Totally Correct Logic Program Transformations
Using Well-Founded Annotations

Alberto Pettorossi\textsuperscript{1}, Maurizio Proietti\textsuperscript{2}

\textsuperscript{(1)} DISP, University of Tor Vergata, Roma, Italy. pettorossi@info.uniroma2.it
\textsuperscript{(2)} IASI-CNR, Roma, Italy. proietti@iasi.rm.cnr.it

(Extended Abstract)

Program transformation is one of the most prominent methodologies for the development of declarative programs and, in particular, functional and logic programs [2, 5, 9]. The main advantage of this methodology is that it allows one to deal with the issue of program correctness and the issue of program efficiency in a separated manner. One first writes a simple, maybe inefficient, program whose correctness can easily be proved, and then one derives a more efficient program by applying some given transformation rules which preserve program correctness.

In the case of definite logic programs, which are of our interest here, the correctness of the initial program is often very easy to prove because, usually, it is very close to the formal specification of that same program. On the contrary, the proof that the rules preserve program correctness is often more intricate (as it is also the case for functional programs). In particular, these correctness proofs cannot be done in isolation, in the sense that the correctness of a single transformation rule depends, in general, on the other rules one applies for transforming programs.

The correctness of the rules can be either partial or total. We say that a rule which transforms program $P_1$ into program $P_2$ is partially correct iff $M(P_1) \supseteq M(P_2)$, where $M(P)$ denotes the least Herbrand model of any given program $P$. Analogously, we say that a rule which transforms program $P_1$ into program $P_2$ is totally correct iff $M(P_1) = M(P_2)$.

Partial correctness is a straightforward consequence of the fact that the transformation rules, and in particular the familiar unfold/fold rules, basically consist in applying logical equivalences [9]. Indeed, whenever we derive a program $P_2$ from a program $P_1$ by replacing a formula $A$ by a formula $B$ such that $M(P_1 \models A \leftrightarrow B$, we get $M(P_1) \supseteq M(P_2)$. However, it is well known that the opposite inclusion $M(P_1) \subseteq M(P_2)$ may not hold and, thus, in general, the unfold/fold transformations are not totally correct as shown by the following simple example. Let us consider the transformation of program $P_1$ into program $P_2$, where $P_1$ and $P_2$ are as follows:

\begin{align*}
P_1: & \quad p \leftarrow q \\
& \quad q \leftarrow \\

P_2: & \quad p \leftarrow p \\
& \quad q \leftarrow \\
\end{align*}

This transformation, which corresponds to an application of the folding rule, is justified by the fact that the equivalence $M(P_1) \models p \leftrightarrow q$ holds. However, the
least Herbrand model is not preserved because we have that \( M(P_1) = \{ p, q \} \supset \{ q \} = M(P_2) \).

In the case of non-propositional programs it is not easy to check whether or not the application of an unfold/fold transformation rule is totally correct (actually, it can be shown that this is an undecidable problem). For this reason, in their landmark paper Tamaki and Sato proposed suitable applicability conditions which ensure the total correctness of the transformations [9]. These conditions are based on: (i) the form of the clauses that can be used in a folding step, and (ii) annotations of the program clauses that depend on the transformation history, that is, on the sequence of transformation rules applied during a program derivation. In particular, they stipulate that: (i) one is allowed to fold a clause by using a non-recursive clause which is marked as ‘foldable’, and (ii) a clause is marked as ‘foldable’ if it is derived by unfolding. Thus, conditions (i) and (ii) express that a clause can be folded only if it is derived by unfolding at a previous transformation step.

Tamaki-Sato’s approach has been extended in several papers (see, for instance, [4, 6, 8, 10]) by: (i) relaxing the restrictions on the clauses that can be used in a folding step, and (ii) generalizing the history dependent program annotations. The most recent of these papers [8] presents sufficient conditions for the total correctness of the unfold/fold transformations in the case where several, possibly recursive clauses are used in a folding step. These conditions are based on some measures which are incremented or decremented when the unfolding or folding rules are applied.

Unfortunately, the proofs of total correctness of the unfold/fold transformations presented in [4, 6, 8, 9, 10] use rather complex, ad hoc techniques, and it is very difficult to understand why they work and how they could be generalized for dealing with other program transformations or language extensions.

The main contribution of this paper is a logical foundation of the theory of total correctness of logic program transformations (and in particular unfold/fold transformations). Our theory is based on the notion of well-founded annotations and the unique fixpoint principle.

A well-founded annotation is a mapping \( \alpha \) that associates with every clause \( H \leftarrow A_1, \ldots, A_k \) of a program \( P \) an annotated clause of the form:

\[
H \leftarrow c(N, N_1, \ldots, N_k) \land A_1 \{ N_1 \} \land \ldots \land A_k \{ N_k \}
\]

where: (i) the annotation variables \( N, N_1, \ldots, N_k \) range over a set \( W \) and should be considered as extra arguments of the atoms occurring in the clause, and (ii) for \( i = 1, \ldots, k \), the relation \( c(N, N_1, \ldots, N_k) \) implies \( N > N_i \) where \( > \) is a well-founded ordering on \( W \). By applying the well-founded annotation \( \alpha \) to every clause of \( P \), we get an annotated program \( \alpha(P) \) that, by construction, enjoys the following two properties: (1) for every ground atom \( A, A \in M(P) \) iff there exists \( n \in W \) such that \( A \{ n \} \in M(\alpha(P)) \), and (2) for every ground annotated atom \( A \{ n \}, \alpha(P) \cup \{ \leftarrow A \{ n \} \} \) has a finite SLD tree, that is, \( \alpha(P) \) is terminating. By Property (2), the least Herbrand model of \( \alpha(P) \) is the unique fixpoint of the immediate consequence operator \( T_{\alpha(P)}[1] \).
Based on well-founded annotations, we propose a method for totally correct transformations of definite logic programs. Given a program $P_1$ our method allows us to derive a program $P_2$ by the following steps: (i) we choose a well-founded annotation $\alpha_1$ so that from program $P_1$ we produce an annotated program $\alpha_1(P_1)$, (ii) we apply suitable variants of the unfold/fold rules for transforming annotated programs so that from $\alpha_1(P_1)$ we derive a new terminating annotated program $\alpha_2(P_2)$, with $\alpha_2$ possibly different from $\alpha_1$, and finally, (iii) from $\alpha_2(P_2)$ we get program $P_2$ by erasing the annotations. The fact that $\alpha_2(P_2)$ is terminating is enforced by the transformation rules because they preserve the well-founded ordering $>$, in the sense that, for every clause derived by applying the rules, the annotation of the head is greater (w.r.t. $>$) than the annotation of every atom in the body.

The total correctness of the transformation, that is, $M(P_1) = M(P_2)$, is proved as follows. On one hand, the transformation rules act on non-annotated clauses like the usual unfold/fold rules and, as already mentioned, they ensure partial correctness, that is, $M(P_1) \supseteq M(P_2)$. On the other hand, since $\alpha_2(P_2)$ is terminating, by the uniquefixpoint principle [3,7] we have that $M(\alpha_1(P_1)) \subseteq M(\alpha_2(P_2))$ and, thus, by Property (1) of well-founded annotations, $M(P_1) \subseteq M(P_2)$.

Notice that in our method neither $P_1$ nor $P_2$ is required to be terminating. Moreover, our method is parametric w.r.t. the well-founded annotations and, in particular, w.r.t. the well-founded ordering $>$ used for the derivation of $P_2$ from $P_1$. By suitable choices of this ordering we can prove the total correctness of the various variants of the unfold/fold rules proposed in the literature [4,6,8,9,10].

An Example

We revisit an example of program transformation taken from [8] where the total correctness proof is rather intricate. We show that, on the contrary, the total correctness of this transformation can easily be established by our well-founded annotation method. Let us consider the following program $P_1$:

1. $\text{thm}(X) \leftarrow \text{gen}(X) \land \text{test}(X)$
2. $\text{gen}([]) \leftarrow$
3. $\text{gen}([0|X]) \leftarrow \text{gen}(X)$
4. $\text{test}(X) \leftarrow \text{canon}(X)$
5. $\text{test}(X) \leftarrow \text{trans}(X, Y) \land \text{test}(Y)$
6. $\text{canon}([]) \leftarrow$
7. $\text{canon}([1|X]) \leftarrow \text{canon}(X)$
8. $\text{trans}([0|X], [1|X]) \leftarrow$
9. $\text{trans}([1|X], [1|Y]) \leftarrow \text{trans}(X, Y)$

where $\text{thm}(X)$ holds iff $X$ is a string of 0’s that can be transformed into a string of 1’s by repeated applications of $\text{trans}(X, Y)$. Given the string $X$, the predicate $\text{trans}(X, Y)$ generates the string $Y$ by replacing the leftmost 0 in $X$ by 1. Let us consider the well-founded annotation $\alpha_1$ that associates with every clause:

$$H \leftarrow A_1 \land \ldots \land A_k$$
the annotated clause:

\[ H\{N\} \leftarrow N \geq N_1 + \ldots + N_k + 1 \land A_1\{N_1\} \land \ldots \land A_k\{N_k\} \]

where the annotation variables \(N, N_1, \ldots, N_k\) range over non-negative integers and \(\geq\) is the usual ‘greater or equal’ ordering over integers. Thus, the annotated program \(\alpha_1(P_1)\) is the following one:

1a. \(\text{thm}(X)\{N\} \leftarrow N \geq N_1 + N_2 + 1 \land \text{gen}(X)\{N_1\} \land \text{test}(X)\{N_2\}\)

2a. \(\text{gen}([[]])\{0\} \leftarrow\)

3a. \(\text{gen}(\{0\}[X])\{N\} \leftarrow N \geq N_1 + 1 \land \text{gen}(X)\{N_1\}\)

4a. \(\text{test}(X)\{N\} \leftarrow N \geq N_1 + 1 \land \text{canon}(X)\{N_1\}\)

5a. \(\text{test}(X)\{N\} \leftarrow N \geq N_1 + N_2 + 1 \land \text{trans}(X, Y)\{N_1\} \land \text{test}(Y)\{N_2\}\)

6a. \(\text{canon}([[]])\{0\} \leftarrow\)

7a. \(\text{canon}(\{1\}[X])\{N\} \leftarrow N \geq N_1 + 1 \land \text{canon}(X)\{N_1\}\)

8a. \(\text{trans}(\{0\}[X], \{1\}[X])\{0\} \leftarrow\)

9a. \(\text{trans}(\{1\}[X], \{1\}[Y])\{N\} \leftarrow N \geq N_1 + 1 \land \text{trans}(X, Y)\{N_1\}\)

As already mentioned, the annotated program \(\alpha_1(P_1)\) can be considered as a logic program where the annotation variables are taken as extra arguments. The annotated program \(\alpha_1(P_1)\) is terminating because \(N \geq N_1 + \ldots + N_k + 1\) implies that, for \(i = 1, \ldots, k, N > N_i\). (Also \(P_1\) is terminating, but we need not use this property.) Now, let us construct a totally correct transformation by using unfold/fold transformation rules for annotated programs. The unfolding and folding rules for annotated programs work exactly like the rules for non-annotated programs, by considering the annotation variables as extra arguments.

By applying several times the unfolding rule, from clause 1a we derive:

10a. \(\text{thm}([[]])\{N\} \leftarrow N \geq 2\)

11a. \(\text{thm}(\{0\}[X])\{N\} \leftarrow N \geq N_1 + N_2 + 5 \land \text{gen}(X)\{N_1\} \land \text{canon}(X)\{N_2\}\)

12a. \(\text{thm}(\{0\}[X])\{N\} \leftarrow N \geq N_1 + N_2 + N_3 + 5 \land \text{gen}(X)\{N_1\} \land \text{trans}(X, Y)\{N_2\} \land \text{test}(Y)\{N_3\}\)

Now we apply the goal replacement rule and we replace the annotated atom \(\text{test}(\{1\}[Y])\{N_3\}\) by \(N_3 \geq N_4 \land \text{test}(Y)\{N_4\}\). This replacement is justified by the following two properties: (1) \(M(P_1) \models \forall Y (\text{test}(\{1\}[Y]) \leftrightarrow \text{test}(Y))\) and (2) \(M(\alpha_1(P_1)) \models \forall Y \forall N_3 (\text{test}(\{1\}[Y])\{N_3\} \rightarrow \exists N_4(N_3 \geq N_4 \land \text{test}(Y)\{N_4\}))\).

By applying the goal replacement rule clause 12a is replaced by the following clause:

13a. \(\text{thm}(\{0\}[X])\{N\} \leftarrow N \geq N_1 + N_2 + N_4 + 5 \land \text{gen}(X)\{N_1\} \land \text{trans}(X, Y)\{N_2\} \land \text{test}(Y)\{N_4\}\)

By folding clauses 11a and 13a using clauses 4a and 5a we get:

14a. \(\text{thm}(\{0\}[X])\{N\} \leftarrow N \geq N_1 + N_3 + 4 \land \text{gen}(X)\{N_1\} \land \text{test}(X)\{N_3\}\)

Finally, by folding clause 14a using clause 1a we derive:

15a. \(\text{thm}(\{0\}[X])\{N\} \leftarrow N \geq N_0 + 3 \land \text{thm}(X)\{N_0\}\)

The final annotated program is \(\alpha_2(P_2) = (\alpha_1(P_1) - \{1a\}) \cup \{10a, 15a\}\). Notice that in clause 15a the annotation of the head is greater than the annotation of the body atom (because \(N \geq N_0 + 3\) implies \(N > N_0\)). Thus, \(\alpha_2(P_2)\) is terminating and \(M(\alpha_2(P_2))\) is the unique fixpoint of \(T_{\alpha_2(P_2)}\). By the unique fixpoint principle [3,7], we deduce that \(M(\alpha_1(P_1)) \subseteq M(\alpha_2(P_2))\).
Now, let us consider the program $P_2$ obtained by dropping the annotations from $\alpha_2(P_2)$. That is, $P_2 = (P_1 - \{1\}) \cup \{10, 15\}$, where clauses 10 and 15 are the following:

10. $\text{thm}([]) \leftarrow$
15. $\text{thm}([X]) \leftarrow \text{thm}(X)$

Notice that, for every ground atom $A$, we have that $A \in M(P_1)$ iff there exists a non-negative integer $n$ such that $A(n) \in M(\alpha_1(P_1))$, and similarly, $A \in M(P_2)$ iff there exists a non-negative integer $n$ such that $A(n) \in M(\alpha_2(P_2))$. Therefore, from $M(\alpha_1(P_1)) \subseteq M(\alpha_2(P_2))$ it follows that $M(P_1) \subseteq M(P_2)$. Since, as already mentioned, the transformation rules act on non-annotated programs like the usual unfold/fold transformations, and these transformations are partially correct, we also have $M(P_1) \supseteq M(P_2)$. Thus, the transformation of $P_1$ into $P_2$ is totally correct.

References