The J. of MIS
Research
Vol.2, No.2, Dec 1992

연역적 데이터베이스의 의미론에 관한 현실적 접근+

이 대 용*

A Practical Approach to Semantics in Deductive Databases

A deductive database consists of collection of stored facts and deductive rules. It can answer queries based on logical deduction from the stored facts and general rules. A deductive database has both a declarative meaning(semantics) and a procedural meaning. The declarative semantics of a deductive database provides a definition of the meaning of the program in a manner which is independent of procedural considerations, context-free, and easy to manipulate, exchange and reason about.

This paper investigates various declarative semantics of deductive databases, dicusses related computational issues, and suggests another declarative semantics for deductive databases which is more practical than others.

I. Introduction

A relational database(RD) is a collection of tables filled with data supplemented with the capability to manipulate its contents and to answer questions about the data. Any question expressed in an appropriate query language can be answered. If a set of elements satisfying the query exists in the tables then the answer is true; otherwise the answer is false. So an RD assumes that the database contains complete information about a context and anything not in the database is assumed to be false. To be useful, an RD must be

+ This paper was partly supported by Korea Research Foundation.

* 조선대 경영학과.

maintained in a consistent and complete manner. Also, an RD must respond to queries in a manner intended by the user.

Although the concept of RD has been proved to be useful, the capabilities of RD are severely limited by their inability to handle deductive information. RD cannot deduce new information from facts already present in the database and from deductive rules which are known and can be included in the database. A deductive database(DD), which generalizes RD, consists of collection of stored facts (extensional database) and deductive rules (intentional database). DD can answer queries based on logical deduction from the stored facts and general rules.

Logic programming refers to high level languages, like Prolog, based on classical logic. Because classical logic has a procedural interpretation in which a logical statement is interpreted as a definition of a procedure, it can be an effective programming language. Logic programming is very appealing because first order language which is machine intelligible, instead of natural language can be used as a specification of the problem domain. Furthermore the correctness of a logic program can be proved by classical logic[Kowalski, 1982]. A logic program is virtually the same as a deductive database in a sense that both consist of logical assertions and the same interpreter or compiler can be used to answer a query in a DD or to perform a user command in a logic program. Thus, we will treat deductive databases and logic programs as indistinguishable in this sense.

A program(logic program or database program) has both a declarative meaning (semantics) and a procedural meaning. The declarative semantics of a program provides a definition of the meaning of the program in a manner which is independent of procedural considerations, context-free, and easy to manipulate, exchange and reason about. Procedural semantics of a program, on the other hand, usually is given by providing a procedural mechanism that is capable of providing answers to queries[Przymusinski, 1989].

Finding a suitable declarative semantics is one of the most important problem in deductive database and logic programming. This problem is originated from the declarative semantics for negation in logic programming. The form of negation currently used in logic programming and deductive databases is different from negation in classical logic. Because of inefficiency of negation in classical logic, Negation as Failure (NF) is used in logic programming and deductive databases. In this rule, if every attempt to prove a positive ground literal p fails then its negation $\neg p$ is assumed to be true. Declarative semantics for this metarule, called a program completion approach, have been suggested by many researchers[Reiter, 1978; Clark, 1978; Fitting, 1985; Kunen, 1987].

Another approach to the semantics of deductive database is called a model theoretic approach. In this approach, a certain model of the original program is presumed to be the

intended meaning of the program, the one that the programmer and program users have in mind. Various models have been suggested to be the intended meaning of the program. In this paper, we investigate these models and suggest a new model which is easy to find and understand as a semantics for deductive databases.

This paper is organized as follows:

Next section introduces the definition of programs and related terminologies, section 3 investigate various model theories which have been suggested as an appropriate meaning of deductive databases. In section 4, we suggest another model as a semantics for deductive databases and conclude the paper in section 5.

Ⅱ. Definition of a Program

This section introduces the syntax of a program and conditions of logical variables. The set of symbols used in a program contains predicate, constant and function symbols. In addition, it is assumed to contain connectives(,, ¬,←), punctuation symbols and a countably infinite set of variables(X, Y, Z, ⋯). A term is defined to be a variable or a constant. An atom is an expression of the form $p(t_1, \cdots, t_n)$, where p is a predicate symbol and $t_1, \cdots, t_n$ are terms. An atom or its negation is a literal. An expression is ground if it does not contain any variables. The set of all ground atoms in a program is the Herbrand base of the program and the set of all the ground terms is the Herbrand universe of the program. By a program we mean a finite set of clauses of the form

$A \leftarrow L_1, \cdots, L_n.$

where $n \geq 0$, A is an atom and $L_1, \cdots, L_n$ are literals. It is also assumed that all variables are universally quantified. Literals L_i are called premises, A is called the head, and $L_1, \cdots, L_n$ is called the body of the clause in which commas are to be interpreted conjunction connectives.

Program1:

```
supply(X, Y)←subpart(Y, Z), supply(X, Z).
subpart(X, Y)←subp(X, Y).
subpart(Y, Z)←subp(X,Z), subpart(Y, X).
supply(s1, p1).
supply(s1, p2).
subp(p3, p1).
subp(p5, p3).
```

The first rule means that if Y is a subpart of Z and X supplies Z then X supplies Y. The second and third rules mean that if X is a subpart of Z and Y is a subpart of X then Y is a subpart of Z. The remaining facts mean that s1 supplies part p1 and p2 and p3 and p5 are subpart of p1 and p3 respectively. Note that subpart relation is a transitive closure of subp relation.

If P is a program then it is always possible to decompose P into two disjunctive subsets P_e and P_i, of the extensional and intentional parts of the program. In the above example, the first three rules are intentional part of the program and last four rules are extentional part of the program. The intentional part P_i is

usually assumed to be relatively small and fairly static and it represents the deductive component of the database. The extensional part P_e is relatively large, subject to changes and represents the relational component of the database[Reiter, 1984]. For the convenence of analysis, we assume that a program contains finite number of objects without functions(Datalog ¬).

A program consisting exclusively of Horn Clauses in which no negative literals appear will be called a Definite Horn Clause Program (or Positive Program). If a Horn Clause Program contains one or more negative literals, than we will call it a Horn Clause Program with Negation(or General Program) to emphasize the presence of negative literals).

1. Model Theory and Fixed Point

An Herbrand interpretation I of a program P is an assignment of truth values to all ground atoms in the Herbrand base of P. Because the set of true ground atoms in a Herbrand interpretation is a subset of the Herbrand base, any 2-valued Herbrand interpretation can be viewed as a subset of the Herbrand base. Also, set inclusion is a partial order on a set of Herbrand interpretations. A Herbrand interpretation I in which all clauses in a program are true is called a Herbrand model of the program. If a ground atom is true in every model of a program then is a logical consequence of the program. In this paper we consider only Herbrand interpretations and Herbrand models for a program because any ground conclusion from a program should be in Herbrand base of the program. So, when we mention an interpretation or model it means a Herbrand interpretation or Herbrand model unless specified explicitly.

The declarative semantics of a program is often defined by using the fixed point of a natural operator T_p acting on ordered sets of interpretations[van Emden, 1976]. Set inclusion $\subseteq$ is an ordering on the set L of Herbrand interpretations of a program and T_p is an operator $T_p : L \rightarrow L$ on L. The operator T_p is called monotone if $I \subseteq J$ implies $T_p(I) \subseteq T_p(J)$ for any I, J in L. An interpretation I is a fixed point of T_p if $T_p(I) = I$. A least fixed point of a monotone operator T_p, which is a subset of every fixed point of T_p, is generated by iterating the operator T_p starting from the smallest Herbrand interpretation(interpreting every ground atom in the program as false) and obtaining the sequence:

$T_p \uparrow 0 = \phi$.

$T_p \uparrow n+1 = T_p(T_p \uparrow n)$.

An iteration $T_p \uparrow n$ is a fixed point of T_p if and only if $T_p \uparrow n = T_p \uparrow n+1$.

Ⅲ. Previous Research

1. Least Model Semantics

A set of ground atoms logically implied by a Positive Program must be true in every model of the program. Models of a Positive

Program have a property that the intersection of two models is also a model and there is a model of the program which is a subset of every model. The model which is a subset of every model is called a least model and a ground atom in the least model is a logical consequence of the program. Thus, the declarative semantics of a program must be the least model of the program and any ground atom not in the model should be assumed to be false because in that way the true facts deduced from the program are minimized.

This semantics can also be characterized as a fixed point of a natural operator T_p on the program too. Given a program P and an interpretation I, an interpretation $T_p(I)$ is defined as follow:

A ground atom A is true in $T_p(I)$ if A is a ground fact or there is a ground rule $A \leftarrow L_1, \cdots, L_n$. and $L_1, \cdots, L_n$. are true in I.

Otherwise A is false in $T_p(I)$.

The T_p operator of a Positive Program is monotonic and the least fixed point of the operator exists and it coincides with the least model of the program.

This clear and intuitive semantics is equivalent to Closed World Assumption (CWA) for a Positive Program because any ground atom that is in the least model is a logical consequence of the program and any ground atom that is a logical consequence of the program is in the least model.

Consider the following process for seeking a least model. This amounts to execution of the following loop:

WHILE a ground atom $p(t_1, \cdots, t_n)$ can be found

assign true to $p(t_1, \cdots, t_n)$;

substitute $t_1, \cdots, t_n$ for logical variables $X_1, \cdots, X_n$ in a rule containing $p(X_1, \cdots, X_n)$ in the program;

remove fact $p(t_1, \cdots, t_n)$ from the program;

remove $p(t_1, \cdots, t_n)$ from the premise of every rule in which $p(t_1, \cdots, t_n)$ appears as a positive literal;

remove all modified ground rules which cannot be binding on the values of remaining literals(i.e. rules with $\neg p(t_1, \cdots, t_n)$ in their premise)

ENDWHILE

If the above process terminates with an empty set of ground rules, then a model has been found. At the very least all literals assigned true in the execution of this WHILE loop must be assigned true in any model(if any model exists). In a Definite Horn Clause Program, a model can be built by assigning false to all still unassigned literals in the Herbrand Base.

For Program1, the loop assigns true to {supply(s1,p1), supply(s1,p2), subpart(p3, p1), subpart(p5,p3), subpart(p5,p1), supply (s1,p3), supply(s1,p5)}. By assigning false to any other ground atoms we can have a model for Program1 in previous page.

It is exactly the above WHILE loop that is executed in searching for a least model. Consider the class of General Programs. Any rule has a positive literal for its head and any

fact is a positive literal. Negative literals may appear only in the body of a rule. For General Programs the least model semantics is not defined because the model intersection property does not hold and the T_p operator is no longer monotonic.

Program2.

innocent(X)←child(X), ¬liar(X).
liar(X)←liar(X), ¬child(X).
child(lee).

The above program has two models {child (lee), liar(lee)} and {child(lee), innocent (lee)}. Because intersection of two models of Program1 is not a model of the program, Program2 does not have a least model.

2. Minimal Supported Model Semantics

One of the reasons of anomalies concerning NF rule is unsafe use of negation in logic programming. Unsafe negation can occur when a negative subgoal is tried when the truth value of its atom is not determined. The issue of safe usage of negation is studied by many researchers[Naish, 1985; Clark, 1978; van Gelder, 1988]. The idea of safe use of negation is suggested by Clark as hierarchical programs and allowed query for which NF rule is sound and complete for Comp(P). A program is hierarchical if every relation has a finite extension that can be computed by SLDNF resolution. It is clear that a hierarchical program is so restrictive that recursion is not permitted in it. Apt, Blair and Walker and others suggest a framework for negation, called a stratified program, in which usage of negation is restricted but recursion is allowed[Apt, 1988; Naqvi, 1987]. Predicates are assigned to strata 1, 2, ···, S so that the use of negation in definitions is appropriately limited. A program is stratified if for each rule in the program $A \leftarrow L_1, \cdots, L_m$, the tratum of A is strictly greater than the stratum of each negative literal in $L_1, \cdots, L_m$ and stratum of A is greater than or equal to stratum of each positive literal in $L_1, \cdots, L_m$. Stratification can be considered as a priority relation between predicates in a program so that a predicate in a lower stratum has a higher priority and the truth value of a predicate with higher priority is decided before the truth value of a predicate with lower priority is decided. Thus only already decided relations(predicates) are used to define a new relation.

A stratification of a program can be found by the following loop:

```
WHILE Program and Stratum are not empty
    WHILE Predicate ⊇ Head
        i=i+1;
        Predicate-Head=Stratum i;
        remove Stratum i from Program;
        ENDWHILE
        i=i+1;
        Positive_head-(Negative_head+
        Positive_head defined recursively
            by Negative_head)=Stratum i;
        remove Stratum i from Program;
```

ENDWHILE

Predicate : All predicates that Program contains.

Head : Predicate defined by a rule.

Stratum i : Predicates assigned in stratum i.

Positive_head : Predicates defined by a rule containing only positive literals.

Negative_head : Predicates defined by a rule containing negative literals.

If this process terminates with empty program then the program is stratified by the process and if terminates with non-empty program then the program can not be stratified.

Using the above loop, Program2 is stratified by Program2={child(lee).} & {liar(X)←liar(X), ¬child(X).} & {innocent(X)←child(X), ¬liar(X).}.

Because a stratified program may have negation in it, generally a least model may not exist. A model(interpretation) of a program P is supported if and only if for each ground atom A in the model(interpretation) there exists a ground clause with head A whose body is true in the model (interpretation). Apt et al. suggested that a minimal and supported model be considered to be the declarative semantics of a program and they show that each stratified program has a unique minimal supported model. The minimal supported model can be defined as a fixed point of an operator also. A natural operator T_i is defined for each stratum i. An operator T_i for a stratum i is monotonic and it has a fixed point. An iterative fixed point of T_i for a program P is defined as follow:

$T_1 \uparrow_w(\phi) = M_1$(fixed point of T_1 with interpretation ϕ)

$T_2 \uparrow_w(M_1) = M_2$(fixed point of T_2 with interpretation M_1)

$T_3 \uparrow_w(M_1 \cup M_2) = M_3$(fixed point of T_3 with interpretation $M_1 \cup M_2$)

.....

$T_n \uparrow_w(M_1 \cup M_2 \cup ... \cup M_{n-1}) = M_n$(fixed point of T_n with interpretation $M_1 \cup M_2 \cup ... \cup M_{n-1}$)

The iterative fixed point M_n of a program P is a minimal supported model of P.

Clearly NF rule is not complete for this semantics because recursion is allowed in stratified programs. Apt et al. elegantly define a top-down interpreter which uses bottom-up information to check for the presence of infinite loops. This interpreter is complete and sound for finite stratified programs (finite Herbrand base) in minimal supported model semantics. NF rule using tight derivations can also be an interpreter for this semantics. A ground atom is proved by a tight NF derivation if any infinite loop in the path of the proof procedure is checked and removed[van Gelder, 1988].

For a stratified logic program, the following process which considers strata 1, ···, s in turn yields a unique minimal supported model also.

FOR Stratum=1 to S DO

WHILE a ground atom $p(t_1, \cdots, t_n)$ can

be found in Stratum
assign true to $p(t_1, \cdots, t_n)$;
substitute $t_1, \cdots, t_n$ for logical variables $X_1, \cdots, X_n$ in a rule containing $p(X_1, \cdots, X_n)$ in the program;
remove fact $p(t_1, \cdots, t_n)$ from the program;
remove $p(t_1, \cdots, t_n)$ from premise of every other rule in which $p(t_1, \cdots, t_n)$ appears;
remove all modified rules which have $\neg p(t_1, \cdots, t_n)$ in their premise;

ENDWHILE;
assign false to all remaining positive literals L associated with Stratum;
substitute false for L in every rule in which L appears;
remove all modified rules which have L in their premise;

ENDFOR

With this approach, we assume that any literal is assigned the truth value false unless it is forced to be true. This is a form of Closed World Assumption(CWA). Unlike the CWA for a logic program presented in Reiter [1978], this CWA is implemented sequentially.

When the condition 1 and condition 2 are satisfied, the result of executing the above WHILE loop is a model(since truth assignments are made so that no rules are violated). It is also a minimal model, since converting truth values from true to false for any subset of literals will no longer yield a model. This model is a supported model since literals are assigned true only as a result of applying a rule whose premise is already true.

Running this loop for Program2, we have a model in which child(lee) and innocent(lee) are true and liar(lee) is false.

The concept of minimal supported model is a natural generalization of least model to stratified programs and is well-defined. One of the drawbacks of this semantics is that only a restricted class of programs can be stratified. There are many useful programs which cannot be stratified and this seamntics can not be applied[Przymusinski, 1989; van Gelder, 1989]. Przymusinski extended the stratified program to locally stratified program by stratifying ground atoms in the Herbrand base instead of predicates. A program is locally stratified if for each ground rule $A \leftarrow L_1, \cdots, L_{m-1}$, the local stratum of A is strictly greater than the local stratum of each ground atom of negative literal in $L_1, \cdots, L_m$ and the local stratum of A is greater than or equal to the local stratum of each ground atom in $L_1, \cdots, L_m$. The class of locally stratified programs is larger than the class of stratified programs and there are natural and useful programs which cannot be stratified but can be locally stratified[Przymusinski, 1988].

Locally stratified programs have a well-defined declarative semantics, called a perfect model, which is minimal and unique for each locally stratified program. A perfect model is

a model of a program in which each ground atom in a lower local stratum is minimized (falsified) before each ground atom in a higher local stratum is minimized. For stratified programs, the perfect model is identical with the minimal supported model and hence the perfect model semantics is more powerful than the semantics of the minimal supported model. On the other hand, although it is quite easy to check whether a program is stratified, it is very hard to check local stratifiability because when functions are present a program may have infinite local stratification. It has been shown that it is not decidable in general whether a program is locally stratified or not[van Gelder, 1989].

3. Weakly Perfect Model Semantics

The weakly perfect model semantics, which is an extension of perfect model semantics, is based on the dynamic decomposition of the program (grounded) into strata and its semantics is based on the iterated least model [Prysmusinska, 1988]. In the Herbrand instantiation of a program (program in which all the variables are substituted by ground terms), many irrelevant relations and clauses are present and they may cause local unstratifiability. The main idea of this semantics is to remove irrelevant relations and clauses from the grounded program.

Although negative recursion is not allowed in stratified and locally stratified programs, a restricted form of negative recursion is allowed in this semantics by the notion of a component. Two ground atoms A and B are in the same component C if there are at least two ground rules $A \leftarrow L_1, \cdots, L_n$. where L_i is $\neg B$ for some i and $B \leftarrow L_1, \cdots, L_n$. where L_i is $\neg A$ for some i. Then there is an order relation between components such that a component C_1 is higher than C_2 if C_1 contains a ground atom which has higher priority(lower local stratum) than a ground atom in C_2. A component is maximal if no other component is higher than that. A component is trivial if it consists of a single atom and there is no rule $A \leftarrow L_1, \cdots, L_n$. in which L is $\neg A$ for some i. The bottom stratum S(P) of a program P is the union of all maximal components of P. The bottom layer L(P) of P is the set of all the clauses from P whose heads belongs to S(P). A reduction of a program P with an interpretation I is a new program P/I created by performing the following reductions:

- removing from P all clauses which contain a false premise in I;
- removing from all remaining clauses premises which are true in I;
- removing from resulting program all non-unit clauses whose heads appear as unit clauses in the program.

The construction of a weakly perfect model is as follows.

Take any program $P=P_0$ and $M_0=\langle \phi, \phi \rangle$ be an interpretation in which any ground atom is neither true nor false.

Let $P_1 = P_0/M_0$, the reduction of P_0 with respect to M_0. Find the least model T_1 of the bottom layer $L(P_1)$ of P_1 and let $M_1=\langle T_1, F_1 \rangle$

where $F_1 = S(P_1) - T_1$.

Let $P_2 = P_1/M_1$, find the least model T_2 of $L(P_2)$, and let $M_2 = \langle T_1 \cup T_2, F_1 \cup F_2 \rangle$ where $F_2 = S(P_2) - T_2$.

Continue this process until either P_k is empty, in which case $M_p = M_k$, or, otherwise, until either $S(P_k)$ is empty or $L(P_k)$ does not have a least model, in which case the weakly perfect model is undefined.

Program 3:

team(kim,park).
senior(X)←team(X,Y), ¬senior(Y).

The first rule represents a team of workers and the second rule means that if the second worker is not senior worker then the first worker is senior worker of the team.

Every locally stratified program has a weakly perfect model and it coincides with the perfect model. Weakly stratified programs which are more general than locally stratified programs, have a unique weakly perfect model. A program is weakly stratified if every maximal component in each stratum is trivial and it has a weakly perfect model. But, (1) programs with a weakly perfect model may not be weakly stratified, (2) the class of programs which have a weakly perfect model is not clearly defined, and (3) some reasonable programs which still do not have weakly perfect models.

4. Stable Model Semantics

Gelfond and Lifschitz developed a stable model semantics for General Programs which is based on a form of non-monotonic reasoning, autoepistemic logic[Gelfond, 1987; Moore, 1985]. An interpretation of a program is a stable model if it reproduces itself in a natural transformation process, called stable transformation. That is, if what we currently know about the context of the program is not changing with respect to a stability operator of the program, then it is the intended meaning of the program. A stability operator S_p is defined as follows:

Given a ground program P and an interpretation I, a reduction of P with respect to I is a new program P/I obtained from P by performing the following reductions:

- removing from P all clauses containing a negative premise which is false in I;
- removing from the remaining clauses those negative literals which are true in I.

Since the resulting P/I is a Positive Program it has a unique least model J. The least model J is defined as the outcome of the operator S_p on I, ie. $S_p(I) = J$. Note that the number of true ground atoms in a model M is greater than the number of true ground atoms in $S_p(M)$ because M is a model of P/I and $S_p(M)$ is a least model of P/I[van Gelder, 1989]. The least fixed point of S_p of a program P is always both a minimal model of P and a stable model of P.

The stable model is an extension of the perfect model semantics. For locally stratified programs the stable model is unique and it coincides with the perfect model. Furthermore, for a wider class of programs than

locally stratified programs, stable models are unique and match with the intended meaning of the programs.

However, there are logic programs with reasonable intended meaning which either does not have a unique stable model or have a stable model which does not match with the intended meaning.

Program 4:

success← ¬failure.
failure← ¬success.

Program4 has two stable models which are not an intended meaning. Program {success← ¬failure. } has no stable model. So, the stable model semantics is defined only for restricted programs but the exact class of programs which have a unique stable model is not found. Another problem involved in this semantics is that no constructive method for building a stable model for a program has not been found. One way is to find all minimal models and test each one for the stablilty transformation. It has been shown that determining whether a program has a stable model is NP-complete[van Gelder, 1989].

5. Well-founded Model Semantics

Since every attempt to find a suitable declarative semantics which is categorical (each ground atom in Herbrand base is true or false) for a general logic program has been failed, a declarative semantics in 3-valued logic seems appropriate. van Gelder, Ross, and Schlipf introduced a well-founded semantics which can be applied for any program[van Gelder, 1990]. The well-founded partial model of a program P, which is the intended meaning of P, is defined in terms of a transformation that involves unfounded sets which provides the basis for negative conclusions in this semantics. Given a program P and an interpretation I, a set U in Herbrand base is an unfounded set if each atom p in U satisfies the following condition:

For each ground rule in P with p as head, one of the following holds:

- a premise of the rule is false in I.
- a positive premise of the rule occurs in U.

A literal that makes one of the conditions is called the witness of unusability of the rule. The union of all unfounded sets with respect to a given I is also unfounded and is called the greatest unfounded set[van Gelder, 1989].

A well-founded partial model is a fixed point of a natural operator W_p defined as follows:

Given a program P and an interpretation I:

- T_p is a van Emden and Kowalski operator.
- $U_p(I)$ is the greatest unfounded set of P with respect to I.
- $W_p = T_p(I) \cup \neg U_p(I)$.

The operator W_p is monotonic in 3-valued logic because the number of ground atoms with known truth values in $W_p(I)$ is not less than the number of ground atoms with known truth values in I for any I. Thus W_p has a least fixed point which is a well-founded

partial model of P.

This semantics can also be characterized as an alternating fixed point of another operator which is a variant of a stability transformation operator. Given a program P and an interpretation I, ¬I is a negated complement of I in the Herbrand base(Hb) and an operator $S_p(\neg I)$ is defined as follows:

Let $P' = P \cup \neg I$ then $S_p(\neg I)$ is defined as a least fixed point of T_p', the set of ground atoms proved from P and ¬I. The negated complement of $S_p(\neg I)$ or $\neg(Hb-S_p(\neg I))$, is denoted by $\neg S_p(\neg I)$. Now the alternating operator with respect to an interpretation I, $A_p(\neg I) = \neg S_p(\neg S_p(\neg I))$. Ap operator is monotonic in 3-valued logic and the least fixed point of Ap is the alternating fixed point of P. An alternating fixed point ¬I consists of ground atoms which are false in the corresponding well-founded partial model and $S_p(\neg I)$ is the set of atoms which are true in the corresponding well-founded partial model. Any ground atom in the Herbrand base that is neither in ¬I nor in $S_p(\neg I)$ is unknown in the well-founded partial model. This alternating fixed point operator can be used to build a well-founded partial model constructively. When a well-founded partial model is categorical it is called a well-founded model. In this case a well-founded model coincides with both a perfect model and a weakly perfect model if they exist. So, the well-founded semantics is an extension of perfect model and weakly perfect model semantics. And a well-founded model, if it exists, coincides with one of the stable models. However, a unique stable model may not be a well-founded model. Furthermore, a well-founded partial model exists for every logic program and it is the same as the results by non-monotonic reasoning approaches[Prysimusinski, 1989]. The procedural semantics corresponding to well-founded model semantics, however, has not been found.

Ⅳ. Practical Model Semantics

1. Conditions of logical variables

The role of logical variables in rules is important to interpret the meaning of a program. Difficulties arise when we try to state general universally quantified (i.e. "for all…") facts or try to reach universally quantified conclusions to rules. The following two statements each meets the definition of a logic programming statement but can be a source of difficulty in analyzing a program's meaning:

```
potential_patron(Y, X)← ¬taken_care_
                          of(X), baby(X).
watches(Everyone, tv).
```

These two statements violate Condition 1:

Condition 1(Covering Axiom): Any variable appearing in the conclusion of a rule must appear in a positive literal in the premise of the rule.

The intended meaning of the first rule is "if X is a baby and it is not taken care of then

any Y is a potential patron of X". Difficulty involved in this rule is that because Y is universally quantified Y can be substituted with any object in the program ie, anythiong is a potential patron of X which is not intended. Since a fact is a rule with an empty premise, when the Covering Axiom is satisfied, the program cannot contain assertions with variables, e.g. "watches(Everyone, tv)." Notice also that the simple rule "female(X) :- ¬male(X)" violates Condition 1.

Another condition concerning the role of logical variable is:

Condition 2(Allowedness): Any variable appearing in a negative literal in the premise of a rule must appear in a positive literal in the same premise. Allowedness is necessary if forward reasoning processor is used to answer a query in a program. Without this condition, we may have a query with negative literal containg variables, called floundering.

When Conditions 1 and 2 are satisfied, a logic program is well-behaved in the following sense. If given a ground query (like "?-flies(tweety)."), any rule used in a forward reasoning process will produce a ground conclusion. Also, by reordering literals in the premise of a rule, any negative literal is ground before being unified. Although negation must always be used with caution, it is generally preferable to be constrained to proving ground negative literals like " ¬flies (tweety)" rather than unground literals like " ¬flies(X)."

2. Practical model semantics

In this section we introduce a new semantics for a program. Unlike other approaches in semantics, this semantics is constructed by alternating process of substitution for logical variables and finding a model of subset of a program. Most of difficulties in finding an appropreate semantics of a program are caused by irrelevant ground rules (rules without practical meaning like, senior(kim) ←team(kim, kim), ¬senior(kim).) in Herbrand instantiation of the program. For example, Herbrand instantiation of Program3 has three irrelevant ground rules and because of these rules Program3 can not be locally stratified. If we force these irrelevant ground rules from being generated then suitable semantics of a program may be found easily. In weakly perfect model semantics[Przymusinski, 1988], all irrelevant rules are removed from Herbrand instantiation of the program in the program reduction step. In this semantics irrelevant ground rules are not generated in the substitution process and only generated ground rules are considered in finding semantics of a program.

In general Herbrand instantiation of a program is large and it contains a lot of irrelevant ground rules. In order to prevent irrelevant ground rules from being generated, we instantiate only ground rules with premise in which positive ground literals are true and negative ground rules are false. Let us define a couple of terms which will be frequently ued

in the following discussion.

Definition : Given an interpretation I, ground part of a program P is a set of ground rules $A \leftarrow L_1, \cdots, L_p, L_{p+1}, \cdots, L_{p+n}$. which can be generated from P by substituting logical variables and $L_1, \cdots, L_p$ are true in I.

Definition : A ground rule is useful in Interpretation I, if all positive literals in the premise are true in I. A ground rule is unuseful if one of positive literals in its premise is false in I.

If a ground rule is useful then the head of the rule can be assigned to be true in I. But if a ground rule is unuseful then the truth value of its head can not be determined in I. Thus, given an interpretation I and ground part of a program which contains only useful ground rules, by making only forced ground atoms true and remaining ground atoms false, we can find a model of the ground part. In this way the number of ground atoms assigned to be true are minimized. Most of anomalies in semantics of programs which can not be locally stratified stem from using unuseful ground rules in Herbrand instantiation of a program.

Therefore, if a ground rule is unuseful in an appropriate model of the program then we can ignore the rule in the process of finding the model. These unuseful rules may not be instantiated in query answering. This semantics is based on this idea. With this strategy we may define an easy to understand and intuitive semantics of a large class of programs in the following manner.

When condition 1 and condition 2 are satisfied, the construction of practical model is as follows:

Given a program P, G_i is a ground part of P using interpretation M_i. If G_i is ϕ or not locally stratified, then no practical model found. If G_i is locally stratified then find a perfect model M_i of G_i using the following process.

FOR Local_Stratum=1 to S DO
  WHILE a ground atom $p(t_1, \cdots, t_n)$ can be found in Local_Stratum
  assign true to $p(t_1, \cdots, t_n)$;
    substitute $t_1, \cdots, t_n$ for logical variables $X_1, \cdots, X_n$ in a rule containing $p(X_1, \cdots, X_n)$ in the ground_part;
    remove fact $p(t_1, \cdots, t_n)$ from the ground_part;
    remove $p(t_1, \cdots, t_n)$ from premise of every other rule in which $p(t_1, \cdots, t_n)$ appears;
    remove all modified rules which have $\neg p(t_1, \cdots, t_n)$ in their premise;
  ENDWHILE;
  assign false to all remaining positive literals L associated with Local_Stratum;
  substitute false for L in every rule in which L appears;
  remove all modified rules which have L in their premise;
ENDFOR
Starting from $M_1=\phi$, continue the process

until $G_n=G_{n+1}$ or $M_n=M_{n+1}$. If G_n is locally stratified then it has a perfect model M_n which is a practical model of P. If G_n is not locally stratified then a practical model of P is not defined.

This model of G_n is a minimal model of P and it is unique when condition 1 and condition 2 are satisfied.

Lemma A ground rule which is not instantiated by the above process is unuseful in M.

Proof: Assume that a ground rule r is not instantiated by the process and it is useful. Then its body is true in M. Positive part of the premise of the rule must be true in M. Thus the rule must be instantiated in the process. Constradiction.

Theorem M is a model of the program P when condition 1 and condition 2 are satisfied.

Proof : Suppose M is not a model of P then there is a ground rule r which is not true in M. If r is in G_n then r is true in M because M is a perfect model of G_n. If r is not in G_n. We can rewrite r as $A \leftarrow L_1, \cdots, L_p, L_{p+1}, \cdots, L_{p+n}$. where p represents numer of positive literals in r and n represents number of negative literals. L_i in $L_1, \cdots, L_p$ is false in M for some i otherwise r is in G_n. Thus r is true in M whatever truth value A and $L_{p+1}, \cdots, L_{p+n}$ have. Contradiction.

M is also a minimal model of P. M is a perfect model of G_n and a perfect model is a minimal model. But G_n is a proper subset of P. Thus M is a minimal model of P.

In Program3, Program3=P. $M_1=\langle \phi, \phi \rangle$. G_1 ={team(kim, park).}. $M_1=<${team(kim, park)}, {team(kim, kim), team(park, park), team(park, kim)}>. G_2={team(kim, park), senior(kim)←team(kim, park), ¬ senior (park)}. $M_2=<${team(kim, park), senior (kim)}, {team(kim, kim), team(park, park), team(park, kim), senior(park)}⟩. G_3={team (kim, park), senior(kim)←team(kim, park), ¬ senior(park)}. $G_2=G_3$ and $M_2=M_3$. Thus the practical model of Program3 is M_2.

As shown above, the practical model semantics is easier to understand than well-founded model semantics which uses well-founded set and unfounded set of ground literals. The process of finding the practical model semantics alternately uses instantiation of ground rules and finding a perfect model of the ground part which are rather simple. And also this model may be found with less computation than others because only useful ground part, which is only a small subset of Herbrand base of the program, is instantiated and considered to find this model. In other approaches the all ground rules in Herbrand base are searched and evaluated to find an appropreate model. Although this semantics has very clear and intuitive meaning, it should be used with care. When condition 1 and condition 2 are not satisfied a model can not be found. However, if general assertions are not allowed and negative subgoals are to be ground then these conditions are automati-

cally satisfied.

V. Conclusion

Logic programming and deductive databases has gained widespread attention because of the popularity of Prolog as a programming language for knowledge-based applications. To enhance the expressive power of logic programming negation is necessary but use of negation in logic program complicates the meaning of the program.

In this paper we have described various proposed semantics for deductive databases and logic programs and discussed their mutual relations. We have particularly emphasized recent research work in this area which appears to be very significant and lead to a change of perspective. There are two essentially different 2-valued model-theoretic semantics of deductive databases and logic programs, both of which are closely related to non-monotonic formalisms. One of them is the stable model semantics, which is based on autoepistemic logic or default theory. The other is the weakly model semantics, based on circumscription. There is a unique 3-valued model-theoretic semantics, namely the well-founded semantics, which is equivalent to suitable forms of all 3-valued non-monotonic formalisms and appears to be the most adequate semantics for logic programs and deductive databases.

We also suggested a semantics for a practical use. This practical model semantics is easy to find and intuitive. The process described above generates a model of a program which is minimal and unique. Any ground rule in a program is intended to be used to prove new fact which is not in extentional database. Thus if a ground rule is not useful then we can ignore the rule. In this semantics unuseful rules are ignored in the process of finding a semantics of a program.

The class of programs having this semantics is wider than locally stratified programs but narrower than well-founded model semantics. And for weakly stratified programs, the practical model is the same as weakly perfect model. Although this semantics has some advantages, the class of programs which have the practical model semantics is not clearly determined.

A compiler or interpreter using this semantics willl have more expressive power and a query or rule which is closer to natural language can be used in logic program and deductive databases.

REFERENCES

Apt, K. and Van Emden. "Contributions to the theory of logicprogramming", *Journal of the ACM*, 29 : 841-862, 1982.

Apt, K. Blaire, H. and Walker. A. "Towards a theory of declarative knowledge", In J. Minker, editor, *Foundations of Deductive*

Database and Logic Programming, pages 89-142, Morgan Kaufmann, Los Altos, CA., 1988.

Chang C. and Lee. R.C. *Symbolic Logic and Mechanical Theorem Proving.*, Academic Press, New York, 1973.

Clark. K.L. "Negation as failure", In H. Gallaire and J.Minker, editors, *Logic and Data Bases*, pages 293-322, Plenum Press, New York, 1978.

Bell, C.E. and Dae Yong Lee. "Analysis of the behavior of logic-based computation for deductive databases and default reasoning", *Decision Support Systems*, 8 : 517-535, 1992.

Gallaire, H. and Minker, J. *Logic and databases*, Plenum Press, New York, 1978.

Gallaire, H. Minker, J. and Nicolas. J. "Logic and databases : a deductive approach", ACM Computing Surveys, 16 : 153-185, 1984.

Gelfond, M. and Lifschitz, V. "The stable model semantics for logic programming", In R. Kowalski and K. Bowen, editors, *Proceedings of the Fifth Logic Programming Symposium*, pages 1070-1080, Association for Logic Programming, MIT Press, Cambridge, Mass., 1988.

Kowalski. R. *Logic for Problem Solving.* North Holland, New York, 1979.

Kowalski. R. "Logic as a Computer Language", In K.L. Clark and S. Tarnlund, editors, *Logic Programming*, pages 3-18, Academic Press Inc., New York, 1982

Lifschitz. V. "On the declarative semantics of logic programs with negation", In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 177-192, Morgan Kaufmann, Los Altos, CA., 1988.

Lloyd. J.W. *Foundations of Logic Programming.* Springer Verlag, New York, N.Y., second edition, 1987.

Minker. J. *Foundations of Deductive Databases and Logic Programming.* Morgan Kaufmann, Los Altos, CA., 1988.

Minker. J. "Perspectives in deductive databases", *Journal of Logic Programming*, 5 (1) : 33-60, 1988.

Naqvi. S.A. "A logic for negation in database systems", In J. Minker, editor, *Proceedings of the Workshop on Foundations of Deductive Databases and Logic Programming*, Washing-ton, D.C., pages 378-387, August 1986.

Przymusinska H. and Przymusinski. T. "Weakly perfect model semantics for logic programs", In R. Kowalski and K. Bowen, editors, *Proceedings of the Fifth Logic Programming Symposium*, pages 1106-

1122, Association for Logic Programming, MIT Press, Cambridge, Mass., 1988.

Przymusinski. T. "On the declarative and procedural semantics of logic programs", *Journal of Automated Reasoning*, 5 : 167-205, 1989.

Reiter. R. "On closed-world data bases", In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 293-322, Plenum Press, New York, 1978.

Reiter. R. "Towards a logical reconstruction of relational database theory", In M. Brodie and J. Mylopoulos, editors, *On Conceptual Model-ling*, pages 341-348, Morgan Kaufmann, 1989.

Shepherdson. J. "Negation in logic programming", In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 19-88, Morgan Kaufmann, Los Altos, CA., 1988.

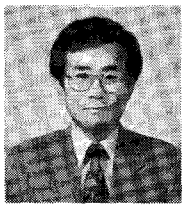
van Emden M. and Kowalski. R. "The semantics of predicate logic as a programming language", *Journal of the ACM*, 23(4) : 733-742, 1976.

van Gelder. A. "Negation as failure using tight derivations for general logic programs", In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 149-176, Morgan Kaufmann, Los Altos, CA., 1988.

van Gelder. A. "The alternating fixpoint of logic programs with negation", *In Proceedings of the Symposium on Principles of Database Systems*, pages 1-10, ACM SIGACT-SIGMOD, 1989.

van Gelder, A. Ross, K.A. and Schlipf. J. S. "The well-founded semantics for general logic programs", *Journal of the ACM*, 1990.

◇ 저자소개 ◇



저자 이대용은 1990년 Univ. of Iowa에서 MIS 박사학위를 취득하고 조선대학교 경영학과 조교수로 재직중이다. 그는 Decision Support System 등에 논문을 발표했으며, 관심분야는 Logic Programming, Deductive databases, Expert Systems, ILP와 Logic과의 관계 등이다.