Computing extended abduction through transaction programs

Katsumi Inoue [a] and Chiaki Sakama [b]

[a] *Department of Electrical and Electronics Engineering*
Kobe University
Rokkodai, Nada-ku, Kobe 657-8501, Japan
E-mail: inoue@eedept.kobe-u.ac.jp

[b] *Department of Computer and Communication Sciences*
Wakayama University
Sakaedani, Wakayama 640-8510, Japan
E-mail: sakama@sys.wakayama-u.ac.jp

To explain observations from nonmonotonic background theories, one often needs *removal* of some hypotheses as well as addition of other hypotheses. Moreover, some observations should *not* be explained, while some are to be explained. In order to formalize these situations, *extended abduction* was introduced by Inoue and Sakama (1995) to generalize traditional abduction in the sense that it can compute negative explanations by removing hypotheses and anti-explanations to unexplain negative observations. In this paper, we propose a computational mechanism for extended abduction. When a background theory is written in a normal logic program, we introduce its *transaction program* for computing extended abduction. A transaction program is a set of non-deterministic production rules that declaratively specify addition and deletion of abductive hypotheses. Abductive explanations are then computed by the fixpoint of a transaction program using a bottom-up model generation procedure. The correctness of the proposed procedure is shown for the class of acyclic covered abductive logic programs. In the context of deductive databases, a transaction program provides a declarative specification of database update.

Keywords: abduction, nonmonotonic reasoning, database update

1. Introduction

1.1. Motivation and Background

Abduction is inference to best explanations, and has recently been recognized as a very important form of reasoning in both AI [25] and logic programming [22]. A traditional formalization of abduction [28,25] defines an *explanation* of a given observation as a set of hypotheses which, together with the background theory, logically entails the observation. Formally, given a background knowledge base K and an observation G, a traditional framework of abduction computes a

set E of hypotheses satisfying

$$K \cup E \models G$$

where $K \cup E$ is consistent. In [28,25], the background theory is a first-order theory which is monotonic.

However, the above framework of abduction is not sufficient when a background knowledge base K is *nonmonotonic*. For example, consider the knowledge base written in nonmonotonic logic programming:

$$\begin{aligned} K : \; & flies(x) \leftarrow bird(x), not\ ab(x), \\ & ab(x) \leftarrow broken\text{-}wing(x), \\ & bird(tweety) \leftarrow , \\ & bird(opus) \leftarrow , \\ & broken\text{-}wing(tweety) \leftarrow . \end{aligned}$$

If we observe that *tweety* flies, there is a good reason to assume that the wound has already healed. Then, removing the fact $broken\text{-}wing(tweety)$ from the knowledge base explains the observation $flies(tweety)$.

The traditional abduction cannot characterize such a situation. That is, abduction introduces hypotheses to a knowledge base, but once they are included, any hypothesis cannot be removed from the knowledge base. To cope with this problem, Inoue and Sakama [19] introduced the notion of "negative explanations". Given a background knowledge base K and an observation G, a set F of hypotheses is called a *negative explanation* of G if

$$K \setminus F \models G$$

where $K \setminus F$ is consistent. An explanation E satisfying $K \cup E \models G$ is then called a *positive explanation*.

On the other hand, suppose that we later notice that *opus* does not fly any more. Since $flies(opus)$ is entailed by K, we now have to revise the knowledge base to block the derivation of $flies(opus)$ by assuming, for instance, $broken\text{-}wing(opus)$. This situation is characterized by the notion of "anti-explanations" which are used to *un*explain observations [19]. Given a background knowledge base K and an observation G, a set E of hypotheses is called a *(positive) anti-explanation* of G if

$$K \cup E \not\models G,$$

and a set F of hypotheses is called a *negative anti-explanation* of G if

$$K \setminus F \not\models G.$$

These extensions of traditional abduction are called *extended abduction* in this paper. Extended abduction is particularly useful when we consider nonmonotonic theories as background knowledge bases. This is justifiable by the following

three reasons. Firstly, the introduction of negative explanations is natural in nonmonotonic theories. Since addition of new formulas into axioms can cause deletion of previous theorems in nonmonotonic reasoning, deletion of such added formulas can revive the deleted theorems. That is, it is often the case that deletion of formulas introduces new formulas in nonmonotonic reasoning. Hence, positive and negative explanations play a complementary role in accounting for an observation in nonmonotonic theories. Secondly, the introduction of anti-explanations is necessary when one wants to account for *negative observations* even in monotonic theories. In this respect, traditional abduction is concerned with explaining *positive observations* only. Negative observations are often perceived in real-life situations. For instance, many *inductive concept-learning* systems consider both positive and negative examples to construct a hypothetical theory. In this sense, the notion of anti-explanations plays a dual role to explanations. Thirdly, extended abduction not only enhances reasoning ability of traditional abduction, but is essential to theories of *nonmonotonic theory change* and *abductive theory revision* [19]. Useful applications of extended abduction include *view update* in deductive databases, *contradiction removal* as well as *diagnosis* and *repair*. For example, model-based diagnosis [29] constructs a diagnostic hypothesis by identifying a minimal set of fault components Δ from the set of system components $COMP$ such that

$$H = \{Ab(c) \mid c \in \Delta\} \cup \{\neg Ab(c) \mid c \in COMP \setminus \Delta\}$$

is consistent with the system description and observations, where $Ab(c)$ represents that a component c is faulty. Suppose that, after repairing some components Δ' in Δ, we get new observations. Then,

$$H' = (H \cup E) \setminus F$$

with $E = \{\neg Ab(c) \mid c \in \Delta'\}$ and $F = \{Ab(c) \mid c \in \Delta'\}$ becomes a new diagnosis. Here, positive and negative explanations are necessary at the same time. Recently, Buccafurri *et al.* [6] apply extended abduction to formalize a system repair problem in the context of model checking, which is a successful technique for the verification of concurrent programs and protocols.

Extended abduction is thus useful for many applications, and is really an extension of traditional abduction whose usefulness has already been recognized too. However, no proof procedure for extended abduction has been proposed so far, and it is anticipated that its computational method extends a previous procedure for traditional abduction in a natural way.

1.2. Purpose and Outline

In this paper, we study the computational aspect of extended abduction. We consider knowledge bases written in normal logic programs. Firstly, we introduce a program transformation which produces a *transaction program* from a given

logic program using the *completion* [7] of the program. A transaction program is a kind of disjunctive logic program, and declaratively specifies relationships among literals expressing requests of addition or deletion of abductive hypotheses. Since abduction is inference to causes from effects, its computation can be done by tracing rules in a program backward, that is, following only-if directions of the completion from a given observation. In fact, abductive computation using Clark completion has been proposed in [8,14] for traditional abduction. Our transaction programs is a natural extension of such previous work, and can be used to compute not only positive explanations but also negative explanations and (positive or negative) anti-explanations.

Next, we show a procedure to compute abductive explanations using transaction programs. Abductive explanations of an observation from a given program can be computed by the fixpoint of the transaction program using bottom-up model generation procedures like [5,20,4]. In this sense, the proposed procedure can be considered as an extension of a bottom-up abductive procedure by Inoue and Sakama [20] where only positive observations can be explained by adding hypotheses only. However, the transformation of abductive logic programs to disjunctive programs in this paper is entirely different from the one in [20], and the notion of *marking* literals is also introduced in this paper to make literals rewritten in bottom-up procedures. The correctness of the proposed procedure is shown for the class of acyclic programs in the ground case and for the class of covered acyclic programs in the non-ground case. In a special case, the correctness is also guaranteed for non-acyclic programs.

Finally, we present applications of transaction programs to database update. In the context of deductive databases, a transaction program provides a declarative specification of database update. In a special case, a transaction program gives a *revision program* in the sense of Marek and Truszczynski [27].

This is an extended version of [21]. The rest of this paper is organized as follows. In Section 2, we outline our extended abductive framework. In Section 3, we introduce a transformation of abductive logic programs into transaction programs, and present a fixpoint computation for extended abduction. Section 4 addresses comparisons with related work, and Section 5 concludes the paper.

2. Abductive framework

An abductive framework considered in this paper is defined as follows.

A *normal logic program* (NLP) is a finite set of rules of the form:

$$A_0 \leftarrow A_1, \ldots, A_m, \mathit{not}\, A_{m+1}, \ldots, \mathit{not}\, A_n \tag{1}$$

($n \geq m \geq 0$), where each A_i is an atom and *not* denotes *negation as failure*. The left-hand side of a rule is the *head*, and the right-hand side is the *body*. We allow a rule with an empty head, which is called an *integrity constraint*. In this paper,

the empty head of any integrity constraint is represented by the special atom $\bot$. A rule with an empty body ($n = 0$) is called a *fact*. Each fact $A \leftarrow$ is identified with the atom A.

A program (rule, atom, term) is *ground* if it contains no variable. An NLP P is *acyclic* if there is a *level mapping* from the ground atoms of P to the natural numbers, such that for any ground rule (1) from P, the level of A_0 is higher than the level of every A_i ($i = 1, \ldots, n$) [3]. P is *Horn* if no rule in P contains negation as failure, that is, $m = n$ holds for every rule of the form (1) in P. P is *definite* if it is a Horn program without integrity constraints.

The *stable model semantics* for NLPs [15] is given as follows. Given any NLP P and a set M of ground atoms, the ground Horn NLP P^M is defined as follows: a rule

$$A_0 \leftarrow A_1, \ldots, A_m$$

is in P^M iff there is a ground rule of the form (1) from P such that $\{A_{m+1}, \ldots, A_n\} \cap M = \emptyset$. Note that P^M is a possibly infinite set of *not*-free ground rules. Then, P^M has the unique minimal model, i.e., the least model. If the least model of P^M is identical to M, M is called a *stable model* of P. A stable model is *consistent* if it does not contain $\bot$, that is, it satisfies every integrity constraint. An NLP is *consistent* if it has a consistent stable model. For a consistent NLP P and an atom L, P *entails* L (written $P \models L$) if L is included in every stable model of P; otherwise $P \not\models L$. Note that any acyclic program has a unique stable model [15], which coincides with the *well-founded model* [32].

An *abductive logic program* (ALP) is defined as a pair $\langle P, \mathcal{A} \rangle$, where P is an NLP and $\mathcal{A}$ is a set of atoms called *abducibles*. Any instance A of an element from $\mathcal{A}$ is also called an *abducible* and is written as $A \in \mathcal{A}$. Without loss of generality, we assume that any rule (1) having an abducible in its head ($A_0 \in \mathcal{A}$) is always a fact.[1] An ALP $\langle P, \mathcal{A} \rangle$ is *acyclic* (resp. *Horn*, *ground*) if P is acyclic (resp. Horn, ground).

Definition 2.1. Let $\langle P, \mathcal{A} \rangle$ be an ALP, and G a ground atom. A pair (E, F) is called an *explanation* (resp. *anti-explanation*) of a *positive* (resp. *negative*) *observation* G wrt $\langle P, \mathcal{A} \rangle$ if

1. $(P \cup E) \setminus F \models G$ (resp. $(P \cup E) \setminus F \not\models G$),
2. $(P \cup E) \setminus F$ is consistent, and
3. E and F are sets of ground instances from $\mathcal{A}$.

[1] If there is a rule $A \leftarrow \Gamma$ with an abducible A and a non-empty body Γ, then A is made a non-abducible by introducing a rule $A \leftarrow A'$ with a new abducible A'. This translation of abducibles is similar to the one used in [22], but here this transformation is not applied to facts while it is also employed to fact abducibles in [22].

Positive or negative observation is also simply called an *observation*. For an (anti-)explanation (E, F) of an observation, E is also called a *positive (anti-)explanation* if $F = \emptyset$; F is called a *negative (anti-)explanation* if $E = \emptyset$; and (E, F) is called a *mixed (anti-)explanation* if $E \neq \emptyset$ and $F \neq \emptyset$.

An (anti-)explanation (E, F) of G is *minimal* if for any (anti-)explanation (E', F') of G, $E' \subseteq E$ and $F' \subseteq F$ imply $E' = E$ and $F' = F$.

Definition 2.1 is a restricted version of more general framework of extended abduction given in [19] where autoepistemic theories are allowed in both P and $\mathcal{A}$. While we consider an NLP P in an ALP $\langle P, \mathcal{A} \rangle$, most definitions and results in this paper are also directly applicable to abductive logic programs containing classical negation (see Section 3.2).

Since there are a number of possible explanations of a given observations in general, some best explanations should be selected from them. The minimality criterion is a syntactical form of Occam's razor, and hence minimal explanations are usually of particular interest. When nonmonotonic theory change is considered as in [19], a minimal (anti-)explanation (E, F) offers a tradeoff between *minimal reduction* F of the background theory P and *minimal augmentation* E of the newly added hypotheses from $\mathcal{A}$. In this sense, each minimal (anti-)explanation accomplishes a *minimal change* of the theory. We thus often consider minimal (anti-)explanations as our best (anti-)explanations in this paper. Note that $E \cap F = \emptyset$ holds for any minimal (anti-)explanation (E, F).

Example 2.2. Consider the introductory program K in Section 1. Let $\langle K, \mathcal{A} \rangle$ be an ALP where $\mathcal{A} = \{ \mathit{broken\text{-}wing}(x) \}$. Then, $G_1 = \mathit{flies}(\mathit{tweety})$ has the minimal explanation $(E, F) = (\emptyset, \{\mathit{broken\text{-}wing}(\mathit{tweety})\})$, while $G_2 = \mathit{flies}(\mathit{opus})$ has the minimal anti-explanation $(\{\mathit{broken\text{-}wing}(\mathit{opus})\}, \emptyset)$.

3. Computing extended abduction

In this section, we provide a method of computing extended abduction. Section 3.1 introduces the notion of *transaction programs*, which declaratively specify the necessary change of hypotheses for observations to hold. Then, Section 3.2 describes a procedure to compute extended abduction by rewriting and marking literals through rules in a transaction program. In Sections 3.1 and 3.2, programs are assumed to be ground, and results are extended to non-ground programs in Section 3.3.

3.1. Transaction program

We first introduce the notion of *transaction programs*, which are obtained by transforming abductive logic programs. This transformation is motivated to

get rules for what must be in the knowledge base under consideration and what must be outside it for any observation to hold.

For any atom A, formulas $in(A)$ and $out(A)$ are called *T-literals* (or *literal* for short). A *transaction program* consists of *(context-free) productions* of the form:

$$L \to \alpha_1 \mid \ldots \mid \alpha_k, \tag{2}$$

where L is a literal and α_i is a sequence of literals.[2] Here, the vertical line "|" is read "or" meaning a disjunction, while a sequence means a conjunction of literals. In a production of the form (2), the left-hand side L is called the *antecedent* and the right-hand side $\alpha_1 \mid \ldots \mid \alpha_k$ is called the *consequent* of the production. Note that each α_i can be a conjunction in a production of the form (2), but such a production can be equivalently written in multiple productions each of which has a disjunction of literals in its consequent and the common antecedent L. Namely, a production of the form

$$L \to L_{1,1}, \ldots, L_{1,n_1} \mid \ldots \mid L_{k,1}, \ldots, L_{k,n_k},$$

where L and $L_{i,j}$'s are literals, stands for the $\prod_{j=1}^{k} n_j$ productions

$$L \to L_{1,i_1} \mid \ldots \mid L_{k,i_k}$$

for every $i_1 = 1, \ldots, n_1, \ \ldots, \ i_k = 1, \ldots, n_k$. In this sense, a set of productions can be seen as a *disjunctive logic program* [16] (without negation as failure) by regarding each T-literal as a propositional atom.

Definition 3.1. Let $\langle P, \mathcal{A} \rangle$ be a ground ALP. Then, the *transaction program τP of P (relative to $\mathcal{A}$)* is defined as follows.

1. Let H be a non-abducible atom ($H \notin \mathcal{A}$) and

$$H \leftarrow B_1,$$
$$\vdots$$
$$H \leftarrow B_k,$$

be all rules in P with H in the head, where

$$B_i = A_1, \ldots, A_{m_i}, not\, C_1, \ldots, not\, C_{n_i}$$

($1 \leq i \leq k, \ m_i \geq 0, \ n_i \geq 0$). Then, these rules are transformed to the following productions:

$$\begin{aligned} R_{in} &: \ in(H) \to in(B_1) \mid \ldots \mid in(B_k), \\ R_{out} &: out(H) \to out(B_1), \end{aligned} \tag{3}$$

[2] In non-ground transaction programs, productions may contain equalities or inequalities in their antecedents or consequents (see Definition 3.12).

$$\vdots \tag{4}$$
$$out(H) \rightarrow out(B_k),$$

where for $i = 1, \ldots, k$, $in(B_i)$ and $out(B_i)$ are defined as follows. If B_i is empty, $in(B_i) = \varepsilon$; otherwise $in(B_i)$ is:

$$in(A_1), \ldots, in(A_{m_i}),\ out(C_1), \ldots, out(C_{n_i}).$$

If B_i is empty, $out(B_i) = false$; else $out(B_i)$ is:

$$out(A_1) \mid \ldots \mid out(A_{m_i}) \mid in(C_1) \mid \ldots \mid in(C_{n_i}).$$

When $H = \bot$ (integrity constrains), only R_{out} is included in τP.

2. When $H \leftarrow$ is in P for $H \in \mathcal{A}$ (note in this case that there is no rule with H in the head and with a non-empty body by the assumption [footnote 1]), it is transformed to

$$R_{in}: \quad in(H) \rightarrow \varepsilon,$$

and we define that R_{out} is empty.

3. Let A be any atom that does not appear in the head of any rule in P. For every such atom A, introduce the production:

$$out(A) \rightarrow \varepsilon \tag{5}$$

to τP. Moreover, when A is a non-abducible ($A \notin \mathcal{A}$), τP also includes:

$$in(A) \rightarrow false. \tag{6}$$

Each production generated in the first step specifies abductive specifications such that to explain H, one of $B_1, \ldots, B_k$ must be explained (3), while to unexplain H, every $B_1, \ldots, B_k$ must be unexplained (4). The meanings of $in(B_i)$ and $out(B_i)$ are obvious, that is, introduction and removal of B_i respectively, which correspond to explaining/unexplaining B_i. When B_i is empty, there is nothing to be introduced and $in(B_i) = \varepsilon$. Here, the empty string ε is logically equivalent to $true$ and $\{\varepsilon\} \cup S$ is identical to S for any set S. On the other hand, $out(B_i) = false$ means that removing a non-abducible H with the empty body is impossible. Note that, for integrity constraints, the removal of $\bot$ means satisfaction of all constraints. In the second step, when an abducible H is in P, nothing is required to add H, while no further transaction is requested to remove H. In the third step, an additional production of the form (5) means that the removal of A implies no requirement if P has no rule with A in the head, while a production of the form (6) means that the addition of A is impossible.

Example 3.2. Let $\langle P, \mathcal{A} \rangle$ be an ALP, where

$$\begin{aligned} P : & p \leftarrow q, not\, a, \\ & p \leftarrow b, not\, r, \\ & q \leftarrow not\, c, \\ & r \leftarrow d, \\ & c \leftarrow, \\ & d \leftarrow, \\ \mathcal{A} : & a,\ b,\ c,\ d. \end{aligned}$$

Then, τP becomes

$$\begin{aligned} & in(p) \rightarrow in(q), out(a) \mid in(b), out(r), \\ & out(p) \rightarrow out(q) \mid in(a), \\ & out(p) \rightarrow out(b) \mid in(r), \\ & in(q) \rightarrow out(c), \\ & out(q) \rightarrow in(c), \\ & in(r) \rightarrow in(d), \\ & out(r) \rightarrow out(d), \\ & in(c) \rightarrow \varepsilon, \\ & in(d) \rightarrow \varepsilon, \\ & out(a) \rightarrow \varepsilon, \\ & out(b) \rightarrow \varepsilon. \end{aligned}$$

3.2. Fixpoint computation

For computing extended abduction, we consider fixpoint computation of transaction programs. The $\mathbf{T}_{\tau P}$ *fixpoint operator* which we define next is similar to that given by Inoue and Sakama [20] for disjunctive logic programs. The only difference between the operator below and that in [20] lies in the notion of *marking*. Here, we view each production also as a *rewriting rule*. For any literal L in the antecedent of a production, once L is rewritten it is marked as $L*$. In this way, it is recognized that the requirement of addition/deletion of L has been translated into requests of another atoms' addition/deletion.

Formally, for T-literals $in(A)$ and $out(A)$, formulas $in(A)*$ and $out(A)*$ are also called *T-literals* (or *literals* for short). T-literals with the "$*$" symbol are called *marked literals*, while other literals are *unmarked*. Given a transaction program τP, let $\mathcal{B}$ be the set of ground T-literals in the language of τP, i.e., the Herbrand base. A *transaction* is a subset of $\mathcal{B}$, i.e., an Herbrand interpretation. Each transaction specifies which atoms should be added or deleted. A transaction

S is *coherent* if for any atom A, S does not include any pair of $\{in(A), out(A)\}$, $\{in(A)*, out(A)\}$, $\{in(A), out(A)*\}$ and $\{in(A)*, out(A)*\}$.

A transaction S *satisfies* a T-literal L (written $S \models L$), where L is either $in(A)$ or $out(A)$, if either $L \in S$ or $L* \in S$ holds. For a conjunction of literals, we also write $S \models L_1, \ldots, L_n$ if $S \models L_i$ for every $i = 1, \ldots, n$. A transaction S *satisfies* a production $(L \rightarrow \alpha_1 \mid \ldots \mid \alpha_k)$ if $S \models L$ implies $S \models \alpha_i$ for some i $(1 \leq i \leq k)$.

Now, we define a mapping over sets of transactions. For this purpose, the following two operations, *marking* $(*)$ and *filtering* $(\overset{*}{-})$, are introduced. For transactions I and J,

$$I * J = (I \setminus J) \cup \{L* \mid L \in I \cap J\},$$
$$I \overset{*}{-} J = \{L \in I \mid L \notin J \text{ and } L* \notin J\}.$$

Intuitively, $I * J$ denotes that each literal in J is marked in I, and $I \overset{*}{-} J$ means that a literal in I is removed from I if it is included in J in either marked or unmarked form. Also, for a conjunction of literals $F = L_1, \ldots, L_m$, we denote the set of its conjuncts as $\{F\} = \{L_1, \ldots, L_m\}$.

Definition 3.3. Let τP be a ground transaction program, $\mathbf{I}$ a set of transactions. Then the mapping $\mathbf{T}_{\tau P} : 2^{2^{\mathcal{B}}} \rightarrow 2^{2^{\mathcal{B}}}$ is defined as

$$\mathbf{T}_{\tau P}(\mathbf{I}) = \bigcup_{I \in \mathbf{I}} \{\, J \in T_{\tau P}(I) \mid J \text{ is coherent} \,\},$$

where the mapping $T_{\tau P} : 2^{\mathcal{B}} \rightarrow 2^{2^{\mathcal{B}}}$ is defined as

$$T_{\tau P}(I) = \begin{cases} \emptyset\,, & \text{if } I \models L \text{ for some production} \\ & \quad (L \rightarrow false) \text{ in } P\,; \\ \{\ (I * J) \cup (K \overset{*}{-} I) \ \mid & \\ \qquad J = \{L_i \mid R_i \in \mathbf{R}\} \text{ and } K = \bigcup_{R_i \in \mathbf{R}} \{\alpha^i\}, & \\ \quad \text{where } \mathbf{R} \text{ is the set of productions of the form:} & \\ \qquad R_i : (L_i \rightarrow F_1 \mid \ldots \mid F_{k_i}) & \\ \quad \text{in } P \text{ such that } I \models L_i \text{ and that} & \\ \qquad \{F_j\} \overset{*}{-} I \neq \emptyset \text{ for every } j = 1, \ldots, k_i, & \\ \quad \text{and } \alpha^i \text{ is one of } F_1, \ldots, F_{k_i} \,\}\,, & \text{otherwise}\,. \end{cases}$$

In particular, $\mathbf{T}_{\tau P}(\emptyset) = \emptyset$.

The intuitive reading of Definition 3.3 is as follows. If a transaction I does not satisfy a production with $false$ in the consequent, then I is rejected. Else, if there is a production R_i that is not satisfied by I (i.e., I satisfies the antecedent of R_i but does not satisfy the consequent of R_i), then I is expanded by adding a single disjunct α^i in the consequent for every such R_i, so that R_i is satisfied

by I. This expansion of I with K is done in every possible way, i.e., α^i can be any disjunct from $F_1 \mid \ldots \mid F_{k_i}$ for each R_i. In expanding I with α^i's, each antecedent of R_i in I is marked, then only literals in α^i filtered by I are added. In the case that R_i is $(L \rightarrow \varepsilon)$, L is simply marked and nothing is added with this production.

The ordinal powers of $\mathbf{T}_{\tau P}$ are defined as follows.

$$\begin{array}{lcl} \mathbf{T}_{\tau P} \uparrow 0 & = & \{\emptyset\}, \\ \mathbf{T}_{\tau P} \uparrow n+1 & = & \mathbf{T}_{\tau P}(\mathbf{T}_{\tau P} \uparrow n), \\ \mathbf{T}_{\tau P} \uparrow \omega & = & \bigcup_{\alpha<\omega} \bigcap_{\alpha \leq n<\omega} \mathbf{T}_{\tau P} \uparrow n, \end{array}$$

where n is a successor ordinal and ω is a limit ordinal. The above definition means that at the limit ordinal ω the closure retains transactions which are persistent in the preceding computation. That is, for any transaction I in $\mathbf{T}_{\tau P} \uparrow \omega$, there is an ordinal α smaller than ω such that, for every n $(\alpha \leq n < \omega)$, I is included in $\mathbf{T}_{\tau P} \uparrow n$. This closure definition is also used in [20] for computing minimal models of disjunctive logic programs. By the result in [20], $\mathbf{T}_{\tau P} \uparrow \omega$ becomes a fixpoint.

Example 3.4. Consider the transaction program τP in Example 3.2. Let $\tau P^{+p} = \tau P \cup \{ \rightarrow in(p) \}$. Then, the fixpoint closure of τP^{+p} becomes

$$\begin{array}{l} \mathbf{T}_{\tau P^{+p}} \uparrow 1 = \{ \{in(p)\} \}, \\ \mathbf{T}_{\tau P^{+p}} \uparrow 2 = \{ \{in(p)*, in(q), out(a)\}, \\ \qquad\qquad \{in(p)*, in(b), out(r)\} \}, \\ \mathbf{T}_{\tau P^{+p}} \uparrow 3 = \{ \{in(p)*, in(q)*, out(c), out(a)*\}, \\ \qquad\qquad \{in(p)*, in(b), out(r)*, out(d)\} \}, \\ \mathbf{T}_{\tau P^{+p}} \uparrow 4 = \mathbf{T}_{\tau P^{+p}} \uparrow 3. \end{array}$$

Thus, $\mathbf{T}_{\tau P^{+p}} \uparrow \omega = \mathbf{T}_{\tau P^{+p}} \uparrow 3$.

Note that the above fixpoint computation constructs minimal models in a *bottom-up* manner when productions are viewed as disjunctive clauses. *Model generation procedures* such as SATCHMO, MGTP, and Hyper-Tableaux used in [5,20,4] are similar to this computation mechanism. They perform hyperresolution to expand Herbrand interpretations and case-splitting on non-unit hyperresolvents. When a program is finite, function-free and range-restricted,[3] closure computation by these procedures stops in a finite step. For a ground transaction program, these conditions are always satisfied.

We are ready to compute extended abduction from the fixpoint of a transaction program. T-literals $in(A)/out(A)$ with $A \in \mathcal{A}$ are called *abducible literals*.

[3] A program P is *range-restricted* if for every rule in P, any variable appearing in the head has an occurrence in a positive atom in the body.

For a transaction S, the set of *unmarked abducible literals* in S is denoted as S°. For a set of transactions $\mathbf{I} \subseteq 2^{\mathcal{B}}$, let

$$min^\circ(\mathbf{I}) = \{\, S^\circ \mid S \in \mathbf{I} \text{ and there is no } S' \in \mathbf{I} \text{ such that } S'^\circ \subset S^\circ \,\}.$$

Let us denote a transaction program with an observation G to be (un)explained as:

$$\tau P^{+G} = \tau P \cup \{ \rightarrow in(G),\ \ \rightarrow out(\bot) \},$$
$$\tau P^{-G} = \tau P \cup \{ \rightarrow out(G),\ \ \rightarrow out(\bot) \}.$$

Here, $out(\bot)$ means that all integrity constraints should be satisfied. When there is no integrity constraint in P, $(\rightarrow out(\bot))$ is not added to τP^{+G} and τP^{-G}. Also, we write

$$IN(S) = \{A \mid in(A) \in S\},$$
$$OUT(S) = \{A \mid out(A) \in S\}.$$

Definition 3.5. For an acyclic ALP $\langle P, \mathcal{A} \rangle$ and a ground observation G, *abduction steps in* P are inductively defined as follows.

1. G is *explainable by* $(\emptyset, \emptyset)$ *at 0-step* if $G \in P$.
   G is *explainable by* $(\{G\}, \emptyset)$ *at 0-step* if $G \in \mathcal{A}$.
2. G is *unexplainable by* $(\emptyset, \emptyset)$ *at 0-step* if P has no ground rule from P with G in the head.
   G is *unexplainable by* $(\emptyset, \{G\})$ *at 0-step* if $G \in P$ and $G \in \mathcal{A}$.
3. G is *explainable by* (E, F) *at* $(s+1)$*-step* if there is a ground rule:
   $$G \leftarrow A_1, \ldots, A_m, \mathit{not}\, A_{m+1}, \ldots, \mathit{not}\, A_n$$
   from P such that (i) each A_i $(i = 1, \ldots, m)$ is explainable by (E_i, F_i) at l_i-step $(l_i \leq s)$ and each A_j $(j = m, \ldots, n)$ is unexplainable by (E_j, F_j) at l_j-step $(l_j \leq s)$, (ii) $E = \bigcup_{i=1}^{m} E_i$, $F = \bigcup_{j=1}^{n} F_j$, $E \cap F = \emptyset$, and (iii) $max(l_1, \ldots, l_m, l_{m+1}, \ldots, l_n) = s$.
4. G is *unexplainable by* (E, F) *at* $(s+1)$*-step* if for every ground rule r_k of the form:
   $$G \leftarrow A_1, \ldots, A_{m_k}, \mathit{not}\, A_{m_k+1}, \ldots, \mathit{not}\, A_{n_k}$$
   from P, (i) there is an atom A_{i_k} $(1 \leq i_k \leq n_k)$ such that A_{i_k} is unexplainable (when $1 \leq i_k \leq m_k$) or explainable (when $m_k + 1 \leq i_k \leq n_k$) by (E_k, F_k) at l_k-step $(l_k \leq s)$, (ii) $E = \bigcup_{r_k} E_k$, $F = \bigcup_{r_k} F_k$, $E \cap F = \emptyset$, and (iii) $max_{r_k} l_k = s$.

In particular, we say that G is *(un)provable in* P at s-step for some $s < \omega$ if G is (un)explainable by $(\emptyset, \emptyset)$ in P at s-step. Note that this definition of provability

(resp. unprovability) of G in P characterizes a positive (resp. negative) conclusion about G in P under the *well-founded semantics* [32] for acyclic NLPs. Namely, G is true/false under the well-founded model of P iff G is provable/unprovable in P. Since the well-founded model and the stable model coincide in acyclic NLPs [32], it holds that: for an acyclic NLP P, $P \models G$ (under the stable model semantics) iff G is provable in P at s-step.

Lemma 3.6. Let $\langle P, \mathcal{A} \rangle$ be an acyclic ALP, G a ground observation, and E, F are sets of ground instances from $\mathcal{A}$. Suppose that $(P \cup E) \setminus F$ is consistent.

(i) If G is (un)explainable by (E, F) in P at s-step for some $s < \omega$, then (E, F) is an (anti-)explanation of G wrt $\langle P, \mathcal{A} \rangle$.
(ii) If (E, F) is a minimal (anti-)explanation of G wrt $\langle P, \mathcal{A} \rangle$, then G is (un)explainable by (E, F) in P at s-step for some $s < \omega$.

Proof. The proof of (i) is obvious by induction on the abduction step in P.

(ii) Let (E, F) be a minimal explanation of G wrt $\langle P, \mathcal{A} \rangle$. Then, $(P \cup E) \setminus F \models G$. By the discussion above, G is provable in $(P \cup E) \setminus F$ at s-step for some $s < \omega$. In other words, G is explainable by $(\emptyset, \emptyset)$ in $(P \cup E) \setminus F$ at s-step. Then, G is explainable by $(\emptyset, F)$ in $P \cup E$ at s-step, because (E, F) is a minimal explanation of G and hence no $F' \subset F$ satisfies that G is explainable by $(\emptyset, F')$ in $P \cup E$. Then, G is explainable by (E, F) in P at s-step, because (E, F) is a minimal explanation of G and hence no $E' \subset E$ satisfies that G is explainable by (E', F) in P.

The proof for a minimal anti-explanation can be shown in a similar way. □

Theorem 3.7. Let $\langle P, \mathcal{A} \rangle$ be a ground acyclic ALP, and G a ground observation. Then,

(i) (E, F) is a minimal explanation of G wrt $\langle P, \mathcal{A} \rangle$ iff there is a transaction $S \in min^{\circ}(\mathbf{T}_{\tau P^{+G}} \uparrow \omega)$ such that $E = IN(S)$ and $F = OUT(S)$.
(ii) (E, F) is a minimal anti-explanation of G wrt $\langle P, \mathcal{A} \rangle$ iff there is a transaction $S \in min^{\circ}(\mathbf{T}_{\tau P^{-G}} \uparrow \omega)$ such that $E = IN(S)$ and $F = OUT(S)$.

Proof. We prove that G is (un)explainable by (E, F) at s-step for some $s < \omega$ iff there is a transaction S in $min^{\circ}(\mathbf{T}_{\tau P^{+G}} \uparrow \omega)$ (or $min^{\circ}(\mathbf{T}_{\tau P^{-G}} \uparrow \omega)$) such that $E = IN(S)$ and $F = OUT(S)$. This is proved by induction on the abduction step s. Then, the results (i) and (ii) follow by Lemma 3.6. Suppose first that $s = 0$.

(i) If G is explainable by (E, F) at 0-step, (E, F) is either $(\emptyset, \emptyset)$ or $(\{G\}, \emptyset)$. In the former case, $G \in P$ holds, and then $(in(G) \rightarrow \varepsilon)$ is in τP^{+G}. Hence, $min^{\circ}(\mathbf{T}_{\tau P^{+G}} \uparrow \omega) = \{\emptyset\}$, $E = IN(\emptyset)$ and $F = OUT(\emptyset)$. In the latter case, $G \in \mathcal{A}$ holds, and then $\{in(G)\} \in min^{\circ}(\mathbf{T}_{\tau P^{+G}} \uparrow \omega)$, $E = IN(\{in(G)\})$ and $F = OUT(\{in(G)\})$.

(ii) If G is unexplainable by (E, F) at 0-step, (E, F) is either $(\emptyset, \emptyset)$ of $(\emptyset, \{G\})$. In the former case, P has no rule with G in the head. Then, $(out(G) \rightarrow \varepsilon)$ is in τP^{-G}, hence $min^{\circ}(\mathbf{T}_{\tau P^{-G}} \uparrow \omega) = \{\emptyset\}$, $E = IN(\emptyset)$ and $F = OUT(\emptyset)$. In the latter case, $G \in P \cap \mathcal{A}$ holds. Then, $\{out(G)\} \in min^{\circ}(\mathbf{T}_{\tau P^{-G}} \uparrow \omega)$, $E = IN(\{out(G)\})$ and $F = OUT(\{out(G)\})$.

Next suppose that the results hold for every step $l \leq s$.

(i) Suppose that G is explainable by (E, F) at $(s+1)$-step, where (E, F) is a minimal explanation of G. Since P is acyclic, there is a ground rule

$$G \leftarrow A_1, \ldots, A_m, \mathit{not}\, A_{m+1}, \ldots, \mathit{not}\, A_n$$

in P such that (i) each A_i $(i = 1, \ldots, m)$ is explainable by (E_i, F_i) at l_i-step $(l_i \leq s)$ and each A_j $(j = m, \ldots, n)$ is unexplainable by (E_j, F_j) at l_j-step $(l_j \leq s)$, (ii) $E = \bigcup_{i=1}^{m} E_i$, $F = \bigcup_{j=1}^{n} F_j$, and $E \cap F = \emptyset$. By the induction hypothesis, each (E_i, F_i) is a minimal explanation of A_i iff $S_i \in min^{\circ}(\mathbf{T}_{\tau P^{+A_i}} \uparrow \omega)$ such that $E_i = IN(S_i)$ and $F_i = OUT(S_i)$. Likewise, each (E_j, F_j) is a minimal anti-explanation of A_j iff $S_j \in min^{\circ}(\mathbf{T}_{\tau P^{-A_j}} \uparrow \omega)$ such that $E_j = IN(S_j)$ and $F_j = OUT(S_j)$.

On the other hand, there is a production

$$in(G) \rightarrow in(B_1) \mid \ldots \mid in(B_l)$$

in τP such that for some $1 \leq k \leq l$, $in(B_k)$ is equal to

$$in(A_1), \ldots, in(A_m), out(A_{m+1}), \ldots, out(A_n).$$

Let $S = \{in(G)*\} \cup \bigcup_{i=1}^{n} S_i$. Here, $E \cap F = \emptyset$ implies that S is coherent. Then, $S \in \mathbf{T}_{\tau P^{+G}} \uparrow \omega$. Also, the minimality of each (E_i, F_i) implies the minimality of S°. Therefore, $S^{\circ} \in min^{\circ}(\mathbf{T}_{\tau P^{+G}} \uparrow \omega)$ and $E = IN(S^{\circ})$ and $F = OUT(S^{\circ})$.

(ii) Suppose that G is unexplainable by (E, F) at $(s+1)$-step, where (E, F) is a minimal anti-explanation of G. Then, for every ground rule of the form

$$r_k : (G \leftarrow A_1, \ldots, A_{m_k}, \mathit{not}\, A_{m_k+1}, \ldots, \mathit{not}\, A_{n_k})$$

in P, there is an atom A_{i_k} $(1 \leq i_k \leq n_k)$ such that A_{i_k} is unexplainable (when $1 \leq i_k \leq m_k$) or explainable (when $m_k + 1 \leq i_k \leq n_k$) by (E_k, F_k) at l_k-step $(l_k \leq s)$, (ii) $E = \bigcup_{r_k} E_k$, $F = \bigcup_{r_k} F_k$, and $E \cap F = \emptyset$. By the induction hypothesis, each (E_k, F_k) is a minimal anti-explanation of A_{i_k} $(1 \leq i_k \leq m_k)$ iff $S_k \in min^{\circ}(\mathbf{T}_{\tau P^{-A_{i_k}}} \uparrow \omega)$ such that $E_k = IN(S_k)$ and $F_k = OUT(S_k)$. Likewise, each (E_k, F_k) is a minimal explanation of A_{i_k} $(m_k + 1 \leq i_k \leq n_k)$ iff $S_k \in min^{\circ}(\mathbf{T}_{\tau P^{+A_{i_k}}} \uparrow \omega)$ such that $E_k = IN(S_k)$ and $F_k = OUT(S_k)$.

On the other hand, there are productions

$$(out(G) \rightarrow out(B_1)),\ \ldots,\ (out(G) \rightarrow out(B_u))$$

in τP, where each $out(B_i)$ is a disjunction of literals. Then, these productions can be rewritten to one production with $out(G)$ in the antecedent:

$$out(G) \rightarrow out(B'_1) \mid \ldots \mid out(B'_v),$$

where each $out(B'_i)$ is now a conjunction of literals formed by collecting one disjunct from every disjunction $out(B_k)$ for $k = 1, \ldots, u$. Then, there is an $out(B'_h)$ which contains either $S_k \in min^{\circ}(\mathbf{T}_{\tau P^{-A_{i_k}}} \uparrow \omega)$ $(1 \leq i \leq m_k)$ or $S_k \in min^{\circ}(\mathbf{T}_{\tau P^{+A_{i_k}}} \uparrow \omega)$ $(m_k + 1 \leq j \leq n_k)$ for every r_k. Let $S = \{out(G)*\} \cup \{out(B'_h)\}$. Here, $E \cap F = (\bigcup_{r_k} E_k) \cap (\bigcup_{r_k} F_k) = \emptyset$ implies that S is coherent. Then, by the fixpoint construction, $S \in \mathbf{T}_{\tau P^{-G}} \uparrow \omega$. Also, the minimality of each (E_k, F_k) implies the minimality of S°. Hence, $S^{\circ} \in min^{\circ}(\mathbf{T}_{\tau P^{-G}} \uparrow \omega)$. By $E = \bigcup_{r_k} IN(S_k)$ and $F = \bigcup_{r_k} OUT(S_k)$, $E = IN(S^{\circ})$ and $F = OUT(S^{\circ})$.

To show the converse direction, put $E = IN(S)$ and $F = OUT(S)$ for $S \in min^{\circ}(\mathbf{T}_{\tau P^{+G}} \uparrow \omega)$ (resp. $S \in min^{\circ}(\mathbf{T}_{\tau P^{-G}} \uparrow \omega)$). Then, we can show that G is explainable (resp. unexplainable) by (E, F) at s-step by following the above proofs backward. □

Corollary 3.8. A ground observation G has no explanation (resp. anti-explanation) in a ground acyclic ALP $\langle P, \mathcal{A} \rangle$ if $\mathbf{T}_{\tau P^{+G}} \uparrow \omega = \emptyset$ (resp. $\mathbf{T}_{\tau P^{-G}} \uparrow \omega = \emptyset$).

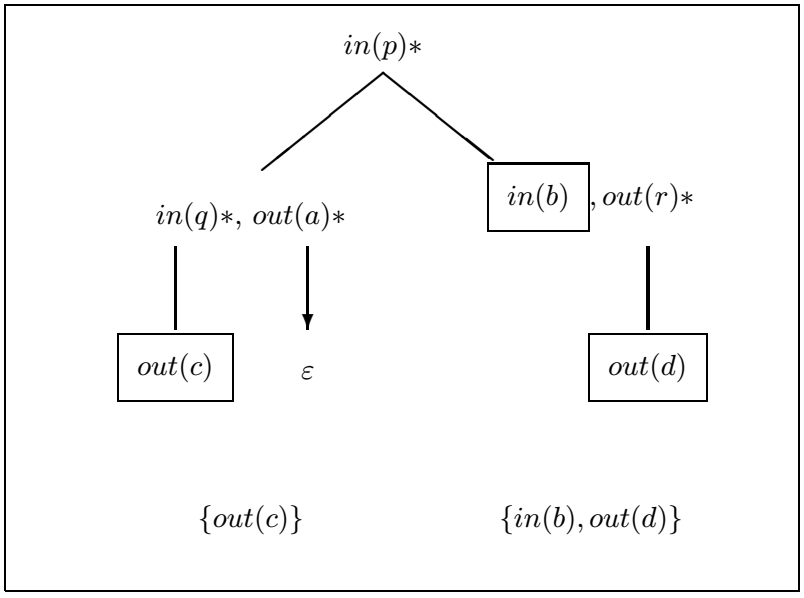

Figure 1. Computing explanations for Example 3.4

Example 3.9. In Example 3.4, $min^{\circ}(\mathbf{T}_{\tau P^{+p}} \uparrow \omega) = \{\{out(c)\}, \{in(b), out(d)\}\}$ (see Figure 1). Then, $G = p$ has two minimal explanations $(E, F) = (\emptyset, \{c\})$, $(\{b\}, \{d\})$.

When a program contains cycles, Theorem 3.7 does not hold in general.

Example 3.10. Let P be the program

$$p \leftarrow q,$$
$$q \leftarrow p,$$

and $\mathcal{A} = \emptyset$. Then, p has no explanation. However,

$$in(p) \rightarrow in(q),$$
$$in(q) \rightarrow in(p)$$

are in τP, then $\mathbf{T}_{\tau P^{+p}} \uparrow \omega = \{\{in(p)*, in(q)\}\}$ and $min^{\circ}(\mathbf{T}_{\tau P^{+p}} \uparrow \omega) = \{\emptyset\}$, which does not imply the absence of explanation of p.

However, the acyclicity is not a necessary condition for Theorem 3.7.

Example 3.11. Let P be the program

$$g \leftarrow p,$$
$$p \leftarrow not\, q,$$
$$q \leftarrow q,$$

and $\mathcal{A} = \emptyset$. Then, g has the explanation $(E, F) = (\emptyset, \emptyset)$. In this case, τP includes

$$in(g) \rightarrow in(p),$$
$$in(p) \rightarrow out(q),$$
$$out(q) \rightarrow out(q).$$

Then, $\mathbf{T}_{\tau P^{+g}} \uparrow \omega = \{\{in(g)*, in(p)*, out(q)\}\}$, and $min^{\circ}(\mathbf{T}_{\tau P^{+g}} \uparrow \omega) = \{\emptyset\}$, which corresponds to the explanation.

In Section 4.3, we will show that the correctness of our procedure to compute negative anti-explanations is also assured for the class of non-acyclic Horn programs (see Theorem 4.3).

While a program P in an ALP $\langle P, \mathcal{A} \rangle$ has been given as an NLP, the results in this section can be directly applied to abductive logic programs containing classical negation as follows. When P is an *extended logic program* (ELP) [16], in which negative literals as well as positive literals are allowed in A_i's of each rule of the form (1), the *answer set semantics* is employed as the semantics of the program and Definition 2.1 is unchanged. The only additional requirement is that

the definition of *coherency* of transactions should be strengthened: Any coherent transaction should not include both $in(L)$ and $in(\overline{L})$ for any complimentary pair of literals $L, \overline{L}$.

3.3. Non-ground case

We extend results in previous subsections to non-ground programs. In this section, we consider a non-ground ALP $\langle P, \mathcal{A} \rangle$ in which P is a *covered* NLP and observations are ground. Here, $\langle P, \mathcal{A} \rangle$ is called a *covered* ALP if P is *covered*, that is, for every rule in P, all variables in the body appear in the head [31].[4] Note that each integrity constraint in a covered program has to be ground. With these restrictions, transactions in any bottom-up computation are guaranteed to be always ground. We also assume that for each predicate p, every atom with p is either abducible or non-abducible, and call such a predicate an *abducible predicate* in the former case. While we might consider a more general case using theories of predicate completion in logic programming, such an extension is too technical and is beyond the scope of this paper.

In the presence of variables in a program, we use *Clark completion* [7] for producing transaction programs. Recall that Clark completion is defined as follows. Any rule

$$p(t_1, \ldots, t_n) \leftarrow B$$

in P, where $t_1, \ldots, t_n$ are terms, is written in the form

$$p(x_1, \ldots, x_n) \leftarrow (\exists \mathbf{y})\, (x_1 = t_1), \ldots, (x_n = t_n),\ B,$$

where $x_1, \ldots, x_n$ are new variables and $\mathbf{y}$ are the variables in the original rule. Note here that when P is covered and each t_i is ground, $(\exists \mathbf{y})$ need not be considered. If there are k rules with the predicate p in the head, they are written in the form

$$\begin{aligned} p(x_1, \ldots, x_n) &\leftarrow E_1, B_1, \\ &\vdots \\ p(x_1, \ldots, x_n) &\leftarrow E_k, B_k, \end{aligned} \tag{7}$$

where E_i's are equalities and B_i's are original bodies. Then, the definition for p is

$$p(x_1, \ldots, x_n) \leftrightarrow E_1, B_1 \mid \ldots \mid E_k, B_k.$$

Now, given an ALP $\langle P, \mathcal{A} \rangle$, its transaction program is defined like Definition 3.1 based on Clark completion.

[4] A covered program is also called a *constrained* program in the literature, e.g., [26].

Definition 3.12. Let $\langle P, \mathcal{A} \rangle$ be a covered ALP. The *transaction program* τP *of* P *(relative to* $\mathcal{A}$*)* is defined as follows.

1. Let p be a non-abducible predicate and $k > 0$ for rules (7). Then, for $H = p(x_1, \ldots, x_n)$, rules (7) are transformed to productions:

$$R_{in}: \ in(H) \to E_1, in(B_1) \mid \ldots \mid E_k, in(B_k), \tag{8}$$

$$R_{out}: out(H), E_1 \to out(B_1),$$

$$\vdots \tag{9}$$

$$out(H), E_k \to out(B_k),$$

$$out(H), \neg E_1, \ldots, \neg E_k \to \varepsilon, \tag{10}$$

   where for $i = 1, \ldots, k$, $in(B_i)$ and $out(B_i)$ are defined in the same way as Definition 3.1. In particular, if B_i is empty, then $in(B_i) = \varepsilon$ and $out(B_i) = false$. Also, $\neg E = (x_1 \neq t_1) \vee \ldots \vee (x_n \neq t_n)$ for $E = (x_1 = t_1), \ldots, (x_n = t_n)$. When $H = \bot$ (integrity constrains), there are no E_i's and only (9) in R_{out} is included in τP.

2. Let p be an abducible predicate and $k > 0$ for rules (7). In this case, every rule in (7) is in the form $p(t_1, \ldots, t_n) \leftarrow$ by the assumption.[5] Then, for $H = p(x_1, \ldots, x_n)$, rules (7) are transformed to

$$R_{in}: \ in(H), E_1 \to \varepsilon,$$

$$\vdots \tag{11}$$

$$in(H), E_k \to \varepsilon,$$

$$R_{out}: out(H), \neg E_1, \ldots, \neg E_k \to \varepsilon. \tag{12}$$

3. Let p be any predicate whose definition is empty, i.e., $k = 0$ for rules (7). For every such predicate p, introduce the production:

$$out(p(x_1, \ldots, x_n)) \to \varepsilon \tag{13}$$

   to τP. Moreover, when p is a non-abducible predicate, τP also includes:

$$in(p(x_1, \ldots, x_n)) \to false. \tag{14}$$

An essential difference from the ground case is that transaction programs now contain equalities E_i's and inequalities $\neg E_i$'s. In a covered program with a ground observation, we can regard each equality $x = t$ or inequality $x \neq t$ just as a literal to be evaluated for comparison of terms: $x = t$ is *true* ($x \neq t$ is *false*) if the terms currently instantiating x and t are literally identical or they are unifiable; otherwise $x = t$ is *false* ($x \neq t$ is *true*). (In)equalities in antecedents of productions are thus evaluated for this checking after T-literals in antecedents are

[5] Recall that any rule with an abducible in the head is always a fact (see Section 2).

instantiated. Then, if x and t are unifiable in an equality $x = t$ in the antecedent of some production (9), every occurrence of t in T-literals in the consequent of the production is substituted with the term currently instantiating x. On the other hand, if x and t are unifiable in $x = t$ in the consequent of (8), every occurrence of t in the same disjunct of the consequent is substituted with the term currently instantiating x. Note that when an NLP P is ground, Definition 3.12 reduces to Definition 3.1.

The mapping $T_{\tau P}$ in Definition 3.3 is also slightly extended as follows. Variables in each production are instantiated with a substitution σ such that $in(H\sigma)$ or $out(H\sigma)$ in the antecedent is contained in a transaction I. Marking is performed upon an instantiated literal in the antecedent of a production. Here, in obtaining a substitution σ in the $T_{\tau P}$ operator, it is sufficient to consider *matching* instead of full unification if a given NLP is covered and an observation is ground. Starting from a ground observation $in(G)$ or $out(G)$, the fixpoint operator is repeatedly applied as long as some transaction is changed. In this case, every transaction I constructed in fixpoint computation contains only ground literals. Model generation procedures introduced in Section 3.2 can be used in this case too, since τP is range-restricted whenever P is constrained.

With this setting, fixpoint closures are defined in the same manner as in the preceding section, and $\mathbf{T}_{\tau P} \uparrow \omega$ reaches the fixpoint. Then, results of Theorem 3.7 are directly extended to computing (anti-)explanations from non-ground acyclic programs.

Theorem 3.13. Let $\langle P, \mathcal{A} \rangle$ be a covered acyclic ALP, and G a ground observation. Then,

(i) (E, F) is a minimal explanation of G wrt $\langle P, \mathcal{A} \rangle$ iff there is a transaction $S \in min^{\circ}(\mathbf{T}_{\tau P^{+G}} \uparrow \omega)$ such that $E = IN(S)$ and $F = OUT(S)$.
(ii) (E, F) is a minimal anti-explanation of G wrt $\langle P, \mathcal{A} \rangle$ iff there is a transaction $S \in min^{\circ}(\mathbf{T}_{\tau P^{-G}} \uparrow \omega)$ such that $E = IN(S)$ and $F = OUT(S)$.

Proof. The proof of Theorem 3.7 can be lifted to the non-ground case. □

Example 3.14. Let $\langle P, \mathcal{A} \rangle$ be an ALP, where

$$\begin{aligned} P :& \, g(x) \leftarrow p(x),\ not\ q(x), \\ & \quad q(a) \leftarrow , \\ \mathcal{A} :& \, p(x),\ q(x). \end{aligned}$$

Then, τP includes

$$\begin{aligned} & in(g(y)) \rightarrow (y = x),\ in(p(x)),\ out(q(x)), \\ & out(g(y)),\ (y = x) \rightarrow out(p(x)) \mid in(q(x)), \\ & out(p(y)) \rightarrow \varepsilon, \end{aligned}$$

$$in(q(y)),\ (y = a) \to \varepsilon,$$
$$out(q(y)),\ (y \neq a) \to \varepsilon.$$

Let $O_1 = g(a)$ be a positive observation. By $min^{\circ}(\mathbf{T}_{\tau P+O_1} \uparrow \omega) = \{\{in(p(a)), out(q(a))\}\}$, the minimal explanation of O_1 is $(\{p(a)\}, \{q(a)\})$.

Next, let $O_2 = g(b)$ be a negative observation. Then, $\mathbf{T}_{\tau P-O_2} \uparrow \omega$ includes $\{out(g(b))*, out(p(b))*\}$ and $\{out(g(b))*, in(q(b))\}$, so $min^{\circ}(\mathbf{T}_{\tau P-O_2} \uparrow \omega) = \{\emptyset\}$. Hence, $(\emptyset, \emptyset)$ is the minimal anti-explanation of O_2.

Example 3.15. Consider the ALP $\langle K, \mathcal{A} \rangle$ in Example 2.2, where K is given in Section 1 as

$$flies(x) \leftarrow bird(x), not\ ab(x),$$
$$ab(x) \leftarrow broken\text{-}wing(x),$$
$$bird(tweety) \leftarrow,\ \ bird(opus) \leftarrow,$$
$$broken\text{-}wing(tweety) \leftarrow,$$

and $\mathcal{A} = \{\, broken\text{-}wing(x) \,\}$. Then, τK becomes

$$in(flies(y)) \to (y = x),\ in(bird(x)),\ out(ab(x)),$$
$$out(flies(y)),\ (y = x) \to out(bird(x)) \mid in(ab(x)),$$
$$in(ab(y)) \to (y = x),\ in(broken\text{-}wing(x)),$$
$$out(ab(y)),\ (y = x) \to out(broken\text{-}wing(x)),$$
$$in(bird(y)) \to (y = tweety) \mid (y = opus),$$
$$out(bird(y)),\ (y = tweety) \to false,$$
$$out(bird(y)),\ (y = opus) \to false,$$
$$out(bird(y)),\ (y \neq tweety),\ (y \neq opus) \to \varepsilon,$$
$$in(broken\text{-}wing(y)),\ (y = tweety) \to \varepsilon,$$
$$out(broken\text{-}wing(y)),\ (y \neq tweety) \to \varepsilon.$$

Then, for the positive observation $G_1 = flies(tweety)$, $min^{\circ}(\mathbf{T}_{\tau K+G_1} \uparrow \omega)$ is $\{\{out(broken\text{-}wing(tweety))\}\}$, which indicates that $(\emptyset, \{broken\text{-}wing(tweety)\})$ is the minimal explanation of G_1. Also, for the negative observation $G_2 = flies(opus)$, $min^{\circ}(\mathbf{T}_{\tau K-G_2} \uparrow \omega)$ is $\{\{in(broken\text{-}wing(opus))\}\}$, which corresponds to the minimal anti-explanation $(\{broken\text{-}wing(opus)\}, \emptyset)$ of G_2.

4. Discussion

4.1. Abductive procedure

The idea of computing abduction through transaction programs is close to that of computing abduction through Clark *completion* [8,24,14]. Our ap-

proach is essentially different from such previous work in the following points. Firstly, our procedure is intended to compute positive, negative and mixed (anti-)explanations in extended abduction, while [8,24,14] are designed to compute positive explanations only. In fact, productions R_{out} are newly introduced in our transformation. However, the use of Clark completion in these procedures enables us to compute anti-explanations for negative observations by treating $out(p)$ as $\neg p$ for an atom p in the completion formula. Secondly, we generate a transaction program which declaratively specifies an abductive procedure, while no such program is generated in those work. Thirdly, the procedure called ABDUCE by [8] is given in an abstract form only. Then, a naive implementation would result in an inefficient procedure. In particular, the same predicate could be rewritten more than once. In our case, such a redundant rewriting of the same predicate is maximally avoided by bottom-up computation with marking. In [8], soundness and completeness are guaranteed for hierarchical NLPs. In our procedure, they are guaranteed for acyclic NLPs which strictly include hierarchical NLPs. In [24], only the classical propositional theory is considered. In [14], the non-ground case is fully considered for the iff form definitions using rewrite rules for equality.

A fixpoint semantics of abductive logic programs is also introduced in [20], but it computes traditional abduction and does not compute the extended one. The transformation of ALPs to disjunctive logic programs in [20] is also entirely different from the one in this paper. However, we have shown in this paper that a similar fixpoint operator can be used to compute extended abduction. In fact, a transaction program is a kind of disjunctive programs and fixpoint computation can be realized by model generation procedures used in [5,20,4] provided that the functions of rewriting and marking are realized in these procedures.

Besides [8,20,14], there are many other abductive procedures for abductive NLPs (e.g., [23,11,12]). Although all other abductive procedures are developed to compute positive explanations only, they can also be used to compute positive anti-explanations as follows. When we want to compute anti-explanations of G, a new rule $G' \leftarrow not\, G$ is introduced with a new atom G'. Then, an anti-explanation of G can be obtained as a *credulous explanation* of G'.[6] On the other hand, negative or mixed (anti-)explanations cannot be directly computed by other abductive procedures.

Alferes, Damásio and Pereira [2,10] propose an abductive framework within the three-valued semantics in a different way. Abduction is performed in their revision system to remove a contradiction in a program by changing the truth-value of abducible literals into true, false or undefined, in which the change from true to false/undefined corresponds to our removal of abducibles in negative or

[6] An explanation is *credulous* if G is included in *some* stable model of $(P \cup E) \setminus F$ in Definition 2.1. In contrast, explanations considered in this paper are *skeptical*. Since an acyclic program has at most one stable model, credulous and skeptical explanations coincide for acyclic ALPs.

mixed (anti-)explanations. Here, the unknown value of an abducible is due to their three-valued semantics, which roughly corresponds to the situation that true is assigned to the abducible in some stable model and false is also assigned to it in another stable model. Their abductive framework is computed using their top-down procedure [2], but they do not produce a transaction program. Usually, top-down procedures like [23,10–12] compute one explanation at a time, but the minimality of an abductive explanation is not guaranteed in general. Our fixpoint operator in Definition 3.3, on the other hand, involves computation of all explanations containing minimal ones, because it expands each transaction in every possible way in parallel. However, a bottom-up model generation procedure can also be designed to get one (not necessarily minimal) solution at a time by expanding each transaction I with K in Definition 3.3 one by one in a breadth-first or depth-first manner.

Transaction programs can also be used for simulating top-down query processing in acyclic NLPs. That is, regarding a query as a positive observation with no abducibles, fixpoint computation corresponds to a top-down bread-first OR-parallel query processing. The query succeeds with a refutation only when the fixpoint contains an empty set.

4.2. Database update

It is known that abduction is used for *database update*. In deductive databases, an update request on an intensional fact is often translated into update on extensional facts. Such database update is called *view update*. More precisely, the view update problem is presented as follows. A *deductive database* (or *database* for short) is defined as a function-free consistent NLP. Let D be a database and $\mathcal{E}$ a set of ground facts in the language of D called *extensional facts*. Then, an *insertion* of a ground fact A into D (resp. *deletion* of A from D) is accomplished by an *updated database*

$$D' = (D \cup E) \setminus F \quad \text{with} \quad E, F \subseteq \mathcal{E}$$

satisfying the conditions:

1. $D' \models A$ (resp. $D \not\models A$) where D' is consistent,
2. there is no database D'' satisfying the condition 1 and $D \sim D'' \subset D \sim D'$, where $D_1 \sim D_2 = (D_1 \setminus D_2) \cup (D_2 \setminus D_1)$.

The first condition says that an updated database accomplishes an insertion/deletion, while the second condition presents that the update *minimally changes* extensional facts stored in D.

Relating to abduction, an update request corresponds to a positive or negative observation and extensional facts correspond to abducibles. Then, view update can be characterized by extended abduction.

Lemma 4.1. [19] Given a database D and a ground atom A, an updated database D' accomplishes an insertion/deletion of A into/from D iff there is a minimal explanation/anti-explanation (E, F) of A in D.

Using this lemma, the next results are obtained by Theorem 3.7.

Theorem 4.2. Let D be a covered acyclic database, and A a ground atom.

(i) An insertion of A into D is achieved by an update (E, F) iff there is a transaction $S \in min^{\circ}(\mathbf{T}_{\tau D^{+A}} \uparrow \omega)$ such that $E = IN(S)$ and $F = OUT(S)$.
(ii) A deletion of A from D is achieved by an update (E, F) iff there is a transaction $S \in min^{\circ}(\mathbf{T}_{\tau D^{-A}} \uparrow \omega)$ such that $E = IN(S)$ and $F = OUT(S)$.

Thus, we can compute database update via fixpoint computation of a transaction program.

Kakas and Mancarella [23] characterize database update through abduction, and use a top-down abductive procedure for computing view update. Their procedure does not explicitly take existing or non-existing facts into account; it reasons only on the rules without the facts, although the facts (EDB) provide useful information which simplifies transactions by removing conditions that have already been satisfied. In our transaction programs, information on EDB is represented by productions like $in(A) \to \varepsilon$ (the fact A need not be inserted), $out(A) \to \varepsilon$ (A need not be deleted), and $in(A) \to false$ (A cannot be inserted). Also, Kakas and Mancarella's approach is not well defined in the presence of integrity constraints, which makes the consistency checking of hypotheses difficult; after the candidate solutions are computed, those candidates that are inconsistent with the integrity constraints are pruned. This difficulty is solved by SLDAI resolution by Decker [11], which satisfies the update request while maintaining integrity. In our case, integrity constraints are again embedded in transaction programs. Hence, abduction, EDB, and integrity are all treated in a uniform manner by computation of transaction programs. In [5], abduction is translated into a disjunctive program and database update is realized by bottom-up computation on a meta-program specifying an update procedure. The procedures in [23,5,11] are all based on traditional abduction and do not produce transaction programs.

In database update, Rossi and Naqvi [30] use Clark's completion to compute update in Horn databases. Given an update request, it is transformed to non-Horn formulas obtained by the only-if parts of database clauses. The resulting formulas represent update on the original database. Such formulas are computed using an SLD-like top-down procedure. Guessoum and Lloyd [18] provide a procedure which performs update in NLPs. They compute update using SLDNF-trees, and the procedure is correct in (locally) call-consistent programs but requires to check if the updates drawn from SLDNF-trees can actually perform the update request. These procedures do not produce transaction programs, and are only

concerned about satisfying update requests, not about the more general topic of finding explanations to observations.

Console *et al.* [9] discuss similarities and differences between view update and abduction, and apply the ABDUCE procedure of [8] to view update without producing transaction programs. Information on EDB is incorporated in their computation of view update, and they argue that this incorporation illustrates a difference between abduction and view update. In our extended abductive framework, there is no such difference because we regard removal of abducibles as a part of abduction.

Grant *et al.* [17] translate view update into a set of disjunctive facts based on expansion of an SLD-tree, in which update is achieved by inserting/deleting these disjunctive facts to/from a database. Fernández *et al.* [13] realize update through construction of minimal models that satisfy an update request. In these work, databases are treated as ground programs possibly containing disjunctive facts, but transaction programs are not produced. Our transaction program specifies update by disjunction but does not introduce disjunctions to a database.

A transaction program is also viewed as a database update language which declaratively specifies dynamic changes in databases. Our program transformation provides an *automatic generation of update specification* from the original program. The generated transaction program is a kind of disjunctive program, and non-determinism in database update is naturally expressed using disjunctive productions. In this sense, transaction programs provide a new application of disjunctive logic programming. Note that several database update languages exist [1], but few consider disjunctions in the language.

4.3. View deletion from definite database

Recently, Aravindan and Baumgartner [4] proposed a transformation of ground definite programs into disjunctive programs to compute view deletion. For definite programs, R_{out} in our transaction programs is similar to their transformation, and can be used to realize view deletion in computing anti-explanations. Like [4], the correctness of our fixpoint computation is guaranteed also for *non-acyclic* Horn programs.

Theorem 4.3. Let $\langle P, \mathcal{A} \rangle$ be a ground Horn ALP, and G a ground negative observation. Then, F is a minimal negative anti-explanation of G wrt $\langle P, \mathcal{A} \rangle$ iff there is a transaction $S \in min^{\circ}(\mathbf{T}_{\tau P^{-G}} \uparrow \omega)$ such that $F = OUT(S)$.

Proof. By Lemma 4.1, a deletion of G is accomplished iff there is a minimal anti-explanation (E, F) of G wrt $\langle P, \mathcal{A} \rangle$. Since P is Horn, negation as failure does not appear and hence $E = \emptyset$, that is, F is a negative anti-explanation of G. Moreover, R_{out} in Definition 3.1 consists of productions containing *out*-literals

only. Since an update request is deletion of G, only *out*-literals are concerned and R_{in} can be ignored in the transaction program τP.

Now, let S be any transaction in $\mathbf{T}_{\tau P-G} \uparrow \omega$, and $F = OUT(S^\circ)$. Also, let $out(L)$ be any unmarked literal in S. There are two cases for L:

1. *L is an abducible and $(L \leftarrow) \in P$.*
   In this case, L actually has to be removed from P.

2. *L is not an abducible.*
   In this case, L has a non-empty definition in P, because otherwise, $(out(L) \rightarrow false)$ is in τP and S is not included in the fixpoint. Then, a production of the form:

   $$out(L) \rightarrow out(A_1) \mid \ldots \mid out(A_m)$$

   is in τP, and for some A_i $(1 \leq i \leq m)$, either $out(A_i)$ or $out(A_i)*$ is in S. When $out(A_i) \in S$, L's removal is accomplished by the deletion of A_i. When $out(A_i)* \in S$, L's removal has already been translated into either ε when A_i has no definition, or otherwise a request of deletion of some other literals.

In the second case above, we can consider again the above two cases for A_i. If A_i is an abducible (i.e., the first case above), L's removal is successfully accomplished. Otherwise, by recursively analyzing the second case above, the chain eventually terminates as L, $L_1(= A_i)$, $\ldots$, L_k, where L_k is either an abducible or identical to L, since the number of rules in P is finite. If L_k is an abducible, we can utilize the definition of abduction steps in Definition 3.5, so that L is unexplainable at k-step and Theorem 3.7 can be applied as well. Else if $L_k = L$, the program P is not acyclic. In this case, L_k cannot be entailed from P, and thus L_k can be unexplained and so can be L. Therefore, $out(L)$ for a non-abducible L can be removed from S by taking S°, and hence $F = OUT(S^\circ)$ is a negative anti-explanation of G. The minimality of anti-explanations is also guaranteed by taking the operation min° to the fixpoint.

The proof of the converse direction can also be shown based on a similar argument as above. □

Example 4.4. [4, Example 3.6] Consider the ALP $\langle P, \mathcal{A} \rangle$:

$$
\begin{array}{lll}
P: & p \leftarrow t, & \tau P: \ out(p) \rightarrow out(t), \\
& p \leftarrow q, u, & \quad\ out(p) \rightarrow out(q) \mid out(u), \\
& q \leftarrow s, & \quad\ out(q) \rightarrow out(s), \\
& u \leftarrow r, & \quad\ out(u) \rightarrow out(r), \\
& t \leftarrow, \quad r \leftarrow . & \quad\ out(s) \rightarrow \varepsilon. \\
\mathcal{A}: & r,\ s,\ t. &
\end{array}
$$

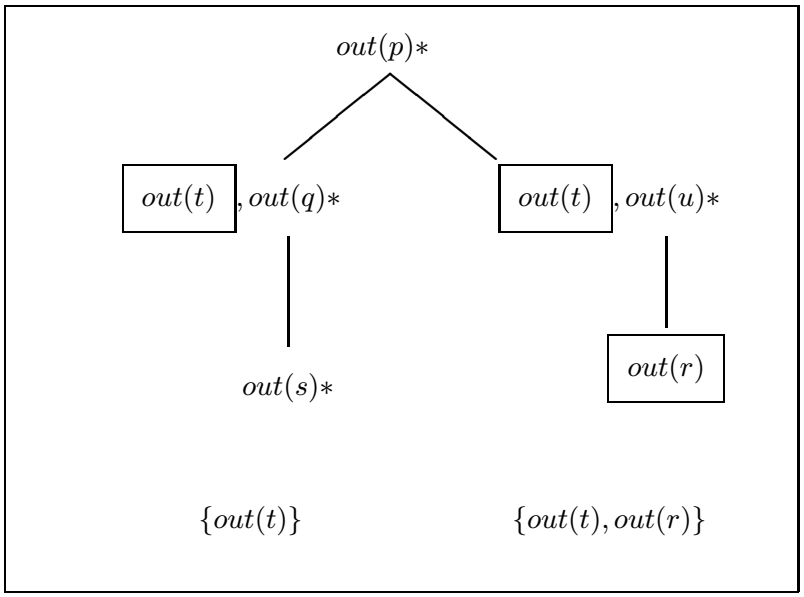

Figure 2. Computing explanations for Example 4.1

Here, only R_{out} is shown for τP. Suppose that p should be deleted from the database. $\mathbf{T}_{\tau P - p} \uparrow \omega$ is illustrated in Figure 2. In the figure, $\{out(t), out(r)\}$ is not minimal. Hence, the minimal negative anti-explanation of p is $\{t\}$.

As [4] pointed out, a general *contraction* algorithm including view deletion for definite programs requires computation of (i) all minimal positive explanations of the sentence to be contracted wrt the background knowledge without abducible facts, and then (ii) a (minimal) *hitting set* [29] for these explanations, which corresponds to a (minimal) negative anti-explanation. Both their method and ours directly compute such hitting sets without computing positive explanations explicitly. In contrast to [4], our transaction programs can be applied to programs with negation as failure, and can also be used to compute positive (negative, and mixed) (anti-)explanations in extended abduction.

4.4. Revision program

Marek and Truszczynski [27] define a language to specify revision in knowledge bases. The meaning of their *revision programs* is similar to the stable model semantics for logic programming. They define a *knowledge base* (KB) as a set of propositional atoms. A *(disjunctive) revision program* Π consists of *revision rules* of the form:

$$in(p_1) \mid \ldots \mid in(p_k) \mid out(r_1) \mid \ldots \mid out(r_l) \\ \leftarrow in(q_1), \ldots, in(q_m), out(s_1), \ldots, out(s_n), \tag{15}$$

where each of p_i, r_i, q_i, s_i is a propositional atom. A revision rule (15) intuitively means a constraint that either p_i $(1 \leq i \leq k)$ belongs to the database D under consideration or r_j $(1 \leq j \leq l)$ does not belong to D, whenever all of $q_1, \ldots, q_m$ belong to D and none of $s_1, \ldots, s_n$ belongs to D. Each revision rule in Π can be regarded as a clause in propositional logic. Then, for each minimal model M of Π, a pair (I, O) where $I = \{a \mid in(a) \in M\}$ and $O = \{a \mid out(a) \in M\}$ is called a *necessary change* for Π. Let D_I (called an *initial KB*) and D_R be two KBs. The revision program $\Pi_{D_R}|D_I$ is obtained by:

1. deleting from Π every rule of the form (15) such that $q_i \notin D_R$ for some $1 \leq i \leq m$ or $s_j \in D_R$ for some $1 \leq j \leq n$, and
2. deleting each $in(a)$ such that $a \in D_I$ and each $out(a)$ such that $a \notin D_I$ from the body of every remaining revision rule.

Let (I, O) be a necessary change for $\Pi_{D_R}|D_I$. Then, D_R is called a Π-*justified revision* of D_I if $I \cap O = \emptyset$ and $D_R = (D_I \cup I) \setminus O$. It has been shown that a Π-justified revision always performs a minimal change that is justified by the use of revision rules.

Although the direction of arrows of revision rules ($\leftarrow$) and that of productions ($\rightarrow$) are opposite, they look very similar in their syntax. In fact, a propositional transaction program is a *context-free* revision program, i.e., a disjunctive revision program in which every revision rule has only one literal in the body.

There are several differences between the framework of [27] and ours. First, in revision programming, a KB contains no rules except facts, and there are no notion of abducibles. Instead, a necessary change represents a whole change of atoms included in the KB, and every atom in the language can be both inserted and deleted. Second, we used the notion of marking to represent literals whose transactions have been translated into another atoms' requests, but such information is never used to compute revision programming. This is because in revision programming every atom is subject to change and view update is not considered.

Taking these differences into account, we can utilize transaction programs in revision programming. As in Section 4.2, a typical application of revision programming is database update. Recall that a database is specified by an NLP P, which is divided into the intensional part (IDB) P_r containing the rules with non-empty bodies in P, and the extensional part (EDB) containing the facts in P. An initial KB D_I is then set to a stable model of P, which includes all facts in EDB. A revision program Π is constructed from P_r and an update request by transforming them to the transaction program ${\tau P_r}^{+G}$ or ${\tau P_r}^{-G}$. Then, a revised KB D_R is computed through the transaction program. Notice that D_R still satisfies IDB of P and minimally changes EDB.

Example 4.5. Let P be the NLP:

$$p(x) \leftarrow s(x), \mathit{not}\, q(x),$$

$$q(a) \leftarrow,$$
$$s(a) \leftarrow,$$
$$s(b) \leftarrow .$$

Here, the first rule is transformed to the productions:

$$R : in(p(y)) \rightarrow (y = x),\ in(s(x)),\ out(q(x)),$$
$$out(p(y)),\ (y = x) \rightarrow out(s(x)) \mid in(q(x)).$$

Set the initial KB to the stable model of P, i.e.,

$$D_I = \{s(a), s(b), q(a), p(b)\}.$$

Let us consider insertion of $G = p(a)$, and put

$$\Pi = R \cup \{\ \rightarrow in(p(a))\ \}.$$

Viewing Π as a revision program, let

$$D_R = \{s(a), s(b), p(a), p(b)\}.$$

Then, we obtain

$$\Pi_{D_R}|D_I : in(s(a)),\ out(q(a)) \leftarrow ,$$
$$in(s(b)),\ out(q(b)) \leftarrow ,$$
$$in(p(a)) \leftarrow ,$$

and the necessary change for $\Pi_{D_R}|D_I$ is $(I, O) = (\{p(a), s(a), s(b)\}, \{q(a), q(b)\})$. Hence,

$$D_R = (D_I \cup I) \setminus O$$

holds, and D_R is a Π-justified revision of D_I.

On the other hand, viewing Π as a transaction program, $\mathbf{T}_\Pi \uparrow \omega$ includes $S = \{in(p(a))*, in(s(a)), out(q(a))\}$. By ignoring marking in S, we obtain $(I', O') = (\{p(a), s(a)\}, \{q(a)\})$. Hence,

$$D' = (D_I \cup I') \setminus O' = \{s(a), s(b), p(a), p(b)\},$$

which coincides with D_R in this case.

Note in the above example that if we do not ignore marking, we get $(IN(S), OUT(S)) = (\{s(a)\}, \{q(a)\})$, which is a smaller explanation of G. Furthermore, our framework can treat extensional facts as abducibles, thereby extends revision programming with *view*.

Hence, our transformation of P into τP provides a method of generating a revision program from the original program P. In deductive databases, such a program P is constructed from IDB. In other application domains, a program P can be any background knowledge base including causal relations, constraints,

general laws and commonsense. As far as the authors know, there have been no other methods to generate revision programs automatically, and few discussion has been done on who decides revision rules and how they can be obtained.

5. Conclusion

This paper presented a method of computing extended abduction. An abductive logic program was transformed to a transaction program which declaratively specifies non-determinism in abduction. Then abductive explanations can be computed by the fixpoint of a transaction program, which is sound and complete for acyclic and covered ALPs. In the context of databases, a transaction program provides a declarative specification of database update, and enables us to compute view update in a constructive manner. A transaction program is a kind of disjunctive programs and fixpoint computation can be realized by model generation procedures.

This paper assumed acyclic and covered NLPs for programs containing variables. Future work includes relaxing these conditions and exploiting further applications of extended abduction.

References

[1] S. Abiteboul, Updates, a new frontier, in: *Proceedings of the 2nd International Conference on Database Theory, Lecture Notes in Computer Science*, 326, Springer, 1988, pp. 1–18.

[2] J. J. Alferes, C. V. Damásio and L. M. Pereira, A logic programming system for nonmonotonic reasoning, *J. Automated Reasoning*, 14 (1995) 93–147.

[3] K. R. Apt and M. Bezem, Acyclic programs, *New Generation Computing*, 9 (1991) 335–363.

[4] C. Aravindan and P. Baumgartner, A rational and efficient algorithm for view deletion in databases, in: *Proceedings of the 1997 International Symposium on Logic Programming*, MIT Press, 1997, pp. 165–179.

[5] F. Bry, Intensional updates: abduction via deduction, in: *Proceedings of the 7th International Conference on Logic Programming*, MIT Press, 1990, pp. 561–575.

[6] F. Buccafurri, T. Eiter, G. Gottlob and L. Leone, Enhancing symbolic model checking by AI techniques, IFIG Research Report 9701, Institut für Informatik, Universität Gießen (1997).

[7] K. L. Clark, Negation as failure, in: H. Gallaire and J. Minker (eds.), *Logic and Data Bases*, Plenum Press, New York, 1978, pp. 119–140.

[8] L. Console, D. Theseider Dupré and P. Torasso, On the relationship between abduction and deduction, *Journal of Logic and Computation*, 1 (1991) 661–690.

[9] L. Console, M. L. Sapino, and D. Theseider Dupré, The role of abduction in database view updating, *Journal of Intelligent Information Systems*, 4 (1995) 261–280.

[10] C. V. Damásio and L. M. Pereira, Abduction over 3-valued extended logic programs, in: V. W. Marek, A. Nerode and M. Truszczyński (eds.), *Proceedings of the 3rd International Conference on Logic Programming and Nonmonotonic Reasoning, Lecture Notes in Artificial Intelligence*, 928, Springer, 1995, pp. 29–42.

[11] H. Decker, An extension of SLD by abduction and integrity maintenance for view updating in deductive databases, in: *Proceedings of the 1996 Joint International Conference and Symposium on Logic Programming*, MIT Press, 1996, pp. 157–169.
[12] M. Denecker and D. De Schreye, SLDNFA: an abductive procedure for abductive logic programs, *Journal of Logic Programming*, 34 (1998) 111–167.
[13] J. Fernández, J. Grant and J. Minker, Model theoretic approach to view updates in deductive databases, *Journal of Automated Reasoning*, 17 (1996) 171–197.
[14] T. H. Fung and R. Kowalski, The iff procedure for abductive logic programming, *Journal of Logic Programming*, 33 (1997) 151–165.
[15] M. Gelfond and V. Lifschitz, The stable model semantics for logic programming, in: *Proceedings of the 5th International Conference and Symposium on Logic Programming*, MIT Press, 1988, pp. 1070–1080.
[16] M. Gelfond and V. Lifschitz, Classical negation in logic programs and disjunctive databases, *New Generation Computing*, 9 (1991) 365–385.
[17] J. Grant, J. Horty, J. Lobo and J. Minker, View updates in stratified disjunctive databases, *Journal of Automated Reasoning*, 11 (1993) 249–267.
[18] A. Guessoum and J. W. Lloyd, Updating knowledge bases II, *New Generation Computing*, 10 (1991) 73–100.
[19] K. Inoue and C. Sakama, Abductive framework for nonmonotonic theory change, in: *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, 1995, pp. 204–210.
[20] K. Inoue and C. Sakama, A fixpoint characterization of abductive logic programs, *Journal of Logic Programming*, 27 (1996) 107–136.
[21] K. Inoue and C. Sakama, Specifying transactions for extended abduction, in: *Proceedings of the 6th International Conference on Principles of Knowledge Representation and Reasoning*, Morgan Kaufmann, 1998, pp. 394–405.
[22] A. C. Kakas, R. A. Kowalski and F. Toni, Abductive logic programming, *Journal of Logic and Computation*, 2 (1992) 719–770.
[23] A. C. Kakas and P. Mancarella, Database updates through abduction, in: *Proceedings of the 16th International Conference on Very Large Databases*, Morgan Kaufmann, 1990, pp. 650–661.
[24] K. Konolige, Abduction versus closure in causal theories, *Artificial Intelligence*, 53 (1992) 255–272.
[25] K. Konolige, Abductive theories in artificial intelligence, in: G. Brewka (ed.), *Principles of Knowledge Representation*, CSLI Publications & FoLLI, 1996, pp. 129–152.
[26] N. Lavrač and S. Džeroski, *Inductive Logic Programming: Techniques and Applications*, Ellis Horwood, 1994.
[27] V. W. Marek and M. Truszczyński, Revision specifications by means of programs, in: *Proceedings of the 4th European Conference on Logics in AI, Lecture Notes in Artificial Intelligence*, 838, Springer, 1994, pp. 122–136.
[28] D. Poole, A logical framework for default reasoning, *Artificial Intelligence*, 36 (1988) 27–47.
[29] R. Reiter, A theory of diagnosis from first principles, *Artificial Intelligence*, 32 (1987) 57–95.
[30] F. Rossi and S. A. Naqvi, Contributions to the view update problem, in: *Proceedings of the 6th International Conference on Logic Programming*, MIT Press, 1989, pp. 398–415.
[31] V. S. Subrahmanian, On the semantics of quantitative logic programs, in: *Proceedings of the 4th IEEE Symposium on Logic Programming*, IEEE Computer Society Press, 1987, pp. 173–182.
[32] A. van Gelder, K. A. Ross and J. S. Schlipf, The well-founded semantics for general logic programs, *Journal of ACM*, 38 (1991) 620–650.