The Well-Founded Semantics for General Logic Programs*

Allen Van Gelder — Univ. of Calif. at Santa Cruz
Kenneth A. Ross — Stanford Univ.
John S. Schlipf † — Univ. of Cincinnati

April 18, 1990

This paper appeared in: *Journal of the Association for Computing Machinery*, volume 38, number 3, pp. 620–650 (1991).

Abstract

A general logic program (abbreviated to "program" hereafter) is a set of rules that have both positive and negative subgoals. It is common to view a deductive database as a general logic program consisting of rules (IDB) sitting above elementary relations (EDB, facts). It is desirable to associate one Herbrand model with a program and think of that model as the "meaning of the program," or its "declarative semantics." Ideally, queries directed to the program would be answered in accordance with this model. Recent research indicates that some programs do not have a "satisfactory" *total* model; for such programs, the question of an appropriate *partial* model arises. We introduce *unfounded sets* and *well-founded partial models*, and define the well-founded semantics of a program to be its well-founded partial model. If the well-founded partial model is in fact a total model, we call it the *well-founded* model. We show that the class of programs possessing a total well-founded model properly includes previously studied classes of "stratified" and "locally stratified" programs. We also compare our method with other proposals in the literature, including Clark's "program completion," Fitting's and Kunen's 3-valued interpretations of it, and the "stable models" of Gelfond and Lifschitz.

Categories and Subject Descriptors: D.3.1 [**Programming Languages**]: Formal Definitions and Theory — semantics; F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic — logic programming, model theory; I.2.3 [**Artificial Intelligence**]: Deduction and Theorem Proving — logic programming, nonmonotonic reasoning and belief revision

General Terms: Languages, Theory

Additional Key Words and Phrases: negation as failure, well-founded models, fixpoints, unfounded sets, stable models, three-valued logic

---

*A preliminary version of this paper was presented at the Seventh ACM Symposium on Principles of Database Systems, 1988.

†Authors' Addresses: Kenneth A. Ross, Computer Science Dept., Stanford University, Stanford, CA 94305; John S. Schlipf, Computer Science Dept., University of Cincinnati, Cincinnati, OH 45221; Allen Van Gelder, Baskin Computer Science Center, University of California, Santa Cruz, CA 95064.

1. Introduction

There has been much recent work on extending Horn rule logic programs to include negative subgoals, giving what are called general logic programs. This research has proceeded in two general directions, which may be summarized as the "program completion" approach and the "canonical model" approach.

1.1. Program Completion Semantics

The original "program completion" approach, due to Clark [6], and discussed in detail by Shepherdson [37, 38], Kunen [17], and Lloyd [20], has been to define a new program, called the *completed program* (sometimes called the *completed database*). The completed program is treated simply as a first order formula (see Section 4). Then the negative literals that are logical consequences of the completed program, and only those, should be considered true. The same applies to positive literals, so the completion treats positive and negative literals symmetrically. A proof method that supports this approach, called SLDNF (SLD resolution plus the negation as failure rule) has been studied extensively. A closely related idea, the *closed world assumption*, was introduced in the context of deductive databases by Reiter [33]. The *generalized closed world assumption* was proposed by Minker to handle disjunctive databases [25] without producing the inconsistency typical of the closed world assumption; it is discussed in Example 3.1.

SLDNF is applied to the original program. Clark showed the procedure to be sound in the sense that if a goal has a finite SLDNF derivation, then it is a logical consequence of the completed program. Jaffar, Lassez and Lloyd showed that SLDNF was complete (in the same sense) for *Horn* programs with non-floundering queries consisting of a conjunction of positive and/or negative literals [15]. SLDNF was further investigated for *general* logic programs by Lloyd [20] (who coined the term SLDNF), Shepherdson [37, 38] (*q.v.* for further bibliography), and others. This approach is "logically" impeccable, but does not address the issue of how the compiler or the interpreter of the general logic program should treat atoms (goals) whose positive and negative literals are neither logical consequences of the completion: the interpreter is not allowed to either succeed *or* fail. Also, for some programs the completed program is inconsistent; for some others, the completed program is consistent but unintuitive. More importantly, on many natural examples it yields a surprisingly weak reasoning ability. We shall illustrate these claims with examples in Section 7.

Fitting [9] and Kunen [17] gave markedly different, more uniform, semantics by interpreting the completed program in a 3-valued constructive logic, elegantly eliminating some difficulties of the Clark program completion approach. The third truth value, $\perp$, connotes unknown truth value and is "less information than" both **true** and **false**, which are incomparable. Fitting showed that the completion of every program has a (unique) minimum 3-valued model, and suggested that this model be taken for the semantics of the program. Kunen describes a variant that is always recursively enumerable, and characterizes the *3-valued logical consequences* of the completed program. From our point of view, however, these semantics are also too weak to capture the "common sense" notion of negation as failure, as discussed later in the motivating examples (Section 7).

A rather different approach to negation is to interpret general rules as disjunctive clauses. In this context, the *generalized closed world assumption* concludes that p is false if there is no minimal positive disjunction $p \vee q_1 \vee \cdots \vee q_k$ that is a (2-valued) logical consequence of the clauses [25]. Here k may be zero, so that p is simply true. Disjunctive databases are quite different from logic programs because clauses have no "direction". Thus $a \leftarrow$**not** b and $b \leftarrow$**not** a are treated alike, as $a \vee b$. Example 3.1 illustrates this distinction.

1.2. Canonical Model Semantics

The "canonical model," or "preferred model," approach has been to declare that a certain model of the original program is presumed to be the "intended" one, i.e., the one that the programmer and program users have in mind. The justification for choosing the preferred model relies on an appeal to "common sense," and what people who write or read the program are likely to think it means. R. W. Topor and E. A. Sonenberg proposed the term "canonical model" to describe a model that is selected (often from many incomparable minimal models) to represent the "meaning" of a logic program or deductive database. The advantage of assigning a canonical model to a program is that one now has a standard for correctness of an interpreter[1] on *all* goals – it must conform to the canonical model, and succeed or fail appropriately. See [41] for a discussion of how the canonical model approach can benefit application development.

Another motivation for concentrating on canonical models is the view, expounded by Reiter [33], that many logic programs are appropriately thought of as having two components, an *intensional* database (IDB) that represents the reasoning component, and the *extensional* database (EDB) that represents a collection of facts. Over the course of time, we may want to "apply" the same IDB to many quite different EDBs. In this context the properties of the IDB merit careful study, and it makes sense to think of the IDB as implicitly defining a transformation from an EDB to a set of derived facts; we would like the set of derived facts to be the canonical model. For finite cases the computational complexity of this transformation can be studied; see Section 8.

One problem with the canonical model approach is that some programs may not *have* a canonical model, or if they do, it is unclear that the model matches the users' expectations. A further difficulty is that the canonical model may be computationally infeasible. One line of research has been to look for a definition of the canonical model that will apply to as broad a class of programs as possible. Two classes of programs that have been studied are called *stratified* and *locally stratified*. The stratified class has been treated in [5, 1, 19, 40], and elsewhere. The locally stratified class, defined and studied by Przymusinski [31], is a superset of the class of stratified programs. He defined *perfect* models, and showed that every locally stratified program has a unique perfect model. These classes are discussed further in Section 6.

For a while there was a feeling that programs that were not at least locally stratified probably did not really make good sense, that they were inherently ambiguous, and thus faulty. Thus failure to have a perfect model was thought to indicate a flaw in the program rather than in the definition of perfect models. Recent experience has cast doubt on this attitude (see [11] for discussion), and spurred the search for further improvements in the definition of the "canonical model."

Gelfond and Lifschitz propose an elegant definition of a *stable model* that is closely related to our work [11]. Drawing on ideas in [10], they define a "stable model" as one that is able to reproduce itself in a certain sense; a program may have zero, one, or many stable models. In their scheme, when a program has a *unique* stable model, that is is considered to be its canonical model. They argue that the unique stable model is the natural one to associate with a logic program, and describe some of its properties. Stable models are discussed further in Section 5.

1.3. Well-Founded Semantics

This paper proposes a new definition of canonical model, which we call the *well-founded* model. We show that for locally stratified programs the perfect model coincides with the well-founded model; in addition,

[1] By "interpreter," we mean any mechanism for executing the program, including a compiler.

certain programs that are not locally stratified have a well-founded model. Examples are given in Section 7.

But even when a program has no well-founded *total* model, it has a well-founded *partial* model; thus we define the *well-founded semantics* of any general logic program to be that literals in the well-founded partial model are true, their complements are false, and other literals' truth values are not determined by the program. Thus, a partial model can also be viewed as a model in 3-valued logic. This relationship is discussed in Section 4.

While stratification is a syntactic property of the IDB, for an unstratified IDB, whether the program has a total well-founded model depends in general on the EDB. One view of well-founded semantics is as an attempt to give a reasonable meaning to as much of the program as possible in the unfavorable cases, when only a partial model exists, as an extension of the semantics for the favorable cases, which have a total model.

The key idea in our formulation is the concept of an "unfounded set," which is an adaptation of the "closed set" developed for disjunctive databases by Ross and Topor [35], and is essentially the same as the "securable set" in [36]. Unfounded sets are defined in Section 3.

Since the preliminary version of this paper was presented at a conference [43], several alternative formulations of negation that appear to be equivalent to the well-founded semantics have been developed [3, 8, 32, 42]. We believe that this indicates a robustness of the semantics, and provides evidence that it coincides well with "common sense" and intuition.

2. General Logic Programs and Partial Interpretations

In this section we introduce our notation and basic definitions, and describe the class of general logic programs that we shall be considering in this paper.

Definition 2.1. A *general logic program* is a finite set of *general rules*, which may have both positive and negative subgoals. A general rule is written with its *head*, or conclusion on the left, and its subgoals (body), if any to the right of the symbol "←," which may be read "if." For example,

$$p(X) \leftarrow a(X),\ \mathbf{not}\ b(X).$$

is a rule in which $p(X)$ is the head, $a(X)$ is a positive subgoal, and $b(X)$ is a negative subgoal. This rule may be read as "$p(X)$ if $a(X)$ and not $b(X)$." A *Horn rule* is one with no negative subgoals, and a *Horn logic program* is one with only Horn rules.
□

Lloyd has recently adopted the word "normal" instead of "general" to describe rules whose bodies consist of a conjunction of literals, and programs of such rules [20]. He reserves the word "general" to allow more involved constructs, such as

$$w(X) \leftarrow m(X,Y),\ \mathbf{not}\ (m(Y,Z),\ \mathbf{not}\ w(Z)).$$

where the first **not** applies to a conjunction rather than an atom. Although we avoid such constructs for simplicity of presentation, the well-founded semantics is easily generalized to such syntax, so we continue to use the word "general."

In keeping with Prolog's convention, logical variables begin with a capital letter; constants, functions, and predicates begin with a lowercase letter. We use the same symbol, e.g., p, to refer to both a predicate and its relation. The arguments of a predicate are terms as customarily defined in logic:

1. A variable or constant is a term.

2. A function symbol with terms as arguments is a term.

Terms may also be viewed as data structures of the program, with function symbols serving as record names. The word *ground* is used as a synonym for "variable-free," in keeping with common practice. Often a constant is treated as a function symbol of arity zero.

The *Herbrand universe* is the set of ground terms that use the function symbols and constants that appear in the program.[2] The *Herbrand base* is the set of atomic formulas formed by predicate symbols in the program whose arguments are in the Herbrand universe. If the program contains a function symbol of positive arity, then the Herbrand universe and Herbrand base are countably infinite; otherwise they are finite.

We shall be considering atoms in the Herbrand base and ground rules whose variables have been instantiated to elements of the Herbrand universe, which we call *instantiated* rules.

Definition 2.2. The *Herbrand instantiation* of a general logic program is the set of rules obtained by substituting terms in the Herbrand universe for variables in every possible way. An *instantiated rule* is one in the Herbrand instantiation. Whereas "uninstantiated" logic programs are assumed to be a finite set of rules, instantiated logic programs may well be infinite.
□

Certain programs exhibit a property called *unsafe negation*, which can cause anomalous behavior if interpreted in the Herbrand universe. Appendix A explains a way to "augment" such programs by introducing an extra rule that removes the anomalies by enlarging the Herbrand universe. Our development is independent of whether this augmentation is used or not.

We shall be working extensively with sets of literals, for which we now introduce some notation and definitions. If p is an atomic formula (atom), then p is its *positive* literal, $\neg p$ is its *negative* literal, and these two literals are said to be *complements* of each other.

Definition 2.3. For a set of literals S we denote the set formed by taking the complement of each literal in S by $\neg \cdot S$.

- We say literal q is *inconsistent* with S if $q \in \neg \cdot S$.

- Sets of literals R and S are *inconsistent* if some literal in R is inconsistent with S, i.e., if

$$R \cap \neg \cdot S \neq \emptyset$$

- A set of literals is *inconsistent* if it is inconsistent with itself; otherwise it is *consistent*.

  □

[2] If there is no constant symbol in the program, then one is added arbitrarily.

Definition 2.4. Given a program **P**, a *partial interpretation* I is a consistent set of literals whose atoms are in the Herbrand base of **P**. A *total interpretation* is a partial interpretation that contains every atom of the Herbrand base or its negation. We say a ground (variable-free) literal is *true in* I when it is in I and say it is *false in* I when its complement is in I. Similarly, we say a conjunction of ground literals is *true* in I if all of the literals are true in I, and is *false* in I if any of its literals is false in I.
$\square$

Definition 2.5. We say that an instantiated rule is *satisfied* in a partial or total interpretation I if the head is true in I or some subgoal is false in I; it is *falsified* if the head is false and all subgoals are true. In addition, if the head of the rule is false in I, but no subgoal is false in I then we say that the rule is *weakly falsified* in I.
$\square$

Definition 2.6. A *total model* of a program **P** is a total interpretation such that every instantiated rule of **P** is satisfied. A *partial model* of **P** is a partial interpretation that can be extended to a total model of **P**.
$\square$

Although it is customary to omit the adjective "total" when speaking of interpretations and models, because we shall be dealing with both 2-valued and 3-valued logics, we shall include it for clarity.

Intuitively, a partial interpretation may contain incomplete information: the positive literals in it are considered to be true atomic facts; the negative literals denote atoms considered to be false; and the truth values of the rest of the atomic facts are unknown, or unspecified, at least "at present." The natural ordering on partial interpretations is $\subseteq$. The idea is that $I \subseteq I'$ if I' contains all the information in I, both positive and negative, plus possibly more.

For us, a partial model is a partial interpretation I such that some instantiated rules may not be satisfied, but there is a (possibly empty) set of literals whose addition to the partial interpretation will satisfy all rules. Clearly, this is impossible if I falsifies any instantiated rule. If I only weakly falsifies some instantiated rule, then the addition of some *negative* literal to I may be necessary to satisfy that rule. Thus recognition of partial models containing weakly falsified rules may be very difficult. The following lemma shows that the situation is much simpler if I does not weakly falsify any instantiated rule.

Lemma 2.1. Let **P** be a program and let I be a partial interpretation. If I weakly falsifies no instantiated rule from **P**, then I is a partial model of **P**.

Proof. Let I' be the total interpretation formed by adding to I all atoms in the Herbrand base that are neither true nor false in I. Let r be an instantiated rule from **P**. If I satisfies r, then clearly so does I'. If I does not satisfy r, then the head of r cannot be false in I, so it is true in I'. Hence I' is a total model. $\square$

Our notion of partial model is not the same as the natural notions of models used in 3-valued logics, such as in the approaches of Fitting [9] and Kunen [17]. Nevertheless, the well-founded partial model we construct will also be a model in Fitting's 3-valued sense. We shall discuss 3-valued models in Section 4.

3. Unfounded Sets and Well-Founded Partial Models

In this section we define *unfounded sets*, which are a variation of *closed sets* that were defined for disjunctive databases by Ross and Topor in [35]. Unfounded sets provide the basis for negative conclusions in the well-founded semantics.

3.1. Unfounded Sets

Definition 3.1. Let a program $\mathbf{P}$, its associated Herbrand base H, and a partial interpretation I be given. We say $A \subseteq H$ is an *unfounded set* (of $\mathbf{P}$) *with respect to* I if each atom $p \in A$ satisfies the following condition: For each instantiated rule R of $\mathbf{P}$ whose head is p, (at least) one of the following holds:

1. Some (positive or negative) subgoal q of the body is false in I.

2. Some positive subgoal of the body occurs in A.

A literal that makes (1) or (2) above true is called a *witness of unusability* for rule R (with respect to I). □

Intuitively, we regard I as what we already know about the intended model of $\mathbf{P}$ (possibly partial). Rules satisfying condition (1) are not usable for further derivations since their hypotheses are already known to be false.

Condition (2) is the unfoundedness condition: of all the rules that still might be usable to derive something in the set A, each requires an atom in A to be true. In other words, there is no one atom in A that can be *first* to be established as true by the rules of $\mathbf{P}$ (starting from "knowing" I). Consequently, if we choose to infer that some or all atoms in A are false, there is no way we could later have to infer one to be true.

As described more formally later, the well-founded semantics uses conditions (1) and (2) to draw negative conclusions. Essentially, it simultaneously infers all atoms in A to be false. By contrast, the semantics of [9] uses only condition (1) to draw negative conclusions. The closed sets of Ross and Topor [35] were defined only with condition (2).

Example 3.1. Consider the program consisting of the eight (instantiated) rules below.

$$\begin{aligned}
&p(a) \leftarrow p(c),\ \mathbf{not}\ p(b).\\
&p(b) \leftarrow \mathbf{not}\ p(a).\\
&p(e) \leftarrow \mathbf{not}\ p(d).\\
&p(c).\\[4pt]
&p(d) \leftarrow q(a),\ \mathbf{not}\ q(b).\\
&p(d) \leftarrow q(b),\ \mathbf{not}\ q(c).\\
&q(a) \leftarrow p(d).\\
&q(b) \leftarrow q(a).
\end{aligned}$$

The atoms $\{p(d), q(a), q(b), q(c)\}$ form an unfounded set with respect to $\emptyset$. In particular, $\{q(c)\}$ is unfounded due to Condition (1); there is no rule usable to establish its truth. The set $\{p(d), q(a), q(b)\}$ is unfounded due to Condition (2); we are given no way to establish $p(d)$ without first establishing $q(a)$ or establishing $q(b)$ (whether we can establish $\neg q(b)$ to support the first rule for $p(d)$ is irrelevant for determining unfoundedness).

Also, there is no way to establish $q(a)$ without first establishing $p(d)$, and no way to establish $q(b)$ without first establishing $q(a)$. Clearly $q(c)$ can never be proven, but we can also see that among $p(d)$, $q(a)$, and $q(b)$, none can be the *first* to be proven.

In contrast, the pair $\{p(a), p(b)\}$ does not form an unfounded set even though they depend on each other, because the only dependence is "through" negation. It is tempting to claim that the proof attempts for $p(a)$ and $p(b)$ will fail also, but such a claim is faulty.

The difference between sets $\{p(d), q(a), q(b)\}$ and $\{p(a), p(b)\}$ is this: Declaring any of $p(d)$, $q(a)$, or $q(b)$ false does not create a proof that any other element of the set is *true*. However, as soon as one of $p(a)$ or $p(b)$ is declared false, it becomes possible to prove the other is true. And if both are declared false at once, we have an inconsistency.

The treatment of $p(a)$ and $p(b)$ has something of the flavor of the *generalized closed world assumption* (GCWA), in that $(p(a) \vee p(b))$ is a (2-valued) logical consequence of the program interpreted as indefinite disjunctive clauses; consequently GCWA also declines to consider them false. However, GCWA behaves quite differently in general. For example, $(p(e) \vee p(d))$ is also a logical consequence, so GCWA does not consider $p(d)$ false, whereas the well-founded semantics does. Similar remarks apply to $q(a)$ and $q(b)$. (However, $q(c)$ *is* considered false by GCWA; it is in the positive disjunction $(q(c) \vee p(d) \vee p(e))$, but this disjunction is not *minimal*.) As a further difference, after $p(d)$ is classified as false in the well-founded semantics, $p(e)$ will become derivable. It is a property of GCWA that the atoms considered false cannot be used to support any further derivations.

□

Simultaneously negating all the atoms in an unfounded set generalizes negation by failure in Horn clause programs; if H is the Herbrand base and I is the set of atoms that represents the minimum Herbrand model of a Horn clause program [39], then $H - I$, the set of atoms not in I, is unfounded with respect to I.

We now formalize the intuition of the preceding discussion. It is immediate that the union of arbitrary unfounded sets is an unfounded set. This leads naturally to:

Definition 3.2. The *greatest unfounded set (of* $\mathbf{P}$*) with respect to* I, denoted $\mathbf{U}_P(I)$, is the union of all sets that are unfounded with respect to I.

□

We now make some easy, but instructive, observations about unfounded sets. To a certain extent, there is a flexibility between having $\neg p \in I$ and having p in an unfounded set. The following lemma shows that, given an interpretation R, if we deduce that certain facts S are in an unfounded set A and add their complements to R, other unfounded atoms remain unfounded.

Lemma 3.1. Let R be a set of literals, and let A be an unfounded set of $\mathbf{P}$ with repect to R. For any subset $S \subseteq A$, $A - S$ is unfounded with respect to $R \cup \neg \cdot S$.

Proof. Any witness of unusability that was an atom in S is now a negative literal in $\neg \cdot S$, and hence is still a witness. □

The next lemma demonstrates a connection between (lack of) weak falsification (Definition 2.5) and unfounded sets. Recall from Lemma 2.1 that I in the next lemma is a partial model.

Lemma 3.2. Let I be a partial interpretation consisting of positive literals Q and negative literals $\neg \cdot S$. If I does not weakly falsify any instantiated rule of program $\mathbf{P}$, then S is an unfounded set with respect to Q.

Proof. Let $p \in S$ and let R be any instantiated rule whose head is p. Because R is not weakly falsified, some subgoal of R is false in I. If this subgoal is positive, it is also in S, so condition (2) of Definition 3.1 is satisfied. If this subgoal is negative, its positive version is in Q so condition (1) is satisfied. □

3.2. Well-Founded Partial Models

We now consider a (possibly transfinite) sequence that results from combining two set transformations. The limit of this sequence defines the well-founded semantics. In what follows the word *transformation* always means a transformation between sets of literals, where their atoms are in the Herbrand base of a given program $\mathbf{P}$. We recall that a transformation T is called *monotonic* if $T(I) \subseteq T(J)$, whenever $I \subseteq J$.

Definition 3.3. Transformations $\mathbf{T}_P$, $\mathbf{U}_P$, and $\mathbf{W}_P$ are defined as follows:

- $p \in \mathbf{T}_P(I)$ if and only if there is some instantiated rule R of $\mathbf{P}$ such that R has head p, and each subgoal literal in the body of R is true in I.
- $\mathbf{U}_P(I)$ is the greatest unfounded set of $\mathbf{P}$ with respect to I, as in Definition 3.2.
- $\mathbf{W}_P(I) = \mathbf{T}_P(I) \cup \neg \cdot \mathbf{U}_P(I)$.

  □

Lemma 3.3. $\mathbf{T}_P$, $\mathbf{U}_P$, and $\mathbf{W}_P$, are monotonic transformations.

Proof. Immediate from definitions. □

We wish to emphasize that, unlike some other methods, our $\mathbf{T}_P$ treats positive and negative subgoals symmetrically. In deciding whether a negative subgoal **not** p is true, some methods look for the absence of p from I. For us the presence or absence of p is immaterial for the truth of the subgoal **not** p; we require the presence of $\neg p$.

Definition 3.4. Let α range over all countable ordinals. The sets I_α and I^∞, whose elements are literals in the Herbrand base of a program $\mathbf{P}$, are defined recursively by:

1. For limit ordinal α,
$$I_\alpha = \bigcup_{\beta < \alpha} I_\beta$$
   Note that 0 is a limit ordinal, and $I_0 = \emptyset$.
2. For successor ordinal $\alpha = \gamma + 1$,
$$I_{\gamma+1} = \mathbf{W}_P(I_\gamma)$$
3. Finally, define
$$I^\infty = \bigcup_\alpha I_\alpha$$

Following Moschovakis [29], for any literal p in I^∞, we define the *stage* of p to be the least ordinal α such that $p \in I_\alpha$. We observe that the *stage* is always a successor ordinal for literals in I^∞.
$\square$

Lemma 3.4. I_α as defined in Definition 3.4 is a monotonic sequence of partial interpretations (i.e., is consistent).

Proof. The proof is by induction on α. The basis, $\alpha = 0$, is immediate. For $\alpha > 0$, assume the lemma is true for $\beta < \alpha$.

For monotonicity, first let $\alpha = \gamma + 1$ be a successor ordinal. if literal $q \in I_\gamma$, there is a smallest $\beta < \gamma$ such that $q \in \mathbf{W}_P(I_\beta)$ (even if γ is a limit ordinal). But $\mathbf{W}_P$ is monotonic, so by the inductive hypothesis $q \in \mathbf{W}_P(I_\gamma)$. Monotonicity for limit α follows from the definition of I_α.

To show consistency for successor ordinal $\alpha = \gamma + 1$, note that every literal in I_α *first* appears in some $I_{\beta+1}$, i.e., at a successor ordinal "stage". Let A be any set of positive ground literals that has a nonempty intersection with (the positive literals of) $I_{\gamma+1}$. It is sufficient to show that A is not unfounded w.r.t. I_γ, for then the *greatest* unfounded set of I_γ is also disjoint from the positive part of $I_{\gamma+1}$. Choose the earliest $I_{\beta+1}$ that intersects A and select an atom p in that intersection. Then p was derived by some rule R all of whose subgoals are in I_β. By the inductive hypothesis, those subgoals are also in I_γ, and I_γ is consistent, so none of the subgoals is false in I_γ. By the choice of β, they are not in A. Thus rule R has no witness of unusability, which demonstrates that A is not an unfounded set w.r.t. I_γ.

For limit ordinal $\alpha > 0$, to show that I_α is a partial interpretation, assume the lemma is true for $\beta < \alpha$. If both q and $\neg q$ are in I_α, there is some *successor* ordinal $\gamma + 1 < \alpha$ such that the same is true. This contradicts the inductive hypothesis. $\square$

It follows by classical results of Tarski that I^∞ is the least fixed point of the operator $\mathbf{W}_P$. The Herbrand base is countable, so for some countable ordinal α, $I^\infty = I_\alpha$.

Definition 3.5. The *closure ordinal* for the sequence I_α is the least ordinal α such that $I^\infty = I_\alpha$ (*cf.* [29]).
$\square$

Examples can be constructed where the closure ordinal is above ω, but the authors believe such examples to be very rare in practical logic programming. In the case of a function-free program with a finite EDB, which is common in deductive databases, the limit is reached after a finite ordinal. The "data complexity" of this case is discussed in Section 8.

Definition 3.6. The *well-founded semantics* of a program $\mathbf{P}$ is the "meaning" represented by the least fixed point of $\mathbf{W}_P$, or the limit I^∞ described above; every positive literal denotes that its atom is true, every negative literal denotes that its atom is false, and missing atoms have no truth value assigned by the semantics.
$\square$

Lemma 3.5. Let I_α be as defined in Definition 3.4. Then I_α does not weakly falsify (Definition 2.5) any instantiated rule of $\mathbf{P}$.

Proof. Let R be any instantiated rule with head p such that $\neg p \in I_\alpha$. We need to show that the body of R is false in I_α. By definition, $p \in \mathbf{U}_P(I_\beta)$ for some $\beta < \alpha$. By Lemma 3.4, $I_\beta \subseteq I_{\beta+1} \subseteq I_\alpha$. Either the body of R is false in I_β, or some subgoal q of the body of R is in the greatest unfounded set w.r.t. I_β. In the latter case, $\neg q \in I_{\beta+1}$, so the body of R is false in $I_{\beta+1}$. In either case, it follows that the body of R is false in I_α. $\square$

Theorem 3.6. For each countable ordinal α, I_α in the sequence described in Definition 3.4 is a partial model of $\mathbf{P}$.

Proof. Immediate by Lemmas 2.1 and 3.5. $\square$

Definition 3.7. Suppose that for each p in the Herbrand Base I^∞ contains either p or $\neg p$, i.e. I^∞ is a total interpretation. Then by the above theorem, I^∞ is a total model, and we call this the *well-founded model*; otherwise we call I^∞ the *well-founded partial model*.
$\square$

Theorem 3.7. Every Horn program has a well-founded model I^∞, which is the minimum model in the sense of Van Emden and Kowalski [39], i.e., its positive literals are contained in every Herbrand model.

Proof. Let H be the Herbrand base and let Q be the set of positive literals of I^∞. Q is a fixed point of $\mathbf{T}_P$ [39]. In view of Theorem 3.6 it is sufficient to show that $H - Q \subseteq \mathbf{U}_P(I^\infty)$. Let p be any positive literal in $H - Q$. Each rule for p must have a positive subgoal that is also in $H - Q$, which subgoal is a witness of unusability for this rule. Thus $H - Q$ is unfounded w.r.t. $\emptyset$, and *a fortiori* w.r.t. I^∞. $\square$

4. Three-Valued Models of the Program Completion

The relationship of the well-founded semantics to other methods based on program completion and 3-valued logics is discussed in this section. Clark introduced the completed program as a way of formalizing the notion that facts not inferable from the rules in the program were to be regarded as false [6]. Fitting studied models of the completed program in a 3-valued logic, and showed that all such models were fixed points of a certain operator [9]. We show that the well-founded partial model is also a model in this logic, but often not the least model.

The idea behind the Clark completion of a program is to collect all rules having the same head predicate into a single rule whose body is a disjunction of conjunctions, then replace the "if" symbol, "$\leftarrow$," by "$\leftrightarrow$." This states in effect that the predicate is completely defined by the given rules. The formal details, including handling of variables and introduction of axioms for equality, are described in several places [6, 2, 20, 9, 17].

Example 4.1. Recall the last four rules of Example 3.1, whose atoms formed an unfounded set:

$$p(d) \leftarrow q(a), \textbf{not } q(b).$$
$$p(d) \leftarrow q(b), \textbf{not } q(c).$$
$$q(a) \leftarrow p(d).$$
$$q(b) \leftarrow q(a).$$

The Clark completion combines the rules for p into one rule, combines the rules for q into another rule, then replaces "$\leftarrow$" by "$\leftrightarrow$". After some simplifications to eliminate bound variables, there results:

$$p(d) \leftrightarrow (q(a) \wedge \neg q(b)) \vee (q(b) \wedge \neg q(c)))$$
$$\forall X\, [q(X) \leftrightarrow ((X = a) \wedge p(d)) \vee ((X = b) \wedge q(a))]$$

The equality freeness axioms (often called the *Clark Equality Theory* or CET) are also part of the completed program. Roughly, they require a one to one interpretation of the terms, so that $q(c)$ cannot be made true by setting $c = a$ or $c = b$.
$\square$

The original "logical consequence" approach essentially declares that only conclusions that are logical consequences (in the classical, 2-valued sense) of the completed program should be inferred [6, 15, 20, 37]. When the completed program is consistent, this approach implicitly defines a 3-valued interpretation: assign value *true* to instantiated atoms that are true in all (2-valued, not necessarily Herbrand) models of the completed program, *false* to instantiated atoms that are false in all models, and $\perp$ (unknown) to all other instantiated atoms. However, because the truth of each literal is based on traditional 2-valued logic, we call this the *2-valued program completion* (2PC) interpretation.

The 3-valued interpretations were made explicit by Fitting [9] and Kunen [17], who also used 3-valued logic to evaluate formulas. Whereas $(p \vee \neg p)$ must be true in 2-valued logic, in 3-valued logic it may also be $\perp$. In addition, the "$\leftrightarrow$" produced by the program completion process was interpreted as Łukasiewicz's operator of "having the same truth value," so that $\perp \leftrightarrow \perp$ evaluates to true. Fitting's and Kunen's treatments eliminated some anomalies in the 2PC interpretation.

Example 4.2. Consider the single rule program

$$p \leftarrow \textbf{not}\ p,\ \textbf{not}\ q.$$

The Clark completion is

$$\begin{aligned} &p \leftrightarrow (\neg p \wedge \neg q) \\ &q \leftrightarrow \mathit{false} \end{aligned}$$

which has no 2-valued model. (The second rule derives from *false* representing the empty disjunction of q's rule bodies.) However, if we add the "meaningless" rule, $p \leftarrow p$, the completed program changes to:

$$\begin{aligned} &p \leftrightarrow (\neg p \wedge \neg q) \vee p \\ &q \leftrightarrow \neg \mathit{true} \end{aligned}$$

which has the unique 2-valued model, $\{p, \neg q\}$. If, instead, we add the "meaningless" rule, $q \leftarrow q$, the completed program changes to:

$$\begin{aligned} &p \leftrightarrow (\neg p \wedge \neg q) \\ &q \leftrightarrow q \end{aligned}$$

which has a different 2-valued model, $\{\neg p, q\}$. However, all three versions have 3-valued models in which $p = \perp$.

Finally, as suggested by a referee, if we add several rules, giving:

$$p \leftarrow \textbf{not } p, \textbf{ not } q.$$
$$q \leftarrow r.$$
$$q \leftarrow s.$$
$$r \leftarrow r.$$
$$s \leftarrow s.$$

the completed program becomes:

$$p \leftrightarrow (\neg p \wedge \neg q)$$
$$q \leftrightarrow (r \vee s)$$
$$r \leftrightarrow r$$
$$s \leftrightarrow s$$

Now there are three 2-valued models, which vary on whether r or s or both are true. Their common part (intersection) is the same 2PC interpretation as above, $\{\neg p, q\}$. However, here the 2PC interpretation is *not a 3-valued model*.

□

One principal result in [9] is that the completion of every program has a (unique) minimum 3-valued Herbrand model. Fitting suggests that this model be taken for the semantics of the program, and hereafter we call it the *Fitting model*. Thus the Fitting model is sometimes "less defined" than the 2PC interpretation, as in the previous example. However, Example A.1 in Appendix A shows that the 2PC interpretation can be "less defined" than the Fitting model.

To any partial interpretation I (in 2-valued logic) there corresponds the obvious 3-valued interpretation in which atoms missing from I are assigned the truth value $\perp$. In this setting, our partial interpretations are the same as Fitting's basic sets [9]. In 3-valued logic literals and conjunctions are true and false in I as specified in Definition 2.4; in addition, the truth value $\perp$ may be assigned:

Definition 4.1. Literal q is called *undefined in I*, denoted by "$\perp$", if neither q nor its complement is in I. A conjunction of literals evaluates to *undefined* in I if no literal in the conjunction is false in I and at least one is undefined in I.

□

Definition 4.2. $\mathbf{N}_P$ is defined as the transformation that, for I a 3-valued interpretation, gives as $\mathbf{N}_P(I)$ the set of atoms p such that for every rule in the Herbrand instantiation of $\mathbf{P}$ with p as its head, the body is false in I, i.e., some subgoal of the rule is false in I. Note that $\mathbf{N}_P$ is the portion of $\mathbf{U}_P$ produced by condition (1) of Definition 3.1.

□

Fitting also constructs 3-valued models with a fixed point operator [9]. For positive inferences, $\mathbf{T}_P$ is as in Definition 3.3. For negative inferences he uses (in effect) the transformation $\mathbf{N}_P(I)$ defined above. A second main theorem of that approach is:

Theorem 4.1. (Fitting) A 3-valued interpretation I is a 3-valued model of the completed program if and only if $I = \mathbf{T}_P(I) \cup \neg \cdot \mathbf{N}_P(I)$. $\square$

This immediately yields a fixed point construction for 3-valued models, and the Fitting model is the least fixed point. We now show that the well-founded partial model is also a 3-valued model in Fitting's sense.

Theorem 4.2. Let I^∞ be as defined in Definition 3.4. Then $I^\infty = \mathbf{T}_P(I^\infty) \cup \neg \cdot \mathbf{N}_P(I^\infty)$. Hence, I^∞ is a 3-valued model of the completion of the logic program.

Proof. Since $I^\infty = \mathbf{T}_P(I^\infty) \cup \neg \cdot \mathbf{U}_P(I^\infty)$, it follows that

1. $T_P(I^\infty) \cup \neg \cdot \mathbf{N}_P(I^\infty) \subseteq I^\infty$, and
2. every positive literal in I^∞ is in $\mathbf{T}_P(I^\infty)$.

It remains to show that every negative literal $\neg p$ that is in I^∞ is also in $\neg \cdot \mathbf{N}_P(I^\infty)$. But by Lemma 3.5 each instantiated rule with head p has its body false in I^∞, so $p \in \mathbf{N}_P(I^\infty)$. $\square$

Corollary 4.3. The Fitting model is a subset of I^∞. $\square$

I^∞ can indeed differ from the smallest 3-valued model of the completion of the program, and need not even be a subset of all 2-valued models, as shown by the one-rule program, $p \leftarrow p$, in which p is false in I^∞ and is undefined in the Fitting model.

Kunen describes a variant that differs from Fitting's in two important ways: (1) the iteration is always stopped at ω, and (2) the Herbrand universe is defined with respect to a language with an infinite set of function symbols, which properly includes those that occur in the program [17]. The resulting 3-valued interpretation is recursively enumerable, but may not be a 3-valued model. Kunen's main theorem is that this interpretation characterizes the 3-valued logical consequences of the completed program.

5. Stable Models

Gelfond introduced an approach to negation through stable models [10], and motivated it by appealing to autoepistemic logic, as developed by Moore [26]. The theory has been further developed by Gelfond and Lifschitz [11], and also by Marek and Truszczynski [23, 24].

In this section we follow the definition of [11], which defines stability without reference to autoepistemic logic. We show that if a program has a total well-founded model, that model is the unique stable model. We also discuss two programs which do not have total well-founded models but do have unique stable models. Whether inferring (or not inferring) the truth of these extra literals is "a bug or a feature" of either approach we leave for the reader's judgement.

Gelfond and Lifschitz [11] define a stable model to be one that reproduces itself in a certain three stage transformation, which we call the *stability transformation*. If a program has only one stable model, that is called its unique stable model. Stable models refer to 2-valued logic. When speaking of total, or 2-valued, interpretations, it is more common to represent models as sets of ground atoms, with the understanding that missing atoms represent the negative literals. In this context a "minimal model" is one that has a minimal set of *positive* literals, and a "monotonic transformation" on total interpretations is one that is monotonic in terms of the positive literals alone. However, for consistency with the rest of the paper, we shall represent models as sets of literals, and use the following notation for sets of positive and negative atoms in interpretations.

Definition 5.1. For any partial interpretation I, let $\mathbf{Pos}(I)$ be the set of positive literals in I, and let $\mathbf{Neg}(I)$ be the set of atoms that represent negative literals in I. Thus $I = \mathbf{Pos}(I) \cup \neg \cdot \mathbf{Neg}(I)$.
□

Definition 5.2. Given a general logic program $\mathbf{P}$, and its Herbrand instantiation, $\mathbf{P}_H$, we define $\mathbf{S}$, the *stability transformation* from total interpretations into total interpretations. Given a total interpretation I, its transformation $\mathbf{S}(I)$ is defined in the following three stages:

1. Define
$$\mathbf{P}' = T_1(\mathbf{P}_H, I)$$
where T_1 is the following transformation: For each rule instantiation, if it contains a *negative* subgoal that is inconsistent with I, then the rule instantiation is discarded. The output of the transformation is the set of rule instantiations that remain.

2. Define
$$\mathbf{P}'' = T_2(\mathbf{P}')$$
where T_2 is the transformation by which all negative subgoals are dropped from rules of $\mathbf{P}'$, leaving a Horn program. We call $\mathbf{P}''$ the *reduction of* $\mathbf{P}$ *with respect to* I.

3. Since $\mathbf{P}''$ is a Horn program, we can form its minimum (2-valued) model as in the standard Van Emden and Kowalski semantics [39]. In this context, "minimum," means that the set of positive literals is minimized, and hence the set of negative literals is maximized.

We define $\mathbf{S}(I)$ to be this minimum model of $\mathbf{P}''$.
□

Example 5.1. Let $\mathbf{P}_H$ be
$$p \leftarrow \mathbf{not}\ p.$$
$$a \leftarrow \mathbf{not}\ b.$$
$$b \leftarrow \mathbf{not}\ a.$$
and let $\mathcal{M} = \{a, \neg b, p\}$, which is a minimal model of $\mathbf{P}_H$. Then $\mathbf{P}'$ consists only of
$$a \leftarrow \mathbf{not}\ b.$$
because the other rules contain negative subgoals whose atoms are in $\mathbf{Pos}(\mathcal{M})$. Now $\mathbf{P}''$ is the Horn rule
$$a.$$
Thus $\mathbf{S}(\mathcal{M}) = \{a, \neg b, \neg p\}$, which, incidentally, is not a model of $\mathbf{P}_H$.
□

The name "stability transformation" is justified in a sense by the following lemma, which shows that $\mathbf{S}$ is a "shrinking" transformation (on positive literals) when applied to total models. However, as shown above, it is possible that $\mathcal{M}$ is a model and $\mathbf{S}(\mathcal{M})$ is not a model; it may "shrink" too much.

Lemma 5.1. Let $\mathcal{M}$ be a total model of general logic program $\mathbf{P}$. Then $\mathbf{Pos}(\mathbf{S}(\mathcal{M})) \subseteq \mathbf{Pos}(\mathcal{M})$.

Proof. Using the terminology of Definition 5.2, $\mathcal{M}$ is a total model of $\mathbf{P}'$ and also of $\mathbf{P}''$, by their construction. But $\mathbf{S}(\mathcal{M})$ is the *minimum* total model of $\mathbf{P}''$. □

The models that are fixed points of $\mathbf{S}$ are of special interest.

Definition 5.3. A total model $\mathcal{M}$ of general logic program $\mathbf{P}$ is *stable* if it is a fixed point of $\mathbf{S}$; that is, if $\mathcal{M} = \mathbf{S}(\mathcal{M})$. If program $\mathbf{P}$ has exactly one stable model, that model is called the *unique stable model* of $\mathbf{P}$.
□

It is immediate that a stable model is minimal (in terms of the set of positive literals), but not every minimal model is stable, as shown in Example 5.1 above and in Example 5.3 below.

Example 5.2. Let $\mathbf{P}_1$ be

$$a \leftarrow \mathbf{not}\ b.$$
$$b \leftarrow \mathbf{not}\ a.$$

Both $\{a, \neg b\}$ and $\{b, \neg a\}$ are stable models, so $\mathbf{P}_1$ has no unique stable model. Its Fitting model, 2PC interpretation, and well-founded partial model are $\emptyset$.
□

Example 5.3. For another example, let $\mathbf{P}_2$ be

$$p \leftarrow \mathbf{not}\ p.$$

The only model of $\mathbf{P}_2$ is $\mathcal{M} = \{p\}$. The one rule in the program drops out of the reduction, making $\mathbf{S}(\mathcal{M}) = \{\neg p\}$ the minimum model of the reduction of $\mathbf{P}_2$. Hence $\mathbf{P}_2$ has no stable model.

As discussed in Example 4.2, the completed program is $p \leftrightarrow \neg p$. Its 2PC interpretation in 2-valued logic is inconsistent. Its Fitting model and well-founded partial model are $\emptyset$.
□

There is a close relationship between stable models and well-founded (partial or total) models. As defined, a unique stable model is demonstrated only through the explicit enumeration of all minimal models followed by testing each for stability. We shall show that well-founded total models are unique stable models. This offers a method to generate the unique stable model directly[3] in such programs. The next lemmas illustrate the close relationship by showing that, for total models, the negative part of the stability transformation $\mathbf{S}$ agrees with the greatest unfounded set U_P, while the positive part of $\mathbf{S}$ is contained in $\mathbf{T}_P$.

Lemma 5.2. Let $\mathcal{M}$ be a total model of a program $\mathbf{P}$. Then $\mathbf{Neg}(\mathbf{S}(\mathcal{M})) = \mathbf{U}_P(\mathcal{M})$.

Proof. Form Horn program $\mathbf{P}''$ as in Definition 5.2, and let $\mathcal{M}' = \mathbf{S}(\mathcal{M})$ be its minimum total model.

First we show that $\mathbf{U}_P(\mathcal{M}) \subseteq \mathbf{Neg}(\mathcal{M}')$. Since $\mathcal{M}'$ is total, it suffices to show that, for any positive literal p, if $p \in \mathbf{Pos}(\mathcal{M}')$ then $p \notin \mathbf{U}_P(\mathcal{M})$. We prove this by induction on the stages of the (Van Emden

[3] if you consider possibly transfinite iteration direct!

and Kowalski type) construction of $\mathcal{M}'$. It is true vacuously for stage 0, which is empty. For stage $k > 0$, suppose positive literal p is derived in stage k of the construction of $\mathcal{M}'$. Then there is a rule

$$p \leftarrow a_1, \ldots, a_k$$

in $\mathbf{P}''$ such that the a_i's have been derived in stages less than k. This rule corresponds to some rule in $\mathbf{P}'$,

$$p \leftarrow a_1, \ldots, a_k, \mathbf{not}\ b_1, \ldots, \mathbf{not}\ b_n$$

such that each $b_j \in \mathbf{Neg}(\mathcal{M})$, which in turn corresponds to a rule in P_H. By Lemma 5.1, all the a_i's are also in $\mathbf{Pos}(\mathcal{M})$. Since $\mathcal{M}$ is consistent, none of the subgoals, the a_i's or the **not** b_j's, are false in $\mathcal{M}$. Finally, by the inductive hypothesis, none of the a_i's are in $\mathbf{U}_P(\mathcal{M})$. Hence, by virtue of this P_H rule, $p \notin \mathbf{U}_P(\mathcal{M})$.

We prove that $\mathbf{Neg}(\mathcal{M}') \subseteq \mathbf{U}_P(\mathcal{M})$. It suffices to show that $\mathbf{Neg}(\mathcal{M}')$ is an unfounded set of $\mathbf{P}_H$ w.r.t. $\mathcal{M}$. Suppose some $p \in \mathbf{Neg}(\mathcal{M}')$ fails to satisfy some condition of unfoundedness, as defined in Definition 3.1. Then there is a rule

$$p \leftarrow a_1, \ldots, a_k, \mathbf{not}\ b_1, \ldots, \mathbf{not}\ b_n$$

in $\mathbf{P}_H$ such that the following facts hold:

1. no a_i is false in $\mathcal{M}$
2. no b_j is true in $\mathcal{M}$
3. no a_i is true in $\mathbf{Neg}(\mathcal{M}')$

the third fact being the negation of the "unfoundedness" condition. Since $\mathcal{M}$ is total, it follows from the second fact that each b_j is in $\mathbf{Neg}(\mathcal{M})$. Hence

$$p \leftarrow a_1, \ldots, a_k$$

is a rule in $\mathbf{P}''$. Since $\mathcal{M}'$ is total, it follows from the third fact that each $a_i \in \mathbf{Pos}(\mathcal{M}')$. Hence $p \in \mathbf{Pos}(\mathcal{M}')$, a contradiction. $\square$

Lemma 5.3. Let $\mathcal{M}$ be a total model of $\mathbf{P}$. Then $\mathbf{Pos}(\mathbf{S}(\mathcal{M})) \subseteq \mathbf{T}_P(\mathcal{M})$.

Proof. Form program $\mathbf{P}'$ and Horn program $\mathbf{P}''$ as in Definition 5.2, and let $\mathcal{M}' = \mathbf{S}(\mathcal{M})$ be the minimum total model of $\mathbf{P}''$. By Lemma 5.1, $\mathbf{Pos}(\mathcal{M}') \subseteq \mathbf{Pos}(\mathcal{M})$, so we have

$$\mathbf{Pos}(\mathcal{M}') = \mathbf{Pos}(\mathbf{T}_{P''}(\mathcal{M}')) \subseteq \mathbf{Pos}(\mathbf{T}_{P''}(\mathcal{M}))$$

by monotonicity of $\mathbf{T}_{P''}$ (on positive literals). Finally,

$$\mathbf{Pos}(\mathbf{T}_{P''}(\mathcal{M})) = \mathbf{Pos}(\mathbf{T}_{P'}(\mathcal{M})) = \mathbf{Pos}(\mathbf{T}_P(\mathcal{M}))$$

by construction. $\square$

The preceding lemmas lead to the next theorem that being a fixed point of $\mathbf{S}$ is equivalent to being a fixed point of $\mathbf{W}_P$ for total models. In fact, this equivalence extends to all total interpretations because being a fixed point of either transformation ensures that the interpretation is a model. As shown in a later example, it is possible that a fixed point of $\mathbf{S}$ is not the *least* fixed point of $\mathbf{W}_P$, but if it is the least fixed point, that stable model is obviously unique.

Theorem 5.4. Let $\mathcal{M}$ be a total model of $\mathbf{P}$. Then $\mathcal{M}$ is stable if and only if it is a fixed point of $\mathbf{W}_P$.

Proof. Form Horn program $\mathbf{P}''$ as in Definition 5.2, and let $\mathcal{M}' = \mathbf{S}(\mathcal{M})$ be its minimum total model.

($\Leftarrow$) We suppose $\mathcal{M}$ is a fixed point of $\mathbf{W}_P$ and prove it is stable. Since $\mathcal{M}$ is a fixed point of $\mathbf{W}_P$, we have $\mathbf{Neg}(\mathcal{M}) = \mathbf{U}_P(\mathcal{M})$. But, by Lemma 5.2, $\mathbf{Neg}(\mathcal{M}') = \mathbf{U}_P(\mathcal{M})$, also. Hence $\mathcal{M} = \mathcal{M}'$.

($\Rightarrow$) We suppose $\mathcal{M}$ is stable and prove it is a fixed point of W_P. Since $\mathcal{M} = \mathcal{M}'$, by Lemma 5.3, $\mathbf{Pos}(\mathcal{M}) = \mathbf{Pos}(\mathcal{M}') \subseteq \mathbf{T}_P(\mathcal{M})$. But $\mathbf{T}_P(\mathcal{M}) \subseteq \mathbf{Pos}(\mathcal{M})$, since $\mathcal{M}$ is a model of $\mathbf{P}$. So $\mathbf{T}_P(\mathcal{M}) = \mathbf{Pos}(\mathcal{M})$. Again, since $\mathcal{M} = \mathcal{M}'$, by Lemma 5.2, $\mathbf{U}_P(\mathcal{M}) = \mathbf{Neg}(\mathcal{M})$. □

Corollary 5.5. Let I be a total interpretation of $\mathbf{P}$. Then I is a fixed point of $\mathbf{S}$ if and only if it is a fixed point of $\mathbf{W}_P$.

Proof. It is routine to show that if I is a fixed point of either $\mathbf{S}$ or $\mathbf{W}_P$, then every instantiated rule is satisfied. Hence I is a model, and Theorem 5.4 applies. □

Corollary 5.6. If $\mathbf{P}$ has a well-founded total model, then that model is the unique stable model. □

Corollary 5.7. The well-founded partial model of $\mathbf{P}$ is a subset of every stable model of $\mathbf{P}$.

Proof. Every stable model is a fixed point of $\mathbf{W}_P$, and the well-founded partial model is the least fixed point. □

In Examples 5.4 and 5.5 below we show that the converse of Corollary 5.6 is not necessarily true.

We agree with Gelfond and Lifschitz that a model that is *intended* to be associated with a program should be able to "derive itself." However, as shown in later examples, the sense of "deriving itself" differs slightly between well-founded semantics and stable model semantics.

5.1. Comparison of Stable and Well-Founded Approaches

We now compare the well-founded semantics with the stable model semantics. On many programs they are identical, and at first it appeared that the only difference was that the well-founded semantics defined a partial model when there were multiple stable models. However, it turns out that there also are programs with a unique stable model and only a partial well-founded model. In other words, the converse of Corollary 5.6 is not necessarily true. These examples and others show that awkward situations arise for well-founded models and unique stable models when the factoring operation of resolution theorem proving (or the law of the excluded middle, in natural deduction) plays a part. Recall that "factoring" of a ground clause is simply the operation of merging two identical literals.

Factoring enters the picture with a rule of the form

$$p \leftarrow \mathbf{not}\ p, \ldots$$

because, as a disjunctive clause, it can be rewritten as

$$p \vee p \leftarrow \ldots$$

and then the two p literals can be merged. Another manifestation of this phenomenon occurs with a pair of rules,

$$\begin{aligned} p &\leftarrow a, \ldots \\ p &\leftarrow \mathbf{not}\ a, \ldots \end{aligned}$$

Again, as disjunctive clauses, they can be resolved on a, giving

$$p \vee p \leftarrow \ldots$$

and then the two p literals can be combined by factoring. Two-valued logical consequences that can be derived only by using factoring cannot be derived in either the well-founded semantics or the 3-valued program completion approaches (*cf.* Examples 4.2 and 5.3).

Example 5.4. Consider the program $\mathbf{P}_3$ given by the four rules:

$$\begin{aligned} &a \leftarrow \textbf{not } b. \\ &b \leftarrow \textbf{not } a. \\ &p \leftarrow \textbf{not } p. \\ &p \leftarrow \textbf{not } b. \end{aligned}$$

Let us first consider $\mathbf{P}'_3$, consisting of just the first three rules above (*cf.* Example 5.1). The first two rules comprise $\mathbf{P}_1$ of Example 5.2, which had two stable models; the third is $\mathbf{P}_2$ of Example 5.3, which had no stable model. Thus the first three rules alone have two minimal models, neither of which is stable:

$$\{a, \neg b, p\} \qquad \text{and} \qquad \{\neg a, b, p\}$$

The program completion of $\mathbf{P}'_3$ is inconsistent. (Just turn each "$\leftarrow$" into "$\leftrightarrow$".) Not too surprisingly, the well-founded partial model and the Fitting model are empty.

Adding the fourth rule would appear to be meaningless at first glance because p is already a (2-valued) logical consequence of the first three rules, and there is no apparent basis to conclude $\neg b$, anyway. Nevertheless, the fourth rule has a strange effect: it stabilizes precisely one of the two models, and so produces a unique stable model for the full program! Moreover, the program completion of the full $\mathbf{P}_3$,

$$\begin{aligned} &a \leftrightarrow \neg b. \\ &p \leftrightarrow (\neg p \vee \neg b). \end{aligned}$$

now has a 2-valued model. Whereas its well-founded partial model and Fitting model remain empty, the unique stable model of $\mathbf{P}_3$ is

$$\mathcal{M} = \{a, \neg b, p\}$$

To verify this, we note that the reduction of $\mathbf{P}_3$ with respect to $\mathcal{M}$ is

$$\begin{aligned} &a. \\ &p. \end{aligned}$$

This model is also the 2PC model.
□

Example 5.5. Consider the program $\mathbf{P}_4$ given by the four rules:

$$\begin{aligned} &a \leftarrow \textbf{not } b. \\ &b \leftarrow \textbf{not } a. \\ &c \leftarrow a, b. \\ &a \leftarrow \textbf{not } c. \end{aligned}$$

Again the Fitting model and well-founded partial model are $\emptyset$, while the unique stable model exists and agrees with the 2PC model:

$$\mathcal{M} = \{a, \neg b, \neg c\}$$

To verify this, we note that the reduction of $\mathbf{P}_4$ with respect to $\mathcal{M}$ is simply

$$\begin{array}{l} a. \\ c \leftarrow a,\ b. \\ a. \end{array}$$

$\square$

6. Stratified and Locally Stratified Programs

A program is *stratified* if all of its predicates can be assigned a *rank* such that

- no predicate depends positively on one of greater rank, and
- no predicate depends negatively on one of *equal* or greater rank

in any rule [5, 1, 19, 40]. In the context of an IDB and EDB, the EDB, being a set of simple facts, has rank 0. IDB predicates whose defining rules involve no negation also have rank 0. IDB predicates whose only negative dependencies are on rank 0 predicates have rank 1, and so on. Stratifiability is easy to check syntactically; in fact it can be checked by examination of the IDB alone.

The *stratified semantics* of such a program is defined by first drawing all rank 0 inferences in the normal way for Horn programs, and concluding $\neg p$ for all rank 0 *atoms* p that have not been inferred. Note that this is not the usual "negation by failure" because some of these atoms may not have failed *finitely*; *cf.* Example 7.2. The definition of stratified semantics is completed inductively: After all atoms of ranks less than k have been classified as positive or negative, use these literals to derive positive rank k atoms; conclude $\neg q$ for all rank k atoms q that have not been inferred. The result is called the *stratified model*.

It is immediate from Theorem 3.7 that the stratified semantics agrees with the well-founded semantics for rank 0, and it is easy to see that the agreement extends to all ranks. We shall prove a somewhat stronger result below. From another point of view, Van Gelder has shown that stratified programs that satisfy certain other conditions have a model based on "tight derivations" that coincides with the stratified model [40].

Przymusinski carried the above idea to a finer grain by defining a program to be *locally stratified* if each atom in its Herbrand base can be assigned a countable ordinal *rank* such that no atom depends on an atom of greater rank or depends *negatively* on one of *equal* or greater rank in any *instantiated* rule [31]. Note that the program is *stratified* if all atoms with the same predicate symbol can be assigned the same rank. The extension handles situations where the "recursive negation" is apparent, but not real. A typical example is the program

$$\begin{array}{l} even(s(X)) \leftarrow \textbf{not}\ even(X). \\ even(0). \end{array}$$

where each ground atom can be given a rank equal to the power of s in its argument.

To give a semantics to locally stratified programs Przymusinski [31] has given a definition for *perfect model*. Essentially, $\mathcal{M}$ is a perfect model (for a given ranking of atoms) if for all other models $\mathcal{M}'$, if positive literal p is the atom of least rank that is in one model, but not the other, then it is in $\mathcal{M}'$. In other words the perfect model minimizes positive literals of low rank in preference to positive literals of greater rank.

Przymusinski has shown that all locally stratified programs have a perfect model, and that it is independent of the ranking system chosen (within the constraints mentioned); moreover, on stratified programs, the perfect model agrees with the stratified model. We show that the well-founded semantics is an extension of this approach in the following sense.

Theorem 6.1. If $\mathbf{P}$ is locally stratified, then it has a well-founded model, which is identical to the perfect model.

Proof. We take as the inductive hypothesis that for any atom p of rank k: if p is in the perfect model, it is in the well-founded partial model I^∞; and if p is not in the perfect model, then $\neg p$ is in I^∞.

The basis, $k = 0$, is immediate.

For $k > 0$, first assume p is in the perfect model. Then we claim that there is an instantiated rule with p as head, say

$$p \leftarrow q_1,\, q_2,\, \ldots,\, \mathbf{not}\ r_1,\, \mathbf{not}\ r_2,\, \ldots$$

such that all q_i are in the perfect model and no r_j is in the perfect model. For if this were not so, we could remove p from the (supposedly) perfect model, and at worst have to add atoms of greater rank than p (because they have a rule containing **not** p) to restore the model. Since the r_j are of lower rank than p, the inductive hypothesis asserts that $\neg r_j$ are in I^∞. Also, any q_i of lower rank than p are in I^∞.

Now consider a program consisting of *all* instantiated rules for atoms of rank k whose subgoals of lower rank are true in I^∞. We modify the rules by removing the subgoals of rank less than k, leaving a Horn program $\mathbf{P}''$ (*cf.* Definition 5.3). Clearly the minimum model of $\mathbf{P}''$ will be precisely the atoms of rank k in the perfect model. But all such atoms are also in I^∞. Moreover, the atoms of rank k not in the minimum model of $\mathbf{P}''$ form an unfounded set of $\mathbf{P}''$ with respect to $\emptyset$ by Theorem 3.7. It follows from the construction of $\mathbf{P}''$ that these atoms also form an unfounded set of $\mathbf{P}$ with respect to I^∞, so their negations are in I^∞. $\square$

7. Motivating Examples

Whether a particular model is the "right" one really depends on people's expectations. After all, programs are tools whose behavior needs to be understood and manageable by people. In this section we compare well-founded semantics with some other recent approaches based on canonical models, the stable model semantics outlined in Section 5, and stratified semantics, which has been studied by many researchers. We present some examples to support our position that well-founded models are natural and intuitive.

Example 7.1. This example is abstracted from the "Yale shootout" example due to Hanks and McDermott [13]. The program $\mathbf{P}$ is

$$\begin{aligned}
&\mathit{noise}(T) \leftarrow \mathit{loaded}(T),\, \mathit{shoots}(T).\\
&\mathit{loaded}(0).\\
&\mathit{loaded}(T) \leftarrow \mathit{succ}(S,T),\, \mathit{loaded}(S),\, \mathbf{not}\ \mathit{shoots}(S).\\
&\mathit{shoots}(T) \leftarrow \mathit{triggers}(T).\\
&\mathit{triggers}(1).\\
&\mathit{succ}(0,1).
\end{aligned}$$

We regard *triggers* and *succ* as EDB predicates, and the others as IDB. The Herbrand instantiation of **P** contains ground versions of the IDB rules as follows:

$$
\begin{aligned}
noise(1) &\leftarrow loaded(1),\ shoots(1).\\
noise(0) &\leftarrow loaded(0),\ shoots(0).\\[4pt]
loaded(1) &\leftarrow succ(0,1),\ loaded(0),\ \mathbf{not}\ shoots(0).\\
loaded(1) &\leftarrow succ(1,1),\ loaded(1),\ \mathbf{not}\ shoots(1).\\
loaded(0) &\leftarrow succ(0,0),\ loaded(0),\ \mathbf{not}\ shoots(0).\\
loaded(0) &\leftarrow succ(1,0),\ loaded(1),\ \mathbf{not}\ shoots(1).\\[4pt]
shoots(1) &\leftarrow triggers(1).\\
shoots(0) &\leftarrow triggers(0).
\end{aligned}
$$

Intuitively, since we have no information that $shoots(0)$ holds, we are led to the (presumably) intended minimal model:

$$
\begin{array}{lll}
loaded(0), & \neg shoots(0), & \neg noise(0),\\
loaded(1), & shoots(1), & noise(1)
\end{array}
$$

However, an alternate minimal model exists:

$$
\begin{array}{c}
loaded(0), shoots(0),\ noise(0),\\
\neg loaded(1), shoots(1), \neg noise(1)
\end{array}
$$

Since $noise(1)$ is not true in all minimal models, the circumscription approach does not allow it to be concluded, which was a main point made in [13]. However, the well-founded model is the intended one.

To compare with other approaches: The 2PC model and Fitting model are also the intended model here. The program is stratified, so the stratified semantics agrees with the well-founded semantics. The intended model is also the unique stable model, as the alternate is not stable.

□

In the preceding example, the 2PC and Fitting models were 2-valued, and gave the intended model. The next example typifies the situation in which we consider the 2PC and Fitting models to be too weak an approach.

Example 7.2. Consider a program with the rules:

$$
\begin{aligned}
p(X,Y) &\leftarrow b(X,Y).\\
p(X,Y) &\leftarrow b(X,U),\ p(U,Y).\\[4pt]
e(X,Y) &\leftarrow g(X,Y).\\
e(X,Y) &\leftarrow g(X,U),\ e(U,Y).\\[4pt]
a(X,Y) &\leftarrow e(X,Y),\ \mathbf{not}\ p(X,Y).
\end{aligned}
$$

and the facts about b and g:

$$
\begin{array}{ll}
b(1,2) & g(2,3)\\
b(2,1) & g(3,2)
\end{array}
$$

Apparently, p is the transitive closure of b and e is the transitive closure of g. We expect a to be the difference of these two relations; in particular, it seems that $a(2,3)$ is true. This appears to be the intended model, and is indeed the well-founded model, as well as the stratified model.

There is another minimal model, in which $p(2,3)$ and $p(1,3)$ are true and $a(2,3)$ is false. Moreover, this alternate model satisfies the Clark completion of the program as well. Thus by the method of logical consequences of the completion of the program, the status of $a(2,3)$ and other literals is either not addressed (2PC interpretation) or declared undefined (Fitting model, Kunen model).

The criterion of stability reinforces the choice of the well-founded model. The alternate model is incapable of reproducing itself in the manner defined in Definition 5.3, and the intended model emerges as the unique stable model.

$\square$

In fact, Kunen has recently shown that in his 3-valued logical consequence semantics, a "strict" logic program without function symbols cannot define a predicate that is true in the transitive closure, false in its complement, and nowhere undefined [18]. Informally, a "strict" program is one in which the dependence of one predicate on another (or itself) is either through an even number of negations or through an odd number, but not both. Because Kunen's semantics is different from Fitting's, even on programs without function symbols (see Example A.1 in the appendix), the question of whether a "strict" program is possible in the Fitting semantics is open. Nonstrict programs in the Fitting semantics are known to exist, by the work of Immermann [14], but are quite complicated; details are discussed elsewhere by Van Gelder [42].

As another motivational example, we consider a program that is not locally stratified, as defined in Section 6, yet has a well-founded model when the EDB relation is acyclic. A more involved example in which constraints on the EDB can be specified to guarantee that the well-founded model is total is discussed elsewhere [41].

Example 7.3. This example is essentially the same as one discussed by Gelfond and Lifschitz [11], and is one of the examples that led to the formulation of well-founded semantics, as well as stable models. Interestly, this program turns out to be closely related to a game described by Kolaitis, and used to prove that there are queries in fixpoint logic that *are not expressible by stratified programs* [16]. In this respect, the program can be viewed as describing a game where one wins if the opponent has no moves, as in checkers (draughts).

$$\begin{aligned} \mathit{winning}(X) \leftarrow\, & \mathit{move}(X,Y), \\ & \textbf{not } \mathit{winning}(Y). \end{aligned}$$

Some sample *move* graphs are shown in Fig. 1. Whenever the *move* EDB relation is acyclic (e.g., part (a) of the figure), the well-founded total model is easily found, by proceeding "up" the directed graph. Part (b) shows a cyclic case in which the well-founded model is partial, but even when a cycle is present in the EDB, there may be a well-founded total model (part (c)). For this program, the Fitting model and the 2PC interpretation agree with the well-founded model.

However, the program is not locally stratified because the Herbrand instantiation contains a rule in which *winning* depends negatively upon itself, as in

$$\begin{aligned} \mathit{winning}(a) \leftarrow\, & \mathit{move}(a,a), \\ & \textbf{not } \mathit{winning}(a). \end{aligned}$$

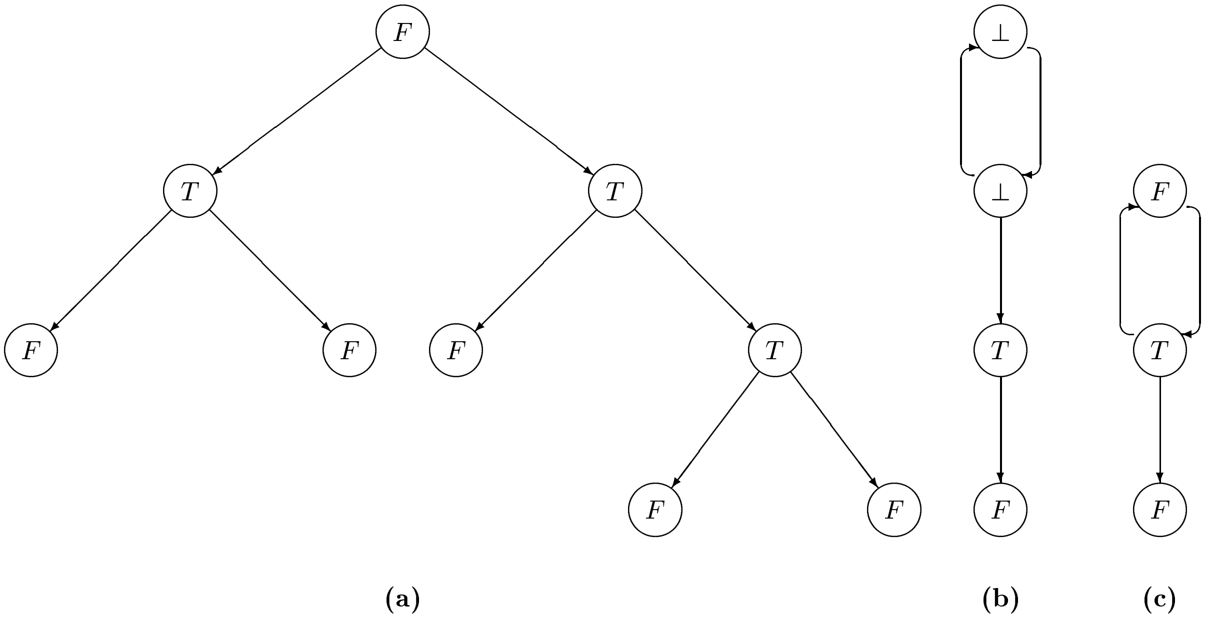

Figure 1: Graphs for Example 7.3: (a) Acyclic; (b) Cyclic with partial model; (c) Cyclic with total model. Entries T, F, and $\perp$ in the nodes indicate whether *winning* is true, false, or undefined in the well-founded (partial) model.

This also destroys the perfect model even though $move(a, a)$ does not occur in the EDB.[4] Recently, Przymusinska and Przmusinski have defined *weakly perfect* models to handle programs such as this example [30].
□

The next example was inspired by an informal presentation by K. Morris of Stanford University [27]. It shows how the negation issues addressed by this paper might easily arise in practical settings.

Example 7.4. We imagine a logic program that might be part of a VLSI CAD system, whose function is to display a VLSI chip that has been hierarchically defined. Each object is modeled as a series of layers, each layer being an array of grid points. The hierarchical definition specifies *basic* and *synthesized* objects: basic objects are distinguished by having *base colors*, while the colors of synthesized objects are defined wholly in terms of their components, and can vary from point to point. The entire chip is the "root" object. Figure 2 shows an example in which root object a is synthesized from objects b, c, d and e, whose further details are not shown.

Assume the program uses these predicates, which may be treated as EDB relations for our purposes.

- $vecSum(P0, P1, Pt)$ is true when $P0 + P1 = Pt$ as two-dimensional vectors, the details of whose representation do not concern us.

[4]Except in the trivial case where the program has only one Herbrand model.

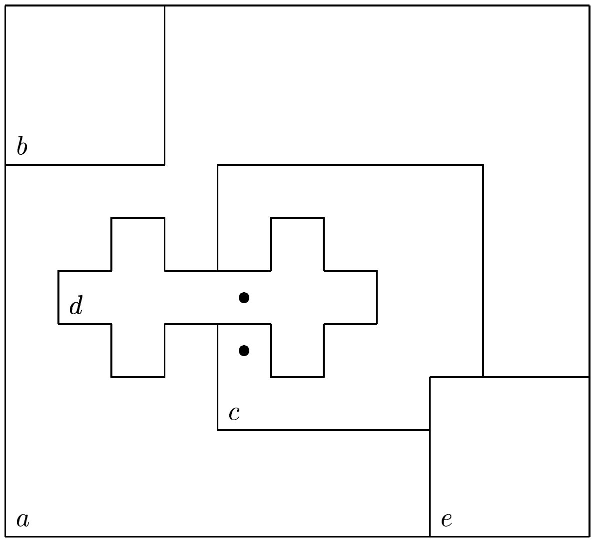

Figure 2: VLSI objects for Example 7.4: Object a is synthesized from b and c at level 1, and d and e at level 2. The color of object a is "inherited" from c at the point represented by the lower dot; however this does not hold at the upper dot, because c is dominated by d there.

- $component(Obj, O1, P0, L)$ is true when object Obj has a component $O1$ whose origin, or reference point, is $P0$, and whose layer number is L. For example, the chip might have many identical ALUs at different points; they would all be the same $O1$, but would have various values of $P0$. The ALUs might have adders as components, and the adders would have still smaller components. Within the same Obj components might overlap, so the layer number specifies their relative "vertical" order.
- $baseColor(Obj, Pt, C)$ means that C is the color of basic object Obj at point Pt.

To specify the *color* property in our rule syntax, we require two mutually recursive IDB relations. The interested reader can work out the equivalent rules using a single relation in a language that supports a richer syntax for rule bodies [21, 28].

- $color(Obj, Pt, C)$ means that the *visible* color of Obj at Pt is C (looking down from above).
- $dominated(Obj, Pt, O1, L1)$ holds when two objects that are components of the same Obj overlap at point Pt and the object $O1$ is in the lower layer $L1$. For the object in the higher layer to actually overlap, it must have *color* defined at that point.

We now formulate rules for determining the color C of a component Obj at a grid point Pt.

$$
\begin{aligned}
color(Obj, Pt, C) &\leftarrow baseColor(Obj, Pt, C). \\
color(Obj, Pt, C) &\leftarrow component(Obj, O1, P0, L1), \\
&\qquad vecSum(P0, P1, Pt), \\
&\qquad color(O1, P1, C), \\
&\qquad \mathbf{not}\ dominated(Obj, Pt, O1, L1). \\
dominated(Obj, Pt, O1, L1) &\leftarrow \\
&\qquad component(Obj, O2, P0, L2), \\
&\qquad L1 \leq L2, \\
&\qquad vecSum(P0, P2, Pt), \\
&\qquad color(O2, P2, C2).
\end{aligned}
$$

Note that *color* depends on itself negatively through the rule for *dominated*, as well as positively. The rule designer expects the *component* relation to be acyclic in its first and second arguments; i.e., $O1$ is expected to be a subcomponent of Obj.

When the expected acyclicity holds, the well-founded model is easily found, just working up the data structure. In this case, the Fitting model is 2-valued, as is the 2PC model. However, there is no perfect model for essentially the same reason as in Example 7.3.

When a cycle *is* present in the EDB, *color* cannot be established for anything in the cycle. For this application, the cycle presumably represents a design error. However, the well-founded semantics still defines *color* correctly in parts of the chip not affected by the error.

□

A theme that runs through these examples is that well-founded semantics frequently agrees with other semantics, but seems to avoid their awkward cases. In this sense it seems quite robust.

8. Computational Complexity

Not only do we want to formulate a reasonable semantics for negation, we also want the set of statements derivable to be "reasonably computable," as far as possible. Unfortunately, the well-founded partial model is not necessarily recursively enumerable, a difficulty it shares with most of the semantics discussed here. However, for function-free logic programs (a class that has come to be known as Datalog), the Herbrand universe is finite and the construction is effective. In this section we show that the *data complexity* of the well-founded semantics, as defined by Vardi [44], is polynomial. From this standpoint it is competitive with other methods, such as stratified semantics, whose data complexity has been studied elsewhere [4, 44, 12, 14], and the Fitting model (as remarked below).

In this discussion of complexity we restrict attention to function-free programs, so a program's Herbrand universe is just the set of constants appearing in it. We consider a fixed IDB, $\mathbf{P}_I$ (which we allow to be any general function-free logic program). As discussed before, $\mathbf{P}_I$ can be thought of as a set of inference rules that might be applied to various EDB's, or sets of facts. The predicates that appear as subgoals in $\mathbf{P}_I$, but do not appear in the *head* of any rule, constitute the EDB predicates. We represent an EDB, $\mathbf{P}_E$, as a set of positive ground literals ranging over the EDB predicates. (The constants in $\mathbf{P}_E$ may or may not appear in $\mathbf{P}_I$.) Given an EDB $\mathbf{P}_E$, we form a logic program $\mathbf{P}(\mathbf{P}_E) = \mathbf{P}_I \cup \mathbf{P}_E$, and we denote its well-founded partial model by $I^\infty(\mathbf{P}_E)$. Finally, regard $\mathbf{P}_I$ as defining the transformation from $\mathbf{P}_E$ to $I^\infty(\mathbf{P}_E)$.

Definition 8.1. The *data complexity* of an IDB is defined as the computational complexity of deciding the answer to a ground atomic query as a function of the size of the EDB; in the context of well-founded semantics, this means deciding whether the ground atom is positive in the well-founded partial model. $\square$

Since the IDB is fixed, the predicates in the well-founded model have fixed number and arity (width, or number of argument places). Hence the Herbrand base has size that is polynomial in the size of the EDB. (Without function symbols, we may add any constants appearing only in the query to the Herbrand universe without having a significant effect on its size.) Also since the IDB is fixed, the size of the Herbrand instantiation of the program is polynomial in the size of the EDB.

Theorem 8.1. The data complexity of the well-founded semantics for function-free programs is polynomial time.

Proof. As usual in the proofs of such theorems, we shall show that the entire well-founded (partial) model can be constructed in polynomial time, after which any query can be answered immediately. The well-founded model is the least fixed point of the construction of I_α, as described in Definition 3.4. At each stage of the induction, until the fixed point is reached, at least one element of the Herbrand base H is added to $I_{\alpha+1}$, so the fixed point must be reached in a number of steps polynomial in the size of the EDB. This sort of argument is standard; see [4, 44, 12, 14], etc. Similar standard arguments show that calculating $\mathbf{T}_P$ can be done in polynomial time. So we need only show that each $\mathbf{U}_P(I_\alpha)$ can be found in polynomial time. Clearly, we may restrict attention to finite α. We shall actually give a polynomial time construction of the set of ground atoms in $(H - \mathbf{U}_P(I_\alpha))$.

Define $\phi(J)$ as the transformation on sets of ground atoms, with implicit parameter I_α, such that: A ground atom p is in $\phi(J)$ if and only if there is a ground instance of a rule in $\mathbf{P}$, say

$$p \leftarrow b_1,\, \ldots,\, b_n,\, \mathbf{not}\ c_1,\, \ldots,\, \mathbf{not}\ c_m$$

such that

- no subgoal (b_i or $\mathbf{not}\ c_j$) is false in I_α, and
- all b_i are in J.

Let $J_\gamma = \phi^\gamma(\emptyset)$. Clearly ϕ is monotonic, and J_γ reaches a limit J^∞ at a γ that is polynomial in $|H|$.

Suppose $p \in \phi(J_\gamma)$ due to the rule shown above. This rule shows that if p were in $\mathbf{U}_P(I_\alpha)$, then some b_i must also be in that set. Thus by a trivial induction on γ, no atom in $\bigcup_\gamma J_\gamma$ is in $\mathbf{U}_P(I_\alpha)$.

To show that the set of ground atoms in $H - J^\infty$ is unfounded w.r.t. I_α, let q be any such atom. Then each rule with q as head has a subgoal that violates the condition that would put q in $\phi(J_\gamma)$, for any γ. If the violation is that some subgoal is false in I_α, this satisfies condition (1) for an unfounded set (Definition 3.1); if the violation is that some positive subgoal is not in J_γ for any γ, then that subgoal is in $H - J^\infty$, satisfying condition (2) for an unfounded set.

It follows that $J^\infty = (H - \mathbf{U}_P(I_\alpha))$. $\square$

The key idea in the above proof, to inductively construct the *complement* of the greatest unfounded set, was first suggested to two of the authors by M. Y. Vardi, and later discovered independently by J. S. Schlipf.

We remark that the Fitting model also has polynomial data complexity (for function-free programs). The proof is identical to that of Theorem 8.1 above, except that a polynomial calculation of $\mathbf{N}_P$ (see Def. 4.2) must be exhibited; but such a calculation is routine.

In contrast, Marek and Truszczynski [24] have shown that, even for *propositional* general logic programs $\mathbf{P}$, determining whether $\mathbf{P}$ has a stable model at all is NP-complete.

9. The Final Frontier?

The major shortcoming of the well-founded semantics that we have found concerns its inability to handle conclusions that can be reached only by using factoring or a similar technique, such as "ancestor resolution." Such techniques are known to be necessary for completeness of non-Horn proof systems, but not for sets of Horn clauses. The need for factoring arises principally from "proof by cases", and sometimes from "proof by contradiction".

However, factoring possibilities in the given program do not always carry over to the completed program, and $a \vee \neg a$ does not simplify to **true** in either 3-valued logic [9, 17] or intuitionistic logic [7]. Thus caution is needed to keep a coherent system.

The overly trivial $\mathbf{P}_2$ in Example 5.3 might lead one to believe that a factoring capability can be easily "patched in" by just checking for a negative subgoal that complements the head of the rule; this conclusion would be incorrect, as shown by

$$\begin{aligned}
&a \leftarrow \mathbf{not}\ b.\\
&b \leftarrow \mathbf{not}\ a.\\
&p \leftarrow a.\\
&p \leftarrow b.
\end{aligned}$$

in which we cannot choose between a and b, but might reasonably be expected to notice that p must hold (in 2-valued logic). In general, recognizing that p is a (2-valued) logical consequence of a *finite* set of instantiated rules is co-NP-complete. Furthermore, we normally start with rules that contain variables. Thus, any extension of logic program semantics that depends on "true non-Horn reasoning" needs to be undertaken with great caution, and represents a significant open problem.

10. Conclusion

We have presented a new semantics, the *well-founded semantics*, for general logic programs that extends several earlier proposals, and has advantages over them in that

1. It is applicable to all programs.

2. Compared to several other methods, a larger portion of the Herbrand base tends to be classified as either true or false.

3. Truth values are assigned (in the authors' judgement) in a reasonably predictable and intuitively satisfying way.

Elsewhere, the expressive power of the well-founded semantics has been compared to several forms of fixpoint logic [42]. A corresponding procedural semantics has been reported for some classes of programs [34, 32].

Acknowledgements

We wish to thank Jerzy Jaromczyk, Phokion Kolaitis, Vladimir Lifschitz, Wiktor Marek, Rodney Topor, and Moshe Vardi for helpful discussions and comments about this work. We also thank the anonymous referees for their careful readings of the manuscript and many useful suggestions.

The work of Kenneth Ross was supported in part by NSF grant IRI-8722886, by a grant from IBM Corporation, and by AFOSR under contract 880266. The work of John Schlipf was supported in part by NSF grants IRI-8705184 and IRI-8901566. The work of Allen Van Gelder was supported in part by NSF grants CCR-8958590 and IRI-8902287.

Appendix A Augmented Programs

Certain programs exhibit undesirable behavior when interpreted in the Herbrand universe, due to their containing what is called *unsafe negation.* A simple way to remove this behavior is to augment the program, as described in this appendix. We proceed informally here, and refer to [22] for a formal discussion.

Definition A.1. Any general logic program **P** has an associated *augmented program* that is formed by adding the apparently nonsensical rule:

$$\$p(\$f(\$c)) \leftarrow \$p(\$f(\$c)).$$

where $\$p$, $\$f$, and $\$c$ are symbols that do not occur elsewhere in the program.
□

Having the extra "\$" terms in the augmented Herbrand universe adds infinitely many elements to the Herbrand universe, elements that have no names in the original program. This ensures that goals with free variables have "room to fail" when they should, even in their instantiated versions. Augmenting achieves an effect similar to Kunen's embedding the program in a language with infinitely many function and constant symbols.

Example A.1. In the following program, without inspecting the a relation, we would expect p to hold wherever a does. (Read d as "differs" and s as "same".)

$$\begin{aligned} p(X) &\leftarrow a(X),\, d(X,Y). \\ d(X,Y) &\leftarrow \textbf{not } s(X,Y). \\ s(U,U)&. \\ a(1)&. \end{aligned}$$

The underlying idea is that, looking at the rule for s, we expect the formula $\forall Y s(X,Y)$ to be false. But in the unaugmented Herbrand universe of one element there is no "room" for $s(1,Y)$ to fail because 1 is the only term. As a result, $p(1)$ fails. However, adding the apparently unrelated fact $b(2)$ to the program means that $s(1,Y)$ can fail, by setting $Y = 2$. This in turn provides a true instance $d(1,2)$, allowing a proof of $p(1)$. Augmenting the program avoids this bizarre behavior; $s(1,\$c)$ fails in all cases, making $p(1)$ always provable, as intuition expects.

To see why this program has unsafe negation, consider a top-down sequence of goal reductions beginning with $p(1)$. Using the rules, $p(1)$ reduces to $(a(1),\, d(1,Y))$, $a(1)$ reduces to true, then $d(1,Y)$ reduces to

not $s(1, Y)$. The occurrence of a free variable Y in the negative subgoal is called "unsafe" because it is not limited to any domain. This derivation is said to have floundered [20].

Finally, let us note that in the unaugmented program $p(1)$ is false in the well-founded semantics and in the Fitting semantics, but not in the 2PC semantics or Kunen semantics. Although $p(1)$ is false in the only 2-valued *Herbrand* model of the completed program, there are other 2-valued models in which $p(1)$ is true. All of these semantics agree that $p(1)$ is true in the augmented program.
$\square$

As noted, the Herbrand universe for the augmented program is infinite. As a result, our proof of polynomial data complexity (Theorem 8.1) fails for the augmented program. But the result is still true for augmented programs; we need only modify the proof slightly. The extra ground terms are all indiscernible with respect to the predicates of the original language, so we can carry out the same construction using only a fixed, finite number (dependent upon the IDB) of the extra ground terms. Essentially, we need as many distinct \$-terms as there are variables in a single rule.

References

[1] K. R. Apt, H. Blair, and A. Walker. Towards a theory of declarative knowledge. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 89–148. Morgan Kaufmann, Los Altos, CA, 1988.

[2] K. R. Apt and M. H. Van Emden. Contributions to the theory of logic programming. *JACM*, 29(3):841–862, 1982.

[3] F. Bry. Logic programming as constructiveism: a formalization and its application to databases. In *Eighth ACM Symposium on Principles of Database Systems*, pages 34–50, 1989.

[4] Ashok Chandra and David Harel. Structure and complexity of relational queries. *JCSS*, 25(1):99–128, 1982.

[5] A. Chandra and D. Harel. Horn clause queries and generalizations. *Journal of Logic Programming*, 2(1):1–15, 1985.

[6] K. L. Clark. Negation as failure. In Gallaire and Minker, editors, *Logic and Databases*, pages 293–322. Plenum Press, New York, 1978.

[7] M. A. E. Dummett. *Elements of Intuitionism.* Clarendon Press, Oxford, 1977.

[8] Ph. M. Dung and K. Kanchanasut. A natural semantics for logic programs with negation. Technical report, Asian Institute of Technology, Bankok 10501, Thailand, 1989. (manuscript).

[9] M. Fitting. A Kripke-Kleene semantics for logic programs. *Journal of Logic Programming*, 2(4):295–312, 1985.

[10] M. Gelfond. On stratified autoepistemic theories. In *Proc. AAAI*, 1987.

[11] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Fifth Int'l Conf. Symp. on Logic Programming*, pages 1070–1080, Seattle, 1988.

[12] Y. Gurevich and S. Shelah. Fixed-point extensions of first order logic. *Annals of Pure and Applied Logic*, 32:265–280, 1986.

[13] S. Hanks and D. McDermott. Default reasoning, nonmonotonic logics, and the frame problem. In *AAAI Conference*, pages 328–333, 1986.

[14] N. Immerman. Relational queries computable in polynomial time. *Information and Control*, 68(1):86–104, 1986.

[15] J. Jaffar, J.-L. Lassez, and J. Lloyd. Completeness of the negation-as-failure rule. In *Int'l Joint Conf. on Artificial Intelligence*, pages 500–506, 1983.

[16] P. G. Kolaitis. The expressive power of stratified programs. *Information and Computation*, 90(1), 1991.

[17] K. Kunen. Negation in logic programming. *Journal of Logic Programming*, 4(4):289–308, 1987.

[18] K. Kunen. Some remarks on the completed database. Technical Report 775, Univ. of Wisconsin, Madison, WI 53706, 1988. (Abstract appeared in 5th Int'l Conf. Symp. on Logic Programming, Seattle, Aug. 1988).

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

[20] J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, New York, 2nd edition, 1987.

[21] J. W. Lloyd and R. W. Topor. Making Prolog more expressive. *Journal of Logic Programming*, 1(3):225–240, 1984.

[22] M. J. Maher. Equivalences of logic programs. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 388–402. Morgan Kaufmann, Los Altos, CA, 1988.

[23] W. Marek. Stable theories in autoepistemic logic. Technical report, University of Kentucky, 1986. (manuscript).

[24] A. Marek and M. Truszczynski. Autoepistemic logic. Technical report, University of Kentucky, 1988. (manuscript).

[25] J. Minker. On indefinite databases and the closed world assumption. In *Sixth Conference on Automated Deduction*, pages 292–308, New York, 1982. Springer-Verlag.

[26] R. C. Moore. Semantical considerations on non-monotonic logic. *Artificial Intelligence*, 28:75–94, 1985.

[27] K. Morris. Talk at Workshop XP8.3i, Oregon Graduate Center, July 1987.

[28] K. Morris, J. D. Ullman, and A. Van Gelder. Design overview of the Nail! system. In *Third Int'l Conf. on Logic Programming*, pages 554–568, 1986.

[29] Y. N. Moschovakis. *Elementary Induction on Abstract Structures*. North-Holland, New York, 1974.

[30] H. Przymusinska and T. C. Przymusinski. Weakly perfect model semantics for logic programs. In *Fifth Int'l Conf. Symp. on Logic Programming*, pages 1106–1120, Seattle, 1988.

[31] T. C. Przymusinski. On the declarative semantics of deductive databases and logic programs. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 193–216. Morgan Kaufmann, Los Altos, CA, 1988.

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

[33] R. Reiter. On closed world databases. In Gallaire and Minker, editors, *Logic and Databases*, pages 55–76. Plenum Press, New York, 1978.

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

[35] K. Ross and R. W. Topor. Inferring negative information from disjunctive databases. *Journal of Automated Reasoning*, 4:397–424, 1988.

[36] J. S. Schlipf. Negation by securable failure in logic programming. (manuscript), 1987.

[37] J. C. Shepherdson. Negation as failure, II. *Journal of Logic Programming*, 2(3):185–202, 1985.

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

[39] M. H. Van Emden and R. A. Kowalski. The semantics of predicate logic as a programming language. *JACM*, 23(4):733–742, 1976.

[40] A. Van Gelder. Negation as failure using tight derivations for general logic programs. *Journal of Logic Programming*, 6(1):109–133, 1989. Preliminary versions appeared in *Third IEEE Symp. on Logic Programming* (1986), and *Foundations of Deductive Databases and Logic Programming*, J. Minker, ed., Morgan Kaufmann, 1988.

[41] A. Van Gelder. Modeling simultaneous events with default reasoning and tight derivations. *Journal of Logic Programming*, 8(1):41–52, 1990.

[42] A. Van Gelder. The alternating fixpoint of logic programs with negation. *Journal of Computer and System Sciences*, 1992. (to appear). Available as UCSC-CRL-89-39. Preliminary abstract appeared in Eighth ACM Symposium on Principles of Database Systems, 1989.

[43] A. Van Gelder, K. A. Ross, and J. S. Schlipf. Unfounded sets and well-founded semantics for general logic programs. In *ACM Symposium on Principles of Database Systems*, pages 221–230, 1988.

[44] Moshe Vardi. The complexity of relational query languages. In *14th ACM Symposium on Theory of Computing*, pages 137–145, 1982.