Programming in Autoepistemic Logic

Kienchung Kuo
Department of Computer Science
SUNY at Stony Brook
Stony Brook, NY 11794
kuokc@cs.sunysb.edu

Abstract

We propose a language for programming in autoepistemic logic that extends the standard logic programming and incorporates incomplete information. By syntactically distinguishing the "true negation" from the "lack of information," we also provide a way to define negative information explicitly. A fixpoint semantics can be defined for stratified and conservative programs. In this paper, we investigate definite autoepistemic programs. We investigate fixpoints of definite autoepistemic programs and show that they coincide with the declarative semantics of these programs. We also define a resolution procedure, called SLSAE-resolution, for such programs. SLSAE-resolution is sound and complete for stratified, conservative, and solvable programs.

1 Introduction

There are two serious limitations of traditional logic programming:

1. Negative information is represented indirectly through the absence of positive data, which may be undesirable in situations where a direct representation is called for. For example, given a rule $fly(x) \leftarrow bird(x)$ and a fact *bird(tweety),* we shall be able to infer *fly(tweety).* However, not every bird can fly. Suppose we know $\neg fly(tweety)$, how can we add this fact to the database and keep it consistent?
2. There is no direct way to deal with incomplete information. The following example from [Gelfond and Lifschitz, 1990] illustrates this:

   $Eligible(x) \leftarrow HighGPA(x);$
   $Eligible(x) \leftarrow Minority(x), FairGPA(x);$
   $\neg Eligible(x) \leftarrow \neg FairGPA(x);$
   $Interview(x) \leftarrow \mathbf{not}\, Eligible(x), \mathbf{not}\, \neg Eligible(x).$

   The first rule says that students with high GPA are eligible for a scholarship; the second rule says a student is eligible if he is a minority and has fair GPA; the third rule says that a student with GPA lower than fairness is not eligible; the fourth rule says that students whose eligibility is not determined by the above rules should be interviewed by the scholarship committee. Obviously, not/ in the above example represents that / is not known, which is different from the negation of /.

In this paper, we present a framework for programming in autoepistemic logic that incorporates unknown information and explicit definition of negative information. We show that for certain programs, consistency is guaranteed and the fixpoint and declarative semantics coincide. There are two major differences between this paper and [Gelfond and Lifschitz, 1990]:

1. We provide a syntactic restriction to guarantee the consistency of a program.
2. Our base language is first-order, and we provide a resolution procedure to compute the answer set for restricted programs.

The structure of the paper is as follows: Section 2 gives a brief introduction to autoepistemic logic, its syntax, semantics, variations, and extensions. Section 3 introduces autoepistemic programs. Sufficient conditions are given for the existence of a unique, consistent iterative expansion. A resolution procedure, SLSAE-resolution, is defined in Section 4. Section 5 presents conclusion and future work.

2 Autoepistemic Logic

Autoepistemic logic is defined by Moore [Moore, 1985]. Later Konolige [Konolige, 1988] gave another definition of expansions in autoepistemic logic and showed that it is equivalent to Moore's. The language of autoepistemic logic, call it $\mathcal{L}$, is constructed out of a *base* sublanguage $\mathcal{L}_0$ (e.g. propositional or first order), augmented with a modal operator K. Every sentence in $\mathcal{L}_0$ is a sentence in $\mathcal{L}$, if ϕ is a sentence in $\mathcal{L}$, then so is $K\phi$. If ϕ, ψ are sentences in $\mathcal{L}$, then so are $\neg\phi$, $\phi \wedge \psi$, $\phi \vee \psi$, $\phi \rightarrow \psi$. Note that autoepistemic logic itself is propositional. A sentence has K-*rank n* if its modal operators are nested to a depth of n. Formally, *objective* formulas, i.e., formulas in $\mathcal{L}_0$, have K-rank 0, K-rank($\neg\phi$) = K-rank(ϕ), K-rank($\phi \wedge \psi$) = max(K-rank(ϕ), K-rank(ψ)), K-rank($\phi \vee \psi$) = max(K-rank(ϕ), K-rank(ψ)), and K-rank(Kϕ) = K-rank(ϕ)+1. For example, K($A \wedge$ KA) has

K-rank 2. We use $\mathcal{L}_n$ to denote the set of all sentences of K-rank n or less. A *theory* is a set of sentences. We will use a subscript to indicate a subset of sentences based on the K-rank. For example, if T is a theory, $T_n = T \cap \mathcal{L}_n$. We call T_0 the *objective* part of T.

In the following discussion, let $\models$ denote the usual entailment relation for classic logic, and Cn be the corresponding consequence operator. Semantically, an *autoepistemic interpretation* is a pair $< I, T >$, where I is an interpretation for $\mathcal{L}_0$[1] and T is a set of formulas. *Satisfaction* by an autoepistemic interpretation $< I, T >$ is defined as follows:

1. For every objective sentence ϕ, $\models_{I,T} \phi$ iff $I \models \phi$.
2. For every $\phi \in \mathcal{L}$, $\models_{I,T} \mathsf{K}\phi$ iff $\phi \in T$.
3. Satisfaction of compound sentences in $\mathcal{L}$ is defined as usual, i.e., by structural induction.

Let F be a set of formulas, we define $F \models_{I,T} \phi$ iff $\models_{I,T} F$ implies $\models_{I,T} \phi$; also define $F \models_T \phi$ iff $F \models_{I,T} \phi$ for all interpretations I.

Now, let $S_F = \{\phi : F \models_T \phi\}$. If F is a set of initial assumptions and T is a set of sentences possessed by a fully introspective reasoner as knowledge, then S_F should coincide with T. This argument is originally from [Moore, 1985], and is further explained in [Konolige, 1988; Marek and Truszczynski, 1988]. We will call F the *base set* of knowledge possessed by the reasoner.

Definition 1 An *expansion* of a base set F is a set T satisfying the following equation:

$$T = \{\phi : F \models_T \phi\}. \qquad \square$$

A base set F may have no expansions and even if it has an expansion, the expansion may not be unique. However, if there is a unique expansion T for a given base set F, then $F \models_T \phi$ can be written as $F \models \phi$ without ambiguity. When both F and ϕ are objective formulas, this notion of implication coincides with the classic one.

2.1 Strong Autoepistemic Logic

Unfortunately, expansions may not be "well grounded" in the initial set of axioms. For example, the axiom $\{\mathsf{K}\phi \to \phi\}$, has an expansion $\{\phi, \mathsf{K}\phi, \ldots\}$. The reason the spurious sentence ϕ is in this expansion is because in the very beginning we may assume that it is, hence $\mathsf{K}\phi$ is true, and ϕ can be derived. Clearly such an expansion is not the intended one. Marek and Truszczynski [Marek and Truszczynski, 1989] introduced a variant of the notion of expansion. Their approach requires that a formula ϕ must be first derived if it is used to prove $\mathsf{K}\phi$. They defined an operator A such that:

$$A(S) = Cn(S \cup \{\mathsf{K}\phi : \phi \in S\}).$$

Let us further define

$$A_0^T(F) = Cn(F \cup \{\neg\mathsf{K}\phi : \phi \notin T\}),$$

$$A_{n+1}^T(F) = A(A_n^T(F)) = Cn(A_n^T(F) \cup \{\mathsf{K}\phi : \phi \in A_n^T(F)\}),$$

[1]If the base language $\mathcal{L}_0$ is propositional, then I is simply a truth assignment.

and

$$A^T(F) = \bigcup_{n=0}^{\infty} A_n^T(F).$$

Then, T is an *iterative expansion* [Marek and Truszczynski, 1989] of F if

$$T = A^T(F).$$

Note that $A^T(F)$ is exactly the set of all consequences of $F \cup \{\neg\mathsf{K}\phi : \phi \notin T\}$ in the logic N that uses modus ponens and necessitation (from ϕ infer $\mathsf{K}\phi$) as inference rules. That is,

$$A^T(F) = Cn_N(F \cup \{\neg\mathsf{K}\phi : \phi \notin T\}).$$

Marek and Truszczynski called the nonmonotonic logic based on the above logic N *strong autoepistemic logic*. Every iterative expansion is an expansion [Marek and Truszczynski, 1989], but not vice versa. For example: $\{\mathsf{K}p \to p\}$ has an expansion $\mathcal{E}(p)$ that is not iterative: adding formulas $\neg\mathsf{K}\phi$ for ϕ not in $\mathcal{E}(p)$ does not help to derive p.

2.2 An Extension

Autoepistemic logic introduced above is propositional. Limited extensions to first order logic were studied in [Konolige, 1989; Marek, 1989]. Marek [Marek, 1989] formulated the *first order stability* conditions for a set of formulas T as follows:

1. T is closed under first order provability.
2. If $\phi \in T$, then $\mathsf{K}\phi \in T$.
3. If $\phi \notin T$, then $\neg\mathsf{K}\phi \in T$.
4. $\forall x\phi \in T \Leftrightarrow$ For all $a \in D, \phi(\mathbf{a}) \in T$, where D is a fixed domain and $\mathbf{a}$ is the name of object a.
5. $\exists x\phi \in T \Leftrightarrow$ There exists $a \in D$ such that $\phi(\mathbf{a}) \in T$.

Theories satisfying conditions 1-5 are called *stable*. A nice result was obtained in [Marek, 1989] for stable theories: If T is stable, then T allows the commutativity of quantifiers and operator K. However, the stability condition is too strong. Usually, we expect $\exists s\mathsf{K}\phi(x) \Rightarrow \mathsf{K}\exists x\phi(x)$ to be true, i.e., if there is some c such that we know $\phi(c)$, then we should know that $\exists x\phi(x)$. The converse of this implication needs not hold, i.e., we may know that there is an x such that $\phi(x)$ is true without knowing exactly which constant x represents.

In this paper we will assume that every element in the domain has a name. This means that the Barcan formula $(\forall x(\mathsf{K}p(x)) \to \mathsf{K}(\forall xp(x)))$ and its converse $(\mathsf{K}(\forall xp(x)) \to \forall x(\mathsf{K}p(x)))$ hold. This restriction raises the "universal query problem [Przymusinski, 1989b]." That is, if the program P contains a single clause $p(a)$ then $\forall xp(x)$ is true. We adopt the solution suggested in [Gelder *et al.*, 1988; Ross, 1989a] and assume that there is an extra symbol, \$, which is never used in the program. The symbol \$ is regarded as a "generic name" for those elements in the domain that have no representation in the language alphabet. In other words, $\forall xp(x)$ is not true unless it is stated in the program explicitly.

3 Definite Autoepistemic Programs

In this section we define definite autoepistemic programs. The syntax is introduced in section 3.1. Two syntactic constraints, namely, stratification and conservativeness, are formally defined. Section 3.2 presents the fixpoint semantics for stratified and conservative programs. It is shown that the fixpoint has a one-to-one correspondence relationship to the unique iterative expansion of such programs.

3.1 Syntax

A *definite autoepistemic program* P is a set of rules of the form:

$$l \leftarrow L_1, \ldots, L_n,$$

where (1) l is a nonequality objective literal, and (2) each $L_i, 1 \leq i \leq n$, is a literal of the form $\mathsf{K}l'$ or $\neg\mathsf{K}l'$, where l' is an objective literal. For simplicity we also assume the following Clark's Equational Theory [Kunen, 1987]:[2]

1. $X = X$
2. $(Y = X) \leftarrow \mathsf{K}(X = Y)$
3. $(X = Z) \leftarrow \mathsf{K}(X = Y), \mathsf{K}(Y = Z)$
4. $f(X_1, \ldots, X_n) = f(Y_1, \ldots, Y_n) \leftarrow \mathsf{K}(X_1 = Y_1), \ldots, \mathsf{K}(X_n = Y_n)$ for every function f.
5. $p(Y_1, \ldots, Y_n) \leftarrow \mathsf{K}(X_1 = Y_1), \ldots, \mathsf{K}(X_n = Y_n), \mathsf{K}(p(X_1, \ldots, X_n))$ for every predicate p.
6. $\neg p(Y_1, \ldots, Y_n) \leftarrow \mathsf{K}(X_1 = Y_1), \ldots, \mathsf{K}(X_n = Y_n), \mathsf{K}(\neg p(X_1, \ldots, X_n))$ for every predicate p.
7. $X \neq Y \leftarrow \neg\mathsf{K}(X = Y)$.

The *definition* of l is the collection of all rules in P with l in the head. Given an objective literal l, we say that l occurs *positively* (resp., *negatively*) in P if $\mathsf{K}l$ (resp., $\neg\mathsf{K}l$) appears in a rule body in P. Similar to classic logic programs, we define stratification as follows:

Definition 2 A program P is *stratified* if there is a partition of the program $P = P_1 \cup \ldots \cup P_n$, such that the following conditions hold:

1. If l occurs positively in P_i, then its definition is contained in $\bigcup_{j \leq i} P_j$.
2. If l occurs negatively in P_i, then its definition is contained in $\bigcup_{j < i} P_j$. □

Example 1 Let P be the following program:

$fly(X) \leftarrow \mathsf{K}bird(X), \neg\mathsf{K}\neg fly(X);$
$\neg fly(X) \leftarrow \mathsf{K}penguin(X);$
$bird(X) \leftarrow \mathsf{K}penguin(X);$
$penguin(tweety).$

Then P is stratified by:
$P_1 = \{penguin(tweety); \neg fly(X) \leftarrow \mathsf{K}penguin(X)\}$,
$P_2 = \{bird(X) \leftarrow \mathsf{K}penguin(X);$
$fly(X) \leftarrow \mathsf{K}bird(X), \neg\mathsf{K}\neg fly(X)\}$. □

[2] A *complete equational theory* is a theory for equality where for any two ground terms s and t, either $s = t$ or $s \neq t$. In Section 4 we will show that a complete equational theory is necessary for the resolution procedure.

In the definition of stratification we treat positive and negative literals in a symmetric way, which raises the question of consistency. To guarantee consistency, we need additional constraints.

Definition 3 (Conservativeness) A rule with a literal l in the head is *positively-conservative* (resp., *negatively-conservative*) if l is a positive (resp., negative) literal and $\neg\mathsf{K}\neg l$ is in the rule body. A predicate p is *positively-conservative* (resp., *negatively-conservative*) if every rule in the definition of p (resp., $\neg p$) is positively-conservative (resp., negatively-conservative). A program P is *positively-conservative* (resp., *negatively-conservative*) if every predicate in P is positively-conservative (resp., negatively-conservative). P is *conservative* if for every predicate $p \in P$, p is either positively-conservative or negatively-conservative. □

For instance, in Example 1, fly is positively-conservative, $bird$ is negatively-conservative,[3] etc; the program itself is thus conservative. The term "conservative" indicates a cautious attitude to inferring the head literals. For example, fly is positively-conservative because before inferring that x can fly, we always check first that $\neg fly(x)$ is not known. On the other hand, we don't check anything before concluding that something cannot fly.

We say that l_1 *refers* to l_2 *positively* (resp., *negatively*) if l_1 is in a rule head and $\mathsf{K}l_2$ (resp., $\neg\mathsf{K}l_2$) is in the rule body. For example, p refers to q positively, and to r negatively in $\{p \leftarrow \mathsf{K}q, \neg\mathsf{K}r\}$.

Definition 4 A *dependency graph* of a program P is a directed graph representing the "refers to" relation between literals. An edge (l_1, l_2) in the graph is *positive* if l_1 refers to l_2 positively in some rule. An edge (l_1, l_2) in the graph is *negative* if l_1 refers to l_2 negatively. □

Lemma 1 P is stratified iff its dependency graph has no cycle containing an negative edge. □

Lemma 2 A stratified program P is conservative iff for each positive literal l in the dependency graph, either $(l, \neg l)$ or $(\neg l, l)$ is a negative edge, but not both. □

3.2 Fixpoint Semantics for Definite Autoepistemic Programs

In this section we present the fixpoint semantics for stratified and conservative programs. We first show that for each stratified program, there exists a mapping induced by the program. This gives a fixpoint characterization of such programs. Then we show that fixpoints are consistent for conservative programs.

Let $U_P = H_P \cup (\neg \cdot H_P) \cup E \cup (\neg \cdot E)$, where H_P is the Herbrand base of program P, $E = \{s = t : s$ and t are ground terms $\}$ and $\neg \cdot S = \{\neg A : A \in S\}$ for any set of literals S. In the following I will denote a set of ground literals (including equality literals). I is *consistent* if $I \cap \neg \cdot I = \emptyset$.

For each program P define T_P as follows:
$T_P(I) = \{l \in U_P : l \leftarrow \mathsf{K}l_1, \ldots, \mathsf{K}l_m, \neg\mathsf{K}l'_1, \ldots, \neg\mathsf{K}l'_n$ is a ground instance of a clause in P, $\forall i,\ 1 \leq i \leq m,\ l_i \in I$

[3] There is no definition for $\neg bird$, so the condition is vacuously true.

and $\forall j,\ 1 \le j \le n,\ l_j' \notin I$. }
Consider a program P stratified by

$$P = (P_1 \dot\cup \ldots \dot\cup P_n)$$

and define

$$I_1 = T_{P_1} \uparrow \omega(\emptyset)$$
$$I_2 = T_{P_2} \uparrow \omega(I_1)$$
$$\ldots$$
$$I_n = T_{P_n} \uparrow \omega(I_{n-1})$$

I_n is also denoted by $iter(T_1, \ldots, T_n, \emptyset)$ to stress the fact that I_n is computed using T_{P_i}'s in an iterative fashion. We have the following fixpoint characterization of a stratified program P, which is proved similarly to the fixed point theory by Apt, Blair, and Walker [Apt *et al.*, 1988]:

Theorem 1 Suppose P is stratified by $P = (P_1 \dot\cup \ldots \dot\cup P_n)$. Then

$$T_P(I_n) = I_n.$$

□

Next we show that stratified and conservative programs preserve consistency.

Theorem 2 Let P be conservative and stratified by $P = (P_1 \dot\cup \ldots \dot\cup P_n)$. Then $I =$ $iter(P_1, P_2, \ldots, P_n, \emptyset)$ is consistent. □

Finally, we establish a main result of this paper:

Theorem 3 Suppose P is conservative and stratified by $P = (P_1 \dot\cup \ldots \dot\cup P_n)$. Let $I =$ $iter(P_1, P_2, \ldots, P_n, \emptyset)$. Then $T = \mathcal{E}(I)$ is the *unique* iterative expansion of P.
(Here $\mathcal{E}$ is the operator of [Marek, 1989] defined in Section 2) □

Theorem 3 shows that the construction of the fixpoint I for a program P is independent of any particular stratification of P.

4 SLSAE-resolution

In this section we define *SLSAE-resolution* (SLS-resolution for AutoEpistemic programs) for stratified and conservative programs. SLSAE-resolution is similar to SLSC-resolution defined in [Przymusinski, 1989a], but is designed for autoepistemic programs. In autoepistemic logic, the deduction rule does not hold, i.e., "from ϕ infer $\mathsf{K}\phi$" does not make $(\phi \rightarrow \mathsf{K}\phi)$ a theorem. We will use $\phi \vdash \psi$ to denote the inference rule "from ϕ infer ψ."

Definition 5 An inference rule $\phi \vdash \psi$ is *sound* with respect to a set of axioms P if for all substitutions θ, $P \models_S \forall\phi\theta$ implies $P \models_S \forall\psi\theta$ for all iterative expansions S of P. Here $\forall$ quantifies over all unbounded variables in the formula. □

Lemma 3 An inference rule $\phi \vdash \psi$ is sound with respect to P iff for all ground substitutions θ, $P \models_S \phi\theta$ implies $P \models_S \psi\theta$ for all iterative expansions S of P. □

Definition 6 A formula ϕ is *complete* with respect to P if for every ground instance ϕ' of ϕ, either $P \models_S \phi'$ or $P \models_S \neg\phi'$ for all iterative expansions S of P. □

For example, introspective formulas (i.e., formulas of the form $\mathsf{K}\phi$, where ϕ is any formula) and equality formulas are complete formulas with respect to any program P.

Lemma 4 Let ϕ be a complete formula with respect to P with free variable x, then $\mathsf{K}\exists x\phi(x) \vdash \exists x\mathsf{K}\phi(x)$ is sound. (Note that, as described in Section 2.2, $\exists x\mathsf{K}\phi(x) \vdash \mathsf{K}\exists x\phi(x)$ is sound for any formula ϕ.) □

Lemma 5 Let ϕ be a complete formula with respect to P. If $\phi \vdash \psi$ is sound then $\neg\psi \vdash \neg\phi$ is sound. □

Note that the converse of Lemma 5 does not hold. For example, suppose p is the only predicate symbol and a, b, c are the only constants. Consider $P =$ $\{p(a),\ \neg p(c)\}$. Here $p(x) \vdash x = a \vee x = b$ and $x \neq \neg a \wedge x \neq \neg b \vdash \neg p(x)$ are sound, but $p(x)$ is not a complete formula.

Similar to classic logic programming, we define goal, query, and answers to a query. The definitions are with respect to a stratified and conservative program P.

Definition 7 A *goal* G is a headless clause $\leftarrow L_1, \ldots, L_k$, where $k \ge 0$ and each L_i is a literal. We also write $G = \leftarrow Q$, where $Q = L_1, \ldots, L_k$, and refer to either G or Q as a *query*. □

Definition 8 An *equality literal* is (i) an equality $(t_1 = t_2)$ or (ii) an inequality $\forall(t_1 \neq t_2)$, where $\forall$ quantifies some (perhaps none) of the variables occurring in the equality literal and t_1 and t_2 are terms. A *simple equality formula* is of the form

$$\exists(l_1 \wedge \ldots \wedge l_n), \quad n \ge 0,$$

where the l_i's are equality literals and $\exists$ quantifies over some (perhaps none) of the variables occurring in the l_i's. A *normal equality formula* is of the form

$$S_1 \vee \ldots \vee S_m, \quad m \ge 0,$$

where each S_j, $1 \le j \le m$, is a simple equality formula. □

The following theorem is from [Chan, 1988; Przymusinski, 1989a], adapted to autoepistemic logic:

Theorem 4 For every equality formula E with n free variables $x_1, \ldots, x_n$, there exists a normal equality formula N_E such that both $F \vdash N_F$ and $N_F \vdash F$ are sound with respect to Clark's Equational theory. □

Definition 9 A simple equality formula $\mathcal{A}$ is an *answer* to a query $\leftarrow Q$ if $\mathcal{A} \vdash Q$ is sound. □

Definition 10 A query Q is *solvable* if there is a finite set $S = \{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$ of SLSAE-answers to Q, as defined in the SLSAE-resolution below, such that for any SLSAE-answer $\mathcal{A}$ to Q, $\mathcal{A} \vdash \mathcal{A}_1 \vee \ldots \vee \mathcal{A}_n$ is sound. $\{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$ is called a *complete-answer-set* to Q. A program P is *solvable* if every query Q to P is solvable. □

In the following, we will assume that the program is solvable. SLSAE-resolution is sound and complete for solvable programs. In particular, if $\{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$ is a complete-answer-set to Q, then $\mathcal{A}_1 \vee \ldots \vee \mathcal{A}_n \vdash Q$ is sound. By Theorem 5 below, $Q \vdash \mathcal{A}_1 \vee \ldots \vee \mathcal{A}_n$ is sound. Since both Q and $\mathcal{A}_1 \vee \ldots \vee \mathcal{A}_n$ are complete formulas, by

Lemma 5 we have that both $\neg(\mathcal{A}_1 \vee \ldots \vee \mathcal{A}_n) \vdash \neg Q$ and $\neg Q \vdash \neg(\mathcal{A}_1 \vee \ldots \vee \mathcal{A}_n)$ are sound. Then, by Theorem 4, there is a normal equality formula $\mathcal{B}_1 \vee \ldots \vee \mathcal{B}_m$ equivalent to $\neg(\mathcal{A}_1 \vee \ldots \vee \mathcal{A}_n)$, so that $\{\mathcal{B}_1, \ldots, \mathcal{B}_m\}$ is a complete-answer-set to $\neg Q$.

Suppose $\{P_1, \ldots, P_n\}$ is a stratification of P. For any literal L, let $\boldsymbol{stratum}(L) = i$ if $L = \mathsf{K}l$ and i is the unique integer such that the definition of l belongs to P_i. If $L = \neg\mathsf{K}l$ and i is the unique integer such that the definition of l belongs to P_i, then define $\boldsymbol{stratum}(L) = i+1$. Note that $\boldsymbol{stratum}(\mathsf{K}l) = 1$ if l is an equality literal and $\boldsymbol{stratum}(\mathsf{K}l) = 2$ if l is an inequality literal. Also assume that $\neg\mathsf{K}\forall(t_1 \neq t_2)$ is simplified to $\mathsf{K}(t_1 = t_2)$ by the equational theory.

Let G be a goal $\leftarrow L_1, \ldots, L_k$. We define $\boldsymbol{stratum}(G)$ as follows:

$$\boldsymbol{stratum}(G) = \begin{cases} 0, & \text{if } G \text{ is empty} \\ max\{\boldsymbol{stratum}(L_i) : i = 1, \ldots, k\} & \text{otherwise} \end{cases}$$

We will also write $\boldsymbol{stratum}(Q)$ instead of $\boldsymbol{stratum}(G)$ for $G = \leftarrow Q$.

Definition 11 Suppose that θ is some substitution for free variables in $\forall(t_1 \neq t_2)$. The inequality $\forall(t_1 \neq t_2)\theta$ is *valid* if $t_1\theta$ and $t_2\theta$ cannot be unified. It is ***satisfiable*** if $t_1\theta$ and $t_2\theta$ cannot be unified by binding only universally quantified variables. □

If an inequality is unsatisfiable, it can not be in any answer. If an inequality is valid, it does not contribute to the information in an answer. Only satisfiable, but non-valid inequalities will appear in an answer, they are called ***primitive*** inequalities [Chan, 1988]. We define ***SLSAE-resolution*** as follows:

Definition 12 (SLSAE-resolution) Let P be a stratified, conservative, and solvable program, R any fixed computation rule that selects exactly one literal from a given goal at a time. The only restriction is that R never selects primitive inequalities relative to the current substitution. Let G be any goal with variables $x_1, \ldots, x_n$. By induction on the stratum of G, define:

1. *SLSAE-tree* $T(G)$ for G, each node of $T(G)$ is labeled with a goal;
2. *Success leaves* of $T(G)$;
3. *SLSAE-answers* $\mathcal{A}(x_1, \ldots, x_n)$ for G — the simple equality formulas associated with success leaves of $T(G)$.

(Base) If $\boldsymbol{stratum}(G) = 0$, then G is an empty clause and its *SLSAE-tree* consists of a single node, denoted G; it is a ***success*** leaf with the associted SLSAE-answer ***true***.

(Induction) Assume now that $\boldsymbol{stratum}(G) = m, m > 0$, and that SLSAE-tree, success leaves, and SLSAE-answers have already been defined for goals whose strata are less than m. Then

(i) The root node of T is G.

(ii) If H is an arbitrary node of T labeled with $\leftarrow L_1, \ldots, L_n$ and $L = L_i$ is the literal selected from H, then the immediate descendents of H in $T(G)$ are defined as follows:

1. Suppose L is $\mathsf{K}l$, where l is not an equality literal. For each rule of the form $l' \leftarrow L'_1, \ldots, L'_m$ such that there is an mgu θ with $l\theta = l'\theta$, an immediate descendent of H is labeled $\leftarrow (L_1, \ldots, L_{i-1}, L'_1, \ldots, L'_m, L_{i+1}, \ldots, L_n)\theta$.
   If there is no such rules then H is a failure node.
2. If L is an equality $\mathsf{K}(t_1 = t_2)$, and t_1 and t_2 are unifiable via a most general unifier θ, then the immediate descendent H is the goal K obtained by applying the substitution θ to H and removing L. Otherwise (if t_1 and t_2 are not unifiable), H has no immediate descendents and is a failure node;
3. If L is an inequality $\mathsf{K}\forall(t_1 \neq t_2)$ or $\neg\mathsf{K}\forall(t_1 = t_2)$ and if L is valid (relative to the current substitution), then the only descendent of H is $K = H - \{L\}$.
   If L is unsatisfiable then H has no descendents and is a failure node. (Recall that R selects only valid or unsatisfiable inequalities.)
4. If L is $\neg\mathsf{K}l$, where l is not an equality literal. Then by definition of stratification, $\boldsymbol{stratum}(\mathsf{K}l) < m$, and therefore the SLSAE-tree $T(G')$ for G', labeled $\leftarrow \mathsf{K}l$, has already been defined. By Theorem 5 below, $\mathsf{K}l \vdash \mathcal{B}_1 \vee \ldots \vee \mathcal{B}_m$ is sound,[4] where $\{\mathcal{B}_1, \ldots, \mathcal{B}_m\}$ is a complete-answer-set to G'. By Lemma 5 and Theorem 4, there is a complete-answer-set $\{\mathcal{A}'_1, \ldots, \mathcal{A}'_q\}$ to $\neg\mathsf{K}l$. We consider three cases:
   (a) If $\mathcal{A}'_1 \vee \ldots \vee \mathcal{A}'_q = true$, then the only immediate descendent of H is $H - \{L\}$;
   (b) If $\mathcal{A}'_1 \vee \ldots \vee \mathcal{A}'_q = false$, then H has no immediate descendents and is a failure node;
   (c) Otherwise, $q > 0$ and H has q immediate descendents $H_j, 1 \leq j \leq q$, labeled with:[5]
   $$\leftarrow L_1, \ldots, L_{i-1}, \mathsf{K}\mathcal{A}'_j, L_{i+1}, \ldots, L_k.$$
5. Finally, if H is labeled with an empty clause or its label contains only satisfiable, but not valid inequalities, then H has no immediate descendents and is a success leaf. Its associated SLSAE-answer is a simple equality formula defined as:
   $$\mathcal{A} = \exists(x_1 = x_1\theta \wedge \ldots \wedge x_n = x_n\theta \wedge L_1 \wedge \ldots \wedge L_k)$$
   where θ is the substitution obtained as a composition of the previously applied most general unifiers, and $\exists$ quantifies over all free variables not in G. □

Theorem 5 Let P be a stratified, conservative, and solvable program. Let R be a fixed computation rule, and $\leftarrow Q$ be a query with a complete-answer-set $\{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$. Then $Q \vdash \mathcal{A}_1 \vee \ldots \vee \mathcal{A}_n$ is sound. □

Theorem 6 (Soundness of SLSAE-resolution) Let P be as above. If $\mathcal{A}$ is an SLSAE-answer to $\leftarrow Q$, then $\mathcal{A} \vdash Q$ is sound. □

[4]From now on, we write $\mathcal{A}$, $\mathcal{B}$, etc. instead of $\mathcal{A}(x_1, \ldots, x_n)$, $\mathcal{B}(x_1, \ldots, x_n)$.

[5]Note that equality formulas are complete, so by Lemma 4 the appearance of $K\mathcal{A}'_j$'s does not violate the syntactic form of a goal.

Theorem 7 (Completeness of SLSAE-resolution) Let P be as above, $\leftarrow Q$ a query to P and $\{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$ a complete-answer-set to $\leftarrow Q$. If $\mathcal{A}$ is an equality formula such that $\mathcal{A} \vdash Q$ is sound, then $\mathcal{A} \vdash (\mathcal{A}_1 \vee \ldots \vee \mathcal{A}_n)$ is sound. □

5 Conclusion and Future Work

We presented a fixpoint semantics for stratified and conservative autoepistemic programs, which uniquely determines the only iterative expansion of such a program. A resolution procedure, called SLSAE-resolution, is also defined. SLSAE-resolution is sound and complete for solvable programs. The following issues merit further investigation:

1. The results of this paper allow us to generalize the framework to permit objective clauses, not only objective literals, in the programs. Such programs are called *disjunctive autoepistemic programs.* Disjunctive autoepistemic programs are strictly more general than any of the current approaches in the literature that combine negation and disjunction. The following example illustrates this:

   Example 2 Consider the following database:

   $proj1(X) \vee proj2(X) \leftarrow Kstudent(X);$
   $undecided(X) \leftarrow \neg Kproj1(X), \neg Kproj2(X);$
   $proj1(a);\ proj2(b);$
   $student(a);\ student(b);\ student(c).$

   The first rule says every student must be in at least one of the two projects *proj1* and *proj2.* The second rule says those students who are neither in *proj1* nor in *proj2* have not yet decided which project to undertake. Given the query $\leftarrow$ *Kundecided(X),* the answer should be $X = c$. Note that $\neg K(A \vee B) \rightarrow \neg KA \wedge \neg KB$ but not vise versa. Without the operator *K,* for example in [Ross, 1989b], this distinction can not be made, hence the first rule can be used to exclude $X = c$ as an answer to the query $\leftarrow undecided(X)$. □

2. Another issue to pursue is the efficient implementation for autoepistemic programs. For example, one may try to generalize the SYGRAF approach in [Kifer and Lozinskii, 1988] to definite and disjunctive autoepistemic programs.

Acknowledgements: I would like to thank Weidong Chen and Michael Kifer for many helpful discussions and invaluable suggestions. An anonymous referee's suggestion is also helpful to improve the presentation of this paper.

References

[Apt *et al.*, 1988] K.R. Apt, H.A. Blair, and A. Walker. Towards a Theory of Declarative Knowledge. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming,* pages 89-148. Morgan Kaufmann Pub., 1988.

[Chan, 1988] D. Chan. C onstructive Negation Based On The Completed Database. In *Proceedings of the Fifth Logic Programming Symposium,* pages 111-125, 1988.

[Gelder *et al.*, 1988] Allen Van Gelder, Kenneth Ross, and John Schlipf. Unfounded Sets and Weil-Founded Semantics for General Logic Programs. In *Proc. 7th Symp. on Principles od Database Systems,* pages 221 — 230, 1988.

[Gelfond and Lifschitz, 1990] M. Gclfond and V. Lifschitz. Logic Programs with Classical Negation. In *Proceedings, 1th International Conference on Logic Programming,* pages 579-597, June 1990.

[Kifer and Lozinskii, 1988] M. Kifer and E. Lozinskii. SYGRAF: Implementing Logic Programs in a Database Style. *IEEE Transaction on Software Engineering,* pages 922-934, 1988.

[Konolige, 1988] K. Konolige. On the Relation between Default and Autoepistemic Logic. *Artificial Intelligence,* 35:343-382, 1988.

IKonolige, 1989] K. Konolige. On the Relation between Circumscription and Autoepistemic Logic. In *Proceedings of IJCAI,* pages 1213-1218, 1989.

[Kunen, 1987] K. Kunen. Negation in Logic Programming. *Journal of Logic Programming,* pages 289-308, 1987.

[Marek and Truszczynski, 1988] W. Marek and M. Truszczynski. Autoepistemic Logic, to appear in JACM, Available as TR 115-88, Dept. of CS, Univ. of Kentucky, Lexington, KY 40506-0027, 1988.

[Marek and Truszczynski, 1989] W. Marek and M. Truszczynski. Relating Autoepistemic and Default Logics. In *Proceedings, First International Conference on Principles of Knowledge Representation and Reasoning,* pages 276-288, 1989.

[Marek, 1989] W. Marek. Stable theories in autoepistemic logic. *Fundarnenta Informaticae,* 12:243-254, 1989.

|Moore, 1985] R. C. Moore. Semantical Considerations on Non-Monotonic Logic. *Artificial Intelligence,* 25:75-94, 1985.

[Przymusinski, 1989a] T. Przymusinski. On Constructive Negation in Logic Orogramming. In *Proceedings of the North American Conference on Logic Programming,* 1989.

[Przymusinski, 1989b] T. Przymusinski. On the declarative and procedural semantics of Logic Programs. *J. of Automated Reasoning,* 5:167-205, 1989.

[Ross, 1989a] K. Ross. A Procedural Semantics for Well Founded Negation in Logic Programs. In *Proceedings, Eighth ACM Symposium on Principles of Database Systems,* pages 22-33, 1989.

[Ross, 1989b] Kenneth Ross. The Well Founded Semantics for Disjunctive Logic Programs. In *Proceedings, First International Conference on Deductive and Object-Oriented Databases,* Dec. 1989.