use in proving temporal properties.

The reader might notice similarities between a proof system based on unfold/fold transformations a proof systems based on tabled resolution [Tamaki and Sato 1986b; Chen and Warren 1996]. Tabled resolution combines resolution proofs with memoing of calls and answers. Since folding corresponds to remembering the original definition of predicates, there is some correspondence between folding and memoing. However, folding can remember conjunctions and/or disjunctions of atoms as the definition of a predicate. This is not possible in tabled resolution. Furthermore, in tabled resolution when a tabled call C is encountered, the answers produced so far for C are used to produce new answers for C. In folding, when the clause bodies in old definition of a predicate is encountered, it is simply replaced by the clause head.

The unfold/fold transformation based proof technique for constructing induction proofs also differs substantially from many existing inductive theorem proving techniques [Boyer and Moore 1990; Bundy et al. 1990]. These provers take in an explicit induction schema and try to dispense the proof obligation in each of these cases. In contrast, the transformation based proof technique does not input any induction schema. The schema is constructed gradually via unfolding of the program predicates. This idea has similarities to the "recursion analysis" technique employed in the Boyer-Moore prover [Boyer and Moore 1975; 1990]. Given some functions, these works exploit the recursive structure of these functions to prove theorems about them. Note that, if the necessary induction schema cannot be derived via unfolding, our transformation based proof technique cannot find a proof. However, this restriction leads to increased automation in the construction of induction proofs, and fewer cases in the induction schema constructed (see [Roychoudhury 2000] for a detailed example).

In conclusion, we would like to note that constructing induction proofs via unfold/fold transformations is different from consistency based proof technqiues such as inductionless induction [Comon and Nieuwenhuis 2000]. These techniques do not employ any induction schema at all. To prove a predicate equivalence $\mathtt{p} \equiv \mathtt{q}$ our proof technique uses an induction schema obtained from the structure of the transformed definitions of p, q. However, this schema is not given a-priori but gradually constructed via program transformations.

7. DISCUSSIONS

The development of a parameterized framework for unfold/fold transformations has several important implications. It enables us to compare existing transformation systems and modify them without redoing the correctness proofs (e.g., extending measures for goal replacement in Section 3). It also facilitates the development of new transformations systems. For instance, we derived SCOUT which permits folding using multiple recursive clauses.

Motivation. The development of our transformation framework is motivated by its application in constructing induction proofs. As described in Section 5, our transformation framework can be used for inductively proving predicate equivalences. In these proofs, the unfolding transformation helps prove the base case and the finite part of the induction step. The folding transformation is useful for uncovering the induction hypothesis. Finally, the goal replacement transformation is used for constructing nested induction proofs (semantic equivalence of the goals interchnaged in a goal replacement step are also proved by program transformations).

Extensions. In [Roychoudhury et al. 2002], we have extended the work reported in this paper to obtain generalized unfold/fold transformation systems for normal logic programs. Aravindan and Dung [1995] developed an approach to parameterize the correctness proofs of the original Tamaki-Sato system with respect to various semantics based on the notion of *semantic kernels*. Incorporating the idea of semantic kernel into our framework yields a framework that is parameterized with respect to the measure structures as well as semantics.

Future Work. In future, it would be interesting to study whether we can develop similar parameterized unfold/fold transformation frameworks for other programming paradigms such as functional programming [Sands 1996], constraint logic programming [Etalle and Gabbrielli 1996; Maher 1993], concurrent constraint programming [Etalle et al. 2001], and process algebraic specification languages (*e.g.* CCS) [Francesco and Santone 1998].

In particular, [Sands 1996] reports powerful unfold/fold transformation rules for functional languages, where the gains accrued from unfolding determine the applicability of folding (similar to our framework). A comparison of Sands' work with our transformation framework appeared in Section 6. It would be interesting to study whether the transformation system of [Sands 1996] can be parameterized w.r.t. measure structures in a manner similar to ours.

8. ACKNOWLEDGEMENTS:

A preliminary version of this article appeared in the proceedings of International Conference on Principles and Practice of Declarative Programming (PPDP) 1999, LNCS 1702. We would like to thank the anonymous referees of PPDP 1999 for their valuable comments. We thank Alberto Pettorossi and Maurizio Proietti for useful discussions and pointers to earlier work. We also thank Sandro Etalle for relevant references, Taisuke Sato for providing us with a copy of his technical report [Tamaki and Sato 1986a], and David S. Warren for his comments about an earlier draft of this paper. This work was partially supported by NSF grants CCR-9711386, EIA-9705998 and CDA-9805735.

REFERENCES

Amtoft, T. 1992. Unfold/fold transformations preserving termination properties. In *4th International Symposium on Programming Language Implementation and Logic Programming (PLILP '92), Leuven, Belgium*, M. Bruynooghe and M. Wirsing, Eds. LNCS, vol. 631. Springer-Verlag, 187–201.

Aravindan, C. and Dung, P. 1995. On the correctness of unfold/fold transformations of normal and extended logic programs. *Journal of Logic Programming 24,* 3, 295–322.

Bossi, A. and Cocco, N. 1994. Preserving universal termination through unfold/fold. In *International Conference on Algebraic and Logic Programming (ALP), LNCS 850*. 269–286.

Bossi, A., Cocco, N., and Dulli, S. 1990. A method of specializing logic programs. *ACM Transactions on Programming Languages and Systems 12,* 2, 253–302.

Bossi, A., Cocco, N., and Etalle, S. 1992. On safe folding. In *Programming Language Implementation and Logic Programming (PLILP), LNCS 631*. 172–186.

Bossi, A., Cocco, N., and Etalle, S. 1996. Simultaneous replacement in normal programs. *Journal of Logic and Computation 6,* 1 (February), 79–120.

Boulanger, D. and Bruynooghe, M. 1993. Deriving unfold/fold transformations of logic programs using extended OLDT-based abstract interpretation. *Journal of Symbolic Computation 15,* 5/6, 495–521.

Boyer, R. and Moore, J. 1975. Proving theorems about Lisp functions. *Journal of the ACM 22,* 1, 129–144.

Boyer, R. and Moore, J. 1990. A Theorem Prover for a Computational Logic. In *International Conference on Automated Deduction (CADE), LNAI 449*. 1–15.

Bundy, A. et al. 1990. The oyster-clam system. In *International Conference on Automated Deduction (CADE), LNAI 449*. 647–648.

Chen, W. and Warren, D. 1996. Tabled evaluation with delaying for general logic programs. *Journal of the ACM 43,* 1, 20–74.

Clarke, E., Grumberg, O., and Peled, D. 1999. *Model Checking.* MIT Press.

Comon, H. and Nieuwenhuis, R. 2000. Induction= i-axiomatization + first-order consistency. *Information and Computation 159,* 1-2, 151–186.

Das, S. K. 1992. *Deductive Databases and Logic Programming.* Addison-Wesley.

Etalle, S. and Gabbrielli, M. 1996. Transformations of CLP modules. *Theoretical Computer Science 166,* 1, 101–146.

Etalle, S., Gabrielli, M., and Meo, M. 2001. Transformations of CCP programs. *ACM Transactions on Programming Languages and Systems 23,* 3, 304–395.

Francesco, N. D. and Santone, A. 1998. A transformation system for concurrent processes. *Acta Informatica 35,* 12, 1037–1073.

Gergatsoulis, M. and Katzouraki, M. 1994. Unfold/fold transformations for definite clause programs. In *International Symposium on Programming Language Implementation and Logic Programming (PLILP), LNCS 844*. 340–354.

Hsiang, J. and Srivas, M. 1987. Automatic inductive theorem proving using Prolog. *Theoretical Computer Science 54*, 3–28.

Jones, N., Gomard, C., and Sestoft, P. 1993. *Partial Evaluation and Automatic Program Generation.* Prentice Hall.

Kanamori, T. and Fujita, H. 1986. Formulation of Induction Formulas in Verification of Prolog Programs. In *International Conference on Automated Deduction (CADE)*. 281–299.

Kanamori, T. and Fujita, H. 1987. Unfold/fold transformation of logic programs with counters. In *USA-Japan Seminar on Logics of Programs, Also available as ICOT Technical Report.*

Kawamura, T. and Kanamori, T. 1990. Preservation of stronger equivalence in unfold/fold logic program transformation. *Theoretical Computer Science 75,* 1&2.

Komorowski, J. 1982. Partial evaluation as a means for inferencing data structures in an applicative language: A theory and implementation in the case of prolog. In *ACM SIGPLAN International Conference on Principles of Programming Languages (POPL).*

Kott, L. 1985. *Unfold/fold program transformations.* Algebraic Methods in Semantics. Cambridge University Press, 412–433.

Lloyd, J. 1993. *Foundations of Logic Programming, Second Edition.* Springer-Verlag.

Maher, M. 1987. Correctness of a logic program transformation system. Tech. rep., IBM T.J. Watson Research Center.

Maher, M. J. 1993. A transformation system for deductive database modules with perfect model semantics. *Theoretical Computer Science 110*, 377–403.

Pettorossi, A. and Proietti, M. 1998. *Transformation of logic programs.* Handbook of Logic in Artificial Intelligence, vol. 5. Oxford University Press, 697–787.

Pettorossi, A. and Proietti, M. 1999. Synthesis and transformation of logic programs using unfold/fold proofs. *Journal of Logic Programming 41,* 2–3, 197–230.

Pettorossi, A. and Proietti, M. 2000. Perfect model checking via unfold/fold transformations. In *Computational Logic, LNCS 1861.*

Pettorossi, A., Proietti, M., and Renault, S. 1997. Reducing nondeterminism while specializing logic programs. In *ACM SIGPLAN International Conference on Principles of Programming Languages (POPL)*. 414–427.

Roychoudhury, A. 2000. Program transformations for verifying parameterized systems. Ph.D. thesis, State University of New York at Stony Brook, Available from `http://www.comp.nus.edu.sg/~abhik/papers.html`.

Roychoudhury, A., Kumar, K. N., Ramakrishnan, C. R., and Ramakrishnan, I. V. 2002. Beyond Tamaki-Sato style unfold/fold transformations for normal logic programs. *International Journal on Foundations of Computer Science 13,* 3, 387–403.

Roychoudhury, A., Kumar, K. N., Ramakrishnan, C. R., Ramakrishnan, I. V., and Smolka, S. A. 2000. Verification of parameterized systems using logic program transformations. In *International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS)*. Vol. LNCS 1785. Springer-Verlag, 172–187.

ROYCHOUDHURY, A. AND RAMAKRISHNAN, I. V. 2001. Automated inductive verification of parameterized protocols. In *International Conference on Computer Aided Verification (CAV).* LNCS, vol. 2102. Springer-Verlag, 25–37.

SANDS, D. 1996. Total correctness by local improvement in the transformation of functional programs. *ACM Transactions on Programming Languages and Systems 18,* 2, 175–234.

SCHREYE, D. D., GLUCK, R., JORGENSEN, J., LEUSCHEL, M., MARTENS, B., AND SORENSEN, M. 1999. Conjunctive partial deduction: Foundations, control, algorithms, and experiments. *Journal of Logic Programming 41*, 233–277.

SEKI, H. 1991. Unfold/fold transformation of stratified programs. *Theoretical Computer Science 86,* 1, 107–139.

SEKI, H. 1993. Unfold/fold transformation of general logic programs for well-founded semantics. *Journal of Logic Programming 16,* 1, 5–23.

TAMAKI, H. AND SATO, T. 1984. Unfold/fold transformations of logic programs. In *Proceedings of International Conference on Logic Programming.* 127–138.

TAMAKI, H. AND SATO, T. 1986a. A generalized correctness proof of the unfold/ fold logic program transformation. Tech. rep., Ibaraki University, Japan.

TAMAKI, H. AND SATO, T. 1986b. OLDT resolution with tabulation. In *Third International Conference on Logic Programming.* 84–98.

APPENDIX

In this appendix, we briefly outline the transformation systems of [Kanamori and Fujita 1987] and [Tamaki and Sato 1986a] for the convenience of the reader. We then prove that SCOUT is a more powerful transformation sequence (in terms of allowed transformation sequences).

A. TRANSFORMATION SYSTEM OF KANAMORI-FUJITA

In this work [Kanamori and Fujita 1987], each clause of any program P_i in a transformation sequence $P_0, P_1, \ldots$ is annotated with an integer counter. The counter of each clause in the initial program P_0 is set to 1. The unfolding and folding rules are as follows.

RULE 8. **Unfolding** *Let C be a definite clause in program P_i with counter γ and A be an atom in the body of C. Let $C_1, \ldots, C_m$ be all the clauses in P_i whose heads are unifiable with A with m.g.u $\sigma_1, \ldots, \sigma_m$. Let the counters of $C_1, \ldots, C_m$ be $\gamma_1, \ldots, \gamma_m$. Let C'_j be the clause that is obtained by replacing $A\sigma_j$ by the body of $C_j\sigma_j$ in $C\sigma_j$ ($1 \leq j \leq m$). Assign $(P_i - \{C\}) \cup \{C'_1, \ldots, C'_m\}$ to P_{i+1}. The counter of C'_j is $\gamma + \gamma_j$ for all $1 \leq j \leq m$.*

RULE 9. **Folding** *Let C be a definite clause in P_i of the form $A{:-}\ A_1, \ldots, A_n$ with counter γ and let D be a clause in P_j ($j \leq i$) of the form $B{:-}\ B_1, \ldots, B_m$ with counter δ. There is no other clause in P_j whose head is unifiable with B. Suppose there is a substitution σ and atoms $A_1, \ldots, A_m$ ($m \leq n$) in the body of C s.t.*

—*$A_j = B_j\sigma$ for $j = 1, \ldots, m$*

—*σ substitutes distinct variables for the internal variables of D and moreover those variables do not occur in $\{A, A_{m+1}, \ldots, A_n\}$.*

—*$m + \delta < n + \gamma$*

Define a clause C' as $A{:-}\ B\sigma, A_{m+1}, \ldots, A_n$, and assign $P_i - \{C\} \cup \{C'\}$ to P_{i+1}. The counter of C' is $\gamma - \delta$.

As an extension, [Kanamori and Fujita 1987] mentions that the predicate symbols of the program P_0 can be partitioned into strata. Folding can then be allowed even if $m + \delta = n + \gamma$. This extension can also be captured by our transformation framework. Furthermore, [Kanamori and Fujita 1987] mentions that the counters of the clauses produced by unfolding (folding) at P_i are given as above, unless this clause is already present in P_i with a lower counter.

B. EXTENDED TAMAKI-SATO SYSTEM

This work [Tamaki and Sato 1986a] starts with a "layered" program P_0 where the predicate symbols are partitioned into n strata or *descent levels*. The stratification should be such that every predicate symbol in the body of a clause C has a levl not greater than the level of the perdicate at the head of C. The level of a clause in P_0 is the level of its head predicate.

The transformation rules are given as in [Kanamori and Fujita 1987]. The only difference is that each clause of program P_i in a transformation sequence $P_0, P_1, \ldots$ is annotated with a descent level (instead of an integer counter). The descent level of a clause C is:

—if $C \in P_0$ then the descent level of C is the descent level of the predicate symbol in its head.

—if $C \in P_{i+1}$ is introduced by unfolding $C' \in P_i$ at atom A, then the descent level of C is tge minimum of the descent level of C' and the descent level of the predicate symbol in A.

—if $C \in P_{i+1}$ is introduced by folding/goal replacement of clause $C' \in P_i$, then the descent level of C is the descent level of C'.

—Finally, a clause $C \in P_i$ can be folded using a clause D as folder provided the descent level of C is smaller than the descent level of D.

C. SCOUT IS A MORE POWERFUL TRANSFORMATION SYSTEM

In this section, we prove that SCOUT allows more transformation sequences than the counter based transformation system of Kanamori Fujita [1987] as well as the stratification based transformation system of Tamaki-Sato [1986a].

Kanamori-Fujita system [1987]. This system is special case of SCOUT where folding is conjunctive and all the predicate symbols of the initial program are placed in a single stratum.

Extended Tamaki-Sato system. For proving that SCOUT covers any transformation sequence $P_0, P_1, P_2, \ldots$ which is allowed by the fold/unfold/goal replacement system of [Tamaki and Sato 1986a], we define the invariants given below. Recall that in [Tamaki and Sato 1986a] each clause in any P_i is associated with a strata number, also called the descent level. Folding of a clause C (folded clause) using a clause D (folder clause) is allowed if: descent level of $C <$ descent level of D. Also, since [Tamaki and Sato 1986a] handles only conjunctive folding, any fold/unfold transformation sequence of [Tamaki and Sato 1986a], *if executable in SCOUT*, will always produce clauses with counters of the form (γ, γ); in other words, the two counters of any clause will always be equal.

We now consider the following invariants :

—$J1(P_i) \equiv$ Any fold/unfold/goal replacement transformation in P_i which is allowed by the extended Tamaki-Sato system [1986a] is allowed by SCOUT (with n strata).

—$J2(P_i) \equiv$ Let C be any clause in program P_i with strata number (*i.e.* descent level in the terminology of [Tamaki and Sato 1986a]) j. Then, in SCOUT (with n strata), $\gamma^i_{lo}(C) = \gamma^i_{hi}(C) = \langle \gamma_1, \ldots, \gamma_n \rangle$ where $\gamma_j > 0 \wedge (\forall 1 \leq k < j \; \gamma_k = 0)$

To prove that any unfold/fold/goal replacement transformation sequence covered by [Tamaki and Sato 1986a] is also covered by SCOUT, it is sufficient to prove that $J1(P_i)$ is an invariant.

THEOREM 5. *Let $P_0, P_1, P_2, \ldots$ be an unfold/fold/goal replacement transformation sequence of the extended Tamaki-Sato system [1986a]. Then, $\forall i \geq 0$. $J1(P_i) \wedge J2(P_i)$*

PROOF. The proof follows by induction on i. $J1(P_0)$ is trivially true by the definition of the fold/unfold transformations in [Tamaki and Sato 1986a] and SCOUT. Also, if a clause C in P_0 has descent level j, then $\gamma^o_{lo}(C) = \gamma^o_{hi}(C) = \langle \gamma_1, \ldots, \gamma_n \rangle$

where $\gamma_j = 1$ and $\gamma_l = 0$ when $l \neq j$. Clearly then $J2(P_0)$ is also true. Thus, we have established the basis for the induction.

Now assume that $\forall i \leq m.\ J1(P_i) \wedge J2(P_i)$. We now show that $J1(P_{m+1}) \wedge J2(P_{m+1})$ holds.

First we prove $J2(P_{m+1})$. Let C be any clause in P_{m+1}. We show that the property mentioned in $J2$ is true for C.

Case 1: C is inherited from P_m

The result holds since $J2(P_m)$ is true by induction hypothesis.

Case 2: C is obtained by unfolding C' using C''

Since, $\forall i \leq m.\ J1(P_i)$, the sequence $P_0, P_1, \ldots, P_m, P_{m+1}$ can be constructed using SCOUT. Then, $\gamma_{lo}^{m+1}(C) = \gamma_{hi}^{m+1}(C) = \gamma_{lo}^{m}(C') \oplus \gamma_{lo}^{m}(C'') = \gamma_{hi}^{m}(C') \oplus \gamma_{hi}^{m}(C'')$. Also let the descent level of C, C' and C'' be k,k' and k'' respectively. Then, by [Tamaki and Sato 1986a], $k = min(k', k'')$. By the induction hypothesis, the property in $J2$ is true for both C' and C''. Hence if $\gamma_{lo}^{m}(C') = \gamma_{hi}^{m}(C') = \langle \gamma_1', \ldots, \gamma_n' \rangle$ and $\gamma_{lo}^{m}(C'') = \gamma_{hi}^{m}(C'') = \langle \gamma_1'', \ldots, \gamma_n'' \rangle$, then $\gamma_1' = \cdots = \gamma_{k-1}' = 0$, $\gamma_1'' = \cdots = \gamma_{k-1}'' = 0$. Also since k is the minimum of k' and k'', we have either $\gamma_k' = 0 \wedge \gamma_k'' > 0$, or $\gamma_k' > 0 \wedge \gamma_k'' = 0$ or $\gamma_k' > 0 \wedge \gamma_k'' > 0$. Now, $\gamma_{lo}^{m+1}(C) = \gamma_{hi}^{m+1}(C) = \gamma_{lo}^{m}(C') \oplus \gamma_{lo}^{m}(C'') = \langle \gamma_1, \ldots, \gamma_n \rangle$ where $\forall 1 \leq l \leq n\ \gamma_l = \gamma_l' + \gamma_l''$. Hence $\gamma_1 = \cdots = \gamma_{k-1} = 0$ and $\gamma_k > 0$. Thus the property in $J2$ holds for C.

Case 3: C is obtained by folding C' using D'

Since $\forall i \leq m.\ J1(P_i)$, the transformation sequence $P_0, P_1, \ldots, P_m, P_{m+1}$ can be constructed using SCOUT. Let C' and D' have descent levels k and l respectively. Then by [Tamaki and Sato 1986a], the descent level of C is also k and $k < l$. But $D' \in P_0$, so $\gamma_{lo}^{0}(D') = \gamma_{hi}^{0}(D') = \langle \delta_1', \ldots, \delta_n' \rangle$ where $\delta_l' = 1$ and $\delta_j' = 0$ when $j \neq l$. Let $\gamma_{lo}^{m}(C') = \gamma_{hi}^{m}(C') = \langle \gamma_1', \ldots, \gamma_n' \rangle$. As the property in $J2$ is true for C', we have $\gamma_1' = \cdots = \gamma_{k-1}' = 0$ and $\gamma_k' > 0$. Now, $\gamma_{lo}^{m+1}(C) = \gamma_{hi}^{m+1}(C) = \gamma_{lo}^{m}(C') \ominus \gamma_{hi}^{0}(D') = \langle \gamma_1, \ldots, \gamma_n \rangle$ where $\forall 1 \leq j \leq n\ \gamma_j = \gamma_j' - \delta_j$. Since $k < l$, therefore $\delta_1' = \cdots = \delta_k' = 0$. Thus, $\gamma_1 = \cdots = \gamma_{k-1} = 0$ and $\gamma_k = \gamma_k' > 0$. Hence the property in $J2$ holds for C.

Case 4: C is obtained by goal replacement from clause $C' \in P_m$.

Again, since $\forall i \leq m.\ J1(P_i)$, the transformation sequence $P_0, \ldots, P_m, P_{m+1}$ can be constructed using SCOUT. Let C' have descent level k. then, by [Tamaki and Sato 1986a], the clause $C \in P_{m+1}$ also has descent level k. Let $\gamma_{lo}^{m}(C') = \gamma_{hi}^{m}(C') = \langle \gamma_1', \ldots, \gamma_n' \rangle$ and $\gamma_{lo}^{m+1}(C) = \gamma_{hi}^{m+1}(C) = \langle \gamma_1, \ldots, \gamma_n \rangle$ Since property $J2$ is true for C' therefore $\gamma_1' = \cdots = \gamma_{k-1}' = 0$ and $\gamma_k' > 0$. Let C be obtained from C' by replacing goal G with G'. Now, from the definition of goal replacement in [Tamaki and Sato 1986a], for any ground instantiation θ we have $\alpha(G\theta) \ominus \alpha(G'\theta) \succeq \delta = \langle 0, 0, \ldots, 0 \rangle$ The clause measure of clause C will be $\gamma_{lo}^{m+1}(C) = \gamma_{hi}^{m+1}(C) = \gamma_{lo}^{m}(C') = \gamma_{hi}^{m}(C')$ Therfore, clearly property $J2$ holds for clause C as well.

Thus, $\gamma_{lo}^{m}(C') \oplus \delta \oplus \sum_{1 \leq l \leq k} \alpha_{min}(A_l') \succeq \mathbf{0}$ where $A_1', \ldots, A_k'$ are the body atoms other than G in clause C' (this holds because $\gamma_{lo}^{m}(C') \succeq \mathbf{0}$, $\delta = \mathbf{0}$ and $\alpha_{min}(A) \succeq \mathbf{0}$ for any atom A. Furthermore,

We now show that $J2(P_{m+1}) \Rightarrow J1(P_{m+1})$. Since the unfolding transformation is independent of any condition on the stratified counter (or descent level) in SCOUT or [Tamaki and Sato 1986a], therefore any unfolding allowed by [Tamaki and Sato 1986a] in P_{m+1} is also allowed by SCOUT.

For folding, let $C \in P_{m+1}$ be folded using the folder $D \in P_0$ in the system of [Tamaki and Sato 1986a]. Let the descent levels of C and D be be k and l respectively. Then, $k < l$ (by [Tamaki and Sato 1986a]) and the property of $J2$ is true for both C and D (since $J2(P_{m+1})$ holds). So, if $\gamma_{lo}^{m+1}(C) = \gamma_{hi}^{m+1}(C) = \langle \gamma_1, \ldots, \gamma_n \rangle$ and $\gamma_{lo}^0(D) = \gamma_{hi}^0(D) = \langle \delta_1, \ldots, \delta_n \rangle$ we have $\gamma_1 = ... = \gamma_{k-1} = 0$, $\gamma_k > 0$, $\delta_1 = ... = \delta_{l-1} = 0$. As $k < l$, this means $\delta_1 = ... = \delta_k = 0$. Clearly then $\gamma_{lo}^{m+1}(C)$ is lexicographically greater than $\gamma_{hi}^0(D)$. Hence C can be folded using D as folder in SCOUT.

For goal replacement, let $C \in P_{m+1}$ be of the form $A{:-}\ G, A'_1, \ldots, A'_k$ and let it be replaced in [Tamaki and Sato 1986a] system to produce clause $C' \equiv A{:-}\ G', A'_1, \ldots, A'_k$. Let the descent level of C be k. Then, the descent level of clause C' is also k. By setting $\delta = \langle 0, \ldots, 0$ we have $\alpha(G\theta) \ominus \alpha(G'\theta) \succeq \delta = \langle 0, 0, \ldots, 0 \rangle$. We also require $\gamma_{lo}^{m+1}(C) \oplus \delta \oplus \sum_{1 \leq l \leq k} \alpha_{min}(A'_l) \succeq \mathbf{0}$ for this goal replacement to be applicable in SCOUT. Since property J2 holds for clause C therefore $\gamma_{lo}^{m+1}(C) \succeq \mathbf{0}$. Furthermore, $\delta = \mathbf{0}$ and $\alpha_{min}(A) \succeq \mathbf{0}$ for any atom A. Therefore, $\gamma_{lo}^{m+1}(C) \oplus \delta \oplus \sum_{1 \leq l \leq k} \alpha_{min}(A'_l) \succeq \mathbf{0}$ and the goal replacement transformation is applicable in SCOUT. This completes the proof. $\square$

Thus, we have proved that SCOUT allows all unfold/fold transformation sequences allowed by [Tamaki and Sato 1986a]. To prove that it is *strictly more powerful*, we need to give an example transformation sequence which is allowed by SCOUT, but not by [Tamaki and Sato 1986a]. Any example requiring disjunctive folding serves this purpose. Hence we conclude that SCOUT is strictly more powerful than [Tamaki and Sato 1986a].