A. Pettorossi, M. Proietti

TRANSFORMATION RULES FOR A HIGHER ORDER LOGIC PROGRAMMING LANGUAGE

R. 525 Maggio 2000

Alberto Pettorossi – Dipartimento di Informatica, Sistemi e Produzione, Università di Roma Tor Vergata, Via di Tor Vergata, I-00133 Roma, Italy, and Istituto di Analisi dei Sistemi ed Informatica del CNR, Viale Manzoni 30, I-00185 Roma, Italy.
Email : `adp@iasi.rm.cnr.it`. URL : `http://www.iasi.rm.cnr.it/~adp`.

Maurizio Proietti – Istituto di Analisi dei Sistemi ed Informatica del CNR, Viale Manzoni 30, I-00185 Roma, Italy. Email : `proietti@iasi.rm.cnr.it`.
URL : `http://www.iasi.rm.cnr.it/~proietti`.

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). A preliminary version of this work appeared 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.

ISSN: 1128–3378

Collana dei Rapporti dell'Istituto di Analisi dei Sistemi ed Informatica, CNR

viale Manzoni 30, 00185 ROMA, Italy

tel. ++39-06-77161
fax ++39-06-7716461
email: `iasi@iasi.rm.cnr.it`
URL: `http://www.iasi.rm.cnr.it`

Abstract

We introduce a logic programming language with higher order features. In particular, in this language the arguments of the predicate symbols may be both terms and goals. We define the operational semantics of our language by extending SLD-resolution, and we propose for this language a set of program transformation rules. The transformation rules are shown to be correct in the sense that they preserve the operational semantics. In our higher order logic language we may transform logic programs using higher order generalizations and continuation arguments, as it is done in the case of functional programs. These program transformation techniques allow us to derive very efficient logic programs and also to avoid goal rearrangements which may not preserve correctness.

Key words: Automatic program derivation, program transformation, logic programming, transformation rules

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:

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

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

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

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

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

where the goal $p(l,\ true)$ 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 before performing a folding step, we are required to modify the positions of the atoms in the body of a clause. This problem arises because 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 an 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 some variables which we call goal variables, to be bound to goals. We also consider an enhancement of the usual unfold/fold transformation rules so that they may be used in the case where goals occur as arguments. For instance, we will allow the unfolding of a clause w.r.t. an atom which is an argument of a predicate symbol. The transformation rules are shown to be correct w.r.t. the operational semantics of our language (see Section 5).

To get this correctness property, we have to carefully define the semantics and the transformation rules for our language. Indeed, our enhanced transformation rules may be incorrect w.r.t. naive extensions of LD-resolution. For instance, 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, because $p(s)$ does not unify with $p(q)$. 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 so to get correctness of our unfolding, folding, and goal replacement rules, we will proceed as follows. We first define the syntax of our logic language by imposing that each clause head occurring in our programs has distinct variables as arguments. This can be achieved at the expense of introducing equality atoms in the body. Thus, for instance, clause 2 is rewritten as:

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

Then we define the semantics of our language so that, given any two goals g_1 and g_2, the equality $g_1 = g_2$ has a semantic value only when g_1 is a goal variable not occurring in the goal g_2 (in particular, for any goal g the operational semantics of the equality $g = g$ is *undefined*). 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 program consisting of 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 consisting of clauses 1*, 2*, and 3, gets stuck because the goal $s = q$ is selected.

Essentially, in our higher order logic language we may introduce goals, pass goals as arguments, and evaluate goals, 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 higher order logic programs. 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: *strong* correctness and *weak* correctness. Strong correctness ensures that a goal succeeds or fails in the original program iff it succeeds or fails, respectively, in the transformed program. Weak correctness consists in the 'only-if' part of strong correctness, that is, if a goal succeeds or fails in the original program then it succeeds or fails, respectively, in the transformed program. Thus, when weak correctness holds, the transformed program may be *more defined* than the original program in the sense that there may be some goals which have no semantic value in the original program (that is, either their evaluation does not terminate or it gets stuck), whereas they have a value in the transformed program (that is, their evaluation terminates and it does not get stuck).

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 higher order logic language

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): $f, f_1, f_2, \ldots$, (iv) *primitive predicate symbols*: *true*, *false*, $=$, and (v) *predicate symbols* (with arity): $p, p_1, p_2, \ldots$ 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:

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

The binary operators '$\wedge$' (conjunction) and '$\vee$' (disjunction) are assumed to be associative with neutral elements *true* and *false*, respectively. Thus, a goal g is the same as $\mathit{true} \wedge g$ and $g \wedge \mathit{true}$. Similarly, g is the same as $\mathit{false} \vee g$ and $g \vee \mathit{false}$. 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 the clause and the goal g is called the *body* of the clause. 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:

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

where $p_1, \ldots, p_k$ are distinct non-primitive predicate symbols, and every non-primitive predicate symbol occurring in $\{g_1, \ldots, g_k\}$ is among $p_1, \ldots, p_k$. Given a program P and a non-primitive predicate p occurring in P, the unique clause in P of the form $p(V_1, \ldots, V_m) \leftarrow g$, is called the *definition* of p in P. We say that a predicate p is *defined* in a program P iff p has a definition in P.

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 for each 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 '$\vee$' 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.

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.* For any given syntactic construct r, we use $\mathit{vars}(r)$ to denote the set of variables occurring in r. In particular, given a substitution ϑ, a variable belongs to $\mathit{vars}(\vartheta)$ iff it occurs either in the domain or in the range of ϑ. Given a goal g_1 and a clause c of the form $p(V_1, \ldots, V_m) \leftarrow g[g_1]$, the *local variables* of g_1 in c are those in the set $\mathit{vars}(g_1) - (\{V_1, \ldots, V_m\} \cup \mathit{vars}(g[\_]))$.

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 $P \vdash g \mapsto b$, 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*.

The relation $P \vdash g \mapsto b$ is defined by using the following *deduction rules*, where:
(1) $mgu(t_1, t_2)$ denotes a most general unifier of the terms t_1 and t_2,
(2) $g\{V_1/u_1, \ldots, V_m/u_m\}$ denotes the substitution of the arguments $u_1, \ldots, u_m$ for the variables $V_1, \ldots, V_m$ in g, and
(3) *or* denotes disjunction in the boolean domain $\{true, false\}$.

$$(tt)\quad \frac{}{P \vdash true \mapsto true} \qquad\qquad (f\!f)\quad \frac{}{P \vdash false \wedge g \mapsto false}$$

$$(teq1)\quad \frac{}{P \vdash (t_1 = t_2) \wedge g \mapsto false} \qquad \text{if } t_1 \text{ and } t_2 \text{ are not unifiable}$$

$$(teq2)\quad \frac{P \vdash g\vartheta \mapsto b}{P \vdash (t_1 = t_2) \wedge g \mapsto b} \qquad \text{if } t_1 \text{ and } t_2 \text{ are unifiable and } \vartheta = mgu(t_1, t_2)$$

$$(geq)\quad \frac{P \vdash g_2\{G/g_1\} \mapsto b}{P \vdash (G = g_1) \wedge g_2 \mapsto b} \qquad \text{if } G \notin vars(g_1)$$

$$(at)\quad \frac{P \vdash body\{V_1/u_1, \ldots, V_m/u_m\} \wedge g \mapsto b}{P \vdash p(u_1, \ldots, u_m) \wedge g \mapsto b}$$

where $p(V_1, \ldots, V_m) \leftarrow body$ is a renamed apart clause of P

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

A *deduction tree* τ for $P \vdash g \mapsto b$ is a tree such that: (i) the root of τ is $P \vdash g \mapsto b$, 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, say r, whose conclusion is n and whose premises are $n_1, \ldots, n_k$. We say that n *is derived by applying rule* r. A *proof* of $P \vdash g \mapsto b$ is a finite deduction tree for $P \vdash g \mapsto b$ where every leaf is a deduction rule which has no premises.

We stipulate that $P \vdash g \mapsto b$ *holds* iff there exists a proof of $P \vdash g \mapsto b$ constructed by using the set of deduction rules given above. If $P \vdash g \mapsto true$ we say that g *succeeds* in P. If $P \vdash g \mapsto false$ we say that g *fails* in P. If g either succeeds or fails in P we say that g *terminates* in P. If g neither succeeds nor fails in P we say that g is *undefined* in P.

Notes on semantics: (1) the rules $\frac{}{P \vdash false \mapsto false}$ and $\frac{P \vdash g \mapsto b}{P \vdash true \wedge g \mapsto b}$ which the reader may expect by symmetry from (tt) and $(f\!f)$, are redundant, because they are derivable as instances of $\frac{}{P \vdash false \wedge g \mapsto false}$ and $\frac{P \vdash g \mapsto b}{P \vdash g \mapsto b}$, respectively, by using the fact that *true* is the neutral element of a conjunction of goals. (2) Our deduction rules for the operational semantics are mutually exclusive, and thus, there exists at most one proof for each $P \vdash g \mapsto b$ (modulo variable names). (3) The first argument of the ternary relation $P \vdash g \mapsto b$ never changes in the presentation of our deduction rules, and thus, we could have omitted it. However, the explicit dependence of the operational semantics on the program P will be useful when presenting our correctness theorem (see Theorem 1 in Section 5).

Our deduction rules extend the LD-resolution for definite logic programs and a conjunction of goals is evaluated using the leftmost selection rule. The evaluation of a disjunction of goals requires the evaluation of each disjunct, and the existence of a proof of $P \vdash g \mapsto b$ implies the *universal termination* of the goal g [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 $P \vdash g \mapsto b$ holds, because every deduction tree with root $P \vdash g \mapsto b$ *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: $P \vdash g_0 \mapsto b_0$ 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 of $P \vdash G \mapsto b$ and there is no proof of $P \vdash (G = p \wedge G = q) \mapsto b$.

Notice also that the computed answers semantics is *not* captured by the ternary relation $P \vdash g \mapsto b$ defined by using the deduction rules given above. 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:

P_1:	$h(G) \leftarrow (G = p) \wedge G$	P_2:	$h(G) \leftarrow (G = q) \wedge G$
	$p \leftarrow q$		$p \leftarrow q$
	$q \leftarrow$		$q \leftarrow$

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 higher order language. 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, from program P_i we get a new program P_{i+1}.

Rule R1 (Definition Introduction). We get the new program P_{i+1} by adding to the current program P_i a new clause of the form:

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

where: (i) *newp* is a new non-primitive predicate symbol not occurring in any program of the sequence $P_0, \ldots, P_i$, (ii) the non-primitive predicate symbols occurring in g are defined in P_0, and (iii) $V_1, \ldots, V_m$ are distinct (possibly not all) variables which occur in g.
The set of all clauses introduced while constructing the transformation sequence $P_0, \ldots, P_i$, is denoted by Def_i. Thus, $Def_0 = \emptyset$.

Rule R2 (Unfolding). Let $c_1 : h \leftarrow body[p(u_1, \ldots, u_m)]$ be a clause of program P_i where p is a non-primitive predicate symbol. Let $c : p(V_1, \ldots, V_m) \leftarrow g$ be the definition of p in

$P_0 \cup \mathit{Def}_i$. By *unfolding* c_1 *w.r.t.* $p(u_1,\ldots,u_m)$ *using* c we get the new clause $c_2 : h \leftarrow \mathit{body}[g\{V_1/u_1,\ldots,V_m/u_m\}]$. We also get the new program P_{i+1} by replacing in program P_i clause c_1 by clause c_2.

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

- $V\vartheta$ is a local variable of $g\vartheta$ in c_1, 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_1 *using* c we get the new program P_{i+1} by replacing in program P_i the clause c_1 by the clause: $h \leftarrow \mathit{body}[p(V_1,\ldots,V_m)\vartheta]$.

In order to present the goal replacement rule (see rule R4 below) we first introduce some laws, called *replacement laws*. A replacement law denotes a pair of goals which can be replaced one for the other in the body of a clause. We have two kinds of replacement laws: the *weak* and *strong* replacement laws.

Given a program P and goals g_1 and g_2, the *weak replacement law* $P \vdash g_1 \Longrightarrow\!\!\succ g_2$ holds iff for every goal context $g[\_]$ and for every $b \in \{\mathit{true},\mathit{false}\}$, if $P \vdash g[g_1] \mapsto b$ has a proof then $P \vdash g[g_2] \mapsto b$ has a proof whose depth is not greater than that of $P \vdash g[g_1] \mapsto b$ (see below for a rigorous definition of *proof depth*).

For the goal replacement rule based on weak replacement laws we will prove a correctness result (see the Improvement Part of the Correctness Theorem of Section 5) stating that if program P_2 is derived from program P_1 by an application of this rule, then for every goal g and for every $b \in \{\mathit{true},\mathit{false}\}$, if $P_1 \vdash g \mapsto b$ then $P_2 \vdash g \mapsto b$. Thus, program P_2 may be *more defined* than program P_1 in the sense that there may be some goal g and some $b \in \{\mathit{true},\mathit{false}\}$ such that $P_2 \vdash g \mapsto b$ holds, while $P_1 \vdash g \mapsto b$ has no proof. Weak replacement laws allow for the derivation of new programs which may terminate more often than the initial programs, like for instance, when replacing the goal $(g \wedge \mathit{false})$ by false, or replacing the goal $(g \vee \mathit{true})$ by true.

Given a program P and goals g_1 and g_2, the *strong replacement law* $P \vdash g_1 \Longleftrightarrow\!\!\succ g_2$ holds iff (i) 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 $b \in \{\mathit{true},\mathit{false}\}$, $P \vdash g[g_1] \mapsto b$ has a proof iff $P \vdash g[g_2] \mapsto b$ has a proof, and
(ii) if $P \vdash g[g_1] \mapsto b$ and $P \vdash g[g_2] \mapsto b$ have proofs, then the depth of the proof of $P \vdash g[g_1] \mapsto b$ is not smaller than the depth of the proof of $P \vdash g[g_2] \mapsto b$.

The latter condition makes $\Longleftrightarrow\!\!\succ$ an *improvement* relation in the sense of [15]. In the Correctness Theorem of Section 5 we will show that the goal replacement rule based on strong replacement laws preserves our operational semantics. Indeed, if program P_2 is derived 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 $P_1 \vdash g \mapsto b$ iff $P_2 \vdash g \mapsto b$.

For this result, we do use the assumption that $\Longleftrightarrow\!\!\succ$ 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 $b \in \{\mathit{true}, \mathit{false}\}$, $P_1 \vdash g[p] \mapsto b$ iff $P_1 \vdash g[q] \mapsto b$, but for the empty goal context $[\_]$, the proof of $P_1 \vdash p \mapsto \mathit{true}$ has greater depth than the proof of $P_1 \vdash q \mapsto \mathit{true}$. 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.

Now, 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 a clause c of the form: $h \leftarrow \mathit{body}[g_1]$, does not depend on the local variables of g_1 and g_2 in c. For this reason we find it convenient to introduce weak replacement laws of the form: $P \vdash g_1 \backslash V_1 \mathrel{\Longrightarrow\mkern-14mu\succ} g_2 \backslash V_2$. They are used to replace in clause c goal g_1 with local variables V_1 by goal g_2 with local variables V_2. Similarly, we also introduce strong replacement laws of the form: $P \vdash g_1 \backslash V_1 \mathrel{\Longleftrightarrow\mkern-14mu\succ} g_2 \backslash V_2$.

Definition 1 (Proof Depth). Let π be a deduction tree for $P \vdash g \mapsto b$ and let m be the maximal number of nodes in a root-to-leaf path of π which are derived by using the deduction rule (at). Then we say that $P \vdash g \mapsto b$ has a proof depth m, and we write $P \vdash g \mapsto^m b$.

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 $\mathit{vars}(g[\_]) \cap (V_1 \cup V_2) = \emptyset$, and for every $b \in \{\mathit{true}, \mathit{false}\}$, we have that:

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

(ii) The relation $P \vdash g_1 \backslash V_1 \mathrel{\Longrightarrow\mkern-14mu\succ} g_2 \backslash V_2$, called a *weak replacement law*, holds iff for every goal context $g[\_]$ such that $\mathit{vars}(g[\_]) \cap (V_1 \cup V_2) = \emptyset$, and for every $b \in \{\mathit{true}, \mathit{false}\}$, we have that:

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

(iii) The relation $P \vdash g_1 \backslash V_1 \mathrel{\Longleftrightarrow\mkern-14mu\succ} g_2 \backslash V_2$, called a *strong replacement law*, holds iff both $P \vdash g_1 \backslash V_1 \mathrel{\Longrightarrow\mkern-14mu\succ} g_2 \backslash V_2$ and $P \vdash g_2 \backslash V_2 \Longrightarrow g_1 \backslash V_1$ hold.

(iv) We write $P \vdash g_1 \backslash V_1 \mathrel{\prec\mkern-14mu\Longleftrightarrow\mkern-14mu\succ} g_2 \backslash V_2$ to mean that: (iv.1) $P \vdash g_1 \backslash V_1 \mathrel{\Longrightarrow\mkern-14mu\succ} g_2 \backslash V_2$ and (iv.2) $P \vdash g_2 \backslash V_2 \mathrel{\Longrightarrow\mkern-14mu\succ} g_1 \backslash V_1$. Thus, if $P \vdash g_1 \backslash V_1 \mathrel{\prec\mkern-14mu\Longleftrightarrow\mkern-14mu\succ} g_2 \backslash V_2$ then the proofs of $P \vdash g_1 \mapsto b$ and $P \vdash g_2 \mapsto b$ have equal depth. Note that the conjunction of Conditions (iv.1) and (iv.2) is equivalent to the conjunction of $P \vdash g_1 \backslash V_1 \mathrel{\Longleftrightarrow\mkern-14mu\succ} g_2 \backslash V_2$ and $P \vdash g_2 \backslash V_2 \mathrel{\Longleftrightarrow\mkern-14mu\succ} g_1 \backslash V_1$.

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

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 non-primitive predicate symbols of $\{g_1, g_2\}$ are defined in P_0, and either (i) $P_0 \vdash g_1 \backslash V_1 \mathrel{\Longrightarrow\mkern-14mu\succ} g_2 \backslash V_2$, or (ii) $P_0 \vdash g_1 \backslash V_1 \mathrel{\Longleftrightarrow\mkern-14mu\succ} g_2 \backslash V_2$, 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.*

The reader may check that, for any program P, and goals g, g_1, g_2, and g_3, the following replacement laws hold:
1. *Boolean Laws*:

$P \vdash \mathit{true} \vee g \mathrel{\Longrightarrow\mkern-14mu\succ} \mathit{true}$	$P \vdash g_1 \vee g_2 \mathrel{\prec\mkern-14mu\Longleftrightarrow\mkern-14mu\succ} g_2 \vee g_1$
$P \vdash \mathit{false} \wedge g \mathrel{\prec\mkern-14mu\Longleftrightarrow\mkern-14mu\succ} \mathit{false}$	$P \vdash (g_1 \wedge g_2) \vee (g_1 \wedge g_3) \mathrel{\prec\mkern-14mu\Longleftrightarrow\mkern-14mu\succ} g_1 \wedge (g_2 \vee g_3)$
$P \vdash g \wedge \mathit{false} \mathrel{\Longrightarrow\mkern-14mu\succ} \mathit{false}$	$P \vdash (g_1 \wedge g_2) \vee (g_3 \wedge g_2) \mathrel{\prec\mkern-14mu\Longleftrightarrow\mkern-14mu\succ} (g_1 \vee g_3) \wedge g_2$
$P \vdash g \wedge g \mathrel{\Longleftrightarrow\mkern-14mu\succ} g$	$P \vdash (g_1 \vee g_2) \wedge (g_1 \vee g_3) \mathrel{\Longrightarrow\mkern-14mu\succ} g_1 \vee (g_2 \wedge g_3)$
$P \vdash g \vee g \mathrel{\prec\mkern-14mu\Longleftrightarrow\mkern-14mu\succ} g$	

In the following replacement laws 2.a, 2.b, and 2.c, according to 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*:

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

2.b *Simplification of equalities*:

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

2.c *Rearrangement of equalities*:

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

When referring to goal variables, 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*:

$P \vdash g \wedge (t_1=t_2) \Longrightarrow\!\!\!\!\succ (t_1=t_2) \wedge g$

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

$P \vdash eq_1\backslash Y \prec\!\!\!\!\Longleftrightarrow\!\!\!\!\succ eq_2\backslash Z$ if 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:

$P \vdash true \Longrightarrow true \vee g$
$P \vdash false \Longrightarrow g \wedge false$
$P \vdash (t_1=t_2) \wedge g \Longrightarrow g \wedge (t_1=t_2)$
$P \vdash g_1 \vee (g_2 \wedge g_3) \Longrightarrow (g_1 \vee g_2) \wedge (g_1 \vee g_3)$
$P \vdash g_2[g_1] \Longrightarrow g_2[G] \wedge (G=g_1)$
$P \vdash g[(G=g_1) \wedge g_2] \Longrightarrow (G=g_1) \wedge g[g_2]$

The following lemmata are often useful to prove that a replacement law holds.

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 $vars(g[\_]) \cap (V_1 \cup V_2) = \emptyset$.
(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 $P \vdash g[g_1]\backslash(V_1 \cup W) \Longrightarrow\!\!\!\!\succ g[g_2]\backslash(V_2 \cup W)$.

Proof. The thesis follows directly from the definition of replacement law. □

Lemma 2. Let P be a program, g_1, g_2 be goals, and V_1, V_2 be sets of variables.
(i) $P \vdash g_1\backslash V_1 \Longrightarrow g_2\backslash V_2$ holds iff for every idempotent substitution ϑ such that $vars(\vartheta) \cap (V_1 \cup V_2) = \emptyset$, and for every goal g, we have that:

if $P \vdash g_1\vartheta \wedge g \mapsto b$ then $P \vdash g_2\vartheta \wedge g \mapsto b$.

(ii) $P \vdash g_1\backslash V_1 \Longrightarrow\!\!\!\!\succ g_2\backslash V_2$ holds iff for every idempotent substitution ϑ such that $vars(\vartheta) \cap (V_1 \cup V_2) = \emptyset$, and for every goal g, we have that:

if $P \vdash g_1\vartheta \wedge g \mapsto^m b$ then $P \vdash g_2\vartheta \wedge g \mapsto^n b$ and $m \geq n$.

Proof. *Only-if* part of (i). Assume that $P \vdash g_1\vartheta \wedge g \mapsto b$ has a proof, where ϑ is the substitution $\{V_1/u_1, \ldots, V_k/u_k\}$ such that, for $i = 1, \ldots, k$, $V_i \notin u_i$. Without loss of generality, we may assume that, for $i = 1, \ldots, k$, $V_i \notin g$. Then, by using rules ($teq2$) and (geq) we may construct a proof of $P \vdash (V_1 = u_1, \ldots, V_k = u_k) \wedge g_1 \wedge g \mapsto b$. By the hypothesis that $P \vdash g_1 \backslash V_1 \Longrightarrow g_2 \backslash V_2$ holds, we have that $P \vdash (V_1 = u_1, \ldots, V_k = u_k) \wedge g_2 \wedge g \mapsto b$ has a proof. The only way of constructing this proof is by using k times the rules ($teq2$) and (geq) and constructing a proof of $P \vdash g_2\vartheta \wedge g \mapsto b$.

If part of (i). We have to show that for every goal context $g[\_]$ and substitution ϑ such that $vars(g[\_]\vartheta) \cap (V_1 \cup V_2) = \emptyset$, and for every $b \in \{true, false\}$, we have that:

if $P \vdash g[g_1]\vartheta \mapsto b$ has a proof, then $P \vdash g[g_2]\vartheta \mapsto b$ has a proof.

We proceed by induction w.r.t. a well-founded order on proofs defined as follows. 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 $m2 < n2$). For any proof π 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 our thesis by induction w.r.t. the measure $\mu(\pi)$ of the proof π of $P \vdash g[g_1]\vartheta \mapsto b$. We reason by cases on the structure of the goal context $g[\_]$. By the definition of the syntax and the fact that $\wedge$ is associative with neutral element *true*, the goal $g[g_1]$ may be of one of the following forms for some goals g_3, g_4, and g_5:
(1) $g_1 \wedge g_3$,
(2) $(g_1 = g_3) \wedge g_4$, where g_1 is a goal variable not occurring in g_3,
(3) $(G = g_3[g_1]) \wedge g_4$, where G is a goal variable not occurring in $g_3[g_1]$,
(4) $p(u_1, \ldots, u_i[g_1], \ldots, u_k) \wedge g_3$,
(5) $(g_3[g_1] \vee g_4) \wedge g_5$,
(6) $(g_3 \vee g_4[g_1]) \wedge g_5$,
(7) $g_3 \wedge g_4[g_1]$, where g_3 is a goal different from *true*.
We consider the following three cases only. The others are similar and we omit them.
- Case 1: $g[g_1]$ is $g_1 \wedge g_3$. Assume that $P \vdash (g_1 \wedge g_3)\vartheta \mapsto b$, that is, $P \vdash g_1\vartheta \wedge g_3\vartheta \mapsto b$. Then, by hypothesis, $P \vdash g_2\vartheta \wedge g_3\vartheta \mapsto b$, and thus, $P \vdash (g_2 \wedge g_3)\vartheta \mapsto b$.
- Case 3: $g[g_1]$ is $(G = g_3[g_1]) \wedge g_4$ and G is a goal variable not occurring in $g_3[g_1]$.
Assume that $P \vdash ((G = g_3[g_1]) \wedge g_4)\vartheta \mapsto b$ has a proof of depth m and size s. Then, $G\vartheta$ is a goal variable not occurring in $g_3[g_1]\vartheta$, the node $P \vdash (G\vartheta = g_3[g_1]\vartheta) \wedge g_4\vartheta \mapsto b$ has been obtained by applying rule (geq), and $P \vdash g_4\vartheta\{G\vartheta/g_3[g_1]\vartheta\} \mapsto b$ has a proof of depth m and size $s-1$. Since $\langle m, s-1\rangle < \langle m, s\rangle$, by the inductive hypothesis we have that $P \vdash g_4\vartheta\{G\vartheta/g_3[g_2]\vartheta\} \mapsto b$ has a proof. Thus, by using rule (geq), we can build a proof of $P \vdash G\vartheta = g_3[g_2]\vartheta \wedge g_4\vartheta \mapsto b$ which is equal to $P \vdash ((G = g_3[g_2]) \wedge g_4)\vartheta \mapsto b$.
- Case 4: $g[g_1]$ is $p(u_1, \ldots, u_i[g_1], \ldots, u_k) \wedge g_3$.
Assume that $P \vdash p(u_1\vartheta, \ldots, u_i[g_1]\vartheta, \ldots, u_k\vartheta) \wedge g_3\vartheta \mapsto b$ has a proof of depth m and size s. Then, for the last step of this proof, rule (at) has been used, and $P \vdash body\{V_1/u_1\vartheta, \ldots, V_i/u_i[g_1]\vartheta, \ldots, V_k/u_k\vartheta\} \wedge g_3\vartheta \mapsto b$ has a proof of depth $m-1$ and size $s-1$, where $p(V_1, \ldots, V_i, \ldots, V_k) \leftarrow body$ is a renamed apart clause of P. Since $\langle m-1, s-1\rangle < \langle m, s\rangle$, by the inductive hypothesis we have that $P \vdash body\{V_1/u_1\vartheta, \ldots, V_i/u_i[g_2]\vartheta, \ldots, V_k/u_k\vartheta\} \wedge g_3\vartheta \mapsto b$ has a proof. Thus, by using rule (at), we can build a proof of $P \vdash p(u_1\vartheta, \ldots, u_i[g_2]\vartheta, \ldots, u_k\vartheta) \wedge g_3\vartheta \mapsto b$.

The proof of (ii) is similar to the proof of (i) and we omit it. □

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:

$p(V_1,\ldots,V_m) \leftarrow (a_1 \wedge g_1) \vee \ldots \vee (a_s \wedge g_s)$ (*)

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

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

Theorem 1 (Correctness). Let $P_0,\ldots,P_k$ be a transformation sequence 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_1 using clause c, then there exist i, j, with $0 \leq i < j \leq k$, such that P_j is obtained from P_i by parallel leftmost unfolding of c.
Then for every goal g and for every $b \in \{true, false\}$, we have that:

(1) *Improvement*: if $P_0 \cup Def_k \vdash g \mapsto^m b$ then $P_k \vdash g \mapsto^n b$ with $m \geq n$, and

(2) *Soundness*: if all applications of the goal replacement rule are based on strong replacement laws, then: if $P_k \vdash g \mapsto b$ then $P_0 \cup Def_k \vdash g \mapsto b$.

As a consequence of this Correctness Theorem we have that if some applications of the goal replacement rule are based on weak replacement laws which are not strong replacement laws, the Soundness Property may not hold, and the derived program P_k may be *more defined* than program $P_0 \cup Def_k$, that is, there may be some goal g and some $b \in \{true, false\}$ such that $P_k \vdash g \mapsto b$ holds, while $P_0 \cup Def_k \vdash g \mapsto b$ has no proof.

The Improvement Part of the Correctness Theorem follows from Lemmata 3 and 4 below.

Lemma 3 (Improvement). Let P and $NewP$ be programs of the form:

$$P: \quad \begin{array}{c} hd_1 \leftarrow bd_1 \\ \vdots \\ hd_s \leftarrow bd_s \end{array} \qquad\qquad NewP: \quad \begin{array}{c} hd_1 \leftarrow newbd_1 \\ \vdots \\ hd_s \leftarrow newbd_s \end{array}$$

For $r = 1,\ldots,s$, let $Y_r = vars(bd_r) - vars(hd_r)$ and $Z_r = vars(newbd_r) - vars(hd_r)$ and suppose that: $P \vdash bd_r \backslash Y_r \Longrightarrow\!\!\!\!> newbd_r \backslash Z_r$.

Then for all goals g, we have:

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

Proof. We consider the measure on proofs defined in the proof of Lemma 2 and we prove the thesis by induction w.r.t. the measure of the proof of $P \vdash g \mapsto b$ which, by hypothesis, has depth m.

Our induction hypothesis is that, for all $\langle m1, s1\rangle < \langle m, s\rangle$, for all goals g, and for all $b \in \{true, false\}$, if $P \vdash g \mapsto b$ has a proof of depth $m1$ and size $s1$, then $NewP \vdash g \mapsto b$ has a proof of depth $n1$, with $m1 \geq n1$. We have to show that $NewP \vdash g \mapsto b$ has a proof of depth n, with $m \geq n$. We proceed by cases on the structure of g. We first notice that, since $\wedge$ is associative with neutral element *true*, the grammar for generating goals given in Section 2 can be replaced by the following one:

$g ::= G \wedge g_1 \,|\, true \,|\, false \wedge g_1 \,|\, (t_1 = t_2) \wedge g_1 \,|\, (G = g_1) \wedge g_2 \,|\, p(u_1,\ldots,u_m) \wedge g_1 \,|\, (g_1 \vee g_2) \wedge g_3$

We consider the following two cases only. The other cases are similar and we omit them.

- Case 1: g is $(G = g_1) \wedge g_2$. Assume that $P \vdash (G = g_1) \wedge g_2 \mapsto b$ has a proof of depth m and size s. Then, $G \notin vars(g_1)$, $P \vdash (G = g_1) \wedge g_2 \mapsto b$ has been derived by applying rule (geq), and $P \vdash g_2\{G/g_1\} \mapsto b$ has a proof of depth m and size $s-1$. Since $\langle m, s-1\rangle < \langle m, s\rangle$, by the

induction hypothesis $NewP \vdash g_2\{G/g_1\} \mapsto b$ has a proof of depth n with $m \geq n$ and, by rule (geq), we have that $NewP \vdash (G=g_1) \wedge g_2 \mapsto b$ has a proof of depth n with $m \geq n$.
- Case 2: g is $p(u_1,\ldots,u_m) \wedge g_1$. Assume that $P \vdash p(u_1,\ldots,u_m) \wedge g_1 \mapsto b$ has a proof of depth m and size s. Then, $P \vdash p(u_1,\ldots,u_m) \wedge g_1 \mapsto b$ has been derived by using rule (at), and $P \vdash bd_r\{V_1/u_1,\ldots,V_m/u_m\} \wedge g_1 \mapsto b$ 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. Now, by (i) the hypothesis that $P \vdash bd_r\backslash Y_r \Longrightarrow\!\!\succ newbd_r\backslash Z_r$ and (ii) the fact that $vars(\{V_1/u_1,\ldots, V_m/u_m\}) \cap (Y_r \cup Z_r) = \emptyset$, by Lemma 2 we have that $P \vdash newbd_r\{V_1/u_1,\ldots, V_m/u_m\} \wedge g_1 \mapsto b$ 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 $NewP \vdash newbd_r\{V_1/u_1,\ldots, V_m/u_m\} \wedge g_1 \mapsto b$ of depth $n2$ with $n1 \geq n2$. Since hd_r is $p(V_1,\ldots,V_m)$, by using rule (at) we can construct a proof for $NewP \vdash p(u_1,\ldots,u_m) \wedge g_1 \mapsto b$ of depth $n = n2+1$, and thus, $m \geq n$. □

In order to simplify the proof of our next Lemma 4, 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 applying the folding rule in the original sequence $P_0,\ldots,P_k$.

Lemma 4. 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 applying 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 applying the folding rule in $P_j,\ldots,P_k$. For $h = i,\ldots,k-1$, let $hd \leftarrow newbd^h$ be the clause in program P_{h+1} derived from a clause $hd \leftarrow bd^h$ in P_h, by applying the unfolding or folding or goal replacement rule, and let Y^h be $vars(bd^h) - vars(hd)$ and Z^h be $vars(newbd^h) - vars(hd)$. (Notice that the unfolding, folding, and goal replacement rules do not change the heads of the clauses.)
Then, (i) for $h = i,\ldots,j-1$, $P_0 \cup Def_k \vdash bd^h\backslash Y^h \Longrightarrow\!\!\succ newbd^h\backslash Z^h$, and
(ii) for $h = j,\ldots,k-1$, $P_j \vdash bd^h\backslash Y^h \Longrightarrow\!\!\succ newbd^h\backslash Z^h$.

Proof. (i) For every clause $p(V_1,\ldots,V_m) \leftarrow g$ in $P_0 \cup Def_k$, we have that:

$P_0 \cup Def_k \vdash p(V_1,\ldots,V_m) \Longrightarrow\!\!\succ g\backslash Z$

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) In order to prove Point (ii) of the thesis, we first notice that the following two properties hold.

Property (†). For every clause c: $p(V_1,\ldots,V_m) \leftarrow g$ in P_0, we have that:

$P_j \vdash p(V_1,\ldots,V_m) \Longrightarrow\!\!\succ g\backslash Z$

where $Z = vars(g) - \{V_1,\ldots,V_m\}$. This property holds because $P_0 \subseteq P_j$, and thus, $c \in P_j$.

Property (††). For every 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:

$P_j \vdash newp(V_1,\ldots,V_m) \prec\!\!\Longleftarrow\!\!\Longrightarrow\!\!\succ g\backslash Z$

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 clauses in Def_k among which is d.

We now prove Point (ii) of the thesis by cases w.r.t. the transformation rule which is used, for $h = j,\ldots,k-1$, to derive program P_{h+1} from program P_h.

- Case 1: P_{h+1} is derived from P_h by the unfolding rule using a clause which is among those also used for folding (in previous transformation steps). The thesis follows from Property (††) and Point (ii) of the Replacement Lemma.

- Case 2: P_{h+1} is derived from P_h by the unfolding rule using a clause c which is *not* among those used for folding. Thus, c belongs to P_0 because the only way of introducing in the body of a clause an occurrence of a non-primitive predicate which is not defined in P_0, is by an application of the folding rule. Then the thesis follows from Property (†) and Point (ii) of the Replacement Lemma.

- Case 3: P_{h+1} is derived from P_h by the folding rule. The thesis follows from Property (††) and Point (ii) of the Replacement Lemma.

- Case 4: P_{h+1} is derived from P_h by the goal replacement rule based on a replacement law of the form $P_0 \vdash g_1\backslash Y \Longrightarrow\!\!\!\!\!\succ g_2\backslash Z$. The thesis follows from Point (ii) of the Replacement Lemma and the fact that $P_j \vdash g_1\backslash Y \Longrightarrow\!\!\!\!\!\succ g_2\backslash Z$ also holds, because the non-primitive predicates of $\{g_1, g_2\}$ are defined in P_0, and for each predicate p defined in P_0, the definition of p in P_0 is equal to the definition of p in P_j. □

In order to get the proof of the Improvement Part of the Correctness Theorem, it is sufficient to notice that: (A) by Lemma 3 and Point (i) of Lemma 4, we have that: for all goals g, if $P_0 \cup Def_k \vdash g \mapsto^m b$ then $P_j \vdash g \mapsto^{n1} b$ with $m \geq n1$, where P_j is the program constructed as indicated in Lemma 4, and (B) by Lemma 3 and Point (ii) of Lemma 4, we have that: for all goals g, if $P_j \vdash g \mapsto^{n1} b$ then $P_k \vdash g \mapsto^n b$ with $n1 \geq n$.

The proof of the Soundness Part of the Correctness Theorem is based on the following Lemmata 5 and 6.

Lemma 5 (Soundness). Let P and $NewP$ be programs of the form:

$$\begin{array}{llll} P: & hd_1 \leftarrow bd_1 & \qquad NewP: & hd_1 \leftarrow newbd_1 \\ & \vdots & & \vdots \\ & hd_s \leftarrow bd_s & & hd_s \leftarrow newbd_s \end{array}$$

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

Then for all goals g and for all $b \in \{true, false\}$, we have that: if $NewP \vdash g \mapsto b$ then $P \vdash g \mapsto b$.

Proof. We proceed by induction on the size of the proof of $NewP \vdash g \mapsto b$. Assume that, for all $m < n$ and for all goals g and for all $b \in \{true, false\}$, if $NewP \vdash g \mapsto b$ has a proof of size m then $P \vdash g \mapsto b$ 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 $(G = g_1) \wedge g_2$. Assume that $NewP \vdash (G = g_1) \wedge g_2 \mapsto b$ has a proof of size n. Then, $G \notin vars(g_1)$, $NewP \vdash (G = g_1) \wedge g_2 \mapsto b$ has been derived by applying rule (geq), and there exists a proof of size $n-1$ of $NewP \vdash g_2\{G/g_1\} \mapsto b$. By the induction hypothesis there exists a proof of $P \vdash g_2\{G/g_1\} \mapsto b$ and, by rule (geq), we have that $P \vdash (G = g_1) \wedge g_2 \mapsto b$ has a proof.
- Case 2: g is $p(u_1, \ldots, u_m) \wedge g_1$. Assume that $NewP \vdash p(u_1, \ldots, u_m) \wedge g_1 \mapsto b$ has a proof of size n. Then, $NewP \vdash p(u_1, \ldots, u_m) \wedge g_1 \mapsto b$ has been derived by applying rule (at), and there exists a proof of size $n-1$ of $NewP \vdash newbd_r\{V_1/u_1, \ldots, V_m/u_m\} \wedge g_1 \mapsto b$ where $p(V_1, \ldots, V_m) \leftarrow newbd_r$ is a renamed apart clause of $NewP$. By the induction hypothesis there exists a proof of $P \vdash newbd_r\{V_1/u_1, \ldots, V_m/u_m\} \wedge g_1 \mapsto b$. Now, by the hypothesis

that $P \vdash newbd_r \backslash Z_r \Longrightarrow bd_r \backslash Y_r$ and the fact that $vars(\{V_1/u_1, \ldots, V_m/u_m\}) \cap (Y_r \cup Z_r) = \emptyset$, by Lemma 2 we have that $P \vdash bd_r\{V_1/u_1, \ldots, V_m/u_m\} \wedge g_1 \mapsto b$ has a proof. Since hd_r is $p(V_1, \ldots, V_m)$, by using rule (at) we can construct a proof for $P \vdash p(u_1, \ldots, u_m) \wedge g_1 \mapsto b$. □

Lemma 6. Let us consider a transformation sequence $P_0, \ldots, P_k$. For $h = 0, \ldots, k-1$, let $hd \leftarrow newbd$ be a clause in program P_{h+1} derived from the clause $hd \leftarrow bd$ in P_h, 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_{h+1} is derived from P_h by the unfolding rule using a clause of the form $p(V_1, \ldots, V_m) \leftarrow g$ 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_{h+1} is derived from P_h by the folding rule using a clause of the form $newp(V_1, \ldots, V_m) \leftarrow g$ 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_{h+1} is derived from P_h by the goal replacement rule, then the thesis follows from the fact that it is based on a strong replacement law and from Point (i) of the Replacement Lemma. □

6. Higher order programs 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 using higher order programs, we can solve the goal rearrangement problem, and in particular, how we can avoid the need of rearranging atoms in the body of clauses when transforming programs for eliminating multiple traversals of data structures. Let us consider the initial program P_0:

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

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) $nat(X)$ holds iff X is a natural number, (iii) $check(X)$ holds iff all labels in the tree X are natural numbers, and (iv) $flip(X, Y)$ 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 programs we may use disjunctions in clause bodies). The second approach is to to use the higher order language and the transformation rules we have proposed in this paper.

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 includes some goal rearrangement steps, produces a program, say P_f, which is *not* correct, while the second derivation which uses our higher order language and transformation rules 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 termination. Indeed, there exists a goal g which terminates in P_0, while it does not terminate in P_f.

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:

5. $\mathit{flipcheck}(X,Y) \leftarrow (X=l(N) \wedge Y=l(N) \wedge \mathit{nat}(N)) \vee$
$(X=t(L,N,R) \wedge Y=t(FR,N,FL) \wedge$
$\mathit{flip}(L,FL) \wedge \mathit{flip}(R,FR) \wedge \mathit{nat}(N) \wedge \mathit{check}(FR) \wedge \mathit{check}(FL))$

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

6. $\mathit{flipcheck}(X,Y) \leftarrow (X=l(N) \wedge Y=l(N) \wedge \mathit{nat}(N)) \vee$
$(X=t(L,N,R) \wedge Y=t(FR,N,FL) \wedge$
$\mathit{nat}(N) \wedge \mathit{flip}(L,FL) \wedge \mathit{check}(FL) \wedge \mathit{flip}(R,FR) \wedge \mathit{check}(FR))$

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:

7. $\mathit{flipcheck}(X,Y) \leftarrow (X=l(N) \wedge Y=l(N) \wedge \mathit{nat}(N)) \vee$
$(X=t(L,N,R) \wedge Y=t(FR,N,FL) \wedge$
$\mathit{nat}(N) \wedge \mathit{flipcheck}(L,FL) \wedge \mathit{flipcheck}(R,FR))$

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 termination. Indeed, we have that the goal $\mathit{flipcheck}(t(l(N),0,l(a)),Y)$ fails in P_0 (that is, $P_0 \vdash \mathit{flipcheck}(t(l(N),0,l(a)),Y) \mapsto \mathit{false}$ holds), while this goal is undefined in program P_f (that is, $P_f \vdash \mathit{flipcheck}(t(l(N),0,l(a)),Y) \mapsto b$ 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 the Improvement Part of the Correctness Theorem. 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 $P_0 \vdash g \mapsto b$ holds, then $P_h \vdash g \mapsto b$ 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 clause:

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

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:

9. $\mathit{flipcheck}(X,Y) \leftarrow (X=l(N) \wedge Y=l(N) \wedge \mathit{nat}(N)) \vee$
$(X=t(L,N,R) \wedge Y=t(FR,N,FL) \wedge$
$\mathit{flip}(L,FL) \wedge \mathit{flip}(R,FR) \wedge$
$\mathit{nat}(N) \wedge GR=\mathit{check}(FR) \wedge GL=\mathit{check}(FL) \wedge GR \wedge GL)$

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

10. $\mathit{flipcheck}(X,Y) \leftarrow (X\!=\!l(N) \wedge Y\!=\!l(N) \wedge \mathit{nat}(N)) \vee$
$\quad (X\!=\!t(L,N,R) \wedge Y\!=\!t(FR,N,FL)\wedge$
$\quad \mathit{flip}(L,FL) \wedge GL\!=\!\mathit{check}(FL)\wedge$
$\quad \mathit{flip}(R,FR) \wedge GR\!=\!\mathit{check}(FR) \wedge \mathit{nat}(N) \wedge GR \wedge GL)$

Now, this clause 10 explains why we have introduced clause 8. Indeed, by using clause 8 we can eliminate the intermediate variables FL and FR occurring in clause 10 by folding each *flip* atom and its 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:

11. $\mathit{newp}(X,Y,G) \leftarrow ((X\!=\!l(N) \wedge Y\!=\!l(N)) \vee$
$\quad (X\!=\!t(L,N,R) \wedge Y\!=\!t(FR,N,FL)\wedge$
$\quad \mathit{flip}(L,FL) \wedge \mathit{flip}(R,FR)))\wedge$
$\quad G\!=\!\mathit{check}(Y)$

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:

12. $\mathit{newp}(X,Y,G) \leftarrow (X\!=\!l(N) \wedge Y\!=\!l(N) \wedge G\!=\!\mathit{nat}(N)) \vee$
$\quad (X\!=\!t(L,N,R) \wedge Y\!=\!t(FR,N,FL)\wedge$
$\quad \mathit{flip}(L,FL) \wedge \mathit{flip}(R,FR)\wedge$
$\quad G\!=\!(\mathit{nat}(N) \wedge \mathit{check}(FR) \wedge \mathit{check}(FL)))$

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

13. $\mathit{newp}(X,Y,G) \leftarrow (X\!=\!l(N) \wedge Y\!=\!l(N) \wedge G\!=\!\mathit{nat}(N)) \vee$
$\quad (X\!=\!t(L,N,R) \wedge Y\!=\!t(FR,N,FL)\wedge$
$\quad \mathit{flip}(L,FL) \wedge V\!=\!\mathit{check}(FL) \wedge \mathit{flip}(R,FR) \wedge U\!=\!\mathit{check}(FR)\wedge$
$\quad G\!=\!(\mathit{nat}(N) \wedge U \wedge V))$

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

13.f $\mathit{newp}(X,Y,G) \leftarrow (X\!=\!l(N) \wedge Y\!=\!l(N) \wedge G\!=\!\mathit{nat}(N)) \vee$
$\quad (X\!=\!t(L,N,R) \wedge Y\!=\!t(FR,N,FL)\wedge$
$\quad \mathit{newp}(L,FL,V) \wedge \mathit{newp}(R,FR,U) \wedge G\!=\!(\mathit{nat}(N) \wedge U \wedge V))$

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

14. $\mathit{flipcheck}(X,Y) \leftarrow \mathit{flip}(X,Y) \wedge G\!=\!\mathit{check}(Y) \wedge G$

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

14.f $\mathit{flipcheck}(X,Y) \leftarrow \mathit{newp}(X,Y,G) \wedge G$

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

14.f $\mathit{flipcheck}(X,Y) \leftarrow \mathit{newp}(X,Y,G) \wedge G$

13.f $\mathit{newp}(X,Y,G) \leftarrow (X\!=\!l(N) \wedge Y\!=\!l(N) \wedge G\!=\!\mathit{nat}(N)) \vee$
$\quad (X\!=\!t(L,N,R) \wedge Y\!=\!t(FR,N,FL)\wedge$
$\quad \mathit{newp}(L,FL,V) \wedge \mathit{newp}(R,FR,U) \wedge G\!=\!(\mathit{nat}(N) \wedge U \wedge V))$

4. $\mathit{nat}(X) \leftarrow (X\!=\!0) \;\vee\; (X\!=\!s(N) \wedge \mathit{nat}(N))$

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 in the initial program P_0 a goal either succeeds or fails, then it either succeeds or fails, respectively, also in the final program P_h. 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 higher order 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 goal 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 $\mathit{flipcheck}(X,Y)$ consists of the evaluation of the goal $\mathit{newp}(X,Y,G)$ 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 which holds for 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. As in [3], we consider for our logic programs an operational semantics based on universal termination (that is, the operational semantics of a goal is defined iff all LD-derivations starting from that goal are finite). Our Correctness Theorem extends the results presented in [3] for definite logic programs 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 applying the folding rule one is required to use exactly one clause whose body is a conjunction of atoms. Notice that, however, the transformations presented in [3] also preserve computed answers, while ours do not.

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 sufficient conditions are not met or their validity is not proved, one 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 program execution.

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.

References

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

[2] M. Bertolino, S. Etalle, and C. Palamidessi, "The replacement operation for CCP programs," in *Proceedings of LoPSTr '99, Venice, Italy* (A. Bossi, ed.), Lecture Notes in Computer Science 1817, pp. 217–234, Springer, 2000.

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

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

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

[6] P. M. Hill and J. Gallagher, "Meta-programming in logic programming," in *Handbook of Logic in Artificial Intelligence and Logic Programming* (D. M. Gabbay, C. J. Hogger, and J. A. Robinson, eds.), vol. 5, pp. 421–497, Oxford University Press, 1998.

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

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

[9] G. Nadathur and D. A. Miller, "Higher-order logic programming," in *Handbook of Logic in Artificial Intelligence and Logic Programming* (D. M. Gabbay, C. J. Hogger, and J. A. Robinson, eds.), vol. 5, pp. 499–590, Oxford University Press, 1998.

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

[11] A. Pettorossi and M. Proietti, "Flexible continuations in logic programs via unfold/fold transformations and goal generalization," in *Proceedings of the 2nd ACM SIGPLAN Workshop on Continuations, January 14, 1997, ENS, Paris (France) 1997* (O. Danvy, ed.), pp. 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, pp. 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*, vol. 142, no. 1, pp. 89–124, 1995.

[14] M. Proietti and A. Pettorossi, "Transforming inductive definitions," in *Proceedings of the 1999 International Conference on Logic Programming* (D. D. Schreye, ed.), pp. 486–499, MIT Press, 1999.

[15] D. Sands, "Total correctness by local improvement in the transformation of functional programs," *ACM Toplas*, vol. 18, no. 2, pp. 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 *Proceedings of the Second International Conference on Logic Programming, Uppsala, Sweden* (S.-Å. Tärnlund, ed.), pp. 127–138, Uppsala University, 1984.

[18] M. Wand, "Continuation-based program transformation strategies," *Journal of the ACM*, vol. 27, no. 1, pp. 164–180, 1980.

[19] D. H. D. Warren, "Higher-order extensions to Prolog: are they needed?," in *Machine Intelligence* (Y.-H. P. J.E. Hayes, D. Michie, ed.), vol. 10, (Chichester), pp. 441–454, Ellis Horwood Ltd., 1982.

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