JOURNAL OF COMPUTER AND SYSTEM SCIENCES 51, 64–86 (1995)

The Expressive Powers of the Logic Programming Semantics*

JOHN S. SCHLIPF†

Center for Intelligent Systems, Department of Computer Science, University of Cincinnati, Cincinnati, Ohio 45221-0008

Received March 23, 1993

We study the expressive of two semantics for deductive databases and logic programming: the well-founded semantics and the stable semantics. We compare them especially to two older semantics, the two-valued and three-valued program completion semantics. We identify the expressive power of the stable semantics and, in fairly general circumstances, that of the well-founded semantics. In particular, over infinite Herbrand universes, the four semantics all have the same expressive power. We discuss a feature of certain logic programming semantics, which we call the Principle of Stratification, a feature allowing a program to be built easily in modules. The three-valued program completion and well-founded semantics satisfy this principle. Over infinite Herbrand models, we consider a notion of translatability between the three-valued program completion and well-founded semantics which is in a sense uniform in the strata. In this sense of uniform translatability we show the well-founded semantics to be more expressive than the three-valued program completion. The proof is a corollary of our result that over non-Herbrand infinite models, the well-founded semantics is more expressive than the three-valued program completion semantics.

1. INTRODUCTION

Deductive databases and logic programming draw inferences. The goal is to simulate, or idealize the way humans draw inferences. One starts with, first, a database of raw information, and second, a set of rules for inferring more information from information in the database. For example, a database might contain *human*(*Socrates*), and there might be a rule $mortal(X) \leftarrow human(X)$—if X is human, then X is mortal—forcing the inference *mortal*(*Socrates*).

Since logic programming attempts to idealize human inference, many inferences are not so obvious. Particularly problematical are inferences related to *negation as failure*: if some (positive) atomic fact is clearly (in some sense that must be specified) not derivable with any of the inference rules, then infer it to be false. For example, if the database has no other rules and does not contain *mortal*(*Thor*) or *human*(*Thor*), infer $\neg mortal(Thor)$. Negation as failure is obviously not sound in classical logic; nevertheless, it seems a normal pattern of human inference in many circumstances. It is the analog of an assumption with traditional databases: if the database does not contain the information that Socrates is the manager of the sales department, then infer that he is not.

There has been a continuing search for the "correct" semantics for logic programming with negation as failure, primarily a search for the semantics with the most elegant and intuitively natural rules for deriving negated atomic facts. Although all these semantics agree in inferring $\neg mortal(Thor)$ above, they disagree on many other logic programs. We investigate two semantics here: the well-founded semantics of [VGRS91] and (a variant of) the stable semantics of [Gel87, GL88]. We compare them with two older semantics: the two-valued and three-valued program completion semantics of [Cla78, Fit85].

Although logic programming semantics are usually thought of as models of human inference, they can also be thought of as methods for defining relations on databases. In the example above, the goal may be thought of as defining (the extension of) the relation *mortal*. The usual question about a logic programming semantics is whether it is intuitively correct. Our primary concern is different: How expressive is it; what relations can it define? The main result of this paper is to determine the expressive power of the stable semantics and, in fairly general circumstances, that of the well-founded semantics.

Expressive power is a standard concern in database query languages. It has also been investigated in several recent papers in relation to normal logic programming, for example, in [AB90] (on the degrees of uncomputability of the perfect models of stratified programs), [KP88] (including a result on the computational complexity of Clark's semantics over finite databases), [Kun88] (some results on the program completion semantics over finite extensional databases), and [MNR92] (an extensive recursion-theoretic discussion of the class of stable Herbrand models of programs with function symbols, which overlaps at one significant point with this paper).

* A preliminary version of this paper was presented at the "Ninth ACM Symposium on Priciples of Database Systems, 1990."

† Work supported by NSF Grants IRI-8705184 and IRI-8905166.

0022-0000/95 $12.00

Of course, increased expressive power is both a blessing and a curse. It is a blessing in that it lets programmers express more complex relations. It is a curse in that increased expressive power is generally equivalent to greater difficulty of computation. This is the case here.

Another way of looking at the comparison of expressive powers is as a question of translatability. Suppose everything that can be expressed in Semantics 1 can also be expressed in Semantics 2. And suppose program **P** is used to define relation r under Semantics 1. Then thee is a (probably different) "equivalent" program **Q** which defines the same relation r under Semantics 2. We investigate some circumstances when such translations are especially natural.

We also discuss a second property of some logic programming semantics, which we call the Principle of Stratification. The Principle of Stratification does not address expressive power *per se*. Rather, it is an assertion that the semantics is well behaved, relative to one of the common intuitions for logic programming semantics. (Thus this is our one discusson in this paper of the intuitive correctness of a semantics, but it differs from other more common example-driven intuitive correctness discussions.) According to this intuition, logic programs are implicit definitions. (This intuition is at the heart of, for example, the perfect model semantics for stratified logic programs.) Extending that, we suggest that certain easily identifiable pieces of logic programs should be treatable as "modules," or "strata," defining certain relations. The Principle of Stratification asserts that, if a logic program is built by assembling such "modules," then these "modules" actually operate relatively independently, allowing separate "modules" to be written separately, with no worry about side effects.

We use the Principle of Stratification to make a finer comparison of expressive powers. Recall that, if two semantics have the same expressive power, it is possible to translate a program in one into an "equivalent" program in the other. We suggest that one reasonable requirement for naturalness of a translation is that, since each stratum can be thought of as a module, it should be possible to translate the strata of a program separately. In particular, we show that although the three-valued program completion semantics and the well-founded semantics have the same expressive power over infinite Herbrand universes, programs can be translated stratum by stratum in only one direction.

2. FORMAL PRELIMINARIES

2.1. *Logic Programming Formalism*

Consider a logic program, such as the program with the two rules

$$ancestor(X, Y) \leftarrow parent(X, Y).$$

$$ancestor(X, Y) \leftarrow parent(X, Z) \wedge ancestor(Z, Y).$$

These two rules can be thought of as defining the relation *ancestor* from the relation *parent*. The relation *parent*, on the other hand, does not appear in the head of any rule in the program, so it is generally not thought of as being defined by the program. One interpretation is that the set of tuples in the relation *parent* will be supplied separately. It is supplied separately in what is called an *extensional database* (EDB).

DEFINITION 2.1. A (normal) logic program is a finite set of formulas, called *rules*, of the form

$$\alpha \leftarrow \beta_1 \wedge \cdots \wedge \beta_n$$

where α is a positive literal (i.e., atomic formula), and each β_i is a positive literal or a negative literal (i.e., a negated atomic formula). (The rule above may be read as "infer α if $\beta_1, \ldots, \beta_n$ are all known to be true," but $\leftarrow$ in most semantics is not the (material) implication of classical logic.) Here α is the *head* of the rule, $\beta_1 \wedge \cdots \wedge \beta_n$ is the *body*, and the β_i's are *subgoals*. A rule with no subgoals is just an atomic formula α, that is, without the $\leftarrow$ symbol.

For α a positive literal, $\neg\alpha$ is called the *negation* of α, and α is called the *negation* of $\neg\alpha$.

Logic programming is traditionally studied in two contexts:

- The most common setting is to treat a logic program as some sort of variant of first-order logic, but with a fixed universe understood. Normally, this is taken to be the *Herbrand universe* of the program, the set of ground (variable-free) terms of the language of the program. In fact, the *Herbrand preinterpretation* is assumed: the universe is the Herbrand universe, each constant symbol is interpreted by itself, and each function symbol is interpreted in the obvious way. The logic program is used to define (the interpretations of) the relation symbols. This domain assumption is generally made with logic programs that contain both function and constant symbols.

By any standard recursive encoding method (e.g., by Gödel numbering), the Herbrand universe and the set of ground atoms of a logic program can be considered to be recursive subsets of the integers. Then computability questions can be asked about the set of atoms inferred from the program.

- With logic programs that contain no function symbols, a somewhat different domain assumption is frequently made, oriented toward deductive databases. Since there are no function symbols, the Herbrand universe is the set of constant symbols of the program, and the program may be used to calculate the interpretations of the relation symbols just as above.

Now think of separating the rules of a logic program into two parts, a database of information (the rules with no

subgoals) and a set of rules (the rules with subgoals) for inferring further information from the database. The database part is called an extensional database (EDB); the inference rule part is called an intensional database (IDB). Presumably, the rules of the IDB correspond to natural definitions or inference rules that would make sense over various databases of information—over various EDBs. Hence a deductive database point of view is to study the inferences made by a single IDB over varying EDBs. One traditional study is the computational complexity of making inferences, as a function of the size of the EDB; this is called *data complexity*.

For the semantics studied in this paper, we can, with no loss of generality, assume that the relation symbols appearing in the EDB never appear in the heads of rules in the IDB. This makes a nice, clean break; the EDB is a database of information, defining certain relations. The EDB is taken as giving complete information. (Under common treatments, any instance of any relation not explicitly given to be true may be inferred to be false.) It turns out that, for the questions studied in this paper, it also causes no loss of generality to assume that no constant symbols appear in the IDB and that no variable symbols appear in the EDB, so we shall make those simplifying assumptions. In this context, the universe of each interpretation is just the set of objects (or constant symbols) appearing in the EDB.

For consistency of vocabulary in general contexts, we shall refer to the IDB as the *program*, and consider the EDB to be specified separately.

Since both these constexts are commonly studied, we address expressive power in both. We shall see that there are interesting contrasts between the contexts. But more general contexts have also been studied. Clark [Cla78] studied not just the Herbrand universe, but all preinterpretations satisfying a set of first-order axioms called the Clark equality theory. Kunen [Kun87] made an even broader domain assumption. Another variant is presented in [VGRS91]. By considering even broader collections of domains, in particular, infinite universes other than Herbrand universes, we derive more precise comparisons among the semantics. At the end of this paper, we shall relate definability on these exotic universes to definability over the traditional Herbrand universes.

The usual treatment for logic programs with function symbols, discussing programs and their Herbrand universes (or preinterpretations), could also be treated in an obvious way as a discussion of EDBs and IDBs. The Herbrand universe itself could be considered as the preferred EDB for the logic program. In this case there are no EDB relations, but instead the function symbols are all treated as being specified in the EDB. The entire logic program may be treated as an IDB for inferring relations over the fixed EDB.

DEFINITION 2.2. Let **P** be a logic program.

- An *intensional relation*, or *IDB relation*, of **P** is a relation which appears in the head of some rule of **P**.
- An *extensional relation*, or *EDB relation*, of **P** is a relation which appears in **P**, but never in the head of a rule of **P**.
- A literal $r(\mathbf{t}(\mathbf{X}))$ or $\neg r(\mathbf{t}(\mathbf{X}))$ is an *EDB literal* if r is an EDB relation; it is an *IDB literal* if r is an IDB relation.
- An *extensional database* (EDB) for **P** is a structure $\mathscr{D}$ (in the sense of first-order logic) for the language consisting of the function, constant, and EDB relation symbols of the program.

If there are no function or constant symbols, it is a (possibly infinite, relational) database for the set of EDB relations of **P**, plus a relation specifying the universe of the database.

In order to have uniform constructions, we shall assume that each EDB contains at least two elements. In any case, definability over one-element databases is not a major concern in this area.

Logic programming semantics are often defined in terms of what are called ground literals and ground instantiations of logic programs. These notions are specialized to Herbrand universes. In order to generalize the semantics to other universes, it is convenient to generalize the notion of ground literals and ground instantiations of programs to arbitrary EDBs. Our definitions of the various semantics will agree with the standard definitions when $\mathscr{D}$ is a finite EDB for program **P** or the Herbrand universe for program **P**; for other universes, our definitions of the well-founded and stable semantics are natural extensions of the original definitions.

DEFINITION 2.3. For an EDB $\mathscr{D}$, a *$\mathscr{D}$-instantiated literal* is an "expression" $r(\mathbf{d})$ or $\neg r(\mathbf{d})$, where $\mathbf{d}$ is a tuple of elements of $\mathscr{D}$. Form a *$\mathscr{D}$-instantiated rule* of program **P** by replacing all the variable symbols in a rule in **P** with elements of $\mathscr{D}$ and evaluating the terms in $\mathscr{D}$. The *$\mathscr{D}$-instantiation* of **P** is the set of all such $\mathscr{D}$-instatiated rules (for all rules in **P** and all possible ways to substitute elements of $\mathscr{D}$) plus the set of all positive EDB literals true in $\mathscr{D}$.

In defining the semantics we shall use the $\mathscr{D}$-instantiated rules, not the original rules of **P**. Thus, when we define what inferences a semantics makes from a program **P** over two different EDBs $\mathscr{D}_1$ and $\mathscr{D}_2$, we start by forming two different programs, the $\mathscr{D}_1$-instantiation of **P** and the $\mathscr{D}_2$-instantiation of **P**. Note that, although a logic program **P** is, by definition, finite, if $\mathscr{D}$ is an infinite EDB, then the $\mathscr{D}$-instantiation of **P** is, in general, infinite. In our definitions, we shall still refer to these as logic programs.

DEFINITION 2.4. A *partial interpretation* for a logic program **P** over an EDB $\mathscr{D}$ is a set I of $\mathscr{D}$-instantiated literals of **P**. For β a positive $\mathscr{D}$-instantiated literal:

- If $\beta \in I$ and $\neg\beta \notin I$, then β is *true in* I and $\neg\beta$ is *false in* I.
- If $\neg\beta \in I$ and $\beta \notin I$, then β is *false in* I and $\neg\beta$ is *true in* I.
- If β, $\neg\beta \notin I$, then β, $\neg\beta$ are *undetermined in* I.
- If β, $\neg\beta \in I$, then β, $\neg\beta$ are *overdetermined in* I.

DEFINITION 2.5. A partial interpretation I is *three-valued*, or *consistent*, if no literal is overdetermined in I; I is *two-valued* if no literal is either over- or un-determined in I.

Undetermined truth values are common in logic programming; they correspond to literals which there is no reason to infer to be either true or false. Overdetermined values will be inferred in this paper only when a semantics "considers" a program incoherent.

We treat a logic programming semantics as specifying, for each program **P** and EDB $\mathscr{D}$, a set of literals to be inferred.[1]

DEFINITION 2.6. A *logic programming semantics* is a function assigning, to each logic program **P** and EDB $\mathscr{D}$ for **P**, a partial interpretation for **P** over $\mathscr{D}$. We say a $\mathscr{D}$-instantiated literal α of **P** is *inferred from* **P** *over* $\mathscr{D}$ in the semantics if α is in the set resulting from applying the semantics to **P** and $\mathscr{D}$. A *partial semantics for logic programming* is defined analogously, but it is a partial function from programs and EDBs to partial interpretations.

DEFINITION 2.7. A relation s on an EDB $\mathscr{D}$ is *definable* in a logic programming semantics if there exist a program **P** and a relation r in **P**, where

$$s = \{\mathbf{d} \text{ in } \mathscr{D} : r(\mathbf{d}) \text{ is inferred from } \mathbf{P} \text{ over } \mathscr{D} \text{ by the semantics}\}.$$

In this case the program **P** is explicitly allowed to refer to (essentially, have names for) arbitrary elements of $\mathscr{D}$.

Suppose s parameterizes a set of relations on a set $\mathscr{E}$ of EDBs, i.e., for each EDB $\mathscr{D} \in \mathscr{E}$, $s_{\mathscr{D}}$ is a relation on $\mathscr{D}$. Then s is *definable* in semantics **S** if there is a program **P** and a relation r in **P**, where for each $\mathscr{D} \in \mathscr{E}$,

$$s_{\mathscr{D}} = \{\mathbf{d} \text{ in } \mathscr{D} : r(\mathbf{d}) \text{ is inferred from } \mathbf{P} \text{ over } \mathscr{D} \text{ by the semantics}\}.$$

[1] This definition of a semantics seems a natural generalization of usual treatments of the van Emden Kowalski semantics. We feel it is also fairly the natural way to treat the three-valued program completion and well-founded semantics. Some researchers feel it is more natural to define the semantics to include other formulas, for example, disjunctions $\alpha \vee \beta$ of literals **P**, or for the semantics instead to specify a set of "preferred" models. This distinction will be significant in Section 5, where we use it to contrast what we call the Principle of Stratification and the Weak Principle of Stratification.

In this case **P** is not allowed to refer to elements of $\mathscr{D}$ unless they are named by constant symbols of the language.[2] (This is just to force **P** to make sense in every $\mathscr{D} \in \mathscr{E}$.)

A logic programming semantics S_1 is *at least as expressive as* a logic programming semantics S_2 over an EDB, or a class of EDBs, if every relation definable in S_2 over the EDB, or each EDB in the class, is also definable in S_1. (Note that this does not mean that S_1 and S_2 draw the same inferences from any specific program.) Semantics S_1 is *at least as expressive as* semantics S_2 if it is at least as expressive as S_2 over all EDBs. Semantics S_1 is *uniformly at least as expressive as* S_2 over a class $\mathscr{E}$ of EDBs if every relation uniformly definable over all EDBs in the class in S_2 is also uniformly definable over all EDBs in the class in S_1.

Semantics S_1 is *more expressive than* S_2 (over an EDB, over a class of EDBs, or uniformly over a class of EDBs, respectively) if it is at least as expressive as S_2, but S_2 is not at least as expressive as S_1 (over the EDB, over the class of EDBs, or uniformly over the class of EDBs, respectively). Semantics S_1 and S_2 *have the same expressive power* (over an EDB, over a class of EDBs, or uniformly over a class of EDBs, respectively) if each is at least as expressive as the other (over the EDB, over the class of EDBs, or uniformly over the class of EDBs, respectively).

2.2. *Inductive Definability*

Our results relate definability in semantics for logic programming to (first order, positive) *inductive definability*, as discussed in [Mos74, Bar75, Acz77]. It is the same as *fixed-point logic*, but with *just one final application of the fixed point*, as discussed in [AU69, CH82, Imm86, GS86].

Inductive definability is a (large) generalization of traditional examples. The simplest is the notion of transitive closure: (X, Y) is in the transitive closure tc of relation r if

$$r(X, Y) \vee \exists Z(\mathrm{tc}(X, Z) \wedge \mathrm{tc}(Z, Y)).$$

Suppose $\mathscr{D}$ is a structure and Φ is a set of first-order formulas $\phi_i(r_1, \ldots, r_n, \mathbf{X})$, $i = 1, \ldots, n$ (built up using $\neg$, $\wedge$, $\vee$, $\exists$, and $\forall$), where no r_i is interpreted in $\mathscr{D}$, every other symbol of each ϕ_i is interpreted in $\mathscr{D}$, and no r_i appears in any ϕ_j inside the scope of a $\neg$ (i.e., each ϕ_i is *positive in* each r_j). Individual elements of $\mathscr{D}$ may be explicitly mentioned (essentially as constant symbols) in Φ.[3] Then consider the set of "rules"

$$r_i(\mathbf{X}) \leftarrow \phi_i(r_1, \ldots, r_n, \mathbf{X}).$$

[2] Clearly, allowing elements of $\mathscr{D}$ as parameters in the non-uniform definability case when they are not allowed in the uniform definability case is a little inconsistent. We have taken this approach since it corresponds to standard treatments in definability theory.

[3] Note that if we are discussing definability over an Herbrand universe, allowing individual constants is moot since each element of the Herbrand universe is named by a term.

There is a $\subseteq$-least interpretation of each r_i over $\mathscr{D}$—let us call in $r_i^{\mathscr{D}}$—so that the "rules" are all satisfied. These will in fact satisfy a stronger property: for each i,

$$r_i(\mathbf{X}) \leftrightarrow \phi_i(r_1, \ldots, r_n, \mathbf{X})$$

will hold. Any relation $r_i^{\mathscr{D}}$ so definable is said to be (*first-order positively*) *inductively definable*, or simply *inductive*. A relation whose complement is inductively definable is said to be *coinductively definable*, or *coinductive*.

The $r_i^{\mathscr{D}}$'s can be built up by transfinite induction. Define

$$r_i^{<\eta} = \bigcup_{\nu<\eta} r_i^{\nu} \quad \text{and} \quad r_i^{\eta} = \{\mathbf{X} : \phi_i(r_1^{<\eta}, \ldots, r_n^{<\eta}, \mathbf{X})\}.$$

(For example, $r_i^{<0} = \varnothing$, and that $r_i^0 = \{\mathbf{X} : \phi_i(\varnothing, \ldots, \varnothing, \mathbf{X})\}$.) The sequence of $r_i^{<\eta}$'s and the sequence of r_i^{η}'s are both $\subseteq$-increasing, so (by a simple cardinality argument) both sequences must reach fixed points. The least η such that each $r_i^{\eta} = r_i^{<\eta}$ (i.e., the least η such that each r_i^{η} is a fixed point) is called the *closure ordinal* of Φ. For any structure $\mathscr{D}$, the supremum of the closure ordinals of all such sets Φ, denoted $\kappa^{\mathscr{D}}$, is an important invariant of $\mathscr{D}$.

Inductive definability is related to definability in a very limited set of formulas in *second-order logic*, which allows quantifiers ranging over *all relations* on a structure as well as quantifiers ranging over all elements of a structure. A Π_1^1 formula is a formula of the form $\forall x_1 \forall x_2 \cdots \forall x_k \phi$, where the $\forall x_i$'s range over relations of the structure and ϕ is first order—involving both the relations, functions, and constants of the structure and the x_i's. A Σ_1^1 formula is a formula in the dual form—$\exists x_1 \exists x_2 \cdots \exists x_k \phi$, where ϕ is first order.

A relation s on a structure $\mathscr{D}$ is Σ_1^1- (resp. Π_1^1-) *definable* if it satisfies $\forall \mathbf{X}(s(\mathbf{X}) \leftrightarrow \Phi(\mathbf{X}))$, where formula Φ is Σ_1^1 (resp. Π_1^1). Again (a finite number of) elements of $\mathscr{D}$ may be explicitly mentioned. Over the natural numbers the class of Π_1^1-definable relations was extensively studied by Kleene and others; there the class coincides with the class of inductively definable relations. Extensions of this result can be found in [Mos74, Bar75]. In particular: (1) Over any structure $\mathscr{D}$, every inductively definable relation is Π_1^1-definable. (2) If $\mathscr{D}$ is countable and has a first-order definable—or inductively definable—*pairing function*, i.e., a function mapping $\mathscr{D} \times \mathscr{D}$ one-to-one into $\mathscr{D}$, then every Π_1^1-definable relation is inductively definable.

Every Herbrand universe can be represented in the obvious way inside the integers by Gödel-numbering.

THEOREM 2.1 [Folklore]. *A relation r on an infinite Herbrand universe U is inductively definable over U if and only if it is Π_1^1-definable over U, if and only if the corresponding set of Gödel numbers is inductively definable over the integers, if and only if the corresponding set of Gödel numbers is Π_1^1-definable over the integers.*

Proof. Suppose U has constants $c_1, \ldots, c_k$, $k \geq 1$, and functions $f_1, \ldots, f_m$, $m \geq 1$.[4] In order to simplify exposition, we shall assume that the f_i's are all binary. (However, we shall avoid a simplification in the proof that is possible if it is known that at least one f_i is not unary.) We shall similarly assume that all relations being defined, inductively or Π_1^1, are binary.

First, we know that inductively definable over $\mathbf{N} = \Pi_1^1$-definable over $\mathbf{N}$. Second, by Gödel numbering we, first, order define an isomorphic copy of U in $\mathbf{N}$. Accordingly, if a relation r is inductively (resp. Π_1^1) definable on U, the set of Gödel numbers for elements of r is inductively (resp. Π_1^1) definable on $\mathbf{N}$.

Third, if the set of Gödel numbers of elements in a relation is inductively (resp. Π_1^1) definable over the natural numbers (with operations $+$ and $\cdot$), show that the relation is also inductively (resp. Π_1^1) definable over the Herbrand universe U. Here we code a copy of $\mathbf{N}$ inside U. The set

$$\mathcal{N} = \{n_0 = c_1, n_1 = f_1(n_0, c_1), n_2 = f_1(n_1, c_1), \ldots, n_{i+1} = f_1(n_i, c_1), \ldots\}$$

is obviously inductively defined. The inductive definition of $U - \mathcal{N}$ over U is also easy, using the fact that U is an Herbrand model: $c_i \in U - \mathcal{N}$ for $i > 1$, $f_i(X, Y) \in U - \mathcal{N}$ for $i > 1$ or $Y \neq c_1$, and $f_1(X, c_1) \in U - \mathcal{N}$ for $X \in U - \mathcal{N}$. Treat each n_i as a code for natural number i. The successor function—$x \mapsto f_1(x, c_1)$—is first-order definable, so the $+$, $\cdot$, and exponentiation functions and their complements can be defined inductively by standard methods.

What we use to finish the proof is that the function F mapping each element t of U to the code $n_{\#t}$ for its Gödel number $\#t$ is inductively and coinductively definable. This is straightforward. To construct it inductively: Explicitly map each c_i to $n_{\#c_i}$. Suppose that the Gödel number for $f_i(X, Y)$ is $2 \cdot 3^i \cdot 5^{\#X} \cdot 7^{\#Y}$. Then if $(X, n_{\#X}) \in F$ and $(Y, n_{\#Y}) \in F$, then $(f_i(X, Y), n_{2 \cdot 3^i \cdot 5^{\#X} \cdot 7^{\#Y}}) \in F$—and all the necessary arithmetic on the codes in inductively defined. Also, if a function is inducitvely definable, it is also coinductively definable:

$$(X, Y) \notin F \leftrightarrow \exists Z((X, Z) \in F \wedge Y \neq Z).$$

Now suppose that r is a relation on Gödel numbers, inductively definable on $\mathbf{N}$. This inductive definition can be

[4] If $k = 0$, the Herbrand universe would be empty. If $m = 0$, the Herbrand universe would be finite, containing only $c_1, \ldots, c_k$.

mimicked inside U on the set $\mathcal{N}$ of codes, giving a set $\hat{r}$ of codes for Gödel numbers. Since F is inductively definable, $F^{-1}(\hat{r})$ is also inductively definable over U. And this is the desired set.

Since F is inductively definable over U, it is also Π^1_1-definable over U. It follows that if r is a Π^1_1-definable set of Gödel numbers then the corresponding set of terms in U is also Π^1_1-definable. ∎

The class of (parameterized classes of) relations which are (uniformly) inductively definable over all finite EDBs for a fixed set of EDB symbols is also well studied. In this case, since the EDBs are varied, using individual elements of the EDBs as parameters is not allowed unless there are specific EDB constants naming these relations. On EDBs with linear ordering relations, this is the class of polynomial-time computable relations; over general EDBs, the class of inductively definable relations is provably somewhat smaller than $\mathcal{P}$. (For example, one cannot test, in general, whether the EDB has an even number of elements.) (See [CH82, Var82, Imm86, GS86].) Fagin [Fa74] showed that the class of (parameterized classes of) relations which are uniformly Σ^1_1-definable over the class of all finite EDBs for a fixed set of EDB symbols is the set of $\mathcal{NP}$-definable relations—and thus also the uniformly Π^1_1-definable relations are the co-$\mathcal{NP}$-definable relations. The result that every inductively definable relation is also Π^1_1-definable can be made uniform in the EDBs.

3. THE SEMANTICS

We give here definitions of the semantics we shall consider, the two-valued and three-valued program completion semantics, the well-founded semantics, a variant of stable semantics, and the original variant of the stable model semantics, which we shall call the unique stable model semantics. We shall also cover some basic expressive power results for the semantics in this section.

3.1. *Program Completion Semantics*

The well-founded and stable semantics, our primary concern in this paper, bear strong simularities to the earlier program completion semantics. We shall start by discussing them and their expressive powers, summarizing one definition and some known results.

Clark [Cla78] proposed a notion of a completion of a logic program, a notion also discussed by Shepherdson [She85, She88]. Think of a logic program as being a set of implicit definitions of its IDB predicates. More specifically, the set of rules with head r is understood to be an implicit definition for r. Thus a relation $r(X_1, ..., X_k)$ should hold if *and only if* it is the head of some rule of the program, where the body of the rule is true. For each finite logic program, Clark defined a set of first-order formulas called the *completion* of the program, which formally state that intuition. Since we base our definitions upon instantiated programs only, our definiotion differs slightly from Clark's. We more nearly follow Fitting's definitions. While Clark's completion was a formula of first-order logic, our completion is a quantifier-free formula of an infinitary logic. Essentially, an infinite disjunction for us will correspond to Clark's existential quantifiers, an infinite conjunction to his universal quantifiers.

DEFINITION 3.1. Let **P** be an instantiated logic program. Then the *completion* of **P** is the set of formulas

$$\alpha \leftrightarrow \bigvee_i (\beta^i_1 \wedge \cdots \wedge \beta^i_{n^i}),$$

where α is an IDB literal of **P** and the disjunction is over all formulas $\beta^i_1 \wedge \cdots \wedge \beta^i_{n^i}$, where

$$\alpha \leftarrow \beta^i_1 \wedge \cdots \wedge \beta^i_{n^i}$$

is a rule of **P**. For **P** a logic program and $\mathcal{D}$ an EDB for **P**, the *completion of* **P** *over* $\mathcal{D}$ is the completion of the $\mathcal{D}$-instantiation of **P**.

Note that if there is a rule, say α, with no subgoals, then the corresponding disjunct is a conjunction of zero literals, which, by convention, is the formula **true**. In particular, if α is a positive EDB literal true in $\mathcal{D}$, then α is an element of the $\mathcal{D}$-instantiation of **P**, so the completion of **P** over $\mathcal{D}$ contains (a formula logically equivalent to) the formula $\alpha \leftrightarrow$ **true**. On the other hand, if there are no rules with head α, then the disjunction is over an empty set of formulas, which, by convention, is the formula **false**. In particular, if α is a positive EDB literal false in $\mathcal{D}$, then the completion of **P** over $\mathcal{D}$ contains the formula $\alpha \leftrightarrow$ **false**.

The original interpretation of the completion was a theory in ordinary two-valued logic. Fitting [Fit85] observed that Clark's completion has an especially nice interpretation in three-valued logic (truth values *true*, *false*, and *undetermined*), where the $\leftrightarrow$ of the completion is given a Łukasiewicz interpretation, shown in the truth tables below.[5] Note that the truth tables agree with the standard truth tables when restricted to the truth values *true* and *false*. Fitting's three-valued interpretations are our consistent partial interpretations, with truth values *true*, *false*, and *undetermined*, as described before.

[5] Kunen [Kun87] developed a very similar semantics which we shall not discuss here except in footnotes. Kunen's semantics makes two significant changes, one of which is to work over a larger EDB than the Herbrand universe.

$\wedge$	*true*	*false*	*undet.*
true	*true*	*false*	*undet.*
false	*false*	*false*	*false*
undet.	*undet.*	*false*	*undet.*

$\vee$	*true*	*false*	*undet.*
true	*true*	*true*	*true*
false	*true*	*false*	*undet.*
undet.	*true*	*undet.*	*undet.*

$\neg$	
true	*false*
false	*true*
undet.	*undet.*

$\leftrightarrow$	*true*	*false*	*undet.*
true	*true*	*false*	*false*
false	*false*	*true*	*false*
undet.	*false*	*false*	*true*

One can think of the truth tables for $\wedge$, $\wedge$, and $\neg$ as the natural extensions of the two-valued truth tables where *undet.* represents lack of knowledge. The extension to conjunctions and disjunctions of infinitely many formulas is the obvious one. Łukasiewicz's truth table for $\leftrightarrow$ is more difficult to grasp intuitively; one might think of it here as representing that all natural inferences have been made.

DEFINITION 3.2. Let **P** be a logic program and $\mathscr{D}$ be an EDB for **P**. A model of the completion of **P** over $\mathscr{D}$ is a consistent partial interpretation I which satisfies all formulas in the completion of **P** over $\mathscr{D}$.

The *three-valued program completion semantics*, given program **P** and an EDB $\mathscr{D}$ for **P**, infers all $\mathscr{D}$-instantiated literals true in all three-valued models of the completion of **P** over $\mathscr{D}$.

The *two-valued program completion semantics*, given a program **P** and an EDB $\mathscr{D}$ for **P**, infers all $\mathscr{D}$-instantiated literals true in all two-valued models of the completion of **P** over $\mathscr{D}$.

Observe that, for any program **P** over any EDB $\mathscr{D}$, every literal inferred by the three-valued program completion semantics is also inferred by the two-valued program completion semantics, since every two-valued model of the completion of **P** over $\mathscr{D}$ is also a (three-valued) model of the completion of **P** over $\mathscr{D}$.

Fitting noted that the completion above corresponds to an operator on partial interpretations:

DEFINITION 3.3. Define an operator **pc** on partial interpretations as follows: Let $\mathscr{D}$ be an EDB for logic program **P** and I a partial interpretation. A $\mathscr{D}$ instantiated literal α is in $\mathbf{pc}(I)$ if

- α is positive literal and it is the head of some $\mathscr{D}$-instantiated rule

$$\alpha \leftarrow \beta_1 \wedge \cdots \wedge \beta_k,$$

where each β_i is in I, or

- α is negative literal $\neg\gamma$ and for every $\mathscr{D}$-instantiated rule

$$\gamma \leftarrow \beta_1 \wedge \cdots \wedge \beta_k$$

of **P** with head α, the negation of some β_i is in I.

Fitting showed that a consistent partial interpretation is a model of the completion of **P** over $\mathscr{D}$ if and only if it is a fixed point of **pc**.[6] Hence the two-valued program completion semantics infers all literals true in all two-valued fixed points of **pc**, and the three-valued program completion semantics infers all literals true in all three-valued fixed points.

The operator **pc** is monotonic in I, so it has a $\subseteq$-least fixed point, which hence coincides with the set of inferences in the three-valued program completions semantics. Moreover, the fixed point can be constructed by transfinite induction.

$$\mathbf{pc}^0 = \mathbf{pc}(\varnothing),\ \mathbf{pc}^1 = \mathbf{pc}(\mathbf{pc}^0),\ \ldots,\ \mathbf{pc}^\omega = \mathbf{pc}\left(\bigcup_{n<\omega} \mathbf{pc}^n\right),\ \ldots.$$

In general,

$$\mathbf{pc}^\eta = \mathbf{pc}\left(\bigcup_{\nu<\eta} \mathbf{pc}^\nu\right).$$

This sequence must reach a fixed point—call it $\mathbf{pc}^\infty$—which is the least partial model of the program completion. (If $\mathscr{D}$ is a finite EDB, then some $\mathbf{pc}^n$ is the fixed point.)

THEOREM 3.1 [Fitting]. *For every logic program* **P** *and every* EDB $\mathscr{D}$ *for* **P**, *the least fixed point of* **pc** *is consistent, and hence is also the intersection of all (three-valued) models of the completion of* **P** *over* $\mathscr{D}$.

Proof. Suppose, for some program **P** and some EDB $\mathscr{D}$ for **P**, $\mathbf{pc}^\infty$ is inconsistent—i.e., contains both some literal α and its negation. Then there is a least ordinal η such that $\mathbf{pc}^\eta$ is inconsistent. Since consistency is a finitary property, $\bigcup_{\nu<\eta} \mathbf{pc}^\nu$ is consistent. Now suppose that both α, $\neg\alpha \in \mathbf{pc}^\eta$. Since $\alpha \in \mathbf{pc}^\eta$, for some instantiated rule with head α, every subgoal is in $\bigcup_{\nu<\eta} \mathbf{pc}^\nu$. On the other hand, since $\neg\alpha \in \mathbf{pc}^\eta$, the negation of some subgoal of that rule is also in $\bigcup_{\nu<\eta} \mathbf{pc}^\nu$. That is a contradiction to the consistency of $\bigcup_{\nu<\eta} \mathbf{pc}^\nu$. ∎

[6] Our definitions are slightly different from Fitting's, but that is insignificant here.

By contrast, the set of inferences in the two-valued program completion semantics is not normally two-valued (it can have either undetermined or overdetermined truth values) and need not be a fixed point of **pc**.

EXAMPLE 3.1. 1. Let **P** be the program

$$\{r(A) \leftarrow r(B),\ r(B) \leftarrow \neg r(C),\ r(B) \leftarrow \neg r(D),$$
$$r(B) \leftarrow r(E),\ r(C) \leftarrow \neg r(D),\ r(D)\}$$

and let $\mathscr{D}$ be the Herbrand universe $\{A, B, C, D, E\}$. Then the completion of **P** over $\mathscr{D}$ is

$$\{r(A) \leftrightarrow r(B),\ r(B) \leftrightarrow \neg r(C) \vee \neg r(D) \vee r(E),$$
$$r(C) \leftrightarrow \neg r(D),\ r(D) \leftrightarrow \textbf{true},\ r(E) \leftrightarrow \textbf{false}\}$$

which has a unique three-valued model,

$$\{\neg r(E),\ r(D),\ \neg r(C),\ r(B),\ r(A)\},$$

which is also two-valued. Hence both semantics infer exactly these literals.

2. Let $\mathbf{P} = \{r(A) \leftarrow \neg r(B),\ r(B) \leftarrow \neg r(A),\ r(C) \leftarrow r(A),\ r(C) \leftarrow r(B)\}$ and $\mathscr{D}$ be the Herbrand universe $\{A, B, C\}$. The completion is

$$\{r(A) \leftrightarrow \neg r(B),\ r(B) \leftrightarrow \neg r(A),\ r(C) \leftrightarrow r(A) \vee r(B)\}$$

which has 2 two-valued models, $\{r(A), \neg r(B), r(C)\}$ and $\{\neg r(A), r(B), r(C)\}$, and an additional three-valued model, $\varnothing$. Hence the two-valued program completion semantics infers $r(C)$ while the three-valued program completion semantics infers nothing.

3. Let $\mathbf{P} = \{r(A) \leftarrow \neg r(A), r(B)\}$, and $\mathscr{D}$ be the Herbrand universe $\{A, B\}$. The completion, $\{r(A) \leftrightarrow \neg r(A), r(B) \leftrightarrow \textbf{true}\}$, has only one three-valued model, $\{r(B)\}$. Hence the three-valued program completion semantics infers $r(B)$. Since the completion has no two-valued models, the two-valued program completion semantics infers the intersection of the empty set of partial models, $\{r(A), \neg r(A), r(B), \neg r(B)\}$.

4. Let $\mathbf{P} = \{r(A) \leftarrow r(A)\}$. The atomic formula $r(A)$ depends positively on itself—a simple *positive recursion*. For $\mathscr{D} = \{A\}$, the completion of **P** is $\{r(A) \leftrightarrow r(A)\}$, which is a tautology. So neither the two-valued program completion semantics nor the three-valued program completion semantics infers any literals. But **P** is a Horn-clause program, and the van Emden–Kowalski semantics [VEK76] infers $\neg r(A)$. Thus the program completion semantics do not capture at least one well-accepted notion of negation as failure. By contrast, the well-founded and stable semantics will identify the positive recursion and infer $\neg r(A)$.

EXAMPLE 3.2. The relation $=$ is not a "logical relation" in logic programming. That is, it is just another relation to be defined by the programmer, however the programmer wishes to define it. However, in the program completion semantics, both the intended relation $=$ and its negation $\neg =$ are definable.

Let **P** be a logic program which uses the relation $=$ as an EDB relation. Let $\mathbf{P}_{\text{Equality}}$ be the program consisting of **P** plus rules defining the symbols $=$ and $\neq$:

$$\mathbf{P} \cup \{=(X, X),\ \neq(X, Y) \leftarrow \neg =(X, Y)\}.$$

Then the completion of $\mathbf{P}_{\text{Equality}}$ over any database $\mathscr{D}$ is the completion of **P** over $\mathscr{D}$, together with

$$\{=(d, d) \leftrightarrow \textbf{true} : d \in \mathscr{D}\}$$
$$\cup \{=(d, e) \leftrightarrow \textbf{false} : d, e \in \mathscr{D}, d \neq e\}$$
$$\cup \{\neq(d, e) \leftrightarrow \neg =(d, e) : d, e \in \mathscr{D}\}.$$

Clearly, this completion correctly defines both $=$ and $\neq$, in both two-valued and three-valued program completion semantics.

Thus, one can freely use $=$ as if it were an EDB relation with the standard interpretation in both program completion semantics. Exactly the same construction shows that $=$ may be treated as an EDB relation in the well-founded and stable semantics, which we define next.

A well-known property of several logic programming semantics is that it does not increase the expressive power of the semantics to allow "rules" $p \leftarrow \phi$, where ϕ is any formula of first-order logic (as opposed to just a conjunction of literals). This is worked out in detail, for example, for the well-founded semantics in [VG93]. We illustrate the procedure in the example below; this translation works for the two- and three-valued program completion semantics, the well-founded semantics, and the stable semantics.[7]

EXAMPLE 3.3. The "rule" $p(X) \leftarrow \forall Y\, \exists Z(r(X, Y, Z) \vee r(Y, Z, X))$ can be simulated as follows: First rewrite the formula ϕ using only $\wedge$, $\neg$, and $\exists$: $p(X) \leftarrow \neg\exists Y \neg\exists Z \neg(\neg r(X, Y, Z) \wedge \neg r(Y, Z, X))$. Now, adding separate relations for most of the subformulas, construct the following logic program, working inductively from the inside out:

$$p_0(X, Y, Z) \leftarrow \neg r(X, Y, Z) \wedge \neg r(Y, Z, X)$$
$$p_1(X, Y, Z) \leftarrow \neg p_0(X, Y, Z)$$
$$p_2(X, Y) \leftarrow p_1(X, Y, Z)$$
$$p_3(X, Y) \leftarrow \neg p_2(X, Y)$$
$$p_4(X) \leftarrow p_3(X, Y)$$
$$p(X) \leftarrow \neg p_4(X, Y).$$

[7] However, with the well-founded and stable semantics one must be careful about the relationship between this translation and positive recursion.

It is fairly easy to see that the above rules "capture the meaning of the rule" $p(X) \leftarrow \forall Y \exists Z(r(X, Y, Z) \vee r(Y, Z, X))$—but we must first be precise about the meaning of the quantifiers. Let **P** be a normal logic program containing the above six rules but no other occurrences of p_0, p_1, p_2, p_3, p_4. Let $\mathscr{D}$ be any EDB $\mathscr{D}$ for **P**.

1. In a two-valued model of the program completion, define $\forall X\psi(X)$ to be true in the model if, for every $m \in \mathscr{D}$, $\psi(m)$ is true in the model, and false otherwise. Define the truth value of $\exists X\psi(X)$ analogously.

2. In a three-valued model of the program completion, define $\forall X\psi(X)$ to be true the model if, for every $m \in \mathscr{D}$, $\psi(m)$ is true in the model, false if for some element m of the model, $\psi(m)$ is false in the model, and undefined otherwise. Define the truth value of $\exists X\psi(X)$ analogously.

Then, in any model (two-valued or three-valued) of the completion of **P**, and for any $m \in \mathscr{D}$, the truth value of $p(m)$ is equal to the truth value of $\forall Y \exists Z(r(m, Y, Z) \vee r(Y, Z, m))$.

THEOREM 3.2. *Over all EDBs, the two-valued program completion semantics is at least as expressive as the three-valued program completion semantics. Over all classes of EDBs, the two-valued program completion semantics is uniformly at least as expressive as the three-valued program completion semantics.*

Proof. Note that, as in Example 3.1, parts 2 and 3, a single program may not define the same relations in the two semantics. To prove the theorem, we show a construction which, given a program **P** defining a relation r in the two-valued semantics, gives another program **P'** defining r in the three-valued semantics—over all EDBs $\mathscr{D}$ for **P**. The proof involves what is sometimes called an "approximation logic"; compare [Fit91, Sch91, BSu91].

For each relation r, add a new IDB relation $\tilde{r}$. For each rule of program **P**—the example

$$r_1(t_1^1, ..., t_k^1) \leftarrow r_2(t_1^2, ..., t_k^2) \wedge r_3(t_1^3, ..., t_k^3) \wedge \neg r_4(t_1^4, ..., t_k^4) \wedge \neg r_5(t_1^5, ..., t_k^5)$$

is general enough to illustrate the translation—include two rules in **P'**:

$$r_1(t_1^1, ..., t_k^2) \leftarrow r_2(t_1^2, ..., t_k^2) \wedge r_3(t_1^3, ..., t_k^3) \wedge \neg \tilde{r}_4(t_1^4, ..., t_k^4) \wedge \neg \tilde{r}_5(t_1^5, ..., t_k^5)$$

and

$$\tilde{r}_1(t_1^1, ..., t_k^1) \leftarrow \tilde{r}_2(t_1^2, ..., t_k^2) \wedge \tilde{r}_3(t_1^3, ..., t_k^3) \wedge \neg r_4(t_1^4, ..., t_k^4) \wedge \neg r_5(t_1^5, ..., t_k^5).$$

Finally, for each EDB relation r, include in **P'** the rule

$$\tilde{r}(X_1, ..., X_k) \leftarrow r(X_1, ..., X_k).$$

(The relation r is an EDB relation, but $\tilde{r}$ must be an IDB relation.)

Given a partial interpretation I for the $\mathscr{D}$-instantiation of P, construct a partial interpretation I' as follows:

- $r(d_1, ..., d_k) \in I'$ if and only if $r(d_1, ..., d_k) \in I$; otherwise, $\neg r(d_1, ..., d_k) \in I'$;
- $\neg \tilde{r}(d_1, ..., d_k) \in I'$ if and only if $\neg r(d_1, ..., d_k) \in I$; otherwise $\tilde{r}(d_1, ..., d_k) \in I'$.

Clearly, this function is a one-to-one correspondence from the set of partial interpretations for **P** over $\mathscr{D}$ to the set of two-valued partial interpretations for **P'** over $\mathscr{D}$. We will be finished if we show that I is a fixed point of the **pc** operator for **P** over $\mathscr{D}$ if and only if I' is a fixed point of the **pc** operator for **P'** over $\mathscr{D}$.

For any positive $\mathscr{D}$-instantiated literal $\alpha = r(d_1, ..., d_k)$ of **P**, let $\tilde{\alpha} = \tilde{r}(d_1, ..., d_k)$. To simplify notation, we write the proof as if each rule of the instantiated program were of the form $\alpha \leftarrow \beta_1 \wedge \beta_2 \wedge \neg\gamma_1 \wedge \gamma_2$, where α, the β_i's, and the γ_j's are all positive literals.

We give one typical step of the proof. Assume that I is a fixed point of the **pc** operator for **P** over $\mathscr{D}$. We check that I' obeys the fixed point property for negative literals $\neg\tilde{\alpha}$:

$\neg\tilde{\alpha} \in I'$ iff $\neg\alpha \in I$ (by definition of I')

iff for every instantiated rule $\alpha \leftarrow \beta_1 \wedge \beta_2 \wedge \neg\gamma_1 \wedge \neg\gamma_2$ of **P**, some $\neg\beta_i \in I$ or some $\gamma_j \in I$ (since I is a fixed point)

iff for every instantiated rule $\tilde{\alpha} \leftarrow \tilde{\beta}_1 \wedge \tilde{\beta}_2 \wedge \neg\gamma_1 \wedge \neg\gamma_2$ of P', some $\neg\tilde{\beta}_i \in I'$ or some $\gamma_j \in I'$ (by definition of I'). ∎

The following theorem is a (minor variant of a) result of Kolaitis and Papadimitriou.

THEOREM 3.3 [KP88]. *A relation s on EDB $\mathscr{D}$ is definable in the two-valued program completion semantics if and only if it is Π_1^1 definable on $\mathscr{D}$. A (parameterized collection of) relation(s) s on the collection of finite EDBs for a fixed set of EDB relations is definable in the two-valued program completion semantics if and only if it is* co-$\mathcal{NP}$.

THEOREM 3.4 (Essentially [Fit85]). *A relation s on an EDB $\mathscr{D}$ is definable in the three-valued program completion smantics if and only if s is inductively definable over $\mathscr{D}$. A (parameterized collection of) relation(s) s on the collection of finite EDBs for a fixed set of EDB relations is definable in the*

three-valued program completion semantics if and only if it is (uniformly) inductively definable.

COROLLARY 3.5. 1. *Over infinite Herbrand models, the two-valued and three-valued program completion semantics are equally expressive, i.e., exactly the same sets are definable in both.*

2. *Over the class of all finite EDBs for a fixed set of EDB relations (since those EDBs are, in general, not linearly ordered by any of their relations) the two-valued program completion semantics is more expressive than the three-valued program completion semantics.*

3. *If attention is restricted to classes of relations over finite EDBs which are trivial if the EDBs are not linearly ordered by some fixed relation, the two-valued program completion semantics is more expressive than the three-valued program completion semantics if and only if* $\mathcal{P} \neq \mathcal{NP}$.

Proof. 1. The three-valued program completion semantics defines the relations which are inductively definable over the Herbrand universe. The two-valued program completion semantics defines the relations which are Π^1_1 definable over the Herbrand universe. By Theorem 2.1, those two are the same.

2. The three-valued program completion semantics defines the relations which are uniformly inductively definable. The two-valued program completion semantics defines the sets which are co-$\mathcal{NP}$-definable. A set which is empty if the EDB has an even number of elements and the entire EDB if the EDB has an odd number of elements is thus definable in the two-valued semantics but not three-valued.

3. The three-valued program completion semantics here defines all **P**-time computable relations; the two-valued, all co-$\mathcal{NP}$-time computable relations.

3.2. *The Well-Founded and Stable Semantics*

The well-founded semantics has one basic difference from the three-valued program completion semantics; it detects positive recursion—see Example 3.1(4), where it will infer $\neg r(A)$. The well-founded semantics was originally proposed in [VGRS91]. Subsequently several equivalent definitions have been given, including well-known definitions such as in [VG93, Prz89]. We give here a variant of Van Gelder's alternating fixed point definition [GVG93], devised jointly with Van Gelder to facilitate this work.[8]

Van Gelder's alternating fixed point definition was based upon the Gelfond–Lifschitz transformation of a logic program [GL88]. The definition below is Van Gelder's minor variant of the original definition.

[8] The definition was derived from several related notions presented at the "Eighth ACM Symposium on Principles of Database Systems" [Ros89, VG93, Prz89, Bry89].

DEFINITION 3.4. Let **P** be a logic program, $\mathcal{D}$ be an EDB for **P**, and I be a set of negative $\mathcal{D}$-instantiated literals of **P**. The *Gelfond–Lifschitz transform* $\mathbf{P}_I$ of **P** is the program formed from the $\mathcal{D}$-instantiation $\mathbf{P}_{\mathcal{D}}$ of **P** by replacing each *negative subgoal* $\neg\beta$ in $\mathbf{P}_{\mathcal{D}}$ with **true** if $\neg\beta \in I$ and with **false** if $\neg\beta \notin I$.

The Gelfond–Lifschitz transform of a program is (trivially equivalent to, under Van Gelder's variant) a Horn clause program. Gelfond and Lifschitz used it in defining the stable semantics; Van Gelder used it in (re)defining the well-founded semantics. A basic step of both is to construct the minimal model of the $\mathbf{P}_I$, exactly as in the van Emden–Kowalski semantics for Horn clause programs [VEK76], by an induction over the natural numbers. For recursion theoretic reasons, we need in this paper to be able to discuss a finite object—a "proof"—that demonstrates that a certain positive literal is inferred in the minimal model of the Gelfond–Lifschitz transform. Our forward proofs from negative hypotheses (see below) exactly capture this construction.

An important point of the van Emden–Kowalski construction is that inferences are made using only two of the traditional proofs rules of first order logic: *modus ponens* and $\wedge$-introduction.[9] This sort of proof does not allow, for example, inference by contraposition. We thus think of this reasoning style as "forward"—in the forward direction of the "if" symbol. One very natural way of motivating semantics for normal logic programs is that this restriction matches the intended meaning of rules (although negation as failure can complicate the issue somewhat).

DEFINITION 3.5. Let $\mathcal{D}$ be an EDB for a logic program **P**, and let α be a $\mathcal{D}$-instantiated positive literal of **P**. A *forward proof of* α *from negative hypotheses over* **P**, $\mathcal{D}$ (or, simply, a *forward proof of* α), is a finite sequence $s_1, \ldots, s_n$ of $\mathcal{D}$-instantiated formulas of **P**, where

- Each statement s_i:
  - (1) is an $\mathcal{D}$-instantiated rule of **P**,
  - (2) follows from two previous s_j's by *modus ponens*,
  - (3) is a conjunction of two previous s_j's (i.e., follows from previous s_j's using $\wedge$-introduction), or
  - (4) is a negative $\mathcal{D}$-instantiated literal of **P**.
- The last formula is α.

The s_i's which are negative literals are called *hypotheses* of the proof. We explicitly allow $\neg\alpha$ to be one of the hypotheses of a forward proof of α from negative hypotheses. A forward proof of a literal α from negative hypotheses is *minimal* if no proper subset of the proof is also a forward proof of α from negative hypotheses.

[9] *Modus ponens* is from $\alpha \leftarrow \beta$ and α infer β; $\wedge$-introduction is from α and β infer $\alpha \wedge \beta$.

Observation 3.6. Since forward proofs from negative hypotheses are finite, every forward proof of any literal α from negative hypotheses contains a subset which is a minimal forward proof from negative hypotheses.

Thus, for any set I of literals and for any instantiated program **P**, there is a forward proof p of α from negative hypotheses, where all the hypotheses of p are true in I, if and only if there is such a minimal proof, and there is a forward proof of α from negative hypotheses, where no hypotheses of p are false in I, if and only if there is such a minimal proof.

If every rule of a program has at most m subgoals, then every formula in a forward proof of a literal α from negative hypotheses will be (1) a literal, or (2) a conjunction of at most m literals, or (3) an instantiated rule of the program.

In Example 3.1, Part 4, note that there is no forward proof of $r(A)$ from negative hypotheses, since any proof of $r(A)$ would have to first prove $r(A)$—an infinite descent of positive literals.

DEFINITION 3.6. The *stable completion* of **P** over $\mathscr{D}$ is the set of all formulas

$$\alpha \leftrightarrow \bigvee_i (\neg\beta_1^i \wedge \cdots \wedge \neg\beta_{k^i}^i),$$

where α is a positive $\mathscr{D}$-instantiated literal of **P**, where the disjunction is over all minimal forward proofs p_i of α from negative hypotheses and where proof p_i has hypotheses $\neg\beta_1^i, \ldots, \neg\beta_{k^i}^i$. Note that the disjunction may be over infinitely many formulas.

The formula

$$\alpha \leftrightarrow \bigvee_i (\beta_1^i \wedge \cdots \wedge \beta_{k^i}^i)$$

above is called the *definition* of α in the stable completion.

DEFINITION 3.7. The *well-founded partial model* of program **P** over EDB $\mathscr{D}$ is the set of all $\mathscr{D}$-instantiated literals **true** in all three-valued models of the stable completion of **P** over $\mathscr{D}$, i.e., the intersection of all three-valued models of the stable completion of **P** over $\mathscr{D}$. Given a program **P** and an EDB $\mathscr{D}$ for **P**, the *well-founded semantics* infers all $\mathscr{D}$-instantiated literals of **P** **true** in the well-founded partial model for **P** over $\mathscr{D}$.

Exactly corresponding to Fitting's operator, which we called **pc**, there is an operator **wf** on partial interpretations giving the well-founded semantics.

DEFINITION 3.8. The operator **wf** on partial interpretations is as follows: Let $\mathscr{D}$ be an EDB for logic program **P** and I a partial interpretation. A $\mathscr{D}$ instantiated literal α is in **wf**(I) if

- α is a positive literal and there is a minimal forward proof p of α from negative hypotheses over **P**, $\mathscr{D}$, where each hypothesis of p is in I, or
- α is a negative literal $\neg\gamma$ and for each minimal forward proof p of γ from negative hypotheses over **P**, $\mathscr{D}$, the negation of some hypothesis of p is in I.

Using this operator, the least fixed point can be constructed by transfinite induction, exactly analogously to the construction of the least (three-valued) model of the program completion:

$$\mathbf{wf}^0 = \mathbf{wf}(\varnothing),\ \mathbf{wf}^1 = \mathbf{wf}(\mathbf{wf}^0),\ \ldots,\ \mathbf{wf}^\omega = \mathbf{wf}\left(\bigcup_{n<\omega} \mathbf{wf}^n\right),\ \ldots.$$

In general,

$$\mathbf{wf}^\eta = \mathbf{wf}\left(\bigcup_{\nu<\eta} \mathbf{wf}^\nu\right).$$

This sequence must reach a fixed point—call it $\mathbf{wf}^\infty$. (If $\mathscr{D}$ is a finite EDB for a program **P** with no function symbols, then some $\mathbf{wf}^n$ is the fixed point.)

THEOREM 3.7 (Essentially [VGRS91, VG93]). *For every logic program* **P** *and every* EDB $\mathscr{D}$ *for* **P**, *the* **least** *fixed point of the operator* **wf** *is consistent. This fixed point is thus the well-founded partial model of* **P** *over* $\mathscr{D}$ *and is a (three-valued) model of the stable completion of* **P** *over* $\mathscr{D}$.

Proof (Analogous to the proof of Theorem 3.1).

In Example 3.1, part 4, since there is no negatively founded proof tree for $r(A)$, the well-founded semantics infers $\neg r(A)$. A basic property of the well-founded semantics, developed in [VGRS91], is that it extends many of the common semantics for logic programming, in particular, the van Emden–Kowalski semantics for Horn clause programs [VEK76], the stratified semantics [ABW88, CH85, Lif88, Prz88, VG86], and the three-valued program competion semantics—extends in the sense that it makes all the inferences those semantics make and makes the same inferences when those semantics infer two-valued sets of inferences.

DEFINITION 3.9. A logic program **P** is *Horn over* EDB $\mathscr{D}$ if no IDB relation appears negatively in **P**.

THEOREM 3.8 [VGRS91]. *Suppose program* **P** *is Horn over EDB* $\mathscr{D}$. *Then the well-founded partial model is two-valued and is the same as the van Emden–Kowalski semantics* [VEK76] *for* **P** *over* $\mathscr{D}$, *which contains the positive* $\mathscr{D}$*-instantiated literals of* **P** *true in all (classical, two-valued) models of* **P** *over* $\mathscr{D}$ *satisfying all EDB literals true in* $\mathscr{D}$, *plus the negations of all other positive* $\mathscr{D}$*-instantiated literals of* **P**.

As Example 3.1, part 4 shows, neither the two-valued program completion semantics nor the three-valued program completion semantics satisfies (the property of) the above theorem.

The stable semantics is defined in terms of what are called stable models of programs.

DEFINITION 3.10 [GL88]. A *stable model* for a program **P** over an EDB $\mathscr{D}$ is a two-valued partial interpretation I, where I is the van Emden–Kowalski (i.e., minimal) model of the Gelfond–Lifschitz transform $\mathbf{P}_I$.[10]

The stable semantics infers all literals true in all stable models of program **P** over $\mathscr{D}$. If there is a unique stable model of **P** over $\mathscr{D}$, then the *unique stable model semantics* infers all literals true in that model. If there are no stable models, or if there is more than one stable model, the *unique stable models semantics* applied to that program and EDB is undefined.

Observation 3.9. Let **P** be a logic program, $\mathscr{D}$ be an EDB for **P**, and I be a two-valued partial interpretation for **P** over $\mathscr{D}$:

1. I is a stable model of **P** if and only if I is a model of the stable completion of **P** over $\mathscr{D}$.

2. I is a stable model of **P** if and only if (1) it is a model of **P** in the sense of classical logic, (2) each $\mathscr{D}$-instantiated EDB literal is in I just in case it is true in $\mathscr{D}$, and (3) for every positive $\mathscr{D}$-instantiated IDB literal α true in I there is a minimal forward proof of α from negative hypotheses, where all the hypotheses are in I.

Gelfond and Lifschitz originally defined the unique stable model semantics, but the stable model semantics defined above is a common generalization. Note that if there are no stable models of **P** over $\mathscr{D}$, the stable semantics infers all $\mathscr{D}$-instantiated literals, so all truth values are overdetermined. One might think of **P** as being incoherent when applied to $\mathscr{D}$.

Gelfond and Lifschitz showed that the unique stable model semantics satisfies many criteria for a logic programming semantics. For example, it agrees with the van Emden–Kowalski semantics on Horn clause programs. It similarly agrees on programs which are Horn over their EDBs. It follows that the stable model semantics has the same nice properties. One difficulty with the unique stable model semantics is that it is hard to decide whether the semantics is defined. For example, there are many logic programs **P** for which it is co-$\mathscr{NP}$ hard (as a function of the size of the EDB $\mathscr{D}$) to decide whether there is a unique stable model of **P** over $\mathscr{D}$. We, on the other hand, would like to speak of having the semantics assign some meaning to every program. Hence we shall concentrate primarily upon the stable semantics rather than upon the unique stable model semantics.[11]

THEOREM 3.10. *The well-founded semantics is at least as expressive as the three-valued program completion semantics—over any EDB, over any class of EDBs, and uniform over any class of EDBs. The stable semantics is at least as expressive as the two-valued program completion semantics—over any EDB, over any class of EDBs, and uniformly over any class of EDBs.*

Proof (Sketch). It will suffice to prove the following: Let **P** be a logic program. Then there is another logic program $\tilde{\mathbf{P}}$, where, for each EDB $\mathscr{D}$ for **P** and each $\mathscr{D}$-instantiated literal α of **P**, α is inferred from **P** and $\mathscr{D}$ in the three-valued program completion semantics (respectively, the two-valued program completion semantics) if and only if α is inferred from $\tilde{\mathbf{P}}$ and $\mathscr{D}$ in the well-founded semantics (respectively, the stable semantics).

Form $\tilde{\mathbf{P}}$ from **P** as follows. Replace positive subgoals with double negatives: If a relation r appears in a positive subgoal of a rule

$$\alpha \leftarrow \beta_1 \wedge \cdots \wedge r(t_1, \ldots, t_k) \wedge \cdots \wedge \beta_k,$$

replace the positive subgoal $r(t_1, \ldots, t_k)$ with a negative subgoal $\neg\tilde{r}(t_1, \ldots, t_k)$, forming

$$\alpha \leftarrow \beta_1 \wedge \cdots \wedge \neg\tilde{r}(t_1, \ldots, t_k) \wedge \cdots \wedge \beta_k,$$

and add a rule

$$\tilde{r}(t_1, \ldots, t_k) \leftarrow \neg r(t_1, \ldots, t_k).$$

Introducing this double negative does not change the program completion semantics, but it eliminates all positive recursion, which makes the well-founded and stable semantics reduce to the three-valued and two-calued program completion semantics.

Somewhat more formally, since no rule has any positive subgoals, a minimal forward proof of a literal α from negative hypotheses must infer α by applying *modus ponens* to a rule with head α and all negative subgoals; these negative subgoals must all be hypotheses in any proof using this rule; by minimality, that is all there is to the proof. Hence a disjunction over all minimal forward proofs corresponds exactly to a disjunction over all rules. ∎

COROLLARY 3.11 [VG93]. *Any relation inductively definable on an EDB $\mathscr{D}$ is definable over $\mathscr{D}$ in the well-founded semantics.*

COROLLARY 3.12. *Any relation Π^1_1 definable on an EDB D is definable over $\mathscr{D}$ in the stable semantics.*[12]

THEOREM 3.13 [Sch91]. *Over all EDBs, the stable semantics is at least as expressive as the well-founded semantics. Over all classes of EDBs, the stable semantics*

[10] Here we treat the van Emden–Kowalski model to be the set of ground literals inferred from $\mathbf{P}_I$, plus the negations of all other ground literals.

[11] Modifying the statement to make the unique stable model semantics infer $\varnothing$ if there is not a unique stable model does not help matters particularly from the perspective of this paper.

[12] The same result, for $\mathscr{D}$ the natural infinite Herbrand model, has been separately proved by Marek, Nerode, and Remmel; see the remark after Theorem 4.4.

is uniformly at least as expressive as the well-founded semantics.

Proof (Analogous to the proof of Theorem 3.2). The proof is worked out in detail in [Sch91].

3.3. *General Expressive Power Summary*

We summarize the expressiveness of the four semantics with the following diagram. An arrow pointing from a semantics S_1 to a semantics S_2 indicates that S_2 is at least as expressive as S_1. Here it turns our that it also indicates, for any program given **P** and for any EDB $\mathscr{D}$ for **P**, that S_2 infers all literals that S_1 infers, and perhaps others:

3-valued program completion semantics → well-founded semantics
↓ ↓
2-valued program completion semantics → stable semantics

4. THE EXPRESSIVE POWERS OF THE WELL-FOUNDED AND STABLE SEMANTICS OVER PARTICULAR CLASSES OF EDBs

4.1. *Uniform Expressibility over Finite Databases*

It has been noted already that uniform definability in the three-valued program completion semantics over finite EDBs is equivalent to inductive definability, which is equivalent in turn to polynomial time definability when the EDBs are linearly ordered.

THEOREM 4.1 [VG93]. *A relation r is definable in the well-founded semantics uniformly over all finite EDBs (for any fixed set of relation symbols) if and only if r is uniformly inductively definable over that set of EDBs.*

Proof. By Corollary 3.11, every uniformly inductively definable relation is also definable in the well-founded semantics. (The construction is uniform in the finite EDBs.) But the converse is also true. The proof in [VGRS91] that, for any fixed program **P**, the well-founded partial model is constructible in polynomial time (in the size of the EDB), can easily be converted into a proof that the well-founded partial model is uniformly inductively definable, using a result of [Imm86]: if a relation is inductively definable uniformly over all finite structures, then it is also coinductively definable uniformly over all finite structures.

The stable semantics is more powerful. Marek and Truszczynski [MT91] proved that deciding whether a propositional logic program **P** has a stable model is $\mathscr{NP}$-complete. A minor modification of their proof gives that there is a function-free logic program **P**, where, as a function of EDBs $\mathscr{D}$, deciding whether the stable completion of **P** over $\mathscr{D}$ is $\mathscr{NP}$-complete. We give a slightly stronger result here.

THEOREM 4.2. *A relation r on finite databases is definable in the stable semantics if and only if it is co-$\mathscr{NP}$.*

Proof. Both directions of the proof hinge upon Fagin's result [Fa74] that a (parameterized collection of) relation(s) r on the class of finite EDBs for a set of EDB relations in $\mathscr{NP}$ if and only if it is Σ^1_1 definable.

First show that if a (parameterized class of) relation(s) is definable in the stable semantics, it is (uniformly) co-$\mathscr{NP}$, or as it seems easier to think of,

$$\{(d_1, ..., d_k) : \exists \mathscr{M} (\mathscr{M} \text{ is a stable model of } \mathbf{P} \text{ and } r(d_1, ..., d_k) \in \mathscr{M})\}$$

is $\mathscr{NP}$. This is already almost of the desired form. The difficulty is that the definition of being a stable model was not first order.

To complete this, we use a single relation, of high arity, to represent both $\mathscr{M}$ and a single forward proof from negative hypotheses—essentially, the concatenation of minimal proofs of all the positive literals in $\mathscr{M}$. To do this, represent each step in the proof as a single tuple. To simplify exposition, we shall make two simplifying assumptions; it is fairly straightforward to adjust this for programs with arbitrary relations: (1) Assume that each relation has the same number—call it n—of arguments. (2) Assume that each rule with any subgoals has the same number—call it m—of subgoals. Also, since the only conjunctions usable in rules are conjunctions of m literals, we can replace normal $\wedge$-introduction with an inference rule which deduces the conjunction of m literals from the m literals separately. And then the proof need only store three types of objects: (1) rules, (2) individual literals, and (3) conjunctions of m literals.

To simplify encoding, we shall assume that there are two constant symbols, c_1 and c_2, appearing in all EDBs.[13] Use fixed-length sequences of these two to represent the relation symbols, in a binary coding; say l bits are used to code the relations. Then any step can of any minimal forward proof from negative hypotheses can be represented with a $2 + 1 + l + n + m(1 + l + n)$ tuple of elements of $\mathscr{D}$. Use the first two positions to identify what type the formula is: c_1, c_1 for a rule, c_2, c_2 for a literal, c_2, c_1 for a conjunction of m literals. The next $1 + l + n$ positions can be used to describe the head of the rule, or the single literal: one position for positive (c_1) versus negative (c_2); l positions give the binary code for the relation symbol using c_1 and c_2, and n positions give the arguments of the instantiated literal—n arbitrary elements of $\mathscr{D}$. The remaining $m(1 + l + n)$ positions similarly store

[13] Recall that we assumed each EDB has at least two elements. The assumption that there are always constant symbols c_1 and c_2 could be avoided in the following encoding, essentially by our replacing c_1 with a pair (x, y), where $x = y$ and c_2 with a pair (x, y) where $x \neq y$. This would only increase the arities of the relations and somewhat complicate the encoding.

the m elements of a conjunction or the m subgoals of a rule. All this coding can be described in first-order logic, uniformly in the EDBs (using the constants c_1, c_2).

To represent

$$\exists \mathscr{M}\ (\mathscr{M} \text{ is a stable model of } \mathbf{P})$$

use a formula $\exists \mathscr{H} \phi$, where ϕ "says" that:

- $\mathscr{H}$ is a linear ordering of a set of (distinct) $2 + 1 + l + n + m(1 + l + n)$-tuples of elements of $\mathscr{D}$, representing instantiated literals, conjunctions of m instantiated literals, and instantiated rules as above. Call the elements of the ordered set *steps* of $\mathscr{H}$. For simplicity of exposition, we now identify the tuples of elements of $\mathscr{D}$ with the literals, conjunctions of literals, and instantiated rules they represent.
- Each step of $\mathscr{H}$ is one of the following:
  1. a negative $\mathscr{D}$-instantiated IDB literal.
  2. a $\mathscr{D}$-instantiated EDB literal (positive or negative) true in $\mathscr{D}$.
  3. a $\mathscr{D}$-instantiated rule of **P**. (Recall that **P** is fixed, so this can be described in first-order logic as being a substitution instance of one of a fixed number of patterns.)
  4. the conjunction of m previous literals.
  5. a positive literal α, where there are two previous steps, $\alpha \leftarrow \beta_1 \wedge \cdots \wedge \beta_m$ and $\beta_1 \wedge \cdots \wedge \beta_m$.
- For each positive $\mathscr{D}$-instantiated literal α of **P**, exactly one of α and $\neg\alpha$ is a step of $\mathscr{H}$.
- Finally, $\{\alpha : \alpha \text{ is a step of } \mathscr{H}\}$ is a two-valued model of **P**, in the sense of *classical logic*.

All these properties of $\mathscr{H}$ can clearly be expressed in first-order logic for any fixed program **P**.

Now if $\mathscr{M}$ is stable, concatenating together the proofs of the positive IDB literals true in $\mathscr{M}$, plus a list of all negative IDB literals and all EDB literals true in $\mathscr{M}$, will give such an $\mathscr{H}$. On the other hand, if such a $\mathscr{H}$ exists, then for each $\mathscr{D}$-instantiated positive IDB literal in $\mathscr{M}$, some subsequence of $\mathscr{H}$ is a minimal forward proof of α from negative hypotheses which are true in $\mathscr{M}$—the only potentially difficult step is in proving that the purported forward proof is actually a *finite* sequence, but since $\mathscr{D}$ is finite, $\mathscr{H}$ must also be finite, so the purported forward proofs are finite.[14]

The other direction, that every co-$\mathscr{NP}$ relation is definable in the stable semantics, follows from Theorems 3.3 and 3.10. ▮

EXAMPLE 4.1. We illustrate half of the result above by constructing a program **P**, where an EDB $\mathscr{D}$ represents an instance of a formula in 3-CNF ϕ, and **P** has a stable model over $\mathscr{D}$ if and only if ϕ is satisfiable. The technique is analogous to that of Example 5.2 and of [KP88].

The objects are proposition letters. There are eight EDB relations, for truth values TTT through FFF: $\text{conj}_{TTF}(X, Y, Z)$ says that $X \vee Y \vee \neg Z$ is a conjunct of the 3-CNF formula.

$$\mathbf{P}_1 = \{isT(X) \leftarrow \neg isF(X),\ isF(X) \leftarrow \neg isT(X)\}$$

$$\mathbf{P}_2 = \{\ldots,\ a \leftarrow \text{conj}_{TTF}(X, Y, Z) \wedge isF(X) \wedge isF(Y) \wedge isT(Z),\ \ldots,\ b \leftarrow \neg a,\ b \leftarrow \neg b\}.$$

In any stable model of $\mathbf{P}_1$, the $isT(X)$ holds if and only if $isF(X)$ does not. Any such model is a stable model of $\mathbf{P}_1$. So $\mathbf{P}_1$ allows totally non-deterministic choice of isT, i.e., in terms of the 3-CNF formula, non-deterministic choice of which proposition letters are true.

The idea is that $\mathbf{P}_2$ should check whether the choice of true proposition letters made for $\mathbf{P}_1$ "works." The 3-CNF formula is satisfiable if and only if there is a stable model of $\mathbf{P}_1 \cup \mathbf{P}_2$. The final rule forces b to be true in any two-valued model without providing any negatively founded proof tree to use in deriving it. So in a stable model, a must be false. In any stable model a will be true if any conjunct of the original formula is false, and hence there is no way to derive b unless every conjunct is true. ▮

COROLLARY 4.3. *If a relation r on finite databases is uniformly definable in the unique stable model semantics, it is in $\mathscr{NP} \cap$ co-$\mathscr{NP}$.* ▮

4.2. *Expressive Power over Infinite Herbrand Universes*

As noted before, for logic programs with function symbols, the normal assumption is that the intended universe is the Herbrand universe of the functions and constants of the program. And even when that assumption is relaxed, the intended universe is normally taken to be the Herbrand universe for a larger set of constant and function symbols.

THEOREM 4.4. *Over the class of infinite Herbrand universes (generated by a positive, finite number of constant symbols and a positive, finite number of function symbols), the three-valued program completion semantics, the two-valued program completion semantics, the well-founded semantics, and the stable semantics all have the same expressive power.*

A relation r on an infinite Herbrand universe is definable in any of these semantics if and only if it is inductively definable over the Herbrand universe.

[14] If $\mathscr{D}$ were infinite, it would be possible to build infinite descending "proofs."

Proof. We already know that all four of the semantics are at least as strong as the three-valued program completion semantics and that the stable semantics is at least as strong as all four semantics. Hence it suffices to show that every inductively definable set is definable in the three-valued program completion semantics and that every set definable in the stable semantics is inductively definable. The former we already know, from Fitting's result, Theorem 3.4. So we prove the latter.

For the latter it suffices, by Theorem 2.1, to prove that every relation on the integers (i.e., in a program with one constant symbol 0 and one function symbol *succ*) definable in stable semantics is Π_1^1 definable on the integers. For any such **P**, all minimal forward proofs from negative hypotheses can be recursively coded with natural numbers—say by Gödel numbering. All obvious predicates about such forward proofs—that an integer p codes a minimal forward proof over the Herbrand universe, that n codes a negative instantiated literal which is an hypothesis of p, etc., can be expressed in first-order logic. Since quantification over forward proofs from negative hypotheses can then be replaced with quantification over integers, the statement that a given interpretation I is a two-valued fixed point of the operator **wf** is then first-order definable (in parameter I). Hence the statement that a literal is an element of all two-valued fixed points of **P** over its Herbrand universe is Π_1^1. ∎

Fitting proved essentially the above result for the three-valued program completion semantics. As he noted—and as Kunen [Kun87] also noted—that may be deemed a significant disadvantage of that semantics, and hence of all the semantics considered here in this paper, in that it proves that, over infinite Herbrand universes, the problem of deciding whether a tuple of elements of the Herbrand universe is a correct answer for a query is highly non-recursive.

Theorem 4.4, for the stable semantics, states that a relation is definable in stable semantics if and only if it is inducitively (Π_1^1) definable over the integers. This result was independently proved by Marek, Nerode, and Remmel [MNR92] (although for recursive logic programs instead of finite logic programs). In a series of papers, they developed a study, not just of the intersection of all stable models (over Herbrand universes) of recursive logic programs with function symbols, but of the classes of all such stable models; the Π_1^1 result is included. The following corollary is also an immediate consequence of one of their results.

THEOREM 4.5 [MNR92]. *A relation on the natural numbers is definable in the unique stable model semantics (for some recursive logic program) if and only if both it and its compolement are Π_1^1-definable subsets of the natural numbers.*

Thus, over infinite Herbrand models, the two- and three-valued program completion semantics, the well-founded semantics, and the stable semantics are all more expressive than the unique stable model semantics.

4.3. *Expressive Powers over Arbitrary Infinite Databases*

We now turn to the expressive powers of the well-founded and stable semantics over infinite EDBs which are not Herbrand universes. This is rather far afield from the usual concerns of logic programming. But for purposes of fully understanding the semantics, finite EDBs and Herbrand universes are too simple. In Subsection 5.2, we shall relate this topic back to expressive power over infinite Herbrand universes.

Recall that, over all infinite Herbrand universes, and uniformly over all finite EDBs, the well-founded semantics has the same expressive power as the three-valued program completion semantics, and the stable semantics has the same expressive power as the two-valued program completion semantics, although, as Theorem 3.8 and Example 3.1, part 4 show, for any particular logic program **P** the well-founded semantics and the stable semantics may make more inferences than the program completion semantics. A more general example is found with transitive closures.

THEOREM 4.6. *Suppose an EDB $\mathscr{D}$ contains a binary relation r (but not relations* tc *or* $\overline{\text{tc}}$*). Let* **P** *be the following program*:

$$\{\text{tc}(X, Y) \leftarrow r(X, Y),\ \text{tc}(X, Y) \leftarrow r(X, Z) \wedge \text{tc}(X, Y),$$
$$\overline{\text{tc}}(X, Y) \leftarrow \neg\text{tc}(X, Y)\}.$$

From **P** the well-founded, stable, and unique stable model semantics infer tc and $\overline{\text{tc}}$ to be the transitive closure of r and its complement.

The program completion semantics, on the other hand, infers tc to be the transitive closure of r, but if in $\mathscr{D}$ it is true that $\forall X \exists Y r(X, Y)$, then the program completion semantics will not infer $\overline{\text{tc}}(d_1, d_2)$ for any pairs d_1, d_2.

Proof. First consider the well-founded and stable semantics. The stable completion of **P** over D consists of a set of formulas equivalent to the set of EDB literals true in D plus the formulas, for $d_0, d_1 \in \mathscr{D}$,

$$\text{tc}(d_0, d_1) \rightarrow \bigvee_n \bigvee_{\substack{d_2, \ldots, d_n \in \mathscr{D} \\ r(d_0, d_2), r(d_2, d_3), \ldots, r(d_{n-1}, d_n), r(d_n, d_1) \text{ true in } \mathscr{D}}} \textbf{true}$$

$$\overline{\text{tc}}(d_0, d_1) \leftrightarrow \neg\text{tc}(d_0, d_1)$$

The assertion is obvious from examination of the stable completion.

The definition of each $\mathrm{tc}(d_0, d_1)$ in the completion of **P** over $\mathscr{D}$ is

$$\mathrm{tc}(d_0, d_1) \leftrightarrow r(d_0, d_1) \vee \bigvee_{d_2 \in \mathscr{D}} (r(d_0, d_2) \wedge \mathrm{tc}(d_2, d_1)).$$

It is straightforward to prove that both program completion semantics infer $\mathrm{tc}(d_0, d_1)$ if and only if the pair (d_0, d_1) is in the transitive closure of r. But observe that the partial interpretation I which interprets r as in $\mathscr{D}$ and contains $\mathrm{tc}(d_0, d_1)$ for all $d_0, d_1 \in \mathscr{D}$ is a model of the completion; hence the program completion semantics cannot infer $\neg\mathrm{tc}(d_0, d_1)$, or $\overline{\mathrm{tc}}(d_0, d_1)$, for any (d_0, d_1). ∎

Of course, since the complements of transitive closures are definable in the well-founded semantics, they are also definable in the program completion semantics both (1) uniformly over finite EDBs, and (2) over infinite Herbrand universes. It is just that the program to define the complement of a transitive closure is less natural than the program above (see, e.g., [Kun88]). However, over *general infinite* EDBs, the complement of a transitive closure may not be definable at all in the program completion semantics.

The remainder of this section is concerned with exotic EDBs. In the next section, however, we shall apply these results to make a finer distinction about the expressive powers of the other semantics over infinite Herbrand universes, the standard world of logic programming.

DEFINITION 4.1 [BS76, Sch78]. An EDB $\mathscr{D}$ is *recursively saturated* if, for each recursive set of formulas $\{\phi_i(X, \mathbf{Y}) : i \in \omega\}$ and for each **d** from $\mathscr{D}$, if for every $n \in \omega$ there is an $e \in \mathscr{D}$, where $\phi_0(e, \mathbf{d}) \wedge \phi_i(e, \mathbf{d}) \wedge \cdots \wedge \phi_n(e, \mathbf{d})$ is true in $\mathscr{D}$, then there is a single $e \in \mathscr{D}$ where every $\phi_i(e, \mathbf{d})$ is true in $\mathscr{D}$.

A basic fact about recursively saturated structures (again see [BS76, Sch78]) is that if $\mathscr{D}$ is recursively saturated, every inductive definition on $\mathscr{D}$ closes off in $\leqslant\omega$ steps. It follows from the Boundedness Theorem of [Mos74] that if a relation r on a recursively saturated EDB is both inductively and coinductively definable, it is first-order definable. Similarly, every relation on a countable recursively saturated structure which is both Π^1_1- and Σ^1_1-definable is first-order definable.

EXAMPLE 4.2. Consider the graph $\mathscr{G}$, whose *Edge* relation looks like the successor–predecessor relation on the natural numbers. It can be pictured:

$$N_0 \xleftrightarrow{\text{Edge}} N_1 \xleftrightarrow{\text{Edge}} N_2 \xleftrightarrow{\text{Edge}} N_3 \xleftrightarrow{\text{Edge}} \cdots.$$

There is (up to isomorphism) one countable, recursively saturated graph $\mathscr{G}^*$ which satisfies all the same first order sentences as $\mathscr{G}$. It contains copies of the elements N_0, N_1, N_2, etc., of $\mathscr{G}$ plus contains countably infinitely many disjoint chains

$$\cdots \xleftrightarrow{\text{Edge}} Z^i_{-2} \xleftrightarrow{\text{Edge}} Z^i_{-1} \xleftrightarrow{\text{Edge}} Z^i_0$$

$$\xleftrightarrow{\text{Edge}} Z^i_1 \xleftrightarrow{\text{Edge}} Z^i_2 \xleftrightarrow{\text{Edge}} \cdots.$$

As in Theorem 4.6, the transitive closure of the *Edge* relation is definable in the program completion semantics, and hence is inductively definable. Since $\mathscr{G}^*$ is recursively saturated and since the complement of the transitive closure of *Edge* is not first-order definable over $\mathscr{G}^*$, it is also not inductively definable; hence it is not definable in the three-valued program completion semantics. Similarly, the complement of the transitive closure of *Edge* is not Π^1_1-definable, so it is not definable in the two-valued program completion semantics.[15]

Conjecture 4.7. There is a relation on an EDB $\mathscr{D}$ which is definable in the well-founded semantics but not in the three-valued program completion semantics if and only if every inductive definition on $\mathscr{D}$ closes off in $\leqslant\omega$ stages and at least one closes off in exactly ω stages.

Our final result in this section, studying the expressive powers of the well-founded and stable semantics in fairly general circumstances, becomes involved in somewhat more technical methods from inductive definability theory than material discussed so far. What it does do is to emphasize rather strongly, as suggested in Example 4.2, the strong difference between these two semantics on the one hand and program completion semantics on the other. Definability in the well-founded and stable semantics can more generally be related to inductive definability, but to inductive definability over an in general somewhat richer structure than the original EDB.

DEFINITION 4.2 [Bar75]. Let $\mathscr{D}$ be an EDB. Consider the elements of $\mathscr{D}$(as anybody but a set-theorist would) as being, not sets, but indivisible objects, or *urelementen*. The set $\mathbf{HF}_{\mathscr{D}}$ is the smallest set where $\mathscr{D} \subseteq \mathbf{HF}_{\mathscr{D}}$ and, if $X_1, \ldots, X_n \in \mathbf{HF}_{\mathscr{D}}$, then $\{X_1, \ldots, X_n\} \in \mathbf{HF}_{\mathscr{D}}$. The EDB $\mathbf{HF}_{\mathscr{D}}$ has all the relations, functions, and constants of $\mathscr{D}$ itself, plus the set-theoretic relations *isAnUrelement* and $\in$.

[15] A saturation property—of a class of ultrapowers—was exploited by Kunen [Kun87] to characterize the derivations made in the first ω steps of the transfinite inductive construction of the least three-valued model of the program completion. Although it does not use an ultrapower, our example clearly is exploiting essentially the same point about the three-valued program completion semantics that Kunen made. Our contribution here is to use it to contrast with the well-founded and stable semantics.

THEOREM 4.8. *Let $\mathscr{D}$ be an infinite EDB and let r be a relation on $\mathscr{D}$. Then:*

1. *Relation r is definable over $\mathscr{D}$ in the stable semantics if and only if it is Π_1^1 definable over $\mathbf{HF}_{\mathscr{D}}$.*

2. *If $\mathscr{D}$ is countable, then r is definable over $\mathscr{D}$ in the stable semantics if and only if it is inductively definable over $\mathbf{HF}_{\mathscr{D}}$.*

3. *If r is definable over $\mathscr{D}$ in the well-founded semantics, it is inductively definable over $\mathbf{HF}_{\mathscr{D}}$.*

4. *If r is inductively definable over $\mathscr{D}$, it is definable over $\mathscr{D}$ in the well-founded semantics.*

5. *If there is a pairing function on $\mathscr{D}$ which is definable over $\mathscr{D}$ in the well-founded semantics, the converse to part 3 holds.*

6. *If a relation r is definable over $\mathscr{D}$ in the unique stable model semantics, both it and its complement are Π_1^1-definable over $\mathbf{HF}_{\mathscr{D}}$.*

Proof (Sketch). 1. Every minimal forward proof from negative hypotheses over $\mathscr{D}$ is an element of $\mathbf{HF}_{\mathscr{D}}$, so the definition of the stable semantics is easily seen to be Π_1^1 over $\mathbf{HF}_{\mathscr{D}}$, just as in the corresponding part of Theorem 4.4.

For the reverse direction, suppose a relation s on $\mathscr{D}$ is Π_1^1 definable on $\mathbf{HF}_{\mathscr{D}}$. We shall build a logic program in pieces. Piece $\mathbf{P}_0$ will "define" $=$; it contains the one rule $=(X, X)$.

Note that, since $\mathscr{D}$ is infinite, $\mathbf{HF}_{\mathscr{D}}$ has the same cardinality as $\mathscr{D}$. We shall nondeterministically "code" elements of $\mathbf{HF}_{\mathscr{D}}$ with elements of $\mathscr{D}$. Piece $\mathbf{P}_1$ will "choose" relations $r_f, r_1', \ldots, r_n', \in'$ and *isAnUrelement'* nondeterministically, as *isT* and *isF* are chosen in Example 4.1.

Piece $\mathbf{P}_2$ will "assert" that the relations chosen by $\mathbf{P}_1$ actually code $\mathbf{HF}_{\mathscr{D}}$—or some larger structure of sets with urelementen $\mathscr{D}$. It will "say" that r_f is the graph of a one-to-one function from $\mathscr{D}$ into itself, that *isAnUrelement'* is true of exactly the elements in the range of r_f, and that each r_i' is the isomorphic copy of the corresponding r_i under r_f. It will also "say" that $\in'$ obeys appropriate properties of the $\in$ relation—that nothing is an element of an unrelement, that two non-unrelementen are equal if they have the same elements, that for each object X, $\{X\}$ exists, and for each pair of sets X, Y, $X \cup Y$ exists. All this will be accomplished by writing formulas which have no stable models if any of these properties are violated. (Again, use the technique of Example 4.1.)

Piece $\mathbf{P}_3$ will "say" that the collection of sets chosen above is really isomorphic to $\mathbf{HF}_{\mathscr{D}}$: that it contains no infinite elements. The essential part here is that, since minimal forward proofs from negative hypotheses are finite, but come in arbitrarily large finite sizes, we can use them to "say" that each element of $\mathbf{HF}_{\mathscr{D}}$ really has only finitely many members; so $\mathbf{P}_3 =$

$$
\begin{aligned}
&\{OK(X) \leftarrow isAnUrelement'(X),\\
&\quad OK(X) \leftarrow OK(Y) \wedge \text{“}X = \{Y\}\text{”},\\
&\quad OK(X) \leftarrow OK(Y) \wedge \neg isAnUrelement'(Y)\\
&\qquad\qquad \wedge OK(Z) \wedge \neg isAnUrelement'(Z)\\
&\qquad\qquad \wedge \text{“}X = Y \cup Z\text{”},\\
&\quad tooBig \leftarrow \neg OK(X),\\
&\quad allOK \leftarrow \neg tooBig,\\
&\quad allOK \leftarrow \neg allOK\},
\end{aligned}
$$

where the first three rules put all (isomorphic copies of) sets in $\mathbf{HF}_{\mathscr{D}}$—and *no other elements*—into *OK*, and the last three rules are the usual trick to prevent there from being a stable model if any element is not in *OK*.

Now piece $\mathbf{P}_4$ will define a relation s', "containing" the Π_1^1 definition of s as we have done before, and also define s to be the isomorphic copy of s' under r_f. It is necessary to replace finitely many parameters in $\mathbf{HF}_{\mathscr{D}}$ with finitely many parameters from $\mathscr{D}$, but this can be done, with the aid of explicit information coded into the definition to distinguish the parameters, say, for $a, b \in \mathscr{D}$, $\{a, b\}$, $\{\{a, b\}\}$, and $\{\{a\}, \{b\}\}$.

For each particular stable model of $\mathbf{P}_1 \cup \mathbf{P}_2 \cup \mathbf{P}_3$, the intersection of all the interpretations of the s's over all stable models of $\mathbf{P}_1 \cup \cdots \cup \mathbf{P}_4$ will be the desired relation s. Hence the intersection of the s's over all stable models of $\mathbf{P}_1 \cup \cdots \cup \mathbf{P}_4$ for all stable models of $\mathbf{P}_1 \cup \mathbf{P}_2 \cup \mathbf{P}_3$ will also be the desired relations s. (It is true that not only does each piece $\mathbf{P}_i$ "do" what we claim when taken in isolation, but it also "does" what we claim when taken in the context of the other parts. We omit the proof of this fact.)

2. Since $\mathscr{D}$ is countable, so is $\mathbf{HF}_{\mathscr{D}}$. Also $\mathbf{HF}_{\mathscr{D}}$ contains a first-order definable pairing function, such as the one mapping X, Y to $\{\{X\}, \{X, Y\}\}$. So any relation on $\mathbf{HF}_{\mathscr{D}}$—and hence, in particular, any relation on $\mathscr{D}$—is Π_1^1-definable over $\mathbf{HF}_{\mathscr{D}}$ if and only if it is inductively definable over $\mathbf{HF}_{\mathscr{D}}$. By part 1, being Π_1^1-definable over $\mathbf{HF}_{\mathscr{D}}$ is the same as being definable in the stable semantics.

3. This is proved just as in the corresponding part of Theorem 4.4.

4. This is just a restatement of part of Corollary 3.11.

5. Given a relation s on $\mathscr{D}$ which is inductively definable over $\mathbf{HF}_{\mathscr{D}}$, we build a logic program to define it. Again, we build the program in pieces.[16] We sketch the details below. Piece $\mathbf{P}_0$ defines the pairing function.

Next, we use the pairing function to create, essentially, Gödel numbers for all the elements of $\mathbf{HF}_{\mathscr{D}}$ and to define

[16] As we shall show in the next section, this type of definition in pieces is unproblematical in the well-founded semantics.

TABLE I

Expressive power over	Semantics: Three-valued program compl.	Two-valued program compl.	Well-founded	Stable
Finite EDBs (uniformly)	Compl. inductive $\subseteq \mathscr{P}$	Co-$\mathscr{NP}$-complete	Compl. inductive $\subseteq \mathscr{P}$	Co-$\mathscr{NP}$-complete
Infinite Herbrand universes	Complete Π^1_1 over **N**	Complete Π^1_1 over **N**	Complete Π^1_1 over **N**	Complete Π^1_1 over **N**
General infinite EDBs $\mathscr{D}$	Compl. inductive over $\mathscr{D}$	Compl. Π^1_1 over $\mathscr{D}$	Inductive over $\mathbf{HF}_{\mathscr{D}}$	Compl. Π^1_1 over $\mathbf{HF}_{\mathscr{D}}$

all its relations on the Gödel numbers. We can write a program $\mathbf{P}_1$, the second piece, which is Horn over $\mathscr{D}$ plus the pairing function and defines all those relations and whose van Emden–Kowalski model is the intended model; by Theorem 3.8, the well-founded semantics captures that model.

Piece $\mathbf{P}_3$ inductively defines an isomorphic copy s' of s over $\mathbf{HF}_{\mathscr{D}}$—which is possible by Theorem 3.4. Program $\mathbf{P}_4$ uses s' and the pairing function to obtain the isomorphic copy of s, on $\mathscr{D}$, of s'.

6. Similar to the proof for Herbrand models. ∎

4.4. *Particular Expressive Power Summary*

This is outlined in Table I.

5. STRATIFICATION AND MODULARITY

In this section we return to a real-world programming concern: how programs can be built up out of modules. We shall discuss a property of programming language semantics, which we call the *Principle of Stratification*. In the first subsection we define the principle, discuss its real-world interest, and determine which of the semantics discussed in this paper satisfy the principle. In the second subsection we relate it back to definability issues. In particular, we use it to tie definability over certain exotic, general EDBs to definability over infinite Herbrand universes.

5.1. *The Principle of Stratification*

The Principle of Stratification asserts in part that a program can be built up, and understood, in natural pieces. It is perhaps most natural here to use Clark's intuition, that a program *defines* its IDB relations. Suppose, in writing a program to define a fairly complicated relation (say "in-law relative") it is easiest first to define another relation (say "relative") and then to *use that term* in defining the original term. It is natural to implement this as follows: first write a program $\mathbf{P}_1$ which defines the simpler term ("relative"); then write a second program $\mathbf{P}_2$ which *uses* the simpler term as an *EDB relation* in the definition of the original term ("in-law relative"). The Principle of Stratification asserts (1) that programs can be built up modularly this way, without concern for unexpected side effects: program $\mathbf{P}_2$, since it uses "relative" only as an EDB relation, will not affect the definition of "relative." Moreover, (2) if $\mathbf{P}_1$ is later replaced with a different program defining the same relation "relative," then the resultant definition of "in-law relative" will not be affected. These two requirements motivate the two parts of the Principle of Stratification (below).

The same motivation (1) can be given in terms of our paradigm of proofs by limited proof rules. For example, since "relative" is only used in $\mathbf{P}_2$, not defined, there is no proof, using only *modus ponens* and $\wedge$-introduction, of any assertion about "relative" which involves (non-trivially) any fact about "in-law relative." Hence $\mathbf{P}_2$ should have no effect on what is inferred about "relative."

This notion is captured strongly by the Principle of Stratification below. The principle is motivated by the stratified semantics for logic programming [CH85, VG86, ABW88, Lif88, Prz88], motivated strongly enough that we reuse the word "stratification." The principle is also motivated by an example of van Gelder, which we describe below. As in the stratified semantics, it is natural to look at a program as being built up out of many layers, or strata. We shall define the notion only for programs with two layers; it is trivial to generate more layers here by further subdivision.

DEFINITION 5.1. An ordered pair $\mathbf{P}_1, \mathbf{P}_2$ of programs is a *stratified pair* of programs if no IDB relation of $\mathbf{P}_2$ appears at all in $\mathbf{P}_1$. If $\mathbf{P}_1, \mathbf{P}_2$ is a stratified pair of programs, $\mathbf{P}_1$ and $\mathbf{P}_2$ are *strata* of the program $\mathbf{P}_1 \cup \mathbf{P}_2$. An EDB for a stratified pair $\mathbf{P}_1$, $\mathbf{P}_2$ is an EDB for $\mathbf{P}_1 \cup \mathbf{P}_2$.

DEFINITION 5.2. A logic programming semantics **S** obeys the *Principle of Stratification* for every stratified pair of programs $\mathbf{P}_1, \mathbf{P}_2$ and every EDB $\mathscr{D}$ for the pair,

1. if α is a $\mathscr{D}$-instantiated literal of $\mathbf{P}_1$; then semantics **S** infers α from program $\mathbf{P}_1$ and $\mathscr{D}$ if and only if it infers α from program $\mathbf{P}_1 \cup \mathbf{P}_2$ and $\mathscr{D}$.

2. If $\mathbf{Q}_1$ is any other logic program over $\mathscr{D}$ with the same EDB and IDB relations as $\mathbf{P}_1$, and if $\mathbf{S}$ infers the same literals from $\mathbf{Q}_1$ and $\mathscr{D}$ as it does from $\mathbf{P}_1$ and $\mathscr{D}$, then $\mathbf{S}$ infers the same literals from $\mathbf{Q}_1 \cup \mathbf{P}_2$ and $\mathscr{D}$ as it does from $\mathbf{P}_1 \cup \mathbf{P}_2$ and $\mathscr{D}$.

EXAMPLE 5.1. The equality relation $=$ is not normally assumed to be one of the relations of an EDB. But the program of Example 3.2 defines the intended interpretation of both $=$ and $\neq$ in many standard logic programming semantics, including the four semantics discussed in this paper. Thus, if a semantics obeys the Principle of Stratification, one may, without loss of generality, use $=$ and $\neq$ as if they were an EDB relations.

THEOREM 5.1. *The three-valued program completion and well-founded semantics satisfy the Principle of Stratification.*

Proof. The two proofs are somewhat analogous; we prove the harder, the result for the well-founded semantics:

1. Suppose $\mathbf{P}$, $\mathbf{P}_2$ is a stratified pair of programs, $\mathbf{P}' = \mathbf{P} \cup \mathbf{P}_2$, and $\mathscr{D}$ is an EDB for $\mathbf{P}$, $\mathbf{P}_2$. Recall the inductive operator $\mathbf{wf}$ used to construct the well-founded partial model by transfinite induction. Use $\mathbf{wf}_{\mathbf{P}}$ to denote the operator for program $\mathbf{P}$ (over EDB $\mathscr{D}$) and $\mathbf{wf}_{\mathbf{P}'}$ to denote the operator for $\mathbf{P}'$ (over $\mathscr{D}$). Let $\mathscr{L}$ be the set of instantiated literals of $\mathbf{P}$.

Observe that, for $\alpha \in \mathscr{L}$, if p is a minimal forward proof of α from negative hypotheses over $\mathbf{P}'$, then, by definition of $\mathbf{P}$, $\mathbf{P}_2$ being a stratified pair, only rules of $\mathbf{P}$ and literals in $\mathscr{L}$ may appear in p. Thus p is also a minimal forward proof of α from negative hypotheses over $\mathbf{P}$.

Prove by transfinite induction that, for each ordinal η, $\mathbf{wf}_{\mathbf{P}}^{\eta} = \mathbf{wf}_{\mathbf{P}'}^{\eta} \cap \mathscr{L}$. So assume the result for all $\nu < \eta$. For α a positive $\mathscr{D}$-instantiated literal of $\mathbf{P}$,

$\alpha \in \mathbf{wf}_{\mathbf{P}}^{\eta}$ iff for some forward proof p from negative hypotheses over $\mathbf{P}$ every hypothesis of p is in $\bigcup_{\nu<\eta} \mathbf{wf}_{\mathbf{P}}^{\nu}$

iff for some forward proof p from negative hypotheses over $\mathbf{P}'$ every hypothesis of p is in $\bigcup_{\nu<\eta} \mathbf{wf}_{\mathbf{P}'}^{\nu}$

iff $\alpha \in \mathbf{wf}_{\mathbf{P}'}^{\eta}$

$\neg\alpha \in \mathbf{wf}_{\mathbf{P}}^{\eta}$ iff for every forward proof p from negative hypotheses over $\mathbf{P}$ some hypothesis of p is false in $\bigcup_{\nu<\eta} \mathbf{wf}_{\mathbf{P}}^{\nu}$

iff for every forward proof p from negative hypotheses over $\mathbf{P}'$ some hypothesis of p is false in $\bigcup_{\nu<\eta} \mathbf{wf}_{\mathbf{P}'}^{\nu}$

iff $\neg\alpha \in \mathbf{wf}_{\mathbf{P}'}^{\eta}$.

2. Let $\mathbf{P}$, $\mathbf{P}_2$, and $\mathbf{Q}$ be as in the statement of Principle of Stratification. Let $\mathbf{P}' = \mathbf{P} \cup \mathbf{P}_2$ and $\mathbf{Q}' = \mathbf{Q} \cup \mathbf{P}_2$. Let $\mathscr{D}$ be an EDB for $\mathbf{P}'$—and hence also an EDB for $\mathbf{Q}'$. We have by the assumptions plus part 1 that, for any $\mathscr{D}$-instantiated literal α of $\mathbf{P}$, α is inferred by the well-founded semantics from $\mathbf{P}'$ and $\mathscr{D}$ if and only if α is inferred by the well-founded semantics from $\mathbf{Q}'$ and $\mathscr{D}$. Thus it is only necessary to prove the result for the IDB literals of $\mathbf{P}_2$. The proof matches the motivation: $\mathbf{P}_2$ uses the relations defined in $\mathbf{P}$ or $\mathbf{Q}$. In any proof, or purported proof, of a literal of $\mathbf{P}_2$ which incorporates subproofs, or purported proofs, of literals of $\mathbf{P}$, it is possible to replace the subproofs with subproofs over $\mathbf{Q}$. This "cutting and pasting" of proofs is the heart of the proof below.

Suppose α is a $\mathscr{D}$-instantiated positive IDB literal of $\mathbf{P}_2$ and suppose $p_{\mathbf{P}}$ is a *minimal* forward proof of α from negative hypotheses over $\mathbf{P}'$ and $\mathscr{D}$. Furthermore, suppose no hypothesis of $p_{\mathbf{P}}$ is inferred false by the well-founded semantics over $\mathbf{P}$ and $\mathscr{D}$.

Suppose β is a positive $\mathscr{D}$-instantiated IDB literal of $\mathbf{P}$ which is a step of $p_{\mathbf{P}}$. Note that the well-founded semantics cannot infer $\neg\beta$ over $\mathbf{P}$ and $\mathscr{D}$, since some subsequence p_β of $p_{\mathbf{P}}$ is a forward proof of β from negative hypotheses over $\mathbf{P}$, where no hypothesis has been inferred false by the well-founded semantics over $\mathbf{P}$ and $\mathscr{D}$. Similarly, if every hypothesis of p_β is inferred true by the well-founded semantics over $\mathbf{P}$ and $\mathscr{D}$, then the well-founded semantics also infers β.

Construct a minimal forward proof $p_{\mathbf{Q}}$ of α from negative hypotheses over $\mathbf{Q}'$ and $\mathscr{D}$, by "cutting and pasting" forward proofs, as follows:

(a) Remove from $p_{\mathbf{P}}$ each step which is an instance of a rule of $\mathbf{P}$.

(b) For each positive $\mathscr{D}$-instantiated IDB literal β of $\mathbf{P}$ which is a step of $p_{\mathbf{P}}$, determine whether the well-founded semantics over $\mathbf{P}$ and $\mathscr{D}$ infers β. If β is inferred, then the well-founded semantics also infers β over $\mathbf{Q}$ and $\mathscr{D}$. So there is a forward proof p_β of β over $\mathbf{Q}$, where each hypothesis is inferred by the well-founded semantics over $\mathbf{Q}$ and $\mathscr{D}$. Insert any such proof p_β ito $p_{\mathbf{P}}$ immediately in front of step β.

If β is not inferred true, then, by the remark above, at least it is not inferred false. Hence it is also not inferred false by the well-founded semantics over $\mathbf{Q}$ and $\mathscr{D}$. So there is a forward proof p_β of β over $\mathbf{Q}$ where no hypothesis is inferred false by the well-founded semantics over $\mathbf{Q}$ and $\mathscr{D}$. Insert any such proof p_β into $p_{\mathbf{P}}$ immediately in front of step β.

(c) The result of this "cutting and pasting" is a forward proof of α from negative hypotheses over $\mathbf{Q}$ and $\mathscr{D}$. Some subsequence of it is hence a minimal forward proof of α from negative hypotheses over $\mathbf{Q}$ and $\mathscr{D}$. Some subsequence of it is hence a

minimal forward proof of α from negative hypotheses over $\mathbf{Q}$ and $\mathscr{D}$. Pick one such minimal forward proof and call it $p_{\mathbf{Q}}$.

Similarly, given such a forward proof $p_{\mathbf{Q}}$ over $\mathbf{Q}$ and $\mathscr{D}$, define a forward proof $p_{\mathbf{P}}$ over $\mathbf{P}$ and $\mathscr{D}$. Now prove, by transfinite induction on η, that

$$\mathbf{wf}^{\eta}_{\mathbf{P}'} \subseteq \mathbf{wf}^{\infty}_{\mathbf{Q}'} \quad \text{and} \quad \mathbf{wf}^{\eta}_{\mathbf{Q}'} \subseteq \mathbf{wf}^{\infty}_{\mathbf{P}'}.$$

We prove the first; the second is analogous. Assume that $\bigcup_{\nu<\eta} \mathbf{wf}^{\eta}_{\mathbf{P}'} \subseteq \mathbf{wf}^{\infty}_{\mathbf{Q}'}$.

For positive IDB literal α of $\mathbf{P}_2$, suppose that $\alpha \in \mathbf{wf}^{\eta}_{\mathbf{P}'}$. So there is a forward proof $p_{\mathbf{P}}$ of α from negative hypotheses over $\mathbf{P}'$, all of whose hypotheses are in $\bigcup_{\nu<\eta} (\mathbf{wf}_{\mathbf{P}'})^{\nu}$. Then $p_{\mathbf{Q}}$ is a forward proof of α from negative hypotheses over $\mathbf{Q}'$, and all of its hypothesis are in either $\mathbf{wf}^{\infty}_{\mathbf{Q}}$ or $\bigcup_{\nu<\eta} \mathbf{wf}^{\eta}_{\mathbf{P}'}$. Hence $\alpha \in \mathbf{wf}^{\infty}_{\mathbf{Q}'}$.

Suppose $\neg\alpha \in \mathbf{wf}^{\eta}_{\mathbf{P}'}$. We must show that $\neg\alpha \in \mathbf{wf}^{\infty}_{\mathbf{Q}'}$. Suppose not. Then there is a forward proof $p_{\mathbf{Q}}$ of α from negative hypotheses over $\mathbf{Q}$ and $\mathscr{D}$, none of whose hypotheses are false in $\mathbf{wf}^{\infty}_{\mathbf{Q}'}$. Construct proof $p_{\mathbf{P}}$ as above. Its hypotheses are all either (1) negative literals of $\mathscr{L}$ not inferred false by the well-founded semantics over $\mathbf{Q}$ and $\mathscr{D}$, and thus over $\mathbf{P}$ and $\mathscr{D}$, or (2) negative literals of $p_{\mathbf{Q}}$. Since, by inductive hypothesis, $\bigcup_{\nu<\eta} \mathbf{wf}^{\eta}_{\mathbf{P}'} \subseteq \mathbf{wf}^{\infty}_{\mathbf{Q}'}$, none of the negative literals of $p_{\mathbf{P}}$ are false in $\bigcup_{\nu<\eta} \mathbf{wf}^{\eta}_{\mathbf{P}'}$. This contradicts the assumption that $\neg\alpha \in \mathbf{wf}^{\eta}_{\mathbf{P}'}$. ∎

EXAMPLE 5.2. The two-valued program completion and stable semantics do not satisfy the Principle of Stratification. We give three examples that work for both semantics. For simplicity of presentation, we use propositional logic examples.

1. Let $\mathbf{P}_1$ be $\{a \leftarrow \neg b\}$ and $\mathbf{P}_2$ be $\{c \leftrightarrow \neg c\}$. The completion and stable completion of $\mathbf{P}_1$ are both $\{a \leftrightarrow \neg b, b \leftrightarrow \mathbf{false}\}$. From $\mathbf{P}_1$ both semantics infer $\{a, \neg b\}$.

The completion and stable completion of $\mathbf{P}_1 \cup \mathbf{P}_2$ are both

$$\{a \leftrightarrow \neg b, b \leftrightarrow \mathbf{false}, c \leftrightarrow \neg c\}.$$

This completion is inconsistent; i.e., it has no two-valued model. Hence from $\mathbf{P}_1 \cup \mathbf{P}_2$ both semantics infer $\{a, \neg a, b, \neg b, c, \neg c\}$. This violates the first part of the Principle of Stratification.

2. This is an example of Van Gelder [VGRS91], broken into two pieces:

$$\mathbf{P}_1 = \{a \leftarrow \neg b, b \leftarrow \neg a\}, \quad \mathbf{P}_2 = \{p \leftarrow \neg p, p \leftarrow \neg a\}.$$

The completion and stable completion of $\mathbf{P}_1$ are both $\{a \leftrightarrow \neg b, b \leftrightarrow \neg a\}$, which has two models, $\{a, \neg b\}$ and $\{\neg a, b\}$. So from $\mathbf{P}_1$ the two semantics infer nothing. But the completion and stable completion of $\mathbf{P}_1 \cup \mathbf{P}_2$ are

$$\{a \leftrightarrow \neg b, b \leftrightarrow \neg a, p \leftrightarrow \neg p \vee \neg a\},$$

which has a unique model, $\{a, \neg b, p\}$. Essentially, the rule $p \leftarrow \neg p$ forces p to be true in all two-valued models without giving any way to make a forward proof of p from negative hypotheses. Hence $\neg a$ must be true in any model of the completion in order for there be a forward proof of p. So from $\mathbf{P}_1 \cup \mathbf{P}_2$ both semantics infer $\{\neg a, b, p\}$. This is a counterexample to the first part of the Principle of Stratification. It gives an abductive inference, which does not match the *modus ponens*-only paradigm at the root of some views of logic programming.

3. The two-valued program completion semantics and the stable semantics also do not obey the second part of the Principle of Stratification. Let

$$\mathbf{P} = \{a \leftarrow \neg b, b \leftarrow \neg a, c \leftarrow \neg d, d \leftarrow \neg c\}$$

$$\mathbf{Q} = \{a \leftarrow \neg d, d \leftarrow \neg a, c \leftarrow \neg b, b \leftarrow \neg c\}$$

$$\mathbf{P}_2 = \{e \leftarrow a, e \leftarrow b\}.$$

From $\mathbf{P}$ and $\mathbf{Q}$ the two semantics both infer noting. But from $\mathbf{P} \cup \mathbf{P}_2$ they both infer e, while from $\mathbf{Q} \cup \mathbf{P}_2$ they both infer noting.

This example shows the second half of the Principle of Stratification to be, in part, a requirement of constructivity. Here the two-valued program completion semantics and the stable semantics "infer" $a \vee b$ from $\mathbf{P}$, and that gives e from $\mathbf{P}_2$ by reasoning by cases. The three-valued semantics never "infer" such a disjunction unless they infer one of the disjunctis; in this they resemble more constructive logics. In ordinary logic such constructivity is a controversial property of, for example, intuitionistic logic. In this case it is, of course, arguable whether the constructivity is a bug or a feature, but it is our opinion that some sort of constructivity is appropriate to be consistent with the limitation of proof rules (to *modus ponens* and $\wedge$-introduction in our discussion) frequently made in negation-as-failure paradigms.

This constructivity in the second part of the Principle of Stratification is an artifact of the definition of a semantics. We defined a semantics to assign a set of ground literals to a program, not, for example, a set of sentances or a set of two-valued models. Obviously, one could also consider the Principle of Stratification under such variants. Also, we find it reasonable that one might require only the first part of the Principle of Stratification of a programming language semantics. Elsewhere [Sch92] we have referred to the first requirement of the Principle of Stratification alone as the Weak Principle of Stratification. There we also suggested a logic programming semantics which obeys the *weak principle* but not the entire Principle of Stratification.

THEOREM 5.2. *The unique stable model semantics satisfies the principle of stratification, in the following sense:*

1. *Suppose* $\mathbf{P}_1, \mathbf{P}_2$ *is a stratified pair programs and, over an EDB* $\mathscr{D}$, *both* $\mathbf{P}_1$ *and* $\mathbf{P}_1 \cup \mathbf{P}_2$ *have unique stable models. If* α *is a* $\mathscr{D}$*-instantiated IDB literal of* $\mathbf{P}_1$, *then the unique stable model semantics infers* α *from program* $\mathbf{P}_1$ *and* $\mathscr{D}$ *if and only if it infers* α *from program* $\mathbf{P}_1 \cup \mathbf{P}_2$ *and* $\mathscr{D}$.

2. *Suppose* $\mathbf{P}_1, \mathbf{Q}_1$ *are logic programs with the same EDB and IDB relations, and* $\mathbf{P}_1, \mathbf{P}_2$ *is a stratified pair of programs. Suppose further that* $\mathbf{P}_1$, $\mathbf{Q}_1$ *and* $\mathbf{P}_1 \cup \mathbf{P}_2$ *all have unique stable models over EDB* $\mathscr{D}$ *and that the stable models for* $\mathbf{P}_1$ *and for* $\mathbf{Q}_1$ *are the same. Then* $\mathbf{Q}_1 \cup \mathbf{P}_2$ *also has a unique stable model over* $\mathscr{D}$, *and the stable model is the same as the unique stable for* $\mathbf{P}_1 \cup \mathbf{P}_2$ *over* $\mathscr{D}$.

Proof. 1. Suppose $\mathbf{P}_1, \mathbf{P}_2$ is a stratified pair and $\mathscr{D}$ is an EDB for the pair. For each positive $\mathscr{D}$-instantiated α of $\mathbf{P}_1$, the definition of α in the stable completion of $\mathbf{P}_1$ is the same as the definition of α in the stable completion of $\mathbf{P}_1 \cup \mathbf{P}_2$. Hence the restriction of a stable model I of $\mathbf{P}_1 \cup \mathbf{P}_2$ to the literals of $\mathbf{P}_1$ is a stable model of $\mathbf{P}_1$. So if I is a stable model of $\mathbf{P}_1 \cup \mathbf{P}_2$ over $\mathscr{D}$, and if J is the unique stable model of $\mathbf{P}_1$ over $\mathscr{D}$, then J must be the restriction of I to the literals of $\mathbf{P}_1$.

2. We omit the proof of the second half. The proof is rather similar to the proof of the second half of the Principle of Stratification for the well-founded semantics.

The method of Example 5.1 can be used to define $=$ in the stable and two-valued program completion semantics, despite the fact that these semantics do not satisfy the Principle of Stratification. This is due to the fact that, for each EDB $\mathscr{D}$, there is a *unique* model of the completion and stable completion of the rule defining $=$.

5.2. *Uniform Translation over Strata*

The results of Subsection 4.3 on the expressive power of logic programming semantics over infinite non-Herbrand EDBs seem far afield from the oridinary con-cerns of logic programming. Here, using the notion of stratified pairs of programs, we use them to prove results concerning logic programming over infinite Herbrand universes. The results concern translations between two different semantics.

Over Herbrand universes we have shown that the three-valued program completion and well-founded semantics have equal expressive power. That is, given a logic program $\mathbf{P}$ defining a relation r on an Herbrand universe $\mathscr{D}$ in one of the semantics, one can construct a logic program $\mathbf{P}'$ defining the same relation r on $\mathscr{D}$ in the other. Think of $\mathbf{P}'$ as a translation of $\mathbf{P}$. Now suppose one starts with a stratified pair $\mathbf{P}_1, \mathbf{P}_2$. Is it possible to do the translation so that $\mathbf{P}_1$ is used to construct $\mathbf{P}_1'$, $\mathbf{P}_2$ is used to construct $\mathbf{P}_2'$, and *in translating* $\mathbf{P}_2$ *we never need to look to see what* $\mathbf{P}_1$ *is*—and similarly, in translating $\mathbf{P}_1$, we never need to look at what $\mathbf{P}_2$ is? We shall say a translation where one can translate one stratum at a time is *uniform in the strata*. The intuition for the Principle of Stratification suggests that each stratum is an independent module. We are asking here for a translation which respects this independence, rather like requiring a compiler to be able to compile separate modules separately. We propose this as necessary for the translation to be considered "natural."

DEFINITION 5.3. Let $\mathbf{S}, \mathbf{S}'$ be two logic programming semantics. An *Herbrand translation function* from $\mathbf{S}$ to $\mathbf{S}'$ is a function mapping each program $\mathbf{P}$ to a program $\mathbf{P}'$, built from the same constant, function, and relation symbols as $\mathbf{P}$ plus possibly extra relation symbols, where, if U is the Herbrand universe for $\mathbf{P}$, for any U-instantiated literal α of $\mathbf{P}$, semantics $\mathbf{S}$ infers α from $\mathbf{P}$ and U if and only if semantics $\mathbf{S}'$ infers α from $\mathbf{P}'$ and U.

Theorem 4.4 implies that there are Herbrand translations functions from each of the three-valued program completion semantics, the two-valued program completions semantics, the well-founded semantics, and the stable semantics to each of the others, for all programs containing at least one constant symbol and one function symbol. (The translatability is, in fact, trivial for programs with no function symbols.)

DEFINITION 5.4. An Herbrand translation function, mapping program each program $\mathbf{P}$ to a program $\mathbf{P}'$, *translates uniformly by strata* if the following holds: Suppose $\mathbf{P}_1, \mathbf{P}_2$ is a stratified pair of logic programs and suppose that each function or constant symbol appearing in $\mathbf{P}_2$ also appears in $\mathbf{P}_1$.[17] Then $(\mathbf{P}_1 \cup \mathbf{P}_2)' = \mathbf{P}_1' \cup \mathbf{P}_2'$.

THEOREM 5.3. 1. *There is an Herbrand translation function from the three-valued program completion semantics to be well-founded semantics that translates uniformly by strata.*

2. *There is no Herbrand translation function from the well-founded semantics to the three-valued program completion semantics that translates uniformly by strata.*

Proof. 1. The double-negative construction of Theorem 3.10 suffices.

2. Choose $\mathbf{P}_1$ to have an IDB relation, *Edge*, so that, for its natural Herbrand universe, it defines the graph $\mathscr{G}^*$ of Example 4.2. (Such a $\mathbf{P}_1$ is easy to construct.) Then suppose that $\mathbf{P}_2$ is the program of Theorem 4.6. In the well-founded semantics, $\overline{\text{tc}}$ obtains its intended meaning. But since $\mathscr{G}^*$ is recursively saturated, no program with just the EDB relation *Edge* can define the intended interpretation of $\overline{\text{tc}}$ in the

[17] This is just a technicality; otherwise the Herbrand universes for the two programs would be different, and the statement of the Principle of Stratification would not apply.

three-valued program completion semantics. Hence the translation of $\mathbf{P}_2$ must use one of the original functions of the Herbrand universe (or some symbol of $\mathbf{P}_1$ other than *Edge*), none of which appear in $\mathbf{P}_2$. So in translating $\mathbf{P}_2$ one must look back at $\mathbf{P}_1$ to see which function symbols appeared in it. ∎

These results can be interpreted to mean that, although the three-valued program completion semantics and the well-founded semantics have equivalent expressive power over infinite Herbrand models, in this notion of *uniform* translatability, the well-founded semantics is more expressive in natural ways.[18]

ACKNOWLEDGMENTS

I am indebted to Phokion Kolaitis, Wiktor Marek, Kenneth Ross, and Allen Van Gelder for helpful discussions. I am indebted to Allen Van Gelder and to the anonymous referees for many suggestions and corrections to earlier versions of the manuscript.

REFERENCES

[AB90] H. R. Apt and H. Blair, Arithmetic classification of perfect models of stratified programs, *Fund. Inform.* **13**, No. 1 (1990), 1–71.

[ABW88] K. R. Apt, H. Blair, and A. Walker, Towards a theory of declarative knowledge, *in* "Foundations of Deductive Databases and Logic Programming," (J. Minker, Ed.), pp. 89–148, Morgan Kaufmann, Los Altos, CA, 1988.

[Acz77] P. Aczel, An introduction to inductive definitions, *in* "Handbook of Mathematical Logic" (J. Barwise, Ed.), pp. 739–782, North-Holland, New York, 1977.

[AU79] A. V. Aho and J. D. Ullman, Universality of data retrieval languages, *in* "6th ACM Symp. on Principles of Programming Languages, 1979," pp. 110–120.

[Bar75] J. Barwise, "Admissible Sets and Structures," Springer-Verlag, New York, 1975.

[Bry89] F. Bry, Logic programming as constructivism: A formalization and its application to databases, *in* "Eighth ACM Symposium on Principles of Database Systems, 1989," pp. 34–50.

[BS76] J. Barwise and J. Schlipf, An introduction to recursively saturated and respendent models, *J. Symbolic Logic* **41**, No. 2 (1976), 531–536.

[BSu91] C. Baral and V. S. Subrahmanian, Dualities between alternative semantics for logic programming and non-monotonic reasoning, *in* "Logic Programming and Non-monotonic reasoning: Proceedings of the First International Workshop" (A. Nerode, W. Marek, and V. S. Subramabrian, Eds.), MIT Press, Cambridge, MA, 1991.

[CH82] A. Chandra and D. Harel, Structure and complexity of relational queries, *J. Comput. System Sci.* **25**, No. 1 (1982), 99–128.

[CH85] A. Chandra and D. Harel, Horn clause queries and generalizations, *J. Assoc. Comput. Mach.* **29**, No. 3 (1982), 841–862.

[18] This result is somewhat analogous to the result of [Kun88], that the complement of a transitive closure cannot be defined in certain circumstances in the three-valued program completion semantics by a *strict* program over all *finite* EDBs. It can be defined in the well-founded semantics by a strict program.

[Cla78] K. L. Clark, Negation as failure, *in* "Logic and Databases" (Gallaire and Minker, Eds.), pp. 293–322, Plenum, New York, 1978.

[Dav80] M. Davis, The mathematics of non-monotonic reasoning, *Artif. Intell.* **28** (1980), 73–80.

[Fa74] R. Fagin, Generalized first-order spectra and polynomial-time recognizable sets, *in* "Complexity of Computation" (Karp, Ed.), pp. 43–74, Amer. Math. Soc., Providence, RI, 1974.

[Fit85] M. Fitting, A Kripke–Kleene semantics for logic programs, *J. Logic Programming* **2**, No. 4 (1985), 295–312.

[Fit91] M. Fitting, Approximation logics and well-founded semantics, Invited talk at the "First International Workshop on Logic Programming and Non-monotonic Reasonining, Washington, DC, July 1991."

[Gel87] M. Gelfond, On stratified autoepistemic theories, *in* "Proc. AAAI, 1987."

[GL88] M. Gelfond and V. Lifschitz, The stable model semantics for logic programming, *in* "Proceedings, 5th Int. Conf. Symp. on Logic Programming, 1988."

[GS86] Y. Gurevich and S. Shelah, Fixed-point extensions of first order logic, *Ann. Pure Appl. Logic* **32** (1986), 265–280.

[Imm86] N. Immerman, Relational queries computable in polynomial time, *Inform. and Control* **68**, No. 1 (1986), 86–104.

[KP88] P. Kolaitis and C. Papadimitriou, Why not negation by fixpoint? *in* "Proceedings, Seventh ACM Symposium on Principles of Database Systems, 1988," pp. 231–239.

[Kun87] K. Kunen, Negation in logic programming, *J. Logic Programming* **4**, No. 4 (1987), 289–308.

[Kun88] K. Kunen, "Some Remarks on the Completed Database," Technical Report 775, University of Wisconsin, Madison.

[Lif88] V. Lifschitz, On the declarative semantics of logic programs with negation, *in* "Foundations of Deductive Databases and Logice Programming" (J. Minker, Ed.), pp. 177–192, Morgan Kaufmann, Los Altos, CA, 1988.

[McC80] J. McCarthy, Circumscription—A form of non-monotonic reasoning, *Artif. Intell.* **13** (1980), 27–39.

[Min82] J. Minker, On indefinite databases and the closed world assumption, "Sixth Conference on Automated Deduction," pp. 292–308, Springer-Verlag, New York, 1982.

[Mos74] Y. N. Moschovakis, "Elementary Induction on Abstract Structures," North-Holland, New York, 1974.

[MNR92] W. Marek, A. Nerode, and J. Remmel, How complicated is the set of stable models of a recursive logic program?, *Ann. Pure Appl. Logic* **56** (1992), 119–135.

[MT91] W. Marek and M. Truszczynski, Autoepistemic logic, *J. Assoc. Comput. Mach.* **38**, No. 3 (1991), 588–619.

[Prz88] T. C. Przymusinski, On the declarative semantics of deductive databases and logic programs, *in* "Foundations of Deductive Databases and Logic Programming" (J. Minker, Ed.), pp. 193–216, Morgan Kaufmann, Los Altos, CA, 1988.

[Prz89] 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, 1989," pp. 11–21.

[Ros89] K. A. Ross, A procedural semantics for well-founded negation in logic programs, *in* "Eight ACM Symposium on Principles of Database Systems, 1989," pp. 22–33.

[Sch78] J. Schlipf, Toward model theory through recursive saturation, *J. Symbolic Logic* **43** (1978), 183–206.

[Sch87] J. Schlipf, Decidability and definability with circumscription, *Ann. Pure Appl. Logic* **35** (1987), 173–191.

[Sch91] J. Schlipf, Representing epistemic intervals in logic programming, *in* "Logic Programming and Non-monotonic

Reasoning: Proceedings of the First International Workshop" (A. Nerode, W. Marek, and V. S. Subramahnian, Eds.), MIT Press, Cambridge, MA, 1991.

[Sch92] J. Schlipf, Formalizing a logic for logic programming, *An. Math. Artif. Intell.* **5** (1992), 279–302.

[She85] J. C. Shepherdson, Negation as failure, II, *J. Logic Programming* **2**, No. 3 (1985), 185–202.

[She88] J. C. Shepherdson, Negation in logic programming, *in* "Foundations of Deductive Database and Logic Programming" (J. Minker, Ed.), pp. 19–88, Morgan Kaufmann, Los Altos, CA, 1988.

[Var82] M. Vardi, The complexity of relational query languages, *in* "14th ACM Symposium on Theory of Computing, 1982," pp. 137–145.

[VEK76] M. H. Van Emden and R. A. Kowalski, The semantics of predicate logic as a programming language, *J. Assoc. Comput. Mach.* **23**, No. 4 (1976), 733–742.

[VG86] A. Van Gelder, Negation as failure using tight derivations for general logic programs, *in* "Proceedings, Third IEEE Symposium on Logic Programming, Salt Lake City, Utah," Springer-Verlag, New York, 1986.

[VG93] A. Van Gelder, The alternating fixpoint of logic programs with negation, *J. Comput. System Sci.* **47** (1993); preliminary abstract in "Eighth ACM Symposium on Principles of Database Systems, 1989," pp. 1–10.

[VGRS91] A. Van Gelder, K. A. Ross, and J. S. Schlipf, The well-founded semantics for general logic programs, *J. Assoc. Comput. Mach.* **38**, No. 3 (1991), 620–650.