TR-607

Extended Logic Programs with Default Assumptions

by
K. Inoue

December, 1990

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03)3456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

Extended Logic Programs with Default Assumptions *

Katsumi Inoue

ICOT Research Center
Institute for New Generation Computer Technology
Mita Kokusai Bldg. 21F
1-4-28 Mita, Minato-ku, Tokyo 108, Japan
phone: +81-3-3456-2514
email: inoue@icot.or.jp

February 14, 1991

Abstract

Extended logic programs were proposed by Gelfond and Lifschitz (1990) as logic programs with classical negation capable of expressing incomplete knowledge. Their work is expanded in this paper to deal with broader classes of commonsense knowledge. Like Poole's framework, some clauses are dealt with as assumptions distinct from a theory about the world and are used to augment the theory. This theory formation framework can be used for default reasoning, abduction and inconsistency resolution. We also show a translation of the framework to an extended logic program whose answer sets correspond to consistent belief sets of augmented theories.

Keywords: classical negation, theory formation, default reasoning, abduction

*This is an extended version of a paper that is to appear under the same title in the *Proceedings of the Eighth International Conference on Logic Programming (ICLP '91)* (Paris, June 1991), MIT Press.

1 Introduction

Recent investigations in theories of logic programming have revealed the close relationship between the semantics of logic programming and other theories of nonmonotonic reasoning developed in AI: negation as failure in logic programming is a nonmonotonic operator. This relationship opened up the new application of logic programming to commonsense reasoning. To deal with incomplete information easily, Gelfond and Lifschitz [8] extended the class of general logic programs by including classical negation, in addition to negation as failure, and showed ways to represent some incomplete knowledge by extended logic programs. The semantics of an extended logic program is given by the *answer sets*, which is a suitable extension of the *stable models* [7] of a general logic program. As a result of incorporation of classical negation in extended logic programs, the notion of *consistency* becomes more important.

In this paper, we expand the idea of Gelfond and Lifschitz extensively, and present methods to deal with broader classes of commonsense knowledge. To fill gaps in knowledge, one wants to represent and use "default" and "prototypic" knowledge. However, since defaults are usually inconsistent as a whole, simply adding all defaults to the theory would often result in no consistent answer set in the framework of Gelfond and Lifschitz. To overcome this difficulty, we shall deal with *default knowledge* as a part of knowledge distinct from a theory about the world, and use defaults to augment the theory and to predict what we expect to be true. This view of default reasoning can be best seen in Poole's framework for consistency-based hypothetical reasoning [20], which relates the Theorist framework [21] to Reiter's default logic [25]. Formally, a *knowledge system* K is represented by a pair, (T, H), where

1. Each of T and H is an *extended logic program*, that is, a set of *clauses* of the form

$$L_0 \leftarrow L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n\,,$$

   where $n \geq m \geq 0$, and each L_i is a *literal*, a formula of the form A or $\neg A$ (A is an atom),

2. T represents a set of *facts* that are known to be true in the domain, and

3. H represents a set of possible *assumptions* that may be expected to be true as defaults.

Then, the main task of a knowledge system is *theory formation*, that is, to find a subset E of H such that $T \cup E$ is consistent (such that there is a consistent answer set of $T \cup E$). We would not like to accept an incoherent theory (a theory with no answer set) as a set of beliefs. By using this mechanism, two types of reasoning can be performed:

1. *Default reasoning.* Find a maximal (with respect to set inclusion) subset E of H such that $T \cup E$ is consistent. Such a maximal set E is the basis of an expansion of the incomplete theory in accordance with default reasoning. The notion of such an answer set of $T \cup E$ corresponds to the set of literals that belong to an *extension* in [20].

2. *Abduction.* Find an explanation E ($\subseteq H$) of a formula O such that (i) $T \cup E$ is consistent and (ii) O is derived from $T \cup E$. The second condition may be expressed in either of the two possible ways: there is an answer set of $T \cup E$ which satisfies O; or, O is satisfied by every answer set of $T \cup E$.

The syntactical difference of our knowledge system from Poole's framework [20] is that while the latter uses the first-order predicate calculus, ours uses extended logic programs to formalize commonsense knowledge. Therefore, another view of the goal of this paper is to demonstrate what occurs if a hypothetical reasoning framework is represented by extended logic programs. Unfortunately, because our knowledge representation language contains the nonmonotonic operator *not* and the constructive implication $\leftarrow$, a knowledge system cannot inherit some of the elegant properties of Poole's framework. In particular, the fact that a formula has an explanation does not imply that the formula holds in an extension (an example will appear in Example 3.13). In this sense, default reasoning is clearly distinguished from abduction. If H represents a set of defaults, then an explanation is acceptable only when it is included in a maximal subset E of H such that $T \cup E$ is consistent.

A simple form of default assumptions can be represented by

$$L \leftarrow not\, \overline{L}\,,$$

where L is a literal and $\overline{L}$ is the literal complementary to L: for instance, when A is an atom, $\overline{A} = \neg A$ and $\overline{\neg A} = A$. These assumptions are also considered by Gelfond and Lifschitz [8] as the *closed world assumption* or assumable atomic predicates. However, they don't deal with them as assumptions distinct from the theory, but include them in the programs. For example, let a theory consist of two clauses:

$$Q \leftarrow \neg P(A), \neg P(B)\,,$$
$$\neg Q \leftarrow,$$

and let us consider the closed world assumption for the predicate P:

$$\neg P(x) \leftarrow not\, P(x)\,.$$

If these clauses are conjoined, no answer set is available. Instead, we would like to get two consistent answer sets, $\{\neg P(A), \neg Q\}$ and $\{\neg P(B), \neg Q\}$, by dealing with the assumptions as distinct clauses which can be invalidated or ignored when they cause inconsistencies. Moreover, sometimes assumptions may be added to make an incoherent program have consistent

answer sets. We will firstly consider this simple form of default assumptions. Then later we will extend the framework to deal with *any* extended logic program as default assumptions.

A naive computation to find each maximal consistent set of assumptions would be carried out to search through the power set of H, starting from the whole set H as the initial E and removing one clause from E at a time until we get consistent answer sets of $T \cup E$. We will show alternative methods for the computation by translating a knowledge system $K = (T, H)$ to an extended logic program K^* so that each answer set of K^* corresponds to an answer set of $T \cup E$ where E is a consistent subset of H. Thus the proposed framework can be viewed as a system for *inconsistency resolution*.

Finally, the proposed framework will be compared to other hypothetical reasoning systems based on logic programming in Section 5. Our system differs from abductive frameworks proposed by Gelfond [10] and Kakas and Mancarella [15], which do not deal with default reasoning. The proposed method for inconsistency resolution can be applied to much broader classes of default knowledge than Kowalski and Sadri's system [16], which handles only a simple form of exceptions. Also, the method is different from the TMS-style contradiction resolution [11]. Basically, the TMS does not deal with retractable assumptions, and when a contradiction occurs, the TMS imposes a new clause to believe a literal that has not been believed. Because our system represents assumptions explicitly, some assumptions are invalidated to remove incoherency; other clauses are not affected and new clauses are never added. In this sense, the proposed framework can be considered as a generalization of nonmonotonic ATMSs [4, 14].

2 Classical Negation and Consistency

This section presents basic properties of extended logic programs that were introduced by Gelfond and Lifschitz [8], on which our framework of theory formation is based. As seen in later sections, even in the notion of the simplest form of assumptions—assuming ground atomic formulas—there is the concept of classical negation. An atom A can be assumed to be true if it is consistent with a theory, that is, if $\neg A$ is not derived from a theory. One may write this kind of assumptions by using a propositional letter, something like A', as $A \leftarrow not\ A'$. However, such an introduced proposition again imposes the concept of consistency because A and A' cannot be believed at the same time. In fact, classical negation can be shown to be computationally eliminated in such a way in [8]. Therefore, it is quite natural to represent hypothetical reasoning, whose central part is maintaining consistency, by using extended logic programs.

In the semantics of extended logic programs, a clause containing variables stands for the set of its ground instances. We denote by *Lit* the set of ground literals in the language. Then the semantics of an extended logic program is given by its *answer sets*.

Definition 2.1 [8] Let Π be a set of ground clauses not containing *not*. The *answer set*, $\alpha(\Pi)$, of Π is the smallest subset S of *Lit* such that

1. for any clause $L_0 \leftarrow L_1, \ldots, L_m \in \Pi$, if $L_1, \ldots, L_m \in S$, then $L_0 \in S$, and
2. if S contains a pair of complementary literals, then $S = Lit$.

Definition 2.2 [8] Let Π be any extended logic program. A set $S \subseteq Lit$ is an *answer set* of Π if S is the answer set of Π^S, that is, $S = \alpha(\Pi^S)$, where Π^S is the set of clauses without *not* obtained from Π by deleting

1. every clause containing a formula *not* L in its body with $L \in S$, and
2. every formula *not* L in the bodies of the remaining clauses.

Intuitively speaking, each answer set is a possible set of beliefs: each literal in an answer set can be considered to be true in the belief set, and any literal not contained in an answer set is believed to be neither true nor false in that belief set. In this semantics, for each atom A, positive and negative literals have the same status so that the result of negation by failure to prove A means neither that A is false nor that $\neg A$ is true [1].

If Π is a *general* logic program, i.e., a set of clauses without classical negation, then the answer sets of Π are identical to the stable models of Π given by Gelfond and Lifschitz [7]. For convenience, we classify extended logic programs as follows.

Definition 2.3 Let Π be an extended logic program. Π is *consistent* if it has a consistent answer set. Π is *contradictory* if it has an inconsistent answer set. Π is *incoherent* if it has no answer set.

The above definition is exclusive and complete: every program is either consistent, contradictory, or incoherent. This is verified by the following two propositions.

Proposition 2.4 *(Minimality of answer sets* [8]*) Let Π be an extended logic program. For any two answer sets S and S' of Π, if $S \subseteq S'$ then $S = S'$.*

Proposition 2.5 *No extended logic program is both consistent and contradictory, and a contradictory program has only one answer set Lit.*

[1] This is a big difference from well-founded semantics [22] or stationary semantics [23]: we do not allow the inference that if A does not match the head of any clause of Π in accordance with the default reasoning behind negation as failure, then put A into the false part.

Gelfond and Lifschitz [8] show the relation between the answer sets of an extended logic program and *extensions* of the corresponding Reiter's default theory [25]. Every clause in an extended logic program Π of the form

$$L_0 \leftarrow L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n \tag{1}$$

can be identified with the *default rule*

$$\frac{L_1 \wedge \ldots \wedge L_m\ :\ \mathsf{M}\,\overline{L_{m+1}}, \ldots, \mathsf{M}\,\overline{L_n}}{L_0}.$$

According to [8], there is a 1-1 correspondence between the answer sets of Π and the extensions of the default theory $(\Pi, \emptyset)$. Note that a clause not containing *not*

$$L_0 \leftarrow L_1, \ldots, L_m \tag{2}$$

can be identified with the default rule

$$\frac{L_1 \wedge \ldots \wedge L_m\ :}{L_0}.$$

While the last form of default rules are not excluded by Reiter's definition, the existence of at least one justification for each default rule is presupposed in [25, Corollary 2.2], which says a closed default theory (D, W) has an inconsistent extension if and only if W is inconsistent [2]. In our case, the default theory $(\Pi, \emptyset)$ may have an inconsistent extension even though the set of wffs W is empty. The precise characterization of contradictory programs can be given as follows.

Proposition 2.6 *An extended logic program Π is contradictory if and only if the set of clauses of the form (2) (i.e., the clauses without not) in Π is contradictory.*

Proof: *Lit* is an answer set of Π if and only if *Lit* is the answer set of Π^{Lit} that is the set of clauses obtained from Π by deleting every clause containing a formula *not* L in its body (by Definition 2.2) if and only if Π^{Lit} has an inconsistent answer set (by Definition 2.1). □

[2] As far as the author knows, this observation for *justification-free defaults* has first been discussed by Brewka [1]. These default rules cannot be replaced with

$$\frac{L_1 \wedge \ldots \wedge L_m\ :\ \mathsf{M}\,true}{L_0}.$$

The above proposition tells us that for a contradictory program, contradictions may not be removed even if either any clause is added to the program or any clause with *not* is removed from the program (see Proposition 3.2). Thus our main goal is to resolve incoherent programs rather than contradictory programs. Although Gelfond and Lifschitz insist that the class of extended logic programs is the place where logic programming meets default logic halfway, the relation itself does not provide us how to do default reasoning by using extended logic programs because every clause is identified with a default rule, and because considering all clauses together may result in an incoherent program, as seen in Section 1.

3 Theory Formation

The last observation encourages us to split an extended logic program Π into two parts (T, H) for hypothetical reasoning such that $T \cup H = \Pi$ and $T \cap H = \emptyset$, where T stands for a set of facts and H for a set of assumptions that may be expected to be true. The resulting system is called a *knowledge system*. As explained in Section 1, the main task of a knowledge system is *theory formation*, that is, to find a subset E of H such that $T \cup E$ is consistent.

Definition 3.1 Let $K = (T, H)$ be a knowledge system. K is *consistent* if there is a set $E \subseteq H$ such that $T \cup E$ is consistent. K is *contradictory* if for any set $E \subseteq H$, $T \cup E$ is contradictory. K is *incoherent* if K is neither consistent nor contradictory.

The above definition is exclusive and complete.

Proposition 3.2 *(Correspondence with* [25, Corollary 2.2]*) A knowledge system (T, H) is contradictory if and only if T is contradictory.*

Proof: The only-if-part is obvious from Definition 3.1. The if-part is a direct consequence of Proposition 2.6. □

Proposition 3.3 *Let $K = (T, H)$ be a knowledge system. If T is consistent then K is consistent. If K is incoherent then T is incoherent.*

Proof: The first claim is obvious from Definition 3.1. We prove the contrapositive of the second claim. Suppose that T is not incoherent, that is, T is either consistent or contradictory. If T is consistent, then K is consistent by the first claim. If T is contradictory, then K is contradictory by Proposition 3.2. In both cases, K is not incoherent. □

The converse directions of Proposition 3.3 do not hold. Adding assumptions to an incoherent program may make the knowledge system obtain consistent answer sets.

Example 3.4 Let us consider a knowledge system (T, H) where $T = \{ P \leftarrow not\, P \}$ and $H = \{ P \leftarrow \}$. While T is incoherent, $T \cup H$ has a consistent answer set $\{P\}$.

In the following subsections, we will consider formalizations for two kinds of commonsense reasoning by using theory formation.

3.1 Default Reasoning

One of the most obvious and important applications of theory formation is default reasoning, where default assumptions are assumed to be true unless there is evidence to the contrary. Thus as many assumptions as possible are taken into account in a set of beliefs. The notion of such an answer set of the augmented program by a maximal consistent set of assumptions roughly corresponds to the set of literals that belong to an extension in [20].

Definition 3.5 Let $K = (T, H)$ be a knowledge system. An *extension base* of K is an answer set of $T \cup E$ where E is a maximal (with respect to set inclusion) subset of H such that $T \cup E$ is consistent.

For default reasoning, the task of a knowledge system is to get its extension bases. This framework can make a contradictory or incoherent program Π become a consistent knowledge system (T, H) such that $\Pi = T \cup H$, provided that prototypic or typical knowledge is appropriately put into a set H of default assumptions that is distinct from a set T of clauses representing factual or exceptional knowledge. To obtain extension bases, some assumptions are allowed to be ignored, but no assumption can be dispensed with unless it is necessary to do so.

Example 3.6 Suppose we have a knowledge system $K = (T, H)$, where

$$\begin{array}{rll} T = \{ & \neg Flies(x) \leftarrow Penguin(x), & \\ & Bird(x) \leftarrow Penguin(x), & \\ & Bird(Polly) \leftarrow, & \\ & Penguin(Tweety) \leftarrow & \}, \\ H = \{ & Flies(x) \leftarrow Bird(x) & \}. \end{array}$$

Here it is easy to see that $T \cup H$ is contradictory. There is the unique extension base of K: $\{ Bird(Polly), Flies(Polly), Penguin(Tweety), Bird(Tweety), \neg Flies(Tweety) \}$. Notice that the assumption is used for $x = Polly$ but is ignored for $x = Tweety$. In this case the extended logic program

$$T \cup \{ Flies(x) \leftarrow Bird(x), not\, \neg Flies(x) \}$$

has the unique answer set that is identical to the extension base of K. The reason why the translation works is that the exceptional clause $\neg Flies(Tweety) \leftarrow Penguin(Tweety)$ cancels the normal default rule $Flies(Tweety) \leftarrow Bird(Tweety), not\ \neg Flies(Tweety)$. This translation is similar to the method used in Kowalski and Sadri's system [16].

Example 3.7 (Barber's Paradox) Let $K = (T, H)$ be a knowledge system where

$$T = \{\ \neg Shaves(Ken, Ken) \leftarrow\ \},$$
$$H = \{\ Shaves(Jun, x) \leftarrow not\ Shaves(x, x)\ \}.$$

Here $T \cup H$ is incoherent because the clause $Shaves(Jun, Jun) \leftarrow not\ Shaves(Jun, Jun)$ is present in the program. This default is ignored in the unique extension base of K that is $\{\neg Shaves(Ken, Ken), Shaves(Jun, Ken)\}$. In this case, $\neg Shaves(Ken, Ken)$ is not an exception of the default conclusion, and therefore the default cannot be translated in the same way as Example 3.6.

3.2 Closed World Assumption

Another interesting application along this line of theory formation is the *closed world assumption* (CWA) [24] for some predicates in the language. Gelfond and Lifschitz use the CWA to fill the gap between a stable model of a general logic program Π and an answer set of Π when interpreted as an extended logic program: an extended program Π' that consists of the clauses of Π and the CWA for each predicate P with n distinct variables in the language

$$\neg P(x_1, \ldots, x_n) \leftarrow not\ P(x_1, \ldots, x_n) \tag{3}$$

precisely characterizes the meaning of Π in stable model semantics [8, Proposition 4].

Proposition 3.8 *Let Π be an extended logic program without classical negation. And let $\Pi' = \Pi \cup CW$ where CW is the CWA (3) for all predicates as above. Then, S is an answer set of Π' if and only if S is an extension base of the knowledge system (Π, CW).*

Proposition 3.8 says that the CWA for all (or some specific) predicates is consistent with any coherent general logic program. However, as shown in Section 1, if the CWA is used together with an extended logic program then the augmented program is not consistent in general. In fact, for an extended logic program Π, Proposition 3.8 does not hold. Thus, we would not like to assume all negative ground literals even if each of them can be consistently assumed [3]. For a ground positive literal A, $\neg A$ can be assumed in a belief set if it is a member

[3]This problem is analogous to the application of the CWA to non-Horn clauses in databases, which may produce an inconsistent augmentation [24]. For disjunctive databases, Minker [18] proposes the generalized closed world assumption (GCWA) which concludes $\neg A$ for a ground positive literal A if A is false in every minimal model of the clauses. In our case, instead of simply using the minimal models, A can be tested for the membership in the extension bases.

of an extension base of (Π, CW), and $\neg A$ can be concluded to hold if it is contained in every extension base of (Π, CW).

Example 3.9 Let an extended logic program Π consist of the following three clauses:

$$\begin{array}{l} Q \leftarrow \neg P(A)\,, \\ \neg Q \leftarrow \neg P(B)\,, \\ P(C) \leftarrow P(A),\ P(B)\,. \end{array}$$

And suppose that H consists of the CWA for the predicate P:

$$\neg P(x) \leftarrow not\ P(x)\,.$$

It is easy to see that $\Pi \cup H$ is incoherent. Now consider a knowledge system $K = (\Pi, H)$. There are two extension bases of K: $\{\ \neg P(A),\ \neg P(C),\ Q\ \}$ and $\{\ \neg P(B),\ \neg P(C),\ \neg Q\ \}$. Since $\neg P(C)$ is contained in both extension bases, it can be concluded.

Unlike Poole's system [20], semimonotonicity [25] does not hold even if either all default assumptions in a knowledge system are clauses without bodies or they can be identified with Reiter's normal default rule

$$L_0 \leftarrow L_1, \ldots, L_m,\ not\ \neg L_0\,.$$

In other words, when we have two knowledge systems $K = (T, H)$ and $K' = (T, H')$ such that $H' \subseteq H$, for an extension base S' of K', there may not be an extension base S of K such that $S' \subseteq S$. This is because the clauses T can be identified with Reiter's nonnormal defaults.

Example 3.10 Let $K = (T, H)$ and $K' = (T, H')$ be two knowledge systems where

$$\begin{array}{rl} T = \{ & P \leftarrow B\,, \\ & Q \leftarrow A,\ not\ P\,, \\ & P \leftarrow not\ Q \qquad \}\,, \\ H = \{ & A \leftarrow not\ \neg A\,, \\ & B \leftarrow not\ \neg B \qquad \}\,, \\ H' = \{ & A \leftarrow not\ \neg A \qquad \}\,. \end{array}$$

K' has two extension bases: $S_1 = \{\ A,\ Q\ \}$ and $S_2 = \{\ A,\ P\ \}$. Clearly $H' \subseteq H$. However, S_1 is not a subset of the unique extension base of K: $S = \{\ A,\ B,\ P\ \}$.

Although many other theories for default reasoning, such as multiple extension problems, priorities between defaults and skeptical reasoning for inheritance hierarchies, are out of the scope of this paper, it might be possible to incorporate them in this framework. Thanks to the

two connectives in logic programming, the nonmonotonic operator *not* and the constructive implication $\leftarrow$, some of these topics would be more naturally dealt with than by systems using the first-order predicate calculus as knowledge representation language [4].

3.3 Abduction

Theory formation was originally motivated by the goal of providing a formal account of inference to the best explanation of observations. This inference has been known as abduction. In this case, the task of a knowledge system is to find an *explanation* E of a given formula O as follows.

Definition 3.11 Let $K = (T, H)$ be a knowledge system, and O be a formula. A set $E \subseteq H$ is an *explanation of* O *(with respect to* K*)* if:

1. $T \cup E$ is consistent, and
2. O is *derived from* $T \cup E$.

While the first condition is clear, the meaning of the second condition is somewhat controversial. It may be expressed in either of the following [5]:

(a) there is an answer set of $T \cup E$ which satisfies O.

(b) O is satisfied by every answer set of $T \cup E$.

Here, we assume that O is simply a conjunction of literals, and we say that O is *satisfied* by a set $S \subseteq Lit$ if every literal in O is contained in S [6]. We write E *explains*$_1$ O if E is an explanation of O in the sense of the first definition of derivability (a), and write E *explains*$_2$ O if E is an explanation of O in the sense of the second definition of derivability (b).

[4] Another feature of using $\leftarrow$ is that, as discussed in [16], while Poole's system [20] needs constraints to prevent the use of contrapositives of clauses, they are not necessary for extended logic programs.

[5] For restricted Hs, the first criterion is used in [15], and the second is in [10].

[6] The assumption that an explained formula is a conjunct of literals can be reduced when the model theoretic semantics for *not* and $\leftarrow$ is provided. For this purpose, we need the following three-valued semantics. Let L be a literal, G be a conjunct of literals and/or formulas with *not*, and $S \subseteq Lit$. L is true in S if $L \in S$; false if $\overline{L} \in S$; otherwise *unknown*. *not* L is true in S if $L \notin S$; otherwise false. G is true in S if every element in G is true in S; false if at least one element in G is false in S; otherwise *unknown*. $L \leftarrow G$ is true in S if either both G and L is true in S or G is *not* true in S; otherwise false. For example, $P \leftarrow \neg P$ is true in $\emptyset$ and in $\{P\}$ but false in $\{\neg P\}$. $P \leftarrow$ *not* P is false in $\emptyset$ and in $\{\neg P\}$ but true in $\{P\}$. S is a *three-valued model* of a set Π of clauses if every clause in Π is true in S. By using this semantics, we can show that every answer set of Π is a minimal (in the sense of set inclusion of literals) three-valued model of Π. Note that this semantics differs from the model theory for three-valued stable model semantics given by Przymusinski [22, 23].

Unlike Poole's system, the fact that a formula has an explanation does not imply that the formula is satisfied by an extension base. That is, for knowledge systems, explicability and membership in an extension differ. In this sense, default reasoning is clearly distinguished from abduction. In default reasoning, a set H of assumptions is used as *defaults*, whereas in abduction it is used as *premises*. If H represents a set of defaults, then an explanation is acceptable only when it is included in a maximal subset E of H such that $T \cup E$ is consistent. In other words,

Definition 3.12 Let $K = (T, H)$ be a knowledge system, and O be a formula. Assume that H represents a set of *defaults* and that $E \subseteq H$. A set E *explains* O *by defaults* (or, E *plausibly explains* O) *(with respect to* K*)* if:

1. E *explains*$_1$ O,
2. there is a maximal set E' such that
   (a) $E \subseteq E' \subseteq H$, and
   (b) $T \cup E'$ is consistent and has answer sets $\mathcal{A}$ (= the extension bases of K),
3. and either
   (a) there is a set $S \in \mathcal{A}$ satisfying O (written E *explains*$_3$ O), or
   (b) every set $S \in \mathcal{A}$ satisfies O (written E *explains*$_4$ O).

K *cautiously predicts* O (written K *predicts*$_5$ O) if every extension base of K satisfies O.

Example 3.13 Let $K = (T, H)$ be a knowledge system where

$$
\begin{array}{lll}
T = \{ & P \leftarrow B, & \\
 & Q \leftarrow A, \mathit{not}\, P, & \\
 & P \leftarrow \mathit{not}\, Q & \}, \\
H = \{ & A \leftarrow, & \\
 & B \leftarrow & \}.
\end{array}
$$

1. $E_1 = \{ A \leftarrow \}$. $T \cup E_1$ has two answer sets: $S_1 = \{ A, Q \}$ and $S_2 = \{ A, P \}$.
   E_1 *explains*$_1$ both Q and P, but cannot *explains*$_1$ $P \wedge Q$.
   E_1 *explains*$_2$ neither Q nor P.
2. $E_2 = \{ B \leftarrow \}$. $T \cup E_2$ has the unique answer set: $S_3 = \{ B, P \}$.
   E_2 *explains*$_1$ and *explains*$_2$ P.

3. $H = E_1 \cup E_2$. K has the unique extension base: $S = \{ A, B, P \}$.
   H *explains*$_{\mathrm{i}}$ P for every $\mathrm{i} = 1, 2, 3, 4$, and K *predicts*$_5$ P.
   E_1 (and E_2) *explains*$_3$ and *explains*$_4$ P.
   Q can be neither *explained*$_3$ nor *explained*$_4$.

If we follow the first definition of derivability, Q has an explanation E_1 because S_1 contains Q. However, since S_1 is not a subset of the unique extension base S of K, Q does not hold in an extension. Notice that in this case E_1 can also explain P because S_2 contains P. It is curious that $P \wedge Q$ cannot be explained by E_1 while E_1 can explain both P and Q.

If we use the second definition of derivability, Q cannot be explained from K because S_2 does not satisfy Q. In this case, P cannot be explained by E_1 either, but P can be explained by either E_2 or H.

Since $T \cup H$ is consistent, if H represents default knowledge, then P can be explained *by defaults*, but Q cannot be explained, whichever definition of derivability we choose.

4 Reduction to Extended Logic Programs

In this section, we will show a method of the transformation from any knowledge system $K = (T, H)$ to an extended logic program K^* such that the extension bases of K correspond to a class of the answer sets of K^*. Recall that even if a program Π is incoherent, an augmented program $\Pi' \supseteq \Pi$ may be consistent (see Example 3.4). Thus, for a set $E \subseteq H$ such that $T \cup E$ is incoherent, we cannot prune the supersets of E in 2^H to find an extension base of K. Hence, the methods have computational advantages because we can characterize all consistent answer sets of $T \cup E$ for any $E \subseteq H$ by analyzing the single program K^*.

In the following, for an extended logic program Π, without loss of generality we can assume that Π does not contain variables. And for Π, we denote the heads of the clauses of Π as

$$\mathsf{Head}(\Pi) = \{ L_0 \mid L_0 \leftarrow L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n \in \Pi \},$$

and the literals complementary to the heads of clauses in Π as

$$\overline{\mathsf{Head}}(\Pi) = \{ \overline{L_0} \mid L_0 \leftarrow L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n \in \Pi \}.$$

4.1 Simple Default Assumptions

We firstly consider simple knowledge systems each of whose assumptions is in the form of an assertion of a ground literal

$$L \leftarrow . \tag{4}$$

Since positive and negative literals are dealt with symmetrically in extended logic programs, we have no reason to restrict the simplest form of assumptions to being positive. Let us consider a knowledge system $K_0 = (T, H_0)$ where H_0 is a set of clauses of the form (4). We will translate K_0 to a non-contradictory program $T \cup H$ then to a consistent program $K^* = T \cup H^*$.

Adding all literal assertions in H_0 to the program T would result in a contradictory or incoherent program. For example, when H_0 is contradictory, $T \cup H_0$ must be contradictory by Proposition 2.6. To remove contradictions, we can simply block the application of a default (4) if it happens that $\overline{L}$ can be derived, by adding a formula $not\ \overline{L}$ to its body. Now let $K = (T, H)$ be the knowledge system obtained from $K_0 = (T, H_0)$ by replacing each clause in H_0 of the form (4) with a clause in H of the form

$$L \leftarrow not\ \overline{L}. \tag{5}$$

Then, $T \cup H$ is not contradictory if T is not contradictory. This is because H does not contain a clause of the form (2) so that the following properties can be shown to hold by Proposition 2.6.

Lemma 4.1 *Let $K = (T, H)$ be a knowledge system such that H is a set of clauses of the form (1) where $0 \leq m < n$. $T \cup H$ is contradictory if and only if T is contradictory.*

Corollary 4.2 *Let $K_0 = (T, H_0)$ and $K = (T, H)$ be two knowledge systems as above. K_0 is contradictory if and only if $T \cup H$ is contradictory.*

Definition 4.3 Let H be any extended logic program, and Π be any consistent extended logic program. An answer set S of Π is *H-maximal* if there is no answer set S' of Π such that $S \cap \mathsf{Head}(H) \subset S' \cap \mathsf{Head}(H)$.

For a knowledge system $K = (T, H)$, the distinction between the H-maximality of an answer set and an extension base of K is important. When $T \cup H$ is consistent, since every assumption is not ignored, every answer set of $T \cup H$ is an extension base of K, but it may not be an H-maximal answer set of $T \cup H$. On the other hand, when S is an H-maximal answer set of $T \cup E$ for some set $E \subseteq H$, it may not be an extension base of K. In an H-maximal answer set, assumptions in a maximal subset of H are used in practice, whereas in an extension base, assumptions just take part in a maximal subset of H but some of them may be canceled.

For the first translation, the next proposition is shown to hold.

Proposition 4.4 *Let $K_0 = (T, H_0)$ and $K = (T, H)$ be two knowledge systems as above. Suppose that $T \cup H$ is consistent. If S is an H-maximal answer set of $T \cup H$, then S is an extension base of K_0.*

Proof: We firstly prove that if S is an answer set of $T \cup H$, then S is an answer set of $T \cup H^S$. Suppose that S is an answer set of $T \cup H$. Since $H^S = \{ L \leftarrow \mid L \leftarrow not\, \overline{L} \in H,\ \overline{L} \notin S \} = \{ L \leftarrow \in H_0 \mid \overline{L} \notin S \} \subseteq H_0$ and $S = \alpha((T \cup H)^S) = \alpha((T \cup H^S)^S)$, S is an answer set of $T \cup H^S$.

Now, suppose that S is an H-maximal answer set of $T \cup H$. Suppose also to the contrary that S is not an extension base of K_0. Then, there exists a set E_0 ($H^S \subset E_0 \subseteq H_0$) such that $T \cup E_0$ is consistent. Let S' be an answer set of $T \cup E_0$. By $H^S \subset E_0 \subseteq H^{S'}$, clearly $\{ L \in S \mid L \in \mathsf{Head}(H) \} \subset \{ L \in S' \mid L \in \mathsf{Head}(H) \}$, contradicting the H-maximality of S. □

The converse of Proposition 4.4 does not hold: there is an extension base of $K_0 = (T, H_0)$ which is not an H-maximal answer set of $T \cup H$ (suppose a case that an extension base S contains neither L nor $\overline{L}$ for a literal $L \in \mathsf{Head}(H)$). Moreover, it cannot give every consistent answer set of $T \cup E_0$ for any set $E_0 \subseteq H_0$, which is sometimes useful for abduction.

Example 4.5 Let us consider a knowledge system $K_0 = (T, H_0)$ and the translated knowledge system $K = (T, H)$, where

$$
\begin{array}{lll}
T = \{ & \neg P \leftarrow not\, P\,, & \\
& C \leftarrow P, Q\,, & \\
& \neg C \leftarrow & \}\,, \\
H_0 = \{ & P \leftarrow, & \qquad H = \{ \quad P \leftarrow not\, \neg P\,, \\
& Q \leftarrow & \}\,, \qquad\qquad\quad Q \leftarrow not\, \neg Q \quad \}\,.
\end{array}
$$

There are two extension bases of K_0: $S_1 = \{ P, \neg C \}$ and $S_2 = \{ \neg P, Q, \neg C \}$. But only S_2 is the unique answer set of $T \cup H$. In S_1, neither Q nor $\neg Q$ holds. Note also that there is an answer set of $T = T \cup \emptyset$: $S_3 = \{\neg P, \neg C\}$, which cannot be obtained from the answer sets of $T \cup H$.

Another difficulty of Proposition 4.4 is that the consistency assumption for $T \cup H$ is indispensable. For example, we have seen in Section 1 and in Example 3.9 that an extended logic program with the CWA may be incoherent but the corresponding knowledge system may have extension bases. Therefore, adding all assumptions $L \leftarrow not\, \overline{L}$ in H to the program T would result in an incoherent program even if T is consistent. Thus our next target is to remove incoherencies. In the following translation, we can characterize all consistent answer sets of $T \cup E_0$ for any set $E_0 \subseteq H_0$ as well as each extension base of K_0.

Now, for knowledge systems $K_0 = (T, H_0)$ and $K = (T, H)$, we shall impose the following restriction on the syntax of T. This restriction will be removed completely in the next subsection.

> For any $L \leftarrow not\, \overline{L} \in H$ ($L \leftarrow \in H_0$), every clause in T does not contain L in its head, and contains neither $\overline{L}$ nor $not\, \overline{L}$ in its body. (6)

Although the restriction (6) on T seems strong, there are still three utilities of assumptions within this restriction. For each $L \leftarrow not\ \overline{L} \in H$, we allow T to include the following clauses:

1. *Conditioned conclusions:* $L_0 \leftarrow L, L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n$.

   L_0 may be concluded if L can be assumed to be true. For example, it can represent properties of normal cases.

2. *Cancellation of defaults:* $\overline{L} \leftarrow L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n$.

   This clause may block to assume L and represents a condition for an exception to hold.

3. *Exceptional conclusions:* $L_0 \leftarrow L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n, not\ L$.

   L_0 may be concluded if L cannot be assumed to be true. For example, it can represent properties of exceptional cases.

Example 4.6 For a default assumption in H

$$\neg Ab(x) \leftarrow not\ Ab(x),$$

the following clauses in T satisfy the condition (6):

$$\begin{array}{ll} Flies(x) \leftarrow Bird(x), \neg Ab(x) & \text{(conditioning)} \\ Ab(x) \leftarrow Ostrich(x) & \text{(cancellation)} \\ \neg Flies(x) \leftarrow not\ \neg Ab(x) & \text{(exception)}. \end{array}$$

Lemma 4.7 *Let $K = (T, H)$ be a knowledge system such that H is a set of clauses of the form (5) and T is not contradictory. Let E be a subset of H. If $T \cup E$ has an answer set S, then for each clause of the form (5) in E, S contains either L or $\overline{L}$ but not both of them.*

Proof: By Lemma 4.1, $T \cup E$ is not contradictory and hence S does not contain both L and $\overline{L}$. If S does not contain $\overline{L}$, then by the existence of $L \leftarrow not\ \overline{L}$, $(T \cup E)^S$ contains $L \leftarrow$ and so $L \in \alpha((T \cup E)^S) = S$. If S does not contain L, then $\overline{L}$ must be contained in S because if $\overline{L} \notin S$ then $L \in S$ holds by the same argument as above contradicting $L \notin S$. □

The basic idea of the next translation is that we *expand* each incomplete extension base S of $K_0 = (T, H_0)$ by adding an extra assumption for each literal undefined in S so that the augmented set of literals contains either L or $\overline{L}$ for each $L \in \mathsf{Head}(H_0)$ and is an answer set of $T \cup H \cup H'$ (H' is the added assumptions) by Lemma 4.7.

At present, we have the original knowledge system $K_0 = (T, H_0)$ such that H_0 is a set of the clauses of the form (4) and T satisfies the condition (6), and have the translated

knowledge system $K = (T, H)$ such that H is a set of the clauses of the form (5). Now, for each clause in H of the form (5)

$$L \leftarrow not\ \overline{L}\,,$$

we shall consider the opposite assumption of the form

$$\overline{L} \leftarrow not\ L\,. \tag{7}$$

For any subset E of H, we denote the set of opposite assumptions of the form (7) for E as

$$\overline{E} = \{\, \overline{L} \leftarrow not\ L \mid L \in \mathsf{Head}(E) \,\}\,.$$

and the union of E and $\overline{E}$ is denoted as $E^* = E \cup \overline{E}$. The result of the second translation is the extended logic program

$$K^* = T \cup H^* = T \cup H \cup \overline{H}\,.$$

The next is the main result of the second translation.

Theorem 4.8 *Let $K = (T, H)$ be a knowledge system such that H is a set of clauses of the form (5) and T satisfies the condition (6). If S is a consistent answer set of $T \cup E$ where E is a subset of H, then*

$$S' = S \cup \overline{\mathsf{Head}}(H \setminus E) \tag{8}$$

is a consistent answer set of $T \cup H^*$. Moreover, every consistent answer set of $T \cup H^*$ can be represented in the form (8) where S is a consistent answer set of $T \cup E$ for some set $E \subseteq H$.

Proof: Let S be a consistent answer set of $T \cup E$ ($E \subseteq H$). Since no literal $L \in \mathsf{Head}(H)$ appears in the head of any clause in T, for any literal $L \in \mathsf{Head}(H \setminus E)$, $L \notin S$. Therefore, S' is consistent. By Lemma 4.7, $\mathsf{Head}(E^S) \subseteq S$ and $(\overline{\mathsf{Head}}(E) \setminus \overline{\mathsf{Head}}(E^S)) \subseteq S$, it follows that $\mathsf{Head}(E^S) \subseteq S'$ and $\overline{\mathsf{Head}}(H \setminus E) \cup (\overline{\mathsf{Head}}(E) \setminus \overline{\mathsf{Head}}(E^S)) = (\overline{\mathsf{Head}}(H) \setminus \overline{\mathsf{Head}}(E^S)) \subseteq S'$. By the way, $T^{S'} = T^S$ (since no clause in T contains *not* $\overline{L}$ for any $\overline{L} \in \overline{\mathsf{Head}}(H)$ in its body), and $H^{*S'} = H^{S'} \cup \overline{H}^{S'} = \{\, L \leftarrow \mid L \in \mathsf{Head}(E^S) \,\} \cup \{\, \overline{L} \leftarrow \mid L \in (\mathsf{Head}(H) \setminus \mathsf{Head}(E^S)) \,\} = E^S \cup \{\, \overline{L} \leftarrow \mid L \in (\mathsf{Head}(H) \setminus \mathsf{Head}(E^S)) \,\}$. Now,

$$\begin{aligned}
\alpha((T \cup H^*)^{S'}) &= \alpha(T^{S'} \cup H^{*S'}) \\
&= \alpha(T^S \cup E^S \cup \{\, \overline{L} \leftarrow \mid L \in (\mathsf{Head}(H) \setminus \mathsf{Head}(E^S)) \,\}) \\
&= \alpha((T \cup E)^S) \cup (\overline{\mathsf{Head}}(H) \setminus \overline{\mathsf{Head}}(E^S)) \\
&\qquad \text{(since no clause in } T \text{ contains any } \overline{L} \in \overline{\mathsf{Head}}(H) \text{ in its body)} \\
&= S \cup \overline{\mathsf{Head}}(H \setminus E) \quad (\text{by } (\overline{\mathsf{Head}}(E) \setminus \overline{\mathsf{Head}}(E^S)) \subseteq S) \\
&= S'\,.
\end{aligned}$$

Hence, S' is a consistent answer set of $T \cup H^*$.

To prove the second claim, take any answer set S' of $T \cup H^*$, and define

$$E = \{ L \leftarrow not\ \overline{L} \in H \mid L \in S' \}.$$

Clearly, $E \subseteq H$ and $H^{S'} = \{ L \leftarrow \mid L \in \mathsf{Head}(E) \} = E^{S'}$. Then,

$$\begin{aligned} S' &= \alpha(T^{S'} \cup H^{*S'}) \\ &= \alpha(T^{S'} \cup H^{S'} \cup \overline{H}^{S'}) \\ &= \alpha(T^{S'} \cup E^{S'} \cup \{ \overline{L} \leftarrow \mid L \in \mathsf{Head}(H \setminus E) \}) \\ &= \alpha(T^{S'} \cup E^{S'}) \cup \overline{\mathsf{Head}}(H \setminus E). \end{aligned}$$

Now, let $S = \alpha(T^{S'} \cup E^{S'})$. Since $S' = S \cup \overline{\mathsf{Head}}(H \setminus E)$, $T^{S'} = T^S$ and $E^{S'} = E^S$ hold by the condition (6). Therefore, $S = \alpha(T^S \cup E^S) = \alpha((T \cup E)^S)$. □

Example 4.9 Let us verify Theorem 4.8 in the example of the CWA introduced in Section 1. Let $K = (T, H)$ be a knowledge system, where

$$\begin{aligned} T = \{ \ & Q \leftarrow \neg P(A), \neg P(B), \\ & \neg Q \leftarrow \qquad\qquad\quad \}, \\ H = \{ \ & \neg P(x) \leftarrow not\ P(x) \quad \}. \end{aligned}$$

In this case, $\overline{H} = \{ P(x) \leftarrow not\ \neg P(x) \}$. There are three answer sets of $T \cup H^*$: $S_1' = \{ \neg P(A), P(B), \neg Q \}$, $S_2' = \{ P(A), \neg P(B), \neg Q \}$, and $S_3' = \{ P(A), P(B), \neg Q \}$. By using the translation in the proof of the second claim of Theorem 4.8, we get the three corresponding answer sets:

$$\begin{aligned} S_1 &= \{ \neg P(A), \neg Q \} \quad \text{for } T \cup \{ \neg P(A) \leftarrow not\ P(A) \}, \\ S_2 &= \{ \neg P(B), \neg Q \} \quad \text{for } T \cup \{ \neg P(B) \leftarrow not\ P(B) \}, \text{ and} \\ S_3 &= \{ \neg Q \} \quad \text{for } T \cup \emptyset = T. \end{aligned}$$

The next two properties characterize the knowledge system K_0 with literal assumptions by the translated program K^*. These are the final results of this subsection.

Corollary 4.10 *Let $K_0 = (T, H_0)$ be a knowledge system such that H_0 is a set of clauses of the form* (4) *and T satisfies the condition* (6). *If S is a consistent answer set of $T \cup E_0$ where E_0 is a subset of H_0, then*

$$S' = S \cup \overline{\mathsf{Head}}(H_0 \setminus E_0) \tag{9}$$

is a consistent answer set of $T \cup H^*$. Moreover, every consistent answer set of $T \cup H^*$ can be represented in the form (9) *where S is a consistent answer set of $T \cup E_0$ for some set $E_0 \subseteq H_0$.*

Proof: The first claim can be proved in a similar way as Theorem 4.8. To prove the second claim, for any answer set S' of $T \cup H^*$, we can define $E_0 = \{ L \leftarrow \in H_0 \mid L \in S' \}$ and use the same argument as the previous proof. □

Theorem 4.11 *Let $K_0 = (T, H_0)$ be the same knowledge system as Corollary 4.10. If S is an extension base of K_0, then*

$$S' = S \cup \overline{\mathsf{Head}}(H_0 \setminus E_0), \quad \text{where } E_0 = \{ L \leftarrow \in H_0 \mid L \in S \} \tag{10}$$

is an H_0-maximal answer set of $T \cup H^*$. Moreover, every H_0-maximal answer set of $T \cup H^*$ can be represented in the form (10) *where S is an extension base of K_0.*

Proof: Suppose that S is an extension base of K_0. Then, S is an answer set of $T \cup E_0$ because $\mathsf{Head}(E_0) \subseteq S$. By Corollary 4.10, S' is a consistent answer set of $T \cup H^*$. Suppose to the contrary that S' is not an H_0-maximal answer set of $T \cup H^*$. Then, there is an answer set S'' of $T \cup H^*$ such that $\{ L \in S' \mid L \in \mathsf{Head}(H_0) \} \subset \{ L \in S'' \mid L \in \mathsf{Head}(H_0) \}$. Since $E_0 = \{ L \leftarrow \in H_0 \mid L \in S \} = \{ L \leftarrow \in H_0 \mid L \in S' \}$, $E_0 \subset \{ L \leftarrow \in H_0 \mid L \in S'' \}$ holds. This contradicts the maximality of E_0 in 2^{H_0}. Hence, S' is an H_0-maximal answer set of $T \cup H^*$.

Now, we prove the second claim. Suppose that S' is an H_0-maximal answer set of $T \cup H^*$. By Corollary 4.10, S' can be represented by $S' = S \cup \overline{\mathsf{Head}}(H_0 \setminus E_0)$, where S is an answer set of $T \cup E_0$ and $E_0 = \{ L \leftarrow \in H_0 \mid L \in S' \} = \{ L \leftarrow \in H_0 \mid L \in S \}$. Suppose to the contrary that S is not an extension base of K_0. Then, there is a set F ($E_0 \subset F \subseteq H_0$) such that $T \cup F$ is consistent. Let R be an answer set of $T \cup F$. By Corollary 4.10, $R' = R \cup \overline{\mathsf{Head}}(H \setminus F)$ is an answer set of $T \cup H^*$. By $E_0 \subset F$, clearly $\mathsf{Head}(E_0) \subset \mathsf{Head}(F) \subseteq R$. Therefore, $\{ L \in S' \mid L \in \mathsf{Head}(H_0) \} \subset \{ L \in R \mid L \in \mathsf{Head}(H_0) \} \subseteq \{ L \in R' \mid L \in \mathsf{Head}(H_0) \}$. This contradicts the H_0-maximality of S'. □

Example 4.12 Let us consider the knowledge system $K_0 = (T, H_0)$, which is the same as Example 4.5 and the translated set of assumptions $H^* = H \cup \overline{H}$:

$$T = \{ \ \neg P \leftarrow not\ P, \ C \leftarrow P, Q, \ \neg C \leftarrow \ \}, \quad H_0 = \{ \ P \leftarrow, \ Q \leftarrow \ \},$$

$$H = \{ \ P \leftarrow not\ \neg P, \ Q \leftarrow not\ \neg Q \ \}, \quad \overline{H} = \{ \ \neg P \leftarrow not\ P, \ \neg Q \leftarrow not\ Q \ \}.$$

There are three answer sets of $K^* = T \cup H^*$: $S_1' = \{ P, \neg Q, \neg C \}$, $S_2' = \{ \neg P, Q, \neg C \}$, and $S_3' = \{ \neg P, \neg Q, \neg C \}$. Of these, S_1' and S_2' are two H_0-maximal answer sets of K^*, and they correspond to the expansions of the two extension bases of K_0: $S_1 = \{ P, \neg C \}$ and $S_2 = \{ \neg P, Q, \neg C \}$. Note that S_3' is the expansion of the answer set of T: $S_3 = \{ \neg P, \neg C \}$.

4.2 Complex Default Assumptions

In the last subsection, we considered a knowledge system $K = (T, H)$ where H is restricted to being either a set of clauses of the form (4) or a set of clauses of the form (5). Moreover, we considered only the case where a set of clauses T satisfies the condition (6). In this subsection, we remove all of these restrictions: we allow any extended logic program for both T and H.

Example 4.13 Let us firstly consider the case in which T does not satisfy the condition (6) for $K = (T, H)$ where H is a set of assumptions of the form (5). Suppose that K is the following knowledge system:

$$\begin{array}{rl} T = \{ & Q \leftarrow P\,, \\ & Q \leftarrow \neg P\,, \\ & \neg Q \leftarrow \quad \}\,, \\ H = \{ & \neg P \leftarrow not\ P \ \}\,. \end{array}$$

K does not satisfy the condition (6) because P appears in the body of the first clause of T. It is easy to see that K has the unique extension base: $S = \{ \neg Q \}$, which is an answer set of T. However, when we introduce the opposite assumption, $\overline{H} = \{ P \leftarrow not\ \neg P \}$, we see that the program $T \cup H^* = T \cup H \cup \overline{H}$ is incoherent. Thus Theorem 4.8 cannot be used in this case. This is because neither P nor $\neg P$ can be consistently added to T but introducing H^* forces an answer set to include either of them by Lemma 4.7.

We shall translate a knowledge system K to an extended logic program K^*. The basic idea is "naming defaults" and is similar to Poole [20]. After the translation, we can utilize the results for literal assumptions presented in the last subsection.

Let $K = (T, H)$ be any knowledge system. For each clause $C \in H$ of the form (1), we shall associate a propositional symbol δ_C which is not appearing elsewhere in K [7]. For any subset E of H, we define the following sets of clauses:

$$\begin{array}{rcl} \Delta_0(E) & = & \{ \delta_C \leftarrow \ \mid C \in E \}\,, \\ \Delta(E) & = & \{ \delta_C \leftarrow not\ \neg\delta_C \mid C \in E \}\,, \\ \overline{\Delta}(E) & = & \{ \neg\delta_C \leftarrow not\ \delta_C \mid C \in E \}\,, \ \text{and} \\ \Gamma(E) & = & \{ L_0 \leftarrow \delta_C, L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n \mid \\ & & \qquad C = (L_0 \leftarrow L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n) \in E \}\,. \end{array}$$

[7] If an assumption C contains n distinct free variables $\mathbf{x} = x_1, \ldots, x_n$, then we can name C with $\delta_C(\mathbf{x})$ where δ_C is an n-ary predicate symbol appearing nowhere in K. Note that every variable appearing in a clause is a free variable in our language.

For $K = (T, H)$, we define the extended logic program K^* as:

$$K^* = T \cup \Gamma(H) \cup \Delta(H) \cup \overline{\Delta}(H) .$$

Before analyzing the program K^*, let us first consider a knowledge system

$$K_0 = (T \cup \Gamma(H), \Delta_0(H)) .$$

This knowledge system has only atomic assumptions and satisfies the condition (6) because no $\delta_C \in \mathsf{Head}(\Delta_0(H))$ appears in any clause other than in the body of one clause in $\Gamma(H)$ [8]. Therefore, we can apply Corollary 4.10 and Theorem 4.11 for K_0.

The basic property of the translation is shown by the next theorem.

Theorem 4.14 *Let $K = (T, H)$ be any knowledge system, and E be a subset of H such that $T \cup E$ is consistent. S is an answer set of $T \cup E$ if and only if*

$$S' = S \cup \mathsf{Head}(\Delta_0(E))$$

is a consistent answer set of $T \cup \Gamma(E) \cup \Delta_0(E)$.

Proof: Suppose that S is an answer set of $T \cup E$. Then S' is obviously consistent. It is easy to see that the knowledge system $(T \cup \Gamma(E), \Delta_0(E))$ satisfies the condition (6). Therefore, $T^{S'} = T^S$ because S' does not contain any new literal other than the names of assumptions of E. Similarly, $\Gamma(E)^{S'} = \{ L_0 \leftarrow \delta_C, L_1, \ldots, L_m \mid C = (L_0 \leftarrow L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n) \in E\ , L_{m+1}, \ldots, L_n \notin S' \} = \Gamma(E)^S$. And $\Delta_0(E)^{S'} = \Delta_0(E) = \{ \delta_C \leftarrow \mid C \in E \}$ holds.

Now,

$$\begin{aligned}
& \alpha((T \cup \Gamma(E) \cup \Delta_0(E))^{S'}) \\
= \ & \alpha(T^S \cup \Gamma(E)^S \cup \Delta_0(E)) \\
= \ & \alpha(T^S \cup \{ \delta_C \leftarrow \mid C \in E \} \\
& \qquad \cup \{ L_0 \leftarrow L_1, \ldots, L_m \mid L_0 \leftarrow L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n \in E\ , \\
& \qquad\qquad L_{m+1}, \ldots, L_n \notin S' \}) \\
& \quad \text{(by unfolding the clauses of } \Gamma(E)^S \text{ by } \Delta_0(E)) \\
= \ & \alpha(T^S \cup E^S \cup \Delta_0(E)) \\
= \ & \alpha((T \cup E)^S) \cup \mathsf{Head}(\Delta_0(E)) \\
= \ & S \cup \mathsf{Head}(\Delta_0(E)) \quad (\text{by } S \cap \mathsf{Head}(\Delta_0(E)) = \emptyset) \\
= \ & S' .
\end{aligned}$$

[8] We can allow T to include clauses containing $\delta_C \in \mathsf{Head}(\Delta_0(H))$ within the restriction (6) and use them for exceptions and cancellations, as in the previous subsection. Since these clauses are not necessary for our purpose, we do not pursue this possibility further in this subsection (see also Example 4.19).

Hence, S' is a consistent answer set of $T \cup \Gamma(E) \cup \Delta_0(E)$.

On the other hand, suppose that S' is a consistent answer set of $T \cup \Gamma(E) \cup \Delta_0(E)$. Since $S \cap \mathsf{Head}(\Delta_0(E)) = \emptyset$, we can immediately identify S from S'. By using the same translation as above, we see that

$$S' = \alpha(T^{S'} \cup \Gamma(E)^{S'} \cup \Delta_0(E)^{S'}) = \alpha(T^S \cup E^S \cup \Delta_0(E)) = \alpha((T \cup E)^S) \cup \mathsf{Head}(\Delta_0(E)).$$

Since $\alpha((T \cup E)^S) \cap \mathsf{Head}(\Delta_0(E)) = \emptyset$, $S = \alpha((T \cup E)^S)$ holds. Hence, S is an answer set of $T \cup E$. □

By combining Theorem 4.14 and Corollary 4.10, we get the following result. Every answer set of any consistent theory from $K = (T, H)$ can be characterized by an answer set of $K^* = T \cup \Gamma(H) \cup \Delta(H) \cup \overline{\Delta}(H)$, and vice versa.

Corollary 4.15 *Let $K = (T, H)$ be any knowledge system. If S is a consistent answer set of $T \cup E$ where E is a subset of H, then*

$$S' = S \cup \mathsf{Head}(\Delta_0(E)) \cup \overline{\mathsf{Head}}(\Delta_0(H \setminus E)) \tag{11}$$

is a consistent answer set of K^*. Moreover, every consistent answer set S' of K^* can be represented in the form (11) where S is a consistent answer set of $T \cup E$ for some set $E \subseteq H$.

Corollary 4.15 shows that for (T, H) if $T \cup E$ $(E \subseteq H)$ has a consistent answer set S then δ_C can be consistently added to S for every assumption C in E and the negated names of all other assumptions can be also added to S, and that we can find these answer sets of the consistent theories from (T, H) by removing all of positive and negative naming assumptions from the answer sets of K^*.

Finally, we can characterize the extension bases of $K = (T, H)$ by combining Theorem 4.11, Theorem 4.14 and Corollary 4.15.

Corollary 4.16 *Let $K = (T, H)$ be any knowledge system. If S is an extension base of K, that is, an answer set of $T \cup E$ for a maximal subset E of H such that $T \cup E$ is consistent, then*

$$S' = S \cup \mathsf{Head}(\Delta_0(E)) \cup \overline{\mathsf{Head}}(\Delta_0(H \setminus E))$$

is a $\Delta_0(H)$-maximal answer set of K^*.

Conversely, if S' is a $\Delta_0(H)$-maximal answer set of K^*, then

$$S = S' \setminus (\mathsf{Head}(\Delta_0(H)) \cup \overline{\mathsf{Head}}(\Delta_0(H)))$$

is an extension base of K.

Example 4.17 Let $K = (T, H)$ be the knowledge system introduced in Example 4.13:

$$\begin{array}{rl} T = \{ & Q \leftarrow P\,, \\ & Q \leftarrow \neg P\,, \\ & \neg Q \leftarrow \qquad \}\,, \\ H = \{ & \neg P \leftarrow not\ P \ \}\,. \end{array}$$

Now, we can name assumptions as

$$\begin{array}{rl} \Delta_0(H) = \{ & \delta_{\neg P \leftarrow not\ P} \leftarrow \qquad\qquad\quad \}\,, \text{ and} \\ \Gamma(H) = \{ & \neg P \leftarrow \delta_{\neg P \leftarrow not\ P},\ not\ P \ \}\,. \end{array}$$

Recall that K has the unique extension base: $S = \{\neg Q\}$. It is easy to check that $S' = S \cup \{\neg\delta_{\neg P \leftarrow not\ P}\}$ is the unique answer set of $K^* = T \cup \Gamma(H) \cup \Delta(H) \cup \overline{\Delta}(H)$.

Example 4.18 Let us see how an incoherent program Π gets consistent answer sets. We can construct a knowledge system $(\emptyset, \Pi)$ and apply the reduction techniques. For example, consider the knowledge system $K = (\emptyset, \Pi)$ where

$$\Pi = \{\ P \leftarrow not\ P\ \}\,.$$

In this case, $S = \emptyset$ is the unique extension base of K. Now,

$$\begin{array}{rl} \Delta_0(\Pi) = \{ & \delta_{P \leftarrow not\ P} \leftarrow \qquad\quad \} \\ \Gamma(\Pi) = \{ & P \leftarrow \delta_{P \leftarrow not\ P},\ not\ P \ \} \end{array}$$

The unique answer set of K^* is

$$S' = S \cup \{\neg\delta_{P \leftarrow not\ P}\}\,.$$

Example 4.19 Consider the knowledge system $K = (T, H)$ introduced in Example 3.6:

$$\begin{array}{rl} T = \{ & \neg Flies(x) \leftarrow Penguin(x)\,, \\ & Bird(x) \leftarrow Penguin(x)\,, \\ & Bird(Polly) \leftarrow, \\ & Penguin(Tweety) \leftarrow \qquad \}\,, \\ H = \{ & Flies(x) \leftarrow Bird(x) \qquad \}\,. \end{array}$$

In this case, we can name defaults as

$$\begin{array}{rl} \Delta_0(H) = \{ & Birdflies(x) \leftarrow \qquad\qquad\qquad\quad \}\,, \text{ and} \\ \Gamma(H) = \{ & Flies(x) \leftarrow Birdflies(x),\ Bird(x) \ \}\,. \end{array}$$

Then, we see that there is the unique $\Delta_0(H)$-maximal answer set of K^*:

$$S' = \{\, Bird(Polly),\ Penguin(Tweety),\ Bird(Tweety),\\ Flies(Polly),\ Birdflies(Polly),\ \neg Flies(Tweety),\ \neg Birdflies(Tweety)\,\}.$$

Removing all the naming literals from S', we get the unique extension base S of K:

$$S = \{\, Bird(Polly),\ Flies(Polly),\ Penguin(Tweety),\ Bird(Tweety),\ \neg Flies(Tweety)\,\}.$$

The difference between Poole's system and ours with respect to the naming is that the naming in [20] has the effects of introducing normal defaults, for example,

$$\frac{:\ \mathsf{M}\, Bird(x) \supset Flies(x)}{Bird(x) \supset Flies(x)}.$$

where $\supset$ is classical implication. This causes two side effects: (1) from $\neg Flies(Sam)$ we can conclude $\neg Bird(Sam)$ (this should not be allowed because we do not know the reason for *Sam*'s inability to fly; *Sam* might be a penguin), and (2) from the assumption $Birdflies(Paul)$ and the contrapositive of that fact $\neg Flies(x) \supset \neg Birdflies(x)$ we can conclude $Flies(Paul)$. To prevent the first inference, we must add a fact like $\neg Flies(x) \supset \neg Birdflies(x)$. To prevent the second inference, we must use this fact as a constraint. In our case, both kinds of pruning rules are unnecessary.

5 Discussion

In this section, we compare the proposed framework to other hypothetical reasoning systems based on logic programming. Our framework makes it possible to deal with incomplete knowledge and to remove inconsistencies, so that comparisons should be made from those viewpoints.

5.1 Reduction to General Logic Programs

The first question is how to compute the proposed framework for theory formation. Since we have seen that every knowledge system can be transformed to a single extended logic program, we can use methods to compute answer sets of extended logic programs [9]. For this purpose, Gelfond and Lifschitz [8] show how to reduce an extended logic program to a general

[9] Alternative methods to compute the framework for theory formation can be conceived. Since we have seen that every clause in H of a knowledge system $K = (T, H)$ can be transformed to the unique naming assumption, we can use *nonmonotonic ATMSs* [4, 14] to compute explanations of each atom.

logic program. The method is to replace every classical negation with a new propositional symbol, for example, $\neg A$ is replaced by A'. However, even if the original extended logic program is incoherent, such a reduced program may have stable models.

Example 5.1 Let Π be an extended logic program shown in the example of the CWA in Section 1, and Π^+ be the corresponding general logic program obtained by the reduction in [8]:

$$\Pi = \{ \quad Q \leftarrow \neg P(A), \neg P(B), \qquad\qquad \Pi^+ = \{ \quad Q \leftarrow P(A)', P(B)',$$
$$\neg Q \leftarrow, \qquad\qquad\qquad\qquad\qquad\qquad Q' \leftarrow,$$
$$\neg P(x) \leftarrow not\ P(x) \qquad \}, \qquad\qquad\qquad\qquad P(x)' \leftarrow not\ P(x) \qquad \}.$$

While Π is an incoherent program, the translated program Π^+ has an inconsistent stable model: $M = \{ P(A)', P(B)', Q, Q' \}$.

Note that not every incoherent program may be translated to a general logic program that has inconsistent stable models (for example, $\Pi = \{ P \leftarrow not\ P \}$). Conversely, not every translated general program that has inconsistent stable models may correspond to an incoherent extended logic program (for example, $\Pi^+ = \{ P \leftarrow, \quad P' \leftarrow \}$). We have classified inconsistent extended logic programs into two types: contradictory programs and incoherent programs. These inconsistent programs may be transformed to general logic program which have either inconsistent stable models or no stable model. In either case, we cannot accept programs, since we would like to get consistent programs by theory formation, that is, programs whose translated programs have consistent stable models. Thus, we can prune all inconsistent stable models regardless of the status of the original extended programs.

By the above argument, we need a mechanism to check whether the resulting stable models have a pair of complementary propositions, say A and A'. If a stable model possesses a pair then we discard it. There are some proposals to prune these undesired models and most of them represent pruning rules as *integrity constraints*. For example, for each atom A such that both positive and negative literals appear in the program, we may add an integrity constraint:

$$\leftarrow A, A'. \tag{12}$$

These constraints have to be considered at the *implementation level*. Eshgi and Kowalski [6], Kakas and Mancarella [15], and the TMS-based system by Giordano and Martelli [11] can handle integrity constraints, but all of them allow more general integrity constraints than simple constraints of the form (12), yet none of them considers classical negation. Although

this kind of expression is not allowed in our system, for an integrity constraint of the form

$$\leftarrow L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n\ ^{10}, \tag{13}$$

we can represent an equivalent set of clauses by introducing a new proposition C as

$$\begin{array}{l} C \leftarrow L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n\ , \\ \neg C \leftarrow\ . \end{array} \tag{14}$$

Thus all we have to deal with are clauses of the form (12) [11]. Neither a general checking mechanism like [15] nor a generator of a new clause to remove inconsistencies [11] is necessary. Moreover, at the *representation level*, it is more convenient to use classical negation because assumptions and exceptions are dealt with naturally by using classical negation, as discussed in Section 2. In fact, Kowalski and Sadri [16] do not use integrity constraints at the representation level but use classical negation in a restricted way.

5.2 Abductive Logic Programming

There are some proposals for abduction by using logic programming.

Eshgi and Kowalski [6] use a backward-chaining procedure to compute stable models of general logic programs, but they do not consider assumptions other than formulas representing negation as failure.

Gelfond [10] and Kakas and Mancarella [15] propose abductive frameworks for logic programs. The most significant difference is that ours allows any extended logic programs as assumptions but both [10] and [15] consider only assumptions of the form of literal assertions (4). For these simple forms of assumptions, our framework is in essence equivalent to

[10] Integrity constraints of the form (13) roughly correspond to quantifier-free formulas with a modal operator K (in the sense of Reiter [26]) of the form

$$\neg K\ L_1 \vee \ldots \vee \neg K\ L_m \vee K\ L_{m+1} \vee \ldots \vee K\ L_n\ .$$

[11] Elkan [5] shows another method to eliminate integrity constraints within the framework of general logic programs. He translates a constraint of the form (13), where L_i $(1 \leq i \leq n)$ is an atom, to the following clauses:

$$\begin{array}{l} C \leftarrow L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n\ , \\ C_1 \leftarrow not\ C\ , \\ C_1 \leftarrow not\ C_2\ , \\ C_2 \leftarrow C_1\ , \end{array}$$

where C, C_1 and C_2 are propositions not mentioned in the original program. However, incorporating classical negation allows us to represent it in a more concise form (14).

them [12]. Kakas and Mancarella [15] deal only with general logic programs with integrity constraints, which are special cases of our framework, as a background theory, and assumptions are only atomic assertions. Gelfond [10] allows a background theory to be an extended *disjunctive* program, whose semantics is given in [9]. It is possible to extend our framework by allowing such programs for both background theories and assumptions according to the semantics.

Other big difference is that their systems [10, 15] consider only abduction as an application and cannot be applied to default reasoning. As explained in Section 3.3, the fact that a formula has an explanation does not imply that the formula is true in an extension of a knowledge system. Thus, when a set of assumptions represents default knowledge, it is not suitable for commonsense reasoning to find only explanations.

5.3 Inconsistency Resolution

The proposed method for inconsistency resolution can be applied to much broader classes of default knowledge than Kowalski and Sadri's system [16], which handles only a simple form of *exceptions*. For the simple form of exceptions, the methods are quite different (for instance, see Example 3.6 and Example 4.19). The limitation of [16] is that clauses are automatically divided into two (those having positive literals as heads and those having negative literals as heads) so that negative literals are always exceptions of the positive ones with the same predicates [13]. Therefore, the techniques proposed in this paper are more flexible than those of [16]. Moreover, our methods are not restricted to dealing only with exceptions. For instance, both Example 3.7 and Example 3.9 cannot be dealt with by [16]. Our framework has also much richer expressive power than [16] because any extended logic program can be a set of assumptions.

The definition of answer sets of extended logic programs appears to fit bottom-up (or forward-reasoning) procedures for its computation more than using top-down (or backward-chaining) procedures. In general, it is more appropriate to use top-down procedures for abduction, but when we want to deal with default reasoning, simply computing explanations may not be appropriate. In this sense, we can compare our method to the TMS-style computation. According to Elkan [5], a set of justifications for Doyle's TMS [3] can be identified with a general logic program with integrity constraints, and the TMS computes a consistent stable model of the program by a bottom-up manner. Classical negation is not incorporated in the TMS.

[12] Note that the definitions of *explanations* are different between [10] and [15]. See Section 3.3.

[13] Kowalski and Sadri, however, claim that their techniques can be extended to deal with exceptions with individual clauses rather than entire clauses and with exceptions having positive conclusions. But if we allow these mixed exceptions at the same time, then we have to take care of the semantics for each exception individually because they change the original answer set semantics of [8].

However, inconsistency resolution by our framework is different from the TMS-style contradiction resolution [3, 11, 5]. When a contradiction occurs, the TMS imposes a new clause in order to believe a literal that has not been believed. As a result of contradiction resolution, the TMS may fail to output a stable model of the original program. Elkan [5] claims that when the TMS finds an inconsistent stable model, it should choose another stable model if the program has one. However, such a strategy is not tolerant of incoherent programs because if the program has no consistent stable model then it does not output anything. On the other hand, Giordano and Martelli [11] consider all possible models which the TMS may output by contradiction resolution (called *dependency-directed backtracking*). Although their method reflects an incremental use of the TMS, its model theory is no longer stable model semantics in the sense that contrapositives of original clauses are interpreted to be valid and that literals interpreted to be false by negation as failure in the original program can be believed through those contrapositive clauses. This kind of semantics may throw us into confusion at the representation level. We consider that this confusion comes from the fact that the TMS does not deal with retractable assumptions.

Because our system represents assumptions explicitly, assumptions alone are invalidated; other clauses are not affected. In this sense, the proposed framework can be considered as a generalization of nonmonotonic ATMSs [4, 14], which deal with general logic program with integrity constraints and atomic assumptions.

Besides the TMSs, there are some proposals for contradiction resolution in nonmonotonic reasoning [19, 12]. To resolve incoherencies in autoepistemic logic, Morris [19] proposes *stable closures* when there is no stable expansion. His proposal is motivated by dependency-directed backtracking in the TMS and therefore some formulas are believed to remove inconsistencies. Again, we do not add any new formulas but remove a minimal set of hypotheses for default reasoning. On the other hand, Guerreiro, Casanova and Hemerly [12] propose an alternative definition for default logic extensions. Although their definitions are quite different from ours, their idea is similar because defaults are allowed to be ignored in their extensions to keep consistency but no default rule can be dispensed with unless it is necessary to do so. We consider such defeasible defaults for some distinguished clauses rather than entire defaults.

5.4 Priority

The last question is how to divide theories into the factual or background theories and default assumptions. As every clause in extended logic programs can be identified with Reiter's default rules, we have to classify two types of rules for a knowledge system. One easy way is, for a knowledge system $K = (T, H)$, to associate *integrity constraints* in the sense of

Reiter [26] with T and other theories with H [14]. Then, integrity constraints must be satisfied by every extension base and all other clauses can be ignored as minimally as possible.

A more natural and widely acceptable view of knowledge systems is to divide the program into subprograms (*categories*) in accordance with the degrees of credibility of defaults, where the priority is determined depending on the problem domain. This view of hypothetical reasoning is exactly the same idea as Rescher [27]. There may be more than two categories for a problem. If these categories can be totally ordered, then we can have an extended knowledge system like $K = (H_0, H_1, \ldots, H_n)$. An extended framework for hypothetical reasoning based on first-order logic is considered by Brewka [2] as an extension of Poole's framework [20]. It may be possible to extend our framework in the same way as [2].

6 Conclusion

We expanded the idea of Gelfond and Lifschitz and presented methods to deal with broader classes of commonsense knowledge. Like Poole's framework, default knowledge H is dealt with as a part of knowledge distinct from a theory T about the world, and defaults are used to augment the theory and to predict what we expect to be true.

One of the main tasks of a knowledge system is to find a maximal (with respect to set inclusion) subset E of H such that there is a consistent answer set of the extended logic program $T \cup E$. If adding assumptions causes inconsistencies, then a minimal set of assumptions can be ignored to remove inconsistencies. This framework can also be used for abduction. Compared with Poole's system which uses the first-order predicate calculus, abduction and default reasoning cannot be related elegantly in our framework, but some commonsense knowledge may be represented more easily.

We also proposed the translation of a knowledge system K to an extended logic program K^* so that each answer set of K^* corresponds to an answer set of a consistent theory from K.

The proposed framework can handle any extended logic program as a set of assumptions. Therefore, the presented methods of naming defaults and inconsistency resolution may also contribute to giving the basis of generalizations of (nonmonotonic) ATMSs.

[14] Note that Reiter considers a database as a set of first-order sentences and defines integrity constraints as a set of epistemic formulas [26]. In our case, both databases and integrity constraints can be any extended logic programs that are identified with default rules. This is an extension of databases to knowledge bases, whose semantics can be partially given by using Levesque's autoepistemic logic [17]. The exact epistemic semantics of default logic and extended logic programs are considered in [13].

Acknowledgment

I am grateful to Vladimir Lifschitz, Ray Reiter, Chiaki Sakama, Ken Satoh and Koichi Furukawa for helpful discussions on this work.

References

[1] G. Brewka, Nonmonotonic reasoning: From theoretical foundations towards efficient computation, PhD Dissertation, University of Hamburg, 1989.

[2] G. Brewka, Preferred subtheories: An extended logical framework for default reasoning, *Proc. of IJCAI-89* (1989) 1043–1048.

[3] J. Doyle, A truth maintenance system, *Artificial Intelligence* **12** (1979) 231–272.

[4] O. Dressler, An extended basic ATMS, *Proc. of the Second International Workshop on Non-Monotonic Reasoning*, Lecture Notes in Artificial Intelligence 346, Springer-Verlag (1989) 143–163.

[5] C. Elkan, A rational reconstruction of nonmonotonic truth maintenance systems, *Artificial Intelligence* **43** (1990) 219–234.

[6] K. Eshghi and R. Kowalski, Abduction compared with negation by failure, *Proc. of the Sixth International Conference on Logic Programming* (1989) 234–254.

[7] M. Gelfond and V. Lifschitz, The stable model semantics for logic programming, *Proc. of the Fifth International Conference and Symposium on Logic Programming* (1988) 1070–1080.

[8] M. Gelfond and V. Lifschitz, Logic programs with classical negation, *Proc. of the Seventh International Conference on Logic Programming* (1990) 579–597.

[9] M. Gelfond and V. Lifschitz, Classical negation in logic programs and disjunctive databases, submitted for publication, 1990.

[10] M. Gelfond, Epistemic approach to formalization of commonsense reasoning, manuscript, Computer Science Department, University of Texas at El Paso, 1990.

[11] L. Giordano and A. Martelli, Generalized stable models, truth maintenance and conflict resolution, *Proc. of the Seventh International Conference on Logic Programming* (1990) 427–441.

[12] R.A. Guerreiro, M.A. Casanova and A.S. Hemerly, Contributions to a proof theory for generic defaults, *Proc. of ECAI-90* (1990) 213–218.

[13] K. Inoue, Epistemic semantics for default theories and logic programs, in preparation, 1991.

[14] U. Junker, A correct non-monotonic ATMS, *Proc. of IJCAI-89* (1989) 1049–1054.

[15] A.C. Kakas and P. Mancarella, Generalized stable models: A semantics for abduction, *Proc. of ECAI-90* (1990) 385–391.

[16] R. Kowalski and F. Sadri, Logic programs with exceptions, *Proc. of the Seventh International Conference on Logic Programming* (1990) 598–613.

[17] H.J. Levesque, All I know: A study of autoepistemic logic, *Artificial Intelligence* **42** (1990) 263–309.

[18] J. Minker, On indefinite databases and the closed world assumption, *Proc. of the Sixth International Conference on Automated Deduction*, Lecture Notes in Computer Science 138, Springer-Verlag (1982) 292–308.

[19] P.H. Morris, Autoepistemic stable closures and contradiction resolution, *Proc. of the Second International Workshop on Non-Monotonic Reasoning*, Lecture Notes in Artificial Intelligence 346, Springer-Verlag (1989) 60–73.

[20] D. Poole, A logical framework for default reasoning, *Artificial Intelligence* **36** (1988) 27–47.

[21] D. Poole, R. Goebel and R. Aleliunas, Theorist: A logical reasoning system for defaults and diagnosis, In: N. Cercone and G. McCalla (eds.), *The Knowledge Frontier: Essays in the Representation of Knowledge*. Springer-Verlag, New York (1987) 331–352.

[22] T. Przymusinski, Well-founded semantics coincides with three-valued stable semantics, manuscript, Department of Mathematical Sciences, The University of Texas at El Paso, 1990.

[23] T. Przymusinski, Stationary semantics for disjunctive logic programs and deductive databases, *Proc. of the North American Conference on Logic Programming* (1990) 40–59.

[24] R. Reiter, On closed world data bases, in: H. Gallaire and J. Minker (Eds.), *Logic and Data Bases* (Plenum Press, New York, 1978) 55–76.

[25] R. Reiter, A logic for default reasoning, *Artificial Intelligence* **13** (1980) 81–132.

[26] R. Reiter, What should a database know?, Technical Reports on Knowledge Representation and Reasoning KRR-TR-90-5, Department of Computer Science, University of Toronto, 1990.

[27] N. Rescher, *Hypothetical Reasoning* (North-Holland, Amsterdam, 1964).