Centrum voor Wiskunde en Informatica

REPORTRAPPORT

More on unfold/fold transformations of normal programs: preservation of fitting's semantics

A. Bossi, S. Etalle

Computer Science/Department of Software Technology

Report CS-R9447 August 1994

CWI is the National Research Institute for Mathematics and Computer Science. CWI is part of the Stichting Mathematisch Centrum (SMC), the Dutch foundation for promotion of mathematics and computer science and their applications.
SMC is sponsored by the Netherlands Organization for Scientific Research (NWO). CWI is a member of ERCIM, the European Research Consortium for Informatics and Mathematics.

Copyright © Stichting Mathematisch Centrum
P.O. Box 94079, 1090 GB Amsterdam (NL)
Kruislaan 413, 1098 SJ Amsterdam (NL)
Telephone +31 20 592 9333
Telefax +31 20 592 4199

More on Unfold/Fold Transformations of Normal Programs: Preservation of Fitting's Semantics

Annalisa Bossi[1], Sandro Etalle[1,2]

[1] *Dipartimento di Matematica Pura ed Applicata, Università di Padova, Via Belzoni 7, 35131 Padova, Italy*

[2] *CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands*

email: bossi@zenone.math.unipd.it, etalle@cwi.nl

Abstract

The unfold/fold transformation system defined by Tamaki and Sato was meant for definite programs. It transforms a program into an equivalent one in the sense of both the least Herbrand model semantics and the Computed Answer Substitution semantics. Seki extended the method to normal programs and specialized it in order to preserve also the finite failure set. The resulting system is correct wrt nearly all the declarative semantics for normal programs. An exception is Fitting's model semantics. In this paper we consider a slight variation of Seki's method and we study its correctness wrt Fitting's semantics. We define an applicability condition for the fold operation and we show that it ensures the preservation of the considered semantics through the transformation.

AMS Subject Classification (1991): 68N17, 68Q55, 68T15.
CR Subject Classification (1991): D.1.6, F.3.2, F.4.1, I.2.2, I.2.3.
Keywords and Phrases: Program's Transformation, Logic Programming, Semantics, Negation, Folding, Replacement.

Note: This work has been partially supported by "Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo" of CNR under grant n. 89.00026.69. This paper will appear in: Proc. of META94 "Fifth Workshop on Metaprogramming in Logic", F. Turini (editor).

1. Introduction

The unfold/fold transformation rules were introduced by Burstall and Darlington [BD77] for transforming clear, simple functional programs into equivalent, more efficient ones. The rules were early adapted to the field of logic programs both for program synthesis [CS77, Hog81] and for program specialization and optimization [AAP78, Kom82]. Soon later, Tamaki and Sato [TS84] proposed an elegant framework for the transformation of logic programs based on unfold/fold rules.

The major requirement of a transformation system is its correctness: it should transform a program into an equivalent one. Tamaki and Sato's system was originally designed for definite programs and in this context a natural equivalence on programs is the one induced by the least Herbrand model semantics. In [TS84] it was shown that the system preserves such a semantics. Afterward, the system was proven to be correct wrt many other semantics: the computed answer substitution semantics [KK90], the Perfect model semantics [Sek91], the Well-Founded semantics [Sek93] and the Stable model semantics [Sek90, AD93].

In [Sek91], Seki modified the method by restricting its applicability conditions. The system so defined enjoys all the semantic properties of Tamaki-Sato's, moreover, it preserves the finite failure set of the original program [Sek89] and it is correct wrt Kunen's semantics [Sat92].

Report CS-R9447
ISSN 0169-118X
CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

However, neither Tamaki-Sato's, nor Seki's system preserve the Fitting model semantics.

In this paper we consider a transformation schema which is similar yet slightly more restrictive to the one introduced by Seki [Sek91] for normal programs. We study the effect of the transformation on the Fitting's semantics [Fit85] and we individuate a sufficient condition for its preservation.

The difference between the method we propose and the one of Seki consists in the fact that here the operations have to be performed in a precise order. We believe that this order corresponds to the "natural" order in which the operations are usually carried out within a transformation sequence, and therefore that the restriction we impose is actually rather mild.

The structure of the paper is the following. In Section 2 we recall the definition of Fitting's operator. In Section 3 the transformation schema is defined and exemplified, and the applicability conditions for the fold operation are presented and discussed. Finally, in Section 4, we prove the correctness of the unfold/fold transformation wrt Fitting's semantics.

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 [Llo87] and [Apt90]. 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. B_P denotes the Herbrand base and *Ground(P)* the set of ground instances of clauses of a program P. We say that a clause is *definite* if the body contains only positive literals (atoms); a definite program is then a program consisting only of definite clauses. Symbols with a $\sim$ on top denote tuples of objects, for instance $\tilde{x}$ denotes a tuple of variables $x_1, \ldots, x_n$, and $\tilde{x} = \tilde{y}$ stands for $x_1 = y_1 \wedge \ldots \wedge x_n = y_n$. We also adopt the usual logic programming notation that uses "," instead of $\wedge$, hence a conjunction of literals $L_1 \wedge \ldots \wedge L_n$ will be denoted by $L_1, \ldots, L_n$ or by $\tilde{L}$.

Three valued semantics for normal programs. In this paper we refer to the usual Clark's completion definition, $Comp(P)$, [Cla78] which consists of the completed definition of each predicate together with CET, Clark's Equality Theory, which is needed in order to interpret "=" correctly. It is well-known that, when considering normal programs, the two valued completion $Comp(P)$ of a program P might be inconsistent an consequently have no model; moreover, when $Comp(P)$ is consistent, it usually has more models, none of which can be considered the *least* (hence the preferred) one. Following [Fit85], we avoid this problem by switching to a three-valued logic, where the truth tables of the connective are the ones given by Kleene [Kle52]. When working with 3-valued logic, the same definition of completion applies, with the only difference that the connective $\leftrightarrow$ is replaced with $\Leftrightarrow$, Lucasiewicz's operator of "having the same truth value". In this context, we have that a *three valued (or partial) interpretation*, is a mapping from the ground atoms of the language $\mathcal{L}$ into the set $\{$*true, false, undefined*$\}$.

Definition 2.1 Let $\mathcal{L}$ be a language. *A three valued (or partial) $\mathcal{L}$-interpretation, I,* is a mapping from the ground atoms of $\mathcal{L}$ into the set $\{$*true, false, undefined*$\}$. □

A partial interpretation I is represented by 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 denoted by I^+; equivalently F is denoted by I^-. Atoms which do not appear in either set are considered to be *undefined*.

If I and J are two partial $\mathcal{L}$-interpretations, then $I \cap J$ is the three valued $\mathcal{L}$-interpretation given by $(I^+ \cap J^+, I^- \cap J^-)$, $I \cup J$ is the three valued $\mathcal{L}$-interpretation given by $(I^+ \cup J^+, I^- \cup J^-)$ and we say that $I \subseteq J$ iff $I = I \cap J$, that is, iff $I^+ \subseteq J^+$ and $I^- \subseteq J^-$. The set of all $\mathcal{L}$-interpretations is then a complete lattice. In the sequel we refer to a fixed but unspecified language $\mathcal{L}$ that we assume contains all the functions symbols and the predicate symbols of the programs that we consider, consequently we will omit the $\mathcal{L}$ prefix and speak of "interpretations" rather than of "$\mathcal{L}$-interpretations".

We now give a definition of Fitting's operator [Fit85]. We denote by $Var(E)$ the set of all the variables in an expression E and we write $\exists y\, B\theta$ as a shorthand for $(\exists y\, B)\theta$, that is, unless explicitly stated, the quantification applies always before the substitution.

Definition 2.2 Let P be a normal program, and I a three valued interpretation. $\Phi_P(I)$ is the three valued interpretation defined as follows:

- A ground atom A is *true* in $\Phi_P(I)$
  iff there exists a clause $c: B \leftarrow \tilde{L}$. in P whose head unifies with A, $\theta = mgu(A, B)$, and $\exists \tilde{w}\, \tilde{\theta}$ is *true* in I, where $\tilde{w}$ is the set of local variables of c, $\tilde{w} = Var(\tilde{L}) \setminus Var(B)$.

- A ground atom A is *false* in $\Phi_P(I)$
  iff for all clauses $c: B \leftarrow \tilde{L}$ in P for which there exists $\theta = mgu(A, B)$ we have that $\exists \tilde{w}\, \tilde{L}\theta$ is *false* in I, where $\tilde{w}$ is the set of local variables of c, $\tilde{w} = Var(\tilde{L}) \setminus Var(B)$. □

Recall that a *Herbrand* model is a model whose universe is given by the set of $\mathcal{L}$-terms.

Φ_P is a monotonic operator, that is $I \subseteq J$ implies $\Phi_P(I) \subseteq \Phi_P(J)$, and characterizes the three valued semantics of $Comp(P)$, in fact Fitting, in [Fit85] shows that the three-valued Herbrand models of $Comp(P)$ are exactly the fixpoints of Φ_P; it follows that any program has a *least* (wrt. $\subseteq$) three-valued Herbrand model, which coincides with the least fixpoint of Φ_P. This model is usually referred to as Fitting's model.

Definition 2.3 Let P be a program, *Fitting's model* of P, $Fit(P)$, is the least three valued Herbrand model of $Comp(P)$. □

We adopt the standard notation: $\Phi_P^{\uparrow 0}$ is the interpretation that maps every ground atom into the value *undefined*, $\Phi_P^{\uparrow \alpha+1} = \Phi_P(\Phi_P^{\uparrow \alpha})$, $\Phi_P^{\uparrow \alpha} = \cup_{\delta<\alpha} \Phi_P^{\uparrow \delta}$, when α is a limit ordinal. From the monotonicity of Φ_P follows that its Kleene's sequence is monotonically increasing and it converges to its least fixpoint. Hence there always exists an ordinal α such that $lfp(\Phi_P) = \Phi_P^{\uparrow \alpha}$. Since Φ_P is monotone but not continuous, α could be greater than ω.

Theorem 2.4 **[Fit85]** Let P be a program, then, for some ordinal α,

- $Fit(P) = \Phi_P^{\uparrow \alpha}$ □

3. Unfold/fold transformations

3.1 Introduction

Unfold and fold are basic transformation rules but their definition may differ depending on the considered semantics.

Unfolding is the fundamental operation for partial evaluation [LS91] and consists in applying a resolution step to the considered atom in all possible ways. Usually, it is applied only to positive literals (an exception is [AD92]).

Folding is the inverse of unfolding when one single unfolding is possible. Syntactically, it consists in substituting a literal L for an equivalent conjunction of literals $\tilde{K}$ in the body of a clause c. This operation is used to simplify unfolded clauses and to detect implicit recursive definitions. In order to preserve the declarative semantics of logic programs, its application must be restricted by some, semantic dependent, conditions. Therefore, the various proposals mainly differ in the choice of such conditions. They can be either a constraint on how to sequentialize the operations while transforming the program [TS84, Sek91], or they can be expressed only in terms of (semantic) properties of the program, independently from its transformation history [BC93, Mah87]. For normal programs different definitions for folding in a particular transformation sequence are given in [Sek91, Sek90, GS91].

3.2 A four step transformation schema

In this section we introduce the unfold/fold transformation *schema.* All definitions are given modulo reordering of the bodies of the clauses and standardization apart is always assumed.

First we define the unfolding operation, which is basic to all the transformation systems.

Definition 3.1 (Unfold) Let $cl : A \leftarrow \tilde{L}, H.$ be a clause of a normal program P, where H is an atom. Let $\{H_1 \leftarrow \tilde{B}_1, \ldots, H_n \leftarrow \tilde{B}_n\}$ be the set of clauses of P whose heads unify with H, by mgu's $\{\theta_1, \ldots, \theta_n\}$.

- *unfolding an atom* H *in* cl consists of substituting cl with $\{cl'_1, \ldots, cl'_n\}$, where, for each i, $cl'_i = (A \leftarrow \tilde{L}, \tilde{B}_i)\theta_i$.

$unfold\,(P, cl, H) \overset{\text{def}}{=} P \backslash \{cl\} \cup \{cl'_1, \ldots, cl'_n\}$. □

Let P be a normal program. A *four step transformation schema* starting in the program P consists of the following steps:

Step 1. Introduction of new definitions. We add to the program P the set of clauses $D_{\text{def}} = \{c_i : H_i \leftarrow \tilde{B}_i\}$, where the predicate symbol of each H_i is *new,* that is, it does not occur in P. On the other hand, we require that the predicate symbols found in each $\tilde{B}_i$ are defined in P, and therefore are not *new.* The result of this operation is then

- $P_1 = P \cup D_{\text{def}}$ □

Example 3.2 (min-max, part 1) Let P be the following program

$$\begin{array}{llcl}
P = \{ & min([X], X). & & \\
 & min([X|Xs], Y) & \leftarrow & min(Xs, Z), inf(X, Z, Y). \\
 & max([X], X). & & \\
 & max([X|Xs], Y) & \leftarrow & max(Xs, Z), sup(X, Z, Y). \\
 & inf(X, Y, X) & \leftarrow & X \leq Y. \\
 & inf(X, Y, Y) & \leftarrow & \neg(X \leq Y). \\
 & sup(X, Y, Y) & \leftarrow & X \leq Y. \\
 & sup(X, Y, X) & \leftarrow & \neg(X \leq Y). \\
c_1 : & med(Xs, Med) & \leftarrow & min(Xs, Min), \\
 & & & max(Xs, Max), \\
 & & & Med\ is\ (Min + Max)/2. \quad \}
\end{array}$$

here $med(Xs, Med)$ reports in Med the average between the minimum and the maximum of the values in the list Xs.

We may notice that the definition of $med(Xs, Med)$ traverses the list Xs twice. This is obviously a source of inefficiency. In order to fix this problem via an unfold/fold transformation, we first have to introduce a new predicate *minmax.* Let us then add to program P the following new definition:

$D_{\text{def}} = \{c_2 : minmax(Xs, Min, Max) \leftarrow min(Xs, Min), max(Xs, Max).\ \}$ □

Step 2. Unfolding in D_{def}. We transform D_{def} into D_{unf} by unfolding some of its clauses. The clauses of P are therefore used as unfolding clauses. This process can be iterated several times and usually ends when all the clauses that we want to fold have been obtained; the result of this operation is

- $P_2 = P \cup D_{\text{unf}}$

□

Example 3.2 (min-max, part 2) We can now unfold the atom $min(Xs, Min)$ in the body of c_2, the result is

$$\begin{array}{lll}
c_3: & minmax([X], X, Max) & \leftarrow & max([X], Max). \\
c_4: & minmax([X|Xs], Min, Max) & \leftarrow & min(Xs, Y), \\
 & & & inf(X, Y, Min), \\
 & & & max([X|Xs], Max).
\end{array}$$

In the bodies of both clauses we can then unfold predicate max. Each clause generates two clauses.

$$\begin{array}{llll}
c_5: & minmax([X], X, X). & & \\
c_6: & minmax([X], X, Max) & \leftarrow & max([\,], Z), sup(Z, X, Max). \\
c_7: & minmax([X], Min, X) & \leftarrow & min([\,], Y), inf(X, Y, Min). \\
c_8: & minmax([X|Xs], Min, Max) & \leftarrow & min(Xs, Y), \\
 & & & inf(X, Y, Min), \\
 & & & max(Xs, Z), \\
 & & & sup(X, Z, Max).
\end{array}$$

Clauses c_6 and c_7 can then be eliminated by unfolding respectively the atoms $max([\,], Z)$ and $min([\,], Y)$. D_{unf} consists then of the following clauses.

$$\begin{array}{llll}
c_5: & minmax([X], X, X). & & \\
c_8: & minmax([X|Xs], Min, Max) & \leftarrow & min(Xs, Y), \\
 & & & inf(X, Y, Min), \\
 & & & max(Xs, Z), \\
 & & & sup(X, Z, Max).
\end{array}$$

Still, $minmax$ traverses the list Xs twice; but now we can apply a *recursive folding* operation. □

Step 3. Recursive folding. Let $c_i : H_i \leftarrow \tilde{B}_i$ be one of the clauses of D_{def}, which was introduced in *Step 1*, and $cl : A \leftarrow \tilde{B}', \tilde{S}$. be (a renaming of) a clause in D_{unf}. If there exists a substitution θ, $Dom(\theta) = Var(c_i)$ such that

(a) $\tilde{B}' = \tilde{B}_i\theta$;

(b) θ does not bind the local variables of c_i, that is for any $x, y \in Var(\tilde{B}_i) \backslash Var(\tilde{H}_i)$ the following three conditions hold

- $x\theta$ is a variable;
- $x\theta$ does not appear in A, $\tilde{S}$, $H_i\theta$;
- if $x \neq y$ then $x\theta \neq y\theta$;

(c) c_i is the only clause of D_{def} whose head unifies with $H_i\theta$;

(d) all the literals of $\tilde{B}'$ are the result of a previous unfolding.

then we can fold $H_i\theta$ in cl, obtaining $cl' : A \leftarrow H_i\theta, \tilde{S}$. This operation can be performed on several conjunctions simultaneously, even on the same clause. The result is that D_{unf} is transformed into D_{fold} and hence

- $P_3 = P \cup D_{\text{fold}}$ □

Example 3.2 (min-max, part 3) We can now fold $min(Xs, Y), max(Xs, Z)$ in the body of c_8. The resulting program D_{fold} consists of the following clauses

$$
\begin{array}{lllll}
c_5: & minmax([X], X, X). & & \\
c_9: & minmax([X|Xs], Min, Max) & \leftarrow & minmax(Xs, Y, Z), \\
 & & & inf(X, Y, Min), \\
 & & & sup(X, Z, Max).
\end{array}
$$

$minmax(Xs, Min, Max)$ has now a recursive definition and needs to traverse the list Xs only once. In order to let the definition of *med* enjoy of this improvement, we need to *propagate* predicate *minmax* inside its body. □

Step 4. Propagation folding. Technically, the difference between this step and the previous one is that now the folded clause comes form the original program P. This allows us to drop condition (d) of the folding operation.

Let $c_i : H_i \leftarrow \tilde{B}_i$ be one of the clauses of D_{def}, which was introduced in *Step 1*, and $cl : A \leftarrow \tilde{B}', \tilde{S}$. be (a renaming of) a clause in the original program P. If there exists a substitution θ, $Dom(\theta) = Var(c_i)$ such that

(a) $\tilde{B}' = \tilde{B}_i\theta$;

(b) θ does not bind the local variables of c_i, that is for any $x, y \in Var(\tilde{B}_i) \backslash Var(\tilde{H}_i)$ the following three conditions hold

- $x\theta$ is a variable;
- $x\theta$ does not appear in A, $\tilde{S}$, $H_i\theta$;
- if $x \neq y$ then $x\theta \neq y\theta$;

(c) c_i is the only clause of D_{def} whose head unifies with $H_i\theta$;

then we can fold $H_i\theta$ in cl, obtaining $cl' : A \leftarrow H_i\theta, \tilde{S}$. Also this operation can be performed on several conjunctions simultaneously, even on the same clause. The result is that P is transformed into P_{fold} and therefore

- $P_4 = P_{\text{fold}} \cup D_{\text{fold}}$ □

Example 3.2 (min-max, part 4) We can now fold $min(Xs, Y), max(Xs, Z)$ in the body of c_1, in the original program P. The resulting program is

$$
\begin{array}{llll}
P_{\text{fold}} = P \backslash \{c_1\} \cup \{c_{10} : med(Xs) & \leftarrow & minmax(Xs, Min, Max), & \\
 & & Med\ is\ (Min + Max)/2. & \}
\end{array}
$$

And then the final program is $P_4 = P_{\text{fold}} \cup D_{\text{fold}} =$

$$
\begin{array}{lllll}
= \{ & c_5: & minmax([X], X, X). & & \\
 & c_9: & minmax([X|Xs], Min, Max) & \leftarrow & minmax(Xs, Y, Z), \\
 & & & & inf(X, Y, Min), \\
 & & & & sup(X, Z, Max). \\
 & c_{10}: & med(Xs) & \leftarrow & minmax(Xs, Min, Max), \\
 & & & & Med\ is\ (Min + Max)/2.
\end{array}
$$

+ definitions for predicates min, max, inf and sup.}

Notice also that predicates *min* and *max* are no longer used by the program. □

3.3 Semantic considerations

The *schema* (that is, the method we propose) is similar but more restrictive than the *transformation sequence* with *modified* folding[1] proposed by Seki [Sek91]. The (only) limitation consists in the fact that the schema requires the operations to be performed in fixed order: for instance it does not allow a *propagation folding* to take place before a *recursive folding.* We believe that in practice this is not a bothering restriction, as it corresponds to the "natural" procedure that is followed in the process of transforming a program. In fact, in all the papers we cite, all the examples that can be reduced to a transformation sequence as in [Sek91], can also be reduced to the given transformation schema.

Since the *schema* can be seen as a particular case of the transformation *sequence*, it enjoys all its properties, among them, it preserves the following semantics of the initial program: the success set [TS84], the computed answer substitution set [KK90], the finite failure set [Sek91], the Perfect model semantics for stratified programs [Sek91], the Well-Founded semantics [Sek93], the Stable model semantics [Sek90, AD93].

However, as it is, the schema suffers of the same problems of the sequence, i.e., Fitting's Models is not preserved. This is shown by the following example.

Example 3.3 Let $P_1 = P \cup D_{\text{def}}$, where P and D_{def} are the following programs

$$\begin{array}{lllll} D_{\text{def}} & =\{ & p & \leftarrow & q(X). \quad \} \\ P & =\{ & q(s(X)) & \leftarrow & q(X), t(0). \\ & & t(0). & & \quad \} \end{array}$$

As we fix a language $\mathcal{L}$ that contains the constant 0 and the function $s/1$, we have that $\exists X\, q(X)$ is *false* in $Fit(P_1)$, consequently, p is also *false* in $Fit(P_1)$. Now let us unfold $q(X)$ in the body of the clause in D_{def}; the resulting program is the following. $P_2 = P \cup D_{\text{unf}}$, where

$$\begin{array}{lllll} D_{\text{unf}} & =\{ & p & \leftarrow & q(Y), t(0). \quad \} \\ P & =\{ & q(s(X)) & \leftarrow & q(X), t(0). \\ & & t(0). & & \quad \} \end{array}$$

We can now fold $q(Y)$ in the body of the clause of D_{unf}, the resulting program is $P_3 = P \cup D_{\text{fold}}$, where

$$\begin{array}{lllll} D_{\text{fold}} & =\{ & p & \leftarrow & p, t(0). \quad \} \\ P & =\{ & q(s(X)) & \leftarrow & q(X), t(0). \\ & & t(0). & & \quad \} \end{array}$$

Now we have that p is *undefined* in the Fitting model of P_3. □

So, in order for the transformation to preserve Fitting's model of the original program, we need some further applicability conditions. Therefore the following.

Theorem 3.4 (Correctness) Let $P_1, \ldots, P_4$ be a sequence of programs obtained applying the transformation *schema* to program P. Let also $D_{\text{def}} = \{H_i \leftarrow \tilde{B}_i\}$ be the set of clauses introduced in *Step 1*, and, for each i, $\tilde{w}_i$ be the set of local variables of c_i: $\tilde{w}_i = Var(\tilde{B}_i) \backslash Var(H_i)$. If each c_i in D_{def} satisfies the following condition:

A each time that $\exists \tilde{w}_i\, \tilde{B}_i\theta$ is *false* in some $\Phi_{P_1}^{\uparrow\beta}$, then there exists a non-limit ordinal $\alpha \leq \beta$ such that $\exists \tilde{w}_i\, \tilde{B}_i\theta$ is *false* in $\Phi_{P_1}^{\uparrow\alpha}$

Then $Fit(P_1) = Fit(P_2) = Fit(P_3) = Fit(P_4)$.

Proof. The proof is given in the subsequent Section 4. □

[1] here we are adopting Seki's notation, and we call *modified* folding the one presented in [Sek89, Sek91], which preserves the finite failure set, as opposed to the one introduced by Tamaki and Sato in [TS84], which does not.

On condition A. Condition **A** is in general undecidable, it is therefore important to provide some other decidable sufficient conditions. For this, in the rest of this Section, we adopt the following notation:
- $D_{\text{def}} = \{c_i : H_i \leftarrow \tilde{B}_i\}$ is the set of clauses introduced in *Step 1*,
and, for each i,
- $\tilde{w}_i = Var(\tilde{B}_i) \backslash Var(H_i)$ is the set of local variables of c_i.

First, it is easy to check that if c_i has no local variables, then it satisfies **A**.

Proposition 3.5 If $\tilde{w}_i = \emptyset$ then c_i satisfies **A**.

Proof. It follows at once from the definition of Fitting's operator. □

This condition, though simple, is met by most of the examples found in the literature; if we are allowed an informal "statistics", of all the papers cited in our bibliography, seven contain practical examples in clausal form which can be assimilated to our method ([BCE92, KK90, PP91, Sek89, Sek91, Sek93, TS84]), and of them, only two contain examples where the "introduced" clause contains local variables ([KK90, PP91]). Our Example 3.2 satisfies the condition as well.

Nevertheless Proposition 3.5 can easily be improved. First let us consider the following Example[2].

Example 3.6 Let $P_1 = P \cup D_{\text{def}}$, where P and D_{def} are the following programs

$$\begin{array}{llll} D_{\text{def}} & = \{\ c_0: & br(X,Y) & \leftarrow\ reach(X,Z), reach(Y,Z).\ \ \} \\ P & = \{ & reach(X,Y) & \leftarrow\ arc(X,Y). \\ & & reach(X,Y) & \leftarrow\ arc(X,Z), reach(Z,Y).\ \ \ \} \cup DB \end{array}$$

Where DB is any set of ground unit clauses defining predicate *arc*. $reach(X,Y)$ holds iff there exists a path starting from node X and ending in node Y, while $br(X,Y)$ holds iff there exists a node Z which is reachable both from node X and node Y. □

In this Example the definition of predicate *br* can be specialized and made recursive via an unfold/fold transformation. Despite the fact that clause c_0 contains the local variable Z, it is easy to see that **A** is satisfied. This is due to the fact that P is actually a DATALOG (function-free) program.

We now show that if (a part of) the original program P is function-free (or recursion-free) then **A** is always satisfied.

Let us first introduce the following notation. Let p, q be predicates, we say that p *refers to* q in program P if there is a clause of P with p in its head and q in its body. The *depends on* relation is the reflexive and transitive closure of *refers to*. Let $\tilde{L}$ be a conjunction of literals, by $P|_{\tilde{L}}$ we denote the set of clauses of P that define the predicates which the predicates in $\tilde{L}$ depend on. We say that a program is *recursion-free* if there is no chain $p_1, \ldots, p_k$ of predicate symbols such that p_i refers to p_{i+1} and $p_k = p_1$. With an abuse of notation, we also call a program *function-free* if the only terms occurring in it are either ground or variables.

We can now state the following.

Proposition 3.7 For each index i, and each $w \in \tilde{w}_i$, let us denote by $\tilde{L}_w$ the subset of $\tilde{B}_i$ formed by those literals where w occurs. If for every $\tilde{L}_w$, one of the following two conditions holds:

(a) $P_1|_{\tilde{L}_w}$ is recursion-free, or

(b) $P_1|_{\tilde{L}_w}$ is function-free;

then each c_i satisfies **A**.

[2]The example is actually a modification of Example 2.1.1 in [Sek89]

Proof. First we need the following Observation.

Observation 3.8 Let Q be a function-free or a recursion-free program, then for some integer k, $Fit(Q) = \Phi_Q^{\uparrow k}$

Proof. Straightforward □

Now fix an index i, and let $\tilde{w}_i = w_1, \dots, w_m$, and let $\tilde{M}$ be the subset of $\tilde{B}_i$ consisting of those literals that do not contain any of the variables in $\tilde{w}_i$. It is immediate that, for any ordinal α, and for any substitution θ

$$\Phi_{P_1}^{\uparrow\alpha} \models \exists\tilde{w}_i\,\tilde{B}_i\theta \ \text{ iff } \ \Phi_{P_1}^{\uparrow\alpha} \models \exists w_1\,\tilde{L}_{w_1}\theta \wedge \ldots \wedge \exists w_m\,\tilde{L}_{w_m}\theta \wedge \tilde{M}\theta \tag{3.1}$$

Now suppose that, for some ordinal α, and substitution θ, $\exists\tilde{w}_i\,\tilde{B}_i\theta$ is *false* in $\Phi_{P_1}^{\uparrow\alpha}$.
By (3.1), either *(i)* $\tilde{M}\theta$ is *false* in $\Phi_{P_1}^{\uparrow\alpha}$, or *(ii)* there exists an i such that $\exists w_i\,\tilde{L}_{w_i}\theta$ is *false* in $\Phi_{P_1}^{\uparrow\alpha}$; we treat the two cases separately.
(i), $\tilde{M}\theta$ is *false* in $\Phi_{P_1}^{\uparrow\alpha}$, then, by the definition of Φ_{P_1}, there exists a non-limit ordinal $\beta \leq \alpha$ such that $\tilde{M}\theta$ is *false* in $\Phi_{P_1}^{\uparrow\beta}$, and, by (3.1), $\exists\tilde{w}_i\,\tilde{B}_i\theta$ is *false* in $\Phi_{P_1}^{\uparrow\beta}$.
(ii), $\exists w_i\,\tilde{L}_{w_i}\theta$ is *false* in $\Phi_{P_1}^{\uparrow\alpha}$, since $P_1|_{\tilde{L}_{w_i}}$ is function or recursion-free, by Observation 3.8 there exists an integer k such that $\exists w_i\,\tilde{L}_{w_i}\theta$ is *false* in $\Phi_{P_1}^{\uparrow k}$; again, by (3.1), $\exists\tilde{w}_i\,\tilde{B}_i\theta$ is *false* in $\Phi_{P_1}^{\uparrow k}$.
So, in any case, there exists a non-limit ordinal $\beta \leq \alpha$ such that $\exists\tilde{w}_i\,\tilde{B}_i\theta$ is *false* in $\Phi_{P_1}^{\uparrow\beta}$. Since this holds for any index i, the thesis follows. □

Checking A "a posteriori". We now show that condition **A** holds in P_0 iff it holds in any program of the unfold part of the transformation sequence. This gives us the opportunity of providing further sufficient conditions.
First let us restate **A** as follows:

A': For each substitution θ and non-limit ordinal β, if $H_i\theta$ is *false* in $\Phi_{P_1}^{\uparrow\beta+1}$, then $H_i\theta$ is *false* in $\Phi_{P_1}^{\uparrow\beta}$ as well.

Now, let P_1' be a program which is obtained from P_1 by applying some unfolding transformation. It is easy to see[3] that H_i satisfies **A'** in P_1 iff H_i satisfies **A'** in P_1'. So the advantage of **A'** over **A** is that it can be checked *a posteriori* at any time during the unfolding part of the transformation. So Proposition 3.7 can be restated as follows.

Proposition 3.9 Let P_1' be a program obtained from P_1 by (repeatedly) applying the unfolding operation. Let D'_{def} be the subset of P' corresponding to D_{def} in P. If for each clause c of D'_{def}, and for every variable y, local to the body of c

- $P_1'|_{\tilde{L}_y}$ is recursion-free or function-free,
  where $\tilde{L}_y$ denotes the subset of the body of c consisting of those literals where y occurs;

then each c_i satisfies **A** in P_1.

Proof. It is a straightforward generalization of the proof of Proposition 3.7. □

4. Correctness of the transformation

The aim of this section is to prove the correctness of the transformation schema wrt Fitting's semantics, Theorem 3.4.

[3]This is a direct consequence of Lemma 4.1, which is given in the next Section

4.1 Correctness of the unfold operation
First we consider the unfold operation. To prove its correctness we need the following technical Lemma.

Lemma 4.1 Let P' be the program obtained by unfolding an atom in a clause of program P. Then for each integer i and limit ordinal β,

- $\Phi_P^{\uparrow i} \subseteq \Phi_{P'}^{\uparrow i}$ and $\Phi_{P'}^{\uparrow i} \subseteq \Phi_P^{\uparrow 2i}$;
- $\Phi_P^{\uparrow i}(\Phi_P^{\uparrow \beta}) \subseteq \Phi_{P'}^{\uparrow i}(\Phi_{P'}^{\uparrow \beta})$ and $\Phi_{P'}^{\uparrow i}(\Phi_{P'}^{\uparrow \beta}) \subseteq \Phi_P^{\uparrow 2i}(\Phi_P^{\uparrow \beta})$.

Proof. The proof is given in [BCE93]. □

This brings us to a preliminary conclusion.

Corollary 4.2 (Correctness of the unfold operation) Let P' be the result of unfolding an atom of a clause in P. Then

- $Fit(P) = Fit(P')$ □

It should be mentioned that, because of the particular structure of the transformation sequence, here we never use self-unfoldings (that is, unfoldings in which the same clause is both the unfolded clause and one of the unfolding ones). Consequently the correctness of *Step 2* follows also from a result of Gardner and Shepherdson [GS91, Theorem 4.1] which states that if the program P' is obtained from P by unfolding (but not self-unfolding), then $Comp(P)$ and $Comp(P')$ are logically equivalent theories[4].

The following is a second, technical result on the consequences of an unfolding operation which will be needed in the sequel.

Lemma 4.3 Let P be a normal program, $cl : A \leftarrow \tilde{K}.$ be a definite, clause of P. Suppose also that cl is the only clause of P whose head unifies with $A\theta$. If P' is the program obtained by unfolding at least once all the atoms in $\tilde{K}$, then, for each non-limit ordinal α

- if $A\theta$ is *true* (resp. *false*) in $\Phi_P^{\uparrow \alpha+1}$ then $A\theta$ is *true* (resp. *false*) in $\Phi_{P'}^{\uparrow \alpha}$

Proof. Let us first give a simplified proof by considering the case when $\tilde{K}$ consists of two atoms H, J and we perform a single unfolding on them; we will later consider the general case.

Let $\{H_1 \leftarrow \tilde{B}_1., \dots, H_n \leftarrow \tilde{B}_n.\}$ be the set of clauses of P whose head unify with H via mgu's $\phi_1, \dots, \phi_n$, and let $\{J_1 \leftarrow \tilde{C}_1., \dots, J_m \leftarrow \tilde{C}_m\}$ be the set of clauses of P whose head unify with J. Unfolding H in cl and then J in the resulting clauses, will lead to the following program:
$P' = P \backslash \{cl\} \cup \{d_{i,j} : (A \leftarrow \tilde{B}_i, \tilde{C}_j.)\theta_{i,j})\}$
Where $\theta_{i,j} = mgu(J\phi_i, J_j)$. Here some of the clauses $d_{i,j}$ may be missing due to the fact that $J\phi_i$ and J_j may not unify, but this is of no relevance in the proof.

Note that the clauses $d_{i,j}$ are the only clauses of P' whose head could possibly unify with A.

Let $\tilde{y} = Var(H, J) \backslash Var(A)$ be the set of variables local to the body. We have to consider two cases.

a) $A\theta$ is *true* in $\Phi_P^{\uparrow \alpha+1}$. By the definition of Φ_P, $(\exists \tilde{y}\, H, J)\theta$ is *true* in $\Phi_P^{\uparrow \alpha}$. There has to be an extension σ of θ, $Dom(\sigma) = Dom(\theta) \cup \tilde{y} = Var(A, H, J)$ such that $(H, J)\sigma$ is *true* in $\Phi_P^{\uparrow \alpha}$. Let $H_i \leftarrow \tilde{B}_i$ and $J_j \leftarrow \tilde{C}_j$ be the clauses used to prove, respectively, $H\sigma$ and $J\sigma$. Hence there exists a τ such that $\theta_{i,j}\tau|_{Dom(\sigma)} = \sigma$, $H\sigma = H_i\theta_{i,j}\tau$, $J\sigma = J_j\theta_{i,j}\tau$, and $(\tilde{B}_i, \tilde{C}_j)\theta_{i,j}\tau$ is *true* in $\Phi_P^{\uparrow \alpha-1}$. By Lemma 4.1, $\Phi_P^{\uparrow \alpha-1} \subseteq \Phi_{P'}^{\uparrow \alpha-1}$, hence $(\tilde{B}_i, \tilde{C}_j)\theta_{i,j}\tau$ is *true* in $\Phi_{P'}^{\uparrow \alpha-1}$. It follows that $A\theta_{i,j}\tau = A\sigma = A\theta$ is *true* in $\Phi_{P'}^{\uparrow \alpha}$.

[4] In [GS91] this result is stated for the usual two-valued program's completion. By looking at the proof it is straightforward to check that it holds also for the three-valued case

b) $A\theta$ is *false* in $\Phi_P^{\uparrow\alpha+1}$. By the definition of Φ_P, $(\exists\tilde{y}\, H, J)\theta$ is *false* in $\Phi_P^{\uparrow\alpha}$. Hence for all extensions σ of θ, such that $Dom(\sigma) = Dom(\theta) \cup \tilde{y} = Var(A, H, J)$, we have that $(H, J)\sigma$ is *false* in $\Phi_P^{\uparrow\alpha}$.

Hence for all such σ's, and for all i, j and τ such that $\theta_{i,j}\tau|_{Dom(\sigma)} = \sigma$, $H\sigma = H_i\theta_{i,j}\tau$, $J\sigma = J_j\theta_{i,j}\tau$, we have that $(\tilde{B}_i, \tilde{C}_j)\theta_{i,j}\tau$ is *false* in $\Phi_P^{\uparrow\alpha-1}$. By Lemma 4.1, $\Phi_P^{\uparrow\alpha-1} \subseteq \Phi_{P'}^{\uparrow\alpha-1}$, hence $(\tilde{B}_i, \tilde{C}_j)\theta_{i,j}\tau$ is *false* in $\Phi_{P'}^{\uparrow\alpha-1}$. Since the clauses $d_{i,j}$ are the only ones that define A in P', we have that $A\theta_{i,j}\tau = A\sigma = A\theta$ is *false* in $\Phi_{P'}^{\uparrow\alpha}$.

Now to complete the proof, we have to observe two facts:

- First, that if we perform some further unfoldings on the resulting clauses, then we can only "speed up" the process of finding the truth value of A. In fact, by the same kind of reasoning used above, if $A\theta$ is *true* in $\Phi_{P'}^{\uparrow\alpha}$, and P'' is obtained from P' by unfolding some atoms in the bodies of the clauses $d_{i,j}$, then, for some $\beta \leq \alpha$, $A\theta$ is *true* in $\Phi_{P''}^{\uparrow\beta}$.

- Second, that if *cl* contains just one atom, or more than two atoms, then the exact same reasoning applies. □

4.2 The replacement operation

In order to prove the correctness of the unfold/fold transformation schema we will use (a simplified version of) the results in [BCE92, BCE93] on the simultaneous replacement operation.

The replacement operation has been introduced by Tamaki and Sato in [TS84] 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.

Similarly, *simultaneous* replacement consists in substituting a set of conjunctions of literals $\{\tilde{C}_1, \ldots, \tilde{C}_n\}$, with another corresponding set of conjunctions $\{\tilde{D}_1, \ldots, \tilde{D}_n\}$ in the bodies of the clauses of program P; here each $\tilde{C}_i$ represents a subset of the body of a clause of P and we assume that if $i \neq j$ then $\tilde{C}_i$ and $\tilde{C}_j$ do not overlap, that is, they are either found in different clauses or they represent disjoint subsets of the same clause.

Note that the fact that each $\tilde{C}_i$ may occur in the body of only one clause of P is not restrictive, as even if $i \neq j$, $\tilde{C}_i$ and $\tilde{C}_j$ may actually represent identical literals.

We now give a simplified version of the *applicability conditions* introduced in [BCE92, BCE93] in order to ensure the preservation of the semantics through the transformation. Such conditions depend on the semantics we associate to the program. Our first requirement is the semantic equivalence of the replacing and the replaced conjunctions of literals.

Definition 4.4 (Equivalence of formulas) Let E, F be first order formulas and P be a normal program.

- *F is equivalent to E wrt $Fit(P)$*, $F \sim_P E$, if for each ground substitution θ
  $E\theta$ is *true* (resp. *false*) in $Fit(P)$ iff $F\theta$ is. □

Note that $F \sim_P E$ iff $Fit(P) \models \forall(F \Leftrightarrow E)$.

Example 4.5 Let P be the program in Example 3.2. We have that

$med(Xs, Med) \sim_P \exists X, Y\; min(Xs, X) \wedge max(Xs, Y) \wedge Med = (X + Y)/2$ □

With many respects, and with some caution, two equivalent (conjunctions of) literals can be used interchangeably; for example, if q is a new predicate we want to give a definition to, and we know that $A \sim_P B$ then defining q by introducing the new clause $q \leftarrow A$ is, from Fitting's semantics viewpoint, equivalent to doing it by introducing $q \leftarrow B$.

Notice that the formula in Example 4.5 we had to specify X and Y as existentially quantified variables. When we want to replace the conjunction $\tilde{C}$ with $\tilde{D}$, in the clause *cl* the first requirement of those applicability conditions is the equivalence of $\exists\tilde{x}\,\tilde{C}$ and $\exists\tilde{x}\,\tilde{D}$, where $\tilde{x}$ is a set of "local variables", that is, variables which appear in $\tilde{C}$ and/or $\tilde{D}$, but which do not occur anywhere else in the clause that we are transforming. The equivalence is required as 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 *cl*, in which case the operation may introduce an infinite loop.

In order to prove that no fatal loops are introduced, we make use of a further concept. Here we say that the (closed) formula G is *defined* in the interpretation I, if the truth value of G in I is not *undefined.*

Definition 4.6 (not-slower) Let P be a normal program, E and F be first order formulas. Suppose that $F \sim_P E$. We say that

- *F is not-slower that E* if for each ordinal α and each ground substitution θ:
  if $E\theta$ is *defined* in $\Phi_P^{\uparrow\alpha}$, then $F\theta$ is *defined* in $\Phi_P^{\uparrow\alpha}$ as well. □

So F is not-slower that E if, for each θ, computing the truth value of $F\theta$ never requires more iterations that computing the one of $E\theta$. In a way we could then say that the definition of F is at least as efficient as the one of E.

The following Theorem shows that if the replacing conjunctions are *equivalent to* and *not-slower than* the replaced ones, then the replacement operation is correct.

Theorem 4.7 Let P' be a program obtained by simultaneously replacing the conjunctions $\{\tilde{C}_1, \dots, \tilde{C}_n\}$ with $\{\tilde{D}_1, \dots, \tilde{D}_n\}$ in the bodies of the clauses of P. If for each $\tilde{C}_i$, there exists a (possibly empty) set of variables $\tilde{x}_i$ such that the following three conditions hold:

(a) [locality of the variables in $\tilde{x}_i$]. $\tilde{x}_i$ is a subset of the variables local to $\tilde{C}_i$ and $\tilde{D}_i$, that is, $\tilde{x}_i \subseteq Var(\tilde{C}_i) \cup Var(\tilde{D}_i)$ and the variables in $\tilde{x}_i$ don't occur in $\{\tilde{D}_1, \dots, \tilde{D}_{i-1}, \tilde{D}_{i+1}, \dots, \tilde{D}_n\}$ nor anywhere else in the clause where $\tilde{C}_i$ is found.

(b) [equivalence of the replacing and replaced parts]. $\exists \tilde{x}_i \, \tilde{D}_i \; \sim_P \; \exists \tilde{x}_i \, \tilde{C}_i$

(c) [the D_i's are not-slower than the C_i's]. $\exists \tilde{x}_i \, \tilde{D}_i$ is *not-slower* than $\exists \tilde{x}_i \, \tilde{C}_i$.

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

Proof. This is a particular case of Corollary 3.16 in [BCE93]. □

A property we will need in the sequel is the following.

Proposition 4.8 Suppose that $A \leftarrow \tilde{C}, \tilde{E}$ is a clause of P and that P' is obtained from P by replacing $\tilde{C}$ with $\tilde{D}$ in such a way that the conditions of Theorem 4.7 are satisfied (so that $Fit(P) = Fit(P')$). Then

- Each time that $A\theta$ is *true* (resp. *false*) in $\Phi_P^{\uparrow\alpha}$ then $A\theta$ is *true* (resp. *false*) in $\Phi_{P'}^{\uparrow\alpha}$

Proof. This is a consequence of the fact that the replacing conjunction is not-slower than the replaced one. The formal proof is omitted here, it can be inferred by analyzing the proof of Theorem 3.15 in [BCE93] □

Before we provide the proof of the correctness of the four step schema, we need to establish some further preliminary results. The first one states that the converse of **A** holds in any case.

Proposition 4.9 Each time that $\exists \tilde{w} \, \tilde{B}\theta$ is *true* in some $\Phi_{P_1}^{\uparrow\beta}$, then there exists a non-limit ordinal $\alpha \leq \beta$ such that $\exists \tilde{w} \, \tilde{B}\theta$ is *true* in $\Phi_{P_1}^{\uparrow\alpha}$.

Proof. It follows at once from the definition of Fitting's operator. □

The following important transitive property holds:

Proposition 4.10 Let P and P' be normal programs, E and F be first order formulas;

- If $E \sim_P F$ and $Fit(P) = Fit(P')$, then $E \sim_{P'} F$. □

Now we can provide the details of the proof.

4.3 Correctness of the four step schema

We now prove the correctness of the four step schema. For the sake of simplicity we restrict ourselves to the case in which *Step 1* introduces only one clause. The extension to the general case is straightforward.

Let $P_1, \dots P_4$ be the sequence of programs obtained via the four step schema: P_1 is the initial program, i.e. the one that contains D_{def}. P_2, P_3 and P_4, are the programs obtained by applying steps *Step 2* through *Step 4*. In order to show that the Fitting's models of programs $P_1, \dots P_4$ coincide, we proceed as follows:

By the correctness of the unfolding operation, Corollary 4.2 we have that $Fit(P_1) = Fit(P_2)$.

We perform some further unfolding on some atoms of P_2, obtaining a new program that we will call P_{2u}, again by Corollary 4.2 we have that $Fit(P_2) = Fit(P_{2u})$; then we produce a "parallel sequence" of programs P_{3u}, P_{4u} by applying the simultaneous replacement operation, miming, to some extent, the original transformation. By applying Theorem 4.7 we will show that $Fit(P_{2u}) = Fit(P_{3u}) = Fit(P_{4u})$.

Finally we show that programs P_{3u} and P_{4u} are obtainable respectively from P_3 and P_4 by appropriately applying the unfold operation, and hence, by Corollary 4.2, that $Fit(P_3) = Fit(P_{3u})$ and that $Fit(P_4) = Fit(P_{4u})$. This will end the proof. Fig.1 illustrates both the original transformation and its parallel sequence.

$$\begin{tikzcd}
\begin{array}{c} P_1 = P \cup D_{\mathrm{def}} \\ \text{where } D_{\mathrm{def}} = \{c_0 : H \leftarrow \tilde{B}\} \end{array} \arrow[d] & \\
\begin{array}{c} P_2 = P \cup D_{\mathrm{unf}} \\ \text{where } D_{\mathrm{unf}} = \{c_i : A_i \leftarrow \tilde{U}_i, \tilde{N}_i\} \end{array} \arrow[r] \arrow[d] & \begin{array}{c} P_{2u} = P \cup D_{unf*} \\ \text{where } D_{unf*} = \{c'_{i,j} : (A_i \leftarrow \tilde{U}_i)\gamma_{i,j}, \tilde{D}_{i,j}\} \end{array} \arrow[d] \\
\begin{array}{c} P_3 = P \cup D_{\mathrm{fold}} \\ \text{where } D_{\mathrm{fold}} = \{c'_i : A_i \leftarrow \tilde{U}'_i, \tilde{N}_i\} \end{array} \arrow[r] \arrow[d] & \begin{array}{c} P_{3u} = P \cup D_{fold*} \\ \text{where } D_{fold*} = \{c'_{i,j} : (A_i \leftarrow \tilde{U}'_i)\gamma_{i,j}, \tilde{D}_{i,j}\} \end{array} \arrow[d] \\
\begin{array}{c} P_4 = P_{\mathrm{fold}} \cup D_{\mathrm{fold}} \\ \text{where } D_{\mathrm{fold}} = \{c'_i : A_i \leftarrow \tilde{U}'_i, \tilde{N}_i\} \end{array} \arrow[r] & \begin{array}{c} P_{4u} = P_{\mathrm{fold}} \cup D_{fold**} \\ \text{where } D_{fold**} = \{c'_{i,j} : (A_i \leftarrow \tilde{U}'_i)\gamma_{i,j}, \tilde{D}'_{i,j}\} \end{array}
\end{tikzcd}$$

Fig. 1. Diagram of the transformation (left) together with the "parallel sequence" (right).

Initial program. Let us establish some notation: $P_1 \dots P_4$ are the programs obtained by applying the four step schema to program P, and $c_0 : H \leftarrow \tilde{B}$. is the (only) clause added to program P in *Step 1*. We also denote by $\tilde{w}$ the set of the local variables of c_i, $\tilde{w} = Var(\tilde{B})\backslash Var(H)$. For the moment, let us make the following restriction:

- till the end of 4.3, we assume that $\tilde{B}$ doesn't contain negative literals.

Later, in subsection 4.4, we will prove the general case.

A simple consequence of the fact that c_0 is the only clause defining the predicate symbol of H is the following.

Observation 4.11

- $H \sim_{P_1} \exists\tilde{w}\, \tilde{B}$; □

P_2 and P_{2u}. P_2 is obtained by unfolding some of the atoms in $\tilde{B}$, so $P_2 = P \cup \{A_i \leftarrow \tilde{U}_i, \tilde{N}_i\}$, where the atoms in $\tilde{N}_i$ are those that have not been unfolded during *Step 1* (N stands for Not unfolded, while U for Unfolded), so $\tilde{N}_i$ is equal to a subset of an instance of $\tilde{B}$ and each A_i is an instance of H. We obtain P_{2u} from P_2 by further unfolding all the atoms in each $\tilde{N}_i$. We denote by $\{c_{i,j} : (A_i \leftarrow \tilde{U}_i)\gamma_{i,j}, \tilde{D}_{i,j}\}$ the set of clauses of P_{2u} obtained from clause c_i by unfolding the atoms in $\tilde{N}_i$.

By the correctness of the unfolding operation, Corollary 4.2, we have that

$$Fit(P_1) = Fit(P_2) = Fit(P_{2u}) \tag{4.2}$$

Moreover, the following properties hold:

Observation 4.12

- $H \sim_{P_{2u}} \exists\tilde{w}\, \tilde{B}$;
- H is not-slower than $\exists\tilde{w}\, \tilde{B}$ in P_{2u}.

Proof. From Observation 4.11 we have that $H \sim_{P_1} \exists\tilde{w}\, \tilde{B}$. The first statement follows then from (4.2) and Proposition 4.10. For the second, fix θ and let β be the least ordinal such that $\exists\tilde{w}\, \tilde{B}\theta$ is *true* (or *false*) in $\Phi_{P_{2u}}^{\uparrow\beta}$. The clauses defining the atoms in $\tilde{B}$ are the same in P_1, P_2 and P_{2u}, so $\exists\tilde{w}\, \tilde{B}$ is *true* (resp. *false*) in $\Phi_{P_1}^{\uparrow\beta}$ as well. From condition **A** and Proposition 4.9 we have that β is a non-limit ordinal. Hence, by the definition of Φ, $H\theta$ is *true* (resp. *false*) in $\Phi_{P_1}^{\uparrow\beta+1}$, and, by Lemma 4.3 $H\theta$ is *true* (resp. *false*) in $\Phi_{P_{2u}}^{\uparrow\beta}$. □

P_3 and P_{3u}. P_{3u} is obtained from P_{2u} as follows.

Suppose that in *Step 2* we performed a recursive folding on the clause $c_i : A_i \leftarrow \tilde{B}\theta, \tilde{R}_i, \tilde{N}_i$ of P_2, obtaining $c_i' : A_i \leftarrow H\theta, \tilde{R}_i, \tilde{N}_i$ in P_3. In the diagram we denote by $\tilde{U}_i'$ the conjunction of literals resulting from the application of the recursive folding on the conjunction $\tilde{U}_i$ (so $\tilde{U}_i = \tilde{B}\theta, \tilde{R}_i$ and $\tilde{U}_i' = H\theta, \tilde{R}_i$).

On P_{2u} we then perform the following. In each of the clauses $c_{i,j}$ we transform $\tilde{U}_i\gamma_{i,j}$ into $\tilde{U}_i'\gamma_{i,j}$ by replacing conjunctions of literals of the form $\tilde{B}\theta\gamma_{i,j}$ with $H\theta\gamma_{i,j}$ wherever needed; we call the resulting clauses $c_{i,j}'$. It is easy to see that if we unfold all the atoms in $\tilde{N}_i$ in the body of clause c_i' in P_3, then the resulting clauses are exactly the $c_{i,j}'$ in P_{3u}; this is best shown by the diagram. Hence P_{3u} is obtainable from P_3 by appropriately applying the unfolding operation. From Corollary 4.2 it follows that

$$Fit(P_3) = Fit(P_{3u}) \tag{4.3}$$

Now we show that $Fit(P_{2u}) = Fit(P_{3u})$. First we need the following.

Proposition 4.13 Let Q be a program, A, B be atoms and $\tilde{y}$ be a set of variables, such that $A \sim_Q \exists\tilde{y}\, B$. Suppose also that η is a renaming over $\tilde{y}$ and that for each variable z that occurs in A or B, but not in $\tilde{y}$, $Var(z\eta) \cap Var(\tilde{y}\eta) = \emptyset$. Then

- $A\eta \sim_Q \exists(\tilde{y}\eta)\ B\eta$

Proof. Straightforward. □

Since $\gamma_{i,j}$ results from unfolding the atoms in $\tilde{N}_i$, we have that $Dom(\gamma_{i,j}) \cap Var(c_i) \subseteq Var(\tilde{N}_i)$. Hence, by the conditions on θ in *Step 2*, $Dom(\gamma_{i,j}) \cap \tilde{w}\theta = \emptyset$ and $\tilde{w}\theta\gamma_{i,j} = \tilde{w}\theta$; so $\theta\gamma_{i,j}$ is a renaming over $\tilde{w}$, and the variables in $\tilde{w}\theta\gamma_{i,j}$ do not occur anywhere else in $c_{i,j}$. From Observation 4.12 and Proposition 4.13 we have that

- $H\theta\gamma_{i,j} \sim_{P_{2u}} \exists(\tilde{w}\theta\gamma_{i,j})\ \tilde{B}\theta\gamma_{i,j}$;
- $H\theta\gamma_{i,j}$ is not-slower than $\exists(\tilde{w}\theta\gamma_{i,j})\ \tilde{B}\theta\gamma_{i,j}$ in P_{2u}.

Since we obtained P_{3u} from P_{2u} by simultaneously replacing conjunctions (of the form) $\tilde{B}\theta\gamma_{i,j}$ with $H\theta\gamma_{i,j}$, by Theorem 4.7

$$Fit(P_{2u}) = Fit(P_{3u}). \tag{4.4}$$

Moreover, the following properties hold:

Observation 4.14

- $H \sim_{P_{3u}} \exists\tilde{w}\,\tilde{B}$;
- H is not-slower than $\exists\tilde{w}\,\tilde{B}$ in P_{3u}.

Proof. The first statement follows from Observation 4.12, (4.4) and Proposition 4.10. For the second first note that going from P_{2u} to P_{3u} we have affected only clauses that define the predicate *new*, moreover no other predicates definition depends on these clauses, in particular the atoms in $\tilde{B}$ are independent from them, hence, since H is not-slower than $\exists\tilde{w}\,\tilde{B}$ in P_{2u}, the statement follows from Proposition 4.8. □

P_4 and P_{4u}. P_4 is obtained from P_3 by transforming some of the clauses of P of the form $A \leftarrow \tilde{B}\theta, \tilde{E}$ into $A \leftarrow H\theta, \tilde{E}$.

Now we want to obtain P_{4u} from P_{3u} in such a way that P_{4u} is obtainable also from P_4 by unfolding the atoms in the conjunctions $\tilde{N}_i$.

Let $d\ :\ A \leftarrow \tilde{B}\theta, \tilde{E}$ be one of the clauses of P_3 that are transformed in *Step 4*. First note that d belongs both to P_3 and P_{3u}, in fact d was already present it the original program P, and never modified. We can then apply the same operations to the clauses of P_{3u}. Observe that for the conditions on θ given in *Step 4*, and by Observation 4.14 we have that

Observation 4.15

- $H\theta \sim_{P_{3u}} \exists(\tilde{w}\theta)\ \tilde{B}\theta$
- $H\theta$ is not-slower than $\exists(\tilde{w}\theta)\ \tilde{B}\theta$ in P_{3u} □

Second, notice that in case that d was used as unfolding clause for going from P_2 to P_{2u}, then some instances of $\tilde{B}\theta$ were propagated into P_{3u}. Using the notation of the diagram, this is the case when some $\tilde{N}_i$ (in P_2) is of the form $A', \tilde{F}_i$ where A and A' are unifiable atoms, then one of the $\tilde{D}_{i,j}$ (in P_{2u}) is of the form $\tilde{D}_{i,j} = (\tilde{B}, \tilde{F}_i)\theta'$. However, if we unfold N_i in P_4, what we get is $\tilde{D}'_{i,j} = H\theta', \tilde{F}_i$, that has $H\theta'$ instead of $\tilde{B}\theta'$. By the same argument used for $\theta\gamma_{i,j}$ in 4.3, we have that

Observation 4.16

- $H\theta' \sim_{P_{3u}} \exists(\tilde{w}\theta')\ \tilde{B}\theta'$
- $H\theta'$ is not-slower than $\exists(\tilde{w}\theta')\ \tilde{B}\theta'$ in P_{3u} □

So in order to obtain P_{4u} from P_{3u} we have then to do two things: First, replace $\tilde{B}\theta$, with the corresponding $H\theta$ in all the clauses d that are transformed in *Step 4*. Second, replace $\tilde{B}\theta'$ with $H\theta'$ in the $\tilde{D}_{i,j}$ so that P_{4u} contains $\tilde{D}'_{i,j}$ instead of $\tilde{D}_{i,j}$. This tantamounts to the application of a simultaneous replacement.
From Observations 4.15 and 4.16, and Theorem 4.7 we have that

$$Fit(P_{3u}) = Fit(P_{4_u}) \tag{4.5}$$

Moreover P_{4u} is obtainable from P_4 by unfolding all the atoms in the conjunctions $\tilde{N}_i$ in the clauses where they occur. Hence

$$Fit(P_4) = Fit(P_{4u}). \tag{4.6}$$

So far, because of (1), (2), (3), (4) and (5), we have the following

Proposition 4.17 If condition **A** holds and $\tilde{B}$ does not contain negative literals, then

- $Fit(P_1) = Fit(P_2) = Fit(P_3) = Fit(P_4)$ □

4.4 The general case
We can finally prove Theorem 3.4. Let us state it again.

Theorem 3.4 Let $P_1, \ldots, P_4$ be a sequence of programs obtained applying the transformation *schema* to program P, Let also $D_{\text{def}} = \{H_i \leftarrow \tilde{B}_i\}$ be the set of clauses introduced in *Step 1*, and, for each i, $\tilde{w}_i$ be the set of local variables of c_i: $\tilde{w}_i = Var(\tilde{B}_i) \backslash Var(H_i)$. If each c_i in D_{def} satisfies the following condition:

A each time that $\exists\tilde{w}_i\, \tilde{B}_i\theta$ is *false* in some $\Phi_{P_1}^{\uparrow\beta}$, then there exists a non-limit ordinal $\alpha \leq \beta$ such that $\exists\tilde{w}_i\, \tilde{B}_i\theta$ is *false* in $\Phi_{P_1}^{\uparrow\alpha}$

Then $Fit(P_1) = Fit(P_2) = Fit(P_3) = Fit(P_4)$.

Proof. We consider here the simplified case in which *Step 1* introduces only one clause which in turn contains only one negative literal in the body, i.e. $D_{\text{def}} = \{c_0 :\ H \leftarrow \neg l(\tilde{y}), \tilde{B}'\}$. The generalization to the case of multiple clauses and multiple negative literals is straightforward and omitted here. Notice that if c_0 contained no negative literals, then the result would following directly from Proposition 4.17.
We now perform a double transformation on P_1: first, we enlarge it with the following new definition: $d : notl(\tilde{y}) \leftarrow \neg l(\tilde{y})$; then, we replace each instance $\neg l(\tilde{t})$ of $l(\tilde{y})$ that occurs in the body of a clause with the corresponding instance $notl(\tilde{t})$ of $notl(\tilde{y})$. This replacement operation clearly preserves Fitting's model of the programs, in fact it can be undone by unfolding. Let us call P_1' the program so obtained. We have that

$$Fit(P_1) = Fit(P_1')|_{B_{P_1}} \tag{4.7}$$

Where $Fit(P_1')|_{B_{P_1}}$ denotes the restriction of $Fit(P_1')$ to the atoms in the Herbrand base of P_1.

Now P_1' contains, instead of clause c_0, the following: $c_0' = H \leftarrow notl(\tilde{y}), \tilde{B}'$. which is a *definite* clause.

Now notice that, since the unfold operation is defined only for positive literals, then $\neg l(\tilde{y})$ is never unfolded in the transformation $P_1 \ldots P_4$. It follows that, by performing the same operations used for going from P_1 to P_4, we can obtain another "parallel sequence" $P_1' \ldots P_4'$ that starts with program P_1'. By the same arguments used to prove (4.7), we have that, for $i \in [1 \ldots 4]$,

$$Fit(P_i) = Fit(P_i')|_{B_{P_1}} \tag{4.8}$$

Moreover, by Proposition 4.17,

$$Fit(P_1') = Fit(P_2') = Fit(P_3') = Fit(P_4') \tag{4.9}$$

From (4.8) and (4.9) the thesis follows. □

REFERENCES

[AAP78] L. Aiello, G. Attardi, and G. Prini. Towards a more declarative programming style. In E. J. Neuhold, editor, *Proc. of the IFIP Conference on Formal Description of Programming Concepts*, pages 121–137. North-Holland, 1978.

[AD92] C. Aravidan and P. M. Dung. Partial deduction of logic programs w.r.t. well-founded semantics. In H. Kirchner G. Levi, editor, *Proceedings of the Third International Conference on Algebraic and Logic Programming*, pages 384–402. Springer-Verlag, 1992.

[AD93] C. Aravidan and P. M. Dung. On the correctness of Unfold/Fold transformation of normal and extended logic programs. Technical report, Division of Computer Science, Asian Institute of Technology, Bangkok, Thailand, April 1993.

[Apt90] K. R. Apt. Introduction to Logic Programming. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics. Elsevier, Amsterdam and The MIT Press, Cambridge, 1990.

[BC93] A. Bossi and N. Cocco. Basic Transformation Operations which preserve Computed Answer Substitutions of Logic Programs. *Journal of Logic Programming*, 16(1&2):47–87, 1993.

[BCE92] A. Bossi, N. Cocco, and S. Etalle. Transforming Normal Programs by Replacement. In A. Pettorossi, editor, *Meta Programming in Logic - Proceedings META'92*, volume 649 of *Lecture Notes in Computer Science*, pages 265–279. Springer-Verlag, Berlin, 1992.

[BCE93] A. Bossi, N. Cocco, and S. Etalle. Simultaneous replacement in normal programs. Technical Report CS-R9357, CWI, Centre for Mathematics and Computer Science, Amsterdam, The Netherlands, August 1993. Available via anonymous ftp at ftp.cwi.nl, or via xmosaic at http://www.cwi.nl/cwi/publications/index.html.

[BD77] R.M. Burstall and J. Darlington. A transformation system for developing recursive programs. *Journal of the ACM*, 24(1):44–67, January 1977.

[Cla78] K. L. Clark. Negation as failure rule. In H. Gallaire and G. Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, 1978.

[CS77] K.L. Clark and S. Sickel. Predicate logic: a calculus for deriving programs. In *Proceedings of IJCAI'77*, pages 419–120, 1977.

[Fit85] M. Fitting. A Kripke-Kleene semantics for Logic Programs. *Journal of Logic Programming*, 2(4):295–312, 1985.

[GS91] P.A. Gardner and J.C. Shepherdson. Unfold/fold transformations of logic programs. In J-L Lassez and G. Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*. MIT Press, 1991.

[Hog81] C.J. Hogger. Derivation of logic programs. *Journal of the ACM*, 28(2):372–392, April 1981.

[KK90] T. Kawamura and T. Kanamori. Preservation of Stronger Equivalence in Unfold/Fold Logic Programming Transformation. *Theoretical Computer Science*, 75(1&2):139–156, 1990.

[Kle52] S.C. Kleene. *Introduction to Metamathematics*. D. van Nostrand, Princeton, New Jersey, 1952.

[Kom82] H.J. 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.

[Llo87] J. W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, Berlin, 1987. Second edition.

[LS91] J. W. Lloyd and J. C. Shepherdson. Partial Evaluation in Logic Programming. *Journal of Logic Programming*, 11:217–242, 1991.

[Mah87] M.J. Maher. Correctness of a logic program transformation system. IBM Research Report RC13496, T.J. Watson Research Center, 1987.

[PP91] 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.

[Sat92] T. Sato. Equivalence-preserving first-order unfold/fold transformation system. *Theoretical Computer Science*, 105(1):57–84, 1992.

[Sek89] H. Seki. Unfold/fold transformation of stratified programs. In G. Levi and M. Martelli, editors, *6th International Conference on Logic Programming*, pages 554–568. The MIT Press, 1989.

[Sek90] H. Seki. A comparative study of the Well-Founded and Stable model semantics: Transformation's viewpoint. In D. Pedreschi W. Marek, A. Nerode and V.S. Subrahmanian, editors, *Workshop on Logic Programming and Non-Monotonic Logic, Austin, Texas, October 1990*, pages 115–123. Association for Logic Programming and Mathematical Sciences Institute, Cornell University, 1990.

[Sek91] H. Seki. Unfold/fold transformation of stratified programs. *Theoretical Computer Science*, 86(1):107–139, 1991.

[Sek93] H. Seki. Unfold/fold transformation of general logic programs for the Well-Founded semantics. *Journal of Logic Programming*, 16(1&2):5–23, 1993.

[TS84] H. Tamaki and T. Sato. Unfold/Fold Transformations of Logic Programs. In Sten-Åke Tärnlund, editor, *Proc. Second Int'l Conf. on Logic Programming*, pages 127–139, 1984.