Transformation Rules for Logic Programs with Goals as Arguments*

Alberto Pettorossi[†] and Maurizio Proietti[‡]

[†]Dipartimento di Informatica, Sistemi e Produzione, Università di Roma Tor Vergata, Via di Tor Vergata, 00133 Roma, Italy.

[‡]IASI-CNR, Viale Manzoni 30, 00185 Roma, Italy.

{adp,proietti}@iasi.rm.cnr.it

http://www.iasi.rm.cnr.it/~{adp,proietti}

Abstract

We introduce a logic language where predicate symbols may have both terms and goals as arguments. We define its operational semantics by extending SLD-resolution with the leftmost selection rule, and we propose a set of transformation rules for manipulating programs written in that language. These transformation rules are shown to be correct in the sense that they preserve the chosen operational semantics. This logic language has higher order capabilities which turn out to be very powerful for the derivation of efficient logic programs. In particular, in our language we can avoid the problem of goal rearrangement which is often encountered during program transformation. Moreover, goals as arguments allow us to perform on logic programs transformation steps similar to the ones performed on functional programs when using higher order generalizations and continuation arguments.

1 Introduction

In the practice of logic programming the idea of having goals as arguments of predicate symbols is not novel (see, for instance, [16, 19]). Goals may occur as arguments when writing meta-interpreters, such as:

$solve(true) \leftarrow$
$solve(\,(A,G)\,) \leftarrow A,\, solve(G)$

or when expressing the meaning of logical connectives, such as:

$or(P,Q) \leftarrow P$
$or(P,Q) \leftarrow Q$

*In: A. Bossi (ed.) Proceedings of Logic-Based Program Synthesis and Transformation (LoPSTr'99), Venezia, Italy, 1999, Lecture Notes in Computer Science 1817, Springer, 2000, pp. 177-196.

or when making use of continuations, as indicated by the following program:

$p([\,],\ \mathit{Cont}) \leftarrow \mathit{Cont}$
$p([X|Xs],\ \mathit{Cont}) \leftarrow p(Xs, q(X, \mathit{Cont}))$
$q(0,\ \mathit{Cont}) \leftarrow \mathit{Cont}$

where the goal $p(l,\ \mathit{true})$ succeeds iff the list l consists of 0's only.

In this paper we consider an extended logic language which allows variables to range over goals and allows goals to occur as arguments of predicate symbols. The possibility of storing and manipulating goals as arguments provides 'higher order' capabilities so that in our logic language we can manipulate not only terms (that is, data) but also goals (that is, procedure calls). These capabilities turn out to be useful for performing program manipulations which are often required during program transformation [5, 10, 17]. Indeed, as we will see, by using goals as arguments we will be able to overcome the goal rearrangement problem, which occurs when for performing a folding step, we are required to modify the positions of the atoms in the body of a clause. This problem is due to the fact that the atoms in the body of a clause cannot always be rearranged without affecting the termination properties of the programs [4].

In our logic language we do not have full higher order capabilities, because in particular, quantification over function or predicate variables is not allowed.

The semantics of our higher order logic programs is a simple extension of the familiar operational semantics of definite logic programs. The operational semantics we consider, is based on SLD-resolution with the leftmost selection rule, also called LD-resolution [1], and we extend unification by allowing goal variables to be bound to goals. We then consider an enhancement of the usual unfold/fold transformation rules so that they may be used also in the case when goals occur as arguments. For instance, we will allow the unfolding of a clause w.r.t. an argument which is an atom. The definition of the semantics of our language guarantees that the transformation rules are correct (see Section 5), and in particular, when defining our semantics, we will take into account the fact that the unfolding of a goal argument may not preserve the set of successes, as indicated by the following example. Let us consider the program:

1. $h \leftarrow p(q)$
2. $p(q) \leftarrow$
3. $q \leftarrow s$

If we unfold the goal argument q in clause 1 using clause 3, we get the clause:

$1'$. $h \leftarrow p(s)$

and by using LD-resolution in the usual way, the goal h succeeds in the original program, while it fails in the derived program, consisting of clauses $1'$, 2, and 3. Thus, the set of successes is not preserved by unfolding a goal argument. Similar incorrectness may also arise with other transformation rules, such as folding and goal replacement.

In order to define the operational semantics of our logic language for which suitable enhancements of the unfolding, folding, and goal replacement rules are correct, we will proceed as follows. We first define the syntax of our logic language so that each clause head occurring in our programs has distinct variables, and unifications are realized by means of the equality predicate. Thus, for instance, clause 2 is rewritten as:

$2'$. $p(G) \leftarrow G = q$

Then the semantics is defined in such a way that, given two goals g_1 and g_2, it gives a value to the equality $g_1 = g_2$ only when g_1 is a goal variable not occurring in the goal g_2 (in particular, our operational semantics is *undefined* for the equality $g = g$, where g is a goal). In other words, we will modify LD-resolution so that a derivation *gets stuck* when it encounters a goal of the form $g_1 = g_2$, where g_1 is either a non-variable goal or a goal variable occurring in g_2. Thus, the derivation starting from the goal h and using the original program (that is, clauses 1, $2'$, and 3) gets stuck because the goal $q = q$ is selected, and also the derivation starting from the goal h and using the transformed program (that is, clauses $1'$, $2'$, and 3) gets stuck because the goal $s = q$ is selected.

Essentially, in our higher order logic language we may introduce new goals, pass them as arguments, and evaluate them, but we cannot inspect their inner structure, because the unification between goals can only be performed via the equality predicate $=$, and as already mentioned, the semantics of the equality $g_1 = g_2$ between goals is defined only when g_1 is a variable not occurring in g_2.

The paper is organized as follows. In the next sections we will introduce the definitions of the syntax (Section 2), the operational semantics (Section 3), and the transformation rules (Section 4) for our logic programs with goals as arguments. The use of the transformation rules according to suitable conditions ensures that the transformation rules are correct in the sense that they preserve the set of successes and the set of finite failures (Section 5). Actually, we will consider two forms of correctness: a *weak* and a *strong* form. The strong form of correctness ensures that a goal succeeds or fails in the original program iff it succeeds or fails, respectively, in the transformed program. The weak form of correctness consists in the 'only-if' part of the strong form, that is, if a goal succeeds or fails in the original program then it succeeds or fails, respectively, in the transformed program. Thus, when the weak form of correctness holds, the transformed program may be *more defined* than the original program in the sense that there may be some goal which has no semantic value (that is, either its evaluation does not terminate or it gets stuck) in the original program, whereas it has a value (that is, it terminates) in the transformed program.

Finally, we show through an example (Section 6) how our extended logic language and our transformation rules can be used for avoiding the problem of rearranging goals while transforming programs for eliminating multiple traversals of data structures.

2 A logic language with goals as arguments

Let us now introduce the logic language we use. It is constructed starting from the following pairwise disjoint sets: (i) *individual variables*: $X, X_1, X_2, \ldots$, (ii) *goal variables*: $G, G_1, G_2, \ldots$, (iii) *function symbols* (with arity): $f, f_1, f_2, \ldots$, and (iv) *predicate symbols* (with arity): *true*, *false*, $=$, $p, p_1, p_2, \ldots$ Individual and goal variables are collectively called *variables*, and they are ranged over by $V, V_1, V_2, \ldots$

Terms: $t, t_1, t_2, \ldots$, *goals*: $g, g_1, g_2, \ldots$, and *arguments*: $u, u_1, u_2, \ldots$, have the following syntax:

$$
\begin{aligned}
t &::= X \mid f(t_1, \ldots, t_n) \\
g &::= G \mid \mathit{true} \mid \mathit{false} \mid t_1 = t_2 \mid G = g \mid p(u_1, \ldots, u_m) \mid g_1, g_2 \mid g_1; g_2 \\
u &::= t \mid g
\end{aligned}
$$

The binary operators ',' and ';' denote, respectively, *conjunction* and *disjunction* of goals. They are assumed to be associative with neutral elements *true* and *false*, respectively. Thus, a goal g is the same as (true, g) and (g, true). Similarly, g is the same as $(\mathit{false}; g)$ and $(g; \mathit{false})$. Goals of the form $p(u_1, \ldots, u_m)$ are also called *atoms*.

Clauses $c, c_1, c_2, \ldots$ have the following syntax:

$$c ::= p(V_1, \ldots, V_m) \leftarrow g$$

where $V_1, \ldots, V_m$ are distinct variables. The atom $p(V_1, \ldots, V_m)$ is called the *head* of clause c and the goal g is called the *body* of c. A clause of the form: $p(V_1, \ldots, V_m) \leftarrow \mathit{true}$ will also be written as: $p(V_1, \ldots, V_m) \leftarrow$.

Programs $P, P_1, P_2, \ldots$ are sets of clauses of the form:

$$
\begin{aligned}
&p_1(V_1, \ldots, V_{m1}) \leftarrow g_1 \\
&\qquad\qquad\vdots \\
&p_k(V_1, \ldots, V_{mk}) \leftarrow g_k
\end{aligned}
$$

where $p_1, \ldots, p_k$ are distinct predicate symbols, and every predicate symbol occurring in g_1 or $\ldots$ or g_k, is among $p_1, \ldots, p_k$.

Notes on syntax: (1) The assumption that clause heads have only variables as arguments is not restrictive, because one may use equalities in the bodies for binding those variables to terms or goals, as required. (2) The assumption that in every program there exists at most one clause per predicate symbol is not restrictive, because one may use disjunction in the body. In particular, every definite logic program written using the familiar syntax [8], can be rewritten into an equivalent program of our language by suitable introductions of equalities and ';' operators in the bodies of clauses. (3) equality between goals is not symmetric because on the left hand side there must be a goal variable.

For any given syntactic construct r, we use $\mathit{vars}(r)$ to denote the set of variables occurring in r.

3 Operational Semantics

In this section we define a 'big step' operational semantics for our logic language (for an elementary presentation of this technique the reader may refer to [20]). Given a program P, we define the semantics of P as a (possibly empty) ternary relation $P \vdash g \rightarrow b$, where g is a goal and b is either *true* or *false*, denoting that for P and g all the LD-derivations are finite, and either one of them is successful, in which case b is *true*, or all of them are failed, in which case b is *false*.

Given a set of deduction rules, a *deduction tree* τ for $P \vdash g \rightarrow b$ is a tree such that: (i) the root of τ is $P \vdash g \rightarrow b$, and (ii) for every node n of τ with sons $n_1, \ldots, n_k$ (with $k \geq 0$), there exists an instance of a deduction rule whose conclusion is n and whose premises are $n_1, \ldots, n_k$. A *proof* of $P \vdash g \rightarrow b$ is a finite deduction tree for $P \vdash g \rightarrow b$ where every leaf is a deduction rule which has no premises.

We stipulate that $P \vdash g \rightarrow b$ holds iff there exists a proof of $P \vdash g \rightarrow b$ when we consider the following set of deduction rules.

$$(tt)\quad \frac{}{P \vdash \mathit{true} \rightarrow \mathit{true}} \qquad\qquad (\mathit{ff})\quad \frac{}{P \vdash (\mathit{false},\ g) \rightarrow \mathit{false}}$$

$$(te1)\quad \frac{}{P \vdash (t_1 = t_2,\ g) \rightarrow \mathit{false}} \qquad \text{if } t_1 \text{ and } t_2 \text{ are not unifiable}$$

$$(te2)\quad \frac{P \vdash g\vartheta \rightarrow b}{P \vdash (t_1 = t_2,\ g) \rightarrow b} \qquad \text{if } t_1 \text{ and } t_2 \text{ are unifiable and } \vartheta = \mathit{mgu}(t_1, t_2)$$

$$(ge)\quad \frac{P \vdash g_2\{G/g_1\} \rightarrow b}{P \vdash ((G = g_1),\ g_2) \rightarrow b} \qquad \text{if } G \not\in \mathit{vars}(g_1)$$

$$(at)\quad \frac{P \vdash (\mathit{body}\{V_1/u_1, \ldots, V_m/u_m\},\ g) \rightarrow b}{P \vdash (p(u_1, \ldots, u_m),\ g) \rightarrow b} \qquad \begin{array}{l}\text{where } p(V_1, \ldots, V_m) \leftarrow \mathit{body} \text{ is}\\ \text{a renamed apart clause of } P\end{array}$$

$$(or)\quad \frac{P \vdash (g_1,\ g) \rightarrow b_1 \qquad P \vdash (g_2,\ g) \rightarrow b_2}{P \vdash ((g_1; g_2),\ g) \rightarrow b_1 \vee b_2}$$

If $P \vdash g \rightarrow \mathit{true}$ we say that g *succeeds* in P. If $P \vdash g \rightarrow \mathit{false}$ we say that g *fails* in P. If g neither succeeds nor fails in P we say that g is *undefined* in P.

Notes on semantics: (1) the rules $\dfrac{}{P \vdash \mathit{false} \rightarrow \mathit{false}}$ and $\dfrac{P \vdash g \rightarrow b}{P \vdash (\mathit{true},\ g) \rightarrow b}$ which the reader may expect by symmetry, are redundant, because they are derivable as instances of $\dfrac{}{P \vdash (\mathit{false},\ g) \rightarrow \mathit{false}}$ and $\dfrac{P \vdash g \rightarrow b}{P \vdash g \rightarrow b}$, respectively, by using the fact that *true* is the neutral element of a conjunction of goals. (2) $\mathit{mgu}(t_1, t_2)$ denotes a most general unifier of the terms t_1 and t_2. (3) $g\{V_1/u_1, \ldots, V_m/u_m\}$ denotes the substitution of $u_1, \ldots, u_m$ for $V_1, \ldots, V_m$ in g. (4) $\vee$ denotes the boolean or. (5) Our deduction rules for the operational semantics are mutually exclusive, and thus, there exists at most one proof for each $P \vdash g \rightarrow b$.

Our deduction rules extend the SLD-resolution for definite logic programs and a conjunction of goals is evaluated with the leftmost selection rule. The evaluation of a disjunction of goals requires the evaluation of each disjunct, and the existence of a proof for $P \vdash g \rightarrow b$ implies the *universal termination* of the goal g in the sense of [3]. (Recall that a goal g universally terminates in a program P iff all LD-derivations starting from g and using P, are finite.) It may be the case that for a given program P and goal g, no b exists such that $P \vdash g \rightarrow b$ holds, because every deduction tree with root $P \vdash g \rightarrow b$ *either* is infinite *or* has a leaf which is not an instance of any rule conclusion. The latter case may occur when the leaf is of the form: $P \vdash g_0 \rightarrow b_0$ and *either* g_0 is a goal variable *or* g_0 is of the form: $g_1 = g_2$ where the goal g_1 is not a goal variable or g_1 occurs in $vars(g_2)$. Thus, for instance, for any $b \in \{true, false\}$, there is no proof for $P \vdash G \rightarrow b$ and there is no proof for $P \vdash (G=p, G=q) \rightarrow b$.

Notice also that the computed answers semantics is *not* captured by our ternary relation $P \vdash g \rightarrow b$ and our semantic rules. This choice has been motivated by the fact that we want the unfolding rule to preserve semantics. Indeed, given the following two programs P_1 and P_2:

$$
\begin{array}{llcll}
P_1: & h(G) \leftarrow G=p,\, G & \qquad\qquad & P_2: & h(G) \leftarrow G=q,\, G \\
 & p \leftarrow q & & & p \leftarrow q \\
 & q \leftarrow & & & q \leftarrow
\end{array}
$$

where P_2 is obtained from P_1 by unfolding p, they are equivalent w.r.t. our operational semantics. However, P_1 and P_2 are not equivalent w.r.t. the computed answer semantics, because for the goal $h(G)$ program P_1 computes the answer $\{G/p\}$, whereas program P_2 computes the answer $\{G/q\}$.

4 The transformation rules

In this section we present the transformation rules for our language with goals as arguments. We assume that starting from an initial program P_0 we have constructed the *transformation sequence* $P_0, \ldots, P_i$ [10]. By an application of a transformation rule we get a new program P_{i+1}.

We use $g[u]$ to denote a goal g where we singled out an occurrence of its subconstruct u, where u may be either a term or a goal. By $g[\_]$ we denote the goal $g[u]$ without the occurrence of its subconstruct u. We say that $g[\_]$ is a *goal context*.

Rule R1 (Definition Introduction). It consists in deriving the new program P_{i+1} by adding to the current program P_i a new clause, called *definition clause*, of the form:

$newp(V_1, \ldots, V_m) \leftarrow g$

where: (i) *newp* is a new predicate symbol not occurring in any program of the sequence $P_0, \ldots, P_i$, (ii) the predicate symbols occurring in g occur in P_0, and (iii) $V_1, \ldots, V_m$ are distinct variables (possibly not all variables) which occur in g.
The set of all definition clauses introduced while constructing the transformation sequence $P_0, \ldots, P_i$, is denoted by Def_i.

Rule R2 (Unfolding). Let c: $h \leftarrow \mathit{body}[p(u_1, \ldots, u_m)]$ be a clause of program P_i. Let c_1: $p(V_1, \ldots, V_m) \leftarrow g$ be the unique, renamed apart clause of $P_0 \cup \mathit{Def}_i$ whose head predicate is p. By *unfolding c w.r.t. $p(u_1, \ldots, u_m)$ using c_1* we get the new clause: $h \leftarrow \mathit{body}[g\{V_1/u_1, \ldots, V_m/u_m\}]$. We also get the new program P_{i+1} by replacing in program P_i the clause c by that new clause.

This definition of the unfolding rule is an extension of the usual unfolding rule for definite logic programs and, indeed, for definite logic programs our unfolding rule corresponds to the application of a resolution step to clause c with the selection of the atom $p(u_1, \ldots, u_m)$ and the input clause c_1.

Rule R3 (Folding). Let c: $h \leftarrow \mathit{body}[g\vartheta]$ be a clause of program P_i and c_1: $\mathit{newp}(V_1, \ldots, V_m) \leftarrow g$ be the unique, renamed apart clause in Def_i with head predicate newp. Suppose that for every variable V in the set $\mathit{vars}(g) - \{V_1, \ldots, V_m\}$, we have:

- $V\vartheta$ is a variable which does not occur in c outside $g\vartheta$ and
- the variable $V\vartheta$ does not occur in $W\vartheta$, for any variable W occurring in g and different from V.

Then, by *folding c using c_1* we get the new program P_{i+1} by replacing in program P_i the clause c by the clause: $h \leftarrow \mathit{body}[p(V_1, \ldots, V_m)\vartheta]$.

In order to introduce the goal replacement rule (see rule R4 below) we have to introduce the so called *replacement laws*. The replacement of goal g_1 by goal g_2 in the body of a clause of the form: $h \leftarrow \mathit{body}[g_1]$, is allowed when a replacement law holds between g_1 and g_2. We will consider weak and strong replacement laws.

Given a program P, and goals g_1 and g_2, a *strong replacement law* $P \vdash g_1 \Longleftrightarrow\!\!\!> g_2$ holds iff
- g_1 and g_2 are equivalent in P w.r.t. the operational semantics defined in Section 3, that is, for every goal context $g[\_]$ and for every $b \in \{\mathit{true}, \mathit{false}\}$, $P \vdash g[g_1] \to b$ has a proof iff $P \vdash g[g_2] \to b$ has a proof, and
- the depth of the proof of $P \vdash g[g_2] \to b$ is not greater than the depth of the proof of $P \vdash g[g_1] \to b$ (see below for a rigorous definition of *proof depth*).

The latter condition makes $\Longleftrightarrow\!\!\!>$ to be an *improvement* relation in the sense of [15]. We will show in the Correctness Theorem of the next section, that the goal replacement rule based on strong replacement laws preserves our operational semantics. Indeed, if we derive program P_2 from program P_1 by an application of the goal replacement rule based on a strong replacement law, then P_1 and P_2 are equivalent in the sense that for every goal g and for every $b \in \{\mathit{true}, \mathit{false}\}$, we have that $P_1 \vdash g \to b$ iff $P_2 \vdash g \to b$.

For this result, we do use the assumption that the $\Longleftrightarrow\!\!\!>$ relation is an improvement relation. To see this, let us consider the following example:

$$P_1: \quad \begin{array}{l} p \leftarrow q \\ q \leftarrow \end{array} \qquad\qquad P_2: \quad \begin{array}{l} p \leftarrow p \\ q \leftarrow \end{array}$$

where P_2 is obtained by replacing q by p in the first clause of P_1. We have that p and q are equivalent in P_1, that is, for every goal context $g[\_]$ and for every $b \in \{true, false\}$, $P_1 \vdash g[p] \to b$ iff $P_1 \vdash g[q] \to b$, but for the empty goal context $[\_]$, the proof of $P_1 \vdash p \to true$ has greater depth than the proof of $P_1 \vdash q \to true$. In fact P_1 and P_2 are not operationally equivalent, because p succeeds in P_1, while p does not terminate in P_2.

We will also consider the notion of a *weak replacement law*, denoted $P \vdash g_1 \Longrightarrow\!\!\!\!\!> g_2$, for which we only require that, for every goal context $g[\_]$ and for every $b \in \{true, false\}$, if $P \vdash g[g_1] \to b$ has a proof then $P \vdash g[g_2] \to b$ has a proof which has not a greater depth.

For this rule we will prove a weaker correctness result, that is, if we derive program P_2 from program P_1 by an application of the goal replacement rule based on a weak replacement law, then for every goal g and for every $b \in \{true, false\}$, if $P_1 \vdash g \to b$ then $P_2 \vdash g \to b$. Thus, program P_2 may be *more defined* than program P_1, that is, there may be some goal g and some $b \in \{true, false\}$ such that $P_2 \vdash g \to b$ holds, while $P_1 \vdash g \to b$ has no proof. This notion is justified by the fact that we often want to apply transformations that may increase the termination of the program at hand, like for instance, the replacement of the goal $(g, false)$ by *false*, or the replacement of the goal $(g; true)$ by *true*.

In Definition 2 below, we refine the above notions of weak and strong replacement laws. This refinement is motivated by the fact that the correctness of the replacement of goal g_1 by goal g_2 in the clause c: $h \leftarrow body[g_1]$, does not depend on the *local variables* of g_1 and g_2 in c, where the local variables of goal g_i in c, for $i = 1, 2$, are defined to be those in the set $vars(g_i) - vars(h \leftarrow body[\_])$. For this reason we find convenient to introduce the notion of a goal g with local variables V, denoted $g \backslash V$.

Definition 1 (Proof Depth). Let π be the proof for $P \vdash g \to b$ and let m be the maximal number of instances of the deduction rule (at) in the root-to-leaf paths of π. Then we say that $P \vdash g \to b$ has a proof depth m, and we write $P \vdash g \to^m b$.

Definition 2 (Replacement Laws). Given a program P, let g_1 and g_2 be two goals, and let V_1 and V_2 be two sets of variables.
(i) $P \vdash g_1 \backslash V_1 \Longrightarrow g_2 \backslash V_2$ holds iff for every goal context $g[\_]$ such that $vars(g[\_]) \cap (V_1 \cup V_2) = \emptyset$, and for every $b \in \{true, false\}$, we have that:

if $P \vdash g[g_1] \to b$ then $P \vdash g[g_2] \to b$.

(ii) The relation $P \vdash g_1 \backslash V_1 \Longrightarrow\!\!\!\!\!> g_2 \backslash V_2$, called a *weak replacement law*, holds iff for every goal context $g[\_]$ such that $vars(g[\_]) \cap (V_1 \cup V_2) = \emptyset$, and for every $b \in \{true, false\}$, we have that:

if $P \vdash g[g_1] \to^m b$ then $P \vdash g[g_2] \to^n b$ with $m \geq n$.

(iii) The relation $P \vdash g_1 \backslash V_1 \Longleftarrow\!\!\!\!\!\Longrightarrow\!\!\!\!\!> g_2 \backslash V_2$, called a *strong replacement law*, holds iff $P \vdash g_1 \backslash V_1 \Longrightarrow\!\!\!\!\!> g_2 \backslash V_2$ and $P \vdash g_2 \backslash V_2 \Longrightarrow g_1 \backslash V_1$.

(iv) We write $P \vdash g_1\backslash V_1 \Longleftrightarrow\!\!\!\!\gg g_2\backslash V_2$ to mean that $P \vdash g_1\backslash V_1 \Longrightarrow\!\!\!\!\gg g_2\backslash V_2$ and $P \vdash g_2\backslash V_2 \Longrightarrow\!\!\!\!\gg g_1\backslash V_1$ (which is equivalent to $P \vdash g_1\backslash V_1 \Longleftrightarrow\!\!\!\gg g_2\backslash V_2$ and $P \vdash g_2\backslash V_2 \Longleftrightarrow\!\!\!\gg g_1\backslash V_1$). Thus, if $P \vdash g_1\backslash V_1 \Longleftrightarrow\!\!\!\!\gg g_2\backslash V_2$ then the proofs for $P \vdash g_1 \rightarrow b$ and for $P \vdash g_2 \rightarrow b$ have equal depth.

(v) If $V = \emptyset$ the goal $g\backslash V$ is also written as g. If V is the singleton $\{Z\}$, the goal $g\backslash V$ is also written as $g\backslash Z$.

For any program P we have that the following hold:

1. *Tautologies*:

$P \vdash true\,;\, g \Longrightarrow\!\!\!\!\gg true$	$P \vdash g_1\,;\, g_2 \Longleftrightarrow\!\!\!\!\gg g_2\,;\, g_1$
$P \vdash false\,,\, g \Longleftrightarrow\!\!\!\!\gg false$	$P \vdash (g_1\,,\, g_2)\,;\,(g_1\,,\, g_3) \Longleftrightarrow\!\!\!\!\gg g_1\,,\,(g_2\,;\, g_3)$
$P \vdash g\,,\, false \Longrightarrow\!\!\!\!\gg false$	$P \vdash (g_1\,,\, g_2)\,;\,(g_3\,,\, g_2) \Longleftrightarrow\!\!\!\!\gg (g_1\,;\, g_3)\,,\, g_2$
$P \vdash g\,,\, g \Longleftrightarrow\!\!\!\gg g$	$P \vdash (g_1\,;\, g_2)\,,\,(g_1\,;\, g_3) \Longrightarrow\!\!\!\!\gg g_1\,;\,(g_2\,,\, g_3)$
$P \vdash g\,;\, g \Longleftrightarrow\!\!\!\!\gg g$	

In the following replacement laws 2.a, 2.b, and 2.c, as usual for our conventions, V stands for either a goal variable or an individual variable, and u stands for either a term or a goal.

2.a *Generalization + equality introduction*:

$P \vdash g[u] \Longleftrightarrow\!\!\!\gg ((V\!=\!u),\ g[V])\backslash V$ if V is a variable not occurring in $g[u]$

2.b *Simplification of equalities*:

$P \vdash ((V\!=\!u),\ g[V])\backslash V \Longleftrightarrow\!\!\!\gg g[u]$ if V is a variable not occurring in $g[u]$

2.c *Rearrangement of equalities*:

$P \vdash (g[(V\!=\!u),\ g_1])\backslash V \Longleftrightarrow\!\!\!\!\gg ((V\!=\!u),\ g[g_1])\backslash V$ if V is a variable not occurring in $(g[\_], u)$

When referring to goal variables, the laws 2.a, 2.b, and 2.c will also be called 'goal generalization + equality introduction', 'simplification of goal equalities', and 'rearrangement of goal equalities', respectively.

3. *Rearrangement of term equalities*:

$P \vdash g,\ (t_1\!=\!t_2) \Longrightarrow\!\!\!\!\gg (t_1\!=\!t_2),\ g$

4. *Clark Equality Theory* (CET, see [8]):

$P \vdash eq_1\backslash Y \Longleftrightarrow\!\!\!\!\gg eq_2\backslash Z$ if $\mathrm{CET} \vdash \forall X\ (\exists Y\ eq_1 \leftrightarrow \exists Z\ eq_2)$

where: (i) eq_1 and eq_2 are goals constructed using *true*, *false*, term equalities, conjunctions, and disjunctions, and (ii) $X = vars(eq_1, eq_2) - (Y \cup Z)$.

Notice that, for some program P and for some goals g, g_1, g_2, and g_3, the following do *not* hold:

$P \vdash true \Longrightarrow true\,;\, g$
$P \vdash false \Longrightarrow g\,,\, false$
$P \vdash (t_1\!=\!t_2)\,,\, g \Longrightarrow g\,,\,(t_1\!=\!t_2)$
$P \vdash g_1\,;\,(g_2\,,\, g_3) \Longrightarrow (g_1\,;\, g_2)\,,\,(g_1\,;\, g_3)$
$P \vdash g_2[g_1] \Longrightarrow g_2[G]\,,\,(G\!=\!g_1)$
$P \vdash g[(G\!=\!g_1)\,,\, g_2] \Longrightarrow (G\!=\!g_1)\,,\, g[g_2]$

Rule R4 (Goal Replacement). We get program P_{i+1} from program P_i by replacing clause: $h \leftarrow \mathit{body}[g_1]$ by clause: $h \leftarrow \mathit{body}[g_2]$, if all predicate symbols of (g_1, g_2) occur in P_0, and either (i) $P_0 \vdash g_1 \backslash V_1 \Longrightarrow\!\!\!\!\!\!\succ g_2 \backslash V_2$, or (ii) $P_0 \vdash g_1 \backslash V_1 \Longleftrightarrow\!\!\!\!\!\!\succ g_2 \backslash V_2$, where $V_1 = \mathit{vars}(g_1) - \mathit{vars}(h, \mathit{body}[\_])$, and $V_2 = \mathit{vars}(g_2) - \mathit{vars}(h, \mathit{body}[\_])$.
In case (i) we say that the goal replacement is *based on a weak replacement law*. In case (ii) we say that the goal replacement is *based on a strong replacement law*.

Lemma 1 (Replacement). Let g_1, g_2 be goals and V_1, V_2, W be sets of variables. Let $g[\_]$ be a goal context such that $\mathit{vars}(g[\_]) \cap (V_1 \cup V_2) = \emptyset$.
(i) If $P \vdash g_1 \backslash V_1 \Longrightarrow g_2 \backslash V_2$ then $P \vdash g[g_1] \backslash (V_1 \cup W) \Longrightarrow g[g_2] \backslash (V_2 \cup W)$.
(ii) If $P \vdash g_1 \backslash V_1 \Longrightarrow\!\!\!\!\!\!\succ g_2 \backslash V_2$ then $P \vdash g[g_1] \backslash (V_1 \cup W) \Longrightarrow\!\!\!\!\!\!\succ g[g_2] \backslash (V_2 \cup W)$.

5 Correctness of the transformation rules

This section is devoted to the proof of the correctness of our transformation rules. We first introduce the notion of *parallel leftmost unfolding* of a clause c.

Let c be a clause in a program P. If c is of the form:

$$p(V_1, \ldots, V_m) \leftarrow (a_1, g_1); \ldots; (a_k, g_k) \qquad (*)$$

where $k > 0$ and $a_1, \ldots, a_k$ are atoms, then the *parallel leftmost unfolding* of clause c in program P is the program Q obtained from P by k applications of the unfolding rule w.r.t. $a_1, \ldots, a_k$, respectively.

If the clause c is not of the form (*), then the parallel leftmost unfolding of c is not defined.

Theorem 1 (Correctness). Let us consider a transformation sequence $P_0, \ldots, P_k$ constructed by using the rules R1, R2, R3, and R4. Let us suppose that for every h, with $0 \leq h < k$, if P_{h+1} has been obtained from P_h by folding clause c using clause c_1, then there exist i, j, with $0 \leq i < j \leq h$, such that P_j is obtained from P_i by parallel leftmost unfolding of c_1.
Then for every goal g and for every $b \in \{\mathit{true}, \mathit{false}\}$, we have that:

(1) *Improvement*: if $P_0 \cup \mathit{Def}_k \vdash g \rightarrow^m b$ then $P_k \vdash g \rightarrow^n b$ with $m \geq n$, and

(2) *Soundness*: if all applications of the goal replacement rule are based on strong replacement laws, then: if $P_k \vdash g \rightarrow b$ then $P_0 \cup \mathit{Def}_k \vdash g \rightarrow b$.

Notice that according to our Correctness Theorem, if some applications of the goal replacement rule are not based on strong replacement laws, then the transformed program P_k may be *more defined* than program $P_0 \cup \mathit{Def}_k$, that is, there may be some goal g and some $b \in \{\mathit{true}, \mathit{false}\}$ such that $P_k \vdash g \rightarrow b$ holds, while $P_0 \cup \mathit{Def}_k \vdash g \rightarrow b$ has no proof. This fact may be due to applications of the goal replacement rule based on weak replacement laws, such as $P_i \vdash g, \mathit{false} \Longrightarrow\!\!\!\!\!\!\succ \mathit{false}$, for which the $\Longleftrightarrow\!\!\!\!\!\!\succ$ relation does not hold.

Part (1) of the Correctness Theorem follows from Lemmata 2 and 3 below.

Lemma 2 (Improvement). Let P and $NewP$ be programs of the form:

$$
\begin{array}{llll}
P: & hd_1 \leftarrow bd_1 & \quad NewP: & hd_1 \leftarrow newbd_1 \\
 & \vdots & & \vdots \\
 & hd_k \leftarrow bd_k & & hd_k \leftarrow newbd_k
\end{array}
$$

For $r = 1, \ldots, k$, let $Y = vars(bd_r) - vars(hd_r)$ and $Z = vars(newbd_r) - vars(hd_r)$ and suppose that: $P \vdash bd_r \backslash Y \Longrightarrow newbd_r \backslash Z$.

Then for all goals g, we have:

if $P \vdash g \rightarrow^m b$ then $NewP \vdash g \rightarrow^n b$ with $m \geq n$.

Proof. We define the size of a proof to be the number of its nodes. We then define a measure μ from the set of proofs to the set $N \times N$ of pairs of natural numbers ordered lexicographically (that is, $\langle m1, m2\rangle < \langle n1, n2\rangle$ iff *either* $m1 < n1$ *or* ($m1 = n1$ and $m2 < n2$)). We stipulate that $\mu(\pi) = \langle m, s\rangle$, where m is the depth of π and s is the size of π. We prove this lemma by induction w.r.t. the measure of the proof of $P \vdash g \rightarrow^m b$ (which, by definition, has depth m).

We assume that, for all $\langle m1, s1\rangle < \langle m, s\rangle$, for all goals g, and for all $b \in \{true, false\}$, if $P \vdash g \rightarrow b$ has a proof of depth $m1$ and size $s1$, then $NewP \vdash g \rightarrow b$ has a proof of depth $n1$, with $m1 \geq n1$. We have to show that $NewP \vdash g \rightarrow b$ has a proof of depth n, with $m \geq n$. We then proceed by cases on the structure of g. We consider the following two cases. The other cases are similar and we omit them.

- Case 1: g is $((G = g_1), g_2)$. Assume that $P \vdash ((G = g_1), g_2) \rightarrow b$ has a proof of depth m and size s. Then, $G \notin vars(g_1)$, $P \vdash ((G = g_1), g_2) \rightarrow b$ has been proved by using rule (ge) as the last step, and $P \vdash g_2\{G/g_1\} \rightarrow b$ has a proof of depth m and size $s-1$. Since $\langle m, s-1\rangle < \langle m, s\rangle$, by the induction hypothesis $NewP \vdash g_2\{G/g_1\} \rightarrow b$ has a proof of depth n with $m \geq n$ and, by rule (ge), we have that $NewP \vdash ((G = g_1), g_2) \rightarrow b$ has a proof of depth n with $m \geq n$.
- Case 2: g is $(p(u_1, \ldots, u_m), g_1)$. Assume that $P \vdash (p(u_1, \ldots, u_m), g_1) \rightarrow b$ has a proof of depth m and size s. Then, the last step of the proof of $P \vdash (p(u_1, \ldots, u_m), g_1) \rightarrow b$ has been performed by using rule (at), and $P \vdash (bd_r\{V_1/u_1, \ldots, V_m/u_m\}, g_1) \rightarrow b$ has a proof of depth $m-1$ and size $s-1$, where $p(V_1, \ldots, V_m) \leftarrow bd_r$ is a renamed apart clause of P and $p(V_1, \ldots, V_m)$ is hd_r. Now, by (i) the hypothesis that $P \vdash bd_r \backslash Y \Longrightarrow newbd_r \backslash Z$ and (ii) the fact that the substitution $\{V_1/u_1, \ldots, V_m/u_m\}$ does not affect the variables in $Y \cup Z$, we have that $P \vdash (newbd_r\{V_1/u_1, \ldots, V_m/u_m\}, g_1) \rightarrow b$ has a proof of depth $n1$ and size $s1$, with $m-1 \geq n1$. Since $\langle n1, s1\rangle < \langle m, s\rangle$, by the induction hypothesis there exists a proof of $NewP \vdash (newbd_r\{V_1/u_1, \ldots, V_m/u_m\}, g_1) \rightarrow b$ of depth $n2$ with $n1 \geq n2$. Thus, by using rule (at), we can build a proof of $NewP \vdash (p(u_1, \ldots, u_m), g_1) \rightarrow b$ of depth $n = n2+1$, and thus, $m \geq n$. □

In order to simplify the proof of our next Lemma 3, we notice that we may rearrange any transformation sequence $P_0, \ldots, P_k$ constructed according to the hypothesis of the Correctness Theorem, into a new sequence $P_0, \ldots, P_0 \cup Def_k, \ldots, P_j, \ldots, P_k$ such that: (1) $P_0, \ldots, P_0 \cup Def_k$ is constructed by applications of the definition rule only, and (2) $P_0 \cup Def_k, \ldots, P_j$ is constructed by parallel leftmost unfolding of every clause d in Def_k which is used for an application of the folding rule in the original sequence $P_0, \ldots, P_k$.

Lemma 3. Let us consider a transformation sequence $P_0, \ldots, P_i, \ldots, P_j, \ldots, P_k$ such that: (1) $P_0, \ldots, P_i$ is constructed by applications of the definition rule only, (2) $P_i = P_0 \cup Def_k$, and (3) $P_i, \ldots, P_j$ is constructed by parallel leftmost unfolding of every $d \in Def_k$ which is used for an application of the folding rule in $P_j, \ldots, P_k$. For $h = i, \ldots, k-1$, let $hd \leftarrow newbd^h$ be a clause in program P_{h+1} derived from the clause $hd \leftarrow bd^h$ in P_h, by an application of the unfolding or folding or goal replacement rule, and let Y be $vars(bd^h) - vars(hd)$ and Z be $vars(newbd^h) - vars(hd)$.

Then, (i) for $h = i, \ldots, j-1$, $P_0 \cup Def_k \vdash bd^h \backslash Y \Longrightarrow\!\!\!\!> newbd^h \backslash Z$, and

(ii) for $h = j, \ldots, k-1$, $P_j \vdash bd^h \backslash Y \Longrightarrow\!\!\!\!> newbd^h \backslash Z$.

Proof. (i) For every clause $p(V_1, \ldots, V_m) \leftarrow g$ in $P_0 \cup Def_k$, we have that: $P_0 \cup Def_k \vdash p(V_1, \ldots, V_m) \Longrightarrow\!\!\!\!> g \backslash Z$, where $Z = vars(g) - \{V_1, \ldots, V_m\}$. Thus, Point (i) of the thesis follows from Point (ii) of the Replacement Lemma and the fact that for $h = i, \ldots, j-1$, each clause in P_{h+1} is derived from a clause in P_h by the unfolding rule.

(ii) For every definition clause d: $newp(V_1, \ldots, V_m) \leftarrow g$ in Def_k which is used for folding in the sequence $P_j, \ldots, P_k$, we have that:
(†) $P_j \vdash newp(V_1, \ldots, V_m) <\!\!\!\!\Longleftarrow\!\!\!\Longrightarrow\!\!\!\!> g \backslash Z$, where $Z = vars(g) - \{V_1, \ldots, V_m\}$. Property (†) is a consequence of the fact that P_j is derived from $P_0 \cup Def_k$ by parallel leftmost unfolding of a set of definition clauses among which is d.

Moreover, for every clause c: $p(V_1, \ldots, V_m) \leftarrow g$ in P_0, we have that:
(‡) $P_j \vdash p(V_1, \ldots, V_m) \Longrightarrow\!\!\!\!> g \backslash Z$, where $Z = vars(g) - \{V_1, \ldots, V_m\}$, because $c \in P_j$.

For $h = j, \ldots, k-1$, (1) if P_{h+1} is derived from P_h by the unfolding rule using a clause $newp(V_1, \ldots, V_m) \leftarrow g$ which is among those also used for folding (in previous transformation steps), then the thesis follows from Point (ii) of the Replacement Lemma and (†).

(2) If P_{h+1} is derived from P_h by the unfolding rule using a clause c of the form $p(V_1, \ldots, V_m) \leftarrow g$ which is *not* among those used for folding, then c belongs to P_0. Thus, the thesis follows from (‡) and Point (ii) of the Replacement Lemma.

(3) If P_{h+1} is derived from P_h by the folding rule using a clause of the form $newp(V_1, \ldots, V_m) \leftarrow g$, then the thesis follows from (†) and Point (ii) of the Replacement Lemma.

Finally, (4) if P_{h+1} is derived from P_h by the goal replacement rule based on a replacement law of the form $P_0 \vdash g_1 \backslash Y \Longrightarrow\!\!\!\!> g_2 \backslash Z$, then Point (ii) of the thesis follows from Point (ii) of the Replacement Lemma and the fact that $P_j \vdash g_1 \backslash Y \Longrightarrow\!\!\!\!> g_2 \backslash Z$ also holds, because the predicates of g_1 and g_2 occur in P_0 and the definitions of the predicates in P_0 are not modified during the transformation sequence $P_0, \ldots, P_j$. □

In order to get the proof of Part (1) of the Correctness Theorem, it is sufficient to notice that: (A) by Lemma 2 and Point (i) of Lemma 3, we have that: for all goals g, if $P_0 \cup Def_k \vdash g \rightarrow^m b$ then $P_j \vdash g \rightarrow^r b$ with $m \geq r$, where

P_j is the program constructed as indicated in Lemma 3, and (B) by Lemma 2 and Point (ii) of Lemma 3, we have that: for all goals g, if $P_j \vdash g \rightarrow^r b$ then $P_k \vdash g \rightarrow^n b$ with $r \geq n$.

The proof of Part (2) of the Correctness Theorem is based on the following Lemmata 4 and 5.

Lemma 4 (Soundness). Let P and *NewP* be programs of the form:

$$\begin{array}{llcll} P: & hd_1 \leftarrow bd_1 & \qquad & \mathit{NewP}: & hd_1 \leftarrow newbd_1 \\ & \vdots & & & \vdots \\ & hd_k \leftarrow bd_k & & & hd_k \leftarrow newbd_k \end{array}$$

For $r = 1, \ldots, k$, let $Y = vars(bd_r) - vars(hd_r)$ and $Z = vars(newbd_r) - vars(hd_r)$, and suppose that: $P \vdash newbd_r \backslash Z \Longrightarrow bd_r \backslash Y$.

Then for all goals g, we have: if $\mathit{NewP} \vdash g \rightarrow b$ then $P \vdash g \rightarrow b$.

Proof. We proceed by induction on the size of the proof of $\mathit{NewP} \vdash g \rightarrow b$. Assume that, for all $m < n$ and for all goals g, if $\mathit{NewP} \vdash g \rightarrow b$ has a proof of size m then $P \vdash g \rightarrow b$ has a proof.

We now proceed by cases on the structure of g. We consider the following two cases. The other cases are similar and we omit them.

- Case 1: g is $((G = g_1), g_2)$. Assume that $\mathit{NewP} \vdash ((G = g_1), g_2) \rightarrow b$ has a proof of size n. Then, $G \notin vars(g_1)$, $\mathit{NewP} \vdash ((G = g_1), g_2) \rightarrow b$ has been proved by using rule (ge) as the last step, and there exists a proof of size $n-1$ of $\mathit{NewP} \vdash g_2\{G/g_1\} \rightarrow b$. By the induction hypothesis there exists a proof of $P \vdash g_2\{G/g_1\} \rightarrow b$ and, by rule (ge), we have that $P \vdash ((G = g_1), g_2) \rightarrow b$ has a proof.

- Case 2: g is $(p(u_1, \ldots, u_m), g_1)$. Assume that $\mathit{NewP} \vdash (p(u_1, \ldots, u_m), g_1) \rightarrow b$ has a proof of size n. Then, $\mathit{NewP} \vdash (p(u_1, \ldots, u_m), g_1) \rightarrow b$ has been proved by using rule (at) as the last step, and there exists a proof of size $n-1$ of $\mathit{NewP} \vdash (newbd_r\{V_1/u_1, \ldots, V_m/u_m\}, g_1) \rightarrow b$ where $p(V_1, \ldots, V_m) \leftarrow newbd_r$ is a renamed apart clause of *NewP* and $p(V_1, \ldots, V_m)$ is hd_r. By the induction hypothesis there exists a proof of $P \vdash (newbd_r\{V_1/u_1, \ldots, V_m/u_m\}, g_1) \rightarrow b$. Now, by the hypothesis that $P \vdash newbd_r \backslash Z \Longrightarrow bd_r \backslash Y$ and the fact that the substitution $\{V_1/u_1, \ldots, V_m/u_m\}$ does not affect the variables in $Y \cup Z$, we have that $P \vdash (bd_r\{V_1/u_1, \ldots, V_m/u_m\}, g_1) \rightarrow b$ has a proof. Thus, by rule (at), $P \vdash (p(u_1, \ldots, u_m), g_1) \rightarrow b$ has a proof. □

Lemma 5. Let us consider a transformation sequence $P_0, \ldots, P_k$. For $i = 0, \ldots, k-1$, let $hd \leftarrow newbd$ be a clause in program P_{i+1} derived from the clause $hd \leftarrow bd$ in P_i, by an application of the unfolding rule, or folding rule, or goal replacement rule based on a strong replacement law. Then $P_0 \cup Def_k \vdash newbd \backslash Z \Longrightarrow bd \backslash Y$, where $Y = vars(bd) - vars(hd)$ and $Z = vars(newbd) - vars(hd)$.

Proof. If P_{i+1} is derived from P_i by the unfolding rule using a clause c: $p(V_1, \ldots, V_m) \leftarrow g$ in $P_0 \cup Def_k$, then the thesis follows from Point (i) of the Replacement Lemma and the fact that $P_0 \cup Def_k \vdash g \backslash Z \Longrightarrow p(V_1, \ldots, V_m)$, where $Z = vars(g) - \{V_1, \ldots, V_m\}$. Similarly, if P_{i+1} is derived from P_i by the folding rule using clause d: $newp(V_1, \ldots, V_m) \leftarrow g$ in Def_k, then the thesis follows from Point (i) of the Replacement Lemma and the fact that $P_0 \cup Def_k \vdash newp(V_1, \ldots, V_m) \Longrightarrow g \backslash Z$, where $Z = vars(g) - \{V_1, \ldots, V_m\}$. Finally, if P_{i+1} is derived from P_i by the goal replacement rule, then the thesis follows from the fact it is based on a strong replacement law and from Point (i) of the Replacement Lemma. □

6 Goals as arguments for avoiding incorrect goal rearrangements

In this section we illustrate through an example borrowed from [7], the use of our logic language and our transformation rules. This example indicates how by allowing goals as arguments we can solve the goal rearrangement problem, in the sense that when transforming programs for eliminating multiple traversals of data structures, we can avoid the need of rearranging atoms in the body of clauses.

Let us consider the initial program P_0:

1. $\mathit{flipcheck}(X,Y) \leftarrow \mathit{flip}(X,Y),\ \mathit{check}(Y)$
2. $\mathit{flip}(X,Y) \leftarrow (X = l(N),\ Y = l(N))\,;$
   $(X = t(L,N,R),\ Y = t(FR,N,FL),\ \mathit{flip}(L,FL),\ \mathit{flip}(R,FR))$
3. $\mathit{check}(X) \leftarrow (X = l(N),\ \mathit{nat}(N))\,;$
   $(X = t(L,N,R),\ \mathit{nat}(N),\ \mathit{check}(L),\ \mathit{check}(R))$
4. $\mathit{nat}(X) \leftarrow (X = 0)\,;$
   $(X = s(N), \mathit{nat}(N))$

where: (i) the term $l(N)$ denotes a leaf with label N and the term $t(L,N,R)$ denotes a tree with label N and the two subtrees L and R, (ii) $\mathit{nat}(X)$ holds iff X is a natural number, (iii) $\mathit{check}(X)$ holds iff all labels in the tree X are natural numbers, and (iv) $\mathit{flip}(X,Y)$ holds iff the tree Y can be obtained by flipping all subtrees of the tree X.

We would like to transform this program and avoid multiple traversals of trees (see the double occurrence of Y in the body of clause 1). We may do so by following two approaches. The first approach is to apply the folding/unfolding rules and the strategy for the *elimination of unnecessary variables* described in [13] or, equivalently, the *conjunctive partial deduction* technique [7] (with the obvious adaptations due to the fact that in our initial program we avoid multiple clauses and use the operator ';' instead). The second approach is to apply the transformation rules we have proposed in Section 4 and to use programs with goals as arguments.

Below we will present the corresponding two derivations. In both derivations we get a program which makes one traversal only of the input trees. However, the first derivation which does *not* use goals as arguments, produces a program, say P_f, which is *not* correct, while the second derivation which uses goals as arguments produces a program, say P_h, which is correct. We have that P_f is *not* correct because during the transformation from P_0 to P_f, we do not preserve universal termination [3]. Indeed, there exists a goal g such that LD-resolution terminates for goal g and program P_0, while it does not terminate for goal g and program P_f (obviously, in order to apply LD-resolution we have to view programs P_0 and P_f as definite logic programs by eliminating ';' in favour of multiple clauses with the same head predicate).

Here is the first derivation from program P_0 to program P_f. In this first derivation we silently assume that clause 1 has been derived from the initial clauses 2, 3, and 4 by applying the definition introduction rule. This allows us to use clause 1 for performing folding steps, and also it allows us to state our correctness results using P_0, instead of $P_0 \cup Def_k$.

We first unfold twice clause 1 and we apply the goal replacement rule whereby deriving the clause:

$$\begin{aligned} 5.\ \mathit{flipcheck}(X,Y) \leftarrow\ & (X = l(N),\ Y = l(N),\ \mathit{nat}(N))\,; \\ & (X = t(L,N,R),\ Y = t(FR,N,FL), \\ & \ \mathit{flip}(L,FL),\ \mathit{flip}(R,FR),\ \mathit{nat}(N),\ \mathit{check}(FR),\ \mathit{check}(FL)) \end{aligned}$$

Then we perform goal rearrangements, which unfortunately do not preserve our operational semantics, and we get:

$$\begin{aligned} 6.\ \mathit{flipcheck}(X,Y) \leftarrow\ & (X = l(N),\ Y = l(N),\ \mathit{nat}(N))\,; \\ & (X = t(L,N,R),\ Y = t(FR,N,FL), \\ & \mathit{nat}(N),\ \mathit{flip}(L,FL),\ \mathit{check}(FL),\ \mathit{flip}(R,FR),\ \mathit{check}(FR)) \end{aligned}$$

Now we can fold twice clause 6 using clause 1 and we get the final program P_f consisting of clause 4 together with the following clause:

$$\begin{aligned} 7.\ \mathit{flipcheck}(X,Y) \leftarrow\ & (X = l(N),\ Y = l(N),\ \mathit{nat}(N))\,; \\ & (X = t(L,N,R),\ Y = t(FR,N,FL), \\ & \mathit{nat}(N),\ \mathit{flipcheck}(L,FL),\ \mathit{flipcheck}(R,FR)) \end{aligned}$$

Program P_f performs one traversal only of any input tree given as first argument of *flipcheck*. However, as already mentioned, we have not preserved universal termination. Indeed, the goal $\mathit{flipcheck}(t(l(N),0,l(a)),Y)$ fails in P_0 (that is, $P_0 \vdash \mathit{flipcheck}(t(l(N),0,l(a)),Y) \rightarrow \mathit{false}$ holds), while it is undefined in P_f (that is, $P_f \vdash \mathit{flipcheck}(t(l(N),0,l(a)),Y) \rightarrow b$ does not hold for any $b \in \{\mathit{true}, \mathit{false}\}$).

Now we present a second derivation starting from the same initial program P_0 and producing a final program P_h which traverses the input tree only once. This derivation is correct in the sense of Point (1) of Theorem 1. In particular, for every goal g of the form: $\mathit{flipcheck}(t_1,t_2)$ where t_1 and t_2 are any two terms, and for every $b \in \{\mathit{true}, \mathit{false}\}$, we have that: if $P_0 \vdash g \rightarrow b$ holds, then $P_h \vdash g \rightarrow b$ holds. During this second derivation we introduce goal arguments and we make use of the transformation rules introduced in Section 4. The initial step of this derivation is the introduction of the following new definition:

$$8.\ \mathit{newp}(X,Y,G) \leftarrow \mathit{flip}(X,Y),\ G = \mathit{check}(Y)$$

Before continuing our second derivation, let us motivate the introduction of this clause 8 (although this issue is not related to the correctness of our derivation). The incorrect goal rearrangement steps which have led from clause 5 to clause 6, can be avoided by applying, instead, (i) goal generalization + equality introduction and (ii) rearrangement of goal equalities, which are instances of the goal replacement rule, and thus, preserve correctness. Indeed, from clause 5 we can derive by goal generalization + equality introduction the following clause where we have introduced the goal variables GL and GR:

$$\begin{aligned}
9.\ \mathit{flipcheck}(X,Y) \leftarrow\ & (X=l(N),\ Y=l(N),\ \mathit{nat}(N))\,;\\
& (X=t(L,N,R),\ Y=t(FR,N,FL),\\
& \mathit{flip}(L,FL),\ \mathit{flip}(R,FR),\\
& \mathit{nat}(N),\ GR=\mathit{check}(FR),\ GL=\mathit{check}(FL),\ GR,\ GL)
\end{aligned}$$

and then we get the following clause by rearranging the goal equalities:

$$\begin{aligned}
10.\ \mathit{flipcheck}(X,Y) \leftarrow\ & (X=l(N),\ Y=l(N),\ \mathit{nat}(N))\,;\\
& (X=t(L,N,R),\ Y=t(FR,N,FL),\\
& \mathit{flip}(L,FL),\ GL=\mathit{check}(FL),\\
& \mathit{flip}(R,FR),\ GR=\mathit{check}(FR),\\
& \mathit{nat}(N),\ GR,\ GL)
\end{aligned}$$

Now, this clause 10 explains why we have introduced clause 8. Indeed, by using clause 8 we can avoid the intermediate variables FL and FR which occur in clause 10 by folding each *flip* atom with the adjacent *check* atom.

Let us then continue our second derivation starting from clause 8 together with clauses 1, 2, 3, and 4. From clause 8 we perform the transformation steps analogous to those performed in the derivation leading from clause 1 to clause 10. We first unfold clause 8 w.r.t. $\mathit{flip}(X,Y)$ and we get:

$$\begin{aligned}
11.\ \mathit{newp}(X,Y,G) \leftarrow\ & ((X=l(N),\ Y=l(N))\,;\\
& \ (X=t(L,N,R),\ Y=t(FR,N,FL),\\
& \ \mathit{flip}(L,FL),\ \mathit{flip}(R,FR))),\\
& \ G=\mathit{check}(Y)
\end{aligned}$$

We then unfold the goal argument $\mathit{check}(Y)$, and after some applications of the goal replacement rule based on weak replacement laws, we get:

$$\begin{aligned}
12.\ \mathit{newp}(X,Y,G) \leftarrow\ & (X=l(N),\ Y=l(N),\ G=\mathit{nat}(N))\,;\\
& (X=t(L,N,R),\ Y=t(FR,N,FL),\\
& \mathit{flip}(L,FL),\ \mathit{flip}(R,FR),\\
& G=(\mathit{nat}(N),\mathit{check}(FR),\mathit{check}(FL)))
\end{aligned}$$

We apply the goal generalization + equality introduction rule and the rearrangement of goal equalities rule, and we get:

$$\begin{aligned}
13.\ \mathit{newp}(X,Y,G) \leftarrow\ & (X=l(N),\ Y=l(N),\ G=\mathit{nat}(N))\,;\\
& (X=t(L,N,R),\ Y=t(FR,N,FL),\\
& \mathit{flip}(L,FL),\ V=\mathit{check}(FL),\ \mathit{flip}(R,FR),\ U=\mathit{check}(FR),\\
& G=(\mathit{nat}(N),U,V))
\end{aligned}$$

Now we can fold clause 13 using clause 8 and we get:

$$\begin{aligned}
13.\text{f}\ \mathit{newp}(X,Y,G) \leftarrow\ & (X=l(N),\ Y=l(N),\ G=\mathit{nat}(N))\,;\\
& (X=t(L,N,R),\ Y=t(FR,N,FL),
\end{aligned}$$

$$newp(L, FL, V),\ newp(R, FR, U),\ G = (nat(N), U, V))$$

In order to express *flipcheck* in terms of *newp* we apply the goal generalization + equality introduction rule to clause 1 and we derive:

14. $flipcheck(X, Y) \leftarrow flip(X, Y),\ G = check(Y),\ G$

Then we fold clause 14 using clause 8 and we get:

14.f $flipcheck(X, Y) \leftarrow newp(X, Y, G),\ G$

Thus, the final program P_h consists of the following three clauses:

$$\begin{array}{ll}
\text{14.f} & flipcheck(X, Y) \leftarrow newp(X, Y, G),\ G \\
\text{13.f} & newp(X, Y, G) \leftarrow (X = l(N),\ Y = l(N),\ G = nat(N))\,; \\
& \qquad\qquad\qquad\quad (X = t(L, N, R),\ Y = t(FR, N, FL), \\
& \qquad\qquad\qquad\quad newp(L, FL, V),\ newp(R, FR, U),\ G = (nat(N), U, V)) \\
\text{4.} & nat(X) \leftarrow (X = 0)\,; \\
& \qquad\quad (X = s(N), nat(N))
\end{array}$$

Program P_h traverses the input tree only once, because of the structure of the recursive calls of the predicate *newp*. Moreover, the Correctness Theorem ensures that if a goal in the initial program P_0 either succeeds or fails, then also in the final program P_h it either succeeds or fails, respectively. However, a goal which in program P_0 is undefined, may either succeed or fail in program P_h, because as already mentioned, in the derivation we have used replacement laws which are not strong (see the derivation of clause 12 from clause 11).

7 Final Remarks and Related Work

We have proposed a logic language where goals may appear as arguments of predicate symbols. The idea is not novel and, in particular, in a previous paper of ours [11] we have already used goal generalization and we have allowed the use of variables which are bound to goals. Here we have formally introduced the syntax and the operational semantics of a language with goals as arguments and goal variables. The operational semantics we have introduced, extends the familiar operational semantics for definite logic programs with the leftmost selection rule. We have also presented a set of transformation rules for our logic language and we have shown their correctness w.r.t. the given operational semantics. We have shown that variables which range over goals are useful in the context of program transformation for the derivation of efficient logic programs and the use of these variables may avoid the need for atom rearrangement during program transformation, in particular before folding steps. Our extension of definite logic programs makes it possible to perform in the case of logic programs, the kind of transformations which are done in the case of functional programming through the use of higher order generalizations (or lambda abstractions) [12] and *continuations* [18]. For instance, with reference to the example of Section 6, we have that in clause 14.f the third argument G of the predicate *newp* plays the role of a continuation, and indeed, the evaluation of $flipcheck(X, Y)$ consists of the evaluation of the goal $newp(X,Y,G)$ followed by the evaluation of the continuation which is the goal bound to G.

There are several other proposals in the literature for higher order logic languages (see [6, 9] for recent surveys). Our main contribution in this paper is a transformational approach to the development of higher order logic programs, which we believe, has received very little attention so far. Notice that, as already mentioned, our language has only limited higher order capabilities, because quantified function or predicate variables are not allowed.

The Correctness Theorem we have proved about our transformation rules ensures that the if a goal succeeds or fails in the given program then also in the derived program it succeeds or fails, respectively. Since the operational semantics defined in Section 3 models universal termination (that is, the operational semantics of a goal is defined iff all LD-derivations starting from that goal are finite), in this respect our Correctness Theorem extends the results for definite logic programs presented in [3]. (In fact, the transformations presented in [3] also preserve computed answers while ours do not.) Also we would like to notice that our transformation rules extend those considered in [3] because: (i) our language is an extension of definite logic programs, and (ii) our folding rule is more powerful. Indeed, even restricting ourselves to programs that do not contain goal variables and goal arguments, we allow folding steps which use clauses whose bodies contain disjunctions, and this is not possible in [3], where for folding one may use exactly one definition clause whose body is a conjunction of atoms.

The approach we have proposed in the present paper to avoid incorrect goal rearrangements, is complementary to the approach described in [4], where the authors give sufficient conditions for the goal rearrangements to preserve *left termination.* (Recall that a program P left terminates iff all *ground* goals universally terminate in P.) Thus, when these conditions are not met or we are not able to prove their validity, we may apply our technique which avoids incorrect goal rearrangements by using, instead, goal generalization + equality introduction and rearrangement of goal equalities. We have proved in this paper that these transformation rules preserve universal termination, and thus, they also preserve left termination.

Notice, however, that sometimes goal rearrangements may be desirable because they improve program performance by anticipating tests and reducing the time complexity of the execution of logic programs.

Finally, our idea of introducing replacement laws which are based on a notion of improvement has been already considered by various authors in various contexts such as higher order functional languages [15], concurrent constraint logic languages [2], and inductive definitions [14].

We leave for future work the development of suitable strategies for directing the use of the transformation rules we have proposed in this paper.

Acknowledgements

We would like to thank Michael Leuschel for pointing out an error in the preliminary version of this paper and also for helpful comments. Thanks also to the anonymous referees of the LoPSTr 99 Workshop for their suggestions. This work has been partially supported by MURST Progetto Cofinanziato 'Tecniche Formali per la Specifica, l'Analisi, la Verifica, la Sintesi e la Trasformazione di Sistemi Software' (Italy), and Progetto Coordinato CNR 'Verifica, Analisi e Trasformazione dei Programmi Logici' (Italy).

References

[1] K. R. Apt and D. Pedreschi. Reasoning about termination of pure logic programs. *Information and Computation*, 106:109–157, 1993.

[2] M. Bertolino, S. Etalle, and C. Palamidessi. The replacement operation for CCP programs. In A. Bossi, editor, *Proceedings of LoPSTr '99, Venice, Italy*, Lecture Notes in Computer Science. Springer, 2000. (this volume).

[3] A. Bossi and N. Cocco. Preserving universal termination through unfold/fold. In *Proceedings ALP '94*, Lecture Notes in Computer Science 850, pages 269–286. Springer-Verlag, 1994.

[4] A. Bossi, N. Cocco, and S. Etalle. Transforming left-terminating programs: The reordering problem. In M. Proietti, editor, *Logic Program Synthesis and Transformation, Proceedings LoPSTr '95, Utrecht, The Netherlands*, Lecture Notes in Computer Science 1048, pages 33–45. Springer, 1996.

[5] R. M. Burstall and J. Darlington. A transformation system for developing recursive programs. *Journal of the ACM*, 24(1):44–67, January 1977.

[6] P. M. Hill and J. Gallagher. Meta-programming in logic programming. In D. M. Gabbay, C. J. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 5, pages 421–497. Oxford University Press, 1998.

[7] J. Jørgensen, M. Leuschel, and B. Martens. Conjunctive partial deduction in practice. In J. Gallagher, editor, *Logic Program Synthesis and Transformation, Proceedings of LoPSTr '96, Stockholm, Sweden*, Lecture Notes in Computer Science 1207, pages 59–82. Springer-Verlag, 1997.

[8] J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, Berlin, 1987. Second Edition.

[9] G. Nadathur and D. A. Miller. Higher-order logic programming. In D. M. Gabbay, C. J. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 5, pages 499–590. Oxford University Press, 1998.

[10] A. Pettorossi and M. Proietti. Transformation of logic programs: Foundations and techniques. *Journal of Logic Programming*, 19,20:261–320, 1994.

[11] A. Pettorossi and M. Proietti. Flexible continuations in logic programs via unfold/fold transformations and goal generalization. In O. Danvy, editor, *Proceedings of the 2nd ACM SIGPLAN Workshop on Continuations, January 14, 1997, ENS, Paris (France) 1997*, pages 9.1–9.22. BRICS Notes Series, N6-93-13, Aahrus, Denmark, 1997.

[12] A. Pettorossi and A. Skowron. Higher order generalization in program derivation. In *International Joint Conference on Theory and Practice of Software Development, TAPSOFT '87*, Lecture Notes in Computer Science 250, pages 182–196. Springer-Verlag, 1987.

[13] M. Proietti and A. Pettorossi. Unfolding-definition-folding, in this order, for avoiding unnecessary variables in logic programs. *Theoretical Computer Science*, 142(1):89–124, 1995.

[14] M. Proietti and A. Pettorossi. Transforming inductive definitions. In D. De Schreye, editor, *Proceedings of the 1999 International Conference on Logic Programming*, pages 486–499. MIT Press, 1999.

[15] D. Sands. Total correctness by local improvement in the transformation of functional programs. *ACM Toplas*, 18(2):175–234, 1996.

[16] L. S. Sterling and E. Shapiro. *The Art of Prolog*. The MIT Press, 1986.

[17] H. Tamaki and T. Sato. Unfold/fold transformation of logic programs. In S.-Å. Tärnlund, editor, *Proceedings of the Second International Conference on Logic Programming, Uppsala, Sweden*, pages 127–138. Uppsala University, 1984.

[18] M. Wand. Continuation-based program transformation strategies. *Journal of the ACM*, 27(1):164–180, 1980.

[19] D. H. D. Warren. Higher-order extensions to Prolog: are they needed? In Y-H Pao J.E. Hayes, D. Michie, editor, *Machine Intelligence*, volume 10, pages 441–454, Chichester, 1982. Ellis Horwood Ltd.

[20] G. Winskel. *The Formal Semantics of Programming Languages: An Introduction*. The MIT Press, 1993.