NEGATION AS FAILURE. II

JOHN C. SHEPHERDSON

▷ The use of the negation as failure rule in logic programming is often considered to be tantamount to reasoning from Clark's "completed data base" [2]. Continuing the investigations of Clark and Shepherdson [2, 7], we show that this is not fully equivalent to negation as failure either using classical logic or the more appropriate intuitionistic logic. We doubt whether there is any simple and useful logical meaning of negation as failure in the general case, and study in detail some special kinds of data base where the relationship of the completed data base to negation as failure is closer, e.g. where the data base is definite Horn or hierarchic. ◁

1. INTRODUCTION

This paper is a sequel to [7] (which should be consulted for unexplained details), and is concerned with the use of a negation as failure rule in logic programming, in the way it is used in Clark's query evaluation procedure [2] (roughly as in PROLOG). Clark showed there that the use of a negation as failure rule was compatible with the use of a *completed data base* (CDB) obtained by replacing "if" in the clauses of the data base by "iff". We showed in [7] that it was equally compatible with the *closed world assumption* (CWA) of Reiter [6], according to which a positive ground literal not implied by the data base is supposed to be false. We showed that in general the CDB and CWA were different; for example, one could be consistent and the other not, or they could both be separately consistent but incompatible. The fact that negation as failure is compatible with such different assumptions means that in general it cannot coincide with either of them and raises the question of exactly what its logical status is. Doubts as to whether it has a clear logical meaning were raised in [7, Examples 3, 4, 5] by examples of a query $Q_1 \& Q_2$ succeeding but Q_1 not, or a

Address correspondence to J. C. Shepherdson, School of Mathematics, University Walk, Bristol BS8 1TW, England.

52 Vanderbilt Ave., New York, NY 10017 0743-1066/85/$03.30

query failing when some ground instance of it didn't, or a query not succeeding although some ground instance of it did. In Section 2 of this paper we investigate the source of these illogicalities and discuss ways of removing them. The discussion suggests that intuitionistic logic is more appropriate than classical logic, but leaves it doubtful whether there is in general any simple and useful logical characterisation of negation as failure.

This focuses attention on restrictions on the queries or the data base which make the query evaluation procedure more complete and closer to the completed data base. In Section 3 we study Clark's notion of an *allowed query* which cannot flounder (i.e. lead to a query containing unground negative literals to which no evaluation rule applies), and give stronger conditions which are easier to test. In Section 4 we deal with definite Horn data bases, pointing out that if the original data base is really the intended one, then the appropriate rule for negation is not negation as failure but no rule at all. However, if the completed data base was intended, then all is well: an allowed query succeeds under the negation as failure rule iff it is an intuitionistic consequence of CDB. For classical logic the results are almost as good. But for failure they are not; the class of queries Q for which Q fails when $\sim Q$ is a consequence (even an intuitionistic one) of CDB is much more restricted. In Section 5 we consider conditions which are sufficient to ensure completeness in the sense that every allowed query either succeeds or fails. When in addition all evaluation trees are finite we have, for allowed queries, a perfect match with the CDB. In Section 6 we develop Clark's notion of a hierarchic data base which satisfies the conditions of Section 5. These conditions may be relaxed slightly to admit some relations not belonging to the hierarchy. It is possible these apply to most of the situations in which negation as failure is used in practice. Section 7 is an appendix containing some technical lemmas about changing the order in which literals are selected, which are needed because in the presence of negation the results of the query evaluation procedure are no longer independent of the rule used for selecting literals.

2. LOGICAL STATUS OF NEGATION AS FAILURE

Clark's query evaluation procedure of [2] amounts to adding the *negation as failure* rule

$\sim A$ *succeeds if* A *fails* NF

and the corresponding logically impeccable, *failure of negation as success* rule

$\sim A$ *fails if* A *succeeds* FN

for the case where A is a positive ground literal, to SLD resolution (using the most general unifier). (SLD resolution is also called LUSH resolution. Lloyd [5] uses the term SLDNF resolution for SLD resolution plus NF.) The programs, which we shall usually call *data bases* (DB), considered are composed of *clauses* of the form

$$A \leftarrow L_1 \& \ldots \& L_m, \qquad m \geq 0,$$

where A is a positive literal and $L_1, \ldots, L_m$ are positive or negative literals. *Queries* are conjunctions of literals, written in the form

$$\leftarrow L_1 \& \ldots \& L_m, \qquad m \geq 0.$$

The fact that we have not only Clark's result [2]: for all queries Q and substitutions θ,

if Q succeeds with answer θ then $CDB \vdash Q\theta$,

but also

if Q succeeds with answer θ then $\mathrm{CWA} \vdash Q\theta$

and that, as pointed out in [7], in general neither of CDB, CWA implies the other, means that neither of these implications can be strengthened to the equivalence we would like to have:

Q succeeds with answer including θ iff $\mathrm{CDB} \vdash Q\theta$,

where "with answer including θ" means "with answer θ_1 such that $\theta = \theta_1\phi$ for some ϕ". Thus we cannot regard the use of the query evaluation procedure based on negation as failure as being *equivalent* to "completing the data base" (replacing DB by CDB) or to using the closed world assumption (replacing DB by CWA), but only as an incomplete attempt to do both of these, which obviously cannot get closer to either of them than their common part. It is natural to ask whether there is any other way of completing or extending the data base, to DB^+ say, which is equivalent to the query evaluation procedure in the sense that, for all queries Q and substitutions θ,

Q succeeds with answer including θ iff $\mathrm{DB}^+ \vdash Q\theta$.

Since whether a query succeeds depends on the *selection rule* (called "computation rule" by Lloyd [5]) used for choosing literals from queries, and since the above soundness result holds for all selection rules, what we should aim for is

Q succeeds with answer including θ under some selection rule iff $\mathrm{DB}^+ \vdash Q\theta$. (1)

This leaves the possibility that a particular selection rule $\mathscr{R}$, e.g. the PROLOG rule "take the first literal", could be incomplete. There are however, as we show in Section 7, maximal selection rules $\mathscr{R}_m$ such that if a query succeeds with answer θ under any rule, then it succeeds with answer equivalent to θ under $\mathscr{R}_m$. Such rules would then be complete, but it is not clear whether there are any recursive maximal rules. But we are still aiming too high, for it is easy to check that the rules of SLD resolution are intuitionistically valid and that the soundness results of [2] hold intuitionistically, i.e.

if Q succeeds with answer θ then $\mathrm{CDB} \vdash_I Q$

where $\vdash_I$ denotes the intuitionistic consequence relation. It would therefore be unreasonable to expect *classical* completeness. That this is indeed unobtainable is shown by the following example similar to one given by Clark. Take DB to be $P(a) \leftarrow \sim P(a)$. Then $P(a)$ is, according to classical logic, a consequence of DB, yet the query $\leftarrow P(a)$ does not succeed, but has an infinite evaluation tree. So for no extension DB^+ of DB could we have (1) above. The most we can expect is intuitionistic completeness and equivalence, i.e.

Q succeeds with answer including θ under some selection rule iff $\mathrm{DB}^+ \vdash_I Q$. (2)

However the restriction of the negation rules NF and FN to ground literals means that one cannot deal adequately with queries such as $\leftarrow \sim P(x)$ because no rule is

applicable to them. Following Clark [2], we make the

Definition. A query all of whose literals are unground negative literals is called a *flounder*, and an evaluation path which ends in a flounder is said to *flounder*.

Thus if the DB is $P(b) \leftarrow$, $R(a) \leftarrow$, then the query $\leftarrow \sim P(x)$ flounders. However, $\leftarrow \sim P(a)$ succeeds, so if we had (2), then $\sim P(a)$ should be a consequence of DB^+, so $\leftarrow \sim P(x)$ should succeed with answer $x = a$. There is no obvious way of detecting and excluding such queries, for if we add to the above DB the clause $Q(x) \leftarrow \sim P(x)$, the totally positive query $\leftarrow Q(x)$ flounders although $\leftarrow Q(a)$ succeeds. But one way of proceeding is to try and give simpler conditions sufficient to ensure that a query does not flounder. We explore this in the next section, starting with Clark's notion of *allowed query*. So one possible sense in which we could find a meaning for negation as failure is to try to find DB^+ such that

(2) *is true for all allowed queries* (3)

Another way of dealing with flounders is to accept them as a fairly harmless kind of incompleteness. For a flounder is immediately evident; so the evaluation procedure is then telling you clearly and finitely that it cannot deal with the query. So it might be quite acceptable to have, instead of (2), *intuitionistic soundness*: for all Q, θ

if Q succeeds with answer θ under some selection rule then $\mathrm{DB}^+ \vdash_I Q\theta$ (4a)

and *weak intuitionistic completeness*: for all Q, θ

if $\mathrm{DB}^+ \vdash_I Q\theta$ *then Q succeeds or flounders with answer including θ under some selection rule*, (4b)

i.e. ends in a flounder at a point where the answer substitution includes θ. Since allowed queries cannot flounder, (4a, b) imply (3).

A third way of trying to obtain completeness, which was suggested by Clark [2, p. 314], is to extend the query evaluation procedure to deal with nonground negative literals while preserving the intended meaning of queries and of reasoning from some extension of DB. The rule NF is valid, intuitionistically as well as classically, also for nonground negative literals $\sim A$ in the form

$\sim A$ succeeds with answer θ if $A\theta$ fails NFE

because

$$\sim (\exists x) A(x\theta) \vdash_I (\forall x) \sim A(x\theta).$$

The other rule FN, "failure of negation as success", is valid in the form

$\sim A$ fails if A succeeds with answer the identity, FNE

because

$$(\forall x) A(x) \vdash_I \sim (\exists x) \sim A(x).$$

The *extended query evaluation procedure* using rules NFE, FNE instead of NF, FN differs from the old one in allowing the selection of a nonground negative literal $\sim A(t_1, \ldots, t_k)$. When this is done we recursively enter the extended query evaluation process with $\leftarrow A(t_1, \ldots, t_k)$ as the query. If $\leftarrow A(t_1, \ldots, t_k)$ succeeds with answer the identity, then the original query is failed by FNE. If not, then the queries

$\leftarrow A(t_1,\ldots,t_k)\theta$ for all substitutions θ are tried. For each of these which fails, we enter an alternative evaluation path in which, using NFE, $\sim A(t_1,\ldots,t_k)$ is selected from the original query and the rest of the query (Q_1) is replaced by $Q_1\theta$. If we left it like this and failed $\sim A(t_1,\ldots,t_k)$ when there were no such paths, i.e. if none of $\leftarrow A(t_1,\ldots,t_k)\theta$ failed, we should be more or less committing ourselves to Herbrand models (which would be a restriction even for a definite Horn data base; see Section 4) and denying the possibility that if $x_1,\ldots,x_n$ are the variables in $A(t_1,\ldots,t_k)$ there might be new constants $a_1,\ldots,a_n$, not obtainable by substitution, for which $A(t_1,\ldots,t_k)$ failed. So we add another "open" path to cover this possibility. This can never be used positively: it is just there to stop us failing $\sim A(t_1,\ldots,t_k)$ when there are no other paths. This extended procedure is not implementable, because in general there may be infinitely many substitutions, giving an evaluation tree of infinite width. But since it is more complete than the original procedure, it enables us to consider the possibility of a closer logical fit. For we may now ask for intuitionistic soundness and completeness of this extended query evaluation procedure, i.e. for all Q, θ,

$$Q \textit{ succeeds}_E \textit{ with answer including } \theta \textit{ under some selection rule iff } DB^+ \vdash_I Q\theta \tag{5}$$

where $\textit{succeeds}_E$ denotes success under the extended query evaluation procedure. This gives more precise information than (4a, b) and is easily seen to imply them. To prove that (5) → (4a) you prove by an easy induction on the total height of a success tree (including all auxiliary trees used for evaluation of ground negative literals—precise definitions are given after Theorem 4.2) or failure tree that if a query Q succeeds (fails) it succeeds_E (fails_E) according to some selection rule. To prove (5) → (4b) we have to show that succeeds_E implies succeeds or flounders under some selection rule. Use the switching Lemma 7.2 to defer as long as possible the choice of any nonground negative literals; if the query has not succeeded by then, it must flounder.

We shall see in Section 4 that in the case where DB is Horn and definite there exists a DB^+ satisfying (6), namely CDB, which, by Theorem 2 of [7] (or earlier by Apt and van Emden [1]) in this case is equal to CDB ∩ CWA. But even if we ask for the apparently weaker (3) instead of (5), Example 1 of [7] shows that DB^+ cannot always be taken to be CDB or CWA or CWA_I or CDB ∩ CWA or CDB ∩ CWA_I (where CWA_I differs from CWA in having $\sim A$ as an axiom for ground literal A when $DB \nvdash_I A$ instead of $DB \nvdash A$).

Whether there is always a DB^+ satisfying (3) or, better still, (4a, b) or, best of all, (5), and preferably an extension of DB, is not clear. If there is, it can be taken to be the set of all queries Q which succeed with answer 1 [or succeed_E in the case of (5)] under some selection rule. But that would be a rather unhelpful, self-referential explanation of the logical meaning of negation as failure. A slightly more illuminating candidate for DB^+ is the union of DB and the set of all negative literals which succeed with answer 1. If one really wanted to use logic programming in a purely declarative sense—i.e. being able to write a program or data base thinking only of the meaning of the logical terms involved, and not at all of how the evaluation would be carried out—then one would want a simple, clearly understandable definition of DB^+ like CDB or CWA.

One thing is clear:

The results of the query evaluation procedure using negation as failure depend not only on the logical content of the data base clauses but also on the way they are written.

Similarly, if DB^+ satisfying (3) or (4) or (5) exists, it depends not only on the logical content of DB but on the way it is written. As far as classical logic is concerned, this is shown by the difference in the query evaluation procedure caused by writing the data base $P(a) \leftarrow \sim Q(a)$ in the form $Q(a) \leftarrow \sim P(a)$ [$\leftarrow P(a)$ succeeds in the first but fails in the second]. These are not intuitionistically equivalent, but one can show that the query evaluation procedure and DB^+ (if it exists) do not depend only on the intuitionistic logical content of DB, by adding to DB for each relation $R(x_1, \ldots, x_k)$ a clause

$$R(x_1, \ldots, x_k) \leftarrow R(x_1, \ldots, x_k).$$

This is a tautology intuitionistically (and, surely, according to any sort of logic which might conceivably be of use), but it effectively neuters the negation as failure rule, since every positive literal has an infinite evaluation tree and cannot fail. For example, with the DB $P(a) \leftarrow, Q(b) \leftarrow$, the query $\leftarrow \sim P(b)$ succeeds, but when $P(x) \leftarrow P(x)$ is added, it loops.

This fact, that the effect of negation as failure depends on the way the data base is written, lends strength to Gabbay's remark (personal communication) that instead of trying to find a logical meaning for negation as failure in its present form, one should investigate ways of loop cutting which would give a more complete query evaluation procedure and, hopefully, one with a clearer logical meaning.

We have talked so far only about the success of queries. Clark's soundness results in [2] also apply to failure in the form

if Q fails then $\mathrm{CDB} \vdash_I \sim Q$,

so it is natural to ask whether the above completeness criteria could be extended to cover failure, e.g. whether there exists DB^+ such that, for all allowed queries Q

Q fails under some selection rule iff $DB^+ \vdash_I \sim Q$.

Unfortunately this is not possible even for the case of Q a ground query, for if we take DB to be $P(a) \leftarrow P(a)$ and Q to be $\leftarrow P(a) \& \sim P(a)$, then $\vdash_I \sim Q$ but Q doesn't fail. Perhaps the most useful approach here too is to devise ways of making negation as failure more complete by cutting loops or by failing queries which contain more or less evident contradictions.

3. ALLOWED QUERIES

In ordinary SLD resolution with a definite Horn data base, incompleteness arises from infinite evaluation paths. A finite path can only end in success or failure. But in the query evaluation procedure we are discussing, which allows negative literals and uses the negation as failure rule, there are two other kinds of inconclusive result. There is what, following Clark, we have called a *flounder*, where the path terminates in a query containing only nonground negative literals, to which no evaluation rule is applicable. There is also what we shall call a *dead end*, where the chosen literal is a

negative ground literal L which neither succeeds nor fails (so it must have evaluation paths which are infinite or flounder or themselves come to a dead end). In this case the mainstream evaluation is held up because no answer is received about the subsidiary evaluation of L.

We now consider conditions on the data base and the query which are sufficient to prevent these inconclusive results.

For the prevention of dead ends all we can suggest is simply to say that they do not occur, that we have *completeness for ground literals*:

Every ground literal succeeds or fails.

This is a very strong assumption; as shown in [7] it implies that the CDB is consistent, and that if the CWA is consistent, then so is CDB $\cup$ CWA, and if you restrict to Herbrand domains, CDB and CWA are equivalent. It means you have complete information about all relations on elements of the Herbrand universe. However, it is something one would expect to be satisfied if DB really is a data base in the usual sense.

For flounders we can give a necessary and sufficient condition. Let the *positive part* of a query be the query obtained by deleting all its negative literals. Just as we talk about a query succeeding with answer θ, we shall say a query *flounders with answer* θ if there is an evaluation branch which ends in a flounder where θ is the composition of the sequence of unifying substitutions used. Then:

Theorem 3.1. A query Q flounders iff there exists a substitution θ such that the positive part of Q succeeds or flounders with answer θ, every negative literal of Q which is grounded by θ succeeds, and, in the case of success, some negative literal of Q is not grounded by θ.

PROOF. The "if" part is proved by induction on the length of the flounder or success branch of the positive part of Q, using the switching Lemma 7.1 to rearrange this so that if the evaluation of Q starts by choosing a positive literal L_i, then the evaluation path of the positive part of Q also starts off choosing L_i. Similarly for the "only if", using induction on the length of the flounder branch of Q. □

We have the immediate:

Corollary 3.2. If the positive part of a query Q doesn't flounder and each of its success paths grounds all negative literals of Q, then Q doesn't flounder.

A condition expressed in terms of the separate positive literals of a query would be easier to check. Such a condition was given by Clark [2, p. 317]:

Firstly, the constraint that every variable in a negated literal should have its range specified by an unnegated literal that will generate a candidate set of ground substitutions is perfectly acceptable. Let us call this an *allowed query*. For an allowed query no evaluation can flounder because it encounters a query with only unground positive literals.

On reflection it seems to me now that what he intended was the following:

Definition. A query is *allowed* if none of its positive literals flounder and every variable in a negative literal also occurs in a positive literal, all of whose success branches ground it.

A superficial reading of Clark led me in [7] to state that I was following him when I used the term to denote what I shall now call *weakly allowed*:

Definition. A query is *weakly allowed* if every variable in a negative literal also occurs in a positive literal.

This condition is not sufficient to prevent a query floundering; e.g., in the data base $P(x) \leftarrow \sim Q(x)$ the weakly allowed query $\leftarrow P(x)$ flounders, and in the data base $P(x) \leftarrow, Q(a) \leftarrow$ the weakly allowed query $\leftarrow P(x) \& \sim Q(x)$ flounders. But it does have the advantage of being instantly decidable and independent of the selection rule, whereas whether a query is allowed or flounders can depend on the selection rule and, I imagine, may be undecidable for some data bases. And although not a necessary condition for not floundering [if $P(a)$ fails then $P(a) \& \sim Q(x)$ doesn't flounder], it is obviously a necessary condition for success, so there is little point in considering queries which don't satisfy it. We now show that the new definition of allowed does satisfy Clark's statement:

Theorem 3.3. An allowed query cannot flounder.

PROOF. Suppose Q is allowed and has a flounder branch. Then some nonground literal L is present at the end. Suppose L arises from one of the negative literals of Q. Then each variable x of L is in some positive literal L_i of Q, which does not flounder, and all of whose success branches ground x. Rearranging according to Lemma 7.4, we get an equivalent path which has x finally grounded. Hence the original path has all variables of L grounded, contrary to hypothesis.

So L must have been introduced by a unification step applied to some literal derived from some positive literal L_i of Q. Rearranging by Lemma 7.4 so as to start with an evaluation of $\leftarrow L_i$ again gives a contradiction. □

In view of the difficulty in deciding whether a query is allowed, it seems worthwhile to look for conditions which are sufficient to ensure that a *weakly* allowed query cannot flounder. Obviously we have

Theorem 3.4. If no positive literal flounders and every success branch of a positive literal grounds all its variables, then weakly allowed is equivalent to allowed.

But it would be more useful to have some simple conditions on the data base which ensured this hypothesis. The two examples given above of weakly allowed queries which flounder suggest the conditions in the following

Theorem 3.5. If the right hand side of each data base clause is a weakly allowed query, and every variable on the left hand side also appears on the right hand side, then weakly allowed is equivalent to allowed.

PROOF. By Theorem 3.4 it's enough to show that for a positive literal no evaluation path can end in a flounder or a success branch which doesn't ground all its variables. This is proved by taking a shortest evaluation path of this kind and obtaining a contradiction by reasoning as in the proof of Theorem 3.3. □

These conditions are immediately testable but rather restrictive, since they prohibit data base clauses of the form $P(x) \leftarrow$. Clearly there is no harm in such a clause unless $P(x)$ is being used to ground a variable x in a negative literal:

Theorem 3.6. Let a subset of the relations called grounding relations be singled out. Let the corresponding positive literals also be called grounding. Let a query be called grounded if every variable in a negative literal also occurs in a grounding positive literal. Suppose that the right hand side of each data base clause is a grounded query and that if it is a clause about a grounding relation, then every variable occurring on the left hand side also appears in a grounding positive literal on the right hand side. Then

a grounded query is allowed.

PROOF. Similar to that of Theorem 3.5, showing that no positive literal flounders and that every success branch of a grounding positive literal grounds all its variables. □

4. DEFINITE HORN DATA BASES

Logic programming is particularly concerned with data bases consisting of *definite Horn clauses* of the form

$$B \leftarrow A_1 \& \ldots \& A_n$$

where $B, A_1, \ldots, A_n$ are positive literals. Indeed, the term "logic program" is often restricted to programs of this kind. This is the type occurring in pure PROLOG, and it is important because the efficient SLD resolution (using the most general unifier) is complete in this case for all selection rules [1]. So negation as failure might be expected to have a clearer logical meaning for a data base of this kind than it does in general. Results of Apt and van Emden [1] and Jaffar, Lassez, and Lloyd [3] confirm that in this case it is a reasonably complete way of drawing consequences from the completed data base. A convenient source of results is Lloyd [5, Theorems 14.5, 16.1, 8.6, 9.2]:

Theorem 4.1. If DB *is a definite Horn data base, then: If Q is a positive query, then* CDB $\vdash Q$ *iff* DB $\vdash Q$. *If Q is a positive query and* CDB $\vdash Q\theta$, *then Q succeeds with answer including θ under all selection rules. If Q is a positive query and* CDB $\vdash \sim Q$, *then Q fails under every fair selection rule.*

The first result says that the positive information which can be deduced from CDB is exactly the same as the positive information which can be deduced from DB.

A *fair selection rule* is one in which, on each infinite branch of an evaluation tree, every literal is eventually chosen. The "leftmost literal" rule of PROLOG is not a fair rule: you can make it fair by cycling the literals round after each step. The last result does not hold if only Herbrand models are considered; on p. 84 of [5] Lloyd gives an example of a positive query Q such that $\sim Q$ is true in all Herbrand models of CDB but Q does not fail under any rule. These results imply that the strongest soundness and completeness property [5] of Section 2 holds, and the same for classical logic, which gives the same results as intuitionistic logic here.

Theorem 4.2. If DB *is a definite Horn data base, then for all queries* Q *and substitutions* θ, Q *succeeds*$_E$ *with answer including* θ *under any fair selection rule iff* $\text{CDB} \vdash Q\theta$ *iff* $\text{CDB} \vdash_I Q\theta$.

PROOF. For the soundness half we show by an easy induction on the height of the success$_E$ or failure$_E$ tree that if Q succeeds$_E$ with answer θ then $\text{CDB} \vdash_I Q$, and if Q fails then $\text{CDB} \vdash \sim Q$. □

Since this sort of induction is used a lot, we give a precise definition of the notion of "success or failure tree" used here. The idea is that, instead of just counting the steps of the main line evaluation (as we did in the proof of Theorem 3.1 when we talked about the length of a success or flounder branch), we also hang on a side tree for every evaluation of a negative literal.

Definition. The notion of *success tree* (*failure tree*) of a query Q is defined recursively as follows:

Basis. If Q is SUCCESS (FAIL), then the tree consisting of the single node Q is a success (failure) tree for Q.

Inductive step. If L_i is the chosen literal of Q and L_i is a positive literal which doesn't match any DB clause, then the tree with a single FAIL node hanging from the root Q is a *failure tree* for Q. If L_i is a positive literal which matches one or more DB clauses and $Q_1, \ldots, Q_m$ are the resulting derived queries then a *success tree* for Q is a tree consisting of a success tree for some Q_j hanging from the root Q; a *failure tree* for Q is a tree consisting of failure trees for each of $Q_1, \ldots, Q_m$ hanging from the root Q. If L_i is a negative ground literal $\sim A$ a *success tree* for Q is a tree consisting of a failure tree of A and a success tree of Q_1 hanging from the root Q, where Q_1 is the query obtained from Q by deleting $\sim A$. A failure tree for Q is a tree consisting of a failure tree of A and a failure tree of Q_1 hanging from the root Q, or a success tree of A hanging from the root Q.

A *success tree with answer* θ is defined similarly in the obvious way. So are a *flounder tree* and *flounder tree with answer* θ.

For the extended query evaluation procedure the notions of *success*$_E$ *tree*, *success*$_E$ *tree with answer* θ, *failure*$_E$ *tree* are defined similarly, attaching a failure$_E$ tree for $A\theta$ when rule NFE is used to make $\sim A$ succeed with answer θ and a success$_E$ tree with answer 1 when rule FNE is used to fail $\sim A$.

For the completeness half we must show that if $\text{CDB} \vdash Q\theta$, then Q succeeds$_E$ with answer including θ under every fair selection rule. Suppose

Q is $\leftarrow A_1 \& \ldots \& A_r \& \sim B_1 \& \ldots \& \sim B_s$,

where the A's and B's are positive literals. Using the switching Lemma 7.2, we may defer choice of the negative literals until the positive ones have been dealt with. Since $\text{CDB} \vdash Q\theta$, we have, for all i, j, $\text{CDB} \vdash A_i\theta$ and $\text{CDB} \vdash \sim B_j\theta$. So by Theorem 4.1 each A_i succeeds with answer including θ and each $B_j\theta$ fails. So all the positive literals of Q succeed with answer including θ, and when we come to the negative literals these all succeed using rule NFE. □

As in Section 2, it follows that if $\text{CDB} \vdash Q\theta$, then under the original evaluation procedure Q succeeds or flounders with answer including θ, and that if Q is allowed, it must succeed.

However for classical logic the completeness results are not quite perfect even for allowed queries and definite Horn data bases. For we have been talking so far about $\mathrm{CDB} \vdash Q\theta$, i.e. specific substitution instances of Q being consequences of CDB. What we are really supposed to be asking with a query Q is whether $\mathrm{CDB} \vdash \exists(Q)$, where $\exists(Q)$ denotes the existential closure of Q, so we would like to have, at least for allowed queries Q,

if $\mathrm{CDB} \vdash \exists(Q)$ *then* Q *succeeds under some selection rule*.

The following is a counterexample: take DB to be $A(a) \leftarrow, A(f(a)) \leftarrow A(f(a))$ [for a slightly less pathological example replace the latter by $A(f(a)) \leftarrow A(g(a)), A(g(a)) \leftarrow A(f(a))$], and Q to be $\leftarrow A(x) \& \sim A(f(x))$. Here $\sim A(f(f(a)))$ is a consequence of CDB; hence so is $\exists(Q)$, i.e. $(\exists x)(A(x) \& \sim A(f(x)))$, because if $\sim A(f(a))$ holds, then $x = a$ satisfies, and if $A(f(a))$ holds, then $x = f(a)$ satisfies. But there is no single term t such that $\mathrm{CDB} \vdash A(t) \& \sim A(f(t))$, so Q cannot succeed (or $\mathrm{succeed}_E$), nor does it flounder; it has an infinite evaluation tree. So even for a definite Horn data base intuitionistic logic is the appropriate one; the only reason there are also partial completeness results for classical logic is that after the existential quantifier has been removed by choice of a specific term, classical and intuitionistic logic give the same results.

Note that even for a definite Horn data base the CDB is not determined solely by the logical content of DB; e.g. adding the tautologies $R(x_1, \ldots, x_k) \leftarrow R(x_1, \ldots, x_k)$ for all relations makes CDB equivalent to DB.

For *failure* the completeness result, even in the form

if $\mathrm{CDB} \vdash \sim Q$ *then* Q *fails or flounders*,

does not go very far and can fail for a *ground* query, as shown by the example of Section 2, the query $P(a) \& \sim P(a)$ with data base $P(a) \leftarrow P(a)$. But it is true for *positive* queries as stated in Theorem 4.1, in the form

if $\mathrm{CDB} \vdash \sim Q$ *then* Q *fails under every fair selection*.

This is also true for *purely negative ground* queries $\leftarrow \sim B_1 \& \ldots \& \sim B_s$. For if, for some i, $\mathrm{DB} \vdash B_i$, then B_i succeeds, so $\sim B_i$ fails and so does Q. And if, for all i, $\mathrm{DB} \nvdash B_i$, then, as shown in [7, proof of Theorem 2], the Herbrand model with a positive ground literal L true iff $\mathrm{DB} \vdash L$ is a model for CDB, so in this model Q is true and $\sim Q$ is false, so $\mathrm{CDB} \nvdash \sim Q$. It can be shown to hold for all purely negative queries if *fails* is replaced by fails_E.

Note that in the case of a definite Horn data base the use of negation as failure is very limited. The only negations are those in the original query $\leftarrow A_1 \& \ldots \& A_r \& \sim B_1 \& \ldots \& \sim B_s$, and the negation as failure rule is only used at the end, to declare a grounded $\sim B_j\theta$ a success or failure according as $B_j\theta$ fails or succeeds using the ordinary SLD resolution for positive clauses and queries.

Note also that the completeness results we have are for CDB, not DB. If the intended data base is really DB, then in order to deal with queries involving negation and to achieve full classical soundness and completeness for all queries, i.e.

Q *succeeds under every selection rule iff* $\mathrm{DB} \vdash \exists(Q)$,

it is not the negation as failure rule which should be used, but *no additional rule at all*. For a query Q containing a negative literal cannot then succeed; nor can $\mathrm{DB} \vdash \exists(Q)$, for DB has a model in which all relations are true for all arguments.

And for positive queries this is just the usual soundness and completeness result for SLD resolution.

5. WHEN QUERY EVALUATION IS COMPLETE

In view of the difficulties discussed in Section 2 in the way of finding a clear logical meaning for the query evaluation procedure incorporating negation as failure when it is incomplete, it is worth looking for cases where it is complete, i.e. where allowed queries either succeed or fail. The results here are extensions of those of Clark [2] and of [7].

Theorem 5.1. If the query Q succeeds or fails and if DB is consistent, then

$$Q \text{ succeeds } \textit{iff } \mathrm{CDB} \vdash \exists(Q),$$
$$Q \text{ fails } \quad \textit{iff } \mathrm{CDB} \vdash \sim Q.$$

Similarly with CWA *in place of* CDB *and* (*at least for* CDB) *with* $\vdash_I$ *instead of* $\vdash$.

PROOF. It is an immediate consequence of Clark's soundness results that if Q succeeds with answer θ then $\mathrm{CDB} \vdash Q\theta$, and if Q fails then $\mathrm{CDB} \vdash \sim Q$. Similarly for CWA using the results of [7], and for $\vdash_I$ by the intuitionistic soundness of CDB observed in Section 2. (I haven't checked CWA using intuitionistic logic; you might need to use CWA_I as defined in Section 2 instead.) □

So in this case, if CDB and CWA are both consistent, they agree as far as Q is concerned. As shown in [7], the consistency of CDB is guaranteed if all ground literals succeed or fail.

However, we may not have completeness with respect to CDB in the sense of obtaining all answers, i.e.

$$Q \textit{ succeeds with answer including } \theta \textit{ if } \mathrm{CDB} \vdash Q\theta,$$

for if you take the data base to be $P(a) \leftarrow$, $P(x) \leftarrow \sim P(x)$ and take Q to be $\leftarrow P(x)$, then $\mathrm{CDB} \vdash Q$, but Q does not succeed with answer the identity substitution 1 (although it does succeed with answer $x = a$, which is equivalent from the point of view of the CDB, which implies $x = a$). In this case one branch of the evaluation tree flounders.

Definition. A query has the *finite tree property* if its evaluation tree is finite and each branch ends in success or failure. If in addition all successful branches result in full instantiation, it is said to have the *finite grounded tree property.*

Definition. A query Q is said to be *answer complete* if

$$Q \textit{ succeeds} \qquad \textit{iff } \mathrm{CDB} \vdash (\exists)Q,$$
$$Q \textit{ fails} \qquad \textit{iff } \mathrm{CDB} \vdash \sim Q,$$
$$Q \textit{ succeeds with answer including } \theta \textit{ iff } \mathrm{CDB} \vdash Q\theta.$$

Theorem 5.2. A query with the finite tree property is answer complete.

PROOF. Clark [2, pp. 317, 318]. □

From the practical point of view it is just as desirable that a query should have a finite evaluation tree as that it should succeed or fail, and we now look into conditions which are sufficient to ensure this. Clark [2, p. 318] stated that if all queries of the form $\leftarrow R(x_1, \ldots, x_k)$ have the finite grounded tree property for all selection rules, then so do all allowed queries. It was pointed out in [7] that this was not quite correct, but needed the property for ground literals as well. However, Clark's result can be rescued by replacing "all selection rules" with "some selection rule". We need a lemma:

Lemma 5.3. If a query Q has the finite tree property under some selection rule $\mathscr{R}$, then so, under some selection rule $\mathscr{R}_\theta$, does $Q\theta$. Similarly for the finite grounded tree property.

PROOF. By induction on the height of the evaluation tree for Q under $\mathscr{R}$, where, as in Section 4, we consider the evaluation tree to include side trees for the evaluation of negative literals. If this height is zero, then Q is either SUCCESS or FAIL, and so is $Q\theta$.

Inductive step. If the first step in the evaluation of Q under $\mathscr{R}$ is to choose the literal L_i, then define $\mathscr{R}_\theta$ to choose the literal $L_i\theta$ of $Q\theta$.

If L_i is a negative ground literal, then $L_i\theta = L_i$ and we define $\mathscr{R}_\theta$ with root L_i to behave the same as $\mathscr{R}$. If L_i fails, we then get a finite failure tree for $Q\theta$. If L_i succeeds, we apply the induction hypothesis to the new queries Q' and $Q'\theta$ resulting from the deletion of L_i from $Q, Q\theta$.

If L_i is a positive literal and L_i doesn't match any DB clause, then $Q\theta$ fails at once. If $L_i\theta$ unifies with some R from a DB clause, with most general unifier θ_1', then by Lemma 1 of [7], L_i unifies with R, and if the new queries are $Q', (Q\theta)'$, we have $(Q\theta)' = Q'\theta\theta_1'$. Now each Q' has under $\mathscr{R}$ a finite evaluation tree of lower height than Q; hence, by the induction hypothesis, each $(Q\theta)'$ has a finite evaluation tree under some selection rule. □

Theorem 5.4. If each query of the form $\leftarrow R(x_1, \ldots, x_k)$ has the finite tree property under some selection rule, then there is a selection rule $\mathscr{R}$ under which all positive literals have this property. Similarly for the finite grounded tree property.

PROOF. By the last lemma each positive literal has the finite tree property under some selection rule. Take for $\mathscr{R}$ a rule which, for a query with the finite tree property under some rule, follows a rule which gives a shortest evaluation tree. □

This result is of no practical use, because it gives no way of finding the rule $\mathscr{R}$. It is not true that it can be taken to be any fair rule, because the fairness does not prevent an evaluation branch ending in the evaluation of $\sim L$ where L has an infinite evaluation tree. This suggests that we might add to fairness the condition that negative ground literals are only to be chosen when there are no positive literals left. But this is not enough, for if DB is taken to be $P(x) \leftarrow \sim P(x) \& \sim Q(a)$, $Q(x) \leftarrow$, then under every rule $\leftarrow Q(x)$ succeeds and $\leftarrow P(x)$ fails, both with finite trees. But $\leftarrow P(a)$ neither succeeds nor fails under the fair rule which takes negative literals only when there are no positive, and takes the first permitted literal, cycling the literals round after each mainstream evaluation step. To ensure that one always reaches a failing literal [like $\sim Q(a)$ here] it seems to be necessary to alter the form

of the evaluation. In the method which (following Clark, Lloyd, and others) we are using, when a negative literal $\sim L$ is chosen, the evaluation of L is done as a subsidiary evaluation before proceeding with the main evaluation, and a success or failure tree for L is tacked on to the side of the main tree. What is needed here seems to be some sort of intercalation of the evaluation of L with further choices of literal in the main evaluation.

This sample example shows that we cannot improve the result by replacing "some selection rule" with "all selection rules" or with some arbitrarily given fixed selection rule (even if it is assumed to be fair and to prefer positive literals). It is unfortunate that this least useful "some selection rule" form should be the one which is needed, but not surprising, because every answer given by any rule is a consequence of the CDB, so success or failure under some selection rule is the most complete of the three forms.

Theorem 5.5. If every query of the form $\leftarrow R(x_1, \ldots, x_n)$ *has the finite tree property, every ground literal succeeds or fails, and the selection rule is fair, then every query has a finite evaluation tree.*

PROOF. If a query Q has an infinite evaluation tree, then (since we assume there are only finitely many clauses in DB about any relation) it must by König's lemma have an infinite branch. Since all ground literals succeed or fail, there are no dead ends, so this must be a mainstream branch. By Lemma 7.3 there will be an infinite mainstream branch under any other selection rule which succeeds all ground literals which succeed under the given rule. A rule $\mathscr{R}$ given by Theorem 5.4 under which all positive literals have the finite tree property can easily be arranged to satisfy this. If we now use the rule which takes the positive literals of Q in turn and applies $\mathscr{R}$ to them, we get a contradiction, for all evaluation branches are finite. □

NOTE:. A similar argument shows that the condition that all queries of the form $\leftarrow R(x_1, \ldots, x_n)$ have the finite tree property may be replaced by "all positive literals in the query have the finite tree property".

Corollary 5.6. Under the conditions of Theorem 5.5 *every allowed query is answer complete. If "finite tree" is strengthened to "finite grounded tree", then every weakly allowed query is answer complete.*

PROOF. The first part follows from Theorems 3.3 and 5.2. For the second use the second part of Theorem 5.4 and the switching Lemma 7.3 to show that all positive literals have the finite grounded tree property. Theorem 3.4 then shows that "weakly allowed" is equivalent to allowed. □

There is a similar "all selection rules" (including unfair ones) version. This is a bit stronger than Theorem 7 of [7], but follows by the same proof:

Theorem 5.7. If under all selection rules every query of the form $\leftarrow R(x_1, \ldots, x_n)$ *has the finite tree property and every ground literal succeeds or fails, then every query has a finite evaluation tree under all selection rules and every allowed query is answer complete under all selection rules. If "finite tree" is strengthened to "finite grounded tree", then every weakly allowed query is answer complete.*

6. HIERARCHIC DATA BASES

Clark [2] gave conditions on the data base sufficient to imply the finite tree property for all allowed queries under all selection rules. They are based on a hierarchical condition:

Definition. The data base satisfies the *hierarchical condition* if the relations can be assigned to *levels* so that in each clause

$$R(t_1,\ldots,t_k) \leftarrow L_1 \& \ldots \& L_m$$

about an ith level relation R, the relations occurring in $L_1,\ldots,L_m$ are of level less than i.

This prevents any evaluation tree having infinite branches, because a unification step replaces a literal by lower level literals.

Definition. The relations *involved in* a query Q are all those occurring in Q together with all those occurring on the right hand side of data base clauses about these relations and so on.

There are only finitely many relations involved in a query, since we are assuming there are only finitely many clauses about each relation. Let N be the maximum number of literals on the right hand side of any clause about any of the relations involved in Q. Then it is easily seen that the height of any evaluation tree for Q is not more than $h(Q) = \Sigma_j n_j (N+1)^j$, where n_j is the number jth level literals in Q; and the number of different evaluation trees (arising from different relation rules) is not more than $h(Q)^{h(Q)}$. The branches of such trees may not all end in success or failure, but in a flounder or dead end. In principle it is possible to check out all evaluation trees of a query, in particular for a single literal, for all selection rules and hence to decide whether the conditions of any of the completeness theorems of the last section are satisfied for all the relations involved in a query. But is is probably of more practical use to give simple conditions sufficient to ensure these. The following condition is due to Clark [2].

Definition. A data base satisfies the *allowability condition* if the right hand side of each data base clause is an allowed query.

Theorem 6.1. If the data base satisfies the hierarchical condition and the allowability condition under all selection rules, then all allowed queries are answer complete under all selection rules.

PROOF. This follows from Theorem 5.7, since an easy induction on the level (using Theorem 5.7) shows that all positive literals have the finite tree property under all selection rules. Or that all allowed queries have the finite tree property can be proved directly by induction on the quantity $h(Q)$ defined above. □

In [7] I wrongly stated that Theorem 6.1 was not quite correct because by an oversight I had assumed that Clark was using the weak allowability condition. With

that one also needs a covering condition [7, Theorem 8]:

Theorem 6.2. If the data base satisfies the hierarchical condition, if the right hand side of each data base clause is a weakly allowed query, and if every variable on the left hand side appears on the right hand side, then all weakly allowed queries are answer complete under all selection rules.

PROOF. By Theorem 3.5 *weakly allowed* is equivalent to *allowed* here. As in Theorem 3.6, the covering condition may be weakened by singling out a class of "grounding" relations used for covering. □

The hierarchical condition is a very strong one which forbids any recursive or mutually recursive definitions. However, it seems to me a very appropriate condition when you are talking about drawing consequences from the completed data base. For to get this from the original data base you have to convert "if" clauses into "iff". Now the only time a mathematician writes "if" when he really means "iff" is when he is defining some new relation in terms of already defined relations, which is precisely the hierarchic situation. And whenever you have a non-Horn clause, e.g. $P(x) \vee Q(x)$, you have to choose whether to write this as

$$P(x) \leftarrow \sim Q(x)$$

or

$$Q(x) \leftarrow \sim P(x),$$

which give different CDB. The choice here establishes some sort of priority. Of course, if there are recursive or mutually recursive definitions, then this priority may not be an order relation. But in that case the effect of replacing DB by CDB is not easy to predict.

However, there are some cases where recursive definitions are used when the CDB is what is intended, e.g. for the relations "append" or "sorted". And it may be a fairly common situation for the data base to be hierarchical apart from some relations of this kind. To be more precise, suppose the relations and corresponding literals are divided into complementary classes of *hierarchical* and *nonhierarchical* relations.

Definition. A data base satisfies the *modified hierarchical and allowability condition* if the hierarchical relations can be assigned to levels so that in a clause about an ith level relation the only hierarchical relations on the right hand side are of level less than i, the nonhierarchical literals on the right hand side have the finite tree property, and the right hand side is an allowed query.

Theorem 6.4. If the data base satisfies the modified hierarchical and allowability condition, if all nonhierarchical ground literals succeed or fail, and if the selection rule is fair, then every allowed query all of whose nonhierarchical literals have the finite tree property is answer complete.

PROOF. By induction on the level the note after theorem 5.5 shows that all positive literals involving hierarchical relations also have the finite tree property and that all hierarchical ground literals succeed or fail. By the same note it follows that an

allowed query all of whose nonhierarchical literals have the finite tree property has the finite tree property and hence, by Theorem 5.2, is answer complete.

7. SWITCHING LEMMA; MAXIMAL SELECTION RULES

Since the result of query evaluation may depend on the selection rule used, it is useful to note that the switching lemma which Lloyd gave for SLD resolution [5, Lemma 9.1] also applies when negative literals and the negation as failure rule are used.

Definition. Two queries Q_1, Q_2 are *variants* of each other when there exist substitutions θ_1, θ_2 such that $Q_1\theta_1 = Q_2$ and $Q_2\theta_2 = Q_1$.

As Lloyd shows, θ_1, θ_2 may be taken to be *renaming* substitutions, i.e. such that θ_1 maps certain variables of Q_1 onto variables distinct from each other and from the other variables of Q_1 (and similarly for θ_2, Q_2).

Definition. Two finite evaluation paths for a query Q, with answers θ_1, θ_2, are *equivalent* if $Q\theta_1, Q\theta_2$ are variants of each other. Two paths which both fail or both dead-end are also said to be equivalent.

So equivalent paths give the same answers apart from the names of the variables.

Lemma 7.1 (Switching lemma). If L_i, L_j are positive or negative ground literals of a query, and an evaluation path chooses first L_i and then L_j, and L_j does not immediately fail, there is an equivalent evaluation path which differs only in choosing first L_j, then L_i.

PROOF. Strictly speaking we should say "first L_i and then the literal $L_j\theta$ into which L_j is sent by the first step." By saying that L_j does not immediately fail we mean that $L_j\theta$ is not a failing negative ground literal or nonmatching positive literal. Of course the equivalent evaluation path uses a different selection rule. If L_i, L_j are positive literals, this is Lemma 9.1 of [5], for the fact that some other literals may be negative does not affect the proof. If one or both of L_i, L_j is a negative ground literal, the result is obvious, since a negative ground literal is unaffected by unification. □

Lemma 7.2 (Switching lemma). If L_i, L_j are literals of a query and an extended evaluation chooses first L_i and then L_j, and L_j does not immediately fail, there is an equivalent evaluation path which chooses first L_j, then L_i.

PROOF. Lemma 2 of [7], that if Q fails then $Q\theta$ fails under some selection rule, is easily extended to the extended query evaluation procedure. Now the application of the new rule NFSE, that if $A\theta$ fails then $\sim A$ succeeds with answer θ, can be regarded as being obtained by unifying $\sim A$ with the clause $\sim A\theta \leftarrow$ obtained by extending the data base with clauses $\sim L \leftarrow$ for each L which fails. So whether L_i, L_j are positive or negative, the argument of Lemma 9.1 of [5] now applies. □

Theorem 7.3. Suppose every negative ground literal which succeeds under selection rule $\mathscr{R}_1$ also succeeds under selection rule $\mathscr{R}_2$. Then if a query succeeds (flounders) with answer θ under $\mathscr{R}_1$, it succeeds (flounders) with answer equivalent to θ under $\mathscr{R}_2$. If, in addition, $\mathscr{R}_1$ is a fair rule and there is an infinite branch under $\mathscr{R}_1$, then there is an infinite branch under $\mathscr{R}_2$.

PROOF. Suppose a query has a success (flounder) branch of length l under $\mathscr{R}_1$. We show by induction on l that there is an equivalent success (flounder) branch under $\mathscr{R}_2$. If $l=0$ the result is obvious. For the inductive step suppose $\mathscr{R}_2$ starts by choosing the literal L_i. Whether L_i is a positive or ground negative literal it must be removed, i.e. chosen at some stage, by $\mathscr{R}_1$. Now use the switching lemma to make it the first choice and use the induction hypothesis. If L_i is a ground negative literal, it must succeed under $\mathscr{R}_1$ and hence, by hypothesis, also under $\mathscr{R}_2$.

Similarly, if we have an infinite branch under a fair selection rule $\mathscr{R}_1$ and some other rule $\mathscr{R}_2$ starts by choosing L_i, we can use the switching lemma to make this the first choice, and by continuing the process construct an infinite branch under $\mathscr{R}_2$. All ground literals must succeed under $\mathscr{R}_1$ and, by hypothesis, also under $\mathscr{R}_2$. □

Lemma 7.4. For each success or flounder branch of a query Q under a selection rule $\mathscr{R}$, and each positive literal L of Q, there is an equivalent evaluation path which starts by following a success or flounder branch of $\leftarrow L$ under the rule $\mathscr{R}$.

PROOF. Apply the last theorem with $\mathscr{R}$ as one of the rules, and as the other the rule which, on the query Q, starts by following $\mathscr{R}$ on the query $\leftarrow L$. □

Definition. A selection rule $\mathscr{R}_m$ is *maximal* if each query which succeeds (fails) under some other rule succeeds (fails) under $\mathscr{R}_m$.

In [7, Theorem 5] maximal rules were shown to exist, but the question whether they obtained all possible answers to queries was left open. In fact it is an immediate consequence of Theorem 7.3:

Theorem 7.5. If a query succeeds with answer θ under some selection rule, then it succeeds with answer equivalent to θ under each maximal selection rule.

REFERENCES

1. Apt, K. R. and van Emden, M. H., Contributions to the Theory of Logic Programming, *J. Assoc. Comput. Mach.* 29:841–863 (1982).
2. Clark, K. L., Negation as Failure, in: H. Gallaire and J. Minker (eds.), *Logic and Data Bases*, Plenum, New York, 1978.
3. Jaffar, J., Lassez, J.-L., and Lloyd, J. W., Completeness of the Negation as Failure Rule, in: *IJCAI-83*, Karlsruhe, 1983, pp. 500–506.
4. Lassez, J.-L. and Maher, M. J., Closures and Fairness in the Semantics of Programming Logic, *Theoret. Comput. Sci.*, to appear.
5. Lloyd, J. W., *Foundations of Logic Programming*, Springer, New York, 1984.
6. Reiter, R., On Closed World Data Bases, in: H. Gallaire and J. Minker (eds.), *Logic and Data Bases*, Plenum, New York, 1978.
7. Shepherdson, J. C., Negation as Failure: A Comparison of Clark's Completed Data Base and Reiter's Closed World Assumption, *J. Logic Programming* 1:1–48 (1984).