Assigning an Appropriate Meaning to Database Logic with Negation

Jeffrey D. Ullman
Department of Computer Science
Stanford University, Stanford CA, USA

ABSTRACT

Deductive database systems — that is, database systems with a query language based on logical rules — must allow negated subgoals in rules to express an adequate range of queries. Adherence to classical deductive logic rarely offers the intuitively correct meaning of the rules. Thus, a variety of approaches to defining the "right" meaning of such rules have been developed. In this paper we survey the principal approaches, including stratified negation, well-founded negation, stable-model semantics, and modularly stratified semantics.

1. Motivation

Database query languages based on logic in some form are a promising trend in the development of modern database systems. Building on SQL, a language whose logical nature is hidden by the syntax, logical query languages preserve the declarative "say what you want, not how to get it" nature of SQL, while extending the expressiveness of the language beyond what SQL provides. See Ramakrishnan and Ullman [1993] for a survey of languages and systems based on the logical or "deductive" approach.

Simple forms of logic, such as Horn-clause rules (if-then rules), have a unique natural meaning, which is the least fixpoint, or minimal model of the rules. However, many interesting queries require us to use more general forms of rules, such as those in which one or more subgoals appear negatively. It is on these extensions that we focus.

The problem with logic that involves negative subgoals is that there is rarely a unique minimal model. In conventional logic, the rules would be said to "mean" whatever can logically be inferred from these rules and nothing more. That is the same as saying the "meaning" is the intersection of all the minimal models. However, that intersection is rarely what the programmer intuitively expects the rules to "mean." Thus, research in deductive databases with negation can be seen as a series of proposals to define one particular minimal model as the meaning of rules with negation. To some degree, this program of research has been successful, and there are important classes of logical rules with negation for which there is a widely accepted meaning. In other cases, the proper choice of meaning is less clear.

The subject of negation in deductive databases is related to the matter of nonmonotonic reasoning as discussed in the literature of Artificial Intelligence. There is a tendency in the database community to value more the issues of efficiency, i.e., how fast we can answer queries using a given definition of the "correct" minimal model. The AI approaches, on the other hand, tend to focus on the correctness of the model, regardless of its tractability. The difference will be emphasized when we compare well-founded and stable-model approaches below.

2. Introduction

Logical rules are usually expressed in the form

$$\text{Head} \leftarrow \text{Body}$$

where the head is an atomic formula (predicate with arguments) and the body is the logical AND of literals, i.e., atomic formulas or their negation. Each literal is called a *subgoal*. A collection of rules is often called a *logic program*, or just *program*.

Predicates are divided into two classes:

Work supported by NSF grant IRI–92–23405 and ARO grant DAAL03–91–G–0177.

1. EDB (extensional database) predicates are stored as relations in the database. If p is an EDB predicate, then there will be a corresponding relation, say P, in the database, and $p(a_1, \ldots, a_k)$ is true if and only if there is a tuple $(a_1, \ldots, a_k)$ in relation P. The EDB tuples are sometimes called *data*.
2. IDB (intensional database) predicates, which are defined by the rules. Only IDB predicates can appear in the head of a rule. Both IDB and EDB predicates can appear in the body.

Example 1: Here are rules for computing the transitive closure of a graph. That is, *arc* is an EDB predicate, and $arc(a, b)$ means that there is an arc from node a to node b. IDB predicate *path* represents path facts; $path(a, b)$ is intended to be true if and only if there is a path of one or more arcs from node a to node b.

```
r1: path(X,Y) ← arc(X,Y)
r2: path(X,Y) ← arc(X,Z), path(Z,Y)
```

We use Prolog-style notation, where capital letters begin variables and lower-case letters begin constants. The ← operator means "if," and the comma between subgoals in rule r_2 is interpreted as "and." Thus, rule r_1 says "there is a path from X to Y if there is an arc from X to Y," while r_2 says "there is a path from X to Y if there is some node Z such that there is an arc from X to Z and a path from Z to Y. □

Given relations for the EDB predicates of a logic program, there are certain *models* or *fixpoints* for the rules. *Interpretations* are selections of *ground atoms* or *facts* for the IDB predicates; the latter consist of an IDB predicate with constant arguments. An interpretation is a model (with respect to a given selection of EDB facts) for the program if whenever constants are substituted for the variables, the rules become true. Since rules are of an if-then form, the only way a rule can become false is if all the subgoals are true but the head is false. Thus, in Example 1, r_1 forces us to include in every model for a given EDB the fact $path(a, b)$ whenever the fact $arc(a, b)$ is in the EDB. Rule r_2 forces us to add $path(a, c)$ whenever our model has a fact $path(b, c)$ and the fact $arc(a, b)$ is in the given EDB.

However, those are all the facts that need to be in the IDB. We can compute the IDB by starting with the given EDB and adding to the IDB all and only those facts that we are forced to add because of the rules. This model is *minimal* in the sense that any proper subset either is missing an EDB fact that is given or it fails to be a model because there is some substitution of constants for variables that makes all the subgoals true but makes the head false. In Example 1, the minimal model is the expected set of path facts; $path(a, b)$ is true exactly when a path from a to b exists in the graph with the set of arcs that are given by the EDB relation *arc*.

3. Rules with Negated Subgoals

When we have a collection of *Horn clauses* (rules without negated subgoals), we are assured of a unique minimal model, and this model is universally accepted as the meaning of the rules. However, this form of rule is often too limited, regardless of whether one is interested in logic programming as in Prolog, or in deductive database systems as is our focus. As soon as we introduce negation in rules, we no longer have a guarantee of a unique minimal model, and in fact it is normal to have more than one.

Example 2: There are two bus lines, *red* and *blue*, each of which runs buses between pairs of cities. The predicate $blue(X, Y)$ is true if the blue line runs a bus between city X and city Y, while $red(X, Y)$ has the corresponding meaning for the red line. The president of the red line wants to find out where red has a monopoly, that is, a pair of cities such that red runs a bus between them, but on blue buses you cannot even travel from X to Y through a sequence of intermediate cities. The following rules express the idea.†

```
(1) bluePath(X,Y) ← blue(X,Y)
(2) bluePath(X,Y) ← bluePath(X,Z),
            bluePath(Z,Y)

(3) monopoly(X,Y) ← red(X,Y),
            not bluePath(X,Y)
```

† The bus lines are directed arcs, while we might in practice expect them to be undirected edges. However, this liberty makes the example clearer and can be carried over to the more realistic case of undirected edges.

Suppose further that the data (relations) for predicates *red* and *blue* are $blue(1,2)$, $red(1,2)$, and $red(2,3)$, as suggested by Fig. 1.

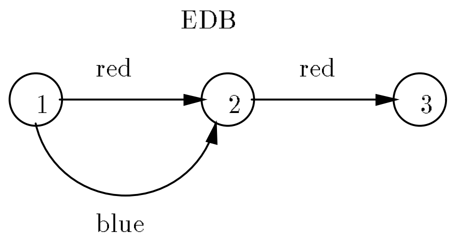

Fig. 1. EDB for Example 2.

Interestingly, there are two minimal models for the EDB of Fig. 1. The IDB components of these models are listed below.

(A)	(B)
$bluePath(1,2)$	$bluePath(1,2)$
$monopoly(2,3)$	$bluePath(2,3)$
	$bluePath(1,3)$

(A) makes sense. The only blue path is the one that follows from the data by rule (1). The *monopoly* fact then follows by rule (3).

(B) does not "make sense." It involves $bluePath$ facts that do not appear to come from anywhere, namely $bluePath(2,3)$ and $bluePath(1,3)$. However, it is also a minimal model, in that

1. When you make any substitution of constants for X, Y, and (if necessary) Z, rules (1)–(3) are true if the true ground atomic formulas are those listed in (B), plus the given data.
2. If you delete one or more facts from (B), point (1) above no longer holds. □

4. Approaches to Selection of Models

As we see from Example 2, the fundamental question of coping with negation, or "nonmonotonic reasoning," is to select the appropriate model from all those that satisfy the given logic program and data. That is, we need to develop a *model preference theory* (Shoham [1987]).

When the rules involve no negated subgoals, there is no argument whatsoever that the preferred model is the least fixpoint. But when there are negated subgoals, there is at least some question as to the preferred model.

All approaches to defining a preferred model in the presence of negation eschew "classical negation" in favor of some version of *negation as failure* (Clark [1978]). In classical negation, the rules "$p \leftarrow \neg q$" and "$q \leftarrow \neg p$" are each equivalent to $p \vee q$. In negation-as-failure approaches, the direction of the implication is given significance. For example, from $p \leftarrow \neg q$ and $\neg q$ we might readily infer p, but from $p \leftarrow \neg q$ and $\neg p$ we might not infer q.

5. Stratified Logic

The least controversy surrounds *stratified negation*, where there is no recursion involving negated subgoals. This idea was arrived at independently (and in the same year) by Van Gelder [1986], Apt, Blair, and Walker [1988], and Naqvi [1986]. The idea is that when a logic program is stratified, we can find an order for the predicates, so we may "solve" for a predicate p only after we have solved for all the predicates on which p depends negatively.

For instance, the program of Example 2 is stratified. Although *monopoly* depends negatively on *blue-Path*, the latter does not depend at all on *monopoly*. Thus, we can find the relation for $bluePath$ using only rules (1) and (2) of Example 2, and then use rule (3) to find the relation for *monopoly*. This process, with the *red* and *blue* data mentioned in Example 2, yields model (A), confirming our intuition that this model is "right." The result of computing the predicates in this ways is often called the *perfect* model.

An alternative view of what is going on concerns *circumscription* (McCarthy [1980]), which is an approach to coping with negation that allows us to declare that the only facts true for some predicate are those that follow from given rules. In Example 2, the stratified negation approach allows us first to circumscribe the predicate *bluePath*, in effect declaring that $bluePath(X, Y)$ is true only when it so follows from rules (1) and (2) and the given *blue* data. We can then assert that $\neg bluePath(X, Y)$ is true for all other pairs of values X and Y. It is these negated facts that are used in rule (3) to infer *monopoly* facts.

Przymusinska and Przymusinski [1989] extended the idea of stratification to *locally stratified* programs and data, where predicates may depend negatively on themselves as long as when the rules of the logic program are instantiated by constants no cycles occur. A program can be locally stratified for one EDB but not for another. Surely every stratified logic program is locally stratified regardless of the data.

Example 3: Let us introduce an important paradigm problem, the "win" rule:

```
    win(X) ← move(X,Y), not win(Y)
```

This rule describes a board game. We suppose that $move(X, Y)$ is true if and only if it is legal to move from position X to position Y in one move. We also suppose that the game is won by stalemating your opponent; that is, you lose if you are presented with a position from which there is no legal move. The rule says that position X is a win if there is a choice of move to some position Y, and Y is not a win.

Clearly, *win* depends negatively, on itself, so this rule is not stratified. However, suppose *move* is acyclic, i.e., if you can move from X to Y then there is no sequence of moves leading from Y back to X. Then if we instantiate the rule in all possible ways, there is no way the fact $win(a)$, for a particular board a, can depend negatively on itself. Put another way, for an acyclic *move* (Nim is an example of such a game), we can decide whether $win(a)$ is true by expanding the game tree until we reach stalemate positions. There is no way a sequence of moves can avoid stalemate, as long as the number of positions is finite and the *move* predicate represents an acyclic graph. Thus, the "win" rule is locally stratified provided *move* is acyclic. $\square$

6. Well-Founded Semantics

More recently, there have been two competing thrusts ("well-founded" and "stable" models) that attempt to provide a sensible preferred model for logic programs that are not locally stratified. The *well-founded semantics* of Van Gelder, Ross, and Schlipf [1991] (first published in 1988) is a 3-valued semantics, where some ground atoms are declared true, others are declared false, and the remainder are "unknown."

We shall give a definition of the well-founded semantics shortly. However, to focus on the importance of 3-valued models, let us first observe that in Example 3, the "win" rule, the well-founded model is the intuitively correct one, even when *move* has cycles. That is, for each position a, $win(a)$ is true if best play yields a win from position a; $win(a)$ is false if a player forced to move from position a is forced to lose with best play on both sides, and $win(a)$ has truth value "unknown" if best play leads to a draw from position a.

Construction of the well-founded model for a logic program and data proceeds as follows. First, we in principle instantiate all the rules by constants in all possible ways (in practice, there are more efficient approaches, such as those of Przymusinski [1989], Ross [1989], and Van Gelder [1989], Kerasit and Pugin [1988], and Morishita [1993]. We start with no positive or negative ground atoms in our model, except that the facts in the EDB, which are positive ground atoms, are in the model. We then add positive and negative facts in rounds using two kinds of inference.

a) We consider all instantiations of the rules. If each of the subgoals of a rule is true in the current model, then we add the head of the instantiated rule to the model. This fact is always a positive ground atom.

b) Now we compute the largest *unfounded set* of ground atoms. Intuitively, an unfounded set U is a collection of positive ground atoms such that each instantiated rule whose head is in U has either

    i) A subgoal whose negation is in the model constructed so far. In this case, we know the instantiated rule can never be used to infer its head.

ii) A subgoal in U.

Since each fact in U can only be inferred by an instantiated rule that has another member of U as a subgoal, no member of U can ever be the first of them to be proved. Thus, we shall never infer any of the members of the unfounded set using ordinary, body-to-head deduction as described in step (a). Since we shall never prove a member of U to be in the model, we take a metalogical leap and place each of their negations in the model. In this manner we infer some negative atoms.

We shall give three examples of this process to give an idea of some of the ways inference proceeds. The first two examples deal with circuits, where the well founded model has a natural interpretation. The third is an abstract example that shows how the above two types of inference (a) and (b) can alternate several times before the well-founded model is reached.

Example 4: In this example, we shall represent circuits consisting of an unusual sort of logic gate, with one positive input and one negative input. If a gate has positive input X and negative input Y, then its output is 1 or "true" if and only if X is 1 and Y is 0 ("false"). There is an EDB predicate $g(X, Y, Z)$ that says there is a gate of this type with positive input X, negative input Y, and output Z. We may think of the inputs and outputs as being terminals or wire nets. There is also an EDB predicate t_0 that is true of those input terminals that are set externally to 1. Input terminals that are set to 0 do not appear in t_0.

The IDB predicate is t. The intended significance of the positive ground atom $t(a)$ being in the model is that the circuit value of terminal a is 1. If $\neg t(a)$ is in the model, then the value of terminal a is 0. What if the value that terminal a has is ambiguous; either it depends on a critical race in the circuit or oscillates in normal circuit operation? Then we expect $t(a)$ to have the third, "unknown" value of the three-valued well-founded semantics.

The following are the rules that define the operation of the gates.

```
t(Z) ← t0(Z)
t(Z) ← g(X,Y,Z), t(X), not t(Y)
```

The data in the EDB will consist of the following facts: $t0(2)$, $g(5,1,3)$, $g(1,2,4)$, $g(3,4,5)$, representing the circuit of Fig. 2 with only the second input set to "true."

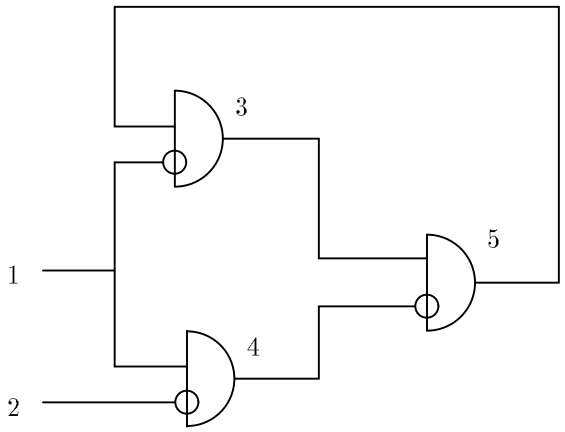

Fig. 2. Circuit for Example 4.

Following are all instantiations without a known false subgoal (initially, we can only know that EDB facts *not* in the database are false):

```
t(2) ← t0(2)
t(3) ← g(5,1,3), t(5), not t(1)
t(4) ← g(1,2,4), t(1), not t(2)
t(5) ← g(3,4,5), t(3), not t(4)
```

For convenience, once a subgoal has been found true, we eliminate it from the body of the instantiated rule. That way, we know we can make a type (a) inference when the body of the rule becomes empty. If we eliminate known true subgoals from bodies, we get:

```
t(2) ←
t(3) ← t(5), not t(1)
t(4) ← t(1), not t(2)
t(5) ← t(3), not t(4)
```

In Round 1(a), we infer heads of rules with an empty body. Thus we infer $t(2)$.

Also for convenience, we can eliminate rules whose head has already been inferred, and we can eliminate rules with a false subgoal. For Round 1(b) we are thus left with

```
t(3) ← t(5), not t(1)
t(5) ← t(3), not t(4)
```

In Round 1(b), we find the maximal unfounded set. In this case, $t(1)$ and $t(4)$ have no rules at all and hence belong in the maximal unfounded set. Also, $t(3)$ and $t(5)$ are mutually dependent, so they belong in the unfounded set.

In general, we can find the maximal unfounded set by looking for *poison* positive facts that cannot be in an unfounded set. One way a fact can be poison is if it is the head of a rule with only negative subgoals and at least one such subgoal. Recursively, a fact is poison if it is the head of a rule all of whose positive subgoals are poison. Note that only positive subgoals can be poison.

Thus, another way to find the maximal unfounded set is to find the poison facts. All other positive ground atoms, such that neither they nor their negation is in the model, are in the maximal unfounded set. In our example, there are *no* "poison" facts, i.e., facts that depend on a body with only negative subgoals. Hence, all remaining facts belong in the maximal unfounded set: $\{t(1), t(3), t(4), t(5)\}$.

Rounds 2(a) and 2(b) have nothing more to add, so we are done. Thus, the well-founded model is

$$\{\neg t(1), t(2), \neg t(3), \neg t(4), \neg t(5)\}$$

In this case, there are no unknown facts; we have a two-valued well-founded model. In this model, the output of each of the three gates is 0. □

Example 5: Our next example, the circuit is more complicated and we get some unknown values. We also see how to deal with the possibility that there may be several instantiated rules for one ground atom. The gates are different; now $g(X, Y, Z)$ means that the output Z is 1 if either input X is 1 or input Y is 0. The rules describing the operation of the circuit are:

```
t(Z) ← t0(Z)
t(Z) ← g(X,Y,Z), t(X)
t(Z) ← g(X,Y,Z), not t(Y)
```

The data is:

$$t0(1),\ g(1,2,3),\ g(2,5,4),\ g(2,4,5),\ g(5,3,6)$$

representing the circuit of Fig. 3.

Following are all instantiations without a known false subgoal:

```
t(1) ← t0(1)
t(3) ← g(1,2,3), t(1)
t(3) ← g(1,2,3), not t(2)
t(4) ← g(2,5,4), t(2)
t(4) ← g(2,5,4), not t(5)
t(5) ← g(2,4,5), t(2)
t(5) ← g(2,4,5), not t(4)
t(6) ← g(5,3,6), t(5)
t(6) ← g(5,3,6), not t(3)
```

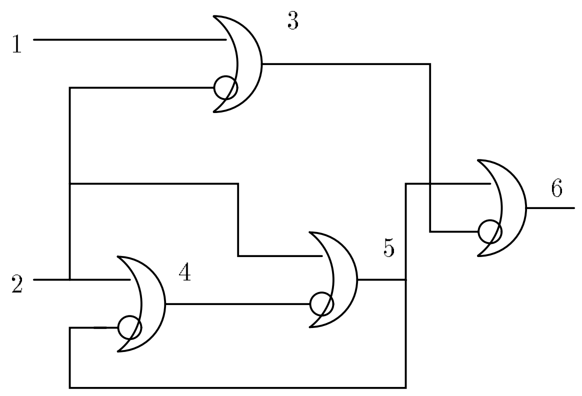

Fig. 3. Circuit for Example 5.

When we eliminate known true subgoals from bodies we get:

```
t(1) ←
t(3) ← t(1)
t(3) ← not t(2)
t(4) ← t(2)
t(4) ← not t(5)
t(5) ← t(2)
t(5) ← not t(4)
t(6) ← t(5)
t(6) ← not t(3)
```

Round 1(a): Infer heads of rules with empty bodies. Thus, $t(1)$ is true. Then, we can eliminate $t(1)$ from rule `t(3) ← t(1)` and infer $t(3)$. No further inferences of this type are possible.

We now eliminate rules whose heads have been inferred, and we eliminate rules with a false subgoal (the last rule, above), leaving the following instantiated rules for $t(2)$, $t(4)$, $t(5)$, and $t(6)$:

```
t(4) ← t(2)
t(4) ← not t(5)
t(5) ← t(2)
t(5) ← not t(4)
t(6) ← t(5)
```

Round 1(b): $t(4)$ and $t(5)$ are "poison," because they have rules with only negated subgoals remaining. Also, $t(6)$ has a rule with a poison subgoal, so it is poison. $\{t(2)\}$ is the maximal unfounded set. Infer $\neg t(2)$.

Round 2(a): Eliminate rules with false subgoals.

```
t(4) ← not t(5)
t(5) ← not t(4)
t(6) ← t(5)
```

No remaining rules have empty bodies, so there are no more inferences to make.

Round 2(b): $t(4)$ and $t(5)$ are poison, and $t(6)$ depends on $t(5)$, so again we make no inferences.

The well-founded model is $\{t(1), \neg t(2), t(3)\}$. Ground atoms $t(4)$, $t(5)$, and $t(6)$ have "unknown" truth-value. Notice that the eventual value for terminals 4, 5, and 6 depends on unknowable conditions when the circuit is "turned on." That is, one of gates 4 and 5 will arrive at a "true" output and the other will have output "false." However, we cannot tell which from the circuit, and the answer depends on physical phenomena not modeled by the circuit. Then, the output of gate 6 will be the complement of the output of gate 5. □

Example 6: In the following example, where several rounds are necessary, we shall use propositional rather than first-order logic. Well-founded semantics applies to propositional logic as well as first-order logic, but we may also imagine that the following are instantiated first-order rules, in which the letters p through u stand for ground atoms.

$p \leftarrow \neg q$
$q \leftarrow r$
$r \leftarrow q$
$s \leftarrow p$
$t \leftarrow \neg s$
$t \leftarrow u$
$u \leftarrow t$

Round 1(a): Infer heads with empty bodies (none).

Round 1(b): p, t are "poison"; u and s depend on these. The remaining set of propositional variables, $\{q, r\}$, is the maximal unfounded set. Infer $\neg q, \neg r$.

Round 2(a): Eliminate rules with false subgoals q and r, and eliminate true subgoal $\neg q$ from body of first rule. The result is:

$p \leftarrow$
$s \leftarrow p$
$t \leftarrow \neg s$
$t \leftarrow u$
$u \leftarrow t$

Now, p can be inferred, and s follows from p.

Round 2(b): Eliminate rules with inferred heads and the rule with false subgoal $\neg s$, leaving:

$t \leftarrow u$
$u \leftarrow t$

Now, $\{t, u\}$ is maximal unfounded set. Infer $\neg t$ and $\neg u$.

The resulting well-founded model is

$$\{p, \neg q, \neg r, s, \neg t, \neg u\}$$

7. Stable Models

At about the same time as well-founded semantics was proposed, Gelfond and Lifschitz [1988] proposed the *stable model semantics*. In its original form, this is a two-valued model; i.e., every ground atom is either true or false. A model M is *stable* for a logic program and some data if:

1. The true EDB ground atoms in M are exactly the given data, and

2. When the rules are instantiated in all possible ways and the resulting bodies are evaluated according to the model, the set of heads of the rules with true bodies is exactly the set of true IDB ground atoms in M.

Example 7: Let us return to the "win" rule of Example 3. If *move* is acyclic, then there is a two-valued well-founded model, in which $win(a)$ is true or false depending on whether or not the player to move can force a win from position a. Call this model M.

We claim M is a stable model when *move* is acyclic. For if $win(a)$ is true in M, then there is a position b such that $move(a, b)$ is true, and $win(b)$ is false in M. Thus, in the instantiated rule

```
win(a) ← move(a,b), not win(b)
```

the body is true according to M, and therefore $win(a)$ is proved.

On the other hand, if $win(a)$ is false in M — that is, $\neg win(a)$ is in M — then for every position (if any) c such that $move(a,c)$ is true, it must be that c is a win for the player whose move it is. That is, $win(c)$ is true in M. It follows that there cannot be an instantiation with head $win(a)$ and a true body. □

A logic program may or may not have a stable model, and it may have more than one. If there is a unique stable model, then this model is taken to be the preferred model; otherwise, the stable-model semantics is mute on the question of a preferred model.

If there is a two-valued well-founded model, i.e., no ground atom is assigned value "unknown," then this model is the unique stable model (Van Gelder, Ross, and Schlipf [1991]). Example 7 is an illustration of this phenomenon. Moreover, both the stable and well-founded approaches are generalizations of local stratification, in the sense that whenever the program is locally stratified, the resulting model is the well-founded model, which is thus two-valued and therefore also the unique stable model.

However, if the well-founded model is not two-valued, then it is still possible that there is a unique stable model.

Example 8: An example, taken from propositional logic, is

$$\begin{array}{ll} p \leftarrow \neg q & q \leftarrow \neg p \\ r \leftarrow p & r \leftarrow \neg r \end{array}$$

Here, the only stable model makes p and r true, and q false. That is, assuming p, r, and $\neg q$, the rule $p \leftarrow \neg q$ lets us infer p, and the rule $r \leftarrow p$ lets us infer r, but q cannot be inferred. Thus, we get the model we started back again, proving it is stable. However, the well-founded model leaves all three of p, q, and r unknown. □

An important difference between the well-founded and stable approaches is tractability. Well-founded models can be computed in time that is polynomial in the size of the database. However, unless the well-founded model is two-valued (so the stable model is the same), there is no polynomial time algorithm for finding a stable model or even telling if one exists.

There have been a number of developments modifying and relating the stable and well-founded semantics. For example, Sacca and Zaniolo [1990] look at intersections of stable models. Baral and Subrahmanian [1992] consider sets of stable models as a meaning for programs. Przymusinski [1990] gives a three-valued extension to the original two-valued definition of stable models, and shows that these coincide with the well-founded model.

8. Modularly Stratified Semantics

Modular stratification was an idea of Ross [1990] that attempts to find a subset of logic programs for which the well-founded semantics could be implemented efficiently. Here, efficiency concerns the "magic-sets" rule transformation that allows queries with bound arguments to be answered without looking at parts of the database that are irrelevant. For instance, in Example 3 we would like to answer the query whether $win(a)$ is true for some particular board a without looking at the move data for positions not reachable from a. We shall not go into magic-sets techniques here; see Ullman [1989] for the motivation.

Here, let us define modularly stratified programs and give examples. In order for a logic program and data to be *modularly stratified*, it must be possible to divide the predicates into *modules* with the following properties.

1. It is possible to order the modules so that predicates in a module depend only on predicates in that module and previous modules. Put another way, there may be recursion among predicates, but it takes place solely within modules.

2. Each module has a locally stratified ("perfect") model when we instantiate its rules and treat all subgoals whose predicates are in previous modules as true or false according to the well founded model for its module.

Thus, we can compute the locally stratified model for each module, in order, using the results from previous modules to determine the truth or falsehood of subgoals that are outside the module. Since each module is thus given a two-valued model, there is no ambiguity as to the truth value of subgoals outside the module.

Example 9: Consider the following rules which augment the rule of Example 3 by introducing an IDB predicate *move*1 that is really the same as the EDB predicate *move*.

```
win(X) ← move1(X,Y), not win(Y)
move1(X,Y) :- move(X,Y)
```

Unlike Example 3, an acyclic *move* relation does not make these rules modularly stratified. The reason is that when we instantiate the rules we get rules like

```
win(1) :- move1(1,2) & not win(2)
win(2) :- move1(2,1) & not win(2)
```

If *move* is acyclic, we cannot have both $move1(1.2)$ and $move1(2,1)$ true, so the apparent cycle will be resolved when we compute *move*1. However, the definition of "locally stratified" fails, because we cannot immediately place $win(1)$ and $win(2)$ in strata.

On the other hand, the rules are modularly stratified. We place *move* and *move*1 in one module and *win* in a higher module. Then, the first module lets us compute *move*1 to be a copy of *move*. When we work on the *win* module, *move*1 behaves like an EDB predicate, and if it is acyclic we can compute the locally stratified model for *win* just as we did in Example 3. □

9. Beyond Well-Founded Models

There has been a perception, shared by the author, that well-founded models (or stable models, depending on your beliefs), are the "right" interpretation to put on logic programs with negation. However, as the subject has received more exploration, anomalies have surfaced for both approaches. Perhaps the clearest expression of the problem is from Torres [1992], the example below.

Example 10: The designers of a complex of buildings wish to put either a cafeteria or a lounge (but not both) in each building. To reduce the number of cafeterias, they will not put two cafeterias in adjacent buildings, but will make sure that there are enough cafeterias that each building either has one, or is adjacent to a building with a cafeteria. The following logic program expresses the situation.

```
lounge(A) ← adjacent(A,B), cafeteria(B)
cafeteria(A) ← not lounge(A)
```

This program is equivalent to the "win" program introduced in Example 3, although here we expect that *adjacent* will be symmetric, and games with a symmetric move relation are uninteresting. The similarity may be clearer if we introduce into the "win" program the intermediate predicate *loss*, as:

```
win(A) ← move(A,B), loss(B)
loss(A) ← not win(A)
```

While the "win" problem seems to need the well-founded semantics, the logically identical (but semantically different) "cafeteria" problem is really solved by the *set* of stable models (pointed out by V. S. Subrahmanian), which correspond in this case to the maximal independent sets of the undirected graph with edges given by *adjacent*. □

In another direction, Geffner [1991] and Torres [1994] generalize the notions of stable and well-founded models by considering *assumption sets*, which are sets of negative literals. Such a set is a candidate for assumption if, when we

1. Instantiate the rules in all possible ways by substituting constants for the variables (needed if the rules are of predicate logic rather than propositional logic),
2. Delete from the bodies of the instantiated rules all negated subgoals that are in the assumption set (instances of the assumption set in the case of predicate logic),
3. Remove those rule instances that still have negated subgoals in the body,

the resulting rules do not logically imply the negation of any (negative) literal in the assumption set. Note that

- Deduction is used only *after* we delete certain negated subgoals from the instantiated rules and delete certain of the rule instances themselves, and therefore applies to a set of rules different from the original rules.

Example 11: Consider the following rules of propositional logic

$p \leftarrow \neg q, \neg s$
$q \leftarrow \neg p, \neg s$
$r \leftarrow \neg p, \neg q$

$\{\neg p, \neg q, \neg s\}$ is not an argument, since when we delete these from the bodies, all the bodies become empty and the rules reduce to their heads, i.e., $\{p, q, r\}$. We can thus prove p (and q also), contradicting a member of the assumption set.

On the other hand, $\{\neg p, \neg s\}$ is an argument, since when we delete these from the bodies, we get

$p \leftarrow \neg q$
q
$r \leftarrow \neg q$

Now, the first and third of these have remaining negated subgoals, so they go away, leaving only the trivial rule

q

When we use these "rules" to infer facts, we prove only q. The implication is that a possible model for our original rules is $\{q, \neg p, \neg r, \neg s\}$ □

The Geffner/Torres approach generalizes the well-founded and stable semantics in that the previous approaches always treat all negative subgoals in a uniform way. In the newer approach, we are free to assume any subset of the negative ground atoms are true and leave the others open.

10. Summary

Figure 4 shows the relationship between the principal classes of logic programs that we have considered. A line directed downward from class A to class B means that

1. Every program/data pair of class B is also a program/data pair of class A, and

2. If a program/data pair is in both classes A and B, then the selected models are the same in both classes.

Bibliography

Apt, K. R., H. A. Blair, and A. Walker [1988]. "Towards a theory of declarative knowledge," in *Foundations of Deductive Databases and Logic Programming* (J. Minker, ed.), Morgan-Kaufmann, San Mateo, pp. 89–148.

Baral, C. and V. S. Subrahmanian [1992]. "Dualities between alternative semantics for logic programming and non-monotonic reasoning," *Proc. First International Workshop on Logic Programming and Non-Monotonic Reasoning*, MIT Press.

Clark, K. L. [1978]. "Negation as failure," in *Logic and Databases* (H. Galliare and J. Minker, eds.), pp. 293–322, Plenum, New York.

Geffner, H. [1991]. "Beyond negation as failure," *Proc. 2nd Intl. Conf. on Principles of Knowledge Representation and Reasoning*, pp. 218–229.

Gelfond, M. and V. Lifschitz [1988]. "The stable model semantics for logic programming," *Proc. Fifth ICLP*, MIT Press, Cambridge MA, pp. 1070–1080.

Kemp, D. B., P. J. Stuckey, and D. Srivastava [1991]. "Magic sets and bottom-up evaluation of well-founded models," *Intl. Conf. on Logic Programming*, pp. 337–350.

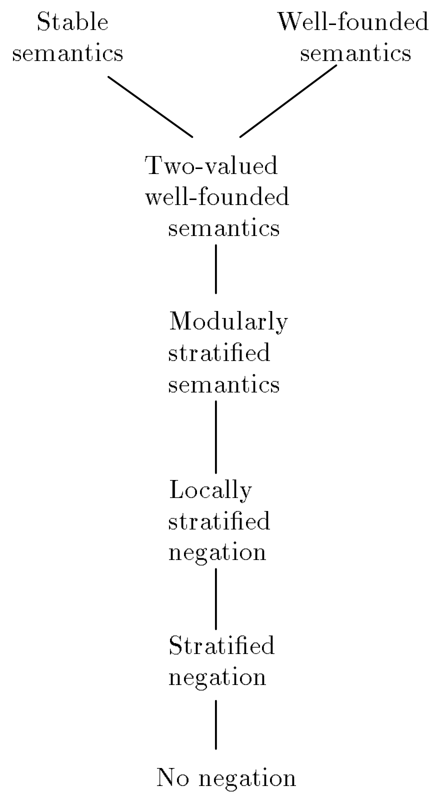
Stable semantics
Well-founded semantics
Two-valued well-founded semantics
Modularly stratified semantics
Locally stratified negation
Stratified negation
No negation

Fig. 4. Relationships between semantics classes.

Kerasit, J. M. and J. M. Pugin [1988]. "Efficient query answering on stratified databases," *Proc. Intl. Conf. on Fifth Generation Computer Systems*, pp. 719–726.

McCarthy, J. [1980]. "Circumscription — a form of non-monotonic reasoning," *Artificial Intelligence* **13**:1–2, pp. 27–39.

Morishita, S. [1993]. "An alternating fixpoint tailored to magic programs," *Proc. Twelfth ACM Symposium on Principles of Database Systems*, pp. 123–134.

Naqvi, S. [1986]. "Negation as failure for first-order queries," *Proc. Fifth ACM Symposium on Principles of Database Systems*, pp. 114–122.

Papadimitriou, C. H. and M. Yannakakis [1992]. "Tie-breaking semantics and structural totality," *Proc. Eleventh ACM Symposium on Principles of Database Systems*, pp. 16–22.

Przymusinska, H. and T. Przymusinski [1989]. "Semantic issues in deductive databases and logic programs," *Sourcebook on the Formal Approaches in Artificial Intelligence* (A. Banerji, ed.),

Przymusinski, T. [1990]. "Well-founded semantics coincides with three-valued stable semantics," *Fundamenta Informaticae* **13**, pp. 445–463.

Przymusinski, T. C. [1989]. "Every logic program has a natural stratification and an iterated least fixed point model," *Proc. Eighth ACM Symposium on Principles of Database Systems*, pp. 11–21.

Ramarkishnan, R. and J. D. Ullman [1993]. "A survey of research in deductive database systems," submitted to *J. Logic Programming*.

Ross, K. A. [1989]. "A procedural semantics for well founded negation in logic programs," *Proc. Eighth ACM Symposium on Principles of Database Systems*, pp. 22–33.

Ross, K. A. [1990]. “Modular stratification and magic sets for datalog programs with negation,” *Proc. Ninth ACM Symposium on Principles of Database Systems*, pp. 161–171.

Sacca, D. and C. Zaniolo [1990]. “Stable models and non-determinism in logic programs with negation,” *Proc. Ninth ACM Symposium on Principles of Database Systems*, pp. 205–217.

Shoham, Y. [1987]. “Nonmonotonic logics: meaning and utility,” *IJCAI*, pp. 388–393.

Torres, A. [1992]. “Is there a right semantics for negation as failure?,” *Third Intl. Symp. on the Deductive Approach to Information Systems and Databases.*

Torres, A. [1994]. “The hypothetical semantics of logic programs,” doctoral thesis, Stanford Univ., Dept. of CS.

Ullman, J. D. [1989]. *Princples of Database and Knowledge-Base Systems* Vol. II, Computer Science Press, New York.

Van Gelder, A. [1986]. “Negation as failure using tight derivations for general logic programs,” *Proc. Symp. on Logic Programming*, IEEE, pp. 127–139.

Van Gelder, A. [1989]. “The alternating fixpoint of logic programs with negation,” *Proc. Eighth ACM Symposium on Principles of Database Systems*, pp. 1–10.

Van Gelder, A., K. A. Ross, and J. S. Schlipf [1991]. “The well-founded semantics for general logic programs,” *J. ACM* **38**:3, pp. 620–650.