

Abductive Logic Programming by Nonground Rewrite Systems

Fangzhen Lin

Department of Computer Science
Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong

Jia-Huai You

Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada

Abstract

Logic programming with negation offers a compelling approach to abductive reasoning. This paper shows a simple view of abduction in this context for the completion semantics, under which the problem of abduction becomes one of solving quantified equations and disequations. By this way of treating abduction, the problems with nonground negative queries in the previous approaches no longer exist. We show the soundness and completeness results for our approach.

Introduction

Logic programming with negation has been considered an attractive approach to abduction (Kakas, Kowalski, & Toni 1995). The goal is to show that an observation, in terms of a query, can be explained by a reasoning process supported with hypotheses, called abducibles, while satisfying posted constraints.

A number of approaches to abductive logic programming have been proposed. One approach is based on the (partial) stable model semantics (e.g., (Eshghi & Kowalski 1989; Kakas, Michael, & Mourlas 2000; Lin & You 2002)). These proof procedures are designed essentially for ground programs. Another family of procedures are proposed for nonground programs under the completion semantics (Console, Theseider, & Porasso 1991; Fung & Kowalski 1997; Endriss *et al.* 2004), where the use of the IFF definitions as rewrite rules has considerably simplified the proof process, and made abductive reasoning more intuitive and transparent. This is in contrast with the approach based on nested proof trees called the SLDNFA procedure (Denecker & De Schreye 1998).

Iff definitions may contain existential variables. When negated, they become universally quantified. Since logic programming traditionally can only process queries with existential variables, processing queries with universal variables has been said to be *unsafe*. In the past, all proof procedures are designed to satisfy some safety conditions.

A goal is said to *flounder* if all of its subgoals are nonground negative literals. To prevent a goal from floundering, the best-known syntactic condition is the so-called *allowedness condition*. An iff definition $p(\bar{X}) \leftrightarrow D_1 \vee \dots \vee D_m$

is *allowed* if every variable other than \bar{X} occurring in a D_i occurs in a positive nonequality atom in the same D_i ; similarly for queries, constraints, and sets of iff definitions.

Allowedness reinforces a programming style in which every variable that may potentially become universally quantified must be “grounded” by a domain definition, so that floundering can be avoided by instantiating the variables in a negative subgoal before its proof is attempted. For small domains, such an approach is fine. But for large or infinite domains, it makes programming tedious and reasoning inefficient, as in such a procedure a mechanism of enumerating domain elements is tightly coupled with the process of proving negative subgoals. For example, if we have a fact $member(X, [X|R])$ in our program, the completed definition of $member$ is not allowed:

$$member(X, Y) \leftrightarrow Y = [X|R] \vee \dots$$

since variable R doesn't appear in a positive nonequality atom. One can make the definition allowed by a recursive definition of list and restricting R to that domain.

One would expect that, designed specifically for a class of well-behaved programs, namely allowed programs, a proof procedure would be simpler. It turns out all of the previous proof procedures are formulated in some complex ways.

This paper shows that a much simpler view of abduction is possible, even for the class of all programs and goals. We show that the problem of abduction can be partitioned into two subproblems, the first of which is to transform a goal to a formula containing quantified equalities and disequalities, which can then be solved separately. Since variable quantification is represented explicitly according to the semantics, the problem in the past with nonground queries simply becomes a non-issue in our approach. This approach is sound where \models is interpreted in 2-valued logic. For completeness, \models has to be weakened to 3-valued logic.

Definitions

We consider a first-order language L . Variables begin with a capital letter, and predicate/function/constant symbols begin with a lower case letter.

For any syntactic entity Φ , $V(\Phi)$ denotes the set of the variables appearing free in Φ . A conjunction $A_1 \wedge \dots \wedge A_k$ may be written as $[A_1, \dots, A_k]$. We write $\forall\Phi$ to mean that all the variables appearing free in formula Φ are universally

quantified, similarly for $\exists\Phi$. We use the notation $\overline{\Theta}$ to denote a vector of variable quantification, such as $\exists X\forall Y$ for instance.

We assume the *Clark's equality theory* (Clark 1978), written CET below, which essentially says that all syntactically distinct ground terms are not equal, and that functions are one-to-one: $\forall\overline{X}, \overline{Y} f(\overline{X}) \neq g(\overline{Y})$ for any two distinct function symbols (including constants) f and g , and $\forall\overline{X}, \overline{Y} f(\overline{X}) = f(\overline{Y}) \rightarrow \overline{X} = \overline{Y}$. We also assume that the language includes all symbols in the given program. Thus if a and b occur in the program as two constants, then CET will include the axiom $a \neq b$.

An *abductive program* is a triple $\langle T, IC, Ab \rangle$, where

1. T is a finite set of iff definitions of the form

$$p(X_1, \dots, X_n) \leftrightarrow D_1 \vee \dots \vee D_m \quad (1)$$

where p is an n -ary *defined* predicate symbol (denoted p/n), X_1, \dots, X_n are distinct variables, and D_i a conjunction of literals. The variables X_1, \dots, X_n are implicitly universally quantified with the scope being the entire definition, and any other variable occurring in any D_i is existentially quantified with the scope being D_i . When $m = 0$, the right hand side of (1) is *false*.

Iff definitions are the completed definitions of the predicates in a normal program P , denoted $comp(P)$, according to (Clark 1978). For convenience, and also for intuition, in exposition we may just give a normal program instead of its completion. A normal program is a finite set of normal rules of the form $A \leftarrow B_1, \dots, B_j, \text{not } C_1, \dots, \text{not } C_k$, where $j, k \geq 0$, A, B_i and C_i are (nonequality) atoms.

The logical formula for the completion of p/n is

$$\forall\overline{X} p(\overline{X}) \leftrightarrow \bigvee_{i=1}^m \exists\overline{Y}_i [(\overline{X} = \overline{s}_i) \wedge \Phi_i] \quad (2)$$

where \overline{X} is an abbreviation of X_1, \dots, X_n , and each disjunct corresponds to a D_i in the form (1), where, for the corresponding normal rule with p/n in the head, $\overline{X} = \overline{s}_i$ is the conjunction of equations representing unifiability, \overline{Y}_i denotes the variables other than \overline{X} that appear in the disjunct, and Φ_i is the rule body with not replaced by \neg .

The negation of an equation, $\neg(s=t)$, is called a *disequation* and will be written as $s \neq t$. We use *equation* and *equality* interchangeably. We write $\neg(\overline{t} = \overline{s})$ by $\overline{t} \neq \overline{s}$.

2. IC is a consistent finite set of constraints of the form $\perp \leftarrow A_1, \dots, A_k, \neg B_1, \dots, \neg B_m$ where A_i and B_i are atoms. All variables in a constraint are universally quantified.

3. Ab is a finite set of predicate symbols, called *abducibles*, which are different from $=$ and any defined predicate symbol.

A *query*, also called a *goal*, is a conjunction of literals. All variables appearing in a goal are free. During derivation, a derived goal may be a complex formula involving \wedge, \vee and \neg , and some variables may become universally quantified. Thus, a goal is generally called a *goal formula*, which formally is a formula of the form $\overline{\Theta}\Phi$, where Φ is a quantifier-free formula with negation appearing only in front of an atom.

Given an abducible program $\langle T, IC, Ab \rangle$ and a query G , the *initial query* is the goal formula $\forall\overline{X}. G \wedge \Phi_{IC}$, where

- Φ_{IC} is the conjunction of the constraints in IC by converting a constraint of the form $\perp \leftarrow A_1, \dots, A_k, \neg B_1, \dots, \neg B_m$ into the disjunction $[\neg A_1 \vee \dots \vee \neg A_k \vee B_1 \vee \dots \vee B_m]$.
- \overline{X} is the tuple of variables in IC . We assume here that G and IC do not have common variables - this can be achieved by variable renaming if necessary. Thus the initial query is logically equivalent to $G \wedge \forall\overline{X} \Phi_{IC}$. We move all quantifiers to the outside during rewriting.

In the following, we shall abuse the language and use IC to stand for its corresponding formula Φ_{IC} as well. Thus we'll write the initial query as $\forall\overline{X}. G \wedge IC$.

An *answer* to a query G is a pair (Δ, σ) , where Δ is a finite set of ground abducible atoms, and σ is a substitution of ground terms for variables in $V(G)$, such that

$$T \cup comp(\Delta) \cup CET \models G\sigma \wedge \forall\overline{X} IC. \quad (3)$$

We may simply call σ an *answer* when there are no abducibles involved. The symbol \models refers to the 2-valued entailment relation if not said otherwise.

Our objective in this paper is to propose a set of rules to rewrite an initial query into a formula of the form $\overline{\Theta}(E \vee \Phi)$, and then *extract* answers from E when E contains only $=$ and abducibles.

Goal Rewriting under Completion Semantics

A goal rewrite system is a transformation system that consists of iff definitions for unfolding, augmented by some simple rules for equivalence-preserving transformations.

Simplification rules

We use symbols T and F for boolean constants *true* and *false*, respectively. Let $\overline{\Theta}\Pi$ be a goal formula. Π can be rewritten according to the following rules (and their symmetric cases).

- SR1. $F \vee \Phi \rightarrow \Phi$ SR2. $F \wedge \Phi \rightarrow F$
SR3. $T \wedge \Phi \rightarrow \Phi$ SR4. $T \vee \Phi \rightarrow T$
SR5. $(\Phi_1 \vee \Phi_2) \wedge \Phi_3 \rightarrow (\Phi_1 \wedge \Phi_3) \vee (\Phi_2 \wedge \Phi_3)$

We assume that $\neg T$ will be rewritten to F and $\neg F$ to T , automatically.

Rewrite rules for unification

Let E be a conjunction of equalities. Let δ be the mgu of E when E is unifiable. We will write δ as a set of equations. We define $\delta_E = T$ if $\delta = \emptyset$, and $\delta_E = \delta$ otherwise.

Unification rules:

- (1) $E \rightarrow F$ if E is not unifiable
- (2) $E \rightarrow \delta_E$ if E is unifiable
- (3) $X = t \wedge \Phi \rightarrow X = t \wedge \Phi\{X/t\}$, where X does not occur in t and $\Phi\{X/t\}$ is the result of replacing X in Φ by t .
- (4) $X \neq t \vee \Phi \rightarrow X \neq t \vee \Phi\{X/t\}$

The unification rules should be applied whenever possible to propagate the unifier and to prevent non-unifiable patterns from being pursued.

Unfolding

Unfolding a positive literal is clear. Unfolding a negative literal raises the question of quantification for variables in D_i 's other than X_1, \dots, X_n .

Given a completed definition in the form (2), a negative literal $\neg p(\bar{t})$ will be unfolded according to the following equivalences.

$$\neg p(\bar{t}) \leftrightarrow \bigwedge_{i=1}^m \forall \bar{Y}_i [\bar{t} \neq \bar{s}_i \vee \neg \Phi_i] \quad (\text{i})$$

$$\leftrightarrow \bigwedge_{i=1}^m \forall \bar{Y}_i [\bar{t} \neq \bar{s}_i \vee [\bar{t} = \bar{s}_i \wedge \neg \Phi_i]] \quad (\text{ii})$$

$$\leftrightarrow \bigwedge_{i=1}^m \forall \bar{Y}'_i [\bar{t} \neq \bar{s}_i] \vee \exists \bar{Z}_i \forall \bar{R}_i [(\bar{t} = \bar{s}_i \wedge \neg \Phi_i) \{ \bar{Y}'_i / \bar{Z}_i \}] \quad (\text{iii})$$

where $\bar{Y}'_i = V(\bar{s}_i)$, which are renamed to \bar{Z}_i in the second disjunct (by substitution $\{ \bar{Y}'_i / \bar{Z}_i \}$) and become existentially quantified, and $\bar{R}_i = \bar{Y}_i \setminus \bar{Y}'_i$ are the remaining variables. Here we have assumed that $V(\bar{t}) \cap \bar{Y}_i = \emptyset$ for each i . This can always be achieved by renaming variables in \bar{Y}_i when necessary.

For instance, given

$$p(V) \leftrightarrow \exists X, Y, Z (V = s(X, Y) \wedge \neg q(Y, f(Z))),$$

for $\neg p(f(X))$, the formula (i) above is

$$\forall X_1, X_2, X_3 (f(X) \neq s(X_1, X_2) \vee q(X_2, f(X_3))),$$

and the formula (iii) is

$$\begin{aligned} & \forall X_1, X_2 (f(X) \neq s(X_1, X_2)) \vee \\ & \exists Y_1, Y_2 \forall X_3 (f(X) = s(Y_1, Y_2) \wedge q(Y_2, f(X_3))). \end{aligned}$$

The significance of the formula (iii) is the following. As mentioned earlier, the purpose of rewriting is to generate a goal formula of the form $\bar{\Theta}(E \vee \Phi)$ such that E mentions only = and abducibles and then to extract answers from $\bar{\Theta}E$. In general, $\bar{\Theta}(E \vee \Phi) \not\equiv \bar{\Theta}E \vee \bar{\Theta}\Phi$. However, in some situations, e.g., when E and Φ do not share universal variables, the two are logically equivalent. In this case, $\bar{\Theta}E$ can be evaluated independently, so is $\bar{\Theta}\Phi$. The formula (iii) says some universal variables resulted from an iff definition can be converted to existential ones, thus reducing the chances of shared universal variables. This will simplify the process of answer extraction (cf. the next section).

We now prove that the equivalence (iii) indeed holds under the completion semantics.

Lemma 1 (Correctness of unfolding)

Given a program P , a predicate p , and a completed definition in the form (2), let $p(\bar{t})$ be an atom such that $V(\bar{t}) \cap \bar{Y}_i = \emptyset$ for each i . Then, $\text{comp}(P) \cup \text{CET}$ entails

$$(a) \quad \forall p(\bar{t}) \leftrightarrow \bigvee_{i=1}^m \exists \bar{Y}_i [(\bar{t} = \bar{s}_i) \wedge \Phi_i]$$

$$(b) \quad \forall \neg p(\bar{t}) \leftrightarrow \text{the formula in (iii) above.}$$

Proof. The correctness of part (a) is clear. Let's prove part (b). It is clear that under $\text{comp}(P)$, $\neg p(\bar{t})$ is equivalent to the formula (ii) above, which is equivalent to

$$\bigwedge_{i=1}^m \forall \bar{Y}'_i \forall \bar{R}_i [\bar{t} \neq \bar{s}_i \vee [\bar{t} = \bar{s}_i \wedge \neg \Phi_i]],$$

thus equivalent to

$$\bigwedge_{i=1}^m \forall \bar{Y}'_i [\bar{t} \neq \bar{s}_i \vee \forall \bar{R}_i [\bar{t} = \bar{s}_i \wedge \neg \Phi_i]]$$

as the variables in \bar{R}_i do not occur in \bar{t} and \bar{s}_i . It is easy to see that this formula entails the formula in (iii) above. We show that the other way around is also true under CET. To show this, we prove that under CET, for each i ,

$$\exists \bar{Z}_i \forall \bar{R}_i [(\bar{t} = \bar{s}_i \wedge \neg \Phi_i) \{ \bar{Y}'_i / \bar{Z}_i \}] \quad (4)$$

entails

$$\forall \bar{Y}'_i [\bar{t} \neq \bar{s}_i \vee \forall \bar{R}_i [\bar{t} = \bar{s}_i \wedge \neg \Phi_i]]. \quad (5)$$

To show this, suppose

$$\forall \bar{R}_i [(\bar{t} = \bar{s}_i \wedge \neg \Phi_i) \{ \bar{Y}'_i / \bar{Z}_i \}], \quad (6)$$

$$\bar{t} = \bar{s}_i. \quad (7)$$

From (6), we get $(\bar{t} = \bar{s}_i) \{ \bar{Y}'_i / \bar{Z}_i \}$, thus $\bar{t} = (\bar{s}_i \{ \bar{Y}'_i / \bar{Z}_i \})$ since \bar{Y}'_i does not occur in \bar{t} . So by (7) we have $\bar{s}_i = (\bar{s}_i \{ \bar{Y}'_i / \bar{Z}_i \})$, thus $\bar{Z}_i = \bar{Y}'_i$ by CET. From this and (6), we get $\forall \bar{R}_i [(\bar{t} = \bar{s}_i \wedge \neg \Phi_i)]$. Thus we have shown that

$$\text{CET} \models (6) \rightarrow [\bar{t} \neq \bar{s}_i \vee \forall \bar{R}_i [\bar{t} = \bar{s}_i \wedge \neg \Phi_i]].$$

So by \forall -introduction, we have $\text{CET} \models (6) \rightarrow (5)$, and by \exists -introduction, we have $\text{CET} \models (4) \rightarrow (5)$. \square

With this lemma, we can now define unfolding rules.

Unfolding Let $\Pi = \bar{\Theta}\Phi$ be a goal formula.

1. If $p(\bar{t})$ occurs in Φ positively (i.e. not under \neg operator), then rewrite Π into the following goal formula

$$\bar{\Theta} \exists \bar{Y}_1 \dots \bar{Y}_m \Phi',$$

where Φ' is the result of replacing this occurrence of $p(\bar{t})$ by

$$\bigvee_{i=1}^m (\bar{t} = \bar{s}_i) \wedge \Phi_i.$$

Here we assume that the following sets of variables

$$V(\Phi), \bar{Y}_1, \dots, \bar{Y}_m$$

are pair-wise disjoint. This can be achieved by renaming variables in \bar{Y}_i if necessary.

2. If $\neg p(\bar{t})$ occurs in Φ , then rewrite Π into the following goal formula

$$\bar{\Theta} \forall \bar{Y}'_1 \exists \bar{Z}_1 \forall \bar{R}_1 \dots \forall \bar{Y}'_m \exists \bar{Z}_m \forall \bar{R}_m \Phi',$$

where Φ' is the result of replacing this occurrence of $\neg p(\bar{t})$ by

$$\bigwedge_{i=1}^m \bar{t} \neq \bar{s}_i \vee (\bar{t} = \bar{s}_i \wedge \bar{\Phi}_i) \{ \bar{Y}'_i / \bar{Z}_i \},$$

where $\bar{\Phi}_i$, the complement of Φ_i , is

$$\neg A_1 \vee \dots \vee \neg A_k \vee B_1 \vee \dots \vee B_n$$

when Φ_i is

$$A_1 \wedge \dots \wedge A_k \wedge \neg B_1 \wedge \dots \wedge \neg B_n.$$

Again we assume that the following sets of variables

$$V(\Phi), \bar{Y}'_1, \bar{Z}_1, \bar{R}_1, \dots, \bar{Y}'_m, \bar{Z}_m, \bar{R}_m$$

are pair-wise disjoint. Again this can be achieved by variable renaming if necessary.

As an example, suppose we have

$$\begin{aligned} p(V) &\leftrightarrow V = s(X, Y) \wedge \neg q(Y, f(Z)) \\ q(Y, Z) &\leftrightarrow Y = L \wedge Z = L \wedge r(L, K). \end{aligned}$$

For the goal $\exists X(p(X) \wedge \neg q(X, Y))$,

- If we unfold $p(X)$ in it, we get

$$\exists X, X_1, X_2, X_3[X = s(X_1, X_2) \wedge \neg q(X_2, f(X_3)) \wedge \neg q(X, Y)].$$

- If we unfold $\neg q(X, Y)$ in it, we get

$$\exists X \forall X_1 \exists X_2 \forall X_3 [p(X) \wedge ((X \neq X_1 \vee Y \neq X_1) \vee (X = X_2 \wedge Y = X_2 \wedge \neg r(X_2, X_3)))].$$

Rewrite System and Answer Extraction

In the following, given a goal formula Π , we write $\Pi \Rightarrow \Pi'$ if Π can be written into Π' by one application of the rules introduced so far, and denote by \Rightarrow^* the transitive closure of \Rightarrow .

Proposition 1 *If $\Pi \Rightarrow^* \Pi'$, then*

$$\text{comp}(P) \cup \text{CET} \models \forall \bar{X} (\Pi \leftrightarrow \Pi')$$

where \bar{X} is the tuple of free variables in Π .

Proof. We only need to prove this for the single step case. The cases for simplification and unification rules are obvious. For the unfolding rules, they follow from Lemma 1 by noting that in first-order logic, if X and Y are different variables, X does not occur in φ , and Y does not occur in ϕ , then

$$(Q_1 X \varphi \diamond Q_2 Y \phi) \leftrightarrow Q_1 X Q_2 Y (\varphi \diamond \phi),$$

where Q_i is either \forall or \exists , and \diamond is either \wedge or \vee . \square

This shows that our rewrite system is sound under the completion semantics and Clark's equality theory. As we mentioned, given an initial query $\forall \bar{X}. G \wedge IC$, our objective is to rewrite it into a formula of the form $\bar{\Theta}(E \vee \Phi)$ such that E mentions only $=$ and abducibles, and then extract answers from E . We now make this process precise.

In the following, given $\bar{\Theta}(E \vee \Phi)$, we call $\bar{\Theta}E$ a *pre-answer formula* if E is quantifier-free and mentions only $=$ and abducibles. In addition, if E does not mention abducibles, then we call $\bar{\Theta}E$ an *answer formula*. Note that an abducible in E may appear positively or negatively, and $\bar{\Theta}E$ may contain universal or existential variables. Given $\bar{\Theta}E$, and a finite set D of ground atoms about abducible predicates in E , we *abduce* the pre-answer formula $\bar{\Theta}E$ to the answer formula $\bar{\Theta}E_D$ by the following transformation: for every abducible ab that appears in E and the subset of D about ab defined as $D_{ab} = \{ab(\bar{s}_i) \in D\}$, any occurrence of $ab(\bar{t})$ in E is replaced by $\bigvee_i \bar{t} = \bar{s}_i$, and $\neg ab(\bar{t})$ by $\bigwedge_i \bar{t} \neq \bar{s}_i$. The intuition here is that if D defines abducibles in E , then $\bar{\Theta}E$ and $\bar{\Theta}E_D$ are equivalent. In other words, $\bar{\Theta}E$ and $\bar{\Theta}E_D$ are equivalent under $\text{comp}(D)$.

Answer Extraction Let $\langle T, IC, Ab \rangle$ be an abductive program and G a goal. Assume that the initial query $\forall \bar{X}. G \wedge IC$ has been rewritten to $\bar{\Theta}(E \vee \Phi)$, where $\bar{\Theta}E$ is a pre-answer

formula. Let Δ be a finite set of ground abducible atoms, and $\sigma = \{X/t \mid X \in V(G)\}$ a substitution for variables in $V(G)$. We say that (Δ, σ) is extracted as an answer to G , based on E , if $\text{CET} \models \bar{\Theta}E_\Delta \sigma$.

If a goal formula $\bar{\Theta}E$ itself is also a pre-answer formula and no answer can be extracted from it, then the proof of the initial goal failed. It is convenient to make this explicit.

Rule of answer extraction failure: Let $\bar{\Theta}E$ be a goal formula rewritten from some initial query such that $\bar{\Theta}E$ is also a pre-answer formula.

$$\bar{\Theta}E \rightarrow F \quad \text{if no answer can be extracted from } \bar{\Theta}E$$

We note that Proposition 1 can be extended with the addition of this new rule.

We shall show that our rewrite system with answers extracted this way is sound and complete. Before we do this, we show some examples.

Examples

Example 1 Let $\langle T, \emptyset, \emptyset \rangle$ be an abductive program, where T is $\{p(X) \leftrightarrow \neg r(X, Y); r(X, Y) \leftrightarrow X = a \wedge Y = b\}$. Clearly, $T \cup \text{CET} \models p(t)$, for any t in the language, which as we said in Section 2, includes at least a and b . We can show this using our rewrite system as follows:

$$p(V) \Rightarrow \exists Y \neg r(V, Y) \Rightarrow \exists Y (V \neq a \vee Y \neq b)$$

We get three pre-answer formulas from this: $\exists Y (V \neq a)$, $\exists Y (Y \neq b)$, and $\exists Y (V \neq a \vee Y \neq b)$. From the second one, we extract the answer $(\emptyset, \{V/t\})$ for any t .

On the other hand, we expect $\neg p(V)$ to be proved false.

$$\neg p(V) \Rightarrow \forall Y r(V, Y) \Rightarrow \forall Y (V = a \wedge Y = b) \Rightarrow F$$

This is because $\text{CET} \models \neg \forall Y (Y = b)$ as our language contains both a and b . This shows there is no t such that $\neg p(t)$ follows from $T \cup \text{CET}$. \square

This is a simple program. For more involved ones, there would be many variables and quantifiers, and it may become difficult to track whether they are universal ones or existential ones, and their nested order. To alleviate this, we propose an explicit representation of quantification for goal formulas using an annotated notation. Let $\Pi = Q_1 \xi_1 \dots Q_k \xi_k \Phi$ be a goal, where Φ is a formula with k variables ξ_1, \dots, ξ_k , and each Q_i is either \forall or \exists . In the construction of a proof by rewriting, we will write Π as Φ' , where Φ' is obtained from Φ by replacing each variable ξ_i by X_{Q_i} . For example, $\forall X \exists Y [f(X, X) = f(a, Y) \wedge p(Y)]$ can be represented by $f(X_{\forall}, X_{\forall}) = f(a, X_{\exists}) \wedge p(X_{\exists})$. In the following, we shall use these two notations, regular goal formulas and their annotated versions, interchangeably.

In our definition of unfolding, we push all variables to the outside, and to do this, we have to rename variables to avoid name conflicts. We now use an example to show that this is necessary. Otherwise, one would have to design more involved distribution rules.

Example 2 Consider the following program

$$\begin{aligned} p(X) &\leftarrow r(X), q(Y). \quad q(X) \leftarrow q(X). \quad q(X) \leftarrow s(X, Y). \\ r(a). &\quad s(a, b). \end{aligned}$$

and its completion

$$\begin{aligned} p(X) &\leftrightarrow r(X) \wedge \exists Y q(Y), & q(X) &\leftrightarrow q(X) \vee \exists Y s(X, Y), \\ r(X) &\leftrightarrow X = a, & s(X, Y) &\leftrightarrow X = a \wedge Y = b. \end{aligned}$$

It is easy to see that

$$p(a) \Rightarrow^* a = a \wedge q(X_{\exists_1}) \vee (a = a \wedge X_{\exists_1} = a \wedge X_{\exists_2} = b)$$

Since CET entails the answer formula $\exists X_1, X_2 (a = a \wedge X_1 = a \wedge X_2 = b)$, the query is answered positively. However, if we do not move the quantifiers to the outside, we would get:

$$\begin{aligned} p(a) &\leftrightarrow r(a) \wedge \exists Y q(Y) \leftrightarrow a = a \wedge \exists Y (q(Y) \vee \exists Z s(Y, Z)) \\ &\leftrightarrow a = a \wedge \exists Y (q(Y) \vee \exists Z (Y = a \wedge Z = b)). \end{aligned}$$

To extract answers from the above, we would need to have rules that can distribute quantifications to get, for example, the answer formula $a = a \wedge \exists Y, Z (Y = a \wedge Z = b)$. \square

We mentioned earlier that for negative literals, the use of the formula in (iii), instead of the formula in (i) or (ii), can simplify the process of answer extraction. We now illustrate this by the following example.

Example 3 Consider T that consists of

$$even(V) \leftrightarrow [V = 0] \vee [V = s(s(Y))], even(Y)$$

We expect query $\neg even(X)$ to be proved with X bound to $s(0)$, $s^3(0)$, and so on.

$$\begin{aligned} &\neg even(X) \\ \Rightarrow &X \neq 0 \wedge [X \neq s(s(Y_{\exists_1}))] \vee [X = s(s(Y_{\exists_2})) \wedge \neg even(Y_{\exists_2})] \\ \Rightarrow &[X \neq 0, X \neq s(s(Y_{\exists_1}))] \vee [X \neq 0, X = s(s(Y_{\exists_2})), \neg even(Y_{\exists_2})] \\ \Rightarrow^* &[X \neq 0, X \neq s(s(Y_{\exists_1}))] \vee \\ &[X \neq 0, X = s(s(Y_{\exists_2})), Y_{\exists_2} \neq 0, Y_{\exists_2} \neq s(s(Z_{\exists_3}))] \vee \\ &[X \neq 0, X = s(s(Y_{\exists_2})), Y_{\exists_2} \neq 0, Y_{\exists_2} = s(s(Z_{\exists_3})), \neg even(Z_{\exists_3})] \end{aligned}$$

In the last goal formula, an answer $X = s(0)$ is extracted from the first disjunct. By definition, this can be expressed formally as

$$CET \models \forall Y [X \neq 0, X \neq s(s(Y))] \{X/s(0)\}$$

Similarly, $X = s(s(s(0)))$ is extracted from the second disjunct above.

We mentioned that the language can have symbols not in the original program. Suppose now there is one more constant a in the language. Then we can extract additional answers, $X = a$ and $X = s(a)$, from the first disjunct above, and $X = s(s(a))$ and $X = s(s(s(a)))$ from the second. Indeed, it is easy to see that $T \cup CET \models \neg even(t)$, for $t = a, s(a), s(s(a))$, etc. Thus the function “ s ” (successor) does not have its intended meaning when applied to terms formed by the new constant a . This can be fixed by adding $even(a)$ (or $even(s(a))$ if one wants a to represent an odd number) to the program.

Now suppose we use the formula in (i) to rewrite a negative literal in the goal formula, we will get

$$\begin{aligned} &\neg even(X) \\ \leftrightarrow &\forall Y [X \neq 0, X \neq s(s(Y))] \vee [X \neq 0, \neg even(Y)] \\ \leftrightarrow &\forall Y \exists Z [X \neq 0, X \neq s(s(Y))] \vee [X \neq 0, Y \neq 0, Y \neq s(s(Z))] \\ &\vee [X \neq 0, Y \neq 0, \neg even(Z)] \end{aligned}$$

For example, the answer $X = s(s(s(0)))$ cannot be extracted from the second disjunct alone, as the answer formula $\forall Y \exists Z [s(s(s(0)) \neq 0, Y \neq 0, Y \neq s(s(Z))]$ is not entailed by CET. But this answer can be extracted from the first two disjuncts together, namely,

$$CET \models \forall Y \exists Z [s(s(s(0))) \neq 0, s(s(s(0))) \neq s(s(Y))] \vee [s(s(s(0))) \neq 0, Y \neq 0, Y \neq s(s(Z))]$$

In this example, in order not to miss any answer, all answer formulas together must participate in answer extraction. Clearly, this is a much harder job than extracting answers from each disjunct independently. \square

We now show some examples that have abducibles.

Example 4 Consider an abductive program and a goal. $T = \{q(X) \leftrightarrow ab(X); p(X, Y) \leftrightarrow X = f(V) \wedge Y = W\}$ $IC = \{[\neg ab(X) \vee p(X, Y)]\}$, $Ab = \{ab\}$, and the goal is $q(I)$.

$$\begin{aligned} &[q(I), [\neg ab(X_{\forall_1}) \vee p(X_{\forall_1}, X_{\forall_2})] \\ \Rightarrow &[ab(I), [\neg ab(X_{\forall_1}) \vee p(X_{\forall_1}, X_{\forall_2})] \\ \Rightarrow &[ab(I), [\neg ab(X_{\forall_1}) \vee [X_{\forall_1} = f(X_{\exists_3}), X_{\forall_2} = X_{\exists_4}]] \end{aligned}$$

This is a pre-answer formula. Since the conjunct $X_{\forall_2} = X_{\exists_4}$ is entailed by CET independently, it can be dropped. Denote the resulting formula by E . Let $\Delta = \{ab(f(a))\}$. Then,

$$E_{\Delta} = [I = f(a), [X_{\forall_1} \neq f(a) \vee X_{\forall_1} = f(X_{\exists_3})]]$$

Clearly, $(\Delta, \{I/f(a)\})$ is an answer to the goal. But $(\{ab(a)\}, \{I/a\})$ is not, since IC is not satisfied. \square

Example 5 Consider the faulty-lamp example of (Fung & Kowalski 1997). For simplicity, let us consider only one way for the lamp to be faulty. Suppose the abductive program is $\langle T, \emptyset, \{power_failure, empty\} \rangle$, where T is

$$\begin{aligned} &faulty_lamp \leftrightarrow power_failure(X) \wedge \neg backup(X) \\ &backup(X) \leftrightarrow battery(X, Y) \wedge \neg empty(Y) \\ &battery(X, Y) \leftrightarrow X = b \wedge Y = c \end{aligned}$$

The abbreviations used below should be clear.

$$\begin{aligned} fl &\Rightarrow [pf(X_{\exists_1}), \neg backup(X_{\exists_1})] \\ &\Rightarrow [pf(X_{\exists_1}), [\neg batt(X_{\exists_1}, X_{\forall_2}) \vee emp(X_{\forall_2})]] \\ &\Rightarrow [pf(X_{\exists_1}), [X_{\exists_1} \neq b \vee X_{\forall_2} \neq c \vee emp(X_{\forall_2})]] \\ &\Rightarrow [pf(X_{\exists_1}), X_{\exists_1} \neq b] \vee [pf(X_{\exists_1}), [X_{\forall_2} \neq c \vee emp(X_{\forall_2})]] \end{aligned}$$

The first disjunct gives answer $(\{pf(t)\}, \emptyset)$, for any $t \neq b$; for the second, we have $\Delta = \{pf(t), emp(c)\}$, for any t in our language. \square

Soundness and Completeness

Theorem 1 (Soundness)

Let (T, IC, Ab) be an abductive program and G a goal.

(1) Suppose rewriting from $\forall \bar{X}. G \wedge IC$ generates $\Theta(E \vee \Phi)$ such that (Δ, σ) is extracted as an answer to G , based on E . Then,

$$T \cup comp(\Delta) \cup CET \models G\sigma \wedge \forall \bar{X} IC.$$

(2) If rewriting from $\forall \bar{X}. G \wedge IC$ generates F , then $T \cup CET \cup \forall \bar{X} IC \models \neg \exists G$.

Proof. (1) From Proposition 1, our assumption that $\bar{X} \cap V(G) = \emptyset$, and that $\text{comp}(\Delta) \models E \leftrightarrow E_\Delta$, we have $T \cup \text{comp}(\Delta) \cup \text{CET} \models G\sigma \wedge \forall \bar{X}IC \leftrightarrow \bar{\Theta}(E_\Delta \vee \Phi)\sigma$. Since $\text{CET} \models \bar{\Theta}E_\Delta\sigma$, and $\models \bar{\Theta}E_\Delta\sigma \rightarrow \bar{\Theta}(E_\Delta \vee \Phi)\sigma$, we have $T \cup \text{comp}(\Delta) \cup \text{CET} \models G\sigma \wedge \forall \bar{X}IC$.

(2) From $\text{comp}(P) \cup \text{CET} \models \forall \bar{X}(G \wedge IC) \leftrightarrow F$, we get $\text{comp}(P) \cup \text{CET} \cup \forall \bar{X}IC \models \neg G$ by our assumption that $\bar{X} \cap V(G) = \emptyset$. \forall -introduction then leads to $\text{comp}(P) \cup \text{CET} \cup IC \models \forall \neg G$. \square

Theorem 2 (Completeness) *Let $\langle T, IC, Ab \rangle$ be an abductive program and G a goal. Suppose (Δ, σ) is an answer to G under 3-valued logic: $T \cup \text{comp}(\Delta) \cup \text{CET} \models_3 G\sigma \wedge \forall \bar{X}IC$. Then, there is a derivation from $\forall \bar{X}.G \wedge IC$ to a goal formula $\bar{\Theta}(E \vee \Phi)$, where E is not further reducible by any defined predicates, such that an answer (Δ', σ') can be extracted, based on E , where Δ' is a subset of Δ and σ' is more general than σ .*

We can prove the theorem under a *fair* selection rule, using the well-known result of (Kunen 1987). The difficulty for the 2-valued completion semantics is known to be caused by loops, e.g., with $T = \{p \leftrightarrow \neg p\}$, any goal would follow in 2-valued logic. But in 3-valued logic, T has a model where p is *undefined*, hence nontermination of repeated unfolding would not result in loss of completeness.

Related Work and Discussion

Our approach can be viewed as a generalization of the iff procedure of (Fung & Kowalski 1997), which comes with a variety of additional inference rules that can be used to make answer extraction more constructive for the class of allowed programs and goals. An interesting fact is that our rewrite system without these extra rules does not lose the completeness even for arbitrary programs and goals.

One can solve the floundering query problem by constructive negation (CN) (see, e.g., (Chan 1988; Drabent 1993; Stuckey 1995)). All of these approaches are based on the completion semantics, and have similar soundness and completeness results. Briefly, our approach is considerably simpler, as it builds reasoning directly on completed definitions on a *flat* structure, without relying on nested finite failure trees as needed in CN.

In relating to the SLDNFA abductive procedure (Denecker & De Schreye 1998), we note that the completeness of their procedure depends on the condition that derivations terminate (due to finite failure), while ours does not, simply because there is no notion of finite failure in our approach.

One advantage of the completion semantics is that iff definitions are first order formulas where theoremhood is semi-decidable so that a complete procedure is possible. It is also known that if a program has no loops, the completion semantics coincides with the answer set semantics (Gelfond & Lifschitz 1988). All of the example nonground programs in this paper have no loops, so our rewrite system proves goals also for the answer set semantics. E.g., the theory T in Example 1 corresponds to the following program P :

$$p(X) \leftarrow \text{not } r(X, Y). \quad r(a, b).$$

P has a unique answer set that contains both $p(a)$ and $p(b)$, and in fact $p(t)$, for any t in the language. As shown in Example 1, these are precisely what are proved from the goal $p(V)$ by our rewrite system. Note that Prolog would fail the goal $p(V)$, due to its naive implementation of finite failure.

Further work is needed to address three issues arising from this work. The first concerns the possibility of reducing dependencies of disjuncts in a goal formula $\bar{\Theta}(E \vee \Phi)$, so that $\bar{\Theta}E$ and $\bar{\Theta}\Phi$ may be processed independently. We have shown that in unfolding negative literals, certain universal variables can indeed be converted to existential ones, thus reducing the chances of shared universal variables. The second issue is related to the computational properties of solving quantified equations and disequations and the design of an efficient reasoner for it. Finally, rewrite strategies will be an important issue to be addressed in future work.

Acknowledgements Fangzhen Lin's work was supported in part by HK RGC CERG 616806 and by NSFC grant 60496322.

References

- Chan, D. 1988. Constructive negation based on the completed database. In *Proc. ICLP/SLP*, 111–125.
- Clark, K. 1978. Negation as failure. *Logics and Databases* 293–322.
- Console, L.; Theseider, D.; and Porasso, P. 1991. On the relationship between abduction and deduction. *J. Logic Programming* 2(5):661–690.
- Denecker, M., and De Schreye, D. 1998. SLDNFA: an abductive procedure for normal abductive programs. *J. Logic Prog.* 34(2):111–167.
- Drabent, W. 1993. SLS-resolution without floundering. In *Proc. LPNMR '93*, 82–98.
- Endriss, U.; Mancarella, P.; Sadri, F.; Terreni, G.; and Toni, F. 2004. The CIFF procedure for abductive logic programming with constraints. In *JELIA'04*.
- Eshghi, K., and Kowalski, R. 1989. Abduction compared with negation by failure. In *Proc. 6th ICLP*, 234–254.
- Fung, T., and Kowalski, R. 1997. The iff proof procedure for abductive logic programming. *J. Logic Programming* 33(2):151–164.
- Gelfond, M., and Lifschitz, V. 1988. The stable model semantics for logic programming. In *Proc. ICLP/SLP*.
- Kakas, A.; Kowalski, R.; and Toni, F. 1995. The role of abduction in logic programming. In *Handbook of Logic in Artificial Intelligence and Logic Prog.* Oxford University.
- Kakas, A.; Michael, A.; and Mourlas, C. 2000. ACLP: abductive constraint logic programming. *J. Logic Programming* 44:129–177.
- Kunen, K. 1987. Negation in logic programming. *J. Logic Programming* 4(3):289–308.
- Lin, F., and You, J. 2002. Abduction in logic programming: a new definition and an abductive procedure based on rewriting. *Artificial Intelligence* 140(1/2):175–205.
- Stuckey, P. 1995. Negation and constraint logic programming. *Inf. Comput.* 118(1):12–33.