Transforming Normal Programs by Replacement *

Annalisa Bossi, Nicoletta Cocco, Sandro Etalle

Dipartimento di Matematica Pura ed Applicata,
Università di Padova,
Via Belzoni 7, 35131 Padova, Italy.
email: bossi,cocco,etalle@pdmat1.unipd.it
fax: ++39-49-8758596

Abstract. The replacement transformation operation, already defined in [28], is studied wrt normal programs. We give applicability conditions able to ensure the correctness of the operation wrt Fitting's and Kunen's semantics. We show how replacement can mimic other transformation operations such as thinning, fattening and folding, thus producing applicability conditions for them too. Furthermore we characterize a transformation sequence for which the preservation of Fitting's and Kunen's semantics is ensured.

1 Introduction

Program transformation is now a widely accepted technique for the systematic development of correct and efficient programs, see [6,12,15,28,13,18,8,21,22,4] to quote just a few papers on this topic. A main concern when transforming a program is the preservation of its meaning. In order to express the meaning of a program we need to choose a *semantics*. Unfortunately, as regards logic programs, on one hand there is no general agreement on which semantics is the best one, on the other hand a transformation can be correct with respect to one semantics and incorrect with respect to another one. For instance, in the program

$$\{\ p(X) \leftarrow q(X), q(X). \qquad q([a,Y]). \qquad q([Z,b]).\ \}$$

the duplicated atom $q(X)$ in the first clause is superfluous when considering the least Herbrand model semantics and then it can be safely deleted from the body of the clause. The same operation is not safe when the computed answers semantics is considered [3]: in fact the answer substitution $X = [a,b]$ would be missed in the transformed program. The first papers on logic programs transformation considered definite programs and the least Herbrand model semantics. *Normal programs* have been taken into consideration only recently [19,11,25,24,23] together with suitable applicability conditions for guaranteeing the preservation of the meaning of the program. For normal programs the problem of choosing a sensible semantics is in fact more serious, given the logical and computational problems related to the introduction of negation and the amount of semantic proposals (see [26,27] for an almost complete panorama).

In this paper we concentrate on one transformation operation for normal programs: *the replacement*. This operation has been introduced for definite programs

* This work has been partially supported by "Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo" of CNR under grant n. 89.00026.69

by Tamaki and Sato in [28] and after that it has been rather neglected by people working on program transformation apart from Sato himself [23], Maher [19] and Gardner and Shepherdson [11]. It consists in substituting part of a clause body with an equivalent conjunction of literals. It is a very general transformation able to mimic other operations, such as thinning, fattening [3] and folding, which can be seen as particular instances of replacement. We have defined *applicability conditions* able to guarantee the correct application of replacement with respect to *Fitting's and Kunen's semantics for normal programs.* We choose these semantics for several reasons. Three-valued logic is, in our opinion, very reasonable for dealing with normal programs, since it can represent the fact that some query produce a successful answer, some fail, some other produce no answer because of circularity in its derivation. Both the semantics are defined by means of a monotonous immediate consequence operator and this is exploited in our applicability conditions. Moreover, Kunen's semantics corresponds to the top-down evaluation procedure given by SLDNF-resolution when this is complete (for allowed programs and queries [26,2]), while Fitting's semantics corresponds to a bottom-up evaluation procedure, when the program is stratified [2]. Our applicability conditions for replacement are undecidable in general, but other decidable conditions can be derived for special cases. In the paper we consider two such cases when replacement mimics folding.

Structure of the paper: the next section briefly recalls the main definitions related to Fitting's and Kunen's semantics. The definitions of *dependency level of a literal wrt a clause* is also given. In section 3 the definitions of *equivalence* and *semantic delay of a conjunction of literals wrt another one* are given and these concepts are used to define the applicability conditions for replacement in normal programs. Section 4 shows how thinning and fattening can be interpreted as special cases of replacement, thus yielding, as a consequence, conditions for a safe application of these operations to normal programs. Two different definitions of folding are also considered and the corresponding applicability conditions are derived from the ones of replacement. These conditions are easily checkable either syntactically or by considering the transformation history of the program. A short concluding section follows.

2 Preliminaries

We assume that the reader is familiar with the basic concepts of logic programming; throughout the paper we use the standard terminology of [17] and [1]. We consider *normal programs*, that is finite collections of *normal rules*, $A \leftarrow L_1, \ldots, L_m$. where A is an atom and $L_1, \ldots, L_m$ are literals. Let P be a normal program, B_P denotes the Herbrand base and *ground(P)* the set of ground instances of rules of P.

2.1 Fitting's and Kunen's semantics for normal programs

We briefly recall here the definitions of Fitting's and Kunen's semantics, for more details see [10], [16] and [27]. Both semantics are based on Kleene's three-valued logic [14] where the truth values are *true*, *false* and *undefined*. The usual logical connectives have value *true* (or *false*) when they have that value in ordinary two-valued logic for all possible replacements of *undefined* by *true* or *false*, otherwise they have

the value *undefined.*
The usual Clark's completion definition, $Comp(P)$, [7] is extended to three-valued logic by replacing $\leftrightarrow$, in the completed definitions of the predicates, with $\cong_3$, Lukasiewicz's operator of "having the same truth value". This saves $Comp(P)$ from the inconsistency that it can have in two-valued logic. For example the program $P = \{p \leftarrow \neg p.\}$ has $Comp(P) = \{p \cong_3 \neg p\}$ which has a model with p *undefined.*

Definition 1 (three-valued interpretation and model). *A three-valued (or partial) interpretation* I is an ordered couple, (T, F), of disjoint sets of ground atoms. The atoms in T (resp. F) are considered to be *true* (resp. *false*) in I.
T is the positive part of I and is referred as I^+; equivalently F is denoted by I^-. Atoms which do not appear in either set are considered to be *undefined.*
A three-valued model is a three-valued interpretation which is also a model of $Comp(P)$.

Let I and J be two partial interpretations. $I \subseteq J$ iff $I^+ \subseteq J^+$ and $I^- \subseteq J^-$.
$I \models_{HU} \Psi$ indicates that the first order formula Ψ is *true* in the interpretation I, when the underlying universe is the Herbrand Universe.

Definition 2 (Fitting's operator and Fitting's semantics). Let P be a normal program and let (T, F) be a partial interpretation. *Fitting's three-valued immediate consequence operator,* Φ_P, is defined as $\Phi_P(T, F) = (T_1, F_1)$ where

$T_1 = \{A \mid$ there exists a clause $A \leftarrow L_1, \ldots, L_m.$ in $ground(P)$ and
$L_1, \ldots, L_m$ are *true* in $(T, F)\}$;
$F_1 = \{A \mid$ for all clauses $A \leftarrow L_1, \ldots, L_m.$ in $ground(P)$,
the conjunction of literals $L_1, \ldots, L_m$ is *false* in $(T, F)\}$.

Fitting's three valued model of P, $Fit(P)$, is the least fixed point of the associated operator Φ_P.

We adopt the notation:
$\Phi_P^0(T, F) = (T, F)$;
$\Phi_P^{\alpha+1}(T, F) = \Phi_P(\Phi_P^\alpha(T, F))$;
$\Phi_P^\alpha(T, F) = \cup_{\delta<\alpha}\Phi_P^\delta(T, F)$, when α is a limit ordinal.
When the argument is omitted, we assume it to be $(\emptyset, \emptyset)$: $\Phi_P^\alpha = \Phi_P^\alpha(\emptyset, \emptyset)$. It follows directly from the definition that Φ_P is a monotonic operator, hence it converges to its least fixed point, which is given by Φ_P^α, for some ordinal α; then $Fit(P) = \Phi_P^\alpha$. Φ_P being monotone but not continuous, α could be greater than ω. From definition 2 we have the following.

Remark. If a ground atom A is *true* (resp. *false*) in Φ_P^α, where α is a limit ordinal, then there exists a successor ordinal $\beta < \alpha$ such that A is *true* (resp. *false*) in Φ_P^β.

$Fit(P)$ is the minimal three-valued Herbrand model and it is equal to the intersection of all three-valued Herbrand models of $Comp(P)$. Kunen instead proposes to consider as normal programs' semantics the intersection of all (not just Herbrand ones) three-valued models of $Comp(P)$.

Definition 3 (Kunen's semantics). Let P be a normal program. Kunen's semantics of P, $Kun(P)$, is the three-valued interpretation resulting from the intersection of all the three-valued models of $Comp(P)$.

Remark.
- $Kun(P) = \Phi_P^\omega$, hence $Kun(P) \subseteq Fit(P)$.
- $Kun(P)$ coincides with the computable part of the Fitting's model. If $Kun(P) \neq Fit(P)$, then $Kun(P)$ is not a model of P.
- $Kun(P)$ is the set of all logical consequences of $Comp(P)$.

2.2 Dependency degree

Let us consider the following normal program:

$$\begin{aligned} P = \{\ & c1 : p \leftarrow \neg q, s. \\ & c2 : q \leftarrow r. \\ & c3 : r. \\ & c4 : s \leftarrow q. \quad \} \end{aligned}$$

The definitions of the atoms p, q, s and r, all depend from clause $c3$. Informally we could say that *the dependency degree of the predicate p over clause c3 is two*, as the shortest derivation path from a clause having head p to $c3$ contains two arcs: the first from $c1$ to $c2$, through the negative literal $\neg q$; the second from $c2$, to $c3$, through the atom r. Similarly, the dependency degree of q and s on $c3$ are respectively one and two and the dependency degree of r on $c3$ is zero. The next definition formalises this intuitive notion. The atom A and the clause cl are assumed to be standardized apart.

Definition 4 (dependency degree). Let P be a program, cl a clause of P and A an atom. *The dependency degree of A (and $\neg A$) on cl*, $depen_P(A, cl)$, is

0 if A unifies with the head of cl;
n+1 if A does not unify with the head of cl and n is the least integer such that there exists a clause $C \leftarrow C_1, \ldots, C_k.$ in P, whose head unifies with A via mgu, say, θ, and, for some i, $depen_P(C_i\theta, cl) = n$.

A is independent from cl when no such n exists.

The definition can be extended to conjunctions of literals. *The conjunction of literals* $(L_1 \wedge \ldots \wedge L_n)$ *is independent from cl* iff all its components are independent from cl; otherwise *the dependency degree of* $(L_1 \wedge \ldots \wedge L_n)$ *on cl* is equal to the least dependency degree of one of its elements on cl, $depen_P((L_1 \wedge \ldots \wedge L_n), cl) = inf\{depen_P(L_i, cl)$, where $1 \leq i \leq n\}$.

3 Applicability conditions for the replacement operation

The replacement operation has been introduced by Tamaki and Sato in [28] for definite programs. Syntactically it consists in substituting a conjunction, $\tilde{C}$, of literals with another one, $\tilde{D}$, in the body of a clause.

Definition 5 (replacement). Let $c : A \leftarrow \tilde{J}, \tilde{C}, \tilde{H}.$ be a clause in a normal program P and $\tilde{D}$ be a conjunction of literals. Replacing $\tilde{C}$ with $\tilde{D}$ in c consists of substituting c' for c, where $c' : A \leftarrow \tilde{J}, \tilde{D}, \tilde{H}.$ $replace(P, c, \tilde{C}, \tilde{D}) \stackrel{\text{def}}{=} P \backslash \{c\} \cup \{c'\}$.

Some *applicability conditions* are necessary in order to ensure the preservation of the semantics through the transformation. Such conditions depend on the semantics we associate to the program. In [28] definite programs are considered; the applicability condition requires that $\tilde{C}$ and $\tilde{D}$ are logically equivalent in P and that the size of the smallest proof tree for $\tilde{C}$ is greater or equal to the size of the smallest proof tree for $\tilde{D}$. Gardner and Shepherdson, in [11], give different conditions for preserving procedural (SLDNF) semantics and the declarative one. Such conditions are based on Clark's (two valued) completion of the program. Also Maher, in [19,20], studies replacement wrt Success set, Finite Failure Set, Ground Finite Failure Set and Perfect Model semantics. Sato, in [23], considers also replacement of tautologically equivalent formulas in first order programs. We consider the replacement operation for normal programs and state some applicability conditions which ensure that Fitting's and Kunen's semantics are preserved by the transformation.

We say that the replacement operation is *acceptable* only if it does not change the Herbrand base of the program. From now on, we shall consider only acceptable replacements.

3.1 Replacement wrt Fitting's semantics

We now introduce some new definitions for expressing relations between first order formulas, such as conjunctions of literals, in terms of their semantic properties. They are used for defining general applicability conditions for transformation operations, that is conditions based only on the semantics of the program to be transformed.

Definition 6 (equivalence wrt Fitting's semantics). Let E, F be first order formulas and P be a normal program. We can define an order relation based on Fitting's semantics of P in this way: E *is less defined or equal to* F *wrt* $Fit(P)$, $E \preceq_{Fit(P)} F$, iff for each ground substitution θ, if $E\theta$ is *true* (resp. *false*) in $Fit(P)$, then $F\theta$ is *true* (*false*) in $Fit(P)$ as well. F *is equivalent to* E *in* $Fit(P)$, $F \cong_{Fit(P)} E$, iff $E \preceq_{Fit(P)} F$ and $F \preceq_{Fit(P)} E$.

Note that $F \cong_{Fit(P)} E$ iff $Fit(P) \models_{HU} \forall(F \cong_3 E)$.

Consider now the following definite program.

$$
\begin{aligned}
P = \{\ & m(X) \leftarrow n(s(X)).\\
& n(0).\\
& n(s(X)) \leftarrow n(X). \quad \}
\end{aligned}
$$

The predicates m and n have exactly the same meaning, but in order to refute the goal $\leftarrow m(s(0))$. we need four resolution steps, while for refuting $\leftarrow n(s(0))$. two steps are sufficient. Each time $\leftarrow n(t)$. has a refutation (or finitely fails) with j resolution steps, $\leftarrow m(t)$. has a refutation (or fails) with k resolution steps, where $k \leq j+2$. We can formalise this intuitive idea by saying that *the semantic delay of* m *wrt* n *is 2.* By transposing this idea into Fitting's semantics, we have that each time $n(t)$ is *true* (or *false*) in Φ_P^j, $m(t)$ is *true* (resp. *false*) in Φ_P^{j+2}.

Definition 7 (delay in Fitting's semantics). Let P be a normal program, E and F be first order formulas such that F is equivalent to E in $Fit(P)$.

The delay of F wrt E in Fitting's semantics is the least ordinal β such that, for each ordinal α and each ground substitution θ:
if $E\theta$ is *true* (resp. *false*) in Φ_P^α, then $F\theta$ is *true* (resp. *false*) in $\Phi_P^{\alpha+\beta}$.

Intuitively, E is *true* in Φ_P^α iff its truth has been proved from scratch in at most α steps. The semantic delay of F wrt E shows how many steps later than E, we determine the truth value of F (at worse).

Example 1. Let P be the following program:

$$P = \{\ p(0). \qquad\qquad q(0).$$
$$\quad p(s(0)). \qquad\qquad q(s(X)) \leftarrow p(X).$$
$$\quad p(s(s(X))) \leftarrow p(X). \qquad\qquad \}$$

p and q both compute natural numbers and $p(X) \cong_{Fit(P)} q(X)$, but while $q(s^k(0))$ is *true* starting from Φ_P^{k+1}, $p(s^k(0))$ is *true* starting from $\Phi_P^{(k/2)+1}$. The delay of $p(X)$ wrt $q(X)$ in $Fit(P)$ is zero, in fact if for some ground term t and ordinal α, $q(t)$ is *true* (resp. *false*) in Φ_P^α, then $p(t)$ is also *true* (resp. *false*) in Φ_P^α. Vice versa, the delay of $q(X)$ wrt $p(X)$ in $Fit(P)$ is ω, in fact there exists no integer $m < \omega$ such that if, for some ground term t and ordinal α, $p(t)$ is *true* (resp. *false*) in Φ_P^α, then $q(t)$ is *true* (resp. *false*) in $\Phi_P^{\alpha+m}$.

When considering Fitting's semantics, our first requirement is the equivalence of $\tilde{C}$ and $\tilde{D}$ wrt $Fit(P)$. It would make no sense to replace $\tilde{C}$ with something which has a different meaning. Unfortunately this is not enough, in fact we need the equivalence to hold also after the transformation. The equivalence can be destroyed when $\tilde{D}$ depends on the modified clause. This is shown by the next example.

Example 2. Let P be the following definite program:

$$P = \{\quad p \leftarrow q.$$
$$\quad cl : q \leftarrow r.$$
$$\qquad r. \qquad\qquad \}$$

$Fit(P) = (\{p, q, r\}, \emptyset)$.
p, q and r are all *equivalent* in $Fit(P)$, but if we replace r with p in the body of cl we obtain

$$P' = \{\quad p \leftarrow q.$$
$$\quad cl' : q \leftarrow p.$$
$$\qquad r. \qquad\qquad \}$$

which is by no means equivalent to the previous program. In fact $Fit(P') = (\{r\}, \emptyset)$. We have introduced a loop and p and q are no more *true*.
Consider now the following normal program:

$$P = \{\ d : p(X) \leftarrow \neg q(X).$$
$$\quad cl : r \quad \leftarrow \ldots, \neg q(t), \ldots$$
$$\qquad\qquad \ldots \qquad\qquad \}$$

where d is the only clause defining the predicate symbol p. $p(X)$ and $\neg q(X)$ are

equivalent in $Fit(P)$. Now, if we replace $\neg q(t)$ with $p(t)$ in cl, we obtain the following program:

$$P' = \{\ d:\ p(X) \leftarrow \neg q(X).$$
$$cl : r \quad \leftarrow \ldots, p(t), \ldots$$
$$\ldots \qquad\qquad \}$$

which has the same Fitting's semantics as the previous one, that is $Fit(P) = Fit(P')$. This holds even if the definition of p is dependent from cl. The point is that "there is no room for introducing a loop". We can try to clarify further the previous statement: replacing $\neg q(t)$ by $p(t)$ in cl preserves Fitting's semantics of the initial program if

- either p *does not depend on* cl or
- *the dependency level of* p *on* cl (this is how big the loop would be) *is greater or equal to the semantic delay of* $p(X)$ *wrt* $\neg q(X)$ *in* $Fit(P)$ (this is the space where the loop would be introduced).

In our example the *delay* of $p(X)$ wrt $\neg q(X)$ in $Fit(P)$ is one:
$\Phi_P^\alpha \models_{HU} \neg q(X)\tau$ iff $\Phi_P^{\alpha+1} \models_{HU} p(X)\tau$ and
$\Phi_P^\alpha \models_{HU} q(X)\tau$ iff $\Phi_P^{\alpha+1} \models_{HU} \neg p(X)\tau$.
d is the only clause defining predicate p and $d \neq cl$, then $depen_P(p(X), cl) > 0$, thus satisfying the above conditions.

Theorem 8 (applicability conditions wrt Fitting's semantics). *Let P be a normal program, cl : $A \leftarrow \tilde{J}, \tilde{C}, \tilde{H}$. be a clause of P where $\tilde{J}$, $\tilde{C}$, $\tilde{H}$ are conjunctions of literals. Let $\tilde{D}$ be another conjunction of literals and P' be the program resulting from the replacement of $\tilde{C}$ with $\tilde{D}$ in cl, $P' = P \backslash \{cl\} \cup \{cl' : A \leftarrow \tilde{J}, \tilde{D}, \tilde{H}.\}$. Let*
X be the set of variables of $\tilde{C}$ local wrt cl and not in $\tilde{D}$,
$X = var(\tilde{C}) \backslash (var(\tilde{D}) \cup var(A) \cup var(\tilde{H}) \cup var(\tilde{J}))$;
Y be the set of variables of $\tilde{D}$ local wrt cl' and not in $\tilde{C}$,
$Y = var(\tilde{D}) \backslash (var(\tilde{C}) \cup var(A) \cup var(\tilde{H}) \cup var(\tilde{J}))$.
If $\exists X \tilde{C}$ is equivalent to $\exists Y \tilde{D}$ in $Fit(P)$ and one of the following two conditions holds:

1. *$\tilde{D}$ is independent from cl;*
2. *the dependency degree of $\tilde{D}$ on cl is greater or equal to the delay in $Fit(P)$ of $(\exists Y \tilde{D})$ wrt $(\exists X \tilde{C})$;*

then $Fit(P) = Fit(P')$.

The theorem is a direct consequence of the following two lemmas.

Lemma 9. *If $\exists Y \tilde{D} \preceq_{Fit(P)} \exists X \tilde{C}$, then $Fit(P) \supseteq Fit(P')$.*

Lemma 10. *Let $Z = (var(\tilde{C}) \cup var(\tilde{D})) \setminus (X \cup Y)$ be the set of non-local variables in $\tilde{C}$ and $\tilde{D}$. If*

(i) $\exists X \tilde{C} \preceq_{Fit(P)} \exists Y \tilde{D}$;
(ii) for each ground substitution σ having Z as domain, if $\tilde{D}\sigma$ is dependent on cl, then the dependency degree of $\tilde{D}\sigma$ on cl is greater or equal to the delay in $Fit(P)$ of $(\exists Y \tilde{D})\sigma$ wrt $(\exists X \tilde{C})\sigma$,

then $Fit(P) \subseteq Fit(P')$.

The proofs of the lemmata, in the simpler case in which $\tilde{C}$ and $\tilde{D}$ are ground literals are given in [5]. The general case is proved in [9]. The proof strictly depends on the fact that the semantics is defined by means of an immediate consequence operator.

3.2 Replacement wrt Kunen's semantics

When considering Kunen's semantics we have to give a slightly different notion of equivalence between conjunctions of literals. This is illustrated by the following example.

Example 3. Let us consider the following program:

$$P = \{ \quad p(s(X)) \leftarrow p(X). \quad cl : r(b) \leftarrow f. \quad q \leftarrow f. \quad \}$$

In $Fit(P)$, both f and each ground instance of $p(X)$ are *false*Hence, from definition 6, $p(X) \cong_{Fit(P)} f$. Since p is independent from cl, we can replace f with $p(X)$ in the body of cl and, by theorem 8, the resulting program P' has the same Fitting's semantics of the original one.

$$P' = \{ \quad p(s(X)) \leftarrow p(X). \quad cl' : r(b) \leftarrow p(X). \quad q \leftarrow f. \quad \}$$

But such replacement operation is not safe wrt Kunen's semantics: $r(b)$ is *false* in $Kun(P)$, while it is *undefined* in $Kun(P')$. In fact even if all ground instances of $p(X)$ are *false* in $Kun(P)$, just like (all ground instances of) f, $p(X)$ and f cannot be considered *equivalent* wrt Kunen's semantics.

In Kunen's semantics we consider only equivalence below ω and define the semantic delay consequently.

Definition 11 (equivalence wrt Kunen's semantics). Let E, F be first order formulas and P be a normal program. We can define an order relation based on Kunen's semantics of P in this way: E *is less defined or equal to* F *wrt* $Kun(P)$, $E \preceq_{Kun(P)} F$, iff for each integer j there exists an integer k such that
for each ground substitution θ,
if $E\theta$ is *true* (resp. *false*) in Φ_P^j, then $F\theta$ is *true* (*false*) in Φ_P^k.
F *is equivalent to* E *in in* $Kun(P)$, $F \cong_{Kun(P)} E$, iff $E \preceq_{Kun(P)} F$ and $F \preceq_{Kun(P)} E$.

Definition 12 (delay in Kunen's semantics). Let P be a normal program, E and F be first order formulas such that F is equivalent to E in $Kun(P)$.
The delay of F *wrt* E *in Kunen's semantics* is the least integer n such that, for each natural m: if $E\theta$ is *true* (resp. *false*) in Φ_P^m, then $F\theta$ is *true* (resp. *false*) in Φ_P^{m+n}.

With the above definitions theorem 8 can be transposed in a natural way for Kunen's semantics.

Theorem 13 (applicability conditions wrt Kunen's semantics). *In the hypothesis of theorem 8, if* $\exists X \tilde{C}$ *is equivalent to* $\exists Y \tilde{D}$ *in* $Kun(P)$ *and one of the following two conditions holds:*

1. $\tilde{D}$ *is independent from cl;*
2. *the dependency degree of* $\tilde{D}$ *on cl is greater or equal to the delay in* $Kun(P)$ *of* $(\exists Y \tilde{D})$ *wrt* $(\exists X \tilde{C})$*;*

then $Kun(P) = Kun(P')$.

The proof is like the one for Fitting's semantics; it is sufficient to consider only ordinals below ω.

4 Replacement vs other operations

The replacement operation is a very general one. In this section we show how some other transformation operations can be interpreted as special cases of replacement. In this way we indirectly obtain applicability conditions for these other operations on normal programs which ensure that Fitting's and Kunen's semantics are preserved. *A transformation operation is correct wrt Fitting's (or Kunen's) semantics* if it is sound and complete wrt that semantics, that is if $Fit(P) = Fit(P')$ (resp. $Kun(P) = Kun(P')$), where P is the initial program and P' is the result of the operation.

Thinning and Fattening

The *thin* operation allows one to eliminate superfluous literals from the body of a clause.

Definition 14 (thin). Let $c : A \leftarrow \tilde{K}, \tilde{L}.$ be a clause in a program P. *Thinning* c *of the literals* $\tilde{L}$ *in* P consists of substituting c' for c, where $c' : A \leftarrow \tilde{K}.$ $thin(P, c, \tilde{L}) \stackrel{\text{def}}{=} P \backslash \{c\} \cup \{c'\}$.

The applicability condition must guarantee that the literals are actually superfluous. Conditions have been given for the preservation of the least Herbrand model semantics [28,4] and the computed answers semantics for definite programs [18,3] and of the Well Founded semantics for normal programs [24].

Thinning can be seen as a particular case of replacement. From theorems 8 and 13, we get the new applicability conditions.

Theorem 15 (correctness of thinning). *Let* P' *be the result of thinning the literals* $\tilde{L}$ *in the body of* c*, and let* X *be the set of local variables of* $\tilde{L}$*. If* $\exists X\ (\tilde{K} \wedge \tilde{L})$ *is equivalent to* $\tilde{K}$ *in* Fit(P) *and one of the following two conditions holds:*

1. $\tilde{K}$ *is independent from* c*;*
2. $depen_P(\tilde{K}, c)$ *is greater or equal to the delay of* $\tilde{K}$ *wrt* $\exists X\ (\tilde{K} \wedge \tilde{L})$ *in* Fit(P)*;*

then Fit(P)=Fit(P').

The same result holds using Kunen's instead of Fitting's semantics.
The conditions given above are more restrictive than the ones given in [4,3]. In fact theorem 8 considers also the "negative" information that can be inferred from the completion of the program and distinguishes it from non-termination.
Let us consider the following program:

$P = \{ \quad r \leftarrow q.$
$\quad\quad c : p \leftarrow p, q. \}$

Being both *false*, p and q are *equivalent* in $Fit(P)$, but we cannot eliminate q from the body of c, as the delay of p wrt q is one (q is *false* in Φ_P^1, p is *false* in Φ_P^2), while $depen_P(p, cl) = 0$.

The *fatten* operation is the inverse of thinning. It consists in introducing redundant literals in the body of a clause. It is generally used in order to make possible some other transformations such as folding.

Definition 16 (fatten). Let $c : A \leftarrow \tilde{K}.$ be a clause in a program P and $\tilde{L}$ a conjunction of literals. *Fattening c with $\tilde{L}$ in P* consists of substituting c' for c, where $c' : A \leftarrow \tilde{K}, \tilde{L}$. *fatten* $(P, c, B) \stackrel{\text{def}}{=} P \backslash \{c\} \cup \{c'\}$.

The literals added to the body of the clause must be "superfluous". Conditions have been given for the preservation of the least Herbrand model semantics [28,4] and the computed answers semantics [3] of definite programs.

Theorems 8 and 13 supply the applicability conditions also for the *fatten* operation when applied to normal programs.

Theorem 17 (correctness of fattening). *Let P' be the result of fattening the body of c with $\tilde{L}$ and let X be the set of local variables of $\tilde{L}$, $X = var(\tilde{L}) \backslash var(c)$. If $\exists X \ (\tilde{K} \wedge \tilde{L})$ is equivalent to $\tilde{K}$ in* Fit(P) *and one of the following two conditions holds:*

1. $\tilde{K} \wedge \tilde{L}$ is independent from c;

2. $depen_P(\tilde{K} \wedge \tilde{L}, c)$ is greater or equal to the delay of $\exists X \ (\tilde{K} \wedge \tilde{L})$ wrt $\tilde{K}$ in Fit(P);
then Fit(P)=Fit(P').

The same result holds using Kunen's instead of Fitting's semantics.

Folding

The *fold* operation consists in substituting an atom for an equivalent conjunction of literals, in the body of a clause. This operation is generally used in all the transformation systems in order to pack back unfolded clauses and to detect implicit recursive definitions. In the literature we find different definitions for this operation. This is due to the fact that it is not generally safe even for definite programs and declarative semantics and its application must be restricted by some conditions which depend on the semantics we choose. We show here two ways of using the replacement operation in order to perform folding in normal programs. The first folding depends only on the program to be transformed, while the second one depends on a transformation sequence. Both seem to be useful in program transformations.

Definition 18 (reversible folding). Let P be a normal program, $cl : R \leftarrow \tilde{J}, \tilde{K'}, \tilde{L}.$ and $d : Q \leftarrow \tilde{K}$ be distinct clauses of P, Y be the set of local variables of $\tilde{K}$, $Y = var(\tilde{K}) \backslash var(Q)$. If there exists a substitution θ, $dom(\theta) = var(\tilde{K}) \backslash Y$, such that $\tilde{K'} = \tilde{K}\theta$ and d is the only clause of P whose head unifies with $Q\theta$;
then the result of folding $\tilde{K'}$ in cl by using d as folding clause, is the program:
$P' = P \backslash \{cl\} \cup \{cl' : R \leftarrow \tilde{J}, Q\theta, \tilde{L}.\}$.

This operation corresponds to the one considered in [19,11]. To prove its correctness wrt Fitting's and Kunen's semantics we need the following lemma. The proof is omitted since it is straightforward.

Lemma 19. *Let P be a normal program, $cl : Q \leftarrow \tilde{K}.$ be a clause of P, X be the set of local variables of Q, $X = var(Q) \backslash var(\tilde{K})$ and Y be the set of local variables of $\tilde{K}$, $Y = var(\tilde{K}) \backslash var(Q)$. If θ is a substitution such that $dom(\theta) = var(\tilde{K}) \backslash Y$ and cl is the only clause of P whose head unifies with $Q\theta$, then*
a) $\exists XQ\theta$ is equivalent to $\exists Y\tilde{K}\theta$ in Fit(P) *and* Kun(P);
b) the delay of $\exists XQ\theta$ wrt $\exists Y\tilde{K}\theta$ in Fit(P) *and* Kun(P) *is one.*

Theorem 20 (correctness of reversible folding). *The reversible fold operation is correct wrt Fitting's and Kunen's semantics.*

Proof. From lemma 19 we have that $\exists XQ\theta$ is *equivalent* to $\exists Y\tilde{K}\theta$ in $Fit(P)$ and the delay of $\exists XQ\theta$ wrt $\exists Y\tilde{K}\theta$ in $Fit(P)$ is one.
Since d is the only clause that unifies with $Q\theta$ and $d \neq cl$, $depen_P(Q\theta, cl) > 0$.
We can then *replace* $\tilde{K'} = \tilde{K}\theta$ with $Q\theta$ in the body of cl, thus obtaining P'. This operation coincides with the folding defined in 18. By theorem 8, we have that $Fit(P) = Fit(P')$.
A similar reasoning holds also for Kunen's semantics. □

Example 4. Let us consider the following program:

$$P = \{\ \begin{array}{lll} cl : p(X) & \leftarrow q(X,b), \neg s(X), r(a,X). & q(X,a). \\ d : r(Z,Y) & \leftarrow q(Y,Z), \neg s(Y). & q(X,b). \\ r(a,Y) & \leftarrow p(Y). & \end{array} \}$$

With $\theta = \{Z = b\}$, we have that $body(d)\theta$ is a variant of $(q(X,b), \neg s(X))$ and that d is the only clause of P whose head unifies with $r(Z,Y)\theta$. Hence we can fold clause cl, thus obtaining the program:

$$P = \{\ \begin{array}{lll} cl : p(X) & \leftarrow r(b,X), r(a,X). & q(X,a). \\ d : r(Z,Y) & \leftarrow q(Y,Z), \neg s(Y). & q(X,b). \\ r(a,Y) & \leftarrow p(Y). & \end{array} \}$$

We consider now a fold operation similar to the ones defined in [28,25] since it depends on the transformation history. The operation and the transformation sequence are defined in terms of each other.

Definition 21 (recursive folding). Let $P_0, \ldots, P_k$ be a transformation sequence, $cl : R \leftarrow \tilde{J}, \tilde{K'}, \tilde{L}.$ a clause of P_k, $d : Q \leftarrow \tilde{K}.$ a non-recursive, definite clause of P_j,

where $j < k$ and $\tilde{K}$ has no local variables, that is, $var(\tilde{K}) \subseteq var(Q)$.
If there exists a substitution θ, $dom(\theta) = var(\tilde{K})$, such that $\tilde{K}' = \tilde{K}\theta$ and d is the only clause of P_j whose head unifies with $Q\theta$ and if one of the following conditions holds:

1. Q is independent from cl in P_k, or
2. P_k was obtained from P_j by applying the unfolding operation to all the atoms of $\tilde{K}$;

then the result of recursive folding $\tilde{K}'$ in cl, by using d as folding clause, is the program: $P_{k+1} = P_k \backslash \{cl\} \cup \{cl' : R \leftarrow \tilde{J}, Q\theta, \tilde{L}.\}$

It is worth noting that we require the clause d to be non-recursive but indirect recursion is not excluded.

Definition 22 (transformation sequence). A transformation sequence is a sequence $P_0, \ldots, P_n$ where for each i, $0 < i \leq n$, P_i is obtained from P_{i-1} by applying either an unfolding, or a thinning, fattening, reversible folding or recursive folding.

Example 5. A simple example of recursive folding is the following:

$$\begin{array}{ll} P = \{\ d : thereiszero(L) & \leftarrow member(0, L). \\ \quad member(X, [X \mid T]). & \\ \quad member(X, [H \mid T]) & \leftarrow member(X, T). \qquad \} \end{array}$$

Predicate $thereiszero(L)$ is *true* in $Fit(P)$ when L is a list containing a zero. By unfolding the body of d we obtain the following:

$$\begin{array}{ll} P_2 = P \backslash \{d\} \cup \{\ d_1 : thereiszero([0 \mid T]). & \\ \qquad\qquad d_2 : thereiszero([H \mid T]) & \leftarrow member(0, T). \qquad \} \end{array}$$

We can now apply recursive fold to $member(0, T)$ in the body of d_2, by using d as folding clause; the result is:

$$\begin{array}{ll} P_3 = P \backslash \{d\} \cup \{\ d_1 : thereiszero([0 \mid T]). & \\ \qquad\qquad d_3 : thereiszero([H \mid T]) & \leftarrow thereiszero(T). \qquad \} \end{array}$$

Now predicate *thereiszero* is recursive and independent from other definitions.

The *unfold* operation is correct wrt Fitting's and Kunen's semantics. The proof for Fitting's model can be found in [9], while Kunen's case is an easy corollary of the same proof.

To prove the correctness of recursive folding, we need the following results.

Lemma 23. *Let P be a normal program, $d : Q \leftarrow \tilde{K}$. a definite, nonrecursive, clause of P and X the set of local variables of Q. Let θ be a substitution such that $dom(\theta) = var(\tilde{K})$, and P' the program obtained by unfolding all the atoms in $\tilde{K}$. If $\tilde{K}$ has no local variables, $var(\tilde{K}) \subseteq var(Q)$, and d is the only clause of P whose head unifies with $Q\theta$, then*

a) $\exists XQ\theta$ is equivalent to $\tilde{K}\theta$ in Fit(P') *and* Kun(P')*;*

b) the delay of $\exists XQ\theta$ wrt $\tilde{K}\theta$ in Fit(P') *and* Kun(P') *is zero.*

As regards Fitting's semantics, the proof can be found in [5]. For Kunen's semantics it can be obtained in a similar way.

Theorem 24 (correctness of recursive folding). *The recursive folding operation is correct wrt Fitting's and Kunen's semantics.*

Proof. If P_k was obtained from P_j by unfolding all the atoms in $\tilde{K}$, from lemma 23 $\exists XQ\theta$ is equivalent to $\tilde{K}\theta$ and the delay of $\exists XQ\theta$ wrt $\tilde{K}\theta$ is zero in $Fit(P_k)$ and $Kun(P_k)$. This implies that the delay of $\exists XQ\theta$ wrt $\tilde{K}\theta$ cannot be greater than $depen_P(Q\theta, cl)$ and theorems 8 and 13 are applicable. If Q is independent from cl in P_k, by theorems 8 and 13 we can as well replace $\tilde{K}' = \tilde{K}\theta$ with $Q\theta$ in cl. □

Corollary 25 (correctness of a transformation sequence). *All the programs in a transformation sequence have the same Fitting's and Kunen's semantics.*

We give now an example to show that in the definition 21, it is not possible to drop the condition that $\tilde{K}$ must not have local variables.

Example 6. Let P be the program:

$$P = \{\ cl : q(a) \leftarrow p(X).\quad p(s(X)) \leftarrow p(X).\ \}$$

$q(a)$ is equivalent to $\exists Xp(X)$ in $Fit(P)$ and the delay of $q(a)$ wrt $\exists Xp(X)$ in $Fit(P)$ is one. Since the body of cl contains local variables, the unfold operation cannot reduce the semantic delay between $q(a)$ and $\exists Xp(X)$. By unfolding $p(X)$ in cl we obtain:

$$P' = \{\ cl' : q(a) \leftarrow p(Y).\quad p(s(X)) \leftarrow p(X).\ \}$$

which is identical to P modulo renaming of variables. Thus the delay between $q(a)$ and $\exists Xp(X)$ in $Fit(P)$ has not changed. This shows that lemma 23 depends on the condition on local variables.

In the last program, since $p(Y)$ in cl' is the result of an unfold operation, we could apply the *fold* operation defined in [25], using cl as the defining clause. The result would be:

$$P'' = \{\ cl'' : q(a) \leftarrow q(a).\quad p(s(X)) \leftarrow p(X).\ \}$$

But $q(a)$ is *undefined* in $Fit(P'')$, hence $Fit(P'') \neq Fit(P)$.
This can be used as a counterexample for proving the following corollary.

Corollary 26. *The fold operation defined in [25] does not preserve Fitting's semantics.*

5 Conclusions

In this paper we study the replacement transformation operation wrt normal programs. It consists in substituting a conjunction of literals, $\tilde{C}$, in a clause body, with

an equivalent conjunction of literals, $\tilde{D}$. We propose some conditions which guarantee the preservation of Fitting's and Kunen's semantics during the transformation. The equivalence between $\tilde{C}$ and $\tilde{D}$ is obviously necessary but it is generally not sufficient. In fact we also need to preserve the equivalence after the transformation. Such equivalence can be destroyed only when $\tilde{D}$ depends on the modified clause. Hence we establish a relation between the level of dependency of $\tilde{D}$ from the clause and the difference in "semantic complexity" between $\tilde{C}$ and $\tilde{D}$. Such semantic complexity is measured by counting the number of applications of the fixed point operator which are necessary in order to determine the truth or falsity of a predicate. For Fitting's semantics this complexity can go beyond ω.

By considering replacement as a generalization of other transformation operations, such as thinning, fattening and folding, we show how replacement applicability conditions can be used also for them. A variant of the Tamaki-Sato's transformation sequence is defined which preserves Fitting's and Kunen's semantics. The applicability conditions considered for folding are rather simple since they are either syntactic or they depend on the history of the transformation.

Future work requires:

(*i*) to single out further cases where syntactic conditions are sufficient for a safe application of replacement and

(*ii*) to define applicability conditions for replacement wrt other semantics for normal programs. Actually, in [9] the Well Founded Model semantics has already been considered and similar results have been obtained. The proof however is much more complex due to the asymmetric construction of the positive and negative parts of the model.

References

1. K. Apt. Introduction to logic programming. In *Handbook of Theoretical Computer Science*, pages 493–574. Elsevier Science Publishers B.V., 1990.
2. K. R. Apt, H. A. Blair, and A. Walker. Towards a theory of declarative knowledge. In e. J. Minker, editor, *Foundation of Deductive Databases and Logic Programming*, pages 89–148. Morgan Kaufmann, 1988.
3. A. Bossi and N. Cocco. Basic transformation operations for logic programs which preserve computed answer substitutions. Technical Report 16, Dip. Matematica Pura e Applicata, Università di Padova, Italy, April 1990. to appear in Special Issue on Partial Deduction of the Journal of Logic Programming.
4. A. Bossi, N. Cocco, and S. Dulli. A method for specializing logic programs. *ACM Transactions on Programming Languages and Systems*, 12(2):253–302, April 1990.
5. A. Bossi, N. Cocco, and S. Etalle. Transforming normal program by replacement. Technical Report 18, Dip. Matematica Pura e Applicata, Università di Padova, Italy, November 1991.
6. R. Burstall and J. Darlington. A transformation system for developing recursive programs. *Journal of the ACM*, 24(1):44–67, January 1977.
7. K. L. Clark. Negation as failure rule. In H. Gallaire and G. Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, 1978.
8. Y. Deville. *Logic Programming. Systematic Program Development.* Addison-Wesley, 1990.

9. S. Etalle. Transformazione dei programmi logici con negazione, Tesi di Laurea, Dip. Matematica Pura e Applicata, Università di Padova, Padova, Italy, July 1991.
10. M. Fitting. A Kripke-Kleene semantics for logic programs. *Journal of Logic Programming*, (4), 1985.
11. P. Gardner and J. Shepherdson. Unfold/fold transformations of logic programs. In J.-L. Lassez and e. G. Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*. 1991.
12. C. Hogger. Derivation of logic programs. *Journal of the ACM*, 28(2):372–392, April 1981.
13. C. Hogger. *Introduction to Logic Programming*. Academic Press, 1984.
14. S. Kleene. *Introduction to Metamathematics*. D. van Nostrand, Princeton, New Jersey, 1952.
15. H. Komorowski. Partial evaluation as a means for inferencing data structures in an applicative language: A theory and implementation in the case of Prolog. In *Ninth ACM Symposium on Principles of Programming Languages, Albuquerque, New Mexico*, pages 255–267, 1982.
16. K. Kunen. Negation in logic programming. *Journal of Logic Programming*, (4):289–308, 1985.
17. J. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 1987.
18. J. Lloyd and J. Shepherdson. Partial evaluation in logic programming. Technical Report CS-87-09, Department of Computer Science, University of Bristol, England, 1987. to appear in Journal of Logic Programming.
19. M. Maher. Correctness of a logic program transformation system. IBM Research Report RC13496, T.J. Watson Research Center, 1987.
20. M. Maher. A transformation system for deductive databases with perfect model semantics. *Theoretical Computer Science*, to appear.
21. M. Proietti and A. Pettorossi. The synthesis of eureka predicates for developing logic programs. In N. Jones, editor, *ESOP'90, (Lecture Notes in Computer Science, Vol. 432)*, pages 306–325. Springer-Verlag, 1990.
22. M. Proietti and A. Pettorossi. Unfolding, definition, folding, in this order for avoiding unnecessary variables in logic programs. In Maluszynski and M. Wirsing, editors, *PLILP 91, Passau, Germany (Lecture Notes in Computer Science, Vol.528)*, pages 347–358. Springer-Verlag, 1991.
23. T. Sato. An equivalence preserving first order unfold/fold transformation system. In *Second Int. Conference on Algebraic and Logic Programming, Nancy, France, October 1990, (Lecture Notes in Computer Science, Vol. 463)*, pages 175–188. Springer-Verlag, 1990.
24. H. Seki. A comparative study of the well-founded and stable model semantics: Transformation's viewpoint. In D. P. W. Marek, A. Nerode and V. Subrahmanian, editors, *Workshop on Logic Programming and Non-Monotonic Logic, Austin, Texas, October 1990*, pages 115–123, 1990.
25. H. Seki. Unfold/fold transformation of stratified programs. *Journal of Theoretical Computer Science*, 86:107–139, 1991.
26. J. C. Shepherdson. Negation as failure: a comparision of Clark's completed data base and Reiter's closed world assumption. *Journal of Logic Programming*, (1):1–48, 1984.
27. J. C. Shepherdson. Negation in logic programming. In e. J. Minker, editor, *Foundation of Deductive Databases and Logic Programming*, pages 19–88. Morgan Kaufmann, 1988.
28. H. Tamaki and T. Sato. Unfold/fold transformation ol logic programs. In S. Tarnlund, editor, *2nd International Logic Programming Conference, Uppsala, Sweden, July 1984*, pages 127–138, 1984.