Constraint-based correctness proofs for logic program transformations

Alberto Pettorossi¹, Maurizio Proietti² and Valerio Senni¹

¹ Dipartimento di Informatica, Sistemi e Produzione, Università di Roma Tor Vergata, Via del Politecnico 1, 00133 Rome, Italy.
E-mail: pettorossi@info.uniroma2.it, senni@info.uniroma2.it
² IASI-CNR, Viale Manzoni 30, 00185 Rome, Italy, email: maurizio.proietti@iasi.cnr.it

Abstract. Many approaches proposed in the literature for proving the correctness of unfold/fold transformations of logic programs make use of measures associated with program clauses. When from a program $P_1$ we derive a program $P_2$ by applying a sequence of transformations, suitable conditions on the measures of the clauses in $P_2$ guarantee that the transformation of $P_1$ into $P_2$ is correct, that is, $P_1$ and $P_2$ have the same least Herbrand model. In the approaches proposed so far, clause measures are fixed in advance, independently of the transformations to be proved correct. In this paper we propose a method for the automatic generation of clause measures which, instead, takes into account the particular program transformation at hand. During the application of a sequence of transformations we construct a system of linear equalities and inequalities over nonnegative integers whose unknowns are the clause measures to be found, and the correctness of the transformation is guaranteed by the satisfiability of that system. Through some examples we show that our method is more powerful and practical than other methods proposed in the literature. In particular, we are able to establish in a fully automatic way the correctness of program transformations which, by using other methods, are proved correct at the expense of fixing in advance sophisticated clause measures.

Keywords: Constraints; Logic programming; Program correctness; Program transformation; Transformation rules.

1. Introduction

Rule-based program transformation is a program development methodology by which one derives, starting from an initial program, a final program by applying a sequence of transformation rules [BD77, TS84]. The initial program can be regarded as a formal specification of a software module, while the final program can be regarded as an implementation of that specification. The fact that the rules preserve the intended semantics guarantees that the final program is correct by construction. In logic programming [Apt90, Llo87] program transformation is a deductive process. Indeed, programs are logical formulas and the transformation rules are rules for deducing new formulas from old ones. The logical soundness of the transformation rules implies that a transformation is partially correct, which means that an atomic formula is true in the final program only if it is true in the initial program. However, it is usually much harder to prove that a transformation is totally correct, which means that an atomic formula is true in the initial program if and only if it is true in the final program.

Correspondence and offprint requests to: M. Proietti, E-mail: maurizio.proietti@iasi.cnr.it

In particular, the transformations obtained by applying transformation rules such as unfolding and folding, which basically consist in applying equivalences that hold in the least Herbrand model of the initial program, are always partially correct. However, the final program derived by unfolding and folding may terminate (with respect to a suitable notion of termination) less often than the initial one. For instance, let us consider the program:

\[ P : p \leftarrow q \quad r \leftarrow q \quad q \leftarrow \]

The least Herbrand model of \( P \) is \( M(P) = \{p, q, r\} \) and \( M(P) \models p \leftrightarrow q \). If we replace \( q \) by \( p \) in \( r \leftarrow q \) (that is, we fold \( r \leftarrow q \) using \( p \leftarrow q \)), then we get:

\[ Q : p \leftarrow q \quad r \leftarrow p \quad q \leftarrow \]

The transformation of \( P \) into \( Q \) is totally correct, because \( M(P) = M(Q) \). However, if we replace \( q \) by \( p \) in \( p \leftarrow q \) (that is, we fold \( p \leftarrow q \) using \( p \leftarrow q \) itself), then we get:

\[ R : p \leftarrow p \quad r \leftarrow q \quad q \leftarrow \]

and the transformation of \( P \) into \( R \) is partially correct, because \( M(P) \supseteq M(R) \), but it is not totally correct, because \( M(P) \neq M(R) \). Indeed, \( p \notin M(R) \) because program \( R \) does not terminate for the goal \( p \).

A sufficient condition for the total correctness of a transformation obtained by the unfolding and folding rules is that termination is preserved, that is, the final program terminates as often as the initial one. In particular, total correctness is guaranteed if the final program obtained by transformation always terminates. This method for proving total correctness is the one proposed in Burstall and Darlington’s seminal paper [BD77] and is sometimes referred to as McCarthy’s method [McC63]. However, the termination condition may be, in practice, very hard to verify. For this reason, some methods proposed in the context of functional programming are based on properties of the transformations that imply the preservation of termination, without actually having to verify the termination condition. For instance: (i) [Kot78] shows that under suitable assumptions, total correctness is guaranteed if the final program is derived by a sequence of transformations where the number of applications of the unfolding rule is not smaller than the number of applications of the folding rule, and (ii) [San96] identifies some applicability conditions for the unfolding and folding rules which ensure that the number of steps needed to evaluate a given expression is not increased and, therefore, program termination is preserved.

A lot of work has also been devoted to devise methods for proving the total correctness of transformations of logic programs (see, for instance, [BC94, BCE92, CG94, EG96, GK94, KF86, LOPP95, Mah87, Mah93, PP08, RKRR02, RKRR04, Sek91, TS84, TS86]). The simplest among these methods consists in considering invertible transformation rules, that is, rules which allow a program \( P_1 \) to be transformed into a program \( P_2 \), if the transformation can be reversed to return to \( P_1 \) [Mah87, Mah93]. The total correctness of a transformation obtained by an invertible rule immediately follows from the fact that, by partial correctness, both \( M(P_1) \subseteq M(P_2) \) and \( M(P_2) \subseteq M(P_1) \) hold. For instance, the transformation of program \( P \) into program \( Q \) shown above is invertible because \( Q \) can be transformed back into \( P \) by unfolding the clause \( r \leftarrow p \). On the contrary, the transformation of \( P \) into \( R \) is not invertible. Unfortunately, this method of guaranteeing total correctness is of very limited use because many relevant transformations are not invertible (in particular, those transformations that derive recursive definitions from nonrecursive ones).

Other methods [CG94, LOPP95] propose sufficient conditions for total correctness which are explicitly based on the preservation of suitable termination properties such as the universal or the existential termination. However, as already mentioned, termination conditions may be, in practice, very hard to verify.

Some other methods, which we may call history-based methods, are based on conditions on the sequence of applications of the transformation rules that do not deal with termination explicitly, but nevertheless guarantee that termination is preserved. A notable example of these history-based methods is presented in [KF86], where integer counters are associated with program clauses. The counters of the initial program are set to 1 and are incremented (or decremented) when an unfolding (or folding, respectively) takes place. A sequence of transformations is totally correct if the counters of the clauses of the final program are all positive. This result can be viewed as an extension to logic programming of the approach presented in [Kot78].