Using Abstract Interpretation for Goal Replacement

Dmitri Boulanger*

Department of Computer Science, Katholieke Universiteit Leuven
Heverlee, Belgium

Maurice Bruynooghe†

Department of Computer Science, Katholieke Universiteit Leuven
Heverlee, Belgium

Abstract

An abstract interpretation framework for goal replacement in abstract SLD-like derivations is presented. The approach allows to apply correctly goal replacement at static analysis time and to derive non trivial transformations of definite logic programs. These transformations are more general than the well known unfold/fold and goal replacement transformations of Tamaki and Sato which are based upon goal replacement in the bodies of clauses. The transformation of a program is controlled by a complete abstract OLD-T-like tree. The transformation operations include unfolding/folding and goal replacement.

1 Introduction

The most popular approaches for transforming definite logic programs applying unfold/fold and goal replacement operations on the clauses of a source program. Unfold/fold transformations have been introduced in [4] and were adapted to logic programming in [19, 10, 18]. The recent paper [8]¹ revisits the framework developed by Tamaki and Sato.

In this paper we are developing another approach for transforming logic programs. Namely, we split the transformation process in two phases: a first phase performs a complete static data flow analysis of the source program, the second phase uses the results of the first phase to drive the transformation of the source logic program (here we follow some basic ideas of [9]). In [2, 3] we have introduced a rather general and powerful approach to derive unfold/fold transformations of definite logic programs by abstract interpretation. The unfold/fold transformations, which can be obtained in this way, have been shown to be more general than those of Tamaki and Sato. Indeed, as pointed out in [8], SLD-like tree analysis can produce transformations which cannot always be

*Supported by the Katholieke Universiteit Leuven. Permanent address: Keldysh Institute for Applied Mathematics, Russian Academy of Science, Miusskaya sq., 4, 125047 Moscow, Russia.

†Supported by the Belgian National Fund for Scientific Research

¹The examples presented in this paper show that the approach of Tamaki and Sato is not sound in some cases.

expressed in terms of [19, 10, 18]. However the SLD-like tree analysis of [13, 8] does not allow the derivation of recursive predicates, while in our approach we can easily obtain new recursive predicates generating transformations, which can be better than those of [16, 17]. Moreover, it can be shown that, by restricting the static analysis phase, our transformation algorithm reduces to that [7]. The main motivation of the paper is to facilitate the application of goal replacement operations when constructing the OLDT-like trees during the static analysis phase. This results in a rather powerful framework for transforming logic programs which is based on abstract interpretation.

But there is a serious problem when attempting to incorporate goal replacement operations in our framework. In order to perform the static analysis, we need complete finite SLD-like trees, which are constructed by abstracting or generalising certain parts of the concrete derivations. This creates a problem when attempting to apply goal replacement in partially abstracted derivations.

The problem with goal replacement in partially abstracted OLDT-like derivations is that the conditions for correct goal replacement are very strict. One needs exact information about the binding of certain crucial variables occurring in the goal to be replaced. The latter is not always the case for abstract derivations as described in [3]. Consequently, analysis time can only suggest goal replacement. Some conditions have to be checked at transformation time. Moreover, part of the analysis has to be redone when these conditions are violated. The central idea of our approach is to provide a proof that the crucial variables are not influenced by the abstraction of some parts of the derivation. In order to implement this idea we introduce a special abstract interpretation domain, which includes abstract and concrete variables.

The rest of the paper is organised as follows. After some preliminaries, where we introduce some basic concepts, the conditions for goal replacement in SLD-like derivations are described in section 3. Here we introduce necessary conditions for the soundness and completeness of goal replacement for a special variant of the SLD resolution. Section 4 presents our abstract interpretation technique — we show how the soundness and completeness of the goal replacement can be ensured at static analysis time, i.e. for partially abstracted derivations. An extensive example, which illustrates all the most important details of goal replacement in abstract derivations, can also be found in this section.

2 Preliminaries

In what follows the standard terminology and the basic knowledge of the theory of logic programming, as can be found in [14], is assumed. We will use the standard notions of SLD derivation and refutation, unification and substitution. Some knowledge concerning OLDT resolution [20, 11] and goal replacement [19, 10, 18, 8] will be important.

We consider a definite logic program P to be a collection of definite clauses $\{c_1, \dots, c_n\}$ equipped with an initial goal G_0 . The successive goals in a SLD derivation for the initial goal wrt P will be denoted by G_i , $i = 0, 1, \dots, n$.

The capital letters B, Q and R will be used to denote conjunctions of atoms, which will be considered as collections and are called *blocks* in the sequel. $B(\bar{X})$ will denote a conjunction of atoms B , where $\bar{X}$ is the set of variables occurring

in B . By $var(E)$ we denote the set of variables, occurring in the syntactical object E . The greek letters θ , σ and φ will be used to denote substitutions. Given a substitution θ , $\theta|_{var(E)}$ will denote the restriction of the substitution θ to the variables occurring in E .

In the sequel, by equivalent programs we mean programs giving the same computed answer substitutions for the initial goal of the program [15].

3 Goal Replacement for Extended OLD resolution

In this section we discuss goal replacement in SLD-like derivations and the possibility to apply tabulation for this kind of derivations. Extended OLD resolution deviates from standard SLD resolution only by imposing certain restrictions on the computation rule.

Namely, the SLD derivation $G_0 \xrightarrow{c_1 \theta_1} G_1 \dots \xrightarrow{c_n \theta_n} G_n$ is an *EOLD* derivation for the initial goal G_0 wrt a definite logic program P provided that any goal G_i in the derivation is represented as an *ordered* sequence $\langle B_1, B_2, \dots, B_m \rangle$, $m \geq 0$ of conjunctions of atoms (blocks) and the computation rule always selects an atom from the first block. A block will always be considered as a *non empty collection* of atoms. Given a goal G represented as an ordered sequence of blocks, $||G||$ will denote the collection of all atoms occurring in G (In OLD resolution [20, 11] blocks have only one atom). In the sequel we do not distinguish atoms, blocks and goals which differ only by a variable renaming. The following definition gives a more precise description of an *EOLD* resolution step.

Definition 1 *EOLD Resolution Step*

Let G_i be a goal represented as $\langle B_1, B_2, \dots, B_m \rangle$, $m \geq 1$, A an atom selected from B_1 , c_{i+1} a standardised apart input clause from P and θ_{i+1} the *mgu* of A and the head of the clause c_{i+1} . Then G_{i+1} is derived from G_i and c_{i+1} using *mgu* θ_{i+1} via the *EOLD* computation rule if G_{i+1} is obtained by applying the following two steps:

- construct the goal $G' = \langle B'_1, \dots, B'_n \rangle$, where
 - if B_1 is singleton and the body of c_{i+1} is empty then $n = m - 1$,
 $B'_i = B_{i+1}$, $1 \leq i \leq n$
 - else $n = m$, $B'_1 = B_1$ with A replaced by the body of c_{i+1} and $B'_i = B_i$,
 $1 \leq i \leq m$
- partition B'_1 into arbitrary chosen non empty parts (the new blocks) $B'_{11}, \dots, B'_{1k}$, $k \geq 1$ and let $G_{i+1} = \langle B'_{11}, \dots, B'_{1k}, B'_2, \dots, B'_n \rangle \theta_{i+1}$, i.e. the first block can be splitted in a sequence of new non empty blocks.

□

Notice that if a goal G of the *EOLD* refutation is splitted in a sequence of several blocks, then atoms not occurring in the first block of G can be selected only after the first block is completely refuted. This property is crucial for the following definition, which will play an important role in the sequel.

Definition 2 *EOLD Subrefutation*

An *EOLD* subderivation of the form $G_i \xrightarrow{c_{i+1} \theta_{i+1}} G_{i+1} \dots \xrightarrow{c_n \theta_n} G_n$ is an *EOLD* subrefutation for the first block B_1 of the goal $G_i = \langle B_1, B_2, \dots, B_k \rangle$, $k \geq 1$ if either $G_n = \langle \rangle$ and $k = 1$ (i.e. the subderivation is a refutation for B_1) or $\| \langle B_2, \dots, B_k \rangle \theta_{i+1} \dots \theta_n \| = \| G_n \|$ and there is no goal G_j , $i < j < n$ having this property in the subderivation. $\square$

This definition implies that a block can be considered as an "extended" atom having a subrefutation including only resolution steps which are necessary to refute atoms occurring in the block. This is a crucial observation when combining *EOLD* resolution with tabulation.

Notice that *EOLD* resolution is sound and complete because soundness and completeness of SLD resolution is independent from the computation rule [14].

In what follows we address the problem of correct goal replacement for *EOLD* derivations. When replacing some atoms of the current goal of a derivation, the connection between variables occurring in these atoms on one hand, and the variables occurring in the initial goal and the remaining part of the current goal on the other hand, are crucial for the correctness of the replacement. The following definition identifies variables which can be safely dropped in a goal replacement operation.

Definition 3 *Local variables in a EOLD derivation*

Given an *EOLD* derivation $G_0 \xrightarrow{c_1 \theta_1} G_1 \dots \xrightarrow{c_n \theta_n} G_n$ for the initial goal G_0 and the corresponding accumulated substitution $\sigma_n = \theta_1 \dots \theta_n$, a variable X occurring in the goal G_n , $n > 0$ is local to G_n iff the variable X does not occur outside the first block of the goal G_n and for all bindings $Y \leftarrow t \in \sigma_n$ holds: $X \in \text{var}(Y \leftarrow t)$ implies $\text{var}(Y \leftarrow t) \cap \text{var}(G_0) = \emptyset$, i.e. X does not occur in $\text{var}(G_0)$ and it is not connected by the accumulated substitution to the variables of the initial goal.

The set of local variables in the goal G_i of the derivation $G_0 \rightarrow G_1, \dots, G_i$ will be denoted by $\text{local}(G_i)$, $i = 0, \dots, n$. The set of all non local variables occurring in the goal G_i will be denoted by $\neg \text{local}(G_i)$. $\square$

Example 1 Consider the goals in the "skeleton" of an *EOLD* derivation and the corresponding sets of local variables:

- $G_0 = \langle B_0(X) \rangle \quad \text{local}(G_0) = \emptyset$
- ↓
- $G_1 = \langle B_1(Y, Z), B_2(X, Z, U), B_3(X, U) \rangle \quad \text{local}(G_1) = \{Y\} \quad \theta_1 = \{ \}$
- ↓
- $G_2 = \langle B_2(X, Z, U), B_3(X, U) \rangle \quad \text{local}(G_2) = \{Z\} \quad \theta_2 = \{Y \leftarrow Z\}$
- ↓
- $G_3 = \langle B'_3(Z) \rangle \quad \text{local}(G_3) = \emptyset \quad \theta_3 = \{X \leftarrow f(Z), U \leftarrow g(Z)\}$

Notice that $G_1 \rightarrow G_2$ is the subrefutation for the block B_1 and $G_2 \rightarrow G_3$ is the subrefutation for block B_2 , where $B'_3(Z) = B_3(X, U)\theta_3 = B_3(f(Z), g(Z))$. $\square$

The following definition (it is a slightly weaker version of that [8], see below) simply claims what goals should be considered as equivalent ones.

Definition 4 *Permissible lemma*

Given a definite logic program P , a lemma L of the form

$$Q(\bar{X}, \bar{Y}) \longleftrightarrow R(\bar{X}, \bar{Z}), \quad \bar{X} \cap \bar{Y} = \bar{X} \cap \bar{Z} = \bar{Y} \cap \bar{Z} = \emptyset$$

is a permissible lemma wrt P provided that for all substitutions σ

- $P \models \forall Q\sigma$ implies $P \models \forall R\varphi$ for some substitution φ such that $\varphi|_{\bar{X}} = \sigma|_{\bar{X}}$
- $P \models \forall R\sigma$ implies $P \models \forall Q\varphi$ for some substitution φ such that $\varphi|_{\bar{X}} = \sigma|_{\bar{X}}$

□

This definition implies that if a permissible lemma is used to reconstruct the current goal of a derivation then the number and the order of the answers for the initial goal is *not* preserved. The preservation of the number and the order of answers is considered to be important when dealing with Prolog programs in the context of partial evaluation (cf. [13]). On the other hand the latter can be very restrictive when deriving non trivial transformations similar to [16, 17, 1, 5].

Example 2 Consider the following definite logic program P

$$\begin{aligned} &app([], L, L). \\ &app([H|T], L, [H|X]) \leftarrow app(T, L, X). \end{aligned}$$

The lemma $L: app(X, [H], D), app(D, L, Y) \longleftrightarrow app(X, [H|L], Y)$ is a permissible lemma wrt P , where $\bar{X} = \{X, Y, L, H\}$, $\bar{Y} = \{D\}$ and $\bar{Z} = \emptyset$. □

In the sequel we will consider definite logic programs P equipped with a set $\mathcal{L}_P$ of lemmas permissible wrt P . We will also assume, that the set of permissible lemmas $\mathcal{L}_P$ satisfies the following simple conditions²:

- any lemma $L \in \mathcal{L}_P$ has the "clausal" form $Q \leftarrow R$, i.e. it can only be applied to replace the goal Q by the goal R (cf. def.5 below), but not vice versa.
- for any two lemmas $L, L' \in \mathcal{L}_P$ having the forms $Q \leftarrow R$ and $Q' \leftarrow R'$ respectively holds that Q and Q' are not unifiable.

A lemma can be applied to the first block of the current goal only if it preserves bindings for the variables occurring in the initial goal (def.4 does not ensure this). Consider the following definition:

Definition 5 *Correct lemma application*

Given the current goal $G_n = \langle Q(\bar{X}, \bar{Y}), B_2, \dots, B_m \rangle$ of an *EOLD* derivation wrt a definite logic program P , the replacement of the goal G_n by the goal $G'_n = \langle R(\bar{X}, \bar{Z}), B_2, \dots, B_m \rangle$ using a renaming of a permissible lemma from $\mathcal{L}_P$ of the form $Q(\bar{X}, \bar{Y}) \leftarrow R(\bar{X}, \bar{Z})$ is correct if $\bar{Y} \subseteq local(G_n)$ and the variables $\bar{Z}$ are fresh variables. The variables $\bar{Y}$ are said to be *dead end* variables. □

²These conditions are not strictly necessary – we use them to simplify the definitions and the proofs below. The first condition is necessary to avoid the introducing of trivial loops by lemma application, while the second one is used to make lemma application deterministic.

The *EOLD* resolution with correct application of the permissible lemmas is intended to be an extension of the normal *EOLD* resolution. The definition below gives more details:

Definition 6 *EOLD resolution with lemma application*

Given a definite logic program P , the corresponding set of permissible lemmas $\mathcal{L}_P$ and an initial goal G_0 , the *EOLD* resolution with lemma application deviates from the *EOLD* resolution (cf. def.1) by imposing the following restriction upon the computation rule: if there exists a renaming of a lemma $L \in \mathcal{L}_P$ having the form $Q \leftarrow R$, such that it can be applied correctly to the first block of the current goal (the latter means that the block has to be identical to Q), then it *has* to be applied (the first block Q has to be replaced by block R), i.e. the clauses of the program P are not allowed to be used for resolving atoms of Q . $\square$

Example 3 Consider example 2. Then the following sequence is the part of an *EOLD* derivation with (correct) lemma application

$\bullet \dots$
 $\downarrow$
 $\bullet G_i = \langle \text{app}(X, [H], D), \text{app}(D, L, Y) | \dots \rangle$ $c_i : \theta_i = \{\dots\}$
 $\Downarrow$
 $\bullet G_{i+1} = \langle \text{app}(X, [H|L], Y) | \dots \rangle$ $L \in \mathcal{L}_P : \theta_{i+1} = \{\}$
 $\downarrow$
 $\bullet \dots$
 provided that the dead end variable D is local to the goal G_i (the blocks in the goals above are separated by the symbol "|"). $\square$

In the sequel we will denote a definite logic program P equipped with a set of permissible lemmas $\mathcal{L}_P$ by $P \oplus \mathcal{L}_P$. If the set $\mathcal{L}_P$ is empty, then *EOLD* resolution with lemma application (it will be denoted by *EOLD_L*) wrt $P \oplus \mathcal{L}_P$ is exactly *EOLD* resolution (def.1), but if the set of lemmas $\mathcal{L}_P$ is not empty, then in general completeness cannot be ensured. The following proposition shows, that permissibility of the lemmas from $\mathcal{L}_P$ and correctness of their application in *EOLD_L* derivations are sufficient to ensure the soundness of *EOLD_L* resolution. However, completeness is not ensured as example 4 below shows.

Proposition 1 Consider a definite logic program P , an initial goal G_0 and a set $\mathcal{L}_P$ of permissible lemmas: if the goal G_0 has an *EOLD_L* refutation $\mathcal{R}_L$ wrt $P \oplus \mathcal{L}_P$ with computed answer substitution θ_L , then the goal G_0 has an *EOLD* refutation $\mathcal{R}$ with computed answer substitution θ , such that $G_0\theta = G_0\theta_L$.

Proof The proof is by induction on the number of lemma application in the refutation $\mathcal{R}_L$.

Base Case. If the refutation $\mathcal{R}_L$ does not contain lemma applications it is a an *EOLD* refutation. Thus, the proposition holds for $n = 0$.

Induction Step. Assume the proposition holds for *EOLD_L* refutations with $n - 1$ lemma applications. Consider a refutation $\mathcal{R}_L$ with n lemma applications. Let $G_i = \langle Q(\bar{X}, \bar{Y}) | B_2, \dots, |B_n \rangle$ be the goal where the first lemma application is performed with the lemma $Q(\bar{X}, \bar{Y}) \leftarrow R(\bar{X}, \bar{Z})$. Let θ denote the substitution accumulated at G_i . Then we have $G_{i+1} = \langle R(\bar{X}, \bar{Z}) | B_2, \dots, |B_n \rangle$.

After completion of the subrefutation for $R(\overline{X}, \overline{Z})$ with accumulated substitution φ (it has at most $n - 1$ lemma applications) we obtain the goal $G_j = \langle |B_2|, \dots, |B_n| \rangle \varphi$. Because $\overline{Z}$ is local in G_{i+1} we have $\theta\varphi = \theta\varphi|_{\overline{X}}$ and $\langle |B_2|, \dots, |B_n| \rangle \varphi = \langle |B_2|, \dots, |B_n| \rangle \varphi|_{\overline{X}}$.

The induction hypothesis is applicable on the subrefutation for $R(\overline{X}, \overline{Z})$, so an *EOLD* refutation exists for this goal. As the lemma is permissible, there exists an *EOLD* refutation for $Q(\overline{X}, \overline{Y})$ with accumulated substitution σ such that $\sigma|_{\overline{X}} = \varphi|_{\overline{X}}$.

Now we can change the refutation as follows: starting from the goal G_i , we use the subrefutation for $Q(\overline{X}, \overline{Y})$ with accumulated substitution σ . This leads to the goal $G'_j = \langle |B_2|, \dots, |B_n| \rangle \sigma$. Because $\overline{Y}$ is local in G_i , we have $\langle |B_2|, \dots, |B_n| \rangle \sigma = \langle |B_2|, \dots, |B_n| \rangle \sigma|_{\overline{X}} = G'_j$, and as $\sigma|_{\overline{X}} = \varphi|_{\overline{X}}$, we have $G_j = G'_j$.

Thus from here on we can resume the original refutation as it continued from G_j . With ρ the substitution accumulated in the remainder of the refutation, we obtain correct answer substitution $\theta\sigma\rho|_{var(G_0)} = (\theta\sigma|_{\overline{X}}\rho)|_{var(G_0)}$ which is identical to the $\theta\varphi\rho$ of the original refutation as $\theta\varphi\rho|_{var(G_0)} = (\theta\varphi|_{\overline{X}}\rho)|_{var(G_0)}$ and $\sigma|_{\overline{X}} = \varphi|_{\overline{X}}$.

So we now have an *EOLD_C* refutation with at most $n - 1$ lemma applications. Applying the induction hypothesis shows that an *EOLD* refutation exists. This completes the proof. $\square$

The following example shows that the permissibility and correctness of lemma application are not sufficient to ensure the completeness of the *EOLD_C* resolution. The completeness can be lost due to the "loops" which can be introduced into derivations by lemma application. The following example gives some hints what kind of extra conditions are necessary.

Example 4 Given the definite logic program

$$P : \{ p(a). p(X) \leftarrow q(X). q(a). q(f(X)) \leftarrow q(X). \},$$

the lemma $L: q(X) \leftarrow p(X)$ is a permissible lemma whose application is always correct. However *any* *EOLD_C* refutation wrt $P \oplus \{q(X) \leftarrow p(X)\}$ for the initial goal $p(X)$ has only $\{X \leftarrow a\}$ as a computed answer substitution, i.e. the set of answer substitutions is not complete for the initial goal $p(X)$. Notice that incompleteness is due to tautologies (or loops) occurring in the derivations with lemma application. $\square$

The following definition introduces extra conditions, which ensure the completeness of the *EOLD_C* resolution (there is an example below).

Definition 7 *Safe set of lemmas*

A set $\mathcal{L}_P$ of lemmas permissible wrt a definite logic program P is said to be safe if for any lemma $L \in \mathcal{L}_P$ having the form $Q(\overline{X}, \overline{Y}) \leftarrow R(\overline{X}, \overline{Z})$ holds: if the goal Q has an *EOLD* refutation with computed answer substitution θ , then the goal R has an *EOLD_C* refutation wrt $P \oplus \mathcal{L}_P$ with computed answer substitution $\theta_{\mathcal{L}}$ such that $\theta|_{\overline{X}} = \theta_{\mathcal{L}}|_{\overline{X}}$. $\square$

The condition above is very close to that of [8] and is likely as permissive (a detailed comparison is not possible due to the basic difference in the approaches, see the discussion below).

Notice that for the permissible lemma $L: q(X) \leftarrow p(X)$ from example 4 holds: if $q(X)$ has an *EOLD* refutation, then $p(X)$ has an *EOLD_C* refutation. But not all computed answer substitutions of *EOLD* refutations for $q(X)$ are covered by the *EOLD_C* refutations of $p(X)$. On the other hand, if the clause $p(f(X)) \leftarrow p(X)$ is added to the program P , the lemma set $\{ q(X) \leftarrow p(X) \}$ is safe and *EOLD_C* resolution wrt $P \cup \{p(f(X)) \leftarrow p(X)\} \oplus \{q(X) \leftarrow p(X)\}$ becomes complete. Although the lemma L introduces a loop, it is not a dangerous one due to the added clause (The conditions of [8] can be used to produce exactly the same results for this example). Notice that the lemma set $\{ q(X) \leftarrow p(X) \ p(X) \leftarrow q(X) \}$ is not safe, i.e. it introduces a "dangerous" loop.

The safety of the set of permissible lemmas combined with the *EOLD* computation rule ensure the completeness of *EOLD_C* resolution:

Proposition 2 Consider a definite logic program P , an initial goal G_0 and a safe set $\mathcal{L}_P$ of lemmas. If the goal G_0 has an *EOLD* refutation $\mathcal{R}$ with computed answer substitution θ then the goal G_0 has an *EOLD_C* refutation $\mathcal{R}_C$ wrt $P \oplus \mathcal{L}_P$ with computed answer substitution θ_C , such that $\theta|_{\text{var}(G_0)} = \theta_C|_{\text{var}(G_0)}$.

Proof Let $\mathcal{R}$ be an *EOLD* refutation for G_0 . Consider the first goal G in $\mathcal{R}$ such that its first block Q can be replaced correctly by a permissible lemma L from $\mathcal{L}_P$ (if such goal G does not exist, then $\mathcal{R}$ is an *EOLD_C* refutation). Then the safety of the set $\mathcal{L}_P$ ensures that the *EOLD* subrefutation of Q can be replaced by the corresponding *EOLD_C* subrefutation. The remaining part of $\mathcal{R}$ can be treated in the same way obtaining a derivation, which is an *EOLD_C* refutation having the desired computed answer substitution. $\square$

It is important to notice, that the completeness result above relies on the restrictions imposed by the *EOLD* computation rule (cf. def. 1 and 2). Consider the following example:

Example 5 Given the definite logic program $P: \{ p(a), q(a) \}$. The set

$$\{ L_1: p(X) \leftarrow q(X) \quad L_2: q(X), q(Y) \leftarrow p(X), p(Y) \}$$

is a safe set $\mathcal{L}_P$ of permissible lemmas wrt P . If *EOLD* restrictions are dropped, then there exists a computation rule such that the initial goal $q(X), q(Y)$ has only the following infinite derivation wrt $P \oplus \mathcal{L}_P$ (it can be considered as a infinitely failed derivation):

$$\langle q(X), q(Y) \rangle \xrightarrow{L_2} \langle p(X), p(Y) \rangle \xrightarrow{L_1} \langle q(X), p(Y) \rangle \xrightarrow{L_1} \langle q(X), q(Y) \rangle \dots$$

Notice that the derivation above is not an *EOLD_C* derivation, because the attempt to refute the atom $p(Y)$ occurring in the third goal is made before the refutation of the firstly selected atom $p(X)$ of the second goal is complete. $\square$

The *EOLD* restrictions upon the computation rule (cf. def. 1) combined with the conditions above, which ensure the soundness and completeness of the *EOLDL* resolution imply the following proposition:

Proposition 3 *Let P be a definite logic program, G_0 an initial goal and $\mathcal{L}_P$ a safe set of lemmas. Suppose there is an $EOLD_{\mathcal{L}}$ refutation for $P \oplus \mathcal{L}_P \cup G_0$ with computed answer substitution σ . Then for any $EOLD_{\mathcal{L}}$ computation rule R (i.e. the specification of the choice of the selected atom in the first block and of the way the first block is splitted in a sequence of new blocks) there exists an $EOLD_{\mathcal{L}}$ refutation for $P \oplus \mathcal{L}_P \cup G_0$ via R with R -computed answer substitution σ' such that $G_0\sigma = G_0\sigma'$.*

Proof The proof can be obtained from the independence of the computation rule of the *EOLD* resolution combined with the soundness and completeness of the $EOLD_{\mathcal{L}}$ resolution.

□

In general, the safety conditions for the set of permissible lemmas (def. 7) are rather difficult to verify. So the following definition gives more restrictive conditions, which are still sufficiently interesting for a lot of applications and can be checked more easily:

Definition 8 *Set of non trivial permissible lemmas*

A set $\mathcal{L}_P$ of lemmas permissible wrt a definite logic program P , is said to be a set of non trivial lemmas iff for each lemma $L \in \mathcal{L}_P$ of the form $Q \leftarrow R$ holds:

- Q and R are not unifiable
- there is no *EOLD* derivation of the form $\langle |R| \rangle \xrightarrow{c_1} \dots \xrightarrow{c_n} \langle |Q|, Rest \rangle$
- $EOLD_{\mathcal{L}}$ derivations of the form $\langle |R| \rangle \rightarrow \dots \rightarrow G$ do not contain lemma application steps.

□

It is not difficult to show that any set of non trivial lemmas is a safe set of lemmas. The lemma from example 2 is a non trivial lemma. To be convinced in this it is sufficient to notice that the atom *app* cannot be expanded to *app, app* using clauses of the program P .

The main result of this section is that $EOLD_{\mathcal{L}}$ resolution can be equipped with a tabulation mechanism, which is applied to blocks in exactly the same way as *OLD* resolution [20, 11] is equipped with a tabulation mechanism applied to atoms. Def. 1 and 2 ensure that tabulation applied to blocks is sound and complete provided that $EOLD_{\mathcal{L}}$ resolution is sound and complete wrt the given definite logic program and set of lemmas. Indeed, the *EOLD* restrictions upon the computation rule allow to treat blocks as atoms, while the permissibility and safety conditions allow to treat lemmas as ordinary clauses. The exact proof of the completeness, soundness and independence of the properly restricted computation rule of the $EOLDT_{\mathcal{L}}$ resolution ($EOLD_{\mathcal{L}}$ with tabulation) can be obtained by a slight modification of that in [20, 11].

The purpose of the development of the $EOLDT_{\mathcal{L}}$ resolution mechanism is its application during static analysis to obtain a guide for logic program transformation. This approach allows to integrate powerful unfold/fold transformations with lemma applications. The unfold/fold transformations of definite

logic programs, which can be derived using abstract complete *EOLDT* tree are described in [2, 3]. This kind of transformations is more powerful than that of [19, 8]. On the other hand, the conditions for correct goal replacement presented above are very close to those of [8]. However our approach has the following advantages when used for program transformations using the finite (abstract) *EOLDT_C* tree as a guide:

- Our framework makes use of the connectivity of the variables occurring in the derivation preceding a goal replacement step rather than the relations between the variables in the input clause of the resolvent. This makes our conditions less restrictive.
- Taking into account the whole derivation up to the current goal captures implicitly some influence of the initial goal, and thus allows some replacements, which do not satisfy the conditions of [8], because the latter deals with preservation of minimal models rather than preservation of answers wrt the initial goal.

A more detailed and precise comparison with the approaches of [19, 8] is rather difficult, because the latter deals with particular clauses of the source program and addresses the problem of replacing a goal in the body of a clause. In our setting the properties of the derivations are used to ensure equivalence of the source program and the transformed program. The latter allows to apply abstract interpretation for deriving transformations which preserve the set of computed answer substitutions for the variables of the initial goal. The main problem in our approach is to obtain a finite complete abstract *EOLD_C* tree, which is a correct representation of the concrete one (the latter can be infinite). The next section is devoted to the abstract interpretation framework based upon *EOLDT_C* resolution and presents an example of the transformation based upon unfold/fold and goal replacement operations, which are derived from the abstract finite *EOLDT_C* tree.

4 Deriving Transformations Using Abstract Interpretation

In this section we introduce a special abstract interpretation framework which is based upon *EOLDT_C* resolution and suitable for lemma application during static analysis of definite logic programs. It is developed keeping the conditions for correct lemma application in mind. The basic abstract interpretation device is *EOLD* resolution equipped with a tabulation mechanism and applied to an abstract domain, which includes abstract and concrete terms.

The abstract finite *EOLDT_C* tree constructed for a program P , initial goal G_0 with (possible) applications of lemmas from $\mathcal{L}_P$ can be used to derive the corresponding transformation of P . This transformation preserves the set of answer substitutions for the variables of the initial goal provided that abstract interpretation represents the concrete one. So the main topic of this section is the development of conditions for correct abstract interpretation. (The framework which does not allow lemma application during abstract interpretation was introduced in [2, 3].) We start with a presentation of a special abstract interpretation domain.

4.1 Mixed Abstract Terms and Substitutions

Given a countable set of *concrete variables* $\mathcal{V}$ and a finite set of function symbols $\mathcal{F}$, the set of concrete terms $Term$ is defined as usual [14]. We extend the notion of terms by introducing a countable set of *abstract variables* $\mathcal{W}$ such that $\mathcal{V} \cap \mathcal{W} = \emptyset$. The central idea of the framework is to distinguish at compile time between abstract variables which are intended to be used for representation of imprecise information about the possible bindings, and concrete variables with the real run-time bindings. An superscript α will be used to distinguish abstract variables, for example W^α .

The set of *mixed abstract terms* $Term^{mix}$ is constructed from the set of function symbols $\mathcal{F}$ and the set of variables $\mathcal{V} \cup \mathcal{W}$. *Mixed abstract expressions* over the set $Term^{mix}$ are defined in the standard way.

Definition 9 Recognition

A legal assignment for mixed abstract term t^{mix} with a set of abstract variables $W_1^\alpha, \dots, W_n^\alpha$ is a substitution $\rho = \{W_1^\alpha \leftarrow t_1, \dots, W_n^\alpha \leftarrow t_n\}$ such that

$$var(t^{mix}) \cap var\{t_1, \dots, t_n\} = \emptyset.$$

A legal assignment ρ maps a mixed abstract term t^{mix} into a concrete term $t^{mix}\rho$. $\square$

The *concretization* of a mixed abstract term t^{mix} can be defined as follows: $\gamma(t^{mix}) = \{t \in Term \mid t^{mix} \text{ recognises } t \text{ via some legal assignment}\}$.

A *mixed abstract substitution* is a mapping $\theta^{mix} : \mathcal{W} \cup \mathcal{V} \mapsto Term^{mix}$ such that the domain $dom(\theta^{mix}) = \{U \in \mathcal{W} \cup \mathcal{V} \mid \theta^{mix}(U) \neq U\}$ is finite. The application of a substitution to a mixed abstract term $t\theta^{mix}$ (or to a mixed abstract expression) is defined as usual.

Definition 10 *Preorder on the set of mixed abstract terms and substitutions*
Given mixed abstract terms t, t_1, t_2 and mixed abstract substitutions θ_1, θ_2

- $t_1 \trianglelefteq t_2$ if and only if $\gamma(t_1) \subseteq \gamma(t_2)$
- $t_1 \sim t_2$ if and only if $t_1 \trianglelefteq t_2$ and $t_2 \trianglelefteq t_1$.
- $\theta_1 \trianglelefteq \theta_2$ if and only if for any mixed abstract term t holds $t\theta_1 \trianglelefteq t\theta_2$ (θ_2 is said to be *more general* than θ_1)
- $\theta_1 \sim \theta_2$ if and only if $\theta_1 \trianglelefteq \theta_2$ and $\theta_2 \trianglelefteq \theta_1$.

$\square$

A safe abstract interpretation mechanism is based on the notions of unifier and most general unifier. Namely, we have the following property for mixed abstract terms: given two mixed abstract terms t_1 and t_2 such that the concrete terms $\rho_1(t_1)$ and $\rho_2(t_2)$ are unifiable for some legal assignments ρ_1 and ρ_2 , there exists a mixed abstract substitution σ such that $t_1\sigma = t_2\sigma$ (the *mixed abstract unifier* of t_1 and t_2) and there exist a unique, up to the variable renaming, most general mixed abstract unifier σ' such that it is more general than any other unifier σ of t_1 and t_2 . Then the following proposition is obvious:

Proposition 4 Given two mixed abstract terms t_1 and t_2 with mgu σ . Any pair of mixed abstract terms t'_1 and t'_2 such that $t_1 \trianglelefteq t'_1$ and $t_2 \trianglelefteq t'_2$ have a mgu σ' such that $\sigma \trianglelefteq \sigma'$. $\square$

The following example illustrates the notion of most general unifier:

Example 6 $\sigma = mgu(W^\alpha, X) = \{X \leftarrow W^\alpha\}$ and $\sigma' = \{W^\alpha \leftarrow X\}$ is only a unifier of X and W^α because $X \trianglelefteq W^\alpha$. $\square$

4.2 Mixed Abstract EOLD Resolution with Lemma Application

Before introducing our correct goal replacement technique in abstract $EOLD_C$ derivations we introduce some basic notions, such as mixed abstract program and set of mixed abstract lemmas.

A mixed abstract definite logic program P^{mix} is a set of mixed abstract clauses (mixed abstract expressions) over the set of terms $Term^{mix}$. For example, $\{p(U^\alpha) \leftarrow true\}$, is a mixed abstract definite logic program. The program $P^{mix} = \{c_1^{mix}, \dots, c_n^{mix}\}$ recognises the concrete program $P = \{c_1, \dots, c_n\}$, i.e. $P \in \gamma(P^{mix})$, if there exist legal assignments $\rho_1, \dots, \rho_n$, such that $c_i = \rho_i(c_i^{mix})$, $i = 1, \dots, n$. A set of mixed abstract lemmas $\mathcal{L}_P^{mix}$ is defined in exactly the same way.

Given a mixed abstract program P^{mix} and a mixed abstract initial goal G_0^{mix} , a mixed abstract $EOLD$ derivation is the same as an $EOLD$ derivation provided that the notions of mixed abstract term and the corresponding notion of most general unifier are used (cf. def. 1). The definition of subrefutation (def. 2) is also applicable without any modifications. The proposition 4 ensures correctness of these definitions.

As could be seen from section 3, correct lemma application requires special conditions to be satisfied. The following definition prevents the abstraction of dead end variables.

Definition 11 Local variables of mixed abstract $EOLD$ derivation

Given a mixed abstract $EOLD$ derivation $G_0^{mix} \xrightarrow{c_1^{mix} \theta_1^{mix}} G_1^{mix} \dots \xrightarrow{c_n^{mix} \theta_n^{mix}} G_n^{mix}$ with accumulated substitution σ_n^{mix} , a variable V occurring in the goal G_n^{mix} is a local variable if it is a concrete variable and if:

- the variable V is not connected via σ_n^{mix} to any non local variable occurring in the initial goal G_0^{mix}
- the variable V does not occur outside the first block of the goal G_n^{mix}
- the variable V is not connected via σ_n^{mix} to any abstract variable occurring in the derivation

$\square$

The main property of a mixed abstract $EOLD$ resolution is its correctness wrt the concrete one. The following proposition is an immediate consequence of the proposition 4 and the definition above:

Proposition 5 Given a mixed abstract definite logic program P^{mix} with a mixed abstract initial goal G_0^{mix} , a concrete program $P \in \gamma(P^{mix})$ with a concrete initial goal $G_0 \in \gamma(G_0^{mix})$. If there is a concrete EOLD derivation $\mathcal{D}$ of the form

$$G_0 \xrightarrow{c_1 \theta_1} G_1 \dots \xrightarrow{c_n \theta_n} G_n,$$

then there exists a mixed abstract EOLD derivation $\mathcal{D}^{mix}$ of the form

$$G_0^{mix} \xrightarrow{c_1^{mix} \theta_1^{mix}} G_1^{mix} \dots \xrightarrow{c_n^{mix} \theta_n^{mix}} G_n^{mix},$$

and a legal assignment ρ such that $c_i = \rho(c_i^{mix})$, $G_i = \rho(G_i^{mix})$, $local(G_i) \supseteq local(G_i^{mix})$ and $\theta_i \trianglelefteq \theta_i^{mix}$ for all $i = 0, 1, \dots, n$. The derivation $\mathcal{D}$ is said to be represented by the derivation $\mathcal{D}^{mix}$.

Proof The proof is a standard one (see, for example, [9]) and can be obtained by induction on the length of the derivations, if the following property of the proposed abstract interpretation framework is taken into account: a legal assignment always introduces fresh variables (see def. 9). $\square$

Let us provide some important remarks concerning the proposition above. Firstly, notice that the recognition is basically a recognition of the derivation as a whole, i.e. multiple occurrences of variables in the goals have to be concretized simultaneously. Secondly, if there is an occurrence of a local variable in the mixed abstract derivation then this variable always is a local variable in the corresponding concrete derivation. The latter observation is crucial for correct lemma application during abstract interpretation. The following definition gives the properties of the lemmas which should be used during abstract interpretation:

Definition 12 *Permissible mixed abstract lemma*

A permissible mixed abstract lemma L^{mix} wrt a program P^{mix} is a mixed abstract expression such that any of its concretization $L \in \gamma(L^{mix})$ is a permissible lemma wrt any concrete logic program $P \in \gamma(P^{mix})$. $\square$

Example 7 Given the definite logic program P defined in example 2, the mixed abstract lemma L^{mix}

$$app(X^\alpha, [H^\alpha], D), app(D, L^\alpha, Y) \leftarrow app(X^\alpha, [H^\alpha | L^\alpha], Y)$$

is a permissible mixed abstract lemma wrt P . Notice that the dead end variable D is a concrete variable. This is crucial when applying lemmas in abstract derivations — the dead end variables cannot be influenced by the abstraction. $\square$

The condition for correct lemma application in a mixed abstract EOLD resolution is exactly the same as for a concrete one (cf. def. 5 and 6). Thus, we have the following proposition, which can be considered as a "soundness" result for lemma application in the mixed abstract EOLD derivations:

Proposition 6 Given a mixed abstract definite logic program P^{mix} with a mixed abstract initial goal G_0^{mix} , a set of permissible lemmas $\mathcal{L}_P^{mix}$ and a

concrete program $P \in \gamma(P^{mix})$ with a concrete initial goal $G_0 \in \gamma(G_0^{mix})$. If there exist a mixed abstract EOLD derivation with lemma application of the form

$$G_0^{mix} \xrightarrow{s_1^{mix} \theta_1^{mix}} G_1^{mix} \dots \xrightarrow{s_n^{mix} \theta_n^{mix}} G_n^{mix}, n \geq 0$$

where s_i^{mix} denotes a clause or a lemma, then there exists a concrete derivation of the form

$$G_0 \xrightarrow{s_1 \theta_1} G_1 \dots \xrightarrow{s_m \theta_m} G_m, 0 \leq m \leq n$$

and a legal assignment ρ such that $s_i = \rho(s_i^{mix})$, $G_i = \rho(G_i^{mix})$, $local(G_i) \supseteq local(G_i^{mix})$ and $\theta_i \trianglelefteq \theta_i^{mix}$ for all $i = 0, 1, \dots, m$, where either $m = n$ or $G_{m+1} = fail$

Proof The proof can also be obtained by induction on the length of the derivations. Note that if there is a correct application of a permissible lemma in the abstract derivation, then any of its concretizations can be applied to the corresponding concretization of the current goal.

□

Notice that the proposition above *does not ensure* that in general any $EOLD_{\mathcal{L}}$ refutation can be represented by a mixed abstract one because some local variables occurring in the concrete refutation can be lost (i.e. they can be represented as abstract ones) in the abstract derivation. However, the proposition 2 ensure soundness in the following sense: if there is $EOLD_{\mathcal{L}}^{mix}$ derivation, then any corresponding concrete derivation is a derivation with correct lemma application. Moreover, if "completeness" is ensured (see def. 13 and discussion below), then proposition 1 ensures that any $EOLD^{mix}$ refutation is covered by some $EOLD_{\mathcal{L}}^{mix}$ refutation.

In order to have "completeness" of abstract mixed $EOLD$ resolution with lemma application we need the following definition:

Definition 13 *Safe set of mixed abstract permissible lemmas*

A set $\mathcal{L}_P^{mix}$ of lemmas permissible wrt a definite logic program P^{mix} is said to be safe iff for any lemma $L^{mix} \in \mathcal{L}_P^{mix}$ of the form $Q^{mix}(\bar{X}, \bar{Y}) \leftarrow R^{mix}(\bar{X}, \bar{Z})$ holds: if the goal Q^{mix} has an $EOLD$ refutation with computed answer substitution θ^{mix} , then the goal R^{mix} has an $EOLD_{\mathcal{L}}^{mix}$ refutation wrt $P^{mix} \oplus \mathcal{L}_P^{mix}$ with computed answer substitution $\theta_{\mathcal{L}}^{mix}$ such that $\theta_{mix} \upharpoonright_{\bar{X}} = \theta_{\mathcal{L}}^{mix} \upharpoonright_{\bar{X}}$. □

It is important to notice that the conditions above are much weaker than those for abstract permissibility (cf. def. 12). The permissibility of abstract lemma requires that *all* concretizations should be permissible, while the safety condition does not refer to concrete derivations. If the set $\mathcal{L}_P^{mix}$ is safe, then abstract $EOLD_{\mathcal{L}}$ resolution is complete in abstract form, i.e. proposition 2 is true for abstract $EOLD_{\mathcal{L}}$ resolution. The latter implies the following: If there exists an $EOLD$ refutation for the initial goal, then it is represented by some $EOLD^{mix}$ refutation (cf. proposition 1). The safety condition ensures existence of the corresponding $EOLD_{\mathcal{L}}^{mix}$ refutation. Thus, the proposed abstract interpretation schema is correct.

Our purpose is to ensure correctness of lemma application in abstract form combined with a tabulation mechanism (the $EOLDT_{\mathcal{L}}^{mix}$ resolution). Its correctness follows immediately from the discussion above. The $EOLDT_{\mathcal{L}}^{mix}$ resolution can be used to construct finite abstract trees which correctly represent

the concrete ones. Moreover, lemmas can be used to construct these trees. The next section gives an example of the compilation of a mixed abstract $EOLDT_C^{mix}$ tree into a program, which is equivalent to the source one wrt the set of computed answer substitutions for the variables of the initial goal and has the behaviour, which is prescribed by the abstract tree.

Example 8 Given the definite logic program P :

$$\begin{aligned} c_1 &: \text{rev}([], []). \\ c_2 &: \text{rev}([H|T], Y) \leftarrow \text{rev}(T, R), \text{app}(R, [H], Y). \\ c_3 &: \text{app}([], L, L). \\ c_4 &: \text{app}([H|T], L, [H|X]) \leftarrow \text{app}(T, L, X). \end{aligned}$$

and the mixed abstract initial goal $G_0^{mix} = \text{rev}(X_0^\alpha, Y_0)$, the following mixed abstract $EOLD$ derivation is correct

$$\begin{aligned} &\bullet G_0^{mix} = < |\text{rev}(X_0^\alpha, Y_0)| > \\ &\downarrow \\ &\bullet G_1^{mix} = < |\text{rev}(T_1^\alpha, R_1), \text{app}(R_1, [H_1^\alpha], Y_0)| > \quad c_2 : \theta_1^{mix} = \{X_0^\alpha \leftarrow [H_1^\alpha | T_1^\alpha]\} \\ &\downarrow \\ &\bullet G_2^{mix} = < |\text{rev}(T_2^\alpha, R_2), \text{app}(R_2, [H_2^\alpha], R_1), \text{app}(R_1, [H_1^\alpha], Y_0)| > \quad c_2 : \theta_2^{mix} = \{T_1^\alpha \leftarrow [H_2^\alpha | T_2^\alpha]\} \\ &\downarrow \\ &\bullet G_3^{mix} = < |\text{rev}(T_2^\alpha, R_2), \text{app}(R_2, [H_2^\alpha, H_1^\alpha], Y_0)| > \quad L^{mix} : \theta_3^{mix} = \{ \} \end{aligned}$$

where the permissible mixed abstract lemma L^{mix} wrt P has been obtained from the lemma given in example 7 by the following obvious specialisation: given the permissible lemma of the form $Q(\bar{X}, \bar{Y}) \leftarrow R(\bar{X}, \bar{Z})$, the lemma $B(\bar{X})Q(\bar{X}, \bar{Y})\sigma \leftarrow B(\bar{X})R(\bar{X}, \bar{Z})\sigma$ is also permissible lemma provided that $\text{dom}(\sigma) \subseteq \bar{X}$. In fact, the mixed abstract derivation $\mathcal{D}^{mix}$ apparently is a proof that the dead end variable R_1 cannot be disturbed by the abstracted parts of the program, and, thus, the lemma application in abstract form is correct, i.e. given a concrete derivation $\mathcal{D}$ represented by the mixed abstract derivation $\mathcal{D}^{mix}$ the application of the corresponding lemma $L \in \gamma(L^{mix})$ is correct for $\mathcal{D}$. Notice that the lemma used is a non trivial lemma. $\square$

4.3 Logic Program Transformation using Abstract EOLDT resolution

In this section we show how an $EOLDT_C^{mix}$ tree can be used to perform a transformation of the source logic program. The transformation algorithm is presented by an example. Its precise version is the straightforward extension of the algorithm in [2, 3] provided that the correctness of lemma application for $EOLDT_C^{mix}$ resolution is taken into account.

Before going into details, we need some simple definitions. Any expression equivalent (up to renaming of variables) to the first block B of the goal G is said to be the *extended call* of the goal. Given a subrefutation of the first block of the goal G having the first block B and the corresponding sequence of substitutions $\theta_1, \dots, \theta_n$, the *extended answer* for the call is any expression equivalent (up to renaming of variables) to $A = B\theta_1 \dots \theta_n$.

The simple definitions above are used to reconstruct the mixed abstract derivation given in example 8 as follows:

$$\begin{aligned}
& \bullet G_0^{mix} = \langle \underline{rev}(X_0^\alpha, Y_0) \rangle \\
& \downarrow \\
& \bullet G_1^{mix} = \langle \underline{rev}(T_1^\alpha, R_1), \underline{app}(R_1, L_1^\alpha) \rangle \quad c_2 : \theta_1^{mix} = \{X_0^\alpha \leftarrow [H_1^\alpha | T_1^\alpha]\} \\
& \downarrow \\
& \bullet G_2^{mix} = \langle \underline{rev}(T_2^\alpha, R_2), \underline{app}(R_2, [H_2^\alpha], R_1), \underline{app}(R_1, L_1^\alpha, Y_0) \rangle \quad c_2 : \theta_2^{mix} = \{T_1^\alpha \leftarrow [H_2^\alpha | T_2^\alpha]\} \\
& \downarrow \\
& \bullet G_3^{mix} = \langle \underline{rev}(T_2^\alpha, R_2), \underline{app}(R_2, L_2^\alpha, Y_0) \rangle \quad L^{mix} : \theta_3^{mix} = \{\} \\
& \downarrow \\
& \bullet G_4^{mix} = \langle \rangle \quad A^{mix} : \theta_4^{mix} = \{R_2 \leftarrow W_1^\alpha, T_2^\alpha \leftarrow W_1^\alpha, Y_0 \leftarrow W_2^\alpha\}
\end{aligned}$$

The goals G_1^{mix} and G_3^{mix} were obtained by the obvious *generalisation* of the corresponding goal in the derivation of example 8. The goal G_4^{mix} was obtained by application of *abstract tabulation* (the details can be found in [20, 11, 9, 2, 3]) using the following algorithm. Firstly notice that there exists a mixed subrefutation for the first block of the goal G_1^{mix}

$$\begin{aligned}
& \bullet G_1^{mix} = \langle \underline{rev}(T_1^\alpha, R_1), \underline{app}(R_1, L_1^\alpha, Y_0) \rangle \\
& \downarrow \\
& \bullet G_5^{mix} = \langle \underline{app}(\square, L_1^\alpha, Y_0) \rangle \quad c_1 : \theta_5^{mix} = \{T_1^\alpha \leftarrow \square, R_1 \leftarrow \square\} \\
& \downarrow \\
& \bullet G_6^{mix} = \langle \rangle \quad c_3 : \theta_6^{mix} = \{Y_0 \leftarrow L_1^\alpha\}
\end{aligned}$$

with the *generalised abstract extended answer*

$$A^{mix} = \underline{rev}(W_1^\alpha, W_2^\alpha), \underline{app}(W_2^\alpha, W_3^\alpha, W_4^\alpha)$$

Notice also that the abstract extended calls of the goals G_1^{mix} and G_3^{mix} are equivalent to $\underline{rev}(X^\alpha, D), \underline{app}(D, W^\alpha, Y)$. This allows to extend the first block of G_3^{mix} by the mixed abstract answer A^{mix} applying it as the input pseudo clause, i.e using a mixed abstract *EOLD* resolution step for the block as a whole. This does not create new answers for the abstract call of the goal G_1^{mix} due to the chosen generalisation algorithm. The construction of the mixed abstract tree can be completed by noticing that there exists one more simple refutation

$$\begin{aligned}
& \bullet G_0^{mix} = \langle \underline{rev}(X_0^\alpha, Y_0) \rangle \\
& \downarrow \\
& \bullet G_7^{mix} = \langle \rangle \quad c_1 : \theta_7^{mix} = \{X_0^\alpha \leftarrow \square, Y_0 \leftarrow \square\}
\end{aligned}$$

This example shows the importance of lemma application in mixed abstract derivation. In fact, to make the tree finite we *have* to generalise goals and apply generalised abstract answers, which can introduce abstract variables. On the other hand, generalisation should be done as precise as possible keeping in mind the conditions for lemma application. Thus the correct lemma application after several applications of abstract answers needs the proof that dead end variables are not influenced by the abstract answers. This consideration is the main extension of the algorithm presented in [2, 3]. The goal transformation by lemma application is a powerful tool to construct efficient abstract trees (notice that the abstract tree above does not use clause c_4 and corresponds to the program, which uses an accumulating parameter). Thus, it seems reasonable to apply a rather complex abstract interpretation technique to ensure the correctness of lemma application during static analysis time.

The second important information in the tree is the following: the variable R_1 occurring in the goal G_1^{mix} is a local variable. Thus, it can be dropped during the transformation of the source program (this is the second improvement of the algorithm of [2, 3]).

The tree presented above can be used to obtain the following program

$$\begin{aligned} c_1 : & \text{ rev}([], []). \\ c_2 : & \text{ rev}([H|T], Y) \leftarrow \pi_3(T, [H], Y). \\ c_3 : & \pi_3([], L, L). \\ c_4 : & \pi_3([H|T], L, Y) \leftarrow \pi_3(T, [H|L], Y). \end{aligned}$$

by the transformation algorithm given in [2, 3] with the extension that local variables are dropped when generating definitions for the new program. This program is equivalent to the source one wrt the initial goal. In general, the transformations of the source definite logic program, which can be obtained using the presented framework, preserve the set of computed answer substitutions for the variables of the initial goal.

5 Conclusion and Related Work

The presented framework is aimed at deriving complex logic program transformations based upon goal replacement and unfold/fold operations (the latter were described in details in [2, 3]). Our approach makes use of a complete flow analysis of the source definite logic program as a pretransformation phase (here we follow the basic ideas of [9]).

This approach is more powerful than that of [19, 10, 18, 12] because as it was pointed out in [13, 8] the unfold/fold transformations, which can be derived from the SLD-like tree analysis cannot always be expressed in terms of Tamaki and Sato transformations. On the other hand, in our settings recursive predicates can be derived (this is not possible using the framework of [13, 8]). Moreover, it can be shown that our unfold/fold transformations can be easily reduced by restricting abstract interpretation algorithms to obtain exactly the algorithm of [7] (this can be done by allowing only singleton blocks and only one answer for each call when applying tabulation). So the main motivation of the paper is to combine this technique with the goal replacement operations through the lemma application.

Firstly we have presented conditions for the goal replacement in the SLD-like derivations. This approach differs significantly from that of [8] (the latter should be considered as the most advanced in the scope of the goal replacement) because the latter deals with goal replacement in the clause bodies and aimed to preserve the minimal model rather than the set of computed answer substitutions for certain variables of the initial goal. On the other hand our conditions combined with the subsequent transformation algorithms seem to be as permissible as those of [8]. Unfortunately the precise comparison is rather difficult, although it is possible to show that our approach is capable to produce the same results for several interesting examples.

Secondly, in order to derive the transformations we have to construct a finite complete abstract tree, what in general cannot be achieved without abstraction or generalisation. Thus, the main problem in our approach is the correct lemma application for the partially abstracted derivations. As it could be seen from above, the conditions for the goal replacement are very strict, i.e. they require rather precise information about the current goal and its relation with the initial goal, what is not always the case in the abstracted derivations. So we have proposed a special abstract interpretation schema which allows to

construct finite complete abstract trees using abstract tabulation mechanism combined with correct lemma application. As a result, our framework allows to derive non trivial transformations of definite logic programs which could be more powerful than those of [16, 17, 6, 7] because we provide more flexible unfold/fold transformations combined with goal replacement.

Acknowledgements We are grateful to the referees for several accurate remarks and questions.

References

- [1] Bruynooghe, M., De Schreye, D., Krekels, B., Compiling Control, *J. Logic Programming*, 1989, Vol.6, Nos.2-3, 135-162.
- [2] Boulanger, D., Bruynooghe, M., Deriving Transformations of Logic Programs using Abstract Interpretation, *Logic Program Synthesis and Transformation (LOPSTR'92)*, eds. K.K. Lau and T. Clement, *Workshops in Computing*, Springer-Verlag, 1993, 99-117.
- [3] Boulanger, D., Bruynooghe, M., Deriving Fold/Unfold Transformations of Logic Programs Using Extended OLD-T-based Abstract Interpretation, *J. Symbolic Computation*, 1993, Vol.15, 495-521.
- [4] Burstall, R., Darlington, J., A Transformation System for Developing Recursive Programs, *JACM*, Jan.1977, Vol.24, No.1, 44-67.
- [5] De Schreye, D., Martens, B., Sablon, G., Bruynooghe, M., Compiling Bottom-Up and Mixed Derivations into Top-Down Executable Logic Programs, *J. Automated Reasoning*, 1991, Vol.7, 337-358.
- [6] Gallagher, J., Bruynooghe, M., Some Low Level Transformations of Logic Programs, *Proc. 2nd Workshop in Meta-Programming in Logic*, Leuven, 1990, 229-244.
- [7] Gallagher, J., Bruynooghe, M., The Derivation of an Algorithm for Program Specialisation, *New Generation Computing*, 1991, Vol.9, 305-333.
- [8] Gardner, P., Shepherdson, J., Unfold/Fold Transformations of Logic Programs, *Computational Logic, Essays in Honour of Alan Robinson*, eds. J.L.Lassez and G.Plotkin, MIT Press, 1991, 565-583.
- [9] Gallagher, J., Codish M., Shapiro E., Specialisation of Prolog and FCP Programs Using Abstract Interpretation, *New Generation Computing*, 1988, Vol.6, Nos.2-3, 159-186.
- [10] Kawamura, T., Kanamori, T., Preservation of Stronger Equivalence in Unfold/Fold Logic Program Transformation, *Proc. 4th Int. Conf. on FGCS*, Tokyo, 1988.
- [11] Kanamori, T., Kawamura, T., Abstract Interpretation Based on OLD-T Resolution, *J. Logic Programming*, Vol.15, Nos. 1-2, 1-30 1993 .
- [12] Kawamura, T., Derivation of Efficient Logic Programs by Synthesising New Predicates, *Proc. 1991 Int. Symp. on Logic Programming*, San Diego, 1991, 611-625.

- [13] Lloyd,L., Shepherdson,J., Partial Evaluation in Logic Programming, J. Logic Programming, 1991, Vol.11, Nos.3-4, 217-242.
- [14] Lloyd,L., Foundations of Logic Programming, Springer-Verlag, Berlin, 1987.
- [15] Maher,M., Equivalences of Logic Programs, Foundations of Deductive Databases and Logic Programming, Ed. J.Minker, Morgan-Kaufmann, 1988, 627-658.
- [16] Proietti,M., Pettorossi,A., Construction of Efficient Logic Programs by Loop Absorption and Generalisation, Proc. 2nd Workshop in Meta-Programming in Logic, Leuven, 1990, 57-81.
- [17] Proietti,M., Pettorossi,A., Unfolding - Definition - Folding, In this Order, For Avoiding Unnecessary Variables in Logic Programs, Proc. 3rd Int. Symp. on Programming Languages Implementation and Logic Programming, Aug. 1991, LNCS No.528, Springer-Verlag, 1991, 347-358.
- [18] Seki,H., Unfold/Fold Transformation of stratified programs, J. Theoretical Computer Science, 1991, Vol.86, 107-139.
- [19] Tamaki,H., Sato,T., Unfold/Fold Transformation of Logic Programs, Proc. 2nd Int. Conf. on Logic Programming, Uppsala, 1984, 127-138.
- [20] Tamaki,H., Sato,T., OLD Resolution with Tabulation, Proc. 3rd Int. Conf. on Logic Programming, London, July 1986, 84-98.

1994

Yves Deville (Ed.)

Logic Program Synthesis and Transformation

Proceedings of LOPSTR 93,
International Workshop on Logic
Program Synthesis and Transformation,
Louvain-la-Neuve, Belgium,
7-9 July 1993

Published in collaboration with the
British Computer Society

Springer-Verlag
London Berlin Heidelberg New York
Paris Tokyo Hong Kong
Barcelona Budapest