ELSEVIER

The Journal of Logic Programming 35 (1998) 39–78

THE JOURNAL OF LOGIC PROGRAMMING

Negation as failure in the head

Katsumi Inoue [a,*], Chiaki Sakama [b,1]

[a] *Department of Electrical and Electronics Engineering, Kobe University, Rokkodai, Nada-ku, Kobe 657-8501, Japan*

[b] *Department of Computer and Communication Sciences, Wakayama University, Sakaedani, Wakayama 640-8441, Japan*

Received 1 August 1996; received in revised form 1 March 1997; accepted 9 April 1997

Abstract

The class of logic programs with negation as failure in the head is a subset of the logic of MBNF introduced by Lifschitz and is an extension of the class of extended disjunctive programs. An interesting feature of such programs is that the minimality of answer sets does not hold. This paper considers the class of *general extended disjunctive programs* (GEDPs) as logic programs with negation as failure in the head. First, we discuss that the class of GEDPs is useful for representing knowledge in various domains in which the principle of minimality is too strong. In particular, the class of abductive programs is properly included in the class of GEDPs. Other applications include the representation of inclusive disjunctions and circumscription with fixed predicates. Secondly, the semantic nature of GEDPs is analyzed by the syntax of programs. In acyclic programs, negation as failure in the head can be shifted to the body without changing the answer sets of the program. On the other hand, supported sets of any program are always preserved by the same transformation. Thirdly, the computational complexity of the class of GEDPs is shown to remain in the same complexity class as normal disjunctive programs. Through the simulation of negation as failure in the head, computation of answer sets and supported sets is realized using any proof procedure for extended or positive disjunctive programs. Finally, a simple translation of GEDPs into autoepistemic logic is presented.

1. Introduction

Logic programming has been regarded as an appropriate tool for knowledge representation in artificial intelligence. From this viewpoint, theories of logic programming with *negation as failure* (or *default negation*), *classical negation* (or *explicit*

* Corresponding author. Tel.: +81 78 803 1079; fax: +81 78 881 3193; e-mail: inoue@eedept.kobe-u.ac.jp.

[1] E-mail: sakama@sys.wakayama-u.ac.jp.

S0743-1066/98/$19.00
PII: S0743-1066(97)10001-2

negation) and *disjunctive information* have been developed (a comprehensive survey is found in [4]). This paper is concerned with such a theory of an extended class of logic programs as a knowledge representation language. Our extension of logic programs is called *general extended disjunctive programs* (GEDPs). Such programs allow negation as failure not only in the body of a rule as *negative premises* but in the head as *negative conclusions*. That is, a rule in a GEDP is in the form

$$L_1 | \ldots | L_k | not\ L_{k+1} | \ldots | not L_l \leftarrow L_{l+1}, \ldots, L_m, not L_{m+1}, \ldots, not L_n,$$

where each L_i is a positive or negative literal, *not* is the negation-as-failure operator, and "|" represents a disjunction. The class of GEDPs has the following nice properties:

- The syntax of programs is general enough to strictly include the class of *extended disjunctive programs* (EDPs). The class of GEDPs is a quite natural extension of EDPs in the sense that negation as failure appears symmetrically in a rule.
- The semantics of programs is also a natural generalization of the *answer set semantics* for EDPs by Gelfond and Lifschitz [22], thus coincides with the *stable model semantics* [21] for normal logic programs.
- The class of GEDPs is more expressive than the class of EDPs in the sense that a program can have a *non-minimal* answer set.

Furthermore, we will show that the introduction of negation as failure in the head to logic programming is useful and important in the following sense:

- A lot of new applications to knowledge representation can be properly described using GEDPs.
- The class of GEDPs offer a theoretical tool for investigating a theory of existing framework of logic programming. Specifically, it enables us to better understand the *supported model semantics* [3,45] and to have a proof procedure for it.
- There is a procedural semantics for GEDPs that is an extension of existing proof procedures for EDPs.
- There are close relationships between the class of GEDPs and existing non-monotonic formalisms, which are also natural extensions of previously known results [19,39,12].

1.1. Historical background

Historically, the class of GEDPs [2] was introduced by Lifschitz and Woo [41] as a subset of the logic of *minimal belief and negation as failure* (MBNF). MBNF was proposed by Lifschitz [38] as a general non-monotonic logic that includes the class of logic programs permitting both classical negation and negation as failure. In fact, MBNF is one of the most expressive logics and can serve as common framework that unifies several non-monotonic formalisms. As Lifschitz noted, however, MBNF is purely semantical and too intractable to be used directly for representing knowledge. Then, Lifschitz and Woo investigated a large subset of propositional MBNF called *PL-theories* – theories with "protected literals". In brief, protected literals are formulas of the forms BL and *not* L, where L is a literal and B and *not* are two non-monotonic modal operators, respectively, meaning minimal belief and negation as

[2] The name GEDP was introduced in [29].

failure. Then, a PL-theory is defined as a set of PL- formulas which are formed by protected literals using $\neg$, B, *not* and $\wedge$.

The semantics of PL-theories is similar to the answer set semantics for EDPs, and can be described in terms of sets of objective literals. Moreover, each PL-theory is shown to be replaced with an equivalent set of formulas of the form

$$\mathrm{B}L_1 \vee \cdots \vee \mathrm{B}L_k \vee not\, L_{k+1} \vee \cdots \vee not\, L_l$$
$$\vee \neg \mathrm{B}L_{l+1} \vee \cdots \vee \neg \mathrm{B}L_m \vee \neg not\, L_{m+1} \vee \cdots \vee \neg not\, L_n,$$

which is interpreted as the rule

$$L_1 \mid \ldots \mid L_k \mid not\, L_{k+1} \mid \ldots \mid not\, L_l \leftarrow L_{l+1}, \ldots, L_m, not\, L_{m+1}, \ldots, not\, L_n$$

in the logic programming context. Hence, the class of GEDPs is such a "logic programming" fragment of MBNF.

Lifschitz and Woo consider the possibility of *positive occurrences of negation as failure* (*positive not*, in short) in GEDPs. [3] Syntactically, this extension is quite natural and attractive, and each rule with negation as failure in the head can be regarded as a *bisequent* [8], that is, a pair of positive and negative beliefs appears in both the antecedent and the succedent of a sequent.

The semantics of GEDPs is also clearly defined in terms of the notion of answer sets [41]. A unique feature of GEDPs, which distinguishes them from other traditional logic programs, is that the minimality of answer sets for EDPs [22] does not hold in general. For example, the program consisting of the rule

$$p \mid not\, p \leftarrow$$

has two answer sets: one containing p and the other containing neither p nor $\neg p$. In this paper, we will analyze this peculiar property of GEDPs in detail from the two important viewpoints:

(i) applications of non-minimal answer sets in knowledge representation;

(ii) semantical and computational properties of non-minimal answer sets.

1.2. Non-minimality in knowledge representation

Most of the semantics of logic programs proposed so far satisfy the *principle of minimality* in some sense. For example, the least model semantics for definite Horn programs, the minimal model semantics for positive disjunctive programs, the perfect model semantics for stratified (disjunctive) programs, and the stable model semantics for normal (disjunctive) programs satisfy the principle in the sense that every canonical model of a logic program is its minimal model. The answer set semantics for EDPs also satisfies the principle since no answer set of a program is smaller than any other answer set. Hence, it has been argued that the principle of minimality is one of the most important criteria that each semantics should obey if it is used as "commonsense" semantics [59].

[3] There had been some attempts to allow negation as failure in the head before GEDPs appeared in the literature. For example, Gelfond showed such a rule in Example 2 of [20] and Kowalski and Kim considered the possibility in [35], p. 237. However, they did not discuss much about the effect of such rules, and their semantics are different from that of GEDPs.

The situation is similar in research on non-monotonic formalisms. Circumscription [47] is directly based on minimal models, and (disjunctive) default logic [52,23] has the property that an extension of a (disjunctive) default theory is not a subset of any other extension. While an exception can be seen in *autoepistemic logic* [49], the definition of *stable expansions* has been modified so that each obtainable expansion "rationally" satisfies the principle of minimality. For example, $\{Bp \supset p\}$ has two stable expansions, one containing p and the other not in their objective parts, but only the latter is the moderately (or strongly) grounded expansion [33].

On the other hand, recent advances on theories of logic programming and non-monotonic reasoning have revealed the declarative meaning of negation as failure as a non-monotonic modal operator. MBNF is a non-monotonic bimodal logic that directly allows the negation-as-failure operator *not* along with the B operator for minimal belief in a theory. Once *not* is allowed positively in a PL-theory or a GEDP, the principle of minimality does not hold any more. This observation gives a justification of the introduction of *not* in the head in logic programs. Namely, the class of GEDPs strictly includes the class of EDPs both in the syntactical and semantical senses.

Then, a question arises about the use of negation as failure positively in MBNF or logic programming. Namely, one may feel a resistance to the existence of non-minimal answer sets. From the traditional viewpoint, a non-minimal answer set contains a redundant information and is of no use for representing commonsense knowledge. In fact, Lifschitz and Woo raised a question about the utility of a disjunction of literals and their negation like $p \mid not\ p \leftarrow$, and discussed ([41], p. 608):

> It remains to be seen whether rules like this may have applications to knowledge representation.

In this respect, we will show that the non-minimality of answer sets is an important property for applying logic programming or MBNF to represent various domains in which the principle of minimality is too strong. For example, we show all the following applications can be characterized in terms of GEDPs.

- *Abductive logic programming*: Consider the logic program $\{p \leftarrow a\}$ with the abducible atom a. For this program, $\emptyset$ is the least model. However, given the observation p, the non-minimal model $\{a, p\}$ is considered as the intended *belief model.*
- *Inclusive interpretation of disjunctions*: When we interpret the disjunction $p \mid q \leftarrow$ exclusively, both $\{p\}$ and $\{q\}$ are two alternative minimal models. But if it is interpreted inclusively, the non-minimal model $\{p, q\}$ becomes another intended model.
- *Circumscription with fixed predicates*: The circumscription of p in $\{q \supset p\}$ with q fixed has two models, $\emptyset$ and $\{p, q\}$. Here, the second model is not minimal.

On the first point, we will show that the rule $a \mid not\ a \leftarrow$ can be used to represent the statement that a is a hypothesis in a program. The fact that abduction can be represented by a single logic program is a particularly striking result. Since an abductive program is usually represented by a pair of background knowledge and candidate hypotheses, it is important to know how such meta-level information of hypotheses can be expressed at the object level. Such an expression bridges the gap between abductive and usual (non-abductive) logic programming, and contributes to the computational aspect of abduction. Namely, we can apply any proof procedure for usual logic programs to abductive programs.

On the second point, an inclusive interpretation of the disjunction $p \mid q \leftarrow$ is specified by rules $p \mid not\ p \leftarrow$ and $q \mid not\ q \leftarrow$ together with the integrity constraint $\leftarrow not\ p, not\ q$. In the case, the answer sets of the program are $\{p\}$, $\{q\}$, and $\{p, q\}$, where the third one represents the inclusive model. Classical logic programming semantics based on minimal models are always minimal hence cannot represent such inclusive disjunctions in general. By contrast, the *possible model semantics* [11,53,55] has a non-minimal feature, and can represent both inclusive and exclusive disjunctions. We will show that the possible model semantics of positive/normal disjunctive programs are characterized by the answer set semantics for GEDPs.

On the third point, the fact that q is fixed in circumscription is also represented by the rule $q \mid not\ q \leftarrow$. In this sense, we can see that fixed predicates play the same role as abducible predicates in abductive logic programming. In classical logic programming, every predicate is usually minimized under the closed world reasoning. Fixed predicates are also considered in *ECWA* [24], which is equivalent to circumscription under some conditions. We will show that ECWA without varying predicates can be simply computed through GEDPs.

From the viewpoint of non-monotonic reasoning, among many non-monotonic formalisms, Moore's autoepistemic logic can express a stable expansion whose objective part is larger than that of another expansion. We show that this non-minimal feature of autoepistemic logic is applicable to describe the semantics of GEDPs. We justify this result by providing a simple translation of GEDPs into autoepistemic logic, which is due to the results by Lifschitz and Schwarz [39] and Chen [12].

1.3. Semantic nature and computation of GEDPS

One may consider that the use of positive *not* in MBNF or *not* in the head in GEDPs increases the computational complexity and that it is difficult to supply a procedural semantics in the presence of non-minimal answer sets. Two proof theories for MBNF proposed so far are not sufficient in this respect. Chen [12] proposes a proof theory for PL-theories, which relies on the proof theory for the logic of only knowing [36], so that a procedure would have to deal with modal logic K45. Beringer and Schaub [7] provide a proof procedure for a subset of MBNF, but this subset neither includes EDPs nor allows positive *not*.

In this regard, we will analyze the properties of GEDPs and the nature of *not* in the head from the viewpoint of *program transformation*.

First, we show a program transformation (called *shifting*) from a GEDP to an EDP such that the two programs have *exactly the same* answer sets. Such a transformation is possible if a GEDP satisfies the *acyclic* condition.

Secondly, we introduce an alternative semantics of GEDPs, called the *supported set semantics*, which is a natural generalization of the notion of *supported models*. Note that supported sets are not always minimal even for normal logic programs. For example, the logic program $\{p \leftarrow p\}$ has two supported models, $\emptyset$ and $\{p\}$. Unlike the answer set semantics, the supported set semantics is shown to be always preserved by the shifting transformation from GEDPs to EDPs. Moreover, the supported set semantics can be characterized by the answer set semantics. Hence, this gives another application of non-minimal answer sets of GEDPs. These analyses help us to better understand the source of non-minimality of answer sets in GEDPs

or supported sets in EDPs. As a by-product, we will have a procedure to compute the supported sets or supported models defined by [3,45,4,9].

Thirdly, we develop a polynomial-time translation of any GEDP into an EDP by replacing *not* in the head with new literals and constraints. With this translation, there is a *one-to-one correspondence* between the answer sets of the original GEDP and those of the translated EDP. This translation also contributes to the computational theory for GEDPs. Namely the computational complexity of GEDPs is shown to remain in the same complexity class as EDPs, and computation of answer sets of GEDPs is realized using bottom-up model generation techniques for EDPs.

1.4. Outline of the paper

This paper is a much extended version of the paper [29]. The rest of this paper is organized as follows. Section 2 gives the answer set semantics for GEDPs and their basic properties. Section 3 shows practical applications of non-minimal answer sets such as abduction, inclusive disjunctions and circumscription with fixed predicates, and characterizes these applications as GEDPs. Section 4 introduces the shifting transformation, which preserves the answer set semantics for acyclic GEDPs. Section 5 defines the supported set semantics for GEDPs, and compares it with the answer set semantics. Section 6 provides complexity results and computation of the answer set semantics for GEDPs. Section 7 shows connections to autoepistemic logic and other non-monotonic logics. Section 8 discusses some related issues, and Section 9 gives a summary.

2. General extended disjunctive programs

This section overviews the answer set semantics of logic programs with negation as failure in the head. We regard a rule with variables as the set of its ground instances. Hence, in the semantics of logic programming in this paper, we can restrict our attention to (possibly infinite) ground programs.

A *general extended disjunctive program* (GEDP) is a set of rules of the form

$$L_1 | \ldots | L_k | not\, L_{k+1} | \ldots | not\, L_l \leftarrow L_{l+1}, \ldots, L_m, not\, L_{m+1}, \ldots, not\, L_n, \quad (1)$$

where L_i 's are literals and $n \geqslant m \geqslant l \geqslant k \geqslant 0$. The disjunction to the left of $\leftarrow$ is the *head* and the conjunction to the right of $\leftarrow$ is the *body* of the rule. In GEDPs, negation as failure occurs *positively*, that is, *not* $L_j (k+1 \leqslant j \leqslant l)$ may appear in the head of a rule. In this sense, negation as failure in the head is also called *positive not*. Intuitively, the rule (1) can be read as follows: If all $L_{l+1}, \ldots, L_m$ are believed and all $L_{m+1}, \ldots, L_n$ are disbelieved than either some $L_i (1 \leqslant i \leqslant k)$ should be believed or some $L_j (k+1 \leqslant j \leqslant l)$ should be disbelieved.

A GEDP is called an *extended disjunctive program* (EDP) when it does not contain positive *not*, i.e, each rule is in the form (1) with $k = l$. An EDP is called (i) an *extended logic program* (ELP) if for each rule $l \leqslant 1$; and (ii) a *normal disjunctive program* (NDP) if every L_i is an atom. An NDP is called (i) a *normal logic program* (NLP) if for each rule $l \leqslant 1$; and (ii) a *positive disjunctive program* (PDP) if it contains no *not*, i.e., for each rule $m = n$.

In the following, the set of all ground literals in the language is denoted as *Lit*. We say that a set of ground literals $S \subseteq Lit$ *satisfies* a ground rule of the form (1) iff

$\{L_{l+1}, \ldots, L_m\} \subseteq S$ and $\{L_{m+1}, \ldots L_n\} \cap S = \emptyset$ imply either $\{L_1, \ldots, L_k\} \cap S \neq \emptyset$ or $\{L_{k+1}, \ldots, L_l\} \not\subseteq S$.

As the semantics for GEDPs, we mainly consider the answer set semantics in this paper, while we later introduce the *supported set semantics* as an alternative semantics in Section 5.

The *answer sets* of a GEDP are defined by the following two steps. First, let P be a *not*-free EDP (i.e., for each rule $k = l$ and $m = n$), and $S \subseteq Lit$. Then, S is an *answer set* of P iff S is a minimal set satisfying the conditions:

(i) S satisfies every ground rule from P, that is, for each ground rule

$$L_1 | \ldots | L_k \leftarrow L_{k+1}, \ldots, L_m$$

from P, if $\{L_{k+1}, \ldots, L_m\} \subseteq S$ then $\{L_1, \ldots, L_k\} \cap S \neq \emptyset$. In particular, for each ground rule $\leftarrow L_1, \ldots, L_m$ from $P, \{L_1, \ldots, L_m\} \not\subseteq S$;

(ii) If S contains a pair of complementary literals L and $\neg L$, then $S = Lit$.

Secondly, let Π be any GEDP, and $S \subseteq Lit$. The *reduct* Π^S *of* Π *by* S is a *not*-free EDP obtained as follows: A rule

$$L_1 | \ldots | L_k \leftarrow L_{l+1}, \ldots, L_m \tag{2}$$

is in Π^S iff there is a ground rule of the form

$$L_1 | \ldots | L_k | not\ L_{k+1} | \ldots | not L_l \leftarrow L_{l+1}, \ldots, L_m, not L_{m+1}, \ldots, not L_n$$

from P such that

$$\{L_{k-1}, \ldots, L_l\} \subseteq S \quad \text{and} \quad \{L_{m+1}, \ldots, L_n\} \cap S = \emptyset$$

For programs of the form Π^S, their answer sets have already been defined. Then, S is an *answer set* of Π iff S is an answer set of Π^S.

Note that the above definition of answer sets of a GEDP is given in a way slightly different from that by Lifschitz and Woo [41] who additionally include in the language two special atoms T and F. When the language does not contain these special atoms, our definition of the reduct is equivalent to that given in [41], p. 606, and thus both definitions of answer sets coincide. Obviously, when a program Π is an EDP, the above definition of answer sets reduces to that given by Gelfond and Lifschitz [22]. When a program contains no classical negation, answer sets are also called *stable models*. This notion of stable models for programs possibly containing positive *not* also reduces to that for NDPs [51] and NLPs [21].

The next proposition is a generalization of Proposition 4.1(a) in [4].

Proposition 2.1. *Every answer set of a GEDP* Π *satisfies every ground rule from* Π.

Proof. Let S be any answer set of Π. Since S is an answer set of Π^S, S satisfies every ground rule from Π^S. Namely, for any ground rule R of the form (2) from Π^S, if $\{L_{l+1}, \ldots, L_m\} \subseteq S$ then $\{L_1, \ldots, L_k\} \cap S \neq \emptyset$. By the construction of Π^S, for the rule R, there is a corresponding ground rule of the form (1) from Π such that $\{L_{m+1}, \ldots, L_n\} \cap S = \emptyset$ and $\{L_{k+1}, \ldots, L_l\} \subseteq S$. Hence for any ground rule R' of the form (1) from Π, if $\{L_{l+1}, \ldots, L_m\} \subseteq S$ and $\{L_{m+1}, \ldots, L_n\} \cap S = \emptyset$, then either $\{L_1, \ldots, L_k\} \cap S \neq \emptyset$ or $\{L_{k+1}, \ldots, L_l\} \not\subseteq S$. Hence, S satisfies R'. □

We say that a GEDP Π *entails* a literal L if L is included in all answer sets of Π. An answer set is *consistent* if it is not Lit. A GEDP Π is *consistent* if it has a consis-

tent answer set. An answer set S of a GEDP Π is *minimal* if no other answer set S' of Π satisfies $S' \subset S$; otherwise, it is *non-minimal*. It is well known that every answer set of any EDP is minimal ([22] and Theorem 4 of [41]). However, the minimality of answer sets no longer holds for GEDPs. This is an important property of GEDPs which was firstly observed by Lifschitz and Woo. For example, the program consisting of the rules:

$$q \leftarrow p,$$

$$p \,|\, not\, p \leftarrow ,$$

has two answer sets: $\emptyset$ and $\{p, q\}$.

3. Applications of negation as failure in the head

In this section, we show various applications of negation as failure in the head in GEDPs. The most important application is inference to explanation called *abduction*, which is one of the three fundamental modes of reasoning characterized by C.S. Peirce. We will also show that positive *not* is a useful tool to represent other non-minimal semantics for disjunctive logic programs, including the *possible model semantics* and *circumscription with fixed predicates*. Some other applications will also be presented.

3.1. Abductive programs

Abduction is an important form of reasoning not only for various AI problems but also for logic programming. *Abductive logic programming* is an extension of logic programming to perform abductive reasoning [32]. Here, we show that this extension can be characterized exactly using positive *not* in GEDPs, so that both abductive and non-abductive logic programming have the same expressive power.

The semantics of abduction we consider here is based on the *belief set semantics* by Inoue and Sakama [30], but is extended to handle GEDPs. The belief set semantics is a generalization of the *generalized stable model semantics* defined by Kakas and Mancarella [31] for NLPs.

An *abductive (general extended disjunctive) program* is a pair $\langle P, \Gamma \rangle$, where P is a (general extended disjunctive) program and Γ ($\subseteq Lit$) is a set of ground literals from P called *abducibles*. When P is an NLP and Γ is a set of atoms, we will often call an abductive program an *abductive NLP*. We often identify a set E ($\subseteq \Gamma$) of abducibles with the program $\{\gamma \leftarrow \mid \gamma \in E\}$. A set of literals S ($\subset Lit$) is a *belief set* of $\langle P, \Gamma \rangle$ iff S is a consistent answer set of $P \cup E$ where $E = S \cap \Gamma$.[4] A belief set S is Γ-*minimal* iff no belief set T satisfies that $T \cap \Gamma \subset S \cap \Gamma$.

When S is a belief set and $E = S \cap \Gamma$, we often write S as S_E. Note that each belief set reduces to a consistent answer set of P when $\Gamma = \emptyset$. Belief sets are called *belief*

[4] This definition can also be stated as follows: S is a *belief set* of $\langle P, \Gamma \rangle$ iff S is a consistent answer set of $P \cup E$ for some $E \subseteq \Gamma$. The set of all belief sets defined by each definition is equivalent, and hence Theorem 3.2 still holds for this alternative definition. Here, we prefer to identify the abducibles included in a belief set through the equation $E = S \cap \Gamma$.

models when P does not contain classical negation. Belief models are also called *generalized stable models* [31] when P is an NLP.

Let $\langle P, \Gamma\rangle$ be an abductive program, and O a ground literal which represents an *observation*. A set $E\,(\subseteq \Gamma)$ is a *credulous explanation of* O *(with respect to* $\langle P, \Gamma\rangle$*)* iff there is a belief set S_E which satisfies O. On the other hand, $E\,(\subseteq \Gamma)$ is a *skeptical explanation of* O iff every belief set S such that $E = S \cap \Gamma$ satisfies O. When we just say an *explanation*, it is a credulous explanation. An explanation E of O is *minimal* if no $E' \subset E$ is an explanation of O.

As discussed in [30], without loss of generality, we can assume that an observation O is a *non-abducible ground literal*. Furthermore, the problem to find explanations is essentially equivalent to find belief sets since E is a minimal explanation of O with respect to $\langle P, \Gamma\rangle$ iff S_E is a Γ-minimal belief set of $\langle P \cup \{\leftarrow not\, O\}, \Gamma\rangle$.

Example 3.1. Consider the abductive NLP $\langle P_1, \Gamma_1\rangle$ where P_1 consists of

$$P \leftarrow r, b, not\, q,$$

$$q \leftarrow a,$$

$$r \leftarrow$$

and $\Gamma_1 = \{a, b\}$. Then, $S_{E0} = \{r\}, S_{E1} = \{r, p, b\}, S_{E2} = \{r, q, a\}$ and $S_{E3} = \{r, q, a, b\}$ are the belief models of $\langle P_1, \Gamma_1\rangle$, in which S_{E0} is the only Γ_1-minimal belief model of $\langle P_1, \Gamma_1\rangle$. Suppose that p is an observation. Then, $E1 = S_{E1} \cap \Gamma = \{b\}$ is the (minimal) explanation of p. The observation p can be incorporated in the program as

$$P_2 = P_1 \cup \{\leftarrow not\, p\}.$$

and the unique belief model of $\langle P_2, \Gamma_1\rangle$ is $S_{E1} = \{r, p, b\}$. Note that $E3 = \{a, b\}$ is not an explanation of p. Hence, abduction is non-monotonic relative to the addition of abducibles.

The most direct way to embed abducibles in a single program is as follows. Let $\langle P, \Gamma\rangle$ be an adductive program. For each abducible γ in Γ, we supply the rule

$$\gamma \,|\, not\, \gamma \leftarrow . \tag{3}$$

According to the non-minimality of answer sets of GEDPs, this rule has the effect to augment each answer set of P with either γ or nothing. Given an abductive program $\langle P, \Gamma\rangle$, let $abd(\Gamma)$ be the set of rules (3) obtained from Γ.

Theorem 3.2. *A set S is a belief set of $\langle P, \Gamma\rangle$ iff S is a consistent answer set of $P \cup abd(\Gamma)$.*

Proof. Let $E = S \cap \Gamma$. It holds that $abd(\Gamma)^S = abd(\Gamma)^E = E = E^S$.

Hence, S is a belief set of $\langle P, \Gamma\rangle$
iff S is a consistent answer set of $P \cup E$
iff S is a consistent answer set of $P^S \cup E^S$
iff S is a consistent answer set of $P^S \cup abd(\Gamma)^S$
iff S is a consistent answer set of $P^S \cup abd(\Gamma)$. □

Given a GEDP Π and a set Γ of ground literals, we say an answer set S of Π is Γ-*minimal* if no other answer set S' of Π satisfies that $S' \cap \Gamma \subset S \cap \Gamma$.

Corollary 3.3. *A set $E (\subseteq \Gamma)$ is a minimal explanation of O with respect to $\langle P, \Gamma \rangle$ iff $S_E (\subset Lit)$ is a consistent Γ-minimal answer set of $P \cup \{\leftarrow not\, O\} \cup abd(\Gamma)$.*

Example 3.4. The abductive program $\langle P_2, \Gamma_1 \rangle$ in Example 3.1 is transformed to

$$P_2 \cup abd(\Gamma_1) = P_2 \cup \{a \,|\, not\, a \leftarrow,\ \ b \,|\, not\, b \leftarrow\}.$$

Then, $\{r, p, b\}$ is the unique (and hence both minimal and Γ_1- minimal) answer set of $P_2 \cup abd(\Gamma_1)$, which is exactly the (Γ_1-minimal) belief model of $\langle P_2, \Gamma_1 \rangle$. Notice in this example that there is no non-minimal answer set of $P_2 \cup abd(\Gamma_1)$. In other words, embedding abducibles in rules with positive *not* (3) not only enables us to represent non-Γ- minimal belief sets of abductive programs, but plays an important role to obtain a (minimal) explanation.

Example 3.5. Here is an example taken from Example 4.9 in [30], showing an effect of non-minimal answer sets in abductive reasoning. Consider the abductive program consisting of three rules P_3:

$$p \,|\, q \leftarrow not\, r,$$
$$r \leftarrow not\, a,$$
$$\neg q \leftarrow b$$

and two abducible literals $\Gamma_3 = \{a, b\}$. The information of abducibles in Γ_3 can be encoded as rules $abd(\Gamma_3)$:

$$a \,|\, not\, a \leftarrow,$$
$$b \,|\, not\, b \leftarrow .$$

When the observation is p, both $E = \{a\}$ and $E' = \{a, b\}$ are *credulous* explanations of p, and correspondingly, both $S = \{a, p\}$ and $S' = \{a, p, b, \neg q\}$ are answer sets of $P_3 \cup abd(\Gamma_3)$ containing p. Then, E is the minimal explanation of p. However, $P_3 \cup E$ has another answer set $\{a, q\}$ which does not contain p, while S' is the unique answer set of $P_3 \cup E'$. Hence, E' is preferable as the *skeptical* explanation of p, although its corresponding answer set S' of the GEDP is not Γ_3-minimal.

3.2. Assumptions with preconditions

In Section 3.1, a set Γ of a abducibles in an abductive program $\langle P, \Gamma \rangle$ was defined as a set of literals. Often however, we would like to introduce in Γ an *abducible rule* like

$$\gamma \leftarrow L_1, \ldots, L_m, not\, L_{m+1}, \ldots, not\, L_n, \tag{4}$$

where γ and L_i's are literals. This abducible rule intuitively means that if the rule is abduced then it is used for inference together with the background rules from P. This kind of extended abductive framework was introduced by Inoue [26] as a *knowledge system* in which both P and Γ are defined as ELPs, and has been shown to be a useful tool for representing commonsense knowledge.

An abducible rule (4) has the effect to introduce the literal γ as an assumption in a particular context in which the body of the rule is true. In this sense, γ in (4) can be considered as an *assumption with preconditions*. On the other hand, each abducible literal γ in an abductive program $\langle P, \Gamma \rangle$ defined in Section 3.1 is viewed as an abducible rule without precondition $\gamma \leftarrow$, and hence can be abduced globally.

An extended abductive framework can be formally defined as a pair $\langle P, \Gamma \rangle$, where P is a GEDP and Γ is now an ELP consisting of rules of the form (4). The semantics of this abductive framework is slightly extended from that given in Section 3.1 as follows. For any ELP E, let $head(E)$ be the heads of rules in E. A set of literals S ($\subset Lit$) is a *belief set* of $\langle P, \Gamma \rangle$ iff S is a consistent answer set of $P \cup E$ where $E \subseteq \Gamma$ such that $head(E) = S \cap head(\Gamma)$. Clearly, this notion of belief sets reduces to the definition of belief sets in Section 3.1 when Γ is a set of abducible literals without preconditions.

Example 3.6. Suppose that $\langle P_4, \Gamma_4 \rangle$ is an abductive program where

$$P_4 = \{p \leftarrow a, \quad \neg p \leftarrow b, \quad q \leftarrow c\},$$
$$\Gamma_4 = \{a \leftarrow, \quad b \leftarrow, \quad c \leftarrow p\}.$$

Then $\langle P_4, \Gamma_4 \rangle$ has the four belief sets: $\emptyset, \{a, p\}, \{a, p, c, q\}$, and $\{b, \neg p\}$. Notice that $\{b, \neg p, c, q\}$ is not a belief set since c can be assumed only when p is true.

The embedding of assumptions with preconditions in GEDPs is a straightforward generalization of that of abducibles without preconditions. Each rule (4) in Γ is replaced with the rule

$$\gamma \,|\, not\ \gamma \leftarrow L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n. \tag{5}$$

For example, the abducible rules Γ_4 given in Example 3.6 are embedded in

$$a \,|\, not\, a \leftarrow,$$
$$b \,|\, not\, b \leftarrow,$$
$$c \,|\, not\, c \leftarrow p.$$

Theorem 3.7. *Let $\langle P, \Gamma \rangle$ be an abductive framework, and $abd(\Gamma)$ the set of rules* (5) *obtained from the rules* (4) *in Γ. A set S is a belief set of $\langle P, \Gamma \rangle$ iff S is a consistent answer set of $P \cup abd(\Gamma)$.*

Proof. Similar to the proof of Theorem 3.2. □

In the next section, we show that abducible rules are also useful to represent inclusive disjunctions in disjunctive programs.

3.3. Inclusive interpretation of disjunctions

Another important application of positive *not* is to express an alternative semantics for disjunctive logic programs other than Gelfond and Lifschitz's answer set semantics. Here, we show that the *possible model semantics* for NDPs by Sakama and Inoue [55] can be characterized by the answer set semantics for GEDPs.

The possible model semantics was initially introduced for PDPs to enable one to specify both inclusive and exclusive interpretations of disjunctions [53,11] [5]. Sakama and Inoue [56] have presented the equivalence between the possible model semantics for NDPs and the belief model semantics for abductive NLPs. Utilizing this result

[5] Possible model semantics is also called *possible world semantics* in [11]. While Chan [11] gives a different definition from that by [53], these notions are proved equivalent.

and Theorem 3.7, the embedding of the possible model semantics in GEDPs can be obtained. We show below a direct method to do it based on the embedding of abducible rules in GEDPs in Section 3.2.

For an NDP P, let $disj(P)$ be the disjunctive rules of P, i.e., those rules having more than one atom in their heads. A *split program* of P is a ground NLP obtained from P by replacing each ground disjunctive rule from $disj(P)$ of the form

$$A_1 \mid \ldots \mid A_k \leftarrow A_{k+1}, \ldots, A_m, not\, A_{m+1}, \ldots, not\, A_n \quad (k > 1) \tag{6}$$

with rules

$$A_i \leftarrow A_{k+1}, \ldots, A_m, not\, A_{m+1}, \ldots, not\, A_n \quad \text{for every } A_i \in S,$$

where S is some non-empty subset of $\{A_1, \ldots, A_k\}$. Then, a *possible model* of P is defined as an answer set (or stable model) of a split program of P [55]. Note that every stable model of P is a possible model of P, but not vice versa. For example, when

$$P_5 = \{p \mid q \leftarrow,\ \ q \leftarrow p,\ \ r \leftarrow not\, p\},$$

$\{q, r\}$ is both a stable model and a possible model of P_5, but another possible model $\{p, q\}$ is not a stable model of P_5. Clearly, for NLPs, possible models coincide with stable models.

To obtain every possible model, let us consider the transformation pm which maps an NDP to a GEDP. Given an NDP P. $pm(P)$ is obtained by replacing every rule from P of the form (6) with the $k + 1$ rules

$$A_i \mid not\, A_i \leftarrow A_{k+1}, \ldots, A_m, not\, A_{m+1}, \ldots, not\, A_n \quad \text{for } i = 1, \ldots, k, \tag{7}$$

$$\leftarrow A_{k+1}, \ldots, A_m, not\, A_{m+1}, \ldots, not\, A_n, not\, A_1, \ldots, not\, A_k. \tag{8}$$

Recall that the embedding of abducible rules (4) in GEDPs was based on rules (5). The embedding of possible models is achieved in a similar manner by rules (7) except that the empty selection from the disjuncts of each disjunction is rejected by (8) in the transformation pm.

Lemma 3.8 [56]. *Let P be an NDP, and $disj(P)$ the disjunctive rules of P. Suppose that Γ is the NLP obtained from $disj(P)$ by replacing each disjunctive rule* (6) *with k rules*

$$A_i \leftarrow A_{k+1}, \ldots, A_m, not\, A_{m+1}, \ldots, not\, A_n \quad \text{for } i = 1, \ldots, k,$$

and that IC is the set of rules of the form (8) *obtained from the rules of the form* (6) *in $disj(P)$. Then, a set S of atoms is a possible model of P iff S is a belief model of the abductive program $\langle (P \backslash disj(P)) \cup IC, \Gamma \rangle$.*

Theorem 3.9. *Let P be an NDP. A set S of atoms is a possible model of P iff S is an answer set of $pm(P)$.*

Proof. A direct consequence of Theorem 3.7 and Lemma 3.8. □

Example 3.10 [11]. Suppose that the NDP P_6 consists of three rules

$$violent \mid psychopath \leftarrow suspect,$$

$$dangerous \leftarrow violent, psychopath,$$

$$suspect \leftarrow .$$

Here, the first rule is replaced in $pm(P_6)$ with three rules

$$violent \mid not\ vilolent \leftarrow suspect,$$

$$psychopath \mid not\ psychopath \leftarrow suspect,$$

$$\leftarrow suspect,\ not\ violent, not\ psychopath.$$

Then, $pm(P_6)$ has three answer sets, $\{suspect, violent\}$, $\{supect, psychopath\}$ and $\{suspect,\ violent,\ psychopath,\ dangerous\}$, which coincide with the possible models of P_6. Note that the first and second possible models of P_6 are also the answer sets of P_6, while the third possible model is not. If we introduce the rule of *closed world assumption* [22]

$$\neg A \leftarrow not\, A \quad \text{for any atom } A$$

into P_6, then $\neg dangerous$ is entailed in the answer set semantics, which is too strong. By contrast, $\neg dangerous$ is not entailed in both the possible model semantics for P_6 and the answer set semantics for $pm(P_6)$.

3.4. Fixed predicates

One of the interesting differences between *circumscription* [47,37] and disjunctive logic programming is the existence of *fixed predicates*. As a typical example, we expect that something does not fly by default, but if it is a bird then it files. We can write this as an axiom like

$$bird \supset flies$$

with *flies* minimized using circumscription. Without knowing initially whether it is a bird or not, we can even deduce $\neg flies$ if *bird* is also minimized (as in standard logic programming) or is allowed to vary (as in circumscription). In this case, $\neg bird$ is then concluded. However, this side effect about the bird may not be desired in many cases. This problem can be avoided if *bird* is fixed (i.e., not allowed to vary) in circumscribing *flies*. The circumscription actually deduces

$$\neg bird \supset \neg flies,$$

so we conclude that it does not fly unless it is a bird.

In classical logic programming, every predicate is usually minimized in a PDP by *GCWA* [48], in which the answer sets of the program are exactly the minimal Herbrand models. An exception can be seen in *ECWA* proposed by Gelfond et al. [24], which is equivalent to circumscription in the existence of the unique-name and domain-closure assumptions. We now formalize ECWA for PDPs without varying predicates.

Let T be a PDP consisting of rules of the form

$$A_1 \mid \ldots \mid A_k \leftarrow B_1, \ldots, B_m \quad (k, m \geqslant 0),$$

where A_i's and B_j's are atoms. This rule can be identified with a first-order formula

$$B_1 \wedge \cdots \wedge B_m \supset A_1 \vee \ldots \vee A_k.$$

Let P be the set of minimized predicates. The set of all predicates other than those in P are written Q and assumed to be fixed. The following notation is due to Lifschitz

[37]. For any two distinct models M and N of T, we write $M \leqslant N$ if: (i) $|M| = |N|$, (ii) $M[\![q]\!] = N[\![q]\!]$ for every predicate q in Q, and (iii) $M[\![p]\!] \subseteq N[\![p]\!]$ for every predicate p in P. A model M of T is *P-minimal* iff $N \leqslant M$ implies $M \leqslant N$ for any model N of T.

The information of fixed predicates can be encoded in GEDPs in the same way as the encoding of abducibles in abductive programs. Now let

$$cir(T,P) = T \cup \{q(\mathbf{x}) \,|\, not\, q(\mathbf{x}) \leftarrow \,|\, q \in Q\},$$

where $\mathbf{x} = x_1, \ldots, x_n$ is a tuple of variables for n-ary predicate q in Q.

Theorem 3.11. *Let T be a PDP, and P the minimized predicates. Then, M is a P-minimal Herbrand model of T iff M is an answer set of $cir(T,P)$.*

Proof. Let M be a P-minimal Herbrand model of T such that $M \cap Q = \Psi$. Here, we also use Q to denote the set of ground atoms with predicates from Q. Then, M is a minimal Herbrand model (i.e., $(P \cup Q)$-minimal Herbrand model) of $T \cup \Psi$. Here, for each $q \in \Psi$, there is a ground rule of the form

$$q \,|\, not\, q \leftarrow$$

from $cir(T,P)$. Thus, $T \cup \Psi = T^M \cup \Psi^M = cir(T,P)^M$. Therefore, M is a minimal Herbrand model of $cir(T,P)^M$, hence an answer set of $cir(T,P)$.

On the other hand, let M be an answer of $cir(T,P)$ such that $M \cap Q = \Psi$. Then, M is a minimal Herbrand model of $T \cup \Psi$, hence a P-minimal Herbrand model of T. □

In the bird example above, the axiom set can be written as

$$flies \leftarrow bird,$$

$$bird \,|\, not\ bird \leftarrow,$$

which has two answer sets, $\emptyset$ and $\{bird, flies\}$ expressing $flies \equiv bird$.

It should be noted that the representation of fixed predicates by positive *not* is in the same form as that of abducibles in Section 3.1. In this sense, we can see that fixed predicates play the same role as *abducible predicates* in abductive logic programming.

Theorem 3.12. *Let T be a PDP, and P the minimized predicates. Suppose that Γ is $\{q(\mathbf{x}) \,|\, q \in Q\}$ where $\mathbf{x} = x_1, \ldots x_n$ is a tuple of variables for n-ary fixed predicate q in Q. Then, M is a P-minimal Herbrand model of T iff M is a belief model of $\langle P, \Gamma \rangle$.*

Proof. Follows from Theorem 3.2 and 3.11. □

When some predicates are allowed to vary, Sakama and Inoue [57] show that circumscription of a clausal theory can be embedded in GEDPs, in which minimized predicates are specified using negation as failure in bodies of rules, while fixed and varying predicates are expressed by negation as failure in heads. A generalization of Theorem 3.12 is also stated in Theorem 4.2 of [57]. Furthermore, *prioritized circumscription* [37] is shown to be expressed using positive *not* in a logic programming framework extended with priorities [58].

3.5. All-or-nothing

The embeddings of abducibles, possible models and fixed predicates in GEDPs in previous subsections are all based on the generation of the *power set* of a literal set. Namely, given a set Γ of literals, the rules

$$\gamma_i \,|\, not\ \gamma_i \leftarrow \quad \text{for each } \gamma_i \in \Gamma,$$

can be used to produce 2^{Γ}. There are many other variations for representing knowledge with positive *not*. For instance, for a finite set of literal $\Gamma = \{\gamma_1, \ldots, \gamma_n\}$, the *all-or-noting* choice can be represented by rules

$$\begin{aligned}
&\gamma_1 \,|\, not\ \gamma_2 \leftarrow,\\
&\gamma_2 \,|\, not\ \gamma_3 \leftarrow,\\
&\quad\vdots\\
&\gamma_{n-1} \,|\, not\ \gamma_n \leftarrow,\\
&\gamma_n \,|\, not\ \gamma_1 \leftarrow,
\end{aligned}$$

which generate an answer set containing all γ_i's and an answer set containing no γ_i.

Example 3.13. John and Mary are a couple. So, "if John is at the party, so is Mary, and vice versa". If we represent this situation by the rules Π_1:

$$\begin{aligned}
&\textit{Mary-at-Party} \leftarrow \textit{John-at-Party},\\
&\textit{Jonh-at-Party} \leftarrow \textit{Mary-at-Party},
\end{aligned}$$

then we get the answer set $\emptyset$ only. Instead, with the rules Π_2:

$$\begin{aligned}
&\textit{Mary-at-Party} \,|\, \textit{not John-at-Party} \leftarrow,\\
&\textit{Jonh-at-Party} \,|\, \textit{not Mary-at-Party} \leftarrow,
\end{aligned}$$

we have two possibilities: $\emptyset$ and $\{\textit{John-at-Party}, \textit{Mary-at-Party}\}$.

The above example shows the difference between a rule of the form $p \leftarrow q$ and a rule of the form $p \,|\, not\, q \leftarrow$, which will be analyzed more deeply in Section 4.

4. Reduction to extended disjunctive programs

A GEDP has non-minimal answer sets in general. Then, our question is which class of GEDPs has non-minimal answer sets can be distinguished from EDPs. Answering this question highlights the effect of negation as failure in the head and clarifies the expressiveness of GEDPs. In this section, we analyze the syntactic nature of GEDPs and investigate the relations between GEDPs and EDPs.

We first consider the possibility of transforming GEDPs to semantically equivalent EDPs. Since a GEDP may have a non-minimal answer set, the following claim holds.

Observation 4.1. *There is no transformation* **tr** *from GEDPs to EDPs such that any GEDP* Π *and* $\mathbf{tr}(\Pi)$ *have exactly the same answer sets.*

Our concern is a subclass of GEDPs that have such a transformation to EDPs with the equivalence preserved. Consider, for example, the program Π_3 which consists of one rule

$$p \mid not\, q \leftarrow .$$

Π_3 has just one answer set $\emptyset$. Note that $\{p\}$ is not an answer set since the reduct of Π_3 by $\{p\}$ is empty. The above rule can be read as "p is believed or q is not believed", and hence can be viewed as a conditional formula stating that "p is believed if q is believed". In this sense, the rule is similar to

$$p \leftarrow q.$$

In this case, the former rule can actually be replaced with the latter rule by shifting positive *not* into the body. However, suppose that one rule is added to Π_3:

$$\Pi_4 = \Pi_3 \cup \{q \mid not\, p \leftarrow\} .$$

Then, $\emptyset$ is still an answer set of Π_4, but now $\{p, q\}$ becomes another, non-minimal answer set. In fact, Π_4 has the same structure as Π_2 in Example 3.13. Hence, once a "deadlock" loop is constructed with these conditional formulas, a program may have two alternative answer sets, one including every element of the loop and the other including nothing in the loop, In other words, unless there is such a loop, *not* in the head can be shifted into the body without changing the answer sets. To formally identify such cases where positive *not* is not needed, the notion of *acyclic* GEDPs is introduced in the next section, and show that they reduce to EDPs.

4.1. Acyclic GEDPs

We first define acyclic GEDPs. In the following, a *level mapping* for a GEDP Π is any mapping $l: Lit \to \mathbb{N}$ of ground literals in the language of Π to natural numbers [2]. For any $L \in Lit$, we call $l(L)$ the *level* of L.

Let Π be any GEDP. Suppose that any ground rule from Π is in the form

$$A_1 \mid \ldots \mid A_k \mid not\, B_1 \mid \ldots \mid not\, B_l \leftarrow C_1, \ldots, C_m, not\, D_1, \ldots, not\, D_n, \tag{9}$$

where A_i's, B_j's, C_s's, and D_t's $(k, l, m, n \geqslant 0)$ are literals.

(a) Π is *positive acyclic (P-acyclic)* if there is a level mapping l for Π such that $l(A_i) > l(C_s)$ for every $i = 1, \ldots, k$ and $s = 1, \ldots, m$ for any ground rule from Π; otherwise it is *P-cyclic*.

(b) Π is *negative acyclic (N-acyclic)* if there is a level mapping l for Π such that $l(A_i) > l(B_j)$ for every $i = 1, \ldots, k$ and $j = 1, \ldots, l$, and $l(A_i) \geqslant l(C_s)$ for every $i = 1, \ldots, k$ and $s = 1, \ldots, m$ for any ground rule from Π; otherwise it is *N-cyclic*.

(c) Π is *acyclic* if it is both P-acyclic and N-acyclic. Namely, Π is acyclic if there is a level mapping l for Π such that $l(A_i) > l(B_j)$ for every $i = 1, \ldots, k$ and $j = 1, \ldots, l$ and $l(A_i) > l(C_s)$ for every $i = 1, \ldots, k$ and $s = 1, \ldots, m$ for any ground rule from Π.

The notion of N-acyclic property is introduced in order to check the possibility of a well-founded numbering of two literals, L_1 and L_2, where L_1 appears in the head of a rule and *not* L_2 appears in the head of the same or a different rule. The notion of acyclic programs has been discussed by several researches [2,14,6]. Apt and Bezem [2] defined acyclic NLPs, and Dung [14] extended the notion to acyclic NDPs. Their level mappings for acyclic programs additionally require that $l(A_i) > l(D_t)$ for every negation-as-failure formula *not* $D_t (t = 1, \ldots, n)$ in the body of any rule of the form (9).

We ignore the level of each such literal as it is not necessary in the subsequent discussion. Dung [14] also introduced P-acyclic programs, which coincide with our (P-)acyclic NDPs. Our notion of acyclic programs is also a generalization of propositional acyclic EDPs defined by Ben-Eliyahu and Dechter [6].

Observation 4.2. *Any EDP is N-acyclic. This is because every literal in Lit can have the same level in this case. Hence, an EDP is acyclic iff it is P-acyclic.*

Example 4.3. A GEDP consisting of rules

$$p \leftarrow q,$$

$$q \,|\, not\, p \leftarrow$$

is P-acyclic but N-cyclic. It has two answer sets, $\emptyset$ and $\{p, q\}$.

Example 4.4. Let Π_5 be the program consisting of rules

$$p(x) \leftarrow p(s(x)),$$

$$q(0) \leftarrow .$$

Π_5 is not (P-)acyclic since it must have an infinite decreasing chain of levels. Hence, Π_5 is P-cyclic by definition.

Let Π be any GEDP. The EDP *shift*(Π) is obtained from Π by replacing every rule of the form (9) with the rule

$$A_1 \,|\, \ldots \,|\, A_k \leftarrow B_1, \ldots, B_l, C_1, \ldots, C_m, not\, D_1, \ldots, not\, D_n \tag{10}$$

The mapping *shift* of GEDPs to EDPs is called the *shifting* transformation.

The shifting transformation eliminates every positive *not* from a GEDP in the simplest way. In fact, *shift* does not introduce any new literal into the language. This is compared with another translation *edp* that will be presented in Section 6.1, which requires new atoms to simulate positive *not* (see Remark 6.3). Note that a GEDP Π is acyclic if *shift*(Π) is (P-)acyclic. Now, we show that *the shifting transformation is sound with respect to the answer set semantics for all GEDPs and is complete for N-acyclic GEDPs.*

Lemma 4.5. *Let Π be any GEDP. Every answer set of shift (Π) is a minimal answer set of Π.*

Proof. Let S be an answer set of *shift*(Π). Then, S is an answer set of *shift*$(\Pi)^S$. That is, S is a minimal set satisfying the conditions: (i) for any ground rule of the form (10) from *shift*(Π) such that $\{D_1, \ldots, D_n\} \cap S = \emptyset$ (i.e., for any rule of the form

$$A_1 \,|\, \ldots \,|\, A_k \leftarrow B_1, \ldots, B_l, C_1, \ldots, C_m \tag{11}$$

in *shift*$(\Pi)^S$), if $\{B_1, \ldots, B_l, C_1, \ldots, C_m\} \subseteq S$ then $\{A_1, \ldots, A_k\} \cap S \neq \emptyset$; (ii) if S contains a pair of complementary literals then $S = Lit$. In these two conditions, (i) implies the fact that: for any ground rule of the form (9) from Π such that $\{B_1, \ldots, B_l\} \subseteq S$ and $\{D_1, \ldots, D_n\} \cap S = \emptyset$ (i.e., for any rule of the form

$$A_1 \,|\, \ldots \,|\, A_k \leftarrow C_1, \ldots, C_m \tag{12}$$

in Π^S), if $\{C_1, \ldots, C_m\} \subseteq S$ then $\{A_1, \ldots, A_k\} \cap S \neq \emptyset$. That is, S is a minimal set satisfying every ground rule from Π^S. Therefore, S is an answer set of Π^S, and hence an answer set of Π.

To prove the minimality of S in the answer sets of Π, suppose to the contrary that there is an answer set T of Π such that $T \subset S$. Then, there is a ground rule R of the form (12) in $\Pi^S \setminus \Pi^T$ such that: (i) $\{C_1, \ldots, C_m\} \subseteq S$, (ii) $L \in \{A_1, \ldots, A_k\}$ for some ground literal $L \in S \setminus T$, and (iii) $\{B_1, \ldots, B_l\} \subseteq S$ but $\{B_1, \ldots, B_l\} \not\subseteq T$. Since $\{D_1, \ldots, D_n\} \cap S = \emptyset$ and $T \subset S$ imply $\{D_1, \ldots D_n\} \cap T = \emptyset$, there is the corresponding ground rule R' of the form (11) in $shift(\Pi)^T$. By (i), (ii) and (iii), both S and T satisfy R'. This implies that both S and T satisfy every rule in $shift(\Pi)^T$, and hence satisfy every rule in $shift(\Pi)^S$ (by $shift(\Pi)^S \subseteq shift(\Pi)^T$). However, since S is an answer set of $shift(\Pi)^S$, S is a minimal set satisfying the rules of $shift(\Pi)^S$. This contradicts the supposition that $T (\subset S)$ satisfies the rules of $shift(\Pi)^S$. Therefore, no such T exists. □

Theorem 4.6. *let* Π *be an N-acyclic GEDP, and* $S \subseteq Lit$. S *is an answer set of* Π iff S *is an answer set of* $shift(\Pi)$.

Proof. The if-part directly follows from Lemma 4.5. To prove the only-if part, suppose to the contrary that S is an answer set of Π but is not an answer set of $shift(\Pi)$. Then, S is not a answer set of $shift(\Pi)^S$. For each ground rule R of the form (11) in $shift(\Pi)^S$ such that $\{B_1, \ldots, B_l\} \subseteq S$, there is the corresponding rule R' of the form (12) in Π^S. Since S is an answer set of Π^S, S satisfies R', and thus also satisfies R. Thus, S satisfied every ground rule in $shift(\Pi)^S$. Then, since S is not an answer set of $shift(\Pi)^S$, there is a set T of literals such that $T \subset S$ and T satisfies every ground rule in $shift(\Pi)^S$. Now, two cases are considered, and both are shown to derive contradiction.

(a) S *is consistent*. In this case, there must be a ground literal $L_1 \in S \setminus T$ and a ground rule of the form (11) in $shift(\Pi)^S$ such that: (i) $\{A_1, \ldots, A_k\} \cap S = \{L_1\}, \{C_1, \ldots, C_m\} \subseteq S$, and $\{B_1, \ldots B_l\} \subseteq S$ (by Lemma 5.2 in Section 5), and (ii) $\{C_1, \ldots, C_m\} \subseteq T$ but $\{B_1, \ldots, B_l\} \not\subseteq T$ (i.e., the source of non-minimality lies in some positive *not* by Observation 4.1). Now, let L_2 be a literal such that $L_2 \in \{B_1, \ldots, B_l\} \setminus T$. Since Π is N-acyclic, $l(L_1) > l(L_2)$. Then, for $L_2 \in S \setminus T$, there is a ground rule of the form (11) in $shift(\Pi)^S$ such that (i) $\{A_1, \ldots, A_k\} \cap S = \{L_2\}$, $\{C_1, \ldots, C_m\} \subseteq S$, and $\{B_1, \ldots, B_l\} \subseteq S$ (by Lemma 5.2 in Section 5), and (ii) $\{C_1, \ldots, C_m, B_1, \ldots, B_l\} \not\subseteq T$ (otherwise L_2 must be in T). Let L_3 be a literal such that $L_3 \in \{C_1, \ldots, C_m, B_1, \ldots, B_l\} \setminus T$. Notice that $l(L_2) \geqslant l(L_3)$. Repeating this arguments generates a sequence of literal levels

$$l(L_1) > l(L_2) \geqslant l(L_3) \geqslant l(L_4) \geqslant \ldots$$

Then, there must be some k $(2 \leqslant k < \infty)$ such that

$$l(L_{k-1}) > l(L_k) \quad \text{and} \quad l(L_k) = l(L_{k+1}) = l(L_{k+2}) = \cdots,$$

because otherwise, for any k there is $l(> k)$ such that $l(L_k) > l(L_l)$, so that there is an infinite decreasing chain $l(L_k) > l(L_l) > \cdots$, which contradicts the well-foundedness of the level mapping. Let

$$S' = S \setminus \{L_k, L_{k+1}, L_{k+2}, \ldots\}.$$

Now, for each $L_i \in S \setminus T (i \geqslant k), L_i \in \{C_1, \ldots, C_m\}$ and $L_i \notin \{B_1, \ldots, B_l\}$ hold for some rule R of the form (11) in $shift(\Pi)^S$. On the other hand, for each R, there is the corresponding rule R' of the form (12) in Π^S. Obviously, $S' (\subset S)$ satisfies every R' and other rules in Π^S. This contradicts the fact that S is an answer set of Π^S.

(b) $S = Lit$. In this case, consider the *positive form* Π^+ of Π, which is obtained from Π by replacing every negative literal $\neg L$ with a new atom L'. Since Lit is an answer set of Π^{Lit}, there is an answer set $S' \subseteq Lit^+$ of Π^{Lit^+} such that S' contains at least one pair of contradictory literals, L_1 and L_1'. Moreover, S' satisfies every rule in $shift(\Pi^+)^{Lit^+}$. On the other hand, $T \subset S = Lit$ implies that T contains no such contradictory pair of literals. Thus, there is a set $T' \subset S'$ such that T' satisfies every rule in $shift(\Pi^+)^{Lit^+}$ and that T' does not contain both of L_1 and L_1'. Let us assume that $L_1 \in S' \setminus T'$. Then, starting from L_1, we can construct a decreasing chains of levels of literals as in the proof of the case (a), again contradicting the fact that S' is an answer set of Π^{+Lit^+}. □

Corollary 4.7. *Every answer set of an N-acyclic GEDP is minimal.*

Proof. Follows from Lemma 4.4 and Theorem 4.6. □

Example 4.8. consider the N-acyclic GEDP.

$$p \mid not\, r \leftarrow q,$$

$$q \leftarrow r,$$

$$r \leftarrow .$$

This program has the unique answer set $\{p, q, r\}$, which is also the answer set of the program obtained by replacing the first rule with the shifted rule

$$p \leftarrow q, r.$$

Recall that the class of N-acyclic GEDPs properly includes the class of EDPs (Observation 4.2). Now, Theorem 4.6 shows that the shifting transformation preserves the answer set semantics for N-acyclic GEDPs. This fact implies that an (N)-acyclic GEDP can always be reduced to an EDP without changing the answer sets. In other words, *N-acyclic GEDPs collapse to EDPs*. Then, the next question is when positive *not* is really effective. Interestingly, not all of positive *not* are needed even for N-acyclic programs. In fact, the shifting transformation in Theorem 4.6 can also be applied to the N-acyclic *sub-program* of any N-cyclic GEDP.

Corollary 4.9. *Let* Π *be any GEDP, and* $S \subseteq Lit$. *Suppose that* Π_{NA} *is a subset of* Π *such that* $\Pi_{\mathrm{NA}} \cup \Pi'$ *is an N-acyclic GEDP for any N- acyclic set* $\Pi' \subseteq \Pi$. *Then, S is an answer set of* Π *iff S is an answer set of* $(\Pi \setminus \Pi_{\mathrm{NA}}) \cup shift(\Pi_{\mathrm{NA}})$.

Proof. The proof of Theorem 4.6 does not necessarily require that every rule in an N-acyclic GEDP be transformed by the shifting. When Π is N-cyclic, $\Pi \setminus \Pi_{\mathrm{NA}}$ is N-cyclic. Transforming only an N-acyclic part Π_{NA} by the shifting does not affect the answer sets of Π. □

Corollary 4.9 indicates that each set of N-acyclic rules that, if any other N-acyclic rules are added to the set, would never become N-cyclic can be reduced to rules without positive *not* by the shifting transformation. In other words, positive *not* is meaningful and cannot be precisely represented in any other way only if rules are N-cyclic.

Note that transformations of abducibles in abductive logic programming, disjunctions under the possible model semantics, and fixed predicates in circumscription presented in Section 3 all result in N-cyclic rules in GEDPs. We now see that N-acyclic GEDPs (or EDPs) cannot precisely express these semantics as they involve non-minimal answer sets.

The N-acyclic condition is only a sufficient condition for any equivalent transformation. An obvious necessary condition is that every answer set of a GEDP is minimal. However, this is not a sufficient condition. For example, consider the GEDP Π_6 consisting of rules

$$p \mid not\, p \leftarrow,$$

$$q \leftarrow p,$$

$$\leftarrow not\, q.$$

Π_6 has the unique (and hence minimal) answer set $\{p, q\}$. But Π_6 is an N-cyclic program, and $shift(\Pi_6)$ has no answer set. Thus, the converse of Lemma 4.5 does not hold in general. Namely, for an N-cyclic GEDP Π, a minimal answer set of Π is not necessarily an answer set of $shift(\Pi)$.

4.2. Integrity constraints

Using Corollary 4.9, we see that a special kind of rules can always be transformed by the shifting without regarding any other rule. A rule having no literals but positive *not* in its head is called an *integrity constraint*. This is because such a rule is never used to infer a literal directly. For instance.

$$not\, p \leftarrow q$$

is an integrity constraint meaning that if q is believed then p cannot be believed. This rule has exactly the same effect as the integrity constraint

$$\leftarrow p, q,$$

which denotes that both p and q cannot be believed at the same time. In general, every integrity constraint can be represented as a rule with an empty head using the shifting transformation.

Corollary 4.10. *Let* Π *be any GEDP, and* $S \subseteq Lit$. *Let IC be any set of integrity constraints in* Π. *Then,* S *is an answer set of* Π *iff* S *is an answer set of* $(\Pi \backslash IC) \cup shift(IC)$.

Proof. Since every rule in IC has no literal A_i in the form (9), it does not construct a relation of the form $l(A_i) \geqslant l(B_j)$ in the definition of N-acyclic GEDPs. Therefore, IC is an N-acyclic GEDP satisfying the condition for Π_{NA} in Corollary 4.9 □

It is often claimed that, in logic programs with both negation as failure and classical negation, the *coherence principle* [1] is important: for any atom A and any canonical model M, if $M \models \neg A$ then $M \models not\, A$, and if $M \models A$ then $M \models not\, \neg A$. This property can be naturally written using positive *not* as

$$not\, A \leftarrow \neg A, \tag{13}$$

$$not\, \neg A \leftarrow A. \tag{14}$$

Notice that these schemas for the coherence principle are converse to the rules for the *closed world assumption:*

$$\neg A \leftarrow not\, A \quad (\text{or} \quad A \leftarrow not\, \neg A).$$

In [13], rules (13) and (14) are used to compute the extended well- founded semantics for ELPs. The approach in [13] considers a paraconsistent semantics to capture the meaning of rules with positive *not* but without disjunctions. On the other hand, in the answer set semantics, the addition of these schemas for every atom A simply makes every answer set (if exists) consistent. Hence, as in ELPs ([1], Theorem 3.1), the coherency and the consistency coincide in GEDPs.

Example 4.11. The GEDP consisting of the two rules

$$p \,|\, not\, p \leftarrow,$$

$$\neg p \,|\, not\, \neg p \leftarrow$$

has four answer sets: $\emptyset$, $\{p\}$, $\{\neg p\}$ and Lit. If we introduce

$$not\, p \leftarrow \neg p,$$

then Lit is rejected.

Note the that each rule representing the coherence principle is not used to derive any literal. Thus, under the answer set semantics, the rules of the coherence principles are integrity constraints. By Corollary 4.10, schemas (13) and (14) can also be represented as

$$\leftarrow A, \neg A \quad \text{for any atom } A.$$

Hence, the coherence principle can be expressed without positive *not* under the answer set semantics.

5. Supported sets and non-minimal answer sets

One of the most important criteria that any model theoretic semantics should satisfy is the "supportedness". Apt et al. [3] defined *supported models* for NLPs, and Marek and Subrahmanian [45] have shown that every stable model is a supported model. Recently, the notion of supported models has been extended for disjunctive programs by Baral and Gelfond [4] and by Brass and Dix [9]. In this section, we first define the corresponding notion for GEDPs, which is then used to analyze cyclic GEDPs.

Let Π be any GEDP. A set of ground literals $S \subseteq Lit$ is a *supported set* of Π if: (i) S satisfies every ground rule from Π, and (ii) for any literal $L \in S$ there exists a ground rule

$$L_1 \,|\, \ldots \,|\, L_k \,|\, not\, L_{k+1} \,|\, \ldots \,|\, not\, L_l \leftarrow L_{l+1}, \ldots, L_m, not\, L_{m+1}, \ldots, not\, L_n$$

from Π such that

(a) $\{L_{l+1}, \ldots, L_m\} \subseteq S$,
(b) $\{L_{m+1}, \ldots, L_n\} \cap S = \emptyset$,
(c) $\{L_1, \ldots, L_k\} \cap S = \{L\}$, and
(d) $\{L_{k+1}, \ldots, L_l\} \subseteq S$.

This notion of supported sets reduces to that of [4] for EDPs, and to the notion of supported models of [9] for NDPs, and that of [3,45] for NLPs. In fact, conditions (a), (b), and (c) are exactly the same as the definition by [4]. The last condition (d), together with (c), implies that all the disjuncts other than L in the head of the ground rule are not satisfied by S.

Example 5.1. $\emptyset$ is the unique supported set of $\Pi_3 = \{p \mid not\, q \leftarrow\}$. Both $\emptyset$ and $\{p, q\}$ are supported sets of $\Pi_7 = \{p \mid not\, q \leftarrow, q \mid not\, p \leftarrow q\}$, but the latter is not an answer set of Π_7.

In the above example, all answer sets are supported sets. We now formally verify this relationship between supported sets and answer sets.

Lemma 5.2 ([4], Proposition 4.1). *Every consistent answer set of an EDP is a supported set.*

Theorem 5.3. *Every consistent answer set of a GEDP is a supported set.*

Proof. Let Π be a GEDP, and S its consistent answer set. By Proposition 2.1, S satisfies every ground rule from Π. By definition, S is a consistent answer set of Π^S. By Lemma 5.2, S is a supported set of Π^S. Namely, for any $L \in S$, there is a ground rule R of the form (2) from Π^S such that $\{L_{l+1}, \ldots, L_m\} \subseteq S$ and $\{L_1, \ldots, L_k\} \cap S = \{L\}$. By the construction of Π^S, for the rule R, there is a corresponding ground rule of the form (1) from Π such that $\{L_{m+1}, \ldots, L_n\} \cap S = \emptyset$ and $\{L_{k+1}, \ldots, L_l\} \subseteq S$. This completes the proof of the theorem. □

Theorem 5.3 is a generalization of Lemma 5.2, and hence that of Theorem 2 in [45]. Note that the converse of Theorem 5.3 does not hold in general (see Example 5.1). However, we will show in Lemma 5.6 that the converse holds for P-acyclic GEDPs.

5.1. Supported sets in shifting

The supported set semantics is preserved through the shifting transformation for any GEDP. Namely, *any GEDP collapses to an EDP under the supported set semantics.*

Theorem 5.4. *Let Π be any GEDP, and $S \subset Lit$. S is a consistent supported set of Π iff S is a consistent supported set of shift(Π).*

Proof. For a consistent set S of literals, S is a supported set of Π

iff S satisfies every ground rule from Π and for any $L \in S$ there is a ground rule of the form (9) such that $\{C_1, \ldots, C_m\} \subseteq S$, $\{D_1, \ldots, D_n\} \cap S = \emptyset$, $\{A_1, \ldots, A_k\} \cap S = \{L\}$, and $\{B_1, \ldots, B_l\} \subseteq S$

iff S satisfies every ground rule from $shift(\Pi)$ and for any $L \in S$ there is a ground rule of the form (10) such that $\{B_1, \ldots, B_l\} \subseteq S$. $\{C_1, \ldots, C_m\} \subseteq S$, $\{D_1, \ldots, D_n\} \cap S = \emptyset$, and $\{A_1, \ldots, A_k\} \cap S = \{L\}$

iff S is a supported set of $shift(\Pi)$. □

Next, we consider how the shifting transformation affects the relationship between supported sets and answer sets of GEDPs. We have seen in Section 4.1 that the answer set semantics is preserved by the shifting for N-acyclic GEDPs. For P-acyclic GEDPs, we will show that answer sets of programs are precisely supported sets of shifted programs.

Lemma 5.5 ([6], Theorem 2.3). *Let P be a (P-)acyclic EDP, and $S \subset Lit$. S is a consistent answer set of P iff S is a consistent supported set of P.*

Lemma 5.6. *Every consistent supported set of a P-acyclic GEDP Π is a consistent answer set of Π.*

Proof. Let S be a consistent supported set of Π. Then, (i) S satisfies every ground rule from Π and (ii) for any $L \in S$ there is a ground rule R of the form (9) such that $\{C_1, \ldots, C_m\} \subseteq S$, $\{D_1, \ldots, D_n\} \cap S = \emptyset$, $\{A_1, \ldots, A_k\} \cap S = \{L\}$, and $\{B_1, \ldots, B_l\} \subseteq S$. The fact (i) implies that S also satisfies every rule in Π^S. The fact (ii) implies that there is the corresponding rule R' of the form (12) in Π^S for each $L \in S$ such that $\{C_1, \ldots, C_m\} \subseteq S$ and $\{A_1, \ldots, A_k\} \cap S = \{L\}$. Thus, S is a supported set of Π^S. On the other hand, because Π is P-acyclic, Π^S is (P-)acyclic as well. By Lemma 5.5, S is an answer set of Π^S, and hence the result follows. □

Theorem 5.7. *Let Π be a P-acyclic GEDP, and $S \subset Lit$. S is a consistent answer set of Π iff S is a consistent supported set of $shift(\Pi)$.*

Proof. By Theorem 5.4, S is a consistent supported set of $shift(\Pi)$ iff S is a consistent supported set of Π. On the other hand, by Theorem 5.3 and Lemma 5.6, S is a consistent answer set of Π iff S is a consistent supported set of Π. Hence, the theorem holds. □

Theorem 5.7 is a generalization of Lemma 5.5. The consistency of S in this theorem cannot be omitted. For example, the GEDP Π_8 consisting of rules:

$$p \mid q \leftarrow,$$
$$\neg p \leftarrow,$$
$$\neg q \leftarrow$$

is acyclic and has the answer set Lit. However, Lit is not a supported set of $shift(\Pi_8) = \Pi_8$.

Example 5.8. For the GEDP Π_2 introduced in Example 3.13, $shift(\Pi_2)$ is exactly the same as the program Π_1:

$$Mary\text{-}at\text{-}Party \leftarrow John\text{-}at\text{-}Party,$$
$$John\text{-}at\text{-}Party \leftarrow Mary\text{-}at\text{-}Party,$$

whose supported sets are $\emptyset$ and $\{John\text{-}at\text{-}Party, Mary\text{-}at\text{-}Party\}$, which coincide with the answer sets of Π_2.

The next corollary summarizes the properties of acyclic GEDPs. Recall that a GEDP is acyclic iff it is both P-acyclic and N-acyclic.

Corollary 5.9. *Let Π be an acyclic GEDP, and $S \subset Lit$. The following four statements are equivalent.*

(a) *S is a consistent answer set of Π.*
(b) *S is a consistent supported set of Π.*
(c) *S is a consistent answer set of $shift(\Pi)$.*
(d) *S is a consistent supported set of $shift(\Pi)$.*

Proof. Follows from Theorems 4.6, 5.4, and 5.7. □

5.2. Characterizing supported sets by answer sets

As far as the authors know, there does not seem to exist a proof procedure for computing the supported model semantics for normal or extended (disjunctive) programs, although many researchers point out the importance of supported models. The difficulty seems to lie in the fact that there are non-minimal supported models of programs. In this section, we characterize the supported sets for any GEDP in terms of its answer sets. This implies that it provides a method to compute supported models proposed in the literature [3,45,4,9]. To this end, we utilize the *inverse shifting* defined as follows.

Let Π be any GEDP. The GEDP $invshift(\Pi)$ is obtained from Π by replacing every rule of the form (9)

$$A_1 | \ldots | A_k | not\, B_1 | \ldots | not\, B_l \leftarrow C_1, \ldots, C_m, not\, D_1, \ldots, not\, D_n$$

with the rule

$$A_1 | \ldots | A_k | not\, B_1 | \ldots | not\, B_l | not\, C_1 | \ldots | not\, C_m \leftarrow not\, D_1, \ldots, not\, D_n.$$

Observation 5.10. *For any GEDP Π, $invshift(\Pi)$ is P-acyclic.*

Theorem 5.11. *Let Π be any GEDP, and $S \subset Lit$. S is a consistent supported set of Π iff S is a consistent answer set of $invshift(\Pi)$.*

Proof. S is a consistent answer set of $invshift(\Pi)$
iff S is a consistent supported set of $shift(invshift(\Pi))$ (by Theorem 5.7 and Observation 5.10)
iff S is a consistent supported set of $shift(\Pi)$ (by $shift(invshift(\Pi)) = shift(\Pi)$)
iff S is a consistent supported set of Π (by Theorem 5.4). □

Two important results follow from Theorem 5.11. First, the supported set semantics for any GEDP can be completely characterized in terms of the answer set semantics. Secondly, computation of supported sets of a GEDP Π is realized by that of answer sets of the GEDP $invshift(\Pi)$, which is then reduced to that of answer sets of the EDP obtained by the *edp* translation that will be shown in Section 6.1. Then,

to get supported sets, we can use any proof procedure for computing answer sets of EDPs (see Section 6.3).

6. Complexity and computation

In this section, we consider the computational complexity of GEDPs and present an algorithm to compute the answer sets of a finite GEDP. These results indicate that positive *not* can be eliminated from programs so that we can use any proof procedure for computing EDPs or PDPs.

6.1. Simulation of positive not by EDPS

We first show a polynomial-time translation from a GEDP into an EDP. Let Π be any GEDP. The extended disjunctive program $edp(\Pi)$ is obtained from Π by replacing each rule with positive *not* in Π of the form

$$L_1 \mid \ldots \mid L_k \mid not\, L_{k+1} \mid \ldots \mid not\, L_l \leftarrow L_{l+1}, \ldots, L_m, not\, L_{m+1}, \ldots, not\, L_n \tag{15}$$

$(n \geqslant m \geqslant l > k \geqslant 0)$ with the rules without positive *not*

$$\lambda_1 \mid \ldots \mid \lambda_k \mid \lambda_{k+1} \mid \ldots \mid \lambda_l \leftarrow L_{l+1}, \ldots, L_m, not\, L_{m+1}, \ldots not\, L_n, \tag{16}$$

$$L_i \leftarrow \lambda_i \quad \text{for } i = 1, \ldots, k, \tag{17}$$

$$\lambda_i \leftarrow L_i, L_{k+1}, \ldots, L_l \quad \text{for } i = 1, \ldots, k, \tag{18}$$

$$\leftarrow \lambda_i, not\, L_j \quad \text{for } i = 1, \ldots, k \quad \text{and} \quad j = k+1, \ldots, l, \tag{19}$$

$$\leftarrow \lambda_j, L_j \quad \text{for } j = k+1, \ldots, l. \tag{20}$$

Here, λ_i is a new atom not appearing elsewhere in Π and is uniquely associated with each disjunct of a ground rule from Π. [6] Every rule without positive *not* in Π remains in $edp(\Pi)$ as it is. In the following, we denote by Lit_Π the set of all ground literals in the language of Π. Thus, Lit_Π includes no new atom λ_i.

Theorem 6.1. *Let Π be a GEDP, and $edp(\Pi)$ its translated EDP. A set S is an answer set of Π iff a set Σ is answer set of $edp(\Pi)$ such that $S = \Sigma \cap Lit_\Pi$.*

Proof. Let S be an answer set of Π. First, consider the reduct Π^S. If a rule

$$L_1 \mid \ldots \mid L_k \leftarrow L_{l+1}, \ldots, L_m \tag{21}$$

is in Π^S, then for the corresponding rule (15) in Π, it holds that $\{L_{k+1}, \ldots, L_l\} \subseteq S$ and $\{L_{m+1}, \ldots, L_n\} \cap S = \emptyset$. In this case, the reduct $edp(\Pi)^S$ includes the rule

$$\lambda_1 \mid \ldots \mid \lambda_k \mid \lambda_{k+1} \mid \ldots \mid \lambda_l \leftarrow L_{l+1}, \ldots, L_m \tag{22}$$

[6] If a rule contains n distinct free variables $\mathbf{x} = x_1, \ldots, x_n$, then a new atom $\lambda_i(\mathbf{x})$ is associated with each L_i, where λ_i in this case is an n-ary predicate symbol appearing nowhere in P.

and the rules (17), (18) and (20), but does not contain the rule (19). Since S satisfies each rule in Π^S), for each rule R of the form (21) such that $\{L_{l+1}, \ldots, L_m\} \subseteq S$, there exists $L_i \in S$ for some $1 \leqslant i \leqslant k$. Let

$$\Sigma 1 = \bigcup_{R \in \Pi^S} \{\lambda_i \mid L_i \in S, 1 \leqslant i \leqslant k\}.$$

Next, suppose that there is a rule (15) in Π such that $\{L_{m+1}, \ldots, L_n\} \cap S = \emptyset$ but $\exists L_j (k+1 \leqslant j \leqslant l)$ such that $L_j \notin S$. In this case, there is no corresponding rule (21) in Π^S, but the rule (22) is present in $edp(\Pi)^S$. $edp(\Pi^S)$ also contains the rules (17), (18) and (20) and the rules $\leftarrow \lambda_i$ for $i = 1, \ldots, k$ (from the rule (19)). Then, for each such rule R' (22) of $edp(\Pi)^S$, let

$$\Sigma 2 = \bigcup_{R' \in edp(\Pi)^S} \{\lambda_j \mid L_j \notin S, K+1 \leqslant j \leqslant l\}.$$

Now let $\Sigma = S \cup \Sigma 3$, where $\Sigma 3$ is a minimal subset of $\Sigma 1 \cup \Sigma 2$ such that each λ_i or λ_j is chosen in a way that Σ satisfies every rule of the form (22), (17), (18) and (20) and the reduct of (19) by S. Obviously, it holds that $S = \Sigma \cap lit_\Pi$. Because new literals λ_i's never appear within *not*, the program $edp(\Pi)^S$ is exactly the same as the program $edp(\Pi)^\Sigma$. Then, Σ satisfies all the rules of $edp(\Pi)^\Sigma$, and if $S = Lit_\Pi$ then $Lit_\Pi \subseteq \Sigma$.

To see that Σ is a minimal set satisfying the rules of $edp(\Pi)^\Sigma$, notice that S is a minimal set satisfying the rules of Π^S. From the construction of $\Sigma 3$, it is easy to see that Σ is a minimal set *containing* S and satisfying the rules of $edp(\Pi)^\Sigma$. We thus only need to verify that there is no Σ' such that: (i) $\Sigma' \subset \Sigma$, (ii) Σ' satisfies the rules of $edp(\Pi)^\Sigma$, and (iii) $S' \subset S$ for $S' = \Sigma' \cap Lit_\Pi$. Suppose to the contrary that such a Σ' exists. Then, the condition (iii) is satisfied only if there exist rules (17) and (18) such that $L_i \in S \setminus S'$ and $\lambda_i \in \Sigma \setminus \Sigma'$ for some $1 \leqslant i \leqslant k$. For this λ_i, there must be the rule (22) such that $\{L_{l+1}, \ldots, L_m\} \subseteq S'$. By the condition (ii), there is a literal $\lambda_j \in \Sigma'$ for some $k+1 \leqslant j \leqslant l$. This λ_j, however, is not included in $\Sigma 2$ by (20), contradicting the condition (i). Therefore, Σ is an answer set of $edp(\Pi)^\Sigma$, and hence an answer set of $edp(\Pi)$.

Conversely, let Σ be an answer set of $edp(\Pi)$, and $S = \Sigma \cap Lit_\Pi$. Since Σ is an answer set of $edp(\Pi)^\Sigma$, for each rule (22) in $edp(\Pi)^\Sigma$, if $\{L_{l+1}, \ldots, L_m\} \subseteq S$, then $\lambda_i \in \Sigma$ for some $1 \leqslant i \leqslant l$. There are two cases: (a) If $\lambda_i \in \Sigma$ for some $1 \leqslant i \leqslant k$, then $L_i \in S$ by (17) and hence $\{L_{k+1}, \ldots, L_l\} \subseteq S$ by (19). Then, the corresponding rule (21) exists in Π^S and S satisfies it. (b) If $\lambda_i \notin \Sigma$ but $\lambda_j \in \Sigma$ for some $1 \leqslant i \leqslant k$ and $k+1 \leqslant j \leqslant l$, then $L_j \notin S$ by (20). Then, there is no corresponding rule (21) in Π^S. In either case, S satisfies all rules of Π^S.

Suppose that there is a set S' of literals from Lit_Π such that (i) $S' \subset S$ and (ii) S' satisfies the rules of Π^S. Then, two conditions (i) and (ii) are satisfied only if there is a rule (21) such that $\{L_{l+1}, \ldots, L_m\} \subset S'$ and for some two literals L_{i1} and L_{i2} $(1 \leq i1, i2 \leqslant k, i1 \neq i2) L_{i1} \in S'$ but $L_{i2} \in S \setminus S'$. Without loss of generality, we can assume that just one such rule exists in Π^S. Since S and S' contain $L_{k+1}, \ldots, L_l$ in the corresponding rule (15) in Π, $\lambda_{i1}, \lambda_{i2} \in \Sigma$ by (18). Let $\Sigma' = \Sigma \setminus \{L_{i2}, \lambda_{i2}\}$. Then, Σ' satisfies all the rules (22), (17), (18) and (20) existing in $edp(\Pi)^\Sigma$. This contradicts the fact that Σ is an answer set of $edp(\Pi)^\Sigma$. Hence, S is an answer set of Π^S and therefore an answer set of Π. □

We thus see that any GEDP can be translated to an EDP by eliminating positive *not*. The fact that non-minimal answer sets of GEDPs can be expressed by answer

sets of EDPs that must be minimal is a somewhat surprising and unexpected result. The reason why this simulation is possible is that the newly introduced atoms λ_i's have the effect to distinguish each positive *not*, and each answer set of $edp(\Pi)$ becomes minimal by the existence of these new atoms.

Example 6.2. Suppose that the gedp Π_9 is given as

$$p \mid not\, q \leftarrow,$$
$$q \mid not\, p \leftarrow .$$

The answer sets of Π_9 are $\{\{p,q\},\emptyset\}$. Correspondingly, its translated program

$$edp(\Pi_9) = \{\lambda_1 \mid \lambda_2 \leftarrow, \quad \lambda_3 \mid \lambda_4 \leftarrow,$$
$$p \leftarrow \lambda_1, \quad \lambda_1 \leftarrow p,q, \quad \leftarrow \lambda_1, not\, q, \quad \leftarrow \lambda_2, q,$$
$$q \leftarrow \lambda_3, \quad \lambda_3 \leftarrow q,p, \quad \leftarrow \lambda_3, not\, p, \quad \leftarrow \lambda_4, p\}$$

has the answer sets $\{\{\lambda_1, \lambda_3, p, q\}, \{\lambda_2, \lambda_4\}\}$.

Remark 6.3. The *edp* translation maps GEDPs to EDPs so that there is a *one-to-one correspondence* between both answer sets. Although the *edp* translation can be applied to any GEDP, the resultant EDP cannot have exactly the same answer sets as those of the original GEDP (Observation 4.1). This means that we need an extra mechanism to recover an original answer set, that is, removing every new atom λ_i from an answer set of the created EDP. On the other hand, we have provided the shifting transformation of GEDPs to EDPs is Section 4.1. Although the shifting transformation is not complete for every GEDP, it preserves the equivalence of programs for N-acyclic GEDPs and does not need the additional task to remove λ_i's.

6.2. Complexity results

We are now ready to give the complexity results for GEDPs. Since the class of GEDPs includes the class of EDPs and we have shown a polynomial-time translation from a GEDP into an EDP, the next result follows immediately from the complexity results of EDPs given by Eiter and Gottlob [16].

Theorem 6.4. *Let* Π *be a finite propositional GEDP, and* L *a literal.*
(a) *Deciding the existence of an answer set of* Π *is* Σ_2^P*-complete.*
(b) *Deciding whether* L *is true in some answer set of* Π *is* Σ_2^P*-complete.*
(c) *Deciding whether* L *is true in all answer sets of* Π *is* Π_2^P*-complete.*

Theorem 6.4. demonstrates that allowing positive *not* does not increase the computational complexity of the answer set semantics. Eiter and Gottlob also show that the complexity results for EDPs apply to EDPs without classical negation $\neg$ as well. Therefore. GEDPs are in the same complexity class as NDPs. Furthermore, Theorem 6.4 (b) also applies to the minimal model semantics for PDPs. This observation leads us to a further translation in Section 6.3.

Ben-Eliyahu and Dechter [6] have shown the (co-) NP-completeness of a restricted class of EDPs. According to their notations, a *dependency graph* of a ground EDP P is a directed graph in which its nodes are literals in P and there is an edge from L to L'

iff there is a rule in P such that L appears in the body and L' appears in the head of the rule. An EDP is *head-cycle free* if its dependency graph contains no directed cycle that goes through two different literals in the head of the same disjunctive rule. Then, three problems in Theorem 6.4 for propositional head-cycle free EDPs are reducible to the problem of satisfiability or provability of propositional formulas in polynomial-time [6]. Here, we show such a reduction of complexity results is also possible for a restricted class of GEDPs, by generalizing their results.

The dependency graph of a GEDP is defined in the same way as that of an EDP except that an additional edge is considered for positive *not*. Given GEDP Π, its *dependency graph* G_Π is a directed graph in which its nodes are ground literals from Π and there is an edge from L to L' iff there is a ground rule R from Π such that either:

(i) L appears in the body and L' appears in the head of R; or

(ii) both *not* L and L' appears in the head of R.

Thus, while each *not* L in bodies is ignored, each *not* L in heads constructs an edge in G_Π (recall that we also ignored the level of every *not* L in bodies in the definition of *acyclic* GEDPs in Section 4.1). A GEDP Π is *head-cycle free* if G_Π contains no directed cycle that goes through two literals $L_{i1}, L_{i2}\,(1 \leqslant i1, i2 \leqslant k, L_{i1} \neq L_{i2})$ in any ground rule of the form (15) from Π. The class of head-cycle free GEDPs obviously includes the class of head-cycle free EDPs and the class of ELPs. Also, the class of head-cycle free GEDPs includes the class of GEDPs each of whose rule permits in the head at most one L' but any number of *not* L's:

$$L_1 \,|\, not\, L_2 \,|\, \ldots \,|\, not\, L_l \leftarrow L_{l+1}, \ldots, L_m, not\, L_{m+1}, \ldots, not\, L_n.$$

The class of head-cycle free GEDPs further includes the class of acyclic GEDPs:

Observation 6.5. *Every acyclic GEDP is head-cycle-free. Moreover, the dependency graph of an acyclic GEDP has no directed cycle.*

The converse of the above observation does not necessarily hold when the dependency graph has an infinite decreasing chain. For example, the program Π_5 in Example 4.4 (Section 4.1) has no directed cycle, but is not acyclic. Now, we show that the head-cycle free property is preserved by the *edp* translation.

Lemma 6.6. *Let Π be a GEDP. Π is head-cycle free iff edp(Π) is head-cycle free.*

Proof. An edge from L_j to L_i for $j = k+1, \ldots, l$ and $i = 1, \ldots, k$ in the same rule (15) is in G_Π iff a path from L_j to L_i through rules (18) and (17) is in $G_{edp(\Pi)}$. Then, each directed path from L to L' in G_Π is contained in $G_{edp(\Pi)}$, and vice versa. Hence, any two literals L_{i1}, L_{i2} $(1 \leqslant i1, i2 \leqslant k)$ in the same rule (15) are contained in a cycle in G_Π iff the literals $\lambda_{i1}, \lambda_{i2}$ in the corresponding rule (16) are contained in a cycle in $G_{edp(\Pi)}$. □

The next result follows from Theorem 6.1, Lemma 6.6 and complexity results of head-cycle free EDPs by [6]. It says that the computational complexity for the answer set semantics of head-cycle free GEDPs lies at the first level of the polynomial hierarchy, which is exactly the same level as the of head-cycle free EDPs or NLPs.

Theorem 6.7. *Let Π be a finite propositional head-cycle free GEDP, and L a literal.*

(a) *Deciding the existence of an answer set of Π is NP-complete.*

(b) *Deciding whether L is true in some answer set of* Π *is NP-complete.*
(c) *Deciding whether L is true in all answer sets of* Π *is co-NP-complete.*

Note that the class of head-cycle free GEDPs includes, as a special case, the class of programs $P \cup abd(\Gamma)$ obtained from abductive programs $\langle P, \Gamma \rangle$ where both P and Γ are ELPs (see Section 3.2). This fact and results in Section 3 imply that computational problems for *abductive NLPs* [31], *knowledge systems* [26], and *the possible model semantics for NDPs* [55] have all the same complexity results as in Theorem 6.7. These results are also stated in [56] based on translations of such programs into NLPs.

6.3. Computing answer sets of arbitrary GEDP

To compute the answer set semantics for any GEDP Π, we can apply any proof procedure for EDPs to the EDP $edp(\Pi)$ obtained in Section 6.1. To this end, a bottom-up proof procedure for EDPs has been proposed by Inoue et al. [27] to compute answer sets of EDPs using model generation techniques. Here, we present an essence of the method of [27]. First, each EDP P is converted into its *positive form* P^+, which is obtained from P by replacing each negative literal $\neg L$ with a new atom $-L$. Note that P^+ is an NDP. We also denote the positive form of a set S of literals as S^+. Next, P^+ is translated into the set $fo(P^+)$ of first-order formulas by completely eliminating *not* as follows. For each rule in P^+ of the form

$$L_1 | \ldots | L_k \leftarrow L_{k+1}, \ldots, L_m, not\, L_{m+1}, \ldots, not\, L_n, \tag{23}$$

where each L_i is an atom, $fo(P^+)$ contains the formula

$$L_{k+1} \wedge \ldots \wedge L_m \supset H_1 \vee \ldots \vee H_k \vee \mathsf{K}L_{m+1} \vee \ldots \vee \mathsf{K}L_n, \tag{24}$$

where $H_i \equiv L_i \wedge -\mathsf{K}L_{m+1} \wedge \ldots \wedge -\mathsf{K}L_n (i = 1, \ldots, k)$ and $fo(P^+)$ contains the formulas

$$\neg(L \wedge -\mathsf{K}L) \quad \text{for each } L \in Lit_P^+, \tag{25}$$

$$\neg(L \wedge -L) \quad \text{for each pair } L, -L \in Lit_P^+. \tag{26}$$

Here, $\mathsf{K}L$ is a new atom which denotes that L *should be true*, and $-\mathsf{K}L$ is the positive form of $\neg\mathsf{K}L$. Now, let I be an Herbrand interpretation of $fo(P^+)$, i.e., a set of ground atoms in the language of $fo(P^+)$. The, we say that I *satisfies the stability condition* if it holds that

$$\mathsf{K}L \in I \text{ implies } L \in I \quad \text{for every atom } L \in Lit_P^+. \tag{27}$$

Lemma 6.8. [27]. *Let P be an EDP, and* $S \subset Lit_P$. *S is a consistent answer set of P iff M is a minimal Herbrand model of* $fo(P^+)$ *such that* $S^+ = M \cap Lit_P^+$ *and that M satisfies the stability condition.*

The next theorem completely characterizes the consistent answer sets of a GEDP in terms of the above first-order translation. [7]

[7] Although Theorem 6.9 does not cover the contradictory answer set of Π, the methods used in [27] can be applied to identify the answer set Lit_Π.

Theorem 6.9. *Let* Π *be any GEDP, and* $S \subset Lit_\Pi$. *S is a consistent answer set of* Π *if M is a minimal Herbrand model of* $fo(edp(\Pi)^+)$ *such that* $S^+ = M \cap Lit_\Pi^+$ *and that M satisfies the stability condition.*

Proof. The result follows from Theorem 6.1 and Lemma 6.8. □

It is well known that for PDPs minimal Herbrand models coincide with answer sets. Then, the formula (24) can be identified with the rules

$$H_1 | \ldots | H_k | \mathsf{K}L_{m+1} | \ldots | \mathsf{K}L_n \leftarrow L_{k-1}, \ldots, L_m,$$
$$H_i \leftarrow L_i, -\mathsf{K}L_{m+1}, \ldots, -\mathsf{K}L_n \quad (i = 1, \ldots, k),$$
$$L_i \leftarrow H_i \quad (i = 1, \ldots, k),$$
$$-\mathsf{K}L_j \leftarrow H_i \quad (i = 1, \ldots, k; j = m+1, \ldots, n)$$

and schemas (25) and (26) can be written as

$$\leftarrow -\mathsf{K}L, L \quad \text{for each } L \in Lit_P^+,$$
$$\leftarrow -L, L \quad \text{for each pair } L, -L \in Lit_P^+.$$

Hence, the set $fo(P^+)$ can also be viewed as a PDP. We thus now have a polynomial-time translation from GEDPs into PDPs. Hence, to obtain answer sets of GEDPs, any procedure to compute minimal Herbrand models of PDPs can be applied as well. There are several techniques for this computation such as [5,27,18]. In particular, our translation is suitable for applying a bottom-up model generation procedure to compute answer sets of function-free and range-restricted GEDPs. Since we have characterized abductive programs as well as other commonsense reasoning in GEDPs in Section 3, they can also be computed by model generation procedures. Inoue et al. [28] have developed such a parallel abductive procedure, and Inoue and Sakama [30] have given a fixpoint semantics that accounts for the correctness of such bottom-up procedures using a similar translation.

Example 6.10. The abductive program $\langle P_2, \Gamma_1 \rangle$ given in Examples 3.1 and 3.4 is now translated into $fo(P_2 \cup abd(\Gamma_1))$ that consists of the propositional formulas

$$r \wedge b \supset (p \wedge -\mathsf{K}q) \vee \mathsf{K}q, \quad a \supset q, \quad r, \quad \mathsf{K}p,$$
$$\lambda_1 \vee \lambda_2, \quad \lambda_1 \equiv a, \quad \lambda_1 \supset \mathsf{K}a, \quad \neg(\lambda_2 \wedge a),$$
$$\lambda_3 \vee \lambda_4, \quad \lambda_3 \equiv b, \quad \lambda_3 \supset \mathsf{K}b, \quad \neg(\lambda_4 \wedge b),$$

and schema (25). [8] There are five minimal Herbrand models of $fo(P_2 \cup abd(\Gamma_1))$:

$$M_1 = \{r, \mathsf{K}p, \lambda_1, a, \mathsf{K}a, q, \lambda_3, b, \mathsf{K}b, \mathsf{K}q\},$$
$$M_2 = \{r, \mathsf{K}p, \lambda_1, a, \mathsf{K}a, q, \lambda_4\},$$
$$M_3 = \{r, \mathsf{K}p, \lambda_2, \lambda_3, b, \mathsf{K}b, p, -\mathsf{K}q\},$$
$$M_4 = \{r, \mathsf{K}p, \lambda_2, \lambda_3, b, \mathsf{K}b, \mathsf{K}q\},$$
$$M_5 = \{r, \mathsf{K}p, \lambda_2, \lambda_4\}.$$

[8] When an EDP P is an NDP, $P^+ = P$ holds and $fo(P)$ need not include schema (26).

Among these, only M_3 satisfies the stability condition, and corresponds to the Γ_1-minimal belief model $\{r, p, b\}$ of $\langle P_2, \Gamma_1 \rangle$.

7. Relation to non-monotonic formalisms

Recent research on the semantics of logic programming and non-monotonic reasoning has demonstrated that both fields have influenced each other. In this section, we establish the relationship between GEDPs and existing non-monotonic formalisms. In particular, there is a close relationship between GEDPs and autoepistemic logic.

Recall that the class of GEDPs is the "logic programming" fragment of propositional MBNF [38]. The embedding of the rule (1)

$$L_1 | \ldots | L_k | not\, L_{k+1} | \ldots | not\, L_l \leftarrow L_{l+1}, \ldots, L_m, not\, L_{m+1}, \ldots, not\, L_n$$

in MBNF is given by Lifschitz and Woo [41] as the formula

$$\mathrm{B}L_{l+1} \wedge \ldots \wedge \mathrm{B}L_m \wedge not\, L_{m+1} \wedge \ldots \wedge not\, L_n \supset \mathrm{B}L_1 \vee \ldots \vee \mathrm{B}L_k \vee not\, L_{k+1} \vee \ldots \vee not\, L_l.$$

Besides MBNF, there are many non-monotonic formalisms in which EDPs can be embedded. Gelfond et al. [23] use their disjunctive default logic, and Sakama and Inoue [54] show translations into default logic [52], autoepistemic logic [49] and circumscription [47]. Since we have presented the translation of GEDPs into EDPs, these previous results can be directly applied to embed GEDPs in such non-monotonic formalisms via the *edp* translation.

Although these results are all correct, the translation of GEDPs into EDPs introduces new literals like λ_i's. One often wants to see a stronger result such that the logical closure of an answer set is exactly the same as an *extension* of a non-monotonic formalism and that the set of literals true in the extension is exactly the answer set. In such an extension, the introduction of new literals should be avoided. Then, those formalisms that obey the principle of minimality such as (disjunctive) default logic and circumscription are rejected for this purpose. With this regard, the remaining candidate is autoepistemic logic. Lifschitz and Schwarz ([39], Corollary 3.1) and Chen ([12], Theorem 6) have independently provided the correct embedding of EDPs in autoepistemic logic. Moreover, both results are proved in a way applicable to a more general class of programs including consistent PL- theories of [41]. Here, we can take advantage of their proofs. [9]

Recall that a formula in autoepistemic logic is called *objective* if it does not contain the modal operator B; otherwise it is *subjective*. An *autoepistemic theory* is a set of formulas in autoepistemic logic. An autoepistemic theory is *stable* if it is closed under the logical and introspective consequences. Namely, a stable set T satisfies the conditions: (i) $T = cons(T)$, where $cons(T)$ denotes the set of logical consequences (in the sense of classical first-order logic) of T; (ii) if $\varphi \in T$ then $B\varphi \in T$, and (iii) if $\varphi \notin T$ then $\neg B\varphi \in T$. The meaning of each autoepistemic theory is usually character-

[9] Marek and Truszczyński [46] also show a different translation of EDPs into *reflexice autoepistemic logic* [60]. Lifschitz and Schwarz [39] further prove that reflexive autoepistemic logic can be used for the embedding of consistent PL-theories.

ized by the following stable set that is expanded from the theory: Given an autoepistemic theory K, a set T is a *stable expansion* of K iff it satisfies that

$$T = cons(K \cup \{B\varphi \mid \varphi \in T\} \cup \{\neg B\varphi \mid \varphi \notin T\}).$$

It is well known that for each set F of objective formulas, there is a unique stable set $E(F)$ containing F such that the objective formulas in $E(F)$ are exactly the same as those in $cons(F)$. Moreover, if a theory K contains only objective formulas, then $E(K)$ is a unique stable expansion of K [49].

Given a GEDP Π, its *autoepistemic translation* $ae(\Pi)$ is defined as follows: Each rule of the form (1) in Π is translated into the following formula in $ae(\Pi)$:

$$(BL_{l+1} \wedge L_{l+1}) \wedge \cdots \wedge (BL_m \wedge L_m) \wedge \neg BL_{m+1} \wedge \cdots \wedge \neg BL_n \supset (BL_1 \wedge L_1) \vee \cdots \vee (BL_k \wedge L_k) \vee \neg BL_{k+1} \vee \cdots \vee \neg BL_l. \tag{28}$$

Theorem 7.1. *Let Π be consistent GEDP, and S a set of literals. S is an answer set of Π iff $E(S)$ is a stable expansion of $ae(\Pi)$.*

Proof. The result follows from the Main Theorem in [39]. □

The autoepistemic translation $ae(\Pi)$ can be simplified for some class of GEDPs. When Π is GEDP consisting of rules of the form

$$A_1 \mid not\, A_2 \mid \ldots \mid not\, A_l \leftarrow A_{l+1}, \ldots, A_m, not\, A_{m+1}, \ldots, not\, A_n, \tag{29}$$

where $0 \leqslant l \leqslant m \leqslant n$ (A_1 may be empty) and each A_i is an atom, each conjuction $(BA_i \wedge A_i)$ for $i = 1, l+1, \ldots, m$ in $ae(\Pi)$ can be replaced simply with A_i as

$$A_{l+1} \wedge \cdots \wedge A_m \wedge \neg BA_{m+1} \wedge \cdots \wedge \neg BA_n \supset A_1 \vee \neg BA_2 \vee \cdots \vee \neg BA_l. \tag{30}$$

Note that this class of GEDPs is a subset of the class of head-cycle free GEDPs, and includes the class of NLPs [10] and programs $P \cup abd(\Gamma)$ that are translated from abductive NLPs $\langle P, \Gamma \rangle$. Let us denote as $ae_n(\Pi)$ the set of autoepistemic formulas obtained from a GEDP Π by replacing each rule of the form (29) with (30). An essential difference between $ae(\Pi)$ and $ae_n(\Pi)$ for a set Π of rules of the form (29) is that, while ae_n may map two different programs with the same answer sets into two autoepistemic theories with different stable expansions, the stable expansions of $ae(\Pi)$ are uniquely determined by the answer by sets of Π [39]. For example, both $\Pi_3 = \{p \mid not\, q \leftarrow\}$ and $\Pi_{10} = \{p \leftarrow q\}$ have the same unique answer set $\emptyset$, but $ae_n(\Pi_3) = \{Bq \supset p\}$ has the stable expansion $E(\emptyset)$, while $ae_n(\Pi_{10}) = \{q \supset p\}$ has the stable expansion $E(\{q \supset p\})$. On the other hand, both $ae(\Pi_3) = \{Bq \supset (p \wedge Bp)\}$ and $ae(\Pi)_{10} = \{(q \wedge Bq) \supset (p \wedge Bp)\}$ have the same unique stable expansion $E(\emptyset)$. Nevertheless, we have the following one-to-one correspondence between the answer sets of Π and the stable expansions of $ae_n(\Pi)$.

[10] An autoepistemic translation of NLPs, which maps each rule without positive *not* into (30), was firstly introduced by Gelfond [19].

Corollary 7.2. *Let* Π *be a consistent GEDP such that* Π *is a set of rules of the form* (29), *and* S *a set of atoms.* S *is an answer set of* Π *iff* S *is the set of objective atoms true in a stable expansion of* $ae_n(\Pi)$

Proof. Suppose that S is an answer set of Π. By Theorem 7.1, there is a stable expansion E of $ae(\Pi)$ such that $S = E \cap At$ where At is the set of atoms occurring in Π and that

$$E = cons(S \cup \{B\varphi \mid \varphi \in E\} \cup \{\neg B\varphi \mid \varphi \notin E\}) \ .$$

The set $\{B\varphi \mid \in E\}$ includes BA for each $A \in S$. In the presence of these subjective atoms, all the objective atoms S in E also follows from some stable expansion E' of $ae_n(\Pi)$, and vice versa. Hence, $S = E' \cap At$. The converse direction can also be shown in the same manner. □

The above corollary can also be applied to the embedding of the possible model semantics for a NDP P since each rule in the translated GEDP $pm(P)$ is in the form (29).

Corollary 7.3. *Let* P *be a consider NDP that consists of rules of the form*

$$A_1 \mid \ldots \mid A_k \leftarrow A_{k+1}, \ldots, A_m, not\ A_{m+1}, \ldots not A_n.$$

A set S *of atoms is a possible model of* P *iff* S *is the set of objective atoms true in a stable expansion of the set of formulas obtained by translating each above rule in* P *into the formula*

$$A_{k+1} \wedge \cdots \wedge A_m \wedge \neg BA_{m+1} \wedge \cdots \wedge \neg BA_n \supset$$
$$(A_1 \vee \neg BA_1) \wedge \cdots \wedge (A_k \vee \neg BA_k) \wedge (BA_1 \vee \cdots \vee BA_k).$$

Proof. The translated formula is equivalent to the conjunction of the ae_n translation of rules (7) and (8) in $pm(P)$. Then, the corollary follows from Theorem 3.9 and Corollary 7.2. □

Now, let us look again at the embedding of abduction in GEDPs given in Theorem 3.2. The rule (3)

$$\gamma \mid not\ \gamma \leftarrow$$

is translated into

$$(B\gamma \wedge \gamma) \vee \neg B\gamma$$

by the autoepistemic translation, which is then equivalent to

$$B\gamma \supset \gamma. \tag{31}$$

The set consisting of formula (31) produces two stable expansions, one containing γ and $B\gamma$, the other containing $\neg B\gamma$ but neither γ nor $\neg\gamma$. Historically, the first expansion has been regarded as anomalous since the belief of γ is based solely on the assumption that γ is believed with no other support [33]. However, this situation is naturally interpreted in abduction. The fact that the formula (31) is the archetype to generate hypotheses strongly justifies the correctness of our use of positive *not* in the corresponding rule (3).

Finally, the relationship between the supported set semantics of GEDPs and autoepistemic logic can also be equipped in order to highlight the underlying nonmonotonicity of the formalism. Marek and Subrahmanian [45] have proposed such a connection for NLPs. In our case, it is enough to apply the autoepistemic translation *ael* to the supported set semantics via the inverse shifting of Theorem 5.11. Namely, given a GEDP Π, its supported sets can be chracterized by the stable expansions of the inversely shifted GEDP $ael(invshift(\Pi))$.

In summary, we have utilized the "non-minimal" nature of autoepistemic logic to express abduction and inclusive disjunctions in knowledge representation. The introspective nature of autoepistemic logic enable us to believe a certain proposition (say P) either from the lack of belief in other proportions (through $\neg BQ \supset P$ for example) or from no additional precondition (through $BP \supset P$ for example). These properties can completely describe the meanings of negative and positive occurrences of negation as failure in logic programming.

8. Discussion

1. Brewka and Konolige [10] give another semantics for GEDPs which is different from the answer set semantics in this paper. They allow positive *not* in a program but still obey the principle of minimality. Consequently, their semantics can never represent non-minimal canonical models and its relationship to autoepistemic logic must be different from ours. In this respect, they suggest the use of *moderately grounded expansions* [33] for the embedding. However, the following example demonstrates that moderately ground expansions are of no use to characterize the *minimal* answer sets of GEDPs. Instead, *parsimonious stable expansions* [15] appropriately characterize the minimal answer sets.

Recall that a stable expansion of an autoepistemic theory K is *moderately grounded* if its objective part is not larger than the objective part of any other stable set that includes K. A stable expansion of K is *parsimonious* if its objective part is not larger than the objective part of any other stable expansion of K. Note that each moderately grounded expansion is parsimonious but the converse does not necessarily hold.

Example 8.1. Consider the GEDP Π_6 given the Section 4.1:

$$p \,|\, not\, p \leftarrow,$$

$$q \leftarrow p,$$

$$\leftarrow not\, q.$$

Π_6 has the unique (and hence minimal) answer set $\{p, q\}$. The autoepistemic translation of Π_6 is

$$ae_n(\Pi_6) = \{Bp \supset p,\ \ p \supset q,\ \ Bq\}.$$

This autoepistemic theory has no moderately grounded expansion. In fact, $E(\{p, q\})$ is not a minimal stable set that included $ae_n(\Pi_6)$ since $E(\{q\})$ is a stable set containing $ae_n(\Pi_6)$ and its objective part is smaller than that of $E(\{p, q\})$. On the other hand, $E(\{p, q\})$ is the unique parsimonious stable expansion.

Theorem 8.2. *Let* Π *be a consistent GEDP, and* S *a set of literals.* S *is a minimal answer set of* Π iff $E(S)$ *is a parsimonious stable expansion of* $ae(\Pi)$.

Proof. The result follows from Theorem 7.1 and the definition of parsimonious stable expansions. □

Recall that our answer set semantics for GEDPs is characterized by stable expansions of the translated autoepistemic theories. From the complexity viewpoint, Eiter and Gottlob have shown that deciding whether an objective formula belongs to some parsimoniously grounded expansion of an autoepistemic theory is Σ_3^P-complete in general [15], while the same problem for some stable expansion is Σ_2^P-complete [25]. From this observation, it is conjectured that computing with a minimal answer set of GEDP is harder than computing with any answer set unless the polynomial hierarchy collapses.

2. An interesting property of the rule (3) $\gamma \,|\, not\ \gamma \leftarrow$ is that it is *valid* in the sense that every answer set satisfies it, that is, γ is either contained or not contained in it. In autoepistemic logic, the corresponding formula (31) is always contained in any stable expansion. However, the modal axiom schema of the same form

$$T\colon\ B\varphi \supset \varphi$$

cannot be put into the premise set without changing its stable expansions [49]. Similarly, adding the rule $L \,|\, not\, L \leftarrow$ to a program allows the literal L to be sanctioned that otherwise would not be, but this may cause literals that are entailed by the program to decrease since the number of answer sets increases. For example, q is entailed by the program containing one rule

$$q \leftarrow not\ p,$$

but once $p \,|\, not\, p \leftarrow$ is adopted q is no longer entailed. Another example of this property can be seen in the fact that a declaration of *fixed predicates* prevents over-minimization and undesired side effects in circumscription (Section 3.4). Sometimes such an addition of valid rules may make an incoherent program get an answer set.

For example,

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

$$\neg q \leftarrow$$

has no answer set, but with the rule $p \,|\, not\ p \leftarrow$ it obtains the answer set $\{\neg q, p\}$. The rule $L \,|\, not\, L \leftarrow$ in GEDPs and the schema T in autoepistemic logic can thus be applied to various domains other than abduction such as *contradiction resolution* and *reflection* in the sense of Konolige's analysis of *meta reasoning* [34].

3. Gelfond gives another cautious semantics for the closed world assumption in order to treat Example 3.10 properly by introducing the concept of *strong introspection* [20]. However, unlike Theorem 3.9 for our possible model semantics, this concept cannot be embedded in MBNF [38].

On the other hand, Eiter et al. propose a new non-monotonic formalism called *curbing* [17] which interprets disjunctions inclusively. Since their *good models* are nor necessarily minimal models, it is interesting to see whether MBNF can express curbing nor not. In the context of PDPs, it turns out that there is a close relationship between good models and possible models.

4. Lifschitz and Turner [40] show that a special form of positive *not* can be replaced with classical negation. As we have seen in Section 3.1, abducible literals L and $\neg L$ can be encoded as rules

$$L \mid not\, L \leftarrow,$$
$$\neg L \mid not\, \neg L \leftarrow .$$

In their setting (on the domain of reasoning about action), under the existence of the *completeness rule* for L:

$$\leftarrow not\, L, not \neg L,$$

the above two abducible rules can be represented by the rule

$$L \mid \neg L \leftarrow .$$

While this kind of replacement is sometimes possible, positive *not* is generally quite different from classical negation in heads. For example, as noted in Section 4, $\Pi_3 = \{p \mid not\, q \leftarrow\}$ has the unique answer $\emptyset$, but $\{p \mid \neg q \leftarrow\}$ has two answer sets $\{p\}$ and $\{\neg q\}$.

5. Marek et al. [43] characterize the supported models of NLPs by means of their framework of *constraint programming*. Roughly speaking, a rule with the constraint of the form

$$A \leftarrow B_1, \ldots, B_m : \phi_1 \wedge \cdots \wedge \phi_n$$

is read as "A if $B_1, \ldots B_m$ under the condition that $\phi_1 \wedge \cdots \wedge \phi_n$ holds". Thus, the body of this rule contains two kinds of conditions: those to be evaluated minimally (B_i's) as usual and those to be expected its supportedness (ϕ_j's). In other words, constraint programming offers the spectrum of the answer set/supported set semantics for NLPs. This rule corresponds to a rule with positive *not* of the form

$$A \mid not\, \phi_1 \mid \ldots \mid not\, \phi_n \leftarrow B_1, \ldots, B_m.$$

Hence, their framework is also characterized using positive *not*.

6. It is worth nothing that there is a GEDP whose dependency graph contains no directed cycle but it has non-minimal answer sets. For example, the program Π_{11}:

$$p(x) \mid not\, p(s(x)) \leftarrow,$$
$$q(0) \leftarrow$$

has answer sets $\{q(0)\}$ and $\{q(0), p(0), p(s(0)), p(s^2(0)), \ldots\}$. Since Π_{11} is not N-acyclic, Theorem 4.6 cannot be applied to this program. In fact,

$$shift(\Pi_{11}) = \{p(x) \leftarrow p(s(x)),\ \ q(0) \leftarrow\}.$$

which is the same as program Π_5 in Example 4.4 (Section 4.1), has the unique answer set $\{q(0)\}$. However, since Π_{11} is P-acyclic, the infinite answer set of Π_{11} is also a supported set of Π_5 by Theorem 5.7. This example indicates that the theory of positive *not* or supported sets may have interesting applications to *perpetual processes* in the line of [42], Ch. 6.

9. Conclusion

This paper has provided a number of new results in the class of general extended disjunctive programs (GEDPs), i.e., disjunctive programs which permit negation as

failure and classical negation both positively and negatively. The class of GEDPs is a natural extension of previously proposed logic programs. In particular, we have shown in this paper:

- embedding of abductive programs, the possible model semantics for NDPs, fixed predicates in circumscription in the answer set semantics for GEDPs,
- a syntactic condition under which a GEDP collapses to an EDP using the shifting transformation,
- characterization of the supported set semantics for GEDPs in terms of the answer set semantics for GEDPs,
- the computational complexity of GEDPs based on a polynomial-time translation of GEDPs into EDPs,
- an algorithm to compute answer sets of GEDPs based on a translation of GEDPs into PDPs, and
- the relationship between GEDPs and autoepistemic logic.

In conclusion, negation as failure in the head opens new possibilities of logic programming for representing commonsense knowledge. The most interesting property of GEDPs is that they may have non-minimal answer sets. It is due to the N-cyclic property that abductive programs, inclusive disjunctions and fixed predicates can be encoded as GEDPs with positive *not*. Incidentally, supported sets have a similar non-minimal property, and we have actually established the relationship between positive *not* and supported sets. Moreover, from the computational viewpoint, it has been shown that positive *not* does not introduce an extra complexity source. Therefore, computation of answer sets of GEDPs is realized using any proof procedure for computing answer sets of EDPs. With these results, we conclude that the concept of negation as failure in the head is a useful tool for representing knowledge in various domains in which the principle of minimality is too strong.

Acknowledgements

Discussions with Alexander Bochman, Thomas Eiter, George Gottlob, Michael Gelfond, and Vladimir Lifschitz were helpful to make many ideas in this paper clear. We also thank the anonymous referees for their comments.

References

[1] J.J. Alferes, L.M. Pereira, On logic program semantics and two kinds of negation, in: K. Apt (Ed.), Logic Programming: Proceedings of the Joint International Conference and Symposium, MIT Press, Cambridge, MA, 1992, pp. 574–588.

[2] K.R. Apt, M. Bezem, Acyclic programs, New Generation Computing 9 (3/4) (1991) 355–363.

[3] K.R. Apt, H.A. Blair, A. Walker, Towards a theory of declarative knowledge, in: J. Minker (Ed.), Foundations of Deductive Databases and Logic Programming, Morgan Kaufmann, Los Altos, 1988, pp. 89–148.

[4] C. Baral, M. Gelfond, Logic programming and knowledge representation, Journal of Logic Programming 19/20 (1994) 73–148.

[5] C. Bell, A. Nerode, R.T. Ng, V.S. Subrahmanian, Implementing deductive databases by linear programming, in: Proceedings of the Eleventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, ACM, New York, 1992, pp. 283–292.

[6] R. Ben-Eliyahu, R. Dechter, Propositional semantics for disjunctive logic programs, in: K. Apt (Ed.), Logic Programming: Proceedings of the Joint International Conference and Symposium, MIT Press, Cambridge, CA, 1992, pp. 813–827.
[7] A. Beringer, T. Schaub, Minimal belief and negation as failure: A feasible approach, in: Proceedings of AAAI–93, AAAI/MIT press, Cambridge, MA, 1993, pp. 400–405.
[8] A. Bochman, On bimodal nonmonotonic logics and their unimodal and nonmodal equivalents, in: Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, Morgan Kaufmann, Los Altos, CA, 1995, pp. 1518–1524.
[9] S. Brass, J. Dix, Characterizations of the stable semantics by partial evaluation, in: [44], pp. 85–98.
[10] B. Brewka, K. Konolige, An abductive framework for general logic programs and other nonmonotonic systems, in: Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence, Morgan Kaufmann, Los Altos, CA, 1993, pp. 9–15.
[11] E.P.F. Chan, A possible world semantics for disjunctive databases, IEEE Transactions on Data and Knowledge Engineering 5 (2) (1993) 282–292.
[12] J. Chen, Minimal Knowledge + Negation as Failure = Only Knowing (Sometimes), in: [50], pp. 132–150.
[13] C.V. Damásio, L.M. Pereira, Default negated conclusions: Why not?, in: R. Dyckhoff, H. Herre, P. Schroeder-Heister (Eds.), Extensions of Logic Programming: Proceedings of the Fifth International Workshop, Lecture Notes in Artificial Intelligence 1050, Springer, Berlin, 1996, pp. 103–117.
[14] P.M. Dung, Acyclic disjunctive logic programs with abductive procedure as proof procedure, in: Proceedings of the International Conference on Fifth Generation Computer Systems 1992, Ohmsha, 1992, pp. 555–561.
[15] T. Eiter, G. Gottlob, Reasoning with parsimonious and moderately grounded expansions, Fundamenta Informaticae 17 (1992) 31–53.
[16] T. Eiter, G. Gottlob, Complexity results for disjunctive logic programming and application to nonmonotonic logics, in: D. Miller (Ed.), Logic Programming: Proceedings of the 1993 International Symposium, MIT Press, Cambridge, MA, 1993, pp. 266–278.
[17] T. Eiter, G. Gottlob, Y. Gurevich, Curb your theory!–A circumscriptive approach for inclusive interpretation of disjunctive information, in: Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence, Morgan Kaufmann, Los Altos CA, 1993, pp. 634–639.
[18] J.A. Fernández, J. Minker, Disjunctive deductive database, in: Proceedings of the International Conference on Logic Programming and Automated Reasoning, Lecture Notes in Artificial Intelligence 624, Springer, Berlin, 1992, pp. 332–356.
[19] M. Gelfond, On stratified autoepistemic theories, in: Proceedings of AAAI–87, Morgan Kaufmann, Los Altos, 1987, pp. 297–211.
[20] M. Gelfond, Strong introspection, in: Proceedings of AAAI–91, AAAI/MIT Press, Cambridge, MA, 1991, pp. 386–391.
[21] M. Gelfond, V. Lifschitz, The stable model semantics for logic programming, in: R.A. Kowalski, K.A. Bowen (Eds.), Logic Programming: Proceedings of the Fifth International Conference and Symposium, MIT Press, Cambridge, MA, 1988, pp. 1070–1080.
[22] M. Gelfond, V. Lifschitz, Classical Negation in Logic Programs and Disjunctive Databases, New Generation Computing 9 (3,4) (1991) 365–385.
[23] M. Gelfond, V. Lifschitz, H. Przymusinska, M. Truszczyński, Disjunctive defaults, in: J. Allen, R. Fikes, E. Sandewall (Eds.), Principles of Knowledge Representation and Reasoning: Proceedings of the Second International Conference, Morgan Kaufmann, Los Altos, CA, 1991, pp. 230–237.
[24] M. Gelfond, H. Przymusinska, T. Przymusinska, On the relationship between circumscription and negation as failure, Artificial Intelligence 38 (1989) 75–94.
[25] G. Gottlob, Complexity results for nonmonotonic logics, Journal of Logic and Computation 2 (3) (1992) 397–425.
[26] K. Inoue, Hypothetical reasoning in logic programs, Journal of Logic Programming 18 (3) (1994) 191–227.
[27] K. Inoue, M. Koshimura, R. Hasegawa, Embedding negation as failure into a model generation theorem prover, in: D. Kapur (Ed.), Automated Deduction: Proceedings of the Eleventh International Conference, Lecture Notes in Artificial Intelligence 607, Springer, Berlin, 1992, pp. 400–415.
[28] K. Inoue, Y. Ohta, R. Hasegawa, M. Nakashima, Bottom-up abduction by model generation, in: Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence, Morgan Kaufmann, Los Altos CA, 1993, pp. 102–108.

[29] K. Inoue, C. Sakama, On positive occurrences of negation as failure, in: J. Doyle, E. Sandewall, P. Torasso (Eds.), Principles of Knowledge Representation and Reasoning: Proceedings of the Fourth International Conference, Morgan Kaufmann, Los ALtos, CA, 1994, pp. 293–304.
[30] K. Inoue, C. Sakama, A fixpoint characterization of abductive logic programs, Journal of Logic Programming 27 (2) (1996) 107–136.
[31] A.C. Kakas, P. Mancarella, Generalized stable models: A semantics for abduction, in: Proceedings of the Ninth European Conference on Artificial Intelligence, Pitman, London, 1990, pp. 385–391.
[32] A.C. Kakas, R.A. Kowalski, F. Toni, Abductive logic programming, Journal of Logic and Computation 2 (6) (1992) 719–770.
[33] K. Konolige, On the relation between default and autoepistemic logic, Artificial Intelligence 35 (1988) 343–382.
[34] K. Konolige, An autoepistemic analysis of metalevel reasoning in logic programming, in: Proceedings of the Third International Workshop on Meta-Programming in Logic, Lecture Notes in Computer Science 649, Springer, Berlin, 1992, pp. 26–48.
[35] R. Kowalski, J.S. Kim, A metalogic programming approach to multi-agent knowledge and belief, in: V. Lifschitz (Ed.), Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy, Academic Press, New York, 1991, pp. 231–246.
[36] H.J. Levesque, All I know: A study of autoepistemic logic, Artificial Intelligence 42 (1990) 263–309.
[37] V. Lifschitz, Computing circumscription, in: Proceedings of the Ninth International Joint Conference on Artificial Intelligence, Morgan Kaufmann, Los Altos CA, 1985, pp. 121–127.
[38] V. Lifschitz, Minimal belief and negation as failure, Artificial Intelligence 70 (1994) 53–72.
[39] V. Lifschitz, G. Schwarz, Extended logic programs as autoepistemic theories, in: [50], pp. 101–114.
[40] V. Lifschitz, H. Turner, From disjunctive programs to abduction, in: J. Dix, L.M. Pereira, T.C. Przymusinski (Eds.), Non-Monotonic Extensions of Logic Programming: Selected Papers from the ICLP '94 Workshop, Lecture Notes in Artificial Intelligence 927, Springer, Berlin, 1995, pp. 23–42.
[41] V. Lifschitz, T.Y.C. Woo, Answer sets in general nonmonotonic reasoning (Preliminary Report), in: B. Nebel, C. Rich, W. Swartout (Eds.), Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference, Morgan Kaufmann, 1992, Los Altos, CA, pp. 603–614.
[42] J.W. Lloyd, Foundations of Logic Programming, 2nd ed., Springer, Berlin, 1987.
[43] V.W. Marek, A. Nerode, B. Remmel, On logical constraints in logic programming, in: [44], pp. 43–56.
[44] V.W. Marek, A. Nerode, M. Truszczyński, (Eds.), Logic Programming and Nonmonotonic Reasoning: Proceedings of the Third International Conference, Lecture Notes in Artificial Intelligence 928, Springer, Berlin, 1995.
[45] W. Marek, V.S. Subrahmanian, The relationship between stable, supported, default and autoepistemic semantics for general logic programs, Theoretical Computer Science 103 (1992) 365–386.
[46] V.W. Marek, M. Truszczyński, Reflexive autoepistemic logic and logic programming, in: [50], pp. 115–131.
[47] J. McCarthy, Circumscription – A form of nonmonotonic reasoning, Artificial Intelligence 13 (1980) 27–39.
[48] J. Minker, On indefinite data bases and the closed world assumption, in: Proceedings of the Sixth International Conference on Automated Deduction, Lecture Notes in Computer Science 138, Springer, Berlin, 1982, pp. 292–308.
[49] R.C. Moore, Semantical considerations on nonmonotonic logic, Artificial Intelligence 25 (1985) 75–94.
[50] L.M. Pereira, A. Nerode (Eds.), Logic Programming and Non-monotonic Reasoning: Proceedings of the Second International Workshop, MIT Press, Cambridge, MA, 1993.
[51] T.C. Przymusinski, Stable semantics for disjunctive programs, New Generation Computing 9 (3/4) (1991) 401–424.
[52] R. Reiter, A logic for default reasoning, Artificial Intelligence 13 (1980) 81–132.
[53] C. Sakama, Possible model semantics for disjunctive databases, in: Proceedings of the First International Conference on Deductive and Object-Oriented Databases, Ohmsha, 1989, pp. 337–351.
[54] C. Sakama, K. Inoue, Relating disjunctive logic programs to default theories, in: [50], pp. 266–282.
[55] C. Sakama, K. Inoue, An alternative approach to the semantics of disjunctive logic programs and deductive databases, Journal of Automated Reasoning 13 (1) (1994) 145–172.

[56] C. Sakama, K. Inoue, On the equivalence between disjunctive and abductive logic programs, in: P.V. Hentenryck (Ed.), Logic Programming, Proceedings of the Eleventh International Conference, MIT Press, Cambridge, MA, 1994, pp. 489–503.

[57] C. Sakama, K. Inoue, Embedding circumscriptive theories in general disjunctive programs, in: [44], pp. 344–357.

[58] C. Sakama, K. Inoue, Representing priorities in logic programs, in: M. Maher (Ed.), Logic Programming: Proceedings of the 1996 Joint International Conference and Symposium, MIT Press, Cambridge, MA, 1996, pp. 82–96.

[59] J.S. Schlipf, Formalizing a logic for logic programming, Annals of Mathematics and Artificial Intelligence 5 (1992) 279–302.

[60] G.F. Schwarz, Autoepistemic logic of knowledge, in: A. Nerode, W. Marek, V.S. Subrahmanian (Eds.), Proceedings First International Workshop on Logic Programming and Non-monotonic Reasoning, MIT Press, Cambridge, MA, 1991, pp. 260–274.