The Alternating Fixpoint of Logic Programs with Negation*

Allen Van Gelder

Baskin Computer Science Center
University of California
Santa Cruz, CA 95064

February 7, 1995

Abstract

The *alternating fixpoint* of a logic program with negation is defined constructively. The underlying idea is monotonically to build up a set of *negative* conclusions until the least fixpoint is reached, using a transformation related to the one that defines stable models. From a fixed set of negative conclusions, the positive conclusions follow (without deriving any further negative ones), by traditional Horn clause semantics. The union of positive and negative conclusions is called the *alternating fixpoint partial model*. The name "alternating" was chosen because the transformation runs in two passes; the first pass transforms an underestimate of the set of negative conclusions into an (intermediate) overestimate; the second pass transforms the overestimate into a new underestimate; the composition of the two passes is monotonic.

The principal contributions of this work are (1) that the alternating fixpoint partial model is identical to the well-founded partial model, and (2) that alternating fixpoint logic is at least as expressive as fixpoint logic on all structures. Also, on finite structures, fixpoint logic is as expressive as alternating fixpoint logic.

Desk Editor: Suggested running title: Alternating Fixpoint of Logic Programs.

Figures appearing in text are also on separate pages at end.

*This version appeared in *Journal of Computer and System Sciences*, 47(1), pp. 185–221, 1993. Preliminary version appeared as an extended abstract in "Proceedings, 8th ACM Symposium on Principles of Database Systems", Philadelphia, March 1989.

1 Introduction

Horn clause programs have an intuitive and well accepted semantics defined by the least fixpoint of an operator that essentially performs *modus ponens* reasoning. Several early attempts to extend this operator to programs with negative subgoals ran into problems of one sort or another. Two recent proposals to improve matters are named "stable models", due to Gelfond and Lifschitz [18], and "well-founded partial models", due to Van Gelder, Ross, and Schlipf [51]. Both stable models and well-founded partial models were defined somewhat nonconstructively, in the sense that certain sets could be recognized as models if presented, but no algorithm to construct them from the program was given. This paper addresses that problem.

1.1 Logic Programs

A logic program with negation is a set of rules, some of which have *negative subgoals*. A rule specifies that a certain goal (the *head* of the rule) can be solved (or proved) if zero or more subgoals (the rule *body*) can be solved . However, a *negative* subgoal is considered solved just when, in some sense, its positive version cannot be solved. For example, the rule (from Example 8.2)

$$u(X) \leftarrow e(Y, X), \neg w(Y).$$

is read as follows: One way to solve $u(X)$ is to find a Y such that $e(Y, X)$ can be solved and $w(Y)$ cannot be solved.

Salient points about the rule syntax shown here are: (1) "$\leftarrow$" is read as "if"; (2) commas separating subgoals denote conjunction; (3) logical variables begin with capital letters; and (4) variables appearing only in the rule body are implicitly existentially quantified. Adopting terminology from logic, the goals, such as $u(X)$, $e(Y, X)$, and $w(Y)$, are called *atoms* (short for *atomic formulas*); literals denote either atoms (such as $e(Y, X)$) or negated atoms (such as $\neg w(Y)$).

An especially trivial form of a rule is one whose body contains no subgoals, and whose head contains no variables, e.g., $e(1, 2)$. We call such a rule a *fact*. Tuples in a relational database would be regarded as facts in a logic program.

1.2 Negation as Failure

Logic programs do not permit negative literals in the heads of rules, so the question arises, what does it mean to "solve" a negative subgoal? The standard treatment, as mentioned above, is to say that a negative subgoal is considered solved when, in some appropriate sense, its positive version cannot be proved. This natural treatment of negation as *failure to prove* goes back to Clark [11], and to the Closed World Assumption of Reiter [39]. It works quite smoothly in the case that the rules are not recursive, as proof possibilities can be explored exhaustively.

Substantial research has been directed at the surprisingly thorny problems surrounding negation in recursive rules (see Section 2). The right notion of "failure to prove" is not immediately clear, and several definitions have been investigated. One approach that has gained recent popularity is called the *stratified* semantics. It applies to the class of programs in which negation itself is not recursive; i.e., if relation p depends negatively on q, there is no chain of dependencies from q back to p (q need not be distinct from p). Recent research has investigated ways to extend the stratified semantics to cover (some or all) programs that do contain recursive negation. Two recent proposals, closely related to each other, define *stable models* [18] and *well-founded partial models* [51]. This paper extends this line of research, showing the following:

1. The alternating fixpoint is a constructive characterization of the well-founded semantics.
2. One of the operators upon which the alternating fixpoint is based can be used to define stable models.
3. The alternating fixpoint provides at least the expressive power of fixpoint logic [34] uniformly on all structures. On finite structures they are equally expressive.

4. The alternating fixpoint interpretation extends naturally to programs with first order rule bodies; such an extension of well-founded semantics is not obvious.

These results are described in more detail in Section 2. A preliminary version of this work has been presented in conference [49].

1.3 Organization

Section 2 provides background, surveys related work, places the new results in context, and gives an informal overview of the alternating fixpoint construction. Section 3 introduces notation and other preliminaries. Sections 4 and 5 offer a new definition of stable models in terms of an operator on sets of negative literals, and use this operator to define the alternating fixpoint. Section 6 and 7 briefly review well-founded partial models, and show that they are equivalent to alternating fixpoint partial models. Section 8 discusses the expressive power of alternating fixpoint logic *vis à vis* fixpoint logic. Finally, Section 9 discusses future directions.

2 Background and Relation to Other Work

2.1 Program Completion Semantics

The first attempt to put negation as failure on a sound footing was the original *program completion* approach, due to Clark [11], and discussed in detail by Shepherdson [46, 47] and Lloyd [29]. The "completion" of a logic program replaces a set of "if" rules with "if and only if" rules to formalize the intuition that facts are true only if they follow from the rules. As mentioned before, this works out nicely on nonrecursive rules. However, several researchers observed that serious anomalies could arise with recursion. For one thing, the completion could produce an inconsistent program, as shown by the example rule $p \leftarrow \neg p$, which leads to the completed form $p \leftrightarrow \neg p$. To get around this, and other problems, the program completion has been interpreted in 3-valued logic by Fitting [15], and later by Kunen [24], who defined the *3-valued logical consequence semantics*. In the latter approach, "failure to prove" means that at some finite depth *all* proof searches have failed.

However, these developments did not overcome an objection raised in Minker's 1986 workshop [32] that the usual rules to define transitive closure of a directed graph did not yield the value **false** on pairs of nodes not in the transitive closure. Transitive closures arise naturally in applications such as parser generation, type inference, circuit design, theorem proving, and many others. For example, if a graph has edges $e(1,2)$ and $e(2,1)$ and another node 3, then the search for a path from 1 to 3 keeps going around the 1–2 cycle indefinitely, never completely failing. To get around this kind of problem, it was recommended that only searches for "tight" proofs be acceptable; a goal fails when all tight proof attempts have failed [50]. In the context of transitive closure, tight proofs correspond to simple paths.

Interestingly, with respect to the 3-valued logical consequence semantics, Kunen has recently shown that no program that satisfies certain natural restrictions (see Section 8) is able to define a predicate that is true for node pairs that are in the transitive closure and false for those in its complement [25]. No such program is known for the closely related Fitting semantics, either. Fitting and Ben-Jacob have proposed a new 3-valued semantics, but it does not seem to address this problem [16].

2.2 Fixpoint Logic and Variations

A rather different approach to negation grew out of *fixpoint logic* (FP), in which relations are viewed as being defined by induction on first order formulas. Relating this system to logic programming, it is as though rule bodies were permitted to be first order formulas, including formulas with universal quantification. (FP has no function symbols, but the effect of function symbols can be achieved by encoding the graphs of functions in additional infinite EDB relations.) In the original fixpoint logic, studied by Moschovakis [34] on infinite structures, the inductively defined relations are required to appear positively in the defining formula ("given" relations may be positive and/or negative).

This positivity restriction is lifted in the extension called *inductive fixpoint logic* (IFP), studied on infinite structures by Aczel [2] and others, and more recently studied on finite structures by Gurevich and Shelah [19], Kolaitis [21], Abiteboul and Vianu [1], and others. IFP does not really have a concept of negation as "failure to prove". Instead, positive conclusions are drawn in rounds, and any negative literal in a formula that inductively defines a relation evaluates to true if the corresponding positive fact has not been concluded in an earlier round. That fact may well be proved *in the same round*, or later. However, once a positive fact is concluded, it is held forever, even if its proof would no longer work. Operators that retain previously concluded (positive) facts are called *inflationary* [19]. Clearly, the timing of "rule applications" is extremely critical in IFP. Inductive fixpoint logic is discussed further in Section 2.5 from the point of view of its expressive power.

2.3 Stratified Semantics

Another alternative to the program completion approach is *stratified semantics*, in which the positive facts are derived in layers so that each layer only depends negatively on (already completed) lower layers, whose negative facts are taken to be the appropriate complement of the positive facts; in particular, the complement of the transitive closure comes out in the natural way. This approach has been treated in [8, 4, 27, 50], and elsewhere. Since it is only applicable to stratified programs, there has been a search for useful extensions. This search was further justified by the limited expressive power of stratified programs mentioned above.

Perhaps the earliest extension of the stratified class was the locally stratified class, defined and studied by Przymusinski [37]. He defined *perfect* models, and showed that every locally stratified program has a perfect model. Again, not every program is locally stratified, and Cholak has shown recently that it is not decidable in general whether a program *is* locally stratified [10]. Another extension, *weakly perfect models*, is described in [36].

2.4 Well-Founded Semantics and Stable Models

Two further extensions of stratified and locally stratified semantics were introduced approximately concurrently, in the forms of *well-founded semantics* [51] and *stable models* [18]. Every locally stratified program has a total well-founded model and a unique stable model that coincide with each other and with the perfect model. For more general programs, the well-founded semantics provides a partial model, while the stable model semantics provides (possibly multiple, possibly no) total models. Stable models are closely related to well-founded models, and both ideas have been actively studied since their introduction; some principle results are summarized below. The alternating fixpoint, introduced here, provides an additional tie between the well-founded semantics and stable models. First, its definition uses a variant of the stability transformation. Second, it provides a constructive definition of the well-founded partial model.

Van Gelder, Ross, and Schlipf [51] established basic properties of the well-founded semantics: All programs have a well-founded partial model, which can also be interpreted as a 3-valued model in Fitting's sense. The well-founded partial model is defined in terms of a transformation that involves "unfounded sets", which are not constructively defined. Every stable model contains the well-founded partial model. As a corollary, a well-founded total model is always the unique stable model, but not *vice versa*.

Gelfond and Lifschitz [18] proposed their semantics based on *stable models*. Drawing on ideas from autoepistemic logic [33, 17], they define a "stable model" as one that is able to reproduce itself with a certain natural transformation, which we call the *stability transformation* (they call it simply S_Π). They argue that when a program has a *unique* stable model, it is the natural model to associate with the program. More generally, the *stable model semantics* considers an atom true if it is in the intersection of all stable models, and false if it is in the intersection of their complements; no semantics is defined when there is no stable model. The relationship between circumscription and stable models has been explored by Lifschitz [28].

Elkan [14] has shown that stable models on finite domains correspond to *grounded models* of nonmonotonic truth maintenance systems (TMSs) [12], and that the question of whether a set of propositional rules has a stable model is NP-complete. (Independently, Marek and Truszczynski have shown the same NP-

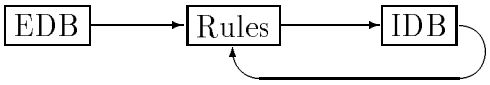

Figure 1: A "concept" as a mapping from the EDB to the IDB.

completeness with a different reduction [31]. In contrast, the well-founded partial model of a propositional program can be found in polynomial time [51].) Elkan also shows stronger results:

1. The question remains NP-complete even if a stable model is known for all but one of the propositional rules.

2. The question remains NP-complete even if each rule has at most one literal in its body.

Saccà and Zaniolo discuss the use of stable models as nondeterministic interpretations [42]. They giving a backtracking fixpoint procedure to construct stable models over finite propositional programs, where a backtracking point corresponds to nondeterministic choice of a (propositional) rule whose only "undefined" premises are negative. These negative premises are presumed to be true, causing the head of the rule to be concluded. A similar idea is mentioned by Doyle [12] for TMSs, and alluded to in other TMS literature, but no precise algorithm has been described. The running time may be unpleasant (factorial in the number of rules!), but this is apparently the only known algorithm that constructs stable models (aside from brute force generation and testing of all subsets of the ground atoms).

In connection with the well-founded semantics, Ross has described and proved properties of a procedural semantics [40], and has studied the semantics on a class of programs called *modularly stratified* [41]. Przymusinski has offered another procedural semantics [38]. *Wellfounded-by-case* and *stable-by-case* models are extensions of well-founded and stable models that were defined and studied by Schlipf [44].

The well-founded semantics apparently has been characterized in several different ways by independent investigations. Przymusinski has constructively defined *generalized perfect models*, using 3-valued logic and employing greatest fixpoints to derive negative facts; they are equivalent to well-founded partial models [38, 35]. Other semantics for negation have been proposed independently by Bidoit and Froidevaux [6], by Bry [7], and by Dung and Kanchanasut [13]. Their points of view are interesting and quite original. However, they seem to have arrived at the same interpretation as is given by the well-founded semantics.

2.5 Concepts and Expressive Power

One important application of logic programs is as a query language over a given database, usually called the *extensional database* (EDB), following Reiter's terminology [39]. The EDB is treated in the logic program as a set of facts. The relations defined by the (nontrivial) rules in the program are called the *intentional database* (IDB). From this point of view, a logic program (interpreted in a specified way) defines a mapping from EDB instances to IDB instances, as suggested in Figure 1. (An *instance* of a database is a set of relations whose names and arities are in accordance with its schema.) We call such a mapping a *concept*. Roughly speaking, a *query* is a question about a concept.

Example 2.1: Suppose we have an EDB schema consisting of one binary relation e, which we interpret as edges in a directed graph. Some well-known concepts we might want a program to express are:

There is a path from X to Y: $p(X, Y)$.

There is *not* a path from X to Y: $np(X, Y)$.

X is a source (has no incoming edges): $s(X)$.

X is well-founded (has no infinite descending chain of edges): $w(X)$.

Some sample queries are:

Is there a path from a to b: $p(a, b)$?

Is there a path from a to anything: $\exists Y\, p(a, Y)$?

What nodes have paths to a, but not to b: $p(X, a) \wedge np(X, b)$?

Is there a path from any source to b: $\exists X\, [p(X, b) \wedge s(X)]$?

To handle such queries, the logic program would have rules for relations p, np, s and w. Queries would be answered by solving appropriate goals.

The separate relation $np(X, Y)$ is listed here because it cannot be represented simply as $np(X, Y) \leftarrow \neg p(X, Y)$ in some semantics, as shown in Example 2.2. □

One way to compare query languages objectively is in terms of their relative expressive power. Informally, this means answering the question, "Are there concepts that are expressible in one query language, but not the other?" If one language is able to simulate another, then it is at least as powerful. Query languages can be separated in terms of expressivity by proving that a certain concept is impossible to express in the weaker language.

A major motivation for introduction of new language constructs is to be able to express some concepts not expressible in the weaker language. The precursors of logic programs were relational calculus queries, which can be thought of as non-recursive logic programs. One of the first results concerning expressive power of query languages was that no relational calculus formula, indeed no first order formula, could express the concept of *transitive closure* (e.g., p in Example 2.1). This fact, well known by logicians, was first observed in a database context by Aho and Ullman [3]. Their article spurred research into the expressive power of various logical systems that incorporated a fixed point operator and were interpreted on finite structures [52, 8, 20, 19, 21, 1].

Inductive fixpoint logic (IFP), mentioned earlier, is sometimes called the *inflationary semantics* of a logic program. IFP has been recommended as an interpretation of logic programs with negation because of its expressive power [22, 21]. IFP was studied on infinite structures by Aczel [2] and others. Among the known results are that, on the integers and other suitably well-behaved infinite structures, the existential fragment of IFP can express Δ_1^1 sets, while full FP can express Π_1^1 sets, a proper superset of Δ_1^1. Also, full IFP expresses a proper superset of Π_1^1, hence is strictly more expressive than FP. (Consult the cited work for further details and definitions of these classes; these definitions are not needed to read this paper.)

More recently, IFP was studied on finite structures by Gurevich and Shelah [19], who showed the surprising result that, on finite structures, FP has the same expressive power as IFP (both methods using full first order formulas). A further, very interesting, result by Abiteboul and Vianu is that the expressive power of the existential fragment of IFP is equal to that of full IFP [1]. Thus, many of the distinctions in expressive power on infinite structures are now known to collapse on finite structures!

Still considering only finite structures, Kolaitis [21] showed that the class of unstratified programs has strictly greater expressive power than the stratified class (discussed above). For unstratified programs, he recommends adoption of IFP.

Example 2.2: Before adopting IFP as the semantics of choice, we would do well to examine a problem considered earlier: expression of the complement of transitive closure. The obvious set of rules is:

$$\begin{aligned} np(X, Y) &\leftarrow \neg p(X, Y). \\ p(X, Y) &\leftarrow e(X, Y). \\ p(X, Y) &\leftarrow e(X, Z),\ p(Z, Y). \end{aligned}$$

or one of its variants, where np is intended to represent the complement of the transitive closure of the edge relation e. This definition works in the stratified semantics (and extensions, well-founded and stable semantics) because p is evaluated completely before considering np, which is in a higher stratum.

However, the inflationary semantics (existential IFP) puts all possible tuples into np because it is evaluated simultaneously with p, and in the first round $\neg p(X, Y)$ is true for all X and Y. Thus it was a significant achievement when a function-free logic program was found that did express the complement of transitive closure in the inflationary semantics [1]. Presumably, in a practical language, we do not want expression of such simple concepts to be significant achievements! This subject is discussed further in Section 8.5. □

One contribution of this paper is that the alternating fixpoint semantics (hence well-founded semantics) is at least as expressive as full fixpoint logic on all structures. In particular, on the integers and other "reasonable" infinite structures, alternating fixpoint logic can express the Π^1_1 sets, and is therefore strictly more expressive than the existential fragment of IFP, which is limited to Δ^1_1 [2]. In the other direction, on finite structures, we show that FP is as expessive as alternating fixpoint logic.

Recent work by Schlipf has shed still more light on the expressive power questions surrounding the well-founded semantics, the stable model semantics, and the Fitting semantics [45]. He has shown that on the integers, on Herbrand universes, and on other "reasonable" infinite structures, all three of these semantics are *equivalent* to Π^1_1. However, on "recursively saturated" infinite structures, the well-founded semantics is *not* contained in Π^1_1 (but the Fitting semantics, as well as FP, still are). (Consult the cited work for additional details, and definitions, including a precise meaning for "reasonable".)

2.6 Modularity of Concepts

Another interesting measure of expressive power is *modularity*: to what extent can different parts of a program be built independently and then fit together? Let $\mathbf{C}$ be a concept that is a natural composition of simpler concepts $\mathbf{C}_1$ and $\mathbf{C}_2$, which have programs $\mathbf{P}_1$ and $\mathbf{P}_2$ that express them. It is desirable that $\mathbf{P} = \mathbf{P}_1 \cup \mathbf{P}_2$ should express the concept $\mathbf{C}$. Many proposed semantics do not enjoy this property.

For example, the global nature of stable models means that combining sets of rules may eliminate some or all stable models. You and Yuan have described *regular* partial models that fall somewhere between well-founded and stable models [53] partly to address this problem. Roughly, if part of a program is "unstable", it is absent from the regular partial models, leaving "stable" parts alone.

Schlipf [45] defined a notion of *uniform translatability* between semantics. Roughly, semantics A is uniformly translatable into semantics B if programs that compose in semantics A to express a concept can be translated individually and independently into programs that compose in semantics B to express the same concept. This means that B enjoys the above modularization/composition property at least to the extent that A does. Schlipf showed that the Fitting program completion semantics is uniformly translatable into the well-founded semantics, but not *vice versa*. In other words, although the Fitting semantics can express any concept on Herbrand structures that is expressible by the well-founded semantics, it cannot necessarily do so modularly.

2.7 Overview of the Alternating Fixpoint

This paper gives a new formulation of the *stability transformation* (originally called simply S_Π [18]) for a given logic program $\mathbf{P}$. We observe that the stability transformation, upon which stable models are based, is *antimonotonic*, a fact that has not been emphasized in previous work, but serves to explain the intractability surrounding stable models. We define $\tilde{\mathbf{S}}_P$ as an operator on a set of *negative* literals, or negative facts, $\tilde{I}$. We shall show in Section 7 that $\tilde{\mathbf{S}}_P$ has a remarkable relationship to the well-founded partial model, which is illustrated in Figure 2. Across the top of the picture, the Herbrand base H is partitioned into the various parts of the well-founded partial model: the negative portion, $\tilde{W}$, the positive portion, $\mathbf{W}^+$, and the undefined portion, $\mathbf{W}^?$. If $\tilde{I}$ is any subset (underestimate) of the negative portion ($\tilde{W}$), then $\tilde{\mathbf{S}}_P(\tilde{I})$ is a superset of the negative and undefined portions combined ($\tilde{W} \cup W^?$), and $\tilde{\mathbf{S}}_P(\tilde{\mathbf{S}}_P(\tilde{I}))$ is again a subset of $\tilde{W}$. As suggested by the picture, the sequence alternates, with one subsequence converging to $\tilde{W}$ from below and

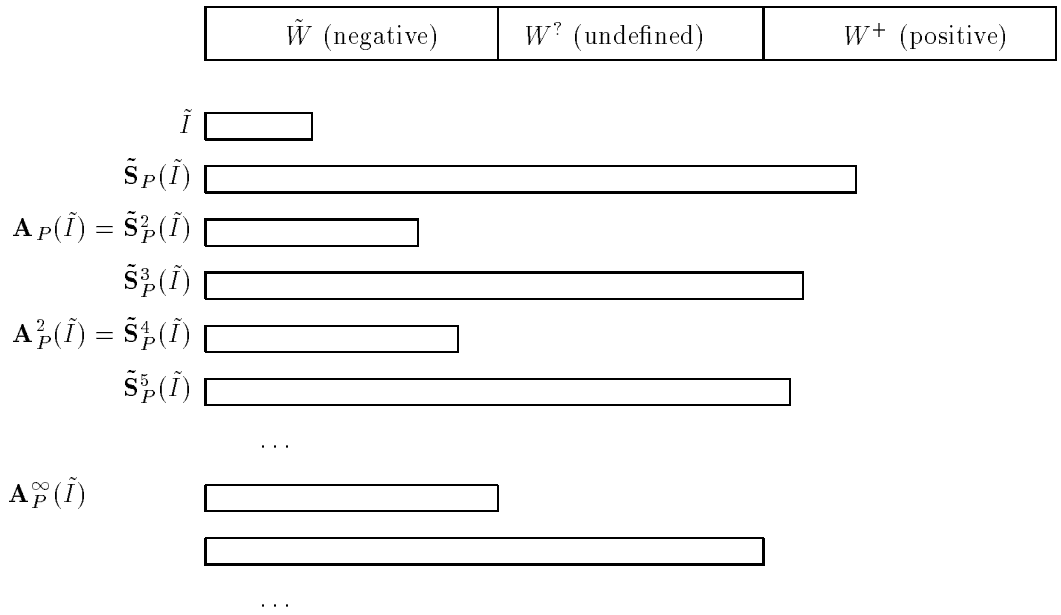

Figure 2: Relationship of $\tilde{\mathbf{S}}_P$ to the well-founded partial model, represented on the top row.

and the other converging to $(\tilde{W} \cup W^?)$ from above. More precisely, let $\mathbf{A}_P(\tilde{I}) = \tilde{\mathbf{S}}_P(\tilde{\mathbf{S}}_P(\tilde{I}))$. Then $\mathbf{A}_P$ is monotonic and its least fixpoint is $\tilde{W}$. Finally, W^+, the positive portion of the well-founded partial model, is found by positive induction, using $\tilde{W}$ as a fixed set of negative facts.

3 Preliminaries

This section covers notational conventions and preliminary facts about fixpoints, operators, and models. We follow established terminology for logic programming as far as possible, which can be found in several standards works, such as [48, 23, 29]. We assume logic programs are normal, in Lloyd's terminology [29], unless stated otherwise; the definition is repeated below.

Definition 3.1: A *normal* rule is one whose body is a conjunction of literals; a *normal* logic program is a finite set of normal rules. □

The Herbrand universe of a logic program $\mathbf{P}$ is the set of all ground (variable-free) terms in the language of $\mathbf{P}$, that is, terms composed of function symbols and constants that appear in $\mathbf{P}$. The Herbrand base H is the set of atoms (atomic formulas) that can be formed from the relations of the program and terms in the Herbrand universe.

We shall distinguish two classes of relations, as described in Section 2.5. The *extensional database* relations (EDB) are relations for which all rules are simply facts (rules without variables or subgoals). The *intentional database* relations (IDB) are relations that are defined by (nontrivial) rules. Since no new conclusions can be drawn about the EDB, its relations are often not mentioned in the Herbrand base and in interpretations.

3.1 Notation

We shall be working with sets of literals (atoms and negated atoms) based on an underlying universe of atoms, usually H, the Herbrand base. We normally use names with a tilde ("$\sim$") for sets of negative literals, and use "+" superscripts in the names of sets of positive literals. These symbols are part of the names of the sets, not operations upon them. We introduce now some notation for frequently used operations on such sets.

Definition 3.2: If I is a set of literals, then $\neg \cdot I$ denotes the set in which each literal in I has been complemented.

1. When H is the Herbrand Universe, $\tilde{H}$ denotes $\neg \cdot H$.
2. We use $I + J$ and $I \Leftrightarrow J$ to denote disjoint union and set difference, respectively. Disjoint union is used primarily when one set consists of positive and the other of negative literals.
3. The *conjugate* of a set of literals is essentially the complement in H, but with the polarity reversed as well; *conjugate* is only defined for sets that are all positive or all negative.
   (a) If I is a set of positive literals, its *conjugate* is the negative set $\overline{I} = \tilde{H} \Leftrightarrow (\neg \cdot I)$.
   (b) If J is a set of negative literals, its *conjugate* is the positive set $\overline{J} = H \Leftrightarrow (\neg \cdot J)$.

□

3.2 Fixpoints of Transformations

In our usage a *transformation* is a mapping of a domain into itself. Let S be a set and let 2^S be its powerset. The important property of 2^S for this section is that it is a complete lattice with partial order $\subseteq$. Let $T : 2^S \rightarrow 2^S$ denote a transformation.

The *ordinal powers*, or *stages*, of T are defined inductively as follows [5]:

$$\begin{array}{rcll} T^0(\emptyset) & = & \emptyset & \\ T^\alpha(\emptyset) & = & T(T^{\alpha-1}(\emptyset)) & \text{for } \alpha \text{ a successor ordinal} \\ T^\alpha(\emptyset) & = & \bigcup\limits_{\beta<\alpha} T^\beta(\emptyset) & \text{for } \alpha \text{ a limit ordinal} \\ T^\infty(\emptyset) & = & \bigcup\limits_{\alpha} T^\alpha(\emptyset) & \end{array}$$

When S is infinite, we may have transfinite induction. As usual, we denote the smallest infinite ordinal by ω.

Often T will be parameterized; it is important that the arguments, which vary through the stages, be clearly separated from the parameters, which are constant through the stages.

Definition 3.3: Let $M : \mathcal{D} \rightarrow \mathcal{R}$ be a general mapping, where both domain $\mathcal{D}$ and range $\mathcal{R}$ are partially ordered (both partial orders are denoted $\subseteq$).

1. M is *monotonic* (or monotonic nondecreasing) if, whenever $I \subseteq J \in \mathcal{D}$, then $M(I) \subseteq M(J) \in \mathcal{R}$.
2. M is *antimonotonic* (or monotonic nonincreasing) if, whenever $I \subseteq J \in \mathcal{D}$, then $M(J) \subseteq M(I) \in \mathcal{R}$.

□

Monotonic transformations have many nice properties, which we shall repeatedly exploit. In particular, the following properties of monotonic transformations are well known.

Theorem 3.1: If T is a monotonic transformation, then $T^\alpha(\emptyset) \subseteq T^{\alpha+1}(\emptyset)$ for all ordinals α, and T has a least fixpoint, which is given by $T^\infty(\emptyset)$.

Proof: See Moschovakis [34] and elsewhere. ▮

Corollary 3.2: If T is a monotonic transformation and $T(I) \subseteq I$, then the least fixpoint of T is a subset of I.

Proof: A routine induction shows that $T^\alpha(\emptyset) \subseteq I$ for all ordinals α. ▮

3.3 Partial Interpretations

A partial *interpretation* I is a partial function from the Herbrand base H into $\{\mathbf{true}, \mathbf{false}\}$. A total interpretation I is such a total function. We shall have no occasion to consider interpretations that do not "correctly interpret" the EDB relations. Therefore, partial and total interpretations are understood to have implicitly $I(r) = \mathbf{true}$ if r is an EDB fact of $\mathbf{P}$, and $I(r) = \mathbf{false}$ if r is an EDB atom that is not a fact of $\mathbf{P}$.

We shall use sets of IDB literals (atoms and negated atoms) to represent partial interpretations: positive literal $p \in I$ denotes that $I(p) = \mathbf{true}$ and negative literal $\neg p \in I$ denotes that $I(p) = \mathbf{false}$. If both p and $\neg p$ are absent from I, then $I(p)$ is undefined. We denote both the function and the set of literals by I.

Definition 3.4: A partial interpretation I is extended to a partial function on literals and conjunctions of literals. The phrase "ϕ is *true in* I(resp. *false* in I)", where ϕ is a formula means the same as $I(\phi) = \mathbf{true}$ (resp. **false**); and similarly for "ϕ is *undefined in* I". First, if p is an atom, then $I(\neg p)$ inverts **true** and **false** for $I(p)$, as expected. However, I is not extended to other negative formulas. Second, let $\phi \stackrel{\text{def}}{=} A \wedge B$ have no free variables. Then

- $I(\phi) = \mathbf{true}$ if $I(A) = \mathbf{true}$ and $I(B) = \mathbf{true}$.
- $I(\phi) = \mathbf{false}$ if $I(A) = \mathbf{false}$ or $I(B) = \mathbf{false}$.
- Otherwise, $I(\phi)$ is undefined.

□

These specifications are sufficient to extend I to normal rule bodies (Definition 3.1). We postpone consideration of more general formulas until Section 8.1. In particular, I is not extended to rules themselves, or to negations other than negative literals.

For a given normal program $\mathbf{P}$, let $\mathbf{P}_H$ be its Herbrand instantiation, in which ground terms in the Herbrand universe are substituted for variables in the rules in every possible way. ($\mathbf{P}_H$ is often infinite.) Each such substitution gives rise to an *instantiated* rule, one in which all variables have been replaced by ground terms.

Definition 3.5: We shall say a partial interpretation I *satisfies* an instantiated normal rule $p \leftarrow \phi$ if any of the following hold:

1. the head of the rule, p is true in I; or
2. the body of the rule, ϕ is false in I; or
3. both p and ϕ are undefined in I.

A partial (resp. total) interpretation that satisfies every rule of $\mathbf{P}_H$ is called a partial (resp. total) *model* of $\mathbf{P}_H$ and of $\mathbf{P}$. □

Observe that satisfaction of a rule cannot be expressed in terms of its truth value in I if we simply interpret $p \leftarrow \phi$ as $p \vee \neg\phi$ (even if we extend I to negations). Under condition 3 the rule's value is undefined. However, not all rules whose truth value is undefined are satisfied. In particular, the value of the rule $p \leftarrow q$ is undefined if $I(p) = \mathbf{false}$ and $I(q)$ is undefined, but this rule is *not* satisfied by I. Some authors introduce 3-valued logic and give a special meaning to $\leftarrow$ to be able to define satisfaction in terms of truth of the rules [15, 24, 38], but we prefer to simply define satisfaction separately. The definition is further motivated by the following theorem and example.

Theorem 3.3: ([51]) For any instantiated normal logic program: (A) a partial model can always be extended to a total model, and (B) there is always a minimum (least defined) partial model. ▮

Example 3.1: This example shows that the above Theorem 3.3 does not hold for reasonable looking alternative definitions of satisfaction.

A partial model might not extend to a total model if any rule whose body "evaluated to undefined" were considered satisfied. Consider the program

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

The partial interpretation $I_1 = \{\neg p\}$ cannot be extended to a total model, because p is true in all models. But the values of all rules are undefined in I_1.

If we drop condition 3 of Definition 3.5 to strengthen the requirements for satisfaction, then there may be no minimum partial model. Without condition 3, $I_2 = \{p\}$ is not a partial model (neither is $I_3 = \{\}$). We are forced to include q or r arbitrarily, giving two incomparable minimal models. □

3.4 The Immediate Consequence Mapping

In this section we provide a uniform framework for a variety of transformations whose fixpoints have been used as the semantics of logic programs. In subsequent sections transformations leading to stable models and to the alternating fixpoint are cast in this framework.

As a starting point, let us consider a Horn clause program $\mathbf{P}$ with associated instantiated program $\mathbf{P}_H$. The transformation $\mathbf{T}_P(I^+)$, where $I^+ \subseteq H$, is called the *immediate consequence transformation* for $\mathbf{P}$ [48, 5]. It is defined by

$$\mathbf{T}_P(I^+) = \{p \mid \begin{array}{l} \mathbf{P}_H \text{ contains a rule whose head is } p \text{ and} \\ \text{every literal of whose body is in } I^+ \end{array}\}$$

In Horn clause programs both I^+ and $\mathbf{T}_P(I^+)$ may represent partial interpretations of $\mathbf{P}$; whether the other atoms are considered false or undefined is immaterial. Essentially the same transformation is used in fixpoint logic [34], where the rule bodies may be first order formulas, but there are still no negative IDB literals.

From now on we distinguish carefully between a *transformation*, which maps a domain into itself, and a *mapping*, which is an arbitrary function. The above transformation for Horn programs has been extended to sets of rules with negative literals in several ways by various researchers. We shall define a *mapping* with two arguments: a set of positive literals and a set of negative literals. Then all transformations can be defined in terms of this two-argument mapping.

Definition 3.6: The *immediate consequence mapping* is the transformation $\mathbf{C}_P(I^+, \tilde{J})$, where $I^+ \subseteq H$ and $\tilde{J} \subseteq \tilde{H}$, defined by

$$\mathbf{C}_P(I^+, \tilde{J}) = \{p \mid \begin{array}{l} \mathbf{P}_H \text{ contains a rule whose head is } p \\ \text{and every literal of whose body is in in} \\ (I^+ + \tilde{J}) \end{array}\}$$

(The combined set $(I^+ + \tilde{J})$ is not required to be a partial interpretation, i.e., it may contain complementary literals.) □

Clearly, for Horn programs $\mathbf{T}_P(I^+) = \mathbf{C}_P(I^+, \emptyset)$; here $\emptyset$ can be replaced by an arbitrary negative set, due to the lack of negative literals in $\mathbf{P}$. The most straightforward extension of $\mathbf{T}_P$ to rules with negative literals is to keep the argument of $\mathbf{T}_P$ as a set of positive literals I^+, and simply consider a negative literal true if its positive counterpart is not in I^+. Recalling Definition 3.2, $\mathbf{T}_P(I^+) \stackrel{\text{def}}{=} \mathbf{C}_P(I^+, \overline{I^+})$. Under this nonmonotonic definition $\mathbf{T}_P^\alpha(\emptyset) \subseteq \mathbf{T}_P^{\alpha+1}(\emptyset)$ frequently fails to hold (*cf.* Theorem 3.1). This extension was studied in [22] and found not to be very satisfactory, in that it led immediately to intractable problems. A modification of the above extension is the one used in *inductive fixpoint logic* (IFP): $\mathbf{T}_P(I^+) \stackrel{\text{def}}{=} \mathbf{C}_P(I^+, \overline{I^+}) \cup I^+$. While this guarantees that $\mathbf{T}_P^\alpha(\emptyset) \subseteq \mathbf{T}_P^{\alpha+1}(\emptyset)$, it is still not monotonic; the term *inflationary* has been used for this operator. Its properties have been studied recently on finite structures [19, 22, 21, 1], and it is the basis for *nonmonotone induction* in older works [2].

A different approach, involving a monotonic extension, has been taken by the logic programming community [5, 30, 29, 15, 24, 46, 37, 50, 51] and elsewhere, and is taken in this paper:

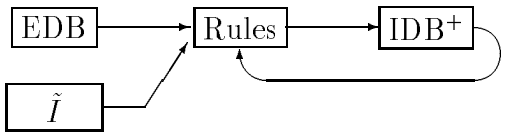

Figure 3: The eventual consequence mapping $\mathbf{S}_P(\tilde{I})$ treats the negative IDB like the EDB.

Definition 3.7: The transformation $\mathbf{T}_P(I)$ is defined for I a set of literals, both positive and negative. Let $I = I^+ + \tilde{I}$, where I^+ is positive and $\tilde{I}$ negative. Then

$$\mathbf{T}_P(I) \stackrel{\text{def}}{=} \mathbf{C}_P(I^+, \tilde{I})$$

□

In this approach, $\mathbf{T}_P$ produces only positive literals; negative literals in the rules are not influenced by positive literals in I. A separate mechanism is used to draw negative conclusions, and various definitions for that mechanism have been studied.

4 The Stability Transformation Revisited

The stability transformation for a normal logic program $\mathbf{P}$, introduced by Gelfond and Lifschitz [18], was described as a transformation on I^+, the set of atoms that are true in the total interpretation I. Here I^+ is a subset of H, the Herbrand base of $\mathbf{P}$. Stable models are fixpoints of this transformation. As originally presented, the transformation involves three stages, intended to correspond to a rational agent's operations on beliefs:

1. Eliminate each rule of $\mathbf{P}_H$ with a negative literal whose atom is in I^+. (Can't believe $\neg p$ if believes p.)
2. Drop negative literals from bodies of remaining rules. (OK to believe $\neg p$ here.)
3. The transformation outputs the minimum model of the resulting Horn program. (Every positive belief is "founded".)

We give an alternative formulation of this transformation, and note that it is antimonotonic. This antimonotonicity property seems to be at the heart of the intractability of stable models (and nonmonotonic truth maintenance systems).

It is quite customary to represent a (total) interpretation I of a logic program as the set I^+ of ground atoms that are true in it; then $\overline{I^+}$ denotes the set of negative literals that are true in I. However, it turns out to be simpler and more intuitive, at least for our purposes, to describe the transformation in terms of a set of *negative* literals.

Definition 4.1: Let $I^+ \subseteq H$ and let $\tilde{I} \subseteq \tilde{H}$. Then

$$\mathbf{T}_{P \cup \tilde{I}}(I^+) \stackrel{\text{def}}{=} \mathbf{C}_P(I^+, \tilde{I})$$

That is, $\tilde{I}$ is thought of as a parameter of a transformation whose domain is H. □

Perhaps the clearest view of $P \cup \tilde{I}$ is that negative literals in $\mathbf{P}$ can be treated as "additional EDB relations" in a Horn program, whose facts are given by $\tilde{I}$.

Definition 4.2: Let $\tilde{I} \subseteq \tilde{H}$. Then the *eventual consequence mapping* is the least fixpoint of $\mathbf{T}_{P \cup \tilde{I}}$, i.e.:

$$\mathbf{S}_P(\tilde{I}) \stackrel{\text{def}}{=} \mathbf{T}^{\infty}_{P \cup \tilde{I}}(\emptyset)$$

Now, our version of the *stability transformation* operates on sets of negative literals (recall Definition 3.2):

$$\tilde{\mathbf{S}}_P(\tilde{I}) \stackrel{\text{def}}{=} \overline{\mathbf{S}_P(\tilde{I})} = \tilde{H} \Leftrightarrow (\neg \cdot \mathbf{S}_P(\tilde{I}))$$

That is, make each atom in $\mathbf{S}_P$ a negative literal and take the complement in $\tilde{H}$. □

Intuitively, $\mathbf{S}_P(\tilde{I})$ gives the set of positive facts that can be (eventually) derived using $\mathbf{P}$ and the fixed set of negative facts $\tilde{I}$, as suggested in Figure 3. Clearly, $\mathbf{S}_P$ is a monotonic mapping. Note that the closure ordinal of $\mathbf{T}_{P \cup \tilde{I}}$ is at most ω [34, 5].

If we adopt the convention that a total model is represented by its negative literals, then a fixpoint of $\tilde{\mathbf{S}}_P$ represents a stable model, by a direct translation of the three-stage definition above. Monotonicity of $\mathbf{S}_P$ implies antimonotonicity of $\tilde{\mathbf{S}}_P$.

5 The Alternating Fixpoint

One way to get a monotonic transformation from an antimonotonic transformation is to compose it with itself, and this is precisely the transformation used for the alternating fixpoint.

Definition 5.1: The *alternating transformation* is defined for $\tilde{I} \subseteq \tilde{H}$ by:

$$\mathbf{A}_P(\tilde{I}) \stackrel{\text{def}}{=} \tilde{\mathbf{S}}_P(\tilde{\mathbf{S}}_P(\tilde{I}))$$

where $\tilde{\mathbf{S}}_P$ is given by Definition 4.2. It is monotonic, so its least fixpoint is given by

$$\tilde{A} = \mathbf{A}^{\infty}_P(\emptyset)$$

This is called *alternating fixpoint* of a program $\mathbf{P}$. □

By its definition, it is clear that every stable model is a fixpoint of $\mathbf{A}_P$; but $\mathbf{A}_P$ may have additional fixpoints, which may or may not correspond to total models.

The closure ordinal of $\tilde{A}$ may be transfinite when the Herbrand universe H is infinite. Of course, for finite H, it is routine to show that the least fixpoint of $\mathbf{A}_P$ is computable in time that is polynomial in the size of H, if the program $\mathbf{P}$ is regarded as fixed.

Definition 5.2: Let $\tilde{A}$ be as defined above and let $A^+ = \mathbf{S}_P(\tilde{A})$. Then the *alternating fixpoint partial model* (AFP model) is $(A^+ + \tilde{A})$. If this is a total model, it is called the *alternating fixpoint total model* (AFP total model). □

Example 5.1: Consider the rules below, where $H = p\{a, b, c, d, e, f, g, h, i\}$. (We abbreviate $\{p(a), p(b), \ldots\}$ by $p\{a, b, \ldots\}$.) The main point of this example is that $p\{d, e, f\}$ eventually become false, while $p\{a, b\}$ remain undefined.

$$\begin{array}{lll} p(a) \leftarrow p(c), \neg p(b). & p(d) \leftarrow p(e), \neg p(f). & p(e) \leftarrow p(d). \\ p(b) \leftarrow \neg p(a). & p(d) \leftarrow p(f), \neg p(g). & p(f) \leftarrow p(e). \\ p(c). & p(d) \leftarrow p(h). & p(f) \leftarrow \neg p(c). \\ & & p(i) \leftarrow p(c), \neg p(d). \end{array}$$

Let us analyze the alternating fixpoint computation, which is governed by $\tilde{I}_{k+1} = \tilde{\mathbf{S}}_P(\tilde{I}_k)$, and is summarized in the following table.

k	$\tilde{I}_k$	$\mathbf{S}_P(\tilde{I}_k)$
0	$\emptyset$	$p\{c\}$
1	$\neg \cdot p\{a, b, d, e, f, g, h, i\}$	$p\{a, b, c, i\}$
2	$\neg \cdot p\{d, e, f, g, h\}$	$p\{c, i\}$
3	$\neg \cdot p\{a, b, d, e, f, g, h\}$	$p\{a, b, c, i\}$
4	$\neg \cdot p\{d, e, f, g, h\}$	$p\{c, i\}$

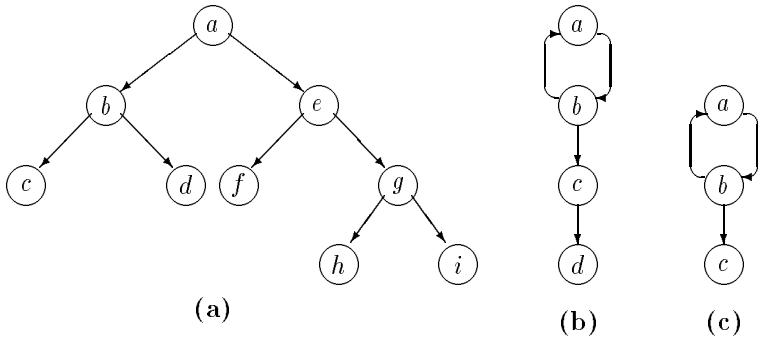

Figure 4: Graphs for Example 5.2: (a) Acyclic; (b) Cyclic with partial model; (c) Cyclic with total model.

We begin with $\tilde{I}_0 = \emptyset$, so only rules with no negative subgoals "have a chance" initially. Thus $\mathbf{S}_P(\emptyset) = \{p(c)\}$. Recall that we treat $\neg p(a)$ and $\neg p(b)$ as separate facts, not as denials of $p(a)$ and $p(b)$, for purposes of $\mathbf{S}_P$. Thus $\mathbf{S}_P(\tilde{I}_1) = p\{a, b, c, i\}$. This illustrates that overestimates of negative facts permit us to derive overestimates of positive facts, and the combination can easily be "contradictory". However, combinations of negative and positive *underestimates* are always "consistent" and define a partial interpretation. By definition, $\mathbf{A}^1_P(\emptyset) = \tilde{I}_2$.

Continuing, we see that $\mathbf{A}^2_P(\emptyset) = \tilde{I}_4 = \tilde{I}_2$, so this is the least fixpoint of $\mathbf{A}_P$. (However, $\tilde{I}_k$ oscillates without converging.) Also, $\mathbf{S}_P(\tilde{I}_4) = p\{c, i\}$ gives the corresponding positive derived facts. Thus

$$\{p(c), p(i), \neg p(d), \neg p(e), \neg p(f), \neg p(g), \neg p(h)\}$$

is the AFP partial model. □

Example 5.2: This example was discussed in previous work [18, 51], and is one of the examples that led to the formulation of well-founded semantics, as well as stable models. Interestingly, this program turns out to be closely related to a game described by Kolaitis, and used to prove that there are queries in fixpoint logic that *are not expressible by stratified programs* [21]. In this respect, the program can be viewed as describing a game where one wins if the opponent has no moves, as in checkers (or draughts).

$$wins(X) \leftarrow move(X, Y), \neg wins(Y).$$

Some sample *move* graphs are shown in Fig. 4. Whenever the *move* EDB relation is acyclic, successive applications of $\mathbf{A}_P$ find nodes that "lose" immediately, then those that lose after one move, then after two moves, etc. For example, in part (a) of the figure, abbreviating *wins* to w, $\mathbf{S}_P(\emptyset) = \emptyset$, so $\tilde{I}_1 = \tilde{\mathbf{S}}_P(\emptyset)$ is "everything". Then $\mathbf{S}_P(\tilde{I}_1)$ is everything with an out-arc, so

$$\begin{aligned} \mathbf{A}_P(\emptyset) &\stackrel{\text{def}}{=} \tilde{I}_2 \stackrel{\text{def}}{=} \tilde{\mathbf{S}}_P(\tilde{I}_1) \\ &= \neg \cdot w\{c, d, f, h, i\} \end{aligned}$$

using the same abbreviation as in the previous example. Continuing, $\mathbf{S}_P(\tilde{I}_2) = w\{b, e, g\}$, so

$$\begin{aligned} \tilde{I}_3 &= \tilde{\mathbf{S}}_P(\tilde{I}_2) \\ &\stackrel{\text{def}}{=} \neg \cdot w\{a, c, d, f, h, i\} \end{aligned}$$

Finally, $\tilde{I}_4 = \tilde{\mathbf{S}}_P(\tilde{I}_3)$ is the same as $\tilde{I}_3$, so is the least fixpoint of both $\mathbf{A}_P$ and $\tilde{\mathbf{S}}_P$.

Part (b) shows a cyclic case in which the AFP model is partial. $\tilde{I}_2 = \mathbf{A}_P(\emptyset) = \{\neg w(d)\}$. Then $\mathbf{S}_P(\tilde{I}_2) = \{w(c)\}$, so the next overestimate is

$$\tilde{I}_3 = \tilde{\mathbf{S}}_P(\tilde{I}_2) = \neg \cdot w\{a, b, d\}$$

which leads back to $\tilde{I}_4 = \mathbf{A}_P(\tilde{I}_2) = \{\neg w(d)\}$. Thus the AFP model is $\{w(c), \neg w(d)\}$.

But even when a cycle is present in the EDB, there may be a total AFP model, as in part (c). Here $\tilde{I}_2 = \mathbf{A}_P(\emptyset) = \{\neg w(c)\}$. Then $\mathbf{S}_P(\tilde{I}_2) = \{w(b)\}$, so the next overestimate is $\tilde{I}_3 = \tilde{\mathbf{S}}_P(\tilde{I}_2) = \neg \cdot w\{a, c\}$, from which $\tilde{I}_4 = \mathbf{A}_P(\tilde{I}_2) = \tilde{\mathbf{S}}_P(\tilde{I}_3) = \neg \cdot w\{a, c\}$. Thus $\{w(b), \neg w(a), \neg w(c)\}$ is the AFP total model.

In parts (a) and (c), we reach fixpoints of not only $\mathbf{A}_P$, but $\tilde{\mathbf{S}}_P$ as well. These are examples of the fact that every AFP total model is also a unique stable model. □

6 Well-Founded Partial Models

This section reviews the definitions of *unfounded sets* and the transformations $\mathbf{U}_P$ and $\mathbf{W}_P$ from [51]. Unfounded sets provide the basis for negative conclusions in the well-founded semantics.

Definition 6.1: Let a program $\mathbf{P}$, its associated Herbrand base H, and a partial interpretation I be given, represented as a set of literals. We say $U \subseteq H$ is an *unfounded set of* $\mathbf{P}$ *with respect to* I if each atom $p \in U$ satisfies the following condition: For each instantiated rule r of $\mathbf{P}$ whose head is p, (at least) one of the following holds:

1. Some literal, q or $\neg q$, in the body of rule r occurs in $\neg \cdot I$, i.e., is false in I.
2. Some positive literal in the body of rule r occurs in U.

A literal that makes (1) or (2) above true is called a *witness of unusability* for rule r (with respect to I). (Note that any atom p that is not in the head of any rule vacuously satisfies the condition to be in U.)

The union of all unfounded sets with respect to a given I is also unfounded, and is called the *greatest unfounded set* (of $\mathbf{P}$ with respect to I). □

Example 6.1: Consider again the rules in Example 5.1. Let $I = \{p(c), \neg p(g), \neg p(h)\}$. Then $U_1 = \{p(d), p(e), p(f)\}$ is an unfounded set with respect to I: The third rule for $p(d)$ and the second rule for $p(f)$ satisfy condition 1 above, and the other rules for $p(d)$, $p(e)$, and $p(f)$ satisfy condition 2.

Note, however, that $U_2 = \{p(a), p(b)\}$ is *not* an unfounded set with respect to I. □

Definition 6.2: For I a set of literals:

- $\mathbf{T}_P(I)$ is the *immediate consequence* transformation (see Definition 3.7).
- $\mathbf{U}_P(I)$ is the greatest unfounded set of $\mathbf{P}$ with respect to I.
- $\mathbf{W}_P(I) = \mathbf{T}_P(I) \cup \neg \cdot \mathbf{U}_P(I)$.

□

It is immediate that $\mathbf{T}_P$, $\mathbf{U}_P$ and $\mathbf{W}_P$, are monotonic transformations. The well-founded partial model is the least fixpoint of $\mathbf{W}_P$.

7 Properties of the Alternating Fixpoint

We now establish the claim presented informally in Section 2.7 and Figure 2, that the alternating fixpoint constructs the well-founded partial model. This section is rather technical. The main points are Lemma 7.5, which states that the alternating fixpoint is contained in the negative portion of the well-founded partial model, and Lemma 7.7, which states that the alternating fixpoint is a superset of the negative portion of the well-founded partial model. They lead to Theorem 7.8, which states the equivalence of the alternating fixpoint and well-founded semantics.

We assume throughout that the program is $\mathbf{P}$, the Herbrand base is H, and the well-founded partial model is W, which consists of positive literals W^+ and negative literals $\tilde{W}$. Let $W^?$ be the undefined portion, but represented as negative literals, i.e., $W^? + \tilde{W} = \overline{W^+}$. Recall the Definition 4.2 of $\mathbf{S}_P$ and $\tilde{\mathbf{S}}_P$.

Lemma 7.1: $\mathbf{S}_P(\tilde{W}) = W^+$.

Proof: Let $I = \mathbf{S}_P(\tilde{W})$ and recall Definition 4.1. Thus I is the least fixpoint of $\mathbf{T}_{P \cup \tilde{W}}$. But

$$\mathbf{T}_{P \cup \tilde{W}}(W^+) = \mathbf{T}_P(W^+ + \tilde{W}) + \tilde{W} = (W^+ + \tilde{W})$$

by definition, so by Cor. 3.2, $I \subseteq W^+$.

Now let $J = \mathbf{U}_P(I + \tilde{W})$. Then $\neg \cdot J \subseteq \tilde{W}$ by monotonicity of U_P. Thus

$$\mathbf{W}_P(I + \tilde{W}) = I + \neg \cdot J \subseteq I + \tilde{W}$$

Again, by Cor. 3.2, $W \subseteq I + \tilde{W}$, so $W^+ \subseteq I$. ∎

Corollary 7.2: If $\tilde{I} \subseteq \tilde{W}$, then $\tilde{\mathbf{S}}_P(\tilde{I}) \supseteq \tilde{W} + W^?$.

Proof: $\mathbf{S}_P$ is monotonic; so by Lemma 7.1, $\mathbf{S}_P(\tilde{I}) \subseteq W^+$. By definition, $\tilde{\mathbf{S}}_P(\tilde{I}) = \overline{\mathbf{S}_P(\tilde{I})} \supseteq \overline{W^+}$. ∎

Lemma 7.3: $\mathbf{S}_P(\tilde{W} + W^?) \supseteq \neg \cdot W^?$, which is the undefined portion of H as positive literals.

Proof: Let $U \subseteq H$ be the set of positive literals:

$$U = \{p \mid p \notin \mathbf{S}_P(\tilde{W} + W^?) \text{ and } p \in \neg \cdot W^?\}$$

Note that $\mathbf{S}_P(\tilde{W} + W^?) \supseteq W^+$ by monotonicity and Lemma 7.1. Let $p \leftarrow q_1, \ldots, q_k$ be any rule for $p \in U$. We claim that some q_i is a "witness of unusability" (see Definition 6.1) for the purpose of showing that U is an unfounded set of $\mathbf{P}$ with respect to W. To prove the claim, note that it cannot be the case that every positive q_i is in $\mathbf{S}_P(\tilde{W} + W^?)$ and every negative q_i is in $(\tilde{W} + W^?)$, or p would be in $\mathbf{S}_P(\tilde{W} + W^?)$. Since $\mathbf{S}_P(\tilde{W} + W^?) \supseteq W^+$, we have three cases:

1. Some positive q_i is not in $\mathbf{S}_P(\tilde{W} + W^?)$ and is in $W^?$, or
2. Some positive q_i is not in $\mathbf{S}_P(\tilde{W} + W^?)$ and is in $\tilde{W}$, or
3. Some negative q_i is in $\neg \cdot W^+$.

Cases (2) and (3) are clear; q_i is false in W. In case (1), q_i is also in U, so the claim is proved. It follows by the definition of $\mathbf{W}_P$ that $U \subseteq \mathbf{U}_P(W) = \neg \cdot \tilde{W}$. But by definition, $U \subseteq W^?$, so $U = \emptyset$. ∎

Corollary 7.4: If $\tilde{I} \supseteq (\tilde{W} + W^?)$, then $\tilde{\mathbf{S}}_P(\tilde{I}) \subseteq \tilde{W}$.

Proof: By Lemma 7.1 and monotonicity, $\mathbf{S}_P(\tilde{I}) \supseteq W^+$. By monotonicity and Lemma 7.3, $\mathbf{S}_P(\tilde{I}) \supseteq \mathbf{S}_P(\tilde{W} + W^?) \supseteq \neg \cdot W^?$. ∎

Lemma 7.5: $\mathbf{A}_P^\infty(\emptyset) \subseteq \tilde{W}$.

Proof: By definition of $\mathbf{A}_P$, Corollaries 7.2 and 7.4, we have $\mathbf{A}_P(\tilde{W}) \subseteq \tilde{W}$. The conclusion follows from Cor. 3.2. ∎

Lemma 7.6: Let $\tilde{A} = \mathbf{A}_P^\infty(\emptyset)$, $A^+ = \mathbf{S}_P(\tilde{A})$, and recall Definition 4.1. Then for all ordinals α,

$$\mathbf{T}^\alpha_{P \cup \overline{A^+}}(\emptyset) \cap \mathbf{U}_P(A^+ + \tilde{A}) = \emptyset$$

Proof: The proof is by induction on α. The basis, $\alpha = 0$, is immediate. For $\alpha > 0$, assume the lemma holds for ordinals $\beta < \alpha$. For limit α, the conclusion is immediate by the inductive hypothesis. For successor ordinal α, let $p \in \mathbf{T}^\alpha_{P \cup \overline{A^+}}(\emptyset)$. Then there is some rule $p \leftarrow q_1, \ldots, q_k, \neg r_1, \ldots, \neg r_m$ such that all q_i are in $\mathbf{T}^{\alpha-1}_{P \cup \overline{A^+}}(\emptyset)$ and all $\neg r_j$ are in $\overline{A^+}$. We shall show that this rule has no "witness of unusability" (see Definition 6.1) with respect to $(A^+ + \tilde{A})$, hence $p \notin \mathbf{U}_P(A^+ + \tilde{A})$. First, by definition:

$$\tilde{A} = \overline{\mathbf{S}_P(\overline{A^+})} \subseteq \overline{\mathbf{T}^{\alpha-1}_{P \cup \overline{A^+}}(\emptyset)}$$

so no q_i of this rule is in $\neg \cdot \tilde{A}$. By the inductive hypothesis, no q_i of this rule is in $\mathbf{U}_P(A^+ + \tilde{A})$. Thus no q_i is a witness of unusability. Similarly, no $\neg r_j$ of this rule is in $\neg \cdot A^+$, so no $\neg r_j$ is a witness of unusability. ∎

Lemma 7.7: $\mathbf{A}_P^\infty(\emptyset) \supseteq \tilde{W}$.

Proof: Let $\tilde{A} = \mathbf{A}_P^\infty(\emptyset)$ and $A^+ = \mathbf{S}_P(\tilde{A})$. By Lemma 7.5, $\tilde{A} \subseteq \tilde{W}$. By monotonicity of $\mathbf{S}_P$ and Lemma 7.1, $A^+ \subseteq W^+$. Consider

$$\mathbf{W}_P(A^+ + \tilde{A}) = \mathbf{T}_P(A^+ + \tilde{A}) + \neg \cdot \mathbf{U}_P(A^+ + \tilde{A})$$

But $\mathbf{T}_P(A^+ + \tilde{A}) = A^+$, and it follows from Lemma 7.6 that

$$\neg \cdot \mathbf{U}_P(A^+ + \tilde{A}) \subseteq \overline{\mathbf{S}_P(\overline{A^+})} = \tilde{A}$$

Thus $\mathbf{W}_P(A^+ + \tilde{A}) \subseteq (A^+ + \tilde{A})$, and therefore by Cor. 3.2, $W \subseteq (A^+ + \tilde{A})$. ∎

Theorem 7.8: The alternating fixpoint partial model is identical to the well-founded partial model.

Proof: By the preceding Lemmas 7.5 and 7.7, the least fixpoint of $\mathbf{A}_P$ is $\tilde{W}$, the negative portion of the well-founded partial model. For the positive portion, we have $A^+ = \mathbf{S}_P(\tilde{W}) = W^+$ by Lemma 7.1. ∎

8 First Order Rule Bodies and Expressive Power

A generalization of logic programs permits rule bodies to be arbitrary formulas of first order logic with equality, instead of being restricted to existential conjunctions of literals; this generalization has been studied by Lloyd and Topor [30], and others. As in [30], we adopt the Clark Equality Theory [11], which essentially specifies that ground terms are equal if and only if they are syntactically identical. In the terminology of [29] a *general* logic program is one that permits arbitrary first order formulas in its rule bodies, whereas a *normal* logic program requires rule bodies to be (existentially quantified) conjunctions of literals.

If a rule body may be any formula of first order logic with equality, then there is no loss of generality in requiring each IDB relation to have just one rule, and we are led to formats that look the same as fixpoint logic [34] and nonmonotonic or inductive fixpoint logic [2, 19]. We show how to generalize the alternating fixpoint (and thus the well-founded semantics) to rules with first order bodies; we call this extension *alternating fixpoint logic*.

Recall that a system in *fixpoint logic* (FP) is essentially a logic program in which rule bodies may be first order formulas, but the inductively defined (IDB) relations are required to appear only positively in those rule bodies, i.e., under an even number of negations. The EDB subgoals may be positive or negative. Fixpoint logic was studied on infinite structures by Moschovakis [34], and more recently on finite structures by numerous researchers [9, 52, 19, 20, 21, 26]. Permitting first order rule bodies in logic programs was studied from the "program completion" point of view by Lloyd and Topor [30].

We show that any FP system can be rewritten into a normal logic program such that the positive part of the AFP model agrees with the original FP model.

8.1 Truth of First Order Rule Bodies

It is reasonably straightforward to generalize alternating fixpoints to programs in which rule bodies may be arbitrary first order formulas, but some care is required. The main point is to be careful about identification of positive and negative atoms, since an atom may be under several negation symbols in the rule body.

Definition 8.1: A subformula of a first order formula is called *positive* if there are an even number of negations above it, and *negative* otherwise. The positivity or negativity of a subformula is called its *polarity*.

A formula is said to be in *explicit literal form* if every negative atom appears in a negative literal, i.e., has a negation immediately above it. In explicit literal form, the *literals of the formula* are defined to be those subformulas of positive polarity that are either atoms or negations of atoms. □

It is easy to convert a formula to explicit literal form by replacing negative atom q by $\neg(\neg q)$ where necessary. For example, $\phi \stackrel{\text{def}}{=} \neg\exists X\, p(X) \equiv \neg\exists X\, [\neg\neg p(X)] \equiv \forall X\, \neg p(X)$.

Definition 8.2: Let I be an arbitrary set of literals. The truth value *assigned by* I to a first order formula without free variables is determined by the following procedure.

1. Put the formula into explicit literal form and identify the literals of the formula.

2. A ground literal is assigned **true** if it occurs in I, and **false** otherwise. Literals must be instantiations of those identified in step 1, and their polarity is based on the whole formula.

3. Logical connectives and quantifiers are evaluated in the standard way.

With some abuse of language we sometimes say "ϕ is true (false) in I", rather than "I assigns **true** (**false**) to ϕ". □

Example 8.1: Applying this definition to the example formula above, $\phi \stackrel{\text{def}}{=} \neg\exists X\, p(X)$, an explicit literal form is $\forall X \neg p(X)$. For ϕ to be assigned **true** by I we require $\neg p(t)$ to be in I for all ground terms t in H. Absence of positive p literals is not enough.

However, let $\psi \stackrel{\text{def}}{=} \neg\phi$. Then $p(X)$ is positive in ψ, so ψ is assigned **false** if I contains $p(t)$ for any ground term t in H. □

Having defined what it means for a first order formula to be assigned **true** by I, the definitions of $\mathbf{T}_P$, $\mathbf{S}_P$, $\tilde{\mathbf{S}}_P$, and $\mathbf{A}_P$ generalize immediately: The head of the rule is in the output if the body is assigned **true** by I. $\mathbf{T}_P$, $\mathbf{S}_P$, and $\mathbf{A}_P$ are still monotonic, and $\tilde{\mathbf{S}}_P$ is still antimonotonic. However, we observe that the closure ordinal of $\mathbf{S}_P$ is no longer bounded by ω, since rule bodies are not necessarily existential formulas.

8.2 Dependency Graphs and Strictness

The dependency graph of a logic program describes how its relations depend on each other with respect to negation [4, 50]. The concept of *strictness* is defined in terms of this graph. Strictness was defined in [4] for normal logic programs (see also [25]); the extension of both concepts to general logic programs is straightforward.

Definition 8.3: The *dependency graph* of a logic program is a directed graph in which the relation symbols are nodes. There is an arc from p to q if the program contains a rule in which p occurs in the head and q occurs as a subgoal in the body. The arc is labeled according to the polarity of q in the body:

1. if q occurs only negatively, the arc is called "negative";

2. if q occurs only positively, the arc is called "positive";

3. if q occurs both positively and negatively, the arc is called "mixed".

Strictness is based on (directed) paths in this graph. In the definitions below p and q may be the same relation symbol. The null path is considered a path for these purposes, so p always has a path to itself with zero negative arcs. Also, paths need not be simple.

1. We say p is *strictly positive in* q if every path from p to q traverses an even number of negative arcs and no mixed arcs.

2. We say p is *strictly negative in* q if every path from p to q traverses an odd number of negative arcs and no mixed arcs.

3. We say the ordered pair of relations (p, q) is *strict* if p is strictly positive in q, or p is strictly negative in q, or there is no path from p to q; otherwise we say the pair (p, q) is *mixed*.

4. A program is called *strict* if every ordered pair of relations is strict.

5. A program is called *strict in the IDB* if every ordered pair of IDB relations is strict.

□

A general logic program that fits the requirements of fixpoint logic is strict in the IDB simply because there are no negative IDB subgoals. We shall study programs that are strict in the IDB but do contain negative subgoals. It is clear that for such programs the IDB relations may be partitioned into two sets, which we call the *globally positive* and *globally negative* relations, such that all pairs from the same set are either strictly positive or unrelated. Similarly, all pairs with one relation from each set are either strictly negative or unrelated.

8.3 Simulation of Fixpoint Logic

It is easy to see (Theorem 8.1 below) that a fixpoint logic system can be interpreted in alternating fixpoint logic as a general logic program with the same resulting semantics. This shows that alternating fixpoint logic is an extension of fixpoint logic, but that in itself is not very significant. The more interesting result is that this general logic program can be transformed straightforwardly into a normal logic program while preserving the positive part of the AFP model on the relations in the original program. This is established by a series of technical lemmas culminating in Theorems 8.6 and 8.7. Thus normal logic programs in alternating fixpoint logic have at least the expressive power of full fixpoint logic.

Theorem 8.1: Let **P** be a general logic program with only positive IDB literals in the rule bodies. Then the positive part of the AFP model is the same as the set of relations defined by fixpoint logic for **P**.

Proof: Since there are no negative IDB literals, $\mathbf{S}_P(\tilde{J})$ is the same for any set of negative literals. But $\mathbf{S}_P(\emptyset)$ is the result of fixpoint logic. ∎

To convert a system Ψ with first order rule bodies into a normal logic program **P**, it is necessary to eliminate universal quantifiers, define new relations to represent negative existential subformulas, and put the final rule bodies into disjunctive normal form (DNF); the procedure is implemented as a system of rewrite rules. The details, and proof of termination may be found in [30]. We call the set of auxiliary relations the *auxiliary database* (ADB) to differentiate it from the original IDB relations of Ψ.

Example 8.2: Consider a general program Ψ to determine the well-founded set of nodes in a binary relation e, which may be finite or infinite. Recall that a node is "well-founded" in standard mathematical terminology if it has no infinite descending chain from it, and unfounded otherwise. Let w represent "well-founded". The well-founded property is expressible in FP by:

$$w(X) \leftarrow \neg\exists Y\,[e(Y,X) \wedge \neg w(Y)]$$

The $w(Y)$ subgoal is positive, but within a negative existential subformula. Since Ψ has no negative IDB subgoals, $\mathbf{S}_\Psi(\tilde{I})$ is the same for any set of negative literals, $\tilde{I}$, and contains the w atoms that represent the well-founded nodes within e. It follows that $\mathbf{A}^\infty_\Psi(\emptyset)$ contains all the elements *not* in the set of well-founded nodes in e.

To transform Ψ into a normal logic program **P**, we "extract" the negative existential subformula, and give it an auxiliary relation name, u. (Think of u as "unfounded".)

$$\begin{aligned} w(X) &\leftarrow \neg u(X) \\ u(X) &\leftarrow \exists Y\,[e(Y,X) \wedge \neg w(Y)] \end{aligned}$$

The final program, in normal syntax, is:

$$\begin{aligned} w(X) &\leftarrow \neg u(X). \\ u(X) &\leftarrow e(Y,X),\ \neg w(Y). \end{aligned}$$

It is easily verified that the positive w literals in its AFP model are indeed the well-founded part of e, as are the negative u literals. The point is that w atoms succeed when the corresponding u atoms fail, but failure of u atoms is influenced by successful w atoms.

Notice that there will be *no* positive literals for the auxiliary relation u in the AFP model. This is typical for auxiliary relations that replace negative subformulas of the original system. Similarly, there are no negative w literals in $\mathbf{A}_P^\infty(\emptyset)$, in contrast to what happened with $A_\Psi^\infty(\emptyset)$. This suggests that the alternating fixpoint on normal programs captures the negation of positive existential closures (such as transitive closure), but not the negation of positive universal closures (such as well-foundedness). □

Let Ψ be a general logic program that is strict in the IDB (see Definition 8.3). To transform Ψ into a normal logic program in a way that we can prove preserves the positive part of the AFP model, we use a special case of the procedure described in [30]. We first put the rule bodies into *existential disjunctive normal form* (EDNF); then we perform a series of rewrites called elementary simplifications.

Rewriting a first order formula into EDNF is accomplished as follows:

1. Replace $\forall X$ by $\neg\exists X\neg$.
2. Push $\neg$'s down to atoms or $\exists$; eliminate $\neg\neg$.
3. Push $\wedge$ down through $\vee$ (distribute).
4. Push $\exists$ through $\vee$; unnecessary $\exists$ may be eliminated.

Do the above until no more changes are possible. Note that the only $\vee$ in an EDNF formula is at the top.

Definition 8.4: Consider a general logic program $\mathbf{P}$ with IDB relation p, first order rule bodies in EDNF form, and only one rule per IDB relation, such that

1. $p(\vec{X}) \leftarrow \psi(\vec{X})$ is the rule for p;
2. $\phi(\vec{U})$ appears as a subformula of $\psi(\vec{X})$, the rule for p;
3. $\phi(\vec{U})$ is an existentially quantified literal or conjunction of literals;
4. ϕ and ψ have no variables other than $\vec{U}$ in common.

Define $\mathbf{P}'$ to be the same program as $\mathbf{P}$ except for these changes:

1. There is a new relation symbol q with the rule $q(\vec{U}) \leftarrow \phi(\vec{U})$, which is a *normal* rule.
2. The subformula $\phi(\vec{U})$ in ψ is replaced by the atom $q(\vec{U})$.

Then $\mathbf{P}'$ is said to be obtained from $\mathbf{P}$ by an *elementary simplification*. □

Once in EDNF we continue "extracting" lowest existential subformulas and introducing new relation names and rules for them until it is no longer possible. The quantifier-free part of a lowest existential subformula must be in the form of a literal or a conjunction of literals; therefore the extraction is an elementary simplification. The final result is a normal logic program $\mathbf{P}$ whose IDB relations are those of Ψ plus the auxiliary relations. We shall show that each elementary simplification preserves the IDB relations in the AFP model.

Definition 8.5: Each auxiliary relation is classified as *globally positive* or *globally negative* in accordance with the polarity of the subformula in Ψ that it represents. The original inductive (IDB) relations in Ψ are globally positive. □

For the discussion leading up to the next theorem we shall be considering two general logic programs $\mathbf{P}$ and $\mathbf{P}'$ that we may think of as intermediate forms in the transformation from Ψ to a normal logic program. Their relations include IDB relations p and q. $\mathbf{P}$ and $\mathbf{P}'$ have first order rule bodies, only one rule per IDB relation, and are related by elementary simplification. Specifically, $\mathbf{P}$ is a program such that

1. $p(\vec{X}) \leftarrow \psi(\vec{X})$ is the rule for p;
2. $\phi(\vec{U})$ appears as a subformula of $\psi(\vec{X})$, the rule for p;
3. $\phi(\vec{U})$ is an existentially quantified literal or conjunction of literals;
4. ϕ and ψ have no variables other than $\vec{U}$ in common;
5. $q(\vec{U}) \leftarrow \phi(\vec{U})$ is the rule for q, but q occurs nowhere in any rule body.

$\mathbf{P}'$ is the same program as $\mathbf{P}$ with the subformula $\phi(\vec{U})$ in ψ replaced by the atom $q(\vec{U})$. Thus $\mathbf{P}$ without the rule for q could lead to $\mathbf{P}'$ by an elementary simplification. Clearly the rule for q in $\mathbf{P}$ does not affect any other relations in the AFP model; it merely gives $\mathbf{P}$ and $\mathbf{P}'$ the same set of relation symbols.

Let us denote

$$\phi(\vec{U}) \stackrel{\text{def}}{=} \exists \vec{V}\,(s_1(\vec{U},\vec{V}) \wedge \cdots \wedge s_k(\vec{U},\vec{V}) \wedge \neg r_1(\vec{U},\vec{V}) \wedge \cdots \wedge \neg r_m(\vec{U},\vec{V}))$$

Since this is a *normal* rule body, it makes sense to talk about "the instantiated rules for $q(\vec{t})$", where $\vec{t}$ is a ground tuple. (These rules range over all assignments of ground terms to $\vec{V}$.) Then

$$\neg\phi(\vec{t}) \equiv \forall \vec{V}\,(\neg s_1(\vec{t},\vec{V}) \vee \cdots \vee \neg s_k(\vec{t},\vec{V}) \vee r_1(\vec{t},\vec{V}) \vee \cdots \vee r_m(\vec{t},\vec{V}))$$

In the terminology of normal rules, $\phi(\vec{t})$ is true when some instantiated rule for $q(\vec{t})$ is true; but $\neg\phi(\vec{t})$ is true when *every* such instantiated rule contains a literal whose *complement* is true. This relationship will be used in subsequent lemmas.

Lemma 8.2: Let general programs $\mathbf{P}$ and $\mathbf{P}'$ be as described above. Let $\tilde{I}$ be a fixpoint of $\mathbf{A}_P$. Then

$$\{\vec{t} \mid \neg q(\vec{t}) \in \tilde{I}\} = \{\vec{t} \mid \neg\phi(\vec{t}) \text{ is true in } (\mathbf{S}_P(\tilde{I}) + \tilde{I})\}$$

The same holds with $\mathbf{P}$ replaced by $\mathbf{P}'$ throughout.

Proof: Define $I^+ = \mathbf{S}_P(\tilde{I})$. Then the set on the left is the same as $\{\vec{t} \mid q(\vec{t}) \notin \mathbf{S}_P(\overline{I^+})\}$, which is the same as

$$\{\vec{t} \mid \phi(\vec{t}) \text{ is false in } (\mathbf{S}_P(\overline{I^+}) + \overline{I^+})\}$$

Note that it is necessary to distinguish between "$\phi(\vec{t})$ false" and "$\neg\phi(\vec{t})$ true" because of partial interpretation (see Example 8.1). Tuple $\vec{t}$ is in the last set if and only if every instantiated rule for $q(\vec{t})$ contains a literal that is false. We show that the complement of that literal *is* true in $(\mathbf{S}_P(\tilde{I}) + \tilde{I})$. This is immediate for EDB literals by our convention that partial interpretations must contain a correct total interpretation of the EDB. Of course, a positive literal cannot be true in $\overline{I^+}$, and a negative literal cannot be true in $\mathbf{S}_P(\overline{I^+})$. A positive IDB literal $s(\vec{t},\vec{v})$ is false in $\mathbf{S}_P(\overline{I^+})$ if and only if its negation is in $\tilde{I}$, because $\tilde{I} = \tilde{\mathbf{S}}_P(\overline{I^+})$. A negative IDB literal $\neg r(\vec{t},\vec{v})$ is false in $\overline{I^+}$ if and only if its positive version is in $I^+ = \mathbf{S}_P(\tilde{I})$. This completes the proof because $\neg\phi(\vec{t})$ is true in $(\mathbf{S}_P(\tilde{I}) + \tilde{I})$ if and only if in each instantiated rule for $q(\vec{t})$ the complement of some literal is true. ∎

When a positive subformula is extracted by elementary simplification, we show that the *entire* AFP model is preserved.

Lemma 8.3: Let general programs $\mathbf{P}$ and $\mathbf{P}'$ be as described above, and suppose that the replaced subformula $\phi(\vec{U})$ occurs positively in $\psi(\vec{X})$. Then for all ordinals α,

$$\begin{array}{lrcl} \text{(a)} & \mathbf{A}^{\alpha}_{P'}(\emptyset) & = & \mathbf{A}^{\alpha}_{P}(\emptyset) \\ \text{(b)} & \mathbf{S}_{P'}(\mathbf{A}^{\alpha}_{P'}(\emptyset)) & = & \mathbf{S}_{P}(\mathbf{A}^{\alpha}_{P}(\emptyset)) \end{array}$$

Also, $\mathbf{P}$ and $\mathbf{P}'$ have the same AFP model.

Proof: By monotonicity, $\mathbf{S}_{P'}(\tilde{J}) = \mathbf{S}_P(\tilde{J})$ for an arbitrary set of negative literals $\tilde{J}$, so $\mathbf{A}^{\alpha}_{P'}(\emptyset) = \mathbf{A}^{\alpha}_{P}(\emptyset)$ by a trivial induction. ∎

The case where $\phi(\vec{U})$ occurs negatively is much less obvious, and normally the entire AFP model is not preserved. Once $\neg q(\vec{t})$ appears in $\mathbf{A}^{\alpha}_{P'}(\emptyset)$ it stays there, but the truth of $\neg\phi(\vec{t})$ has to be rederived in $\mathbf{P}$ each time. We also have to consider the possibility that $\neg\phi(\vec{t})$ can be derived in $\mathbf{P}$ in some cases that $\neg q(\vec{t})$ does not appear in $\mathbf{A}^{\alpha}_{P'}(\emptyset)$. First we show inclusion in one direction.

Lemma 8.4: Let general programs $\mathbf{P}$ and $\mathbf{P}'$ be as described above, and suppose that the replaced subformula $\phi(\vec{U})$ occurs negatively in $\psi(\vec{X})$. Then for all ordinals α,

$$\begin{array}{lrcl} \text{(a)} & \mathbf{A}^{\alpha}_{P'}(\emptyset) & \subseteq & \mathbf{A}^{\alpha}_{P}(\emptyset) \\ \text{(b)} & \mathbf{S}_{P'}(\mathbf{A}^{\alpha}_{P'}(\emptyset)) & \subseteq & \mathbf{S}_{P}(\mathbf{A}^{\alpha}_{P}(\emptyset)) \end{array}$$

Thus, the AFP model of $\mathbf{P}'$ is a subset of that of $\mathbf{P}$.

Proof: The proof is by induction on α. Clearly the lemma holds for $\alpha = 0$. For the induction, let $\alpha > 0$ and assume that the lemma holds for ordinals $\beta < \alpha$.

Consider successor ordinal $\alpha = \beta + 1$ first. Let $\tilde{I} = \mathbf{A}^{\beta}_{P'}(\emptyset)$, and let $I^+ = \mathbf{S}_{P'}(\mathbf{A}^{\beta}_{P'}(\emptyset))$. Let $\tilde{J} = \mathbf{A}^{\beta}_{P}(\emptyset)$, and let $J^+ = \mathbf{S}_P(\mathbf{A}^{\beta}_{P}(\emptyset))$. By the inductive hypothesis, $\tilde{I} \subseteq \tilde{J}$ and $I^+ \subseteq J^+$, so $\overline{I^+} \supseteq \overline{J^+}$.

To prove part (a) we need $\tilde{\mathbf{S}}_{P'}(\overline{I^+}) \subseteq \tilde{\mathbf{S}}_P(\overline{J^+})$, so we need to show that $\mathbf{S}_{P'}(\overline{I^+}) \supseteq \mathbf{S}_P(\overline{J^+})$. The only problem is that $\mathbf{P}'$ has $\neg q(\vec{U})$ where $\mathbf{P}$ has $\neg\phi(\vec{U})$ in the rule for p. It is sufficient to show that

$$\{\vec{t} \mid \neg\phi(\vec{t}) \text{ is true in } (\mathbf{S}_P(\overline{J^+}) + \overline{J^+})\} \subseteq \{\vec{t} \mid \neg q(\vec{t}) \in \overline{I^+}\}$$

If $\neg\phi(\vec{t})$ is true in $(\mathbf{S}_P(\overline{J^+}) + \overline{J^+})$, then each instantiated rule for $q(\vec{t})$ has a literal whose complement is true in $(\mathbf{S}_P(\overline{J^+}) + \overline{J^+})$. The same EDB literals are true in $(\mathbf{S}_{P'}(\overline{I^+}) + \overline{I^+})$, so consider IDB literals. The complement of positive IDB literal $s(\vec{t},\vec{v})$ is true in $(\mathbf{S}_P(\overline{J^+}) + \overline{J^+})$ if and only if $\neg s(\vec{t},\vec{v}) \in \overline{J^+} \subseteq \overline{I^+}$, in which case $s(\vec{t},\vec{v}) \notin I^+$. The complement of negative IDB literal $\neg r(\vec{t},\vec{v})$ is true in $(\mathbf{S}_P(\overline{J^+}) + \overline{J^+})$ if and only if $r(\vec{t},\vec{v}) \in \mathbf{S}_P(\overline{J^+})$. But then $\neg r(\vec{t},\vec{v}) \notin \mathbf{A}_P(\tilde{J})$, and by Theorem 3.1, $\neg r(\vec{t},\vec{v}) \notin \tilde{J}$, either. By the inductive hypothesis, $\neg r(\vec{t},\vec{v}) \notin \tilde{I}$. Since these conclusions hold for every instantiated rule for $q(\vec{t})$, we have that $q(\vec{t}) \notin I^+$, hence $\neg q(\vec{t}) \in \overline{I^+}$. This proves part (a) for successor ordinals. Part (a) for limit ordinals follows immediately from the definitions.

For part (b), where $\alpha = \beta + 1$ we need to show

$$\{\vec{t} \mid \neg q(\vec{t}) \in \mathbf{A}_{P'}(\tilde{I})\} \subseteq \{\vec{t} \mid \neg\phi(\vec{t}) \text{ is true in } (\mathbf{S}_P(\mathbf{A}_P(\tilde{J})) + \mathbf{A}_P(\tilde{J}))\}$$

The set on the left is the same as $\{\vec{t} \mid q(\vec{t}) \notin \mathbf{S}_{P'}(\overline{I^+})\}$. Every instantiated rule for such $q(\vec{t})$ has a literal that is not in $(\mathbf{S}_{P'}(\overline{I^+}) + \overline{I^+})$. Again, EDB literals in each instantiated rule for $q(\vec{t})$ are interpreted the same in $\mathbf{P}$ and $\mathbf{P}'$, so it is sufficient to consider IDB literals. Suppose s is a positive literal of an instantiated rule for $q(\vec{t})$ that is not in $\mathbf{S}_{P'}(\overline{I^+})$. Then $s \notin \mathbf{S}_P(\overline{J^+})$, so $\neg s \in \mathbf{A}_P(\tilde{J})$. Suppose $\neg r$ is a negative literal of an instantiated rule for $q(\vec{t})$ that is not in $\overline{I^+}$. Then $r \in \mathbf{S}_{P'}(\mathbf{A}^{\beta}_{P'}(\emptyset)$, which implies that $r \in \mathbf{S}_P(\mathbf{A}^{\beta}_{P}(\emptyset))$ by the inductive hypothesis, and so $r \in \mathbf{S}_P(\mathbf{A}_P(\tilde{J}))$ by Theorem 3.1. This completes part (b) for successor ordinals.

For α a limit ordinal, and $\neg q(\vec{t}) \in \mathbf{A}^{\alpha}_{P'}(\emptyset)$, there is some successor ordinal $\beta < \alpha$ such that $\neg q(\vec{t}) \in \mathbf{A}^{\beta}_{P'}(\emptyset)$. Then, as shown, $\neg\phi(\vec{t})$ is true in $(\mathbf{S}_P(\mathbf{A}^{\beta}_{P}(\emptyset)) + \mathbf{A}^{\beta}_{P}(\emptyset))$, and part (b) follows by monotonicity of $\mathbf{S}_P$. ∎

The preceding lemma shows that $\mathbf{P}'$ is "slower" than $\mathbf{P}$ in some sense. The next lemma shows that, with additional hypotheses, $\mathbf{P}'$ does eventually "catch up" on critical relations. Examples can easily be constructed that show that additional hypotheses are necessary. First we introduce some notation.

Definition 8.6: Let the partition of the set of IDB relations into *globally positive* and *globally negative* be understood. If I is any set of literals, then

$$\begin{array}{ll} [I]_+ & \stackrel{\text{def}}{=} \{\text{literals in } I \text{ of globally positive relations}\} \\ [I]_- & \stackrel{\text{def}}{=} \{\text{literals in } I \text{ of globally negative relations}\} \end{array}$$

□

Lemma 8.5: Let general programs $\mathbf{P}$ and $\mathbf{P}'$ be as described above, and suppose that the replaced subformula $\phi(\vec{U})$ occurs negatively in $\psi(\vec{X})$. *Furthermore*, assume that $\mathbf{P}$ and $\mathbf{P}'$ are strict in the IDB, have the same sets of globally positive and globally negative relations, and that p is globally positive. Then for all ordinals α,

$$\begin{array}{lrcl} \text{(a)} & [\mathbf{A}^{\alpha}_{P}(\emptyset)]_{-} & \subseteq & \mathbf{A}^{\infty}_{P'}(\emptyset) \\ \text{(b)} & [\mathbf{S}_{P}(\mathbf{A}^{\alpha}_{P}(\emptyset))]_{+} & \subseteq & \mathbf{S}_{P'}(\mathbf{A}^{\infty}_{P'}(\emptyset)) \end{array}$$

Proof: Let $\tilde{A} \stackrel{\text{def}}{=} \mathbf{A}^{\infty}_{P'}(\emptyset)$, and let $A^{+} \stackrel{\text{def}}{=} \mathbf{S}_{P'}(\tilde{A})$. We note that $\overline{\tilde{A}} = \mathbf{S}_{P'}(\overline{A^{+}})$ and $\overline{A^{+}} = \tilde{\mathbf{S}}_{P'}(\tilde{A})$. The lemma is proved by induction on α. The basis, $\alpha = 0$, is immediate for part (a), and the induction is trivial for part (a) when α is a limit ordinal. For successor ordinal α, assume part (a) of the lemma holds for ordinals less than α. Let $\tilde{I} \stackrel{\text{def}}{=} \mathbf{A}^{\alpha-1}_{P}(\emptyset)$, and let $I^{+} \stackrel{\text{def}}{=} \mathbf{S}_{P}(\tilde{I})$. (We suppress the dependence of "local variables" $\tilde{I}$ and I^{+} on α.) We have $\left[\tilde{I}\right]_{-} \subseteq \tilde{A}$ by the inductive hypothesis. First we shall show that $[I^{+}]_{+} \subseteq A^{+}$, which establishes part (b) for $\alpha \Leftrightarrow 1$; note that this covers both successor and limit ordinals, including 0.

To show that $[I^{+}]_{+} \subseteq A^{+}$, we claim (Claim 1) that for all ordinals β:

1. $\left[\mathbf{T}^{\beta}_{P \cup \tilde{I}}(\emptyset)\right]_{+} \subseteq A^{+}$
2. $\{\vec{t} \mid \neg\phi(\vec{t}) \text{ is true in } (\left[\mathbf{T}^{\beta}_{P \cup \tilde{I}}(\emptyset)\right]_{=} + \left[\tilde{I}\right]_{-})\} \subseteq \{\vec{t} \mid \neg\phi(\vec{t}) \text{ is true in } (A^{+} + \tilde{A})\}$.

The claim holds for $\beta = 0$ and the induction is trivial for β a limit ordinal. For β a successor ordinal, assume the claim holds for ordinals less than β. Suppose $\neg\phi(\vec{t})$ is true in $(\left[\mathbf{T}^{\beta}_{P \cup \tilde{I}}(\emptyset)\right]_{+} + \left[\tilde{I}\right]_{-})$. Then each instantiated rule for $q(\vec{t})$ has a literal whose complement is true in $(\left[\mathbf{T}^{\beta}_{P \cup \tilde{I}}(\emptyset)\right]_{+} + \left[\tilde{I}\right]_{-})$. The same EDB literals are true in $(A^{+} + \tilde{A})$, so consider IDB literals. Of course, a positive literal cannot be true in $\left[\tilde{I}\right]_{-}$, and a negative literal cannot be true in $\left[\mathbf{T}^{\beta}_{P \cup \tilde{I}}(\emptyset)\right]_{+}$. Since the programs are strict in the IDB and p is globally positive, q is globally negative. Let $s(\vec{t}, \vec{v})$ be a positive IDB literal of an instantiated rule for $q(\vec{t})$. Then s is globally negative, too. The complement, $\neg s(\vec{t}, \vec{v})$, is true in $\tilde{I}$ if and only if $\neg s(\vec{t}, \vec{v}) \in \left[\tilde{I}\right]_{-} \subseteq \tilde{A}$. Let $\neg r(\vec{t}, \vec{v})$ be a negative IDB literal of an instantiated rule for $q(\vec{t})$. Then r is globally positive. The complement, $r(\vec{t}, \vec{v})$, is true in $\mathbf{T}^{\beta}_{P \cup \tilde{I}}(\emptyset)$ if and only if

$$r(\vec{t}, \vec{v}) \in \mathbf{T}_{P \cup \tilde{I}}(\mathbf{T}^{\beta-1}_{P \cup \tilde{I}}(\emptyset))$$

By strictness in the IDB this is true if and only if the body of the rule for $r(\vec{t}, \vec{v})$ is true in $(\left[\mathbf{T}^{\beta-1}_{P \cup \tilde{I}}(\emptyset)\right]_{+} + \left[\tilde{I}\right]_{-})$.

If r has the same rule in $\mathbf{P}'$ as in $\mathbf{P}$, then $r(\vec{t}, \vec{v}) \in A^{+}$, because $(\left[\mathbf{T}^{\beta-1}_{P \cup \tilde{I}}(\emptyset)\right]_{+} + \left[\tilde{I}\right]_{-}) \subseteq (A^{+} + \tilde{A})$. The only other case is that r is the same relation as p, in which case the rule in $\mathbf{P}'$ has a literal $\neg q$ where the rule in $\mathbf{P}$ has $\neg\phi$. But by part (2) of the inductive hypothesis, Lemma 8.2, and the fact that $\neg\phi$ occurs positively in the rule for p, we have $r(\vec{t}, \vec{v}) \in A^{+}$ in this case, too. It follows that $\neg\phi(\vec{t})$ is true in $(A^{+} + \tilde{A})$, establishing part (2) of the lemma for β. An argument similar to the one for $r(\vec{t}, \vec{v})$ shows that any atom in $\left[\mathbf{T}^{\beta}_{P \cup \tilde{I}}(\emptyset)\right]_{+}$, being globally positive, is also in A^{+}. This completes the proof of Claim 1.

From Claim 1 we have

$$\left[I^{+}\right]_{+} = \bigcup_{\beta} \left[\mathbf{T}^{\beta}_{P \cup \tilde{I}}(\emptyset)\right]_{+} \subseteq A^{+}$$

and it follows that

$$\left[\tilde{\mathbf{S}}_{P}(\tilde{I}) \stackrel{\text{def}}{=} \overline{I^{+}}\right]_{+} \supseteq \left[\overline{A^{+}}\right]_{+} \qquad (*)$$

We now claim (Claim 2) that $(*)$ implies that

$$\left[\mathbf{S}_{P}(\tilde{\mathbf{S}}_{P}(\tilde{I}))\right]_{-} \supseteq \left[\mathbf{S}_{P'}(\overline{A^{+}})\right]_{-}$$

To prove Claim 2, we note that all the globally negative relations have the same rules in $\mathbf{P}'$ and $\mathbf{P}$. Excluding dependence on the EDB, which is not an issue, these rules depend positively only on globally negative IDB relations, and they depend negatively only on globally positive IDB relations. That is, $\left[\mathbf{S}_P(\tilde{\mathbf{S}}_P(\tilde{I}))\right]_-$ is influenced only by $\left[\tilde{\mathbf{S}}_P(\tilde{I})\right]_+$, and $\left[\mathbf{S}_{P'}(\overline{A^+})\right]_-$ is influenced only by $\left[\overline{A^+}\right]_+$. By a trivial induction, for all ordinals β,

$$\left[\mathbf{T}^{\beta}_{P\cup\tilde{\mathbf{S}}_P(\tilde{I})}(\emptyset)\right]_- \supseteq \left[\mathbf{T}^{\beta}_{P'\cup\overline{A^+}}(\emptyset)\right]_-$$

and Claim 2 follows.

To complete the main induction, we observe that

$$\mathbf{A}^{\alpha}_P(\emptyset) = \tilde{\mathbf{S}}_P(\tilde{\mathbf{S}}_P(\tilde{I})) \subseteq \tilde{\mathbf{S}}_{P'}(\overline{A^+}) = \tilde{A}$$

∎

Pulling the pieces together, we have:

Theorem 8.6: Let general programs $\mathbf{P}$ and $\mathbf{P}'$ be as described above. *Furthermore*, assume that $\mathbf{P}$ and $\mathbf{P}'$ are strict in the IDB, have the same sets of globally positive and globally negative relations, and that p is globally positive. Then for all ordinals α,

$$\begin{array}{lrcl} \text{(a)} & [\mathbf{A}^{\infty}_P(\emptyset)]_- & = & \mathbf{A}^{\infty}_{P'}(\emptyset) \\ \text{(b)} & [\mathbf{S}_P(\mathbf{A}^{\infty}_P(\emptyset))]_+ & = & \mathbf{S}_{P'}(\mathbf{A}^{\infty}_{P'}(\emptyset)) \end{array}$$

In particular, the globally positive relations of $\mathbf{P}$ and $\mathbf{P}'$ have the same positive literals in their respective AFP models.

Proof: If q is globally positive, then ϕ occurs positively in ψ, the rule body for p, and Lemma 8.3 applies. If q is globally negative, then ϕ occurs negatively in ψ, and the conclusion follows from Lemmas 8.4 and 8.5. ∎

Theorem 8.7: Let an FP system Ψ be transformed into a normal logic program $\mathbf{P}$ by a sequence of elementary simplifications. Then the positive part of the alternating fixpoint partial model of $\mathbf{P}$ agrees with the least fixpoint of Ψ on the relations of Ψ.

Proof: Elementary simplifications can only be applied to rules for relations in the IDB of Ψ, which are globally positive. By Theorem 8.6 the AFP model of Ψ agrees with the AFP model of $\mathbf{P}$ on the positive literals of the relations of Ψ, since all of these relations are globally positive. By Theorem 8.1 the positive part is the least fixpoint of Ψ interpreted in FP. ∎

8.4 Expressive Power on Finite Structures

We now show how a lemma due to Immermann can be used to prove that, on finite structures, the converse of Theorem 8.7 holds; i.e., FP can simulate the positive part of AFP in this case. (That it can also simulate the negative part (on finite structures) follows from other results in [20], as well as from [19, 26].) For simplicity of presentation we assume the AFP program $\mathbf{P}$ has a single IDB relation, $p(\vec{X})$.

The main difficulty is the conflicting notation. To restate Immermann's lemma, we introduce the notation $\lambda\vec{X}.\,\mu_p\phi[s;p](\vec{X})$ to denote the relation representing the least fixpoint of the rule

$$p(\vec{X}) \leftarrow \phi[s;p](\vec{X})$$

where p and s are relation names whose occurrences in ϕ are positive, and $\vec{X}$ is the vector of the free variables of ϕ. Note that p, being the subscript of μ, functions as the carrier of the induction; and s acts as a parameter.

Lemma 8.8: ([20, Lemma 4.7]) Let $\phi[s;p](\vec{X})$ and $\psi[q;r](\vec{U})$ be first order formulas, positive in (p,s) and (q,r), respectively. Then

$$\alpha(\phi,\psi) \stackrel{\text{def}}{=} \mu_p \phi[\lambda\vec{U}.\neg\mu_r\psi[\neg p;r](\vec{U});\ p](\vec{X})$$

is equivalent to an FP formula on finite structures. ∎

Let $\mathbf{P}$ be an AFP program for $p(\vec{X})$. We collect all the bodies of rules for p into a single first order formula, and replace all the negative p literals by positive s literals, giving $\phi[s;p](\vec{X})$. Let ψ be the same formula. Thus

$$\alpha(\phi,\phi) \stackrel{\text{def}}{=} \mu_p \phi[\lambda\vec{U}.\neg\mu_r\phi[\neg p;r](\vec{U});\ p](\vec{X})$$

Now in AFP, recall that $\tilde{\mathbf{S}}_P$ and $\mathbf{A}_P$ transform a set of negative literals into another set of negative literals. Thus the expression $\lambda\vec{U}.\neg\mu_r\phi[\neg p;r](\vec{U})$ corresponds to $H \Leftrightarrow \mathbf{S}_P(\overline{p})$ in AFP, so $\alpha(\phi,\phi)$ is the least fixpoint of the operator $Q(J)$ that operates on sets of positive literals, and is defined by:

$$Q(J) \stackrel{\text{def}}{=} \mathbf{T}_P(J + \tilde{\mathbf{S}}_P(\overline{J}))$$

Let $J_0 = \emptyset$ and $J_{n+1} \stackrel{\text{def}}{=} Q(J_n)$. Then $J_\infty \stackrel{\text{def}}{=} \bigcup_n J_n = \alpha(\phi,\phi)$. (The closure ordinal is finite on finite structures.)

The positive part of the AFP model can be expressed in terms of the closure of a slightly different operator.

$$Q_A(I) \stackrel{\text{def}}{=} \mathbf{S}_P(\tilde{\mathbf{S}}_P(\overline{I}))$$

Clearly, Q_A is monotonic. Let $I_0 = \mathbf{S}_P(\emptyset)$, $I_{n+1} \stackrel{\text{def}}{=} Q_A(I_n)$, and $I_\infty \stackrel{\text{def}}{=} \bigcup_n I_n$.

Lemma 8.9: I_∞, as defined above, is the positive part of the AFP model of $\mathbf{P}$.

Proof: By definitions, $I_{n+1} = \mathbf{S}_P(\tilde{\mathbf{S}}_P(\overline{I_n}))$. A trivial induction shows that $I_n = \mathbf{S}_P(\mathbf{A}_P^n(\emptyset))$. ∎

Theorem 8.10: For finite structures, the positive part of the alternating fixpoint partial model of $\mathbf{P}$ has a representation in fixpoint logic (FP).

Proof: We shall show that the above relation $\alpha(\phi,\phi)$ equals I_∞, that is, $J_\infty = I_\infty$, as defined above. The theorem then follows by Lemmas 8.8 and 8.9.

By definition of $\mathbf{S}_P$, for any set of positive literals I:

$$I = Q_A(I) \Leftrightarrow I = \mathbf{S}_P(\tilde{\mathbf{S}}_P(\overline{I})) \Rightarrow I = \mathbf{T}_P(I + \tilde{\mathbf{S}}_P(\overline{I})) \Leftrightarrow I = Q(I)$$

Thus I_∞ is a fixpoint of Q, so $I_\infty \supseteq J_\infty$.

To complete the proof, we claim that $I_n \subseteq J_\infty$ for all n. Clearly this holds for I_0. For any n, suppose $I_n \subseteq J_\infty$. Then $\tilde{\mathbf{S}}_P(\overline{I_n}) \subseteq \tilde{\mathbf{S}}_P(\overline{J_\infty})$, and $I_{n+1} = \mathbf{S}_P(\tilde{\mathbf{S}}_P(\overline{I_n})) \subseteq \mathbf{S}_P(\tilde{\mathbf{S}}_P(\overline{J_\infty}))$ by monotonicity of $\mathbf{S}_P$. But $J_\infty = \mathbf{T}_P(J_\infty + \tilde{\mathbf{S}}_P(\overline{J_\infty}))$ implies that $\mathbf{S}_P(\tilde{\mathbf{S}}_P(\overline{J_\infty})) \subseteq J_\infty$, by Corollary 3.2. Thus $I_{n+1} \subseteq J_\infty$, establishing the claim by induction. ∎

8.5 Complement of Finite Transitive Closure

Even when the theoretical expressive powers of two languages are equal, there is a substantial practical issue of usability. While this issue can only be resolved through experience, it appears that FP requires great contortions to express certain concepts that are straightforward in AFP.

For example, *ntc*, the complement of the transitive closure, (on a finite graph with edge relation e) is expressed naturally and concisely in AFP (and the stratified semantics) by

$$\begin{aligned} tc(X,Y) &\leftarrow e(X,Y). \\ tc(X,Y) &\leftarrow e(X,Z),\ tc(Z,Y). \\ ntc(X,Y) &\leftarrow \neg tc(X,Y). \end{aligned}$$

Kunen has addressed the problem of expressing this concept in the 3-valued logical consequence semantics by a program without function symbols [25]. He shows that *no* strict program (see Definition 8.3) is able to express this concept (again, with the restriction to finite structures, which in logic programs means absence of function symbols). Kunen left as an open question whether such a nonstrict program exists.

It is known that *ntc* is expressible in FP, but it requires a system that includes several intermediate relations, some of them 4-ary, and it is far from obvious by a casual inspection what the system computes. The simplest FP program for *ntc* that we know of is based on a construction by John Schlipf [43]; some earlier constructions used 6-ary predicates. He has also shown that, by a sequence of elementary simplifications, *any* FP system can be reduced to a normal logic program that is correctly interpreted (on finite structures) by Fitting's semantics. This is not identical to Kunen's 3-valued logical consequence semantics, even on finite structures, so Kunen's question is still open.

As previously discussed, the transformation of an FP system into a normal logic program by a sequence of elementary simplifications creates a program that is strict in the IDB (including the ADB). Thus the only possible nonstrictness in this situation is with respect to the EDB. In Schlipf's FP definition, *ntc* depends both negatively *and* positively on e. Intuitively, one might expect only negative dependence because the mapping from e to *ntc* is antimonotonic.

To summarize this discussion, it is hopeless to expect that any FP expression for the complement of transitive closure will be considerably simpler or more straightforward than those presently known.

9 Conclusion

The alternating fixpoint provides additional insights into the well-founded semantics, and its relation to stable models. It offers a constructive definition for the well-founded partial model, and an extension of fixpoint logic. The use of monotonic transformations in the definitions has made the study of its properties tractable, and seems to be related to the simplicity with which concepts can be expressed. Future work should explore how to identify classes of unstratified programs and queries on them for which the alternating fixpoint semantics is computationally tractable; successively more tractable classes of interest are recursively enumerable, decidable, and polynomial. Another avenue is further study of programs that are not strict in the IDB.

Acknowledgements

We thank Phokion Kolaitis and John Schlipf for many helpful discussions of this work. We thank the referees for many helpful suggestions, and for bringing the paper by Elkan [14] to our attention. This work was supported in part by NSF grants CCR-8958590 and IRI-8902287.

References

[1] S. Abiteboul and V. Vianu. Procedural languages for database queries and updates. *Journal of Computer and System Sciences*, 41(2):181–229, 1990. Preliminary version appeared in 1988 ACM Symposium on Principles of Database Systems.

[2] P. Aczel. An introduction to inductive definitions. In J. Barwise, editor, *Handbook of Mathematical Logic*, pages 739–782. North-Holland, New York, 1977.

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

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

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

[6] N. Bidoit and C. Froidevaux. Negation by default and nonstratifiable logic programs. Technical Report 437, Laboratoire de Recherche en Informatique, Orsay, France, 1988.

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

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

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

[10] P. Cholak. Post correspondence problem and Prolog programs. Technical report, Univ. of Wisc., Madison, 1988. (manuscript).

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

[12] J. Doyle. A truth maintenance system. *Artificial Intelligence*, 12:231–272, 1979.

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

[14] C. Elkan. A rational reconstruction of nonmonotonic truth maintenance systems. *Artificial Intelligence*, 43(2):219–234, 1990.

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

[16] M. Fitting and M. Ben-Jacob. Stratified and three-valued logic programming semantics. In *Fifth Int'l Conf. Symp. on Logic Programming*, pages 1054–1069, Seattle, 1988.

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

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

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

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

[21] P. G. Kolaitis. The expressive power of stratified programs. *Information and Computation*, 1991. (to appear; also available as UCSC-CRL-89-14 from UC Santa Cruz).

[22] P. G. Kolaitis and C. H. Papadimitriou. Why not negation by fixpoint? In *ACM Symposium on Principles of Database Systems*, pages 231–239, 1988.

[23] R. A. Kowalski. *Logic for Problem Solving.* North-Holland, Amsterdam, 1979.

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

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

[26] D. Leivant. Inductive definitions over finite structures. *Information and Computation*, 89(2):95–108, 1990. Also available from author as CMU–CS–89–153.

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

[28] Vladimir Lifschitz. Between circumscription and autoepistemic logic. In *First International Conference on Principles of Knowledge Representation and Reasoning*, pages 235–244. Morgan Kaufmann, 1989.

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

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

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

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

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

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

[35] H. Przymusinska and T. Przymusinski. Semantic issues in deductive databases and logic programs. In R. Banerji, editor, *Formal Approaches to Artificial Intelligence: A Sourcebook*. North-Holland, New York, 1990.

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

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

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

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

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

[41] K. A. Ross. Modular stratification and magic sets for Datalog programs with negation. In *Ninth ACM Symposium on Principles of Database Systems*, pages 161–171, 1990.

[42] D. Saccà and C. Zaniolo. Partial models, stable models and non-determinism in logic programs with negation. Technical report, MCC, Austin, TX, January 1990. (Extended abstract appeared in Ninth ACM Symposium on Principles of Database Systems, 1990.).

[43] J. S. Schlipf. Inductive definability and semantics of logic programs. Technical report, Univ. of Cincinnati, 1989. (manuscript).

[44] J. S. Schlipf. Formalizing a logic for logic programming. In *International Symposium on Artificial Intelligence and Mathematics*, 1990.

[45] J. S. Schlipf. The expressive powers of the logic programming semantics. In *Ninth ACM Symposium on Principles of Database Systems*, pages 196–204, 1990.

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

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

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

[49] A. Van Gelder. The alternating fixpoint of logic programs with negation. In *Eighth ACM Symposium on Principles of Database Systems*, pages 1–10, 1989.

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

[51] A. Van Gelder, K. A. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 1991. (to appear). Available from first author as UCSC-CRL-89-38. Preliminary abstract appeared in Seventh ACM Symposium on Principles of Database Systems, 1988.

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

[53] J.-H. You and L. Y. Yuan. Three-valued formalization of logic programming: Is it needed? In *Ninth ACM Symposium on Principles of Database Systems*, pages 196–204, 1990.