Theoretical Computer Science 105 (1992) 57–84
Elsevier

Equivalence-preserving first-order unfold/fold transformation systems*

Taisuke Sato

Electrotechnical Laboratory, Umezono, Tsukuba, Ibaraki, Japan 305

Abstract

Sato, T., Equivalence-preserving first-order unfold/fold transformation systems, Theoretical Computer Science 105 (1992) 57–84.

Two unfold/fold transformation systems for first-order programs, one basic and the other extended, are presented. The systems comprise an unfolding rule, a folding rule and a replacement rule. They are intended to work with a first-order theory Δ specifying the meaning of primitives, on top of which new relations are built by programs. They preserve the provability relationship $\Delta \cup \Gamma \vdash G$ between a call-consistent program Γ and a goal formula G such that Γ is strict with respect to G. They also preserve the logical consequence relationship in three-valued logic.

1. Introduction

The unfold/fold transformation has been widely recognized as a powerful program transformation technique. In particular, in logic programming, it has been applied not only to program optimization (as it has been the case with functional programming) but to program derivation as well. Take, for instance, the following first-order specification Γ_0 for all-nonzero(L), which states that every element in the list L is nonzero.

Example 1.1.

$$\begin{aligned}\Gamma_0\colon\quad & \text{all-nonzero}(L) \leftrightarrow \forall Y(\underline{\text{mem}(Y, L)} \rightarrow Y \neq 0)\\ & \text{mem}(Y, [Y \mid Z])\\ & \text{mem}(Y, [U \mid V]) \leftarrow \text{mem}(Y, V)\end{aligned}$$

Correspondence to: T. Sato, Electrotechnical Laboratory, Umezono, Tsukuba, Ibaraki, Japan 305.

* This paper is an expanded version of the one presented at the 2nd Algebraic and Logic Programming Conference held in Nancy, 1990, and the title has been changed a little.

First, unfold the underlined formula using the iff definition [19] of mem[1] and label mem(Y, V) by u to indicate that it was introduced by unfolding.

$$\Gamma_1: \quad \text{all-nonzero}(L) \leftrightarrow \forall Y, Z(L = [Y \mid Z] \rightarrow Y \neq 0) \\ \wedge \forall U, V(L = [U \mid V] \rightarrow \underline{\forall Y(\text{mem}^{u}(Y, V) \rightarrow Y \neq 0))}$$

Seeing that all of the atomic formulae concerned are labeled u, we fold the underlined part by the definition for all-nonzero (see Section 4 for folding conditions) and have

$$\Gamma_2: \quad \text{all-nonzero}(L) \leftrightarrow \forall Y, Z(L = [Y \mid Z] \rightarrow Y \neq 0) \\ \wedge \forall U, V(L = [U \mid V] \rightarrow \text{all-nonzero}(V))$$

which is equivalent in the Herbrand universe to

$$\Gamma_3: \quad \text{all-nonzero}(L) \leftrightarrow \{\forall Y, Z(L \neq [Y \mid Z]) \vee \exists Y, Z(L = [Y \mid Z] \wedge Y \neq 0)\} \\ \wedge \{\forall U, V(L \neq [U \mid V]) \vee \exists U, V(L = [U \mid V] \\ \wedge \text{all-nonzero}(V))\}$$

Note that we already have succeeded in the derivation of an executable program up until this stage, as Γ_3 is actually a definite program[2] (written in the form of an iff definition). However, seeking for a more efficient program, we factor out the common subformulae, $\forall Y, Z(L \neq [Y \mid Z])$ and $\forall U, V(L \neq [U \mid V])$, to get

$$\Gamma_4: \quad \text{all-nonzero}(L) \leftrightarrow \forall Y, Z(L \neq [Y \mid Z]) \vee \{\exists Y, Z(L = [Y \mid Z]) \wedge Y \neq 0 \\ \wedge \text{all-nonzero}(Z)\}$$

Later we see that Γ_4 has the same logical consequence relationship as Γ_0 w.r.t. literals (Theorem 4.1).

Looking back, by the way, it is apparent that the crux of the whole derivation process lies in the elimination of negation by folding done between Γ_1 and Γ_2 (compare the negative occurrence of mem(Y, V) in Γ_1 and the positive occurrence of all-nonzero(V) in Γ_2 in their right-hand sides). To see the potential of unfold/fold transformation for negation elimination, let us consider the negation of $\exists Y \text{mem}(Y, L)$, i.e. $\neg \exists Y \text{mem}(Y, L)$ (L has no elements) and derive a program for it. We start with $\text{empty}(L) \leftrightarrow \forall Y(\text{mem}(Y, L) \rightarrow \mathbf{f})$, where $\mathbf{f}$ stands for falsity.

Example 1.2.

$$\Gamma_0': \quad \text{empty}(L) \leftrightarrow \forall Y(\text{mem}(Y, L) \rightarrow \mathbf{f})$$

[1] Generally speaking, unfolding means the replacement of a procedure call with the procedure body, whereas folding is the reverse operation. As for the present case, the iff definition becomes $\text{mem}(Y, L) \leftrightarrow \exists Z(L = [Y \mid Z]) \vee \exists U, V(L = [U \mid V] \wedge \text{mem}(Y, V))$. We consider the left-hand side (right-hand side) as a procedure call (procedure body).

[2] In this paper, we mean by a clause a formula of the form $H \leftarrow B$, where H is an atom. When B, the body, is a conjunction of atoms, we call it a definite clause. A definite program is a finite set of definite clauses.

The same (an isomorphic) transformation process as Example 1.1 leads to

Γ_4': $\mathrm{empty}(L) \leftrightarrow \forall Y, Z(L \neq [Y|Z]) \vee \{\exists Y, Z(L = [Y|Z]) \wedge \mathbf{f} \wedge \mathrm{empty}(Z)\}$

Simplification using $A \wedge \mathbf{f} \leftrightarrow \mathbf{f}$ and $A \vee \mathbf{f} \leftrightarrow A$ gives

Γ_4'': $\mathrm{empty}(L) \leftrightarrow \forall Y, Z(L \neq [Y|Z])$

By Theorem 4.1 again, this is guaranteed to calculate the finite failure set of $\exists Y \mathrm{mem}(Y, L)$.

These examples suggest that an unfold/fold approach offers a quick way to program derivation from first-order specifications and enables one to take the negation of programs containing internal (local) variables[3] (see, for example, [2] for the problem with internal variables). It is conceptually simple and easy to implement. Above all, a derived program P is always sound, meaning that any answer computed by P satisfies the original specification S because P is nothing more than a logical consequence of S. Unfortunately, the very same fact might cause P to be incomplete; P might be weaker than S as a logical theory. Hence, if we wish P to hold the same proof-theoretic strength as S, it is necessary to place some restrictions on their form or on the transformation process.

Here we introduce a class of formulae called *first-order programs* with a view toward treating uniformly both specifications, such as Γ_0, and programs, such as Γ_4. They are defined as logic programs allowing an arbitrary first-order formula in the clause body. They are straightforward generalizations of definite programs and seem to encompass most of the useful first-order specifications met in daily programming. Some first-order programs are not executable, and can only be considered as specifications. However, many of these are transformable into an executable form by completely mechanical unfold/fold transformation [28].

In this paper, based on the recent results on three-valued semantics for first-order programs, we present general unfold/fold transformation systems, a basic one and an extended one. They preserve the proof-theoretic strength of first-order programs under certain conditions. We say general firstly because an arbitrary algebraic structure can be a domain of discourse, not to speak of the Herbrand universe (the algebra of finite trees). Secondly because, although they are equivalence-preserving, our systems allow an arbitrarily complex formula as the body of a predicate definition, even if it becomes negatively recursive.

Section 2 gives a brief overview of related work on unfold/fold transformation of logic programs. In Section 3, we describe properties of three-valued logic. A basic transformation system will be presented in Section 4 and the extended one in Section 5. Section 6 contains concluding remarks. Since this paper is more of a theoretical nature, examples are small and implementation issues are not discussed. The reader is assumed to be familiar with logic programming, unfold/fold transformation and elementary logic [6, 19, 31].

[3] A variable in a clause is said to be internal if it is free and occurs only in the body.

2. Related work

The technique of unfold/fold transformation emerged during the mid seventies and was immediately transplanted to logic programming [4, 7, 25]. Since then, two lines of development have been observed. One pursues equivalence-preserving (w.r.t. canonical models) unfold/fold transformation of definite or general programs.[4] The other is primarily aimed at the synthesis of definite programs from arbitrary first-order specifications, and does not assume any specific models. Both approaches, however, are intertwined in many ways.

The first approach began with the introduction of unfolding to logic programming by Komorowski [15]. It was proved that unfolding preserves procedural semantics of definite programs. Folding was introduced by Tamaki and Sato [31]. They clarified syntactic conditions under which any combination of folding and unfolding preserves the success set of definite programs. Their system was considerably reinforced later by the introduction of multilayered definitions [32] and also by that of counters [12]. These systems permit recursive clauses to define new predicates. Both of them, however, focused on the preservation of the success set. It remained unknown what condition preserves finite failure.

In 1987, Maher formulated a transformation system which is somewhat restrictive ("the folding clause comes from the current program" and "no unfolding on direct recursion") but can preserve finite failure as well as success of definite programs [22]. Then he extended it to the one for general programs (though unfolding was kept restricted to positive goals) and proved the preservation of the perfect model semantics of locally stratified programs [24]. Independently, Seki [29] proposed a transformation system which preserves not only the success and finite failure sets of definite programs but also those of stratified programs. In his system, a new predicate is introduced by a definite clause and the unfolding of negative goals is not permitted. In a recent paper, he revealed the preservation of the well-founded semantics and stable model semantics of general programs by unfold/fold transformation [30].

In connection with finite failure, Sato and Tamaki presented a transformation algorithm called "negation technique" in [27]. When applied to a definite program without internal variables, it produces one that mimics the finitely failed computations of the original program. The point is that it makes possible, in an indirect manner though, the synthesis of definite programs for universally quantified goals [27]. Kanamori and Horiuchi [13] have shown, however, that the same could be done, more directly, by using their generalized unfold/fold rules incorporating the unfolding of negative goals. The negation technique itself was subsequently refined by Barbuti et al. [2] so as to be able to cope with definite programs with internal variables.

[4] A general program is a finite set of clauses of the form $H \leftarrow B$, with B being a conjunction of literals [19]. Each literal in B is called a goal. A general program with no recursion through negative goals is called a stratified program [1].

To return to unfold/fold transformation for program synthesis, after its initial success [7, 25], the viability of this approach was extensively investigated by Hogger [11]. A more deterministic approach was taken by Dayantis [9] for a certain class of first-order specifications (programs). Sato and Tamaki [28] completely mechanized the synthesis process for a wider class of first-order programs by introducing universal continuation. It automatically synthesizes auxiliary predicates and can take the negation of definite (and general) programs with internal variables. Lugiez [21] specified a deduction procedure for first-order programs with an algorithm for treating formulae including inequalities.

Despite the progress, however, the question of logical equivalence among the synthesized programs was left unanswered. A clue was supplied by Kunen in his investigation of logic programming semantics based on three-valued logic [16, 18]. He proved a theorem which enables us to compare "relative strength" between first-order programs and, hence, facilitates the design of an unfold/fold transformation system that preserves the equivalence of programs. Our transformation systems presented here are just examples. While they belong to the line of synthetic unfold/fold transformation, and cover all first-order programs, their folding conditions, when applied to stratified programs, have direct correspondence with the ones proposed by Seki [29].

3. Preliminaries

3.1. First-order programs

We assume that all syntactic objects are in a fixed countable language of first-order predicate calculus with equality =. Terms and formulae are defined in the usual way. We use $F[x]$ to emphasize that F has free variables among $x=x_1,\dots,x_n$. For terms $t=t_1,\dots,t_n$, $F[t/x]$ means the result of simultaneous substitution of t_i for x_i ($1\leqslant i\leqslant n$) in F. When no confusion occurs, we write $F[t]$ instead of $F[t/x]$. We extend this notation to formulae. For instance, $F[E/G]$ denotes a formula F whose subformula G is replaced by E. We assume that variable name clash is implicitly avoided by suitable renaming. A formula without free variables is called a *sentence*.

A *predicate definition* about p is a formula of the form $p(x_1,\dots,x_n)\leftrightarrow F$, in which $x_1,\dots,x_n$ are distinct variables and F a formula whose free variables are among $x_1,\dots,x_n$. It is a generalization of the iff definition of a definite program [19]. $p(x_1,\dots,x_n)$ is called a head and F a body. $x_1,\dots,x_n$ are implicitly universally quantified at the front. Note that negative self-recursion is allowed in a predicate definition.

A *first-order program* is a finite set of predicate definitions, one for each predicate. We use Γ as a metavariable ranging over first-order programs. In what follows, $p(x_1,\dots,x_n)\leftrightarrow F$ is written as $p(x)\leftrightarrow F$, with x possibly subscripted.

The purpose of first-order programs is to allow a user to define new predicates on top of old ones. It is, therefore, convenient to think that there is a fixed base language

L_B, a first-order language with equality appropriate for defining primitive predicates and functions, and that a first-order program $\Gamma = \{p_i(x_i) \leftrightarrow F_i \mid 1 \leqslant i \leqslant N\}$ $(N \geqslant 0)$ belongs to an expanded language $L_B + \{p_1, \ldots, p_N\}$, i.e., an expansion of L_B with new predicate symbols $\{p_1, \ldots, p_N\}$ not in L_B. We call each p_i a *user predicate*.

Since primitives such as $=$, $<$, $\leqslant$ are to be the building blocks of new predicates, they are prohibited from appearing at the head of a predicate definition. We assume that their meanings are axiomatically determined by a consistent set of sentences in L_B. We call it a *base theory* and use Δ for base theories. Δ depends on our choice. For instance, it can be Clark's equational theory [16], Presburger arithmetic and Peano arithmetic to list a few. Even the empty set and the set of true sentences over natural numbers are among the options.

Although one might naturally question the consistency of $\Delta \cup \Gamma$, where Γ is a program and Δ a base theory, we may say that for most programs, $\Delta \cup \Gamma$ is consistent; call-consistent programs (those having no recursion through an odd number of negative goals, see Section 3 for their definition) and order-consistent ones (the ground version of call-consistent programs) are consistent[5] [18, 26]. In the rest of this section, we restate some results in [16, 18] for the sake of self-containedness.

3.2. Three-valued logic

Semantically, we work with Kleene's three-valued logic [14], as far as user predicates are concerned. It allows a predicate to be undefined at some values of its arguments. It has three truth values $\{\mathbf{t}, \mathbf{f}, \mathbf{u}\}$, where $\mathbf{t}$ stands for true, $\mathbf{f}$ for false and $\mathbf{u}$ for undefined, respectively. The truth table behaves as usual w.r.t. $\mathbf{t}$ and $\mathbf{f}$. For instance, $\neg\mathbf{t} = \mathbf{f}$, $\neg\mathbf{f} = \mathbf{t}$, $A \vee B = \mathbf{t}$ iff either A or B is $\mathbf{t}$ and $A \vee B = \mathbf{f}$ iff both A and B are $\mathbf{f}$. But for $\mathbf{u}$ being undefined, we have $\neg\mathbf{u} = \mathbf{u}$, and $A \vee B = \mathbf{u}$ iff either A or B is $\mathbf{u}$ and neither A nor B is $\mathbf{t}$. $A \wedge B$ is defined as $\neg(\neg A \vee \neg B)$. $\exists x A$ is treated as an infinite disjunction and $\forall x A$ as an infinite conjunction. Finally, we define $A \rightarrow B$ as $\neg A \vee B$ and $A \leftrightarrow B$ as $A \rightarrow B$ and $B \rightarrow A$.

According to the last rule, however, $\mathbf{u} \leftrightarrow \mathbf{u}$ gets the value $\mathbf{u}$, not $\mathbf{t}$, a fact which causes some awkwardness. Another logical connective $A \Leftrightarrow B$ is, therefore, introduced with a definition such that $A \Leftrightarrow B = \mathbf{t}$ iff $A = B$, and $\mathbf{f}$ otherwise. With the introduction of $\Leftrightarrow$, we redefine a predicate definition to be a formula of the form $p(x) \Leftrightarrow F[x]$, where $p(x)$ is an atomic formula and F a formula containing no occurrences of $\Leftrightarrow$. We stipulate that $\Leftrightarrow$ always appears as the top level biconditional of predicate definitions and nowhere else, and further that when we talk about predicate definitions in the context of two-valued logic, $\Leftrightarrow$ implicitly stands for $\leftrightarrow$. As a result of this modification, every program, even $\{p \Leftrightarrow \neg p\}$, has a three-valued model (defined later).

[5] Call-consistency [18, 26] and order-consistency [26] are defined only for general programs. It is obvious, however, that every first-order program is reducible to a general program which is a definitional extension of the original one, by introducing new predicates as many as necessary [20]. Besides, the reduction will be done without destroying the signed dependency relation (see Section 3.5) among the original predicates. Hence, the results in [26] and [16, 18] apply to first-order programs as well.

Suppose a program $\Gamma = \{p_i(x_i) \Leftrightarrow F_i \mid 1 \leqslant i \leqslant N\}$ is given. We assume here that there are three propositional constants $\{\mathbf{t}, \mathbf{f}, \mathbf{u}\}$ available corresponding to the truth values $\{\mathbf{t}, \mathbf{f}, \mathbf{u}\}$ (we use the same symbols), and further assume that $\mathbf{t}$ and $\mathbf{f}$ are in L_B, but that $\mathbf{u}$ is a new symbol.

A *three-valued relational structure* (or a *structure* in short) M for $L_B + \{\mathbf{u}\} + \{p_1, \ldots, p_N\}$ $(N \geqslant 0)$ is a pair $\langle D, I\rangle$ such that D, called M's domain, is a non-empty set and I an interpretation over D, which is *two-valued for* L_B *and three-valued for each user predicate symbol in* $\{p_1, \ldots, p_N\}$. I must take $\mathbf{t}, \mathbf{f}, \mathbf{u}$ as true, false and undefined, respectively, and $=$ as the identity relation in D.

Let $\boldsymbol{a} = a_1, \ldots, a_n$ be elements (may be repeated) of D and suppose that a formula F has free variables among $x = x_1, \ldots, x_n$. Then we designate the truth value of $F[\boldsymbol{a}/x]$ in I by $\text{val}(F[\boldsymbol{a}/x], I)$, where $\boldsymbol{a}/x$ means that a_i is assigned to x_i $(1 \leqslant i \leqslant n)$. We write $F[\boldsymbol{a}]$ instead of $F[\boldsymbol{a}/x]$ provided the context allows. The truth value of $F[\boldsymbol{a}]$ in a structure M is designated by $\text{val}(F[\boldsymbol{a}], M)$.

We say M *satisfies* a sentence F or, equivalently, M is a *model* of F iff $\text{val}(F, M) = \mathbf{t}$. Accordingly, M satisfies a predicate definition $p(x) \Leftrightarrow F[x]$ iff $\text{val}(p(\boldsymbol{a}), M) = \text{val}(F[\boldsymbol{a}], M)$ for any $\boldsymbol{a}$ in M's domain. When M satisfies every sentence in a set Ψ, we call M a model of Ψ, or say that M satisfies Ψ. In parallel with two-valued logic, we use $\models_3$ to designate the *three-valued logical consequence relationship*. Namely,

$$\Psi \models_3 F \quad \text{iff} \quad \text{val}(F, M) = \mathbf{t} \text{ for every model } M \text{ of } \Psi.$$

We use $\models_2$ for the two-valued logical consequence relationship which is identical, by the completeness of first-order logic, to the provability relationship in two-valued logic designated by $\vdash$.

3.3. *Generative chain and generative extension*

Having defined terminology, we proceed to investigate a special model-theoretic relationship between first-order programs and their three-valued models. Let $\Gamma = \{p_i(x_i) \Leftrightarrow F_i \mid 1 \leqslant i \leqslant N\}$ be a program and Δ a base theory in L_B.

Introduce a partial ordering $v < w$ over $\{\mathbf{t}, \mathbf{f}, \mathbf{u}\}$, which is defined by

$$\mathbf{u} < \mathbf{t} \quad \text{and} \quad \mathbf{u} < \mathbf{f}.$$

We use $v \leqslant w$ for $v < w$ or $v = w$. For formulae $F[x]$ and $G[x]$ in $L_B + \{\mathbf{u}\} + \{p_1, \ldots, p_N\}$, define a transitive relation "$F[x] \leqslant_\Delta G[x]$" by

$F[x] \leqslant_\Delta G[x]$ iff $\text{val}(F[\boldsymbol{a}], M) \leqslant \text{val}(G[\boldsymbol{a}], M)$ holds for any structure M for $L_B + \{\mathbf{u}\} + \{p_1, \ldots, p_N\}$ which is a model of Δ, and at any $\boldsymbol{a}$ in its domain.

By definition, $\mathbf{u} \leqslant_\Delta G$ holds for arbitrary G and Δ. In the case of $F[x] \leqslant_\Delta G[x]$ and $G[x] \leqslant_\Delta F[x]$, we write $G[x] =_\Delta F[x]$. It means that in an arbitrary model of Δ, and at any $\boldsymbol{a}$ in its domain, $G[\boldsymbol{a}]$ and $F[\boldsymbol{a}]$ take the same truth value$\in\{\mathbf{t}, \mathbf{f}, \mathbf{u}\}$, whatever three-valued interpretation we may assign to $\{p_1, \dots, p_N\}$. Obviously, $G[x] \Leftrightarrow F[x]$ iff $G[x] =_{\emptyset} F[x]$ and $G[x] =_{\Delta} F[x]$ for any Δ if $G[x] \Leftrightarrow F[x]$.

Call a series of formulae $\sigma_0[x], \sigma_1[x], \dots$ an *increasing chain* w.r.t. Δ if it holds that $\sigma_0[x] \leqslant_{\Delta} \sigma_1[x] \leqslant_{\Delta} \dots$ The most typical increasing chains are those generated from programs. For a program $\Gamma = \{p_i(x_i) \Leftrightarrow F_i \mid 1 \leqslant i \leqslant N\}$, define a series of formulae $p_i^{(n)}[x_i]$ $(1 \leqslant i \leqslant N,\ n = 0, 1, \dots)$ by

$$p_i^{(0)}[x_i] = \mathbf{u}$$

$$p_i^{(n+1)}[x_i] = F_i[p_1^{(n)}, \dots, p_N^{(n)}] \quad \text{for } n > 0$$

Here (and henceforth) $F[p_1^{(n)}, \dots, p_N^{(n)}]$ stands for the result of simultaneous substitution of $p_i^{(n)}[t/x]$ for $p_i(t)$ $(1 \leqslant i \leqslant N)$ in a formula F.

To return, note that the $p_i^{(n)}[x_i]$'s are formulae in the language $L_{\mathrm{B}} + \{\mathbf{u}\}$, and check that they indeed form an increasing chain $\mathbf{u} = p_i^{(0)}[x_i] \leqslant_{\Delta} p_i^{(1)}[x_i] \leqslant_{\Delta} \cdots \leqslant_{\Delta} p_i^{(n)}[x_i] \leqslant_{\Delta} \dots$ irrespective of the choice of Δ (this is proved by induction on n using the monotonicity of logical connectives[6]). We call each $p_i^{(n)}[x_i]$ $(n = 0, 1, \dots)$ a *generative chain for* p_i by Γ. Let us calculate an initial part of a generative chain. The chosen program, even the program below, is intended to define even numbers and $\Delta_{\mathrm{N}} = \{s(X) = s(Y) \rightarrow X = Y,\ s(X) \neq 0,\ s(\dots s(X)\dots) \neq X,\ (N = 0) \vee \exists X (N = s(X))\}$ is assumed as a base theory.

Example 3.1.

$$\mathrm{even}(N) \Leftrightarrow (N = 0) \vee \exists X (N = s(X) \wedge \neg \mathrm{even}(X))$$

$$\mathrm{even}^{(0)}[N] = \mathbf{u}$$

$$\begin{aligned}
\mathrm{even}^{(1)}[N] &= (N = 0) \vee \exists X (N = s(X) \wedge \neg \mathbf{u}) \\
&=_{\Delta_{\mathrm{N}}} (N = 0) \vee \mathbf{u} \\
\mathrm{even}^{(2)}[N] &= (N = 0) \vee \exists X (N = s(X) \wedge \neg \{(X = 0) \vee \exists X' (X = s(X') \wedge \neg \mathbf{u})\}) \\
&=_{\Delta_{\mathrm{N}}} (N = 0) \vee \exists X (N = s(s(X)) \wedge \mathbf{u}) \\
\mathrm{even}^{(3)}[N] &=_{\Delta_{\mathrm{N}}} (N = 0) \vee \exists X (N = s(X) \wedge \neg \{(X = 0) \vee \exists X' (X = s(s(X')) \wedge \mathbf{u})\}) \\
&=_{\Delta_{\mathrm{N}}} (N = 0) \vee (N = s(s(0)) \vee \exists X (N = s(s(X)) \wedge \mathbf{u}) \\
&\ \vdots
\end{aligned}$$

The substitution of generative chains in a formula yields another increasing chain. Let $G[x]$ be a formula in $L_{\mathrm{B}} + \{\mathbf{u}\} + \{p_1, \dots, p_N\}$. Define the *approximating chain* of G by Γ as

$$G_{\Gamma}^{(n)} = G[p_1^{(n)}, \dots, p_N^{(n)}], \quad n = 0, 1, \dots$$

[6] It is immediate from the definition of $\leqslant_{\Delta}$ that $F[x] \leqslant_{\Delta} G[x]$ implies $\neg F[x] \leqslant_{\Delta} \neg G[x]$ and $A \vee F[x] \leqslant_{\Delta} A \vee G[x]$, and that $F[x, y] \leqslant_{\Delta} G[x, y]$ implies $\exists x F[x, y] \leqslant_{\Delta} \exists x G[x, y]$.

It is straightforward to verify that $G_\Gamma^{(n)}$ $(n=0,1,\ldots)$ becomes an increasing chain w.r.t. any base theory Δ.

In addition to approximating chains, there is a more fundamental application of generative chains. It is an extension of a structure M for $L_{\mathrm{B}}+\{\mathbf{u}\}$ to an enriched structure M_Γ for $L_{\mathrm{B}}+\{\mathbf{u}\}+\{p_1,\ldots,p_N\}$. More precisely, let $\{p_i^{(n)}[x_i] \mid 1\leqslant i\leqslant N, n=0,1,\ldots\}$ be the generative chains by Γ. M_Γ has the same domain D as M and agrees with M on the interpretation of symbols of $L_{\mathrm{B}}+\{\mathbf{u}\}$. For each $p_i(\boldsymbol{a})$ $(1\leqslant i\leqslant N)$, where $\boldsymbol{a}\in D$, if $\mathrm{val}(p_i^{(n)}[\boldsymbol{a}],M)$ gets defined at some n, M_Γ takes that value as the value of $p_i(\boldsymbol{a})$. That is, for $v\in\{\mathbf{t},\mathbf{f}\}$ we define M_Γ by

$$\mathrm{val}(p_i(\boldsymbol{a}),M_\Gamma)=v \quad \text{iff} \quad \mathrm{val}(p_i^{(n)}[\boldsymbol{a}],M)=v \text{ for some } n.$$

If there is no such n, we put $\mathrm{val}(p_i(\boldsymbol{a}),M_\Gamma)=\mathbf{u}$. We specifically call M_Γ the *generative extension* of M by Γ. Or in other words, it is a structure generated by ω times iteration on M of T_Γ, a mapping over three-valued interpretations associated with the program Γ [16].

Generally speaking, for M arbitrary, one can hardly expect M_Γ to become a model of Γ. However, as we shall see in Section 3.4, there is a way to extend M to a larger structure M' for $L_{\mathrm{B}}+\{\mathbf{u}\}$ for which M'_Γ gives a model of Γ; hence, the closure ordinal of T_Γ does not exceed ω there. This should contrast with the two-valued case where it might go as high as Church–Kleene ω_1, the least nonconstructive recursive ordinal [3].

3.4. *Elementary extension*

Let M and M' be two structures for $L_{\mathrm{B}}+\{\mathbf{u}\}$ with domains D and D', respectively. We say that M' is an *elementary extension* of M if $D'\supseteq D$ and it holds that, for an arbitrary formula $F[x]$ in $L_{\mathrm{B}}+\{\mathbf{u}\}$,

$$\mathrm{val}(F[\boldsymbol{a}],M)=v \quad \text{iff} \quad \mathrm{val}(F[\boldsymbol{a}],M')=v \text{ for any } \boldsymbol{a}\in D \text{ and } v\in\{\mathbf{t},\mathbf{f},\mathbf{u}\}.$$

It is fortunate that ultraproduct construction carries over to three-valued logic. In particular, ultrapower construction [6] provides us with Lemma 3.2.

Lemma 3.2. *Let M be a structure for $L_{\mathrm{B}}+\{\mathbf{u}\}$ and Γ a program. Then there exists a three-valued elementary extension M' of M with domain D' such that, for a formula $G[x]$ in $L_{\mathrm{B}}+\{\mathbf{u}\}+\{p_1,\ldots,p_N\}$ and elements $\boldsymbol{a}$ in D',*

$$\mathrm{val}(G[\boldsymbol{a}],M'_\Gamma)=v \quad \textit{iff} \quad \mathrm{val}(G_\Gamma^{(n)}[\boldsymbol{a}],M')=v \textit{ for some } n,$$

where $v\in\{\mathbf{t},\mathbf{f}\}$, M'_Γ is the generative extension of M' by Γ and $G_\Gamma^{(n)}$ the approximating chain of G by Γ. Moreover, M'_Γ becomes a model of Γ.

Proof. Take an ω-incomplete filter F over an appropriate set I and construct an ultrapower $M'=M^I/F$ [6, Theorem 6.1.1]. M' then becomes an elementary extension of M which is ω_1-saturated w.r.t. true and false, and M'_Γ has the required property (see also [16]). □

Theorem 3.3 (Kunen [16, Theorem 6.3]). *Let Δ be a base theory in L_B. Also let $\Gamma = \{p_i(x_i) \Leftrightarrow F_i \mid 1 \leqslant i \leqslant N\}$ be a program and G a sentence in $L_B + \{p_1, \ldots, p_N\}$, respectively. Then*

$$\Delta \cup \Gamma \models_3 G \text{ iff } \Delta \models_3 G_\Gamma^{(n)} \text{ for some } n,$$

where $G_\Gamma^{(n)}$ $(n = 0, 1, \ldots)$ is the approximating chain of G by Γ.

Proof. The "if" part is straightforward. So, we prove the "only if" part. Let M be a structure for $L_B + \{\mathbf{u}\}$ which is a model of Δ. Consider its elementary extension M' and the generative extension M'_Γ of M' by Γ mentioned in Lemma 3.2. M'_Γ is a model of Γ and, hence, a model of G as well. Thanks to Lemma 3.2, we see that there exists such an n that $\mathrm{val}(G_\Gamma^{(n)}, M') = \mathbf{t}$. But M' being an elementary extension of M, we have $\mathrm{val}(G_\Gamma^{(n)}, M) = \mathbf{t}$ as well. Since M was chosen arbitrarily, we are done. □

This theorem teaches us how to compare the relative strength under Δ of two first-order programs, say Γ_1 and Γ_2, in three-valued logic. To do so, we have only to compare their generative chains. Let $\Gamma_1 = \{p_i(x_i) \Leftrightarrow F_{i,1} \mid 1 \leqslant i \leqslant N\}$ and $\Gamma_2 = \{p_i(x_i) \Leftrightarrow F_{i,2} \mid 1 \leqslant i \leqslant N\}$ be programs having the same user predicates and $p_{i,1}^{(n)}[x_i]$ and $p_{i,2}^{(n)}[x_i]$ $(1 \leqslant i \leqslant N,\ n = 0, 1, \ldots)$ their generative chains, respectively. Let $\{q_1, \ldots, q_M\}$ be a subset of $\{p_1, \ldots, p_N\}$. We say that Γ_1 and Γ_2 are *chain-equivalent w.r.t.* $\{q_1, \ldots, q_M\}$ under a base theory Δ if, for every j $(1 \leqslant j \leqslant M)$,

$$\forall n \exists m\ q_{j,1}^{(n)}[x_j] \leqslant_\Delta q_{j,2}^{(m)}[x_j] \quad \text{and} \quad \forall m \exists n q_{j,2}^{(m)}[x_j] \leqslant_\Delta q_{j,1}^{(n)}[x_j].$$

When $M = N$, i.e., $\{q_1, \ldots, q_M\} = \{p_1, \ldots, p_N\}$, we say that Γ_1 and Γ_2 are *chain-equivalent* under Δ.

Proposition 3.4. *Suppose Γ_1 and Γ_2 are chain-equivalent under a base theory Δ. For a sentence G in $L_B + \{p_1, \ldots, p_N\}$,*

$$\Delta \cup \Gamma_1 \models_3 G \text{ iff } \Delta \cup \Gamma_2 \models_3 G.$$

Proof. From the chain equivalence, we have $\forall n \exists m G_{\Gamma_1}^{(n)} \leqslant_\Delta G_{\Gamma_2}^{(m)}$ and $\forall m \exists n G_{\Gamma_2}^{(m)} \leqslant_\Delta G_{\Gamma_1}^{(n)}$ (this is proved by induction on the complexity of G). The rest follows from Theorem 3.3. □

Our equivalence-preserving unfold/fold transformation systems, which will be presented in Sections 4 and 5, respectively, are based solely on this proposition.

3.5. Relationship between two-valued and three-valued logic

We investigate the relationship between three-valued logic and two-valued logic for later use. First we introduce classes of consistent programs (in two-valued sense).

For a program $\Gamma = \{p_i(x_i) \Leftrightarrow F_i \mid 1 \leqslant i \leqslant N\}$, define the *signed dependency* among $\{p_1, \ldots, p_N\}$ [18, 26], denoted by $p >_+ q$ (*p depends on q positively*) and $p >_- q$ (*p depends on q negatively*), respectively, as the least relation satisfying

$$p >_+ q \text{ iff } p \gg_+ q, \text{ or for some } r,\ p \gg_+ r \text{ and } r >_+ q, \text{ or } p \gg_- r \text{ and } r >_- q,$$

$$p >_- q \text{ iff } p \gg_- q, \text{ or for some } r,\ p \gg_+ r \text{ and } r >_- q, \text{ or } p \gg_- r \text{ and } r >_+ q,$$

where $p \gg_+ q$ ($p \gg_- q$) is defined as

$p \gg_+ q$ ($p \gg_- q$) iff there is a predicate definition $p_i(x_i) \Leftrightarrow F_i$ in Γ such that $p = p_i$ and q occurs positively (negatively) in F_i.

A program is *call-consistent* [18, 26] if we never have $p >_- p$ for any p in the program, i.e. no predicate calls itself through an odd number of negative goals. Note that call-consistent programs allow recursion through an even number of negative goals (whereas *stratified* programs prohibit any recursion through negative goals [1]). A program is said to be *strict* [18] if for no two predicates p and q, we have $p >_+ q$ and $p >_- q$ at the same time. It is clear that strictness implies call-consistency. Thus, a program $\{a \Leftrightarrow \neg b, b \Leftrightarrow \neg a\}$ is strict (and, hence, call-consistent), while $\{a \Leftrightarrow b \vee \neg b, b \Leftrightarrow b\}$ is not strict (though call-consistent). It is proved that call-consistency guarantees the consistency in two-valued logic [18, 26].

We need one more definition. A program Γ is said to be *strict w.r.t. a goal G* if, for c, a newly introduced predicate symbol, a program $\Gamma \cup \{c \Leftrightarrow G\}$ has no predicate on which c depends both positively and negatively [18]. According to this definition, a strict program $\{a \Leftrightarrow \neg b, b \Leftrightarrow \neg a\}$ fails to be strict w.r.t. $b \vee \neg b$, for $\{a \Leftrightarrow \neg b, b \Leftrightarrow \neg a, c \Leftrightarrow b \vee \neg b\}$ is not strict. By the way, if a program is strict, it is so w.r.t. any literal as well. Now we are ready to state Theorem 3.5.

Theorem 3.5 (Kunen [18, Theorem 3.6]). *Let Δ be a base theory in L_B, $\Gamma = \{p_i(x_i) \Leftrightarrow F_i \mid 1 \leqslant i \leqslant N\}$ a call-consistent program, and G a sentence in $L_B + \{p_1, \ldots, p_N\}$. If Γ is strict w.r.t. G, we have*

$$\Delta \cup \Gamma \models_3 G \text{ iff } \Delta \cup \Gamma \models_2 G.$$

Proof. See [18]. □

Proposition 3.6. *Let Δ be a base theory in L_B. Also let Γ_1 and Γ_2 be call-consistent programs in $L_B + \{p_1, \ldots, p_N\}$. If Γ_1 and Γ_2 are chain-equivalent under Δ and strict w.r.t. a sentence G in $L_B + \{p_1, \ldots, p_N\}$, we have*

$$\Delta \cup \Gamma_1 \vdash G \text{ iff } \Delta \cup \Gamma_2 \vdash G.$$

Proof.

$$\begin{aligned}
\Delta\cup\Gamma_1\vdash G \ &\text{iff}\ \Delta\cup\Gamma_1\models_2 G \quad &&\text{(by the completeness of first-order logic)}\\
&\text{iff}\ \Delta\cup\Gamma_1\models_3 G \quad &&\text{(by the assumptions on } \Gamma_1 \text{ and } G\text{, Theorem 3.5)}\\
&\text{iff}\ \Delta\cup\Gamma_2\models_3 G \quad &&\text{(by the chain equivalence, Proposition 3.4)}\\
&\text{iff}\ \Delta\cup\Gamma_2\models_2 G \quad &&\text{(by the assumptions on } \Gamma_2 \text{ and } G\text{, Theorem 3.5)}\\
&\text{iff}\ \Delta\cup\Gamma_2\vdash G. \quad \square
\end{aligned}$$

We end this section with a proposition that helps us check folding conditions (see Section 4) in terms of two-valued logic. Here we need a notation which makes a distinction between positive and negative occurrences [6] of the symbol **u** in a formula $F[x]$ in $L_B+\{\mathbf{u}\}$. So, write $F[x]=F[\mathbf{u}^+,\mathbf{u}^-,x]$ and let $\mathbf{u}^+$ ($\mathbf{u}^-$) refer to all positive (negative) occurrences of **u** in $F[x]$. Define $F[\mathbf{f}/\mathbf{u}^+,\mathbf{t}/\mathbf{u}^-,x]$ to be a formula obtained from $F[x]$ by substituting **f** (**t**) for all positive occurrences (negative occurrences) of **u** in $F[x]$. Take $R[\mathbf{u}^+,\mathbf{u}^-,x]=(\mathbf{u}^1\rightarrow(x\neq 0\vee \mathbf{u}^2\vee\neg\mathbf{u}^3))$ as an example. Then $\mathbf{u}^+$ refers to $\mathbf{u}^2$ and $\mathbf{u}^-$ to $\mathbf{u}^1$ and $\mathbf{u}^3$, and $R[\mathbf{f}/\mathbf{u}^+,\mathbf{t}/\mathbf{u}^-,x]=(\mathbf{t}\rightarrow(x\neq 0\vee\mathbf{f}\vee\neg\mathbf{t}))$.

Proposition 3.7. *Let Δ be a base theory in L_B, $F[x]$ and $G[x]$ formulae in $L_B+\{\mathbf{u}\}$, respectively.*

$$F[x]=_\Delta\mathbf{u}\ \textit{iff}\ \Delta\vdash\forall x\neg F[\mathbf{f}/\mathbf{u}^+,\mathbf{t}/\mathbf{u}^-,x]\ \textit{and}\ \Delta\vdash\forall xF[\mathbf{t}/\mathbf{u}^+,\mathbf{f}/\mathbf{u}^-,x],$$

$$F[x]\leqslant_\Delta G[x]\ \textit{iff}\ \Delta\vdash\forall x(F[\mathbf{f}/\mathbf{u}^+,\mathbf{t}/\mathbf{u}^-,x]\rightarrow G[\mathbf{f}/\mathbf{u}^+,\mathbf{t}/\mathbf{u}^-,x])\ \textit{and}$$

$$\Delta\vdash\forall x(\neg F[\mathbf{t}/\mathbf{u}^+,\mathbf{f}/\mathbf{u}^-,x]\rightarrow\neg G[\mathbf{t}/\mathbf{u}^+,\mathbf{f}/\mathbf{u}^-,x]).$$

Proof. Let M be a structure for $L_B+\{\mathbf{u}\}$ satisfying Δ with domain D and $F[x]$ a formula in $L_B+\{\mathbf{u}\}$. We first prove that for any elements $\boldsymbol{a}$ in D

(1) $\mathrm{val}(F[\boldsymbol{a}],M)=\mathbf{t}$ iff $\mathrm{val}(F[\mathbf{f}/\mathbf{u}^+,\mathbf{t}/\mathbf{u}^-,\boldsymbol{a}],M)=\mathbf{t}$,

(2) $\mathrm{val}(F[\boldsymbol{a}],M)=\mathbf{f}$ iff $\mathrm{val}(F[\mathbf{t}/\mathbf{u}^+,\mathbf{f}/\mathbf{u}^-,\boldsymbol{a}],M)=\mathbf{f}$.

The proof is by induction on the complexity of F. Suppose F is an atom. If F is **u**, both sides of (1) and (2) are false ($F[\mathbf{f}/\mathbf{u}^+,\mathbf{t}/\mathbf{u}^-,\boldsymbol{a}]=\mathbf{f}$ and $F[\mathbf{t}/\mathbf{u}^+,\mathbf{f}/\mathbf{u}^-,\boldsymbol{a}]=\mathbf{t}$). Otherwise, they are identical. For a composite F, we prove only the case of negation. Suppose $F=\neg G$ and the claim holds for G. Then we have

$$\begin{aligned}
\mathrm{val}(F[\boldsymbol{a}],M)=\mathbf{t}\ &\text{iff}\ \mathrm{val}(G[\boldsymbol{a}],M)=\mathbf{f},\\
&\text{iff}\ \mathrm{val}(G[\mathbf{t}/\mathbf{u}^+,\mathbf{f}/\mathbf{u}^-,\boldsymbol{a}],M)=\mathbf{f},\\
&\text{iff}\ \mathrm{val}(\neg G[\mathbf{t}/\mathbf{u}^+,\mathbf{f}/\mathbf{u}^-,\boldsymbol{a}],M)=\mathbf{t},\\
&\text{iff}\ \mathrm{val}(F[\mathbf{f}/\mathbf{u}^+,\mathbf{t}/\mathbf{u}^-,\boldsymbol{a}],M)=\mathbf{t}
\end{aligned}$$

and, similarly, for $\mathrm{val}(F[a], M)=\mathbf{f}$. So, we conclude (1) and (2). Now, for formulae $F[x]$ and $G[x]$ in $L_{\mathrm{B}}+\{\mathbf{u}\}$, we see that

$F[x] \leqslant_{\Delta} G[x]$ iff for any structure M for $L_{\mathrm{B}}+\{\mathbf{u}\}$ which is a model of Δ and for any a in its domain,

$$\mathrm{val}(F[a], M)=\mathbf{t} \text{ implies } \mathrm{val}(G[a], M)=\mathbf{t} \text{ and}$$

$$\mathrm{val}(F[a], M)=\mathbf{f} \text{ implies } \mathrm{val}(G[a], M)=\mathbf{f},$$

or, equivalently, using (1) and (2),

$$\mathrm{val}(F[\mathbf{f}/\mathbf{u}^+, \mathbf{t}/\mathbf{u}^-, a], M)=\mathbf{t} \text{ implies}$$

$$\mathrm{val}(G[\mathbf{f}/\mathbf{u}^+, \mathbf{t}/\mathbf{u}^-, a], M)=\mathbf{t} \text{ and}$$

$$\mathrm{val}(F[\mathbf{t}/\mathbf{u}^+, \mathbf{f}/\mathbf{u}^-, a], M)=\mathbf{f} \text{ implies}$$

$$\mathrm{val}(G[\mathbf{t}/\mathbf{u}^+, \mathbf{f}/\mathbf{u}^-, a], M)=\mathbf{f}.$$

Recalling that $F[\mathbf{f}/\mathbf{u}^+, \mathbf{t}/\mathbf{u}^-, x]$, $G[\mathbf{f}/\mathbf{u}^+, \mathbf{t}/\mathbf{u}^-, x]$, $F[\mathbf{t}/\mathbf{u}^+, \mathbf{f}/\mathbf{u}^-, x]$, and $G[\mathbf{t}/\mathbf{u}^+, \mathbf{f}/\mathbf{u}^-, x]$ are all two-valued formulae, we have

$F[x] \leqslant_{\Delta} G[x]$ iff $\forall x(F[\mathbf{f}/\mathbf{u}^+, \mathbf{t}/\mathbf{u}^-, x] \rightarrow G[\mathbf{f}/\mathbf{u}^+, \mathbf{t}/\mathbf{u}^-, x])$ and

$$\forall x(\neg F[\mathbf{t}/\mathbf{u}^+, \mathbf{f}/\mathbf{u}^-, x] \rightarrow \neg G[\mathbf{t}/\mathbf{u}^+, \mathbf{f}/\mathbf{u}^-, x])$$

are true for every two-valued model of Δ.

The rest follows from the completeness of first-order logic. The remaining case of $F[x]=_{\Delta}\mathbf{u}$ is obtained similarly from $\mathrm{val}(F[a], M)=\mathbf{u}$ iff $\mathrm{val}(F[\mathbf{f}/\mathbf{u}^+, \mathbf{t}/\mathbf{u}^-, a], M)=\mathbf{f}$ and $\mathrm{val}(F[\mathbf{t}/\mathbf{u}^+, \mathbf{f}/\mathbf{u}^-, a], M)=\mathbf{t}$. □

4. Basic unfold/fold transformation system

Now we present a first unfold/fold transformation system for first-order programs. It preserves chain equivalence and, hence, by Proposition 3.4, is equivalence-preserving in the sense of three-valued logic. We call it the basic system here. The basic system is applicable, not only to the three-valued case but also to the two-valued case by virtue of Proposition 3.6, thereby giving us a transformation system which preserves the proof-theoretic strength (in the usual sense) of call-consistent programs. In particular, it preserves the set of literals provable from strict programs. An extended transformation system will be presented in Section 5.

4.1. Transformation rules

The basic system has three rules: the unfolding rule, the folding rule and the replacement rule. The unfolding rule labels user predicate occurrences with a label u whereas the folding rule erases it. A folding operation is allowed only when every user predicate in the folded formula is labeled u. Hence, for notational convenience, we introduce a notation $F[\boldsymbol{p},\boldsymbol{p}^{\mathrm{u}}]$ for a formula F to make it clear that F's user predicates are among the nonlabeled $\boldsymbol{p}=p_1,\ldots,p_N$ and those labeled $\boldsymbol{p}^{\mathrm{u}}=p_1^{\mathrm{u}},\ldots,p_N^{\mathrm{u}}$. $F[\boldsymbol{p}^{\mathrm{u}}]$, therefore, indicates that all of F's user predicates, if any, are labeled u. We also use $F[\mathbf{u}]$ to denote F in which every atom containing a user predicate, be it labeled or not, is replaced by **u**.

Let Δ be a base theory in L_{B} and $\Gamma_0=\{p_i(x_i)\Leftrightarrow F_{i,0} \mid 1\leqslant i\leqslant N\}$ an initial first-order program in which no user predicate symbol is labeled u.

Suppose Γ_0 has been transformed to $\Gamma_k=\{p_i(x_i)\Leftrightarrow F_{i,k} \mid 1\leqslant i\leqslant N\}$. Choose a predicate definition $h(x)\Leftrightarrow B_k$ from Γ_k and transform B_k into B_{k+1} by applying one of the following rules below, and put

$$\Gamma_{k+1}=(\Gamma_k\setminus\{h(x)\Leftrightarrow B_k\})\cup\{h(x)\Leftrightarrow B_{k+1}\}.$$

⟨*unfolding*⟩

Select an atom $d(t)$ from the body B_k which contains a user predicate d. Whether d is labeled u or not is irrelevant. Let $d(y)\Leftrightarrow D_0[y]$ be a predicate definition about d in Γ_0 and $D_0[\boldsymbol{p}^{\mathrm{u}},t]$ the formula $D_0[t/y]$ such that all user predicates are labeled u. Substitute $D_0[\boldsymbol{p}^{\mathrm{u}},t]$ for $d(t)$ in B_k. Take the result as B_{k+1}. B_{k+1} is written as

$$B_{k+1}=B_k[D_0[\boldsymbol{p}^{\mathrm{u}},t]/d(t)].$$

⟨*folding*⟩

Folding operation requires two conditions, F1 and F2 below, to be satisfied. Select a subformula $D_0[t]$ from B_k for which there exists a predicate definition $d(y)\Leftrightarrow D_0[y]$ in Γ_0. If

(F1) every user predicate in $D_0[t]$ is labeled u,

substitute $d(t)$ for $D_0[t]$ in B_k. Do not mark $d(t)$ with u. Write the result as $B_k[d(t)/D_0[\boldsymbol{p}^{\mathrm{u}},t]]$. Let $h(x)\Leftrightarrow B_0$ be a predicate definition about h in Γ_0. Put

$$B_{k+1}=B_k[d(t)/D_0[\boldsymbol{p}^{\mathrm{u}},t]]$$

provided

(F2) $B_0[\mathbf{u}]\leqslant_\Delta B_{k+1}[\mathbf{u}]$.

We call $h(x)\Leftrightarrow B_k$ the folded definition, $d(y)\Leftrightarrow D_0[y]$ the folding definition, respectively.

⟨*replacement*⟩

Take as B_{k+1} any formula such that

$$B_{k+1}=_\Delta B_k$$

holds even when we consider the p_i's and the p_i^{u}'s appearing in $B_{k+1}=_\Delta B_k$ as independent predicate symbols.

The replacement rule is primarily intended for simplification of transformed programs. To apply it, we must treat the p_i's and the p_i^{u}'s as independent predicate symbols. It means that, while we may shorten $p_1(x) \vee p_1(x) \vee p_1^{\mathrm{u}}(x)$ to $p_1(x) \vee p_1^{\mathrm{u}}(x)$, we are not allowed to further replace the latter with $p_1(x)$ or $p_1^{\mathrm{u}}(x)$. Furthermore, we have to consider the equivalence in the light of three-valued logic. Actually, however, almost all basic logical equivalences familiar in two-valued logic, such as the associativity and commutativity of logical connectives, are available. The following is a (short) list of useful equivalence schemata: $A \wedge A \Leftrightarrow A$, $A \vee B \Leftrightarrow B \vee A$, $A \wedge (B \vee C) \Leftrightarrow (A \wedge B) \vee (A \wedge C)$, $(A \vee B \rightarrow C) \Leftrightarrow (A \rightarrow C) \wedge (B \rightarrow C)$, $(A \wedge B \rightarrow C) \Leftrightarrow (A \rightarrow (B \rightarrow C))$, $\exists x(x = t \wedge F[x]) \Leftrightarrow F[t]$ provided t is free for x in F. Note that we cannot replace $\neg A \vee A$ with **t**, nor can we replace $\neg A \wedge A$ with **f**, except for the two-valued case such as $X = 0 \vee X \neq 0$.

In practice, replacement peculiar to a specific Δ is more interesting and more important. For example, under Clark's equational theory [16, 17], we may use $(s = t) =_{\Delta} \mathbf{f}$ if s and t are not unifiable and so on.

4.2. *Folding conditions*

For folding to be validated, the folding conditions, F1 and F2, must be met. F1 is easy to check. So, we comment on F2. Let $h(x) \Leftrightarrow B_k$ be a folded definition and $h(x) \Leftrightarrow B_0$ a predicate definition about h in Γ_0.

First keep in mind that, despite the somewhat complicated appearance of F2, we can ignore it if B_0 is built up from user predicates and logical connectives, as in $\exists x_3(p(x_1, x_3) \wedge \neg q(x_2, x_3))$. This is so, because in this case, $B_0[\mathbf{u}] =_{\Delta} \mathbf{u}$ holds for any base theory Δ, as seen from $\exists x_3(\mathbf{u}/p(x_1, x_3) \wedge \neg \mathbf{u}/q(x_2, x_3)) \Leftrightarrow \exists x_3(\mathbf{u} \wedge \neg \mathbf{u}) \Leftrightarrow \mathbf{u}$.

This case is thought to correspond directly to the folding of stratified programs proposed by Seki [29]. More precisely, in his system, folding takes place as follows. There are two clauses $A \leftarrow K, L$ and $B \leftarrow K'$, where $K = K'\theta$ holds for some substitution θ. $B \leftarrow K'$ is a unique definite clause in the initial program defining a new predicate. Then if A contains an "old predicate", or else A contains a "new predicate" and no atom in K is "inherited" from the initial program, $A \leftarrow K, L$ is folded into $A \leftarrow B\theta, L$.

The first type does not happen, as every user predicate is a new predicate in the current system (the extended system in Section 5 distinguishes between old predicates and new predicates). For the second case, that no atom in K is "inherited" means that every atom in K is labeled u in our system, thereby satisfying F1. F2 being automatically satisfied for the above-mentioned reason, we can conclude that folding of the second type is legitimate in our basic system.

To return, even if B_0 contains some primitive predicates, the problem of checking F2 is reducible to the two-valued case by Proposition 3.7. If, accordingly, Δ is a decidable theory, F2 becomes mechanically checkable. If, for example, our universe of discourse is the Herbrand universe and, hence, is completely axiomatizable by Clark's equational theory, a checking algorithm is available [8, 17].

For other cases, we have to take a case-by-case approach. Speaking of Example 1.1, F2 is satisfied because the calculation of $B_0[\mathbf{u}]$ comes out to be $\mathbf{u}$ as follows:

$$\begin{aligned} B_0[\mathbf{u}] &= \forall Y(\mathbf{u}/\mathrm{mem}(Y, L) \rightarrow Y \neq 0) \\ &= \forall Y(\mathbf{u} \rightarrow Y \neq 0) \Leftrightarrow \neg \mathbf{u} \vee \forall Y(Y \neq 0) \Leftrightarrow \mathbf{u} \vee \mathbf{f} \Leftrightarrow \mathbf{u}. \end{aligned}$$

4.3. Correctness

The basic transformation system is equivalence-preserving in the following sense.

Theorem 4.1 (Equivalence property). *Let Δ be a base theory in L_B and $\Gamma_0 = \{p_i(x_i) \Leftrightarrow F_{i,0} \mid 1 \leqslant i \leqslant N\}$ a program, respectively. Suppose Γ_0 has been transformed to $\Gamma_k = \{p_i(x_i) \Leftrightarrow F_{i,k} \mid 1 \leqslant i \leqslant N\}$. Then Γ_0 and Γ_k are chain-equivalent under Δ, and for a sentence G in $L_B + \{p_1, \ldots, p_N\}$, we have*

$$\Delta \cup \Gamma_0 \models_3 G \quad \textit{iff} \quad \Delta \cup \Gamma_k \models_3 G.$$

If Γ_0 and Γ_k are both call-consistent and strict w.r.t. G,[7]

$$\Delta \cup \Gamma_0 \vdash G \quad \textit{iff} \quad \Delta \cup \Gamma_k \vdash G.$$

Proof. Suppose we have a transformation sequence $\Gamma_0, \Gamma_1, \ldots$ by the basic system. Let $p_{i,k}^{(n)}$ $(1 \leqslant i \leqslant N,\ k, n = 0, 1, \ldots)$ be the generative chain for p_i by $\Gamma_k = \{p_i(x_i) \Leftrightarrow F_{i,k} \mid 1 \leqslant i \leqslant N\}$. To prove the theorem, we prove Lemmas 4.2 and 4.4. Since Lemma 4.2 implies Corollary 4.3, which says that $p_{i,0}^{(n)} \leqslant_\Delta p_{i,k}^{(n)}$ holds for every i $(1 \leqslant i \leqslant N)$ and k, n $(\geqslant 0)$, and since Lemma 4.4 asserts that $\forall m \exists n\, p_{i,k}^{(m)} \leqslant_\Delta p_{i,0}^{(n)}$ holds for every i $(1 \leqslant i \leqslant N)$ and k, n $(\geqslant 0)$, we conclude that every transformed program is chain-equivalent to Γ_0 under Δ. The rest follows from Propositions 3.4 and 3.6. $\square$

Now we prove Lemmas 4.2 and 4.4. In what follows, $F[\mathbf{p}_k^{(m)}, \mathbf{p}_k^{(n)\mathrm{u}}]$ denotes $F[p_{1,k}^{(m)}, \ldots, p_{N,k}^{(m)}, p_{1,k}^{(n)\mathrm{u}}, \ldots, p_{N,k}^{(n)\mathrm{u}}]$ $(k, m, n \geqslant 0)$, a formula obtained from F by replacing every $p_i(t)$ in F with $p_{i,k}^{(m)}[t]$ and every $p_i^{\mathrm{u}}(t')$ with $p_{i,k}^{(n)}[t']$ $(1 \leqslant i \leqslant N)$.

Lemma 4.2. *The following invariant holds for every k.*

$\langle$*INVARIANT*-1$\rangle$

For every i $(1 \leqslant i \leqslant N)$ and n $(\geqslant 0)$,

$$F_{i,0}[\mathbf{p}_0^{(n)}] \leqslant_\Delta F_{i,k}[\mathbf{p}_0^{(n)}, \mathbf{p}_0^{(n)\mathrm{u}}] \quad \textit{and} \quad F_{i,0}[\mathbf{p}_0^{(n+1)}] \leqslant_\Delta F_{i,k}[\mathbf{p}_0^{n+1}, \mathbf{p}_0^{(n)\mathrm{u}}].$$

Proof. By induction on k. The case for $k = 0$ is obvious. So, suppose INVARIANT-1 holds for k and Γ_{k+1} is obtained from Γ_k by transforming a predicate definition

[7] Note that the replacement rule might destroy the call-consistency (strictness) of programs. For example, it can transform a call-consistent program $\{a \Leftrightarrow a\}$ into a non-call-consistent one $\{a \Leftrightarrow a \vee (\mathbf{f} \wedge \neg a)\}$.

$h(x) \Leftrightarrow B_k$ in Γ_k into $h(x) \Leftrightarrow B_{k+1}$. Let $h(x) \Leftrightarrow B_0$ be a predicate definition about h in Γ_0. We have to prove that, for all n,

$$B_0[p_0^{(n)}] \leqslant_\Delta B_{k+1}[p_0^{(n)}, p_0^{(n)\mathrm{u}}] \quad \text{and} \quad B_0[p_0^{(n+1)}] \leqslant_\Delta B_{k+1}[p_0^{(n+1)}, p_0^{(n)\mathrm{u}}].$$

There are three cases.

Case 1: unfolding. B_{k+1} is obtained by unfolding an atom $d(t)$ in the body B_k using $d(y) \Leftrightarrow D_0[y]$ in Γ_0. We first treat the case of $d(t)$ not labeled u. Write $B_k = B_k[d(t), p, p^{\mathrm{u}}]$ and $B_{k+1}[p, p^{\mathrm{u}}] = B_k[D_0[p^{\mathrm{u}}, t]/d(t), p, p^{\mathrm{u}}]$. Let $d_0^{(n)}$ $(n=0,1,\ldots)$ be the generative chain for d by Γ_0. Then, for all n,

$$\begin{aligned} B_{k+1}[p_0^{(n)}, p_0^{(n)\mathrm{u}}] &= B_k[D_0[p_0^{(n)\mathrm{u}}, t]/d(t), p_0^{(n)}, p_0^{(n)\mathrm{u}}] \\ &= B_k[d_0^{(n+1)}, p_0^{(n)}, p_0^{(n)\mathrm{u}}] \\ &{}_\Delta\!\geqslant B_k[d_0^{(n)}, p_0^{(n)}, p_0^{(n)\mathrm{u}}] \quad \text{(because } d_0^{(n+1)} {}_\Delta\!\geqslant d_0^{(n)}\text{)} \\ &{}_\Delta\!\geqslant B_0[p_0^{(n)}] \quad \text{(by INVARIANT-1 at } k\text{)}, \\ B_{k+1}[p_0^{(n+1)}, p_0^{(n)\mathrm{u}}] &= B_k[D_0[p_0^{(n)\mathrm{u}}, t]/d(t), p_0^{(n+1)}, p_0^{(n)\mathrm{u}}] \\ &= B_k[d_0^{(n+1)}, p_0^{(n+1)}, p_0^{(n)\mathrm{u}}] \\ &{}_\Delta\!\geqslant B_0[p_0^{(n+1)}] \quad \text{(by INVARIANT-1 at } k\text{)}. \end{aligned}$$

When $d(t)$ is labeled u, we may write $B_k = B_k[d^{\mathrm{u}}(t), p, p^{\mathrm{u}}]$ and $B_{k+1}[p, p^{\mathrm{u}}] = B_k[D_0[p^{\mathrm{u}}, t]/d^{\mathrm{u}}(t), p, p^{\mathrm{u}}]$. $B_{k+1}[p_0^{(n)}, p_0^{(n)\mathrm{u}}] {}_\Delta\!\geqslant B_k[p_0^{(n)}, p_0^{(n)\mathrm{u}}]$ is proved similarly to the nonlabeled case, and we also have, for all n,

$$\begin{aligned} B_{k+1}[p_0^{(n+1)}, p_0^{(n)\mathrm{u}}] &= B_k[D_0[p_0^{(n)\mathrm{u}}, t]/d^{\mathrm{u}}(t), p_0^{(n+1)}, p_0^{(n)\mathrm{u}}] \\ &= B_k[d_0^{(n+1)\mathrm{u}}, p_0^{(n+1)}, p_0^{(n)\mathrm{u}}] \\ &{}_\Delta\!\geqslant B_k[d_0^{(n)\mathrm{u}}, p_0^{(n+1)}, p_0^{(n)\mathrm{u}}] \quad \text{(because } d_0^{(n+1)} {}_\Delta\!\geqslant d_0^{(n)}\text{)} \\ &{}_\Delta\!\geqslant B_0[p_0^{(n+1)}] \quad \text{(by INVARIANT-1 at } k\text{)}. \end{aligned}$$

Case 2: folding. B_{k+1} is obtained by folding a subformula $D_0[t]$ of B_k into $d(t)$ using $d(y) \Leftrightarrow D_0[y]$ in Γ_0. Write $B_k = B_k[D_0[p^{\mathrm{u}}, t], p, p^{\mathrm{u}}]$ and $B_{k+1} = B_k[d(t)/D_0[p^{\mathrm{u}}, t], p, p^{\mathrm{u}}]$. Let $d_0^{(n)}$ $(n \geqslant 0)$ be the generative chain for d by Γ_0. First we have, for all n,

$$\begin{aligned} B_{k+1}[p_0^{(n+1)}, p_0^{(n)\mathrm{u}}] &= B_k[d_0^{(n+1)}(t)/D_0[p^{\mathrm{u}}, t], p_0^{(n+1)}, p_0^{(n)\mathrm{u}}] \\ &= B_k[D_0[p_0^{(n)\mathrm{u}}, t], p_0^{(n+1)}, p_0^{(n)\mathrm{u}}] \\ &= B_k[p_0^{(n+1)}, p_0^{(n)\mathrm{u}}] \\ &{}_\Delta\!\geqslant B_0[p_0^{(n+1)}] \quad \text{(by INVARIANT-1 at } k\text{)}. \end{aligned}$$

Hence,

$$\text{(A1)} \quad B_{k+1}[p_0^{(n+1)}, p_0^{(n+1)\mathrm{u}}] {}_\Delta\!\geqslant B_{k+1}[p_0^{(n+1)}, p_0^{(n)\mathrm{u}}] {}_\Delta\!\geqslant B_0[p_0^{(n+1)}]$$

holds for all n. On the other hand, folding condition F2 ensures that

(A2) $B_{k+1}[\mathbf{u}, \mathbf{u}^{\mathrm{u}}] \geqslant_{\Delta} B_0[\mathbf{u}]$.

Putting A1 and A2 together, we obtain

$$B_{k+1}[\boldsymbol{p}_0^{(n)}, \boldsymbol{p}_0^{(n)\mathrm{u}}] \geqslant_{\Delta} B_0[\boldsymbol{p}_0^{(n)}, \boldsymbol{p}_0^{(n)\mathrm{u}}] \quad \text{for all } n.$$

Case 3: replacement. $h(x) \Leftrightarrow B_k$ in Γ_k is replaced with $h(x) \Leftrightarrow B_{k+1}$ such that

$$B_{k+1} =_{\Delta} B_k,$$

where the p_i's and the p_i^{u}'s are regarded as independent predicate symbols. Apparently, the invariant is preserved, because the above condition ensures that

$$B_{k+1}[\boldsymbol{p}_0^{(n)}, \boldsymbol{p}_0^{(m)\mathrm{u}}] =_{\Delta} B_k[\boldsymbol{p}_0^{(n)}, \boldsymbol{p}_0^{(m)\mathrm{u}}]$$

for every n and m. □

Corollary 4.3. *The following holds for every k.*

$$\forall i\ (1 \leqslant i \leqslant N),\ \forall n (\geqslant 0) \quad p_{i,0}^{(n)} \leqslant_{\Delta} p_{i,k}^{(n)}.$$

Proof. By induction on n. For each i $(1 \leqslant i \leqslant N)$,

$$p_{i,0}^{(0)} = \mathbf{u} \leqslant_{\Delta} p_{i,k}^{(0)}$$

and

$$\begin{aligned} p_{i,0}^{(n+1)} &= F_{i,0}[\boldsymbol{p}_0^{(n)}, \boldsymbol{p}_0^{(n)\mathrm{u}}] \\ &\leqslant_{\Delta} F_{i,k}[\boldsymbol{p}_0^{(n)}, \boldsymbol{p}_0^{(n)\mathrm{u}}] \quad \text{(by INVARIANT-1 at } k\text{)} \\ &\leqslant_{\Delta} F_{i,k}[\boldsymbol{p}_k^{(n)}, \boldsymbol{p}_k^{(n)\mathrm{u}}] \quad \text{(by the inductive assumption)} \\ &= p_{i,k}^{(n+1)}. \quad \square \end{aligned}$$

Lemma 4.4. *The following invariant holds for every k.*

⟨INVARIANT-2⟩

For every i $(1 \leqslant i \leqslant N)$, $\forall m \exists n\ p_{i,k}^{(m)} \leqslant_{\Delta} p_{i,0}^{(n)}$.

Proof. By induction on k. The case of $k=0$ is obvious. Assuming the invariant at k, we prove $\forall m \exists n\ p_{i,k+1}^{(m)} \leqslant_{\Delta} p_{i,k}^{(n)}$ for every i $(1 \leqslant i \leqslant N)$ by induction on m. Since $\leqslant_{\Delta}$ is transitive, this establishes the invariant at $k+1$.

Suppose Γ_{k+1} is obtained from Γ_k by transforming a predicate definition $h(x) \Leftrightarrow B_k$ in Γ_k into $h(x) \Leftrightarrow B_{k+1}$. Let $h_k^{(n)}$ and $h_{k+1}^{(n)}$ $(n = 0, 1, \ldots)$ be the generative chains for h by Γ_k and Γ_{k+1}, respectively. Also let $d_0^{(n)}$, $d_k^{(n)}$ and $d_{k+1}^{(n)}$ $(n = 0, 1, \ldots)$ be the generative chains for d by Γ_0, Γ_k and Γ_{k+1}, respectively. Now we assume $\exists n\ p_{i,k+1}^{(m)} \leqslant_{\Delta} p_{i,k}^{(n)}$ (true for

$m=0$) and prove $\exists n\ p_{i,k+1}^{(m+1)} \leqslant_\Delta p_{i,k}^{(n)}$. We treat, however, the most complicated case, that of $p_i = h$ (the remaining cases are easy), and prove $\exists n\ h_{k+1}^{(m+1)} \leqslant_\Delta h_k^{(n)}$.

Since the distinction between labeled and nonlabeled predicates turns out to be irrelevant in the following proof, we lump them together, and for a formula F, we use $F[\boldsymbol{p}_k^{(m)}]$ to denote $F[p_{1,k}^{(m)}, \ldots, p_{N,k}^{(m)}]$, the one obtained from F by replacing every $p_i(t)$ $(1 \leqslant i \leqslant N)$, be it labeled or not, with $p_{i,k}^{(m)}[t]$. There are three cases.

Case 1: unfolding. B_{k+1} is obtained from B_k by unfolding an atom $d(t)$ in B_k using $d(y) \Leftrightarrow D_0[y]$ in Γ_0. Write $B_k = B_k[d(t), \boldsymbol{p}]$ and $B_{k+1}[\boldsymbol{p}] = B_k[D_0[\boldsymbol{p}, t]/d(t), \boldsymbol{p}]$, respectively. First of all,

$$\begin{aligned} D_0[\boldsymbol{p}_{k+1}^{(m)}, t] &\leqslant_\Delta D_0[\boldsymbol{p}_k^{(n)}, t] && \text{for some } n \text{ by the inductive assumption} \\ &\leqslant_\Delta D_0[\boldsymbol{p}_0^{(n')}, t] && \text{for some } n' \text{ by INVARIANT-2 at } k \\ &= d_0^{(n'+1)}[t] \\ &\leqslant_\Delta d_k^{(n'+1)}[t] && \text{by Corollary 4.3} \end{aligned}$$

Thus,

$$\begin{aligned} h_{k+1}^{(m+1)} &= B_{k+1}[\boldsymbol{p}_{k+1}^{(m)}] \\ &= B_k[D_0[\boldsymbol{p}_{k+1}^{(m)}, t]/d(t), \boldsymbol{p}_{k+1}^{(m)}] \\ &\leqslant_\Delta B_k[d_k^{(n'+1)}[t]/d(t), \boldsymbol{p}_{k+1}^{(m)}] && \text{for some } n' \text{ by the above result} \\ &\leqslant_\Delta B_k[d_k^{(n'')}[t]/d(t), \boldsymbol{p}_k^{(n'')}] && \text{for some } n'' \ (n'' > n', m) \text{ by the inductive assumption} \\ &= h_k^{(n''+1)}. \end{aligned}$$

Case 2: folding. B_{k+1} is obtained by folding a subformula $D_0[t]$ of B_k into $d(t)$. Let $d(y) \Leftrightarrow D_0[y]$ be the folding definition in Γ_0. Write $B_k = B_k[D_0[t], \boldsymbol{p}]$ and $B_{k+1}[\boldsymbol{p}] = B_k[d(t)/D_0[t], \boldsymbol{p}]$.

$$\begin{aligned} h_{k+1}^{(m+1)} &= B_{k+1}[\boldsymbol{p}_{k+1}^{(m)}] \\ &= B_k[d_{k+1}^{(m)}[t]/D_0[t], \boldsymbol{p}_{k+1}^{(m)}] \\ &\leqslant_\Delta B_k[d_k^{(n)}[t]/D_0[t], \boldsymbol{p}_k^{(n)}] && \text{for some } n \text{ by the inductive assumption} \\ &\leqslant_\Delta B_k[d_0^{(n')}[t]/D_0[t], \boldsymbol{p}_0^{(n')}] && \text{for some } n' \text{ by INVARIANT-2 at } k \\ &\leqslant_\Delta B_k[d_0^{(n'+1)}[t]/D_0[t], \boldsymbol{p}_0^{(n')}] \\ &\leqslant_\Delta B_k[d_k^{(n'+1)}[t]/D_0[t], \boldsymbol{p}_k^{(n')}] && \text{by Corollary 4.3} \\ &= B_k[\boldsymbol{p}_k^{(n')}] = h_k^{(n'+1)}. \end{aligned}$$

Case 3: replacement. Γ_{k+1} is obtained from Γ_k by the replacement rule. But this case is obvious. □

5. Extended transformation system

Take up the even program in Example 3.1 again, together with $\Delta_N = \{s(X)=s(Y) \to X=Y,\ s(X) \neq 0,\ s(\ldots s(X)\ldots) \neq X,\ (N=0) \vee \exists X(N=s(X))\}$. Consider its unfold/fold transformation under Δ_N.

Example 5.1

Γ_0: $\text{even}(N) \Leftrightarrow N=0 \vee \exists X(N=s(X) \wedge \neg\text{even}(X))$ /fold $\neg\text{even}(N)$ by $\text{odd}(X)$
$\text{odd}(N) \Leftrightarrow \neg\text{even}(N)$

Γ_1: $\text{even}(N) \Leftrightarrow N=0 \vee \exists X(N=s(X) \wedge \text{odd}(X))$ /unfold $\neg\text{even}(N)$
$\text{odd}(N) \Leftrightarrow \neg\text{even}(N)$

Γ_2: $\text{even}(N) \Leftrightarrow N=0 \vee \exists X(N=s(X) \wedge \text{odd}(X))$ /use the replacement rule
$\text{odd}(N) \Leftrightarrow \neg\{N=0 \vee \exists X(N=s(X) \wedge \neg\text{even}(X))\}$
/under Δ_N

Γ_3: $\text{even}(N) \Leftrightarrow N=0 \vee \exists X(N=s(X) \wedge \text{odd}(X))$
$\text{odd}(N) \Leftrightarrow \exists X(N=s(X) \wedge \text{even}(X))$

The transformation has resulted in a completely positive program Γ_3 ("positive" here means no negation in the definition body). Unfortunately, we cannot consider it legitimate, as no predicate in the folded definition was labeled u when Γ_1 was obtained. Nonetheless, it is very apparent that the transition from Γ_0 to Γ_1 is valid in any logical sense, as it is just substitution of equals for equals with the definition of $\text{odd}(X)$ held unchanged. This type of transformation is often seen but is strictly ruled out by the basic transformation system. In the sequel, we show that by making use of the basic system, it is possible to build a more powerful system, the extended system, for which the above transformation process is a legitimate one.

5.1. New rules

⟨*Initial program′*⟩

An initial program $\Gamma_0 = \{p_i(x_i) \Leftrightarrow F_{i,0} \mid 1 \leqslant i \leqslant N\}$ is a first-order program in $L_B + \{p_1, \ldots, p_N\}$ with $\{p_1, \ldots, p_N\}$ being organized into two layers, *old predicates* and *new predicates*, in such a way that

(I1) the body of a predicate definition about an old predicate contains no new predicates.

In other words, while new predicates in the initial program Γ_0 may refer to any predicates, i.e. the new ones, old ones and primitives, the reference of the old predicates is confined to the old ones and primitives. In the course of a transformation process, however, it can happen, by folding, that old predicates start referring to new predicates, thereby forming mutual recursion with them.

Suppose Γ_0 has been transformed to Γ_k. Choose a predicate definition $h(x)\Leftrightarrow B_k$ from Γ_k and transform B_k into B_{k+1} by applying one of the following rules below, and put

$$\Gamma_{k+1}=(\Gamma_k\setminus\{h(x)\Leftrightarrow B_k\})\cup\{h(x)\Leftrightarrow B_{k+1}\}.$$

⟨*unfolding′*⟩

Select an atom $d(t)$ containing a user predicate d from the body B_k. Whether d is labeled or not is irrelevant. Let $d(y)\Leftrightarrow D_0[y]$ be a predicate definition about d in Γ_0. Unfold $d(t)$ into $D_0[t]$ provided

(U1) if d is a new predicate, so is h.

Mark with u all the user predicates introduced by this unfolding. Put

$$B_{k+1}=B[D_0[p^{\mathrm{u}},t]/d(t)].$$

⟨*folding′*⟩

Select a subformula $D_0[t]$ from B_k such that there exists a predicate definition $d(y)\Leftrightarrow D_0[y]$ in Γ_0. Let $h(x)\Leftrightarrow B_0$ be a predicate definition about h in Γ_0. Fold $B_k[D_0[t]]$ into $B_{k+1}=B_k[d(t)/D_0[t]]$, without marking $d(t)$ with u, provided

(F1′) Either both h and d are new predicates and every user predicate in the $D_0[t]$ is labeled u, or h is an old predicate and d is a new predicate and $D_0[t]$ contains no new predicates.

(F2) $B_0[\mathbf{u}]\leqslant_{\Delta} B_{k+1}[\mathbf{u}]$

⟨*replacement′*⟩

Take as B_{k+1} any formula such that

$$B_k=_{\Delta}B_{k+1}$$

holds even when we regard the labeled and nonlabeled predicates as independent ones.

5.2. *Correctness*

Theorem 5.2 (Equivalence property). *Let Δ be a base theory in L_{B} and $\Gamma_0=\{p_i(x_i)\Leftrightarrow F_{i,0}\mid 1\leqslant i\leqslant N\}$ a program. Suppose Γ_0 has been transformed to $\Gamma_k=\{p_i(x_i)\Leftrightarrow F_{i,k}\mid 1\leqslant i\leqslant N\}$ using new rules. Then Γ_0 and Γ_k are chain-equivalent under Δ, and for a sentence G in $L_{\mathrm{B}}+\{p_1,\ldots,p_N\}$,*

$$\Delta\cup\Gamma_0\models_3 G \text{ iff } \Delta\cup\Gamma_k\models_3 G.$$

If Γ_0 and Γ_k are both call-consistent and strict w.r.t. G,

$$\Delta\cup\Gamma_0\vdash G \text{ iff } \Delta\cup\Gamma_k\vdash G$$

Proof. For notational convenience, we use $\boldsymbol{p}=p_1,\ldots,p_M$ for new predicates and $\boldsymbol{q}=q_1,\ldots,q_N$ for old predicates, respectively, and write Γ_0, symbolically, as

$$\Gamma_0=\{p_i(x_i)\Leftrightarrow F_{i,0}[\boldsymbol{p},\boldsymbol{q}]\mid 1\leqslant i\leqslant M\}\cup\{q_j(y_j)\Leftrightarrow G_{j,0}[\boldsymbol{q}]\mid 1\leqslant j\leqslant N\}.$$

Here $F_{i,0}[\boldsymbol{p},\boldsymbol{q}]$ indicates that the user predicates of $F_{i,0}$ are among $\boldsymbol{p}\cup\boldsymbol{q}$ and those of $G_{j,0}[\boldsymbol{q}]$ are among $\boldsymbol{q}$. Prepare fresh predicate symbols $\boldsymbol{q}^*=q_1^*,\ldots,q_N^*$ corresponding to $q_1,\ldots,q_N$. For a formula G, let G^* denote the one obtained from G by replacing each atom of the form $q_j(t)$ with $q_j^*(t)$ with $q_j^{\mathrm{u}}(t')$ with $q_j^{*\mathrm{u}}(t')$ $(1\leqslant j\leqslant N)$, respectively. We will use $\boldsymbol{q}^{\mathrm{u}}$ for $q_1^{\mathrm{u}},\ldots,q_N^{\mathrm{u}}$ and $\boldsymbol{q}^{*\mathrm{u}}$ for $q_1^{*\mathrm{u}},\ldots,q_N^{*\mathrm{u}}$. Now put

$$\Gamma_0^*=\{p_i(x_i)\Leftrightarrow F_{i,0}[\boldsymbol{p},\boldsymbol{q}]\mid 1\leqslant i\leqslant M\}$$
$$\cup\{q_j(y_j)\Leftrightarrow G_{j,0}^*\mid 1\leqslant j\leqslant N\}\cup\{q_j^*(y_j)\Leftrightarrow q_j(y_j)\mid 1\leqslant j\leqslant N\}.$$

Assuming Lemma 5.3, we can see, at any k, by unfolding each $q_j^*(s)$ in $G_{j,k}^*$ into $q_j(s)$ $(1\leqslant j\leqslant N\}$ to obtain $\Gamma_k\cup\{q_j^*(x_j)\Leftrightarrow q_j(x_j)\mid 1\leqslant j\leqslant N\}$, that Γ_k^* is chain-equivalent to Γ_k w.r.t. the p_i's and q_j's under any base theory Δ. Since $\Gamma_0^*,\ldots,\Gamma_k^*$ are all chain-equivalent by the construction and the chain-equivalence is transitive, we can conclude that Γ_0 and Γ_k are chain-equivalent under Δ as well; hence, the theorem. □

Lemma 5.3. *For a transformation sequence $\Gamma_0,\Gamma_1,\ldots,\Gamma_k,\ldots$ by the extended system, there exists an equivalence-preserving transformation sequence $\Gamma_0^*,\Gamma_1^*,\ldots,\Gamma_k^*,\ldots$ by the basic transformation system that maintains the following relationship between Γ_k and Γ_k^* at every k.*

⟨*INVARIANT*-3⟩

$$\Gamma_k=\{p_i(x_i)\Leftrightarrow F_{i,k}\mid 1\leqslant i\leqslant M\}\cup\{q_j(y_j)\Leftrightarrow G_{j,k}\mid 1\leqslant j\leqslant N\},$$
$$\Gamma_k^*=\{p_i(x_i)\Leftrightarrow F_{i,k}\mid 1\leqslant i\leqslant M\}\cup\{q_j(y_j)\Leftrightarrow G_{j,k}^*\mid 1\leqslant j\leqslant N\}$$
$$\cup\{q_j^*(y_j)\Leftrightarrow q_j(y_j)\mid 1\leqslant j\leqslant N\}.$$

Proof. We construct, by way of the basic transformation system, a transformation sequence $\Gamma_0^*,\Gamma_1^*,\ldots,\Gamma_k^*,\ldots$ through which INVARIANT-3 is maintained. Assume that it holds for k (the case for $k=0$ is obvious by definition). We show it also holds for $k+1$.

We suppose that Γ_{k+1} is obtained from Γ_k by transforming a predicate definition $h(x)\Leftrightarrow B_k$ in Γ_k into $h(x)\Leftrightarrow B_{k+1}$. Let $h(x)\Leftrightarrow B_0$ be a predicate definition about h in Γ_0. We have three cases.

Case 1: unfolding'. B_{k+1} is obtained by unfolding an atom $d(t)$ in the body $B_k[d(t)]$ using $d(y)\Leftrightarrow D_0[y]$ in Γ_0. We first treat the case in which d is an old predicate. In this case, $D_0[y]$ includes only old predicates, and in this unfolding, the body $B_k[d(t)]$ is unfolded into $B_k[D_0[\boldsymbol{q}^{\mathrm{u}},t]]$. We show that we can obtain, from Γ_n^* by unfolding, $h(x)\Leftrightarrow B_k[D_0[\boldsymbol{q}^{\mathrm{u}},t]]$ if h is a new predicate, or $h(x)\Leftrightarrow B_k^*[D_0[\boldsymbol{q}^{*\mathrm{u}},t]]$ otherwise, which means that the invariant holds for $k+1$.

When h is a new predicate, we have $h(x) \Leftrightarrow B_k[d(t)]$ in Γ_n^* by INVARIANT-3 at k. Therefore, by unfolding $B_k[d(t)]$ into $B_k[D_0[\boldsymbol{q}^{*\mathrm{u}}, t]]$ using $d(y) \Leftrightarrow D_0[\boldsymbol{q}^*]$ in Γ_0^*, and then successively unfolding it into $B_k[D_0[\boldsymbol{q}^{\mathrm{u}}, t]]$ using $q_j^*(y_j) \Leftrightarrow q_j(y_j)$ $(1 \leqslant j \leqslant N)$ in Γ_0^*, we obtain $h(x) \Leftrightarrow B_k[D_0[\boldsymbol{q}^{\mathrm{u}}, t]]$ by the basic system. Or else, if h is an old predicate, we have $h(x) \Leftrightarrow B_k^*[d^*(t)]$ in Γ_n^* by INVARIANT-3 at k. Similarly as in the above case, by first unfolding $B_k^*[d^*(t)]$ into $B_k^*[d^{\mathrm{u}}(t)]$ using $d^*(y) \Leftrightarrow d(y)$ in Γ_0^*, and then into $B_k^*[D_0[\boldsymbol{q}^{*\mathrm{u}}, t]]$ using $d(y) \Leftrightarrow D_0[\boldsymbol{q}^*]$ in Γ_0^*, we obtain $h(x) \Leftrightarrow B_k^*[D_0[\boldsymbol{q}^{*\mathrm{u}}, t]]$ in the basic system.

In the remaining cases, where d is a new predicate, h must be a new predicate according to the unfolding condition U1. So, this unfolding, being concerned only with $h(x) \Leftrightarrow B_k[d(t)]$ and $d(y) \Leftrightarrow D_0[y]$, which are also included in Γ_0^* and Γ_k^*, respectively, can take place in Γ_k^*, and the invariant holds for $k+1$.

Case 2: folding'. B_{k+1} is obtained by folding a subformula $D_0[t]$ of B_k into $d(t)$. Write $B_{k+1} = B_k[d(t)/D_0[t]]$ and let $d(y) \Leftrightarrow D_0[y]$ be the folding definition in Γ_0. We have two cases depending upon which condition of F1′ is satisfied. If both d and h are new predicates, we have nothing to prove, as Γ_0^* has $d(y) \Leftrightarrow D_0[y]$ and Γ_k^* has $h(x) \Leftrightarrow B_k$ by INVARIANT-3 at k.

Otherwise, d must be a new predicate and h an old predicate, due to the second half of F1′. $B_0[\mathbf{u}] \leqslant_\Delta B_{k+1}[\mathbf{u}]$ holds by F2. Then the invariant at k guarantees that $h(x) \Leftrightarrow B_k^*[D_0^*[t]]$ is in Γ_k^*, and the folding condition F1′ states that, $D_0[t]$ and, hence, $D_0^*[t]$ contains only old predicates (and primitives). So, we may write $D_0^*[t] = D_0^*[\boldsymbol{q}^*, \boldsymbol{q}^{*\mathrm{u}}, t]$. By unfolding $B_k^*[D_0^*[\boldsymbol{q}^*, \boldsymbol{q}^{*\mathrm{u}}, t]]$ as many times as necessary in the basic system, using $\{q_j^*(y_j) \Leftrightarrow q_j(y_j) \mid 1 \leqslant j \leqslant N\}$ in Γ_0^*, we can obtain $B_k^*[D_0[\boldsymbol{q}^{\mathrm{u}}, t]]$, where every user predicate in $D_0[\boldsymbol{q}^{\mathrm{u}}, t]$ is labeled u, thereby satisfying F1 in the basic system. Now the folding of $B_k^*[D_0[\boldsymbol{q}^{\mathrm{u}}, t]]$ into $B_k^*[d(t)]$ using $d(y) \Leftrightarrow D_0[y]$ in Γ_0^* gives $B_{k+1}^* = B_k^*[d(t)/D_0^*[t]]$. From $B_0^*[\mathbf{u}] = B_0[\mathbf{u}] \leqslant_\Delta B_k[\mathbf{u}] = B_k^*[\mathbf{u}]$, the folding condition F2 in the basic system is satisfied as well. Thus, we have obtained $q_j(x_j) \Leftrightarrow B_{k+1}^*$ from $q_j(x_j) \Leftrightarrow B_k^*$ by the basic system, and the invariant holds again for $k+1$.

Case 3: replacement'. Obvious and omitted.

So, INVARIANT-3 holds again for $k+1$. □

5.3. *Elimination of negation*

Example 5.1 is now legitimate in the extended system (even: old predicate, odd: new predicate). This example suggests a general method to eliminate negation as follows.

A program is said to be *positive* if no user predicate occurs negatively in the body of a predicate definition. We prove that a program $\Gamma = \{p_i(x_i) \Leftrightarrow F_i \mid 1 \leqslant i \leqslant N\}$ is transformable to a pair of positive programs, Γ^+ and Γ^-, such that their union is chain-equivalent to Γ w.r.t. $\{p_1, \ldots, p_N\}$.

Prepare fresh predicate symbols p_i' $(1 \leqslant i \leqslant N)$ whose arity equals that of p_i. They are intended for the negation of the p_i's. Write a formula F in $L_{\mathrm{B}} + \{p_1, \ldots, p_N\}$ as $F[p_1^+, \ldots, p_N^+, p_1^-, \ldots, p_N^-]$ and let p_i^+ (p_i^-) refer to all of the positive (negative) occurrences of the predicate symbol p_i $(1 \leqslant i \leqslant N)$ in F. Next define positive formulae F^+ and F^-, respectively, by

$$F^+ = F[p_1^+, \ldots, p_N^+, \neg p_1'/p_1^-, \ldots, \neg p_N'/p_N^-] \quad \text{and}$$
$$F^- = \neg(F[\neg p_1'/p_1^+, \ldots, \neg p_N'/p_N^+, p_1^-, \ldots, p_N^-]).$$

Now for a program $\Gamma = \{p_i(x_i) \Leftrightarrow F_i \mid 1 \leqslant i \leqslant N\}$, put, respectively,

$$\Gamma^+ = \{p_i(x_i) \Leftrightarrow F_i^+ \mid 1 \leqslant i \leqslant N\} \quad \text{and} \quad \Gamma^- = \{p_i'(x_i) \Leftrightarrow F_i^- \mid 1 \leqslant i \leqslant N\}.$$

Proposition 5.4. *Let Γ be a program in $L_B + \{p_1, \ldots, p_N\}$. Then $\Gamma^+ \cup \Gamma^-$ is a positive program and chain-equivalent to Γ w.r.t. $\{p_1, \ldots, p_N\}$ under any base theory Δ. And for a sentence G in $L_B + \{p_1, \ldots, p_N\}$,*

$$\Delta \cup \Gamma \models_3 G \text{ iff } \Delta \cup \Gamma^+ \cup \Gamma^- \vdash G^+.$$

Proof. We use the extended system but do not mention labeling for readability. Construct an initial program $\Gamma' = \Gamma \cup \{p_i'(x_i) \Leftrightarrow \neg p_i(x_i) \mid 1 \leqslant i \leqslant N\}$, in which $\{p_1', \ldots, p_N'\}$ are considered as new predicates and $\{p_1, \ldots, p_N\}$ old predicates. Replace each definition $p_i(x_i) \Leftrightarrow F_i[p_1^+, \ldots, p_N^+, p_1^-, \ldots, p_N^-]$ in Γ' with $p_i(x_i) \Leftrightarrow F_i[p_1^+, \ldots, p_N^+, \neg\neg p_1^-, \ldots, \neg\neg p_N^-]$ and fold it into $p_i(x_i) \Leftrightarrow F_i[p_1^+, \ldots, p_N^+, \neg p_1', \ldots, \neg p_N']$. We, thus, obtain $p_i(x_i) \Leftrightarrow F_i^+$ $(1 \leqslant i \leqslant N)$. Next unfold each $p_i'(x_i) \Leftrightarrow \neg p_i(x_i)$ $(1 \leqslant i \leqslant N)$ in Γ' into $p_i'(x_i) \Leftrightarrow \neg F_i[p_1^+, \ldots, p_N^+, p_1^-, \ldots, p_N^-]$, then replace it with $p_i'(x_i) \Leftrightarrow \neg F_i[\neg\neg p_1^+, \ldots, \neg\neg p_N^+, p_1^-, \ldots, p_N^-]$, and fold into $p_i'(x_i) \Leftrightarrow \neg F_i[\neg p_1', \ldots, \neg p_N', p_1^-, \ldots, p_N^-]$ $(= F_i^-)$. In this way, we obtain $\Gamma^+ \cup \Gamma^-$ from Γ' by the extended system. Since Γ' and Γ are chain-equivalent w.r.t. $\{p_1, \ldots, p_N\}$ under any Δ, so are $\Gamma^+ \cup \Gamma^-$ and Γ. For the second half, we see that

$\Delta \cup \Gamma \models_3 G$ iff $\Delta \cup \Gamma' \models_3 G$ (G does not contain $p_1', \ldots, p_N'$)

iff $\Delta \cup \Gamma' \models_3 G^+$ (replace negative occurrences of $p_1, \ldots, p_N$ in G with $\neg p_1', \ldots, \neg p_N'$ using $\{p_i'(x_i) \Leftrightarrow \neg p_i(x_i) \mid 1 \leqslant i \leqslant N\}$)

iff $\Delta \cup \Gamma^+ \cup \Gamma^- \models_3 G^+$ (Γ' and $\Gamma^+ \cup \Gamma^-$ are chain-equivalent)

iff $\Delta \cup \Gamma^+ \cup \Gamma^- \vdash G^+$ (by Theorem 3.5). □

Observe that, although positive programs are always strict w.r.t. any literal goal, this is false for general goals. For example, $\Gamma = \{a \Leftrightarrow a\}$ is not strict w.r.t. $G = a \vee \neg a$. This is why we use G^+ in $\Delta \cup \Gamma^+ \cup \Gamma^- \vdash G^+$ instead of G. More importantly, quantifiers cannot be eliminated by this method. For instance, if $\Gamma = \{p(X) \Leftrightarrow \exists Y q(X, Y), q(X, Y) \Leftrightarrow q(X, Y)\}$, Γ^- will be $\{p'(X) \Leftrightarrow \forall Y q'(X, Y), q'(X, Y) \Leftrightarrow q'(X, Y)\}$. To eliminate $\forall Y$ further, we have to apply unfold/fold transformation; unfold $p'(X) \Leftrightarrow \forall Y q'(X, Y)$ into $p'(X) \Leftrightarrow \forall Y q'^{\mathrm{u}}(X, Y)$, and then fold the latter into $p'(X) \Leftrightarrow p'(X)$.

Nonetheless, when Γ takes a special form, the iff definition of a general program with no internal variables, like the even program, Γ^- becomes a positive program with no universal quantifiers quantifying user predicates, if Clark's equational theory is assumed as an underlying base theory. In this sense, Proposition 5.4 gives a generalization of the negation technique [27], which is applicable only to definite programs without internal variables.

5.4. Strict programs

Proposition 5.4 states that positive programs form a representative class of first-order programs in the sense that every program is equivalent to some positive program as far as the logical consequence relationship is concerned (and the three-valued logical consequence relationship is reduced to the two-valued one). We prove here, as another application of the extended system, that strict programs also form a representative class. Namely, Proposition 5.5.

Proposition 5.5. *For every first-order program Γ in $L_{\mathrm{B}}+\{p_1,\ldots,p_N\}$, there exists a strict program Γ'_{s} in $L_{\mathrm{B}}+\{p_1,\ldots,p_N\}+\{p'_1,\ldots,p'_N\}$ which is chain-equivalent to Γ w.r.t. $\{p_1,\ldots,p_N\}$ under any base theory Δ. And for a sentence G in $L_{\mathrm{B}}+\{p_1,\ldots,p_N\}$, we have*

$$\Delta\cup\Gamma\models_3 G \text{ iff } \Delta\cup\Gamma'_{\mathrm{s}}\vdash G_{\mathrm{s}},$$

where $G_{\mathrm{s}}=G[p_1^+,\ldots,p_N^+,p'_1/p_1^-,\ldots,p'_N/p_N^-]$.

Proof. Let $\Gamma=\{p_i(x_i)\Leftrightarrow F_i \mid 1\leqslant i\leqslant N\}$ be a given program. Starting from $\Gamma'=\Gamma\cup\{p'_i(x_i)\Leftrightarrow p_i(x_i)\mid 1\leqslant i\leqslant N\}$, in which $\{p'_1,\ldots,p'_N\}$ are considered as new predicates and $\{p_1,\ldots,p_N\}$ as old predicates, we can reach, through an unfold/fold transformation process very similar to the one in the proof of Proposition 5.4,

$$\Gamma'_{\mathrm{s}}=\{p_i(x_i)\Leftrightarrow F_i[p_1^+,\ldots,p_N^+,p'_1/p_1^-,\ldots,p'_N/p_N^-]\mid 1\leqslant i\leqslant N\}$$
$$\cup\{p'_i(x_i)\Leftrightarrow F_i[p'_1/p_1^+,\ldots,p'_N/p_N^+,p_1^-,\ldots,p_N^-]\mid 1\leqslant i\leqslant N\},$$

which is chain-equivalent to Γ w.r.t. $\{p_1,\ldots,p_N\}$ under an arbitrary base theory. To see that Γ'_{s} is strict and strict w.r.t. G_{s}, note that each class, the class of new predicates and that of old predicates, depends positively on itself and negatively on the other class, and they are disjoint. Now

$$\begin{aligned}
\Delta\cup\Gamma\models_3 G &\text{ iff } \Delta\cup\Gamma'\models_3 G && (G \text{ does not contain } p'_1,\ldots,p'_N)\\
&\text{ iff } \Delta\cup\Gamma'\models_3 G_{\mathrm{s}} && (\text{use } \{p'_i(x_i)\Leftrightarrow p_i(x_i)\mid 1\leqslant i\leqslant N\} \text{ in } \Gamma')\\
&\text{ iff } \Delta\cup\Gamma'_{\mathrm{s}}\models_3 G_{\mathrm{s}} && (\Gamma' \text{ and } \Gamma'_{\mathrm{s}} \text{ are chain-equivalent})\\
&\text{ iff } \Delta\cup\Gamma'_{\mathrm{s}}\vdash G_{\mathrm{s}} && (\text{by Theorem 3.2}).\quad\square
\end{aligned}$$

This proposition generalizes Theorem 7 in [10] which requires the allowedness of Γ (and G).

Example 5.6.

Γ: $\quad \text{even}(N) \Leftrightarrow N = 0 \vee \exists X (N = s(X) \wedge \neg \text{even}(X))$

Γ'_s: $\quad \text{even}(N) \Leftrightarrow N = 0 \vee \exists X (N = s(X) \wedge \neg \text{even}'(X))$

$\quad \text{even}'(N) \Leftrightarrow N = 0 \vee \exists X (N = s(X) \wedge \neg \text{even}(X))$

6. Concluding remarks

We have presented two unfold/fold systems (the basic system in Section 4 and the extended system in Section 5) which preserve the logical strength of first-order programs in the sense of Kleene's three-valued logic (Theorem 4.1 and Theorem 5.2). They consist of an unfolding rule, a folding rule and a replacement rule. When applied to call-consistent programs, they preserve the proof-theoretic strength of the transformed programs in two-valued logic. In particular, they preserve the set of literals provable from strict programs, thereby preserving the success and finite failure sets of definite programs as a special case.

Our systems bear close resemblance to Seki's unfold/fold system [29]. They all aim at equivalence-preserving transformation. And interestingly, the folding conditions in the extended system almost coincide with his when ours are restricted to stratified programs. Nevertheless, unfolding is different and the employed semantics is also different. The major difference, however, is that our systems are applicable even when clauses contain arbitrarily complex formulae in their bodies, a fact which, on the other hand, might complicate the task of checking folding conditions in the general case.

We have also revealed that positive programs and strict programs form a representative class of first-order programs, respectively, in the sense that the three-valued logical consequence relationship first-order programs have with goals is reducible to the two-valued one between positive (strict) programs and the corresponding goals.

Since we do not assume any specific theory as an underlying base theory that determines the meaning of primitive predicates, our systems can work with any first-order theory. Such generality is expected to contribute to expanding the field of unfold/fold transformation.

Acknowledgment

We thank the anonymous referees for valuable comments.

References

[1] K.R. Apt, H.A. Blair and A. Walker, Towards a theory of declarative knowledge, in: J. Minker, ed., *Foundations of Deductive Databases and Logic Programming* (Morgan Kaufmann, Los Altos, CA, 1987) 89–148.

[2] R. Barbuti, P. Mancarella, D. Pedreschi and F. Turini, A transformational approach to negation in logic programming, *J. Logic Programming* **8** (1990) 201–228.

[3] H.A. Blair, The recursion-theoretic complexity of the semantics of predicate logic as a programming language, *Inform. and Control* **54** (1982) 25–47.

[4] R.M. Burstall and J. Darlington, A transformation system for developing recursive programs, *J. ACM* **24** (1977) 44–67.

[5] L. Cavendon and J.W. Lloyd, A completeness theorem for SLDNF resolution, *J. Logic Programming* **7** (1989) 177–191.

[6] C.C. Chan, *Model Theory* (North Holland, Amsterdam, 1973).

[7] K. Clark and S. Sickel, Predicate logic: a calculus for deriving programs, in: *Proc. 5th Internat. Conf. on Artificial Intelligence* (1977) 419–420.

[8] H. Comon and P. Lescanne, Equational problems and disunification, *J. Symbolic Comput.* **7** (1989) 371–425.

[9] G. Dayantis, Logic program derivation for a class of first-order logic relations, in: *Proc. 10th Internat. Joint Conf. on Artificial Intelligence* (1987) 9–14.

[10] W. Drabent and M. Martelli, Strict completion of logic programs, *New Generation Computing* **9** (1991) 69–79.

[11] C.J. Hogger, Derivation of logic programs, *J. ACM* **28** (1981) 372–392.

[12] T. Kanamori and H. Fujita, Unfold/fold logic program transformation with counters, ICOT Tech. Report TR-179, 1986.

[13] T. Kanamori and K. Horiuchi, Construction of logic programs based on generalized unfold/fold rules, in: *Proc. 4th Internat. Conf. on Logic Programming* (1987) 745–768.

[14] S.J. Kleene, *Introduction to Metamathematics* (North-Holland, Amsterdam, 1971).

[15] 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: *Proc. 9th Ann. ACM Symp. on Principles of Programming Languages* (1982) 255–257.

[16] K. Kunen, Negation in logic programming, *J. Logic Programming* **4** (1987) 289–308.

[17] K. Kunen, Answer sets and negation as failure, in: *Proc. 4th Internat. Conf. on Logic Programming* (1987) 219–228.

[18] K. Kunen, Signed data dependencies in logic programs, *J. Logic Programming* **7** (1989) 231–245.

[19] J.W. Lloyd, *Foundation of Logic Programming* (Springer, Berlin, 1984).

[20] J.W. Lloyd and R.W. Topor, Making Prolog more expressive, *J. Logic Programming* **1** (1984) 225–240.

[21] D. Lugiez, A deduction procedure for first-order programs, in: *Proc. 6th Internat. Conf. on Logic Programming* (1989) 585–599.

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

[23] M.J. Maher, Complete axiomatizations of the algebras of finite, rational and infinite trees, in: *Proc. 3rd Ann. Symp. on Logic in Computer Science* (1988) 348–357.

[24] M.J. Maher, A transformation system for deductive database modules with perfect model semantics, in: *Proc. 9th Conf. on Foundations of Software Technology and Theoretical Computer Science* (1989) 89–98.

[25] Z. Manna and R. Waldinger, The automatic synthesis of systems of recursive programs, in: *Proc. 5th Internat. Joint Conf. on Artificial Intelligence* (1977) 405–411.

[26] T. Sato, Completed logic programs and their consistency, *J. Logic Programming* **9** (1990) 33–44.

[27] T. Sato and H. Tamaki, Transformational logic program synthesis, in: *Proc. Internat. Conf. on Fifth Generation Computer Systems* (1984) 195–201.

[28] T. Sato and H. Tamaki, First-order compiler: a deterministic logic program synthesis algorithm, *J. Symbolic Comput.* **8** (1989) 605–627.

[29] H. Seki, Unfold/fold transformation of stratified programs, in: *Proc. 6th Internat. Conf. on Logic Programming* (1989) 554–568.
[30] H. Seki, A comparative study of the well-founded and the stable model semantics: transformation's viewpoint, in: *Proc. Workshop on Logic Programming and Non-Monotonic Logic* (1990) 115–123.
[31] H. Tamaki and T. Sato, Unfold/fold transformation of logic programs, in: *Proc. 2nd Internat. Logic Programming Conf.* (1984) 127–137.
[32] H. Tamaki and T. Sato, A generalized correctness proof of the unfold/fold logic program transformation, Tech. Report 86-4, Ibaraki University, 1986.