Common-Sense Axiomatizations for Logic Programs

Allen Van Gelder John S. Schlipf*

Abstract

Various semantics for logic programs with negation are described in terms of a dualized program together with additional axioms, some of which are second order formulas. The semantics of Clark, Fitting, and Kunen are characterized in this framework, and a finite first-order presentation of Kunen's semantics is described. A new axiom to represent "common sense" reasoning is proposed for logic programs. It is shown that the well-founded semantics and stable models are definable with this axiom. The roles of domain augmentation and domain closure are examined. A "domain foundation" axiom is proposed to replace the domain closure axiom.

1 Introduction

Although the semantics of Horn logic programs is standard, as given in the seminal work of van Emden and Kowalski [47], there is presently no universally accepted semantics for logic programs with negation. The first purpose of this paper is to describe various existing proposals using a common framework of classical (two-valued) logic, thereby to delineate their differences more clearly. A second purpose is to explore the effects of various constraints on the universe (or domain) of interpretation on the logical consequences of the program.

Our approach is to identify the program's "declarative semantics" with the *logical consequences* of the rules in the program, together with various additional axioms. Various sets of additional axioms give rise to various semantics. We avoid the use of procedural definitions and nonstandard logics; in some cases we use second order formulas, since they can be very useful in distinguishing concepts.

We introduce a new form of common-sense axiom for logic programs and show its relationship to the recently proposed semantics based on stable models and well-founded models. We shall discuss and compare the earlier semantics proposed by Clark [9], Fitting [11], and Kunen [19], as well as the more recent ones identified with stable models [12] and well-founded models [50].

**Authors' addresses*: Allen Van Gelder, Baskin Computer Science Center, 225AS, University of California, Santa Cruz, CA 95064, USA (`avg@cs.ucsc.edu`) and John S. Schlipf, Computer Science Dept., University of Cincinnati, Cincinnati, OH 45221 USA (`schlipf@jschlipf.csm.uc.edu`)

For the semantics of logic programs it is traditional to restrict attention to Herbrand models. For a definite (Horn, or negation-free) logic program, one makes the same inferences by restricting attention to Herbrand models as one would by considering all models (as in first order logic). However, for logic programs with negation this restriction *does* affect the resulting inferences, and may lead to a number of unnatural features. For example, in modularized software, the interpretation of the rules in a module should not exclude the possibility of objects not known to the module. Some researchers have suggested weakening this domain assumption. In this paper we shall consider various weaker forms of that domain assumptions, and we shall investigate how the choice of domain assumption affects the axiomatization of logic programming semantics described above.

1.1 Related Work

It has long been accepted that a logic program, particularly one with negative subgoals, carries a certain amount of implicit information. Clark was the first to formalize a method to make this implicit content explicit, by defining a *completed logic program* to be associated with the original [9]. Reiter's *closed world assumption* is somewhat related [37]. Independently, McCarthy proposed the concept of *circumscription* to capture the implicit "common sense" element in any first order sentence (not necessarily a logic program) that was intended roughly as a statement of "knowledge about the world" [28, 29]. Whereas Clark remained within a first order framework, McCarthy used a second order formula. To a large extent, subsequent research has refined one of these fundamental approaches in an attempt to cure various perceived problems in circumscription [22, 31, 24, 23, 17, 18] or program completion [45, 11, 19, 26, 46, 20].

Stratified semantics [3, 48] and its generalizations, such as stable models [12] and well-founded semantics [50], constitute a vein that has seen considerable recent activity [34, 8, 15, 35, 25, 38, 44, 49, 33, 42, 40, 43, 6]. One of the main motivations for this type of semantics is that complements of many inductive definitions have their natural expression, a feature lacking in all of the program completion approaches. Example 5.1 illustrates this idea with the complement of transitive closure of a directed graph.

Two other active areas deserve mention, although they are not closely related to the issues studied in this paper. One is the study of logic programs without function symbols, often called deductive databases [7, 13, 14, 1, 16, 10, 21, 15]. Another is the study of disjunctive logic programs [30, 41, 36, 39, 33].

1.2 Summary of Results

In Section 5 we propose a new second order axiom that formalizes "common sense" differently from traditional circumscription. Rather than minimality, it addresses lack of support. One form of this axiom leads to stable models; another leads to well-founded models.

In Section 6 we show that certain infinite sets of axioms that are customarily used to constrain the universes of models have finite presentations, a fact that may facilitate implementations and automated proofs of properties.

Stable and well-founded models, as presented in previous literature, are always defined given fixed universes, almost always Herbrand universes. Thus obtaining either stable or well-founded models requires an additional axiom to constrain the universe (or domain) of interpretation. One sufficient constraint is that the universe be (an isomorphic copy of) the Herbrand universe. As noted above, in many settings such a constraint is unnatural. Therefore, an alternative is investigated in Section 7, which we call the *domain foundation axiom*. This axiom does not specify any particular universe, though it specifies that some of the properties of the Herbrand universe hold. We show that the domain closure requirement can be "factored" into the domain foundation axiom and a first order nameability axiom.

Figure 1 summarizes the relationships found among various axiomatizations studied in the paper.

2 Preliminaries and Terminology

We assume familiarity with the common terminology of logic programming, but review some specifics, and describe our notation and basic definitions.

2.1 Rule Format

A *Horn* logic program may be thought of as a finite set of *rules*, exactly one for each predicate symbol, in the form:

$$q_i(\vec{x_i}) \leftarrow \phi_i(\vec{x_i}) \qquad i = 1, \ldots, n$$

where each $\vec{x_i}$ is a vector of distinct variables, $x_i^1, x_i^2, \ldots$, formula $q_i(\vec{x_i})$, called the *head* of the rule, is an atomic formula, and $\phi_i(\vec{x_i})$, called the *body* of the rule, is a positive, existential, formula of first order logic with equality, whose free variables are exactly $\vec{x_i}$. That is, the rule body is built from atomic formulas (atoms) connected by logical *and* (&), *or* ($\vee$), and existential quantifiers. The propositional constants **true** and **false** may also be used for rule bodies. The backwards implication symbol ($\leftarrow$) is conveniently read as "if". Square brackets are sometimes used as well as parentheses to delimit scope. Should some predicate q_i not otherwise appear in the head of a rule, we shall have the vacuous rule $q_i \leftarrow$ **false**. The reason for this slightly unorthodox description will soon become evident.

Definition 2.1: The rule bodies of a program may contain function symbols (of arity ≥ 1) and constant symbols. The set of all such symbols is called the *functional vocabulary* (or just "vocabulary") of the program, denoted L_P. Frequently, we shall consider functional vocabularies L_i containing additional symbols not present in the program. Several symbols will require the vocabulary L_i as a modifier (e.g., $\mathbf{F}$, to be defined later). To avoid double subscripts, we just use the subscript of L on the other symbol (e.g., $\mathbf{F}_i$ instead of $\mathbf{F}_{L_i}$). Also, the absence of a subscript indicates the symbol is associated with L_P. □

Example 2.1: The following program defines e as an directed edge relation on three nodes, a, b, and c, and defines p as its transitive closure.

$$p(x,y) \leftarrow e(x,y) \vee \exists z\,[p(x,z) \,\&\, p(z,y)]$$
$$e(x,y) \leftarrow [(x=a) \,\&\, (y=b)] \vee [(x=b) \,\&\, (y=a)] \vee [(x=c) \,\&\, (y=c)]$$

The functional vocabulary of the program is $L_P = \{a, b, c\}$.

In deductive database settings e is called an *extensional database* predicate because it can be rewritten as a finite set of ground facts, $e(a,b)$, $e(b,a)$, and $e(c,c)$. In such settings, the rule for e is often regarded as part of the input, not part of the program. For this paper, the rule for e is part of the program, and indeed, *every* predicate must have exactly one rule in the program. □

There are several natural ways to extend the Horn rule format to introduce nonmonotonicity. We are primarily concerned with the extension to *normal* logic programs, in the terminology of Lloyd [26]. This extension permits rule bodies to be built from positive and negative literals (where Horn rules are limited to atoms, or positive literals), connected by "and", "or", and existential quantifiers; essentially, only universal quantifiers are absent. A closely related extension, *general* logic programs, permits rule bodies to be any first order formula. (Lloyd and Topor used the terms "generalized clause" and "extended clause" where we use "normal" and "general" [27].) A quite different road is taken with *disjunctive* logic programs, which permit the head of a rule to be a disjunction of atoms [30].

Although we shall *transform* a normal logic program into a form that contains first order formulas, we wish to emphasize that the *original* program consists of normal rules only.

2.2 Notational Conventions

Some additional notational conventions follow. In all cases, when it is clear from the context, the symbols may represent *vectors* of the designated objects; e.g., ϕ often represents a vector of formulas, R often represents a vector of predicate variables, etc.

- generic formulas are denoted by ψ, ϕ;
- generic terms are s, t, u;
- generic n-ary function symbols ($n \geq 1$) are f, g;
- generic constant symbols are a, b, c;
- when the context makes it clear, ground (variable-free) terms are also written as a, b, c;
- variables (ranging over individuals) are $v, \ldots, z$; free variables are local to individual formulas when vectors of formulas are present;

- *predicate variables* of second order formulas are written using uppercase letters $R, \ldots, Z$.

- generic program predicates are denoted with q.

- As usual, the symbols $=$ and $\neq$ are the logical symbols for true equality and inequality of elements of the universe.

In program examples, any lowercase letter (or word) may represent a predicate or a function symbol, when the syntax dictates this meaning. We emphasize again that each relation symbol q_i must appear in the head of a unique rule $q_i(x) \leftarrow \phi_i(x)$.

Several notations that are introduced as they are needed are mentioned here for ease of cross reference.

- The notation $\tilde{q}$ is defined in Definition 3.1. Here we emphasize that $\sim$ is not a vector notation, and is not an operator.

- The *immediate consequence* operator $\mathbf{T}_{P\tilde{P}}$ is introduced in Definition 3.4.

- *Functional vocabulary* notations, including the avoidance of double subscripting, are discussed in Definition 2.1.

- A unary predicate whose rule defines the Herbrand universe is denoted by $h(x)$ and its rule body is $\phi_H(x)$ (Definition 4.2).

- The formula $\mathbf{F}(x, y)$ is used to define *functional edges* (Definition 6.1).

- The formula $\phi_w(x)$ is used for the domain foundation axiom (Definition 7.2).

To streamline notation when no confusion will arise, we may drop subscripts and omit arrows over vectors. For example, the n generic program rules may be abbreviated to

$$q(x) \leftarrow \phi(x)$$

By doing so we do not intend to limit the discussion to programs with a single unary predicate. Similarly, R normally denotes an n-vector of predicate variables, with each R_i corresponding to q_i in arity.

Definition 2.2: It is necessary to have a concise notation for syntactic substitutions in formulas. If $\phi(x)$ is a formula, then the notation

$$\phi[q_1/\psi_1, \ldots, q_k/\psi_k](x)$$

denotes the formula obtained by replacing each occurrence of the predicate symbol q_i by the formula ψ_i with appropriate unification. That is, if an occurrence of q_i has as arguments the k-vector of terms t, then ψ_i has k free variables, y, and $\psi_i(t)$ is substituted for $q_i(t)$. In usage, ψ_i will usually be a predicate variable, R_i, or a predicate constant **true** or **false**.

As usual, if q is the vector $(q_1, \ldots, q_n)$ we write $\phi[q/R]$ for $\phi[q_1/R_1, \ldots, q_n/R_n](x)$. Similarly ϕ may denote a vector of formulas, each receiving the same substitution. $\square$

As a final notational shorthand, we shall use set operations to abbreviate certain second order formulas. A typical example is $R \subseteq \phi$ to abbreviate $\forall x[R(x) \rightarrow \phi(x)]$, which in turn is really an abbreviation for

$$\mathop{\&}_{i=1}^{n} \forall \vec{x_i}[R_i(\vec{x_i}) \rightarrow \phi_i(\vec{x_i})]$$

3 Dualized Programs

Clark introduced the idea of replacing an "if" rule by its "if and only if" counterpart. We introduce *dualized* rules and programs to achieve a somewhat weaker effect.[1] The dualized program serves as a common point of departure for various semantics. Although the *original* program consists of normal rules only, i.e., rules whose bodies are existentially quantified conjunctions of literals, the *dualized* program may contain universal quantifiers and disjunctions.

The first step of dualization is replace each negative literal $\neg q(t)$ in the rule body by $\tilde{q}(t)$; $\neg(t = u)$ is replaced by $t \neq u$. Since $\tilde{q}$ and $\neq$ are regarded as new predicate symbols, this actually removes all negation from the program. We call q and $=$ *positive* predicates and call $\tilde{q}$ and $\neq$ *tilde* predicates. The rules that result, denoted as a set by $\mathbf{P}$, are called the *positive* rules.

Definition 3.1: For any formula ψ in which negation is absent, the *dual* of ψ, denoted by $\tilde{\psi}$, is defined inductively:

1. Atoms $q(t)$ and $\tilde{q}(t)$ are duals; $(t = u)$ and $(t \neq u)$ are duals; **true** and **false** are duals.
2. The dual of $(\psi \,\&\, \phi)$ is $(\tilde{\psi} \vee \tilde{\phi})$; the dual of $(\psi \vee \phi)$ is $(\tilde{\psi} \,\&\, \tilde{\phi})$.
3. The dual of $\exists x\, \psi(x)$ is $\forall x\, \tilde{\psi}(x)$; the dual of $\forall x\, \psi(x)$ is $\exists x\, \tilde{\psi}(x)$.

The definition extends to all formulas by first pushing negations down to the atoms, then replacing $\neg q$ by $\tilde{q}$, and finally applying the above definition to construct the dual.

The *tilde rule* corresponding to positive rule $q(x) \leftarrow \phi(x)$ is $\tilde{q}(x) \leftarrow \tilde{\phi}(x)$, where $\tilde{\phi}(x)$ is the dual formula of $\phi(x)$. The set of tilde rules corresponding to $\mathbf{P}$ is denoted $\tilde{\mathbf{P}}$. $\square$

The tilde rule $\tilde{q}(x) \leftarrow \tilde{\phi}(x)$ can be thought of as the "only if" version of $q(x) \leftarrow \phi(x)$. The initial form of the "only if" rule is $q(x) \rightarrow \phi(x)$. However, its contrapositive form is $\neg q(x) \leftarrow \neg\phi(x)$, which leads to the tilde rule by pushing down negations in the rule body and replacing negative literals by tilde atoms.

[1] "Doubled" programs used by other researchers are not related; they have a second copy of the *same* rules for bookkeeping purposes.

Example 3.1: The Horn program for transitive closure of Example 2.1 has the tilde rules below. Note the universal quantifier.

$$\tilde{p}(x,y) \leftarrow \tilde{e}(x,y) \;\&\; \forall z\, [\tilde{p}(x,z) \vee \tilde{p}(z,y)]$$
$$\tilde{e}(x,y) \leftarrow [(x \neq a) \vee (y \neq b)] \;\&\; [(x \neq b) \vee (y \neq a)] \;\&\; [(x \neq c) \vee (y \neq c)]$$

Observe that the rule for $\tilde{p}$ is recursive with no base case. □

Traditionally in logic programming, distinct ground terms are assumed to name distinct objects. With ordinary definite clause programs (not involving $=$ or $\neq$), this assumption makes no difference in traditional semantics. Here, due to the introduction of universal quantifiers and the presence of "$\neq$", it is important to rule out models in which syntactically different terms collapse to the same interpreted object. The standard way to do this is to append the Clark Equality Theory, which forces syntactically different variable-free terms to be interpreted as different objects [9, 4, 19, 26]. This is achieved by introducing an infinite recursive set of first order sentences, which depend on the functional vocabulary of the program. They are often called the *Clark Equality Axioms* (CEA) or the *equality freeness axioms*.

Definition 3.2: The Clark equality axioms (CEA) for a given functional vocabulary L consist of the following inequalities. For simplicity of presentation, we give them for a vocabulary that contains just one constant a, one unary function g, and one binary function symbol f.

1. The *global injectivity constraint* requires all functions to be injective (1-1) and to have disjoint ranges, where constants are treated as 0-ary functions:

$$\begin{aligned}
\forall x_1 x_2 \;\; & [x_1 \neq x_2 \rightarrow (g(x_1) \neq g(x_2))] \\
\forall x_1 y_1 x_2 y_2 \;\; & [(x_1 \neq x_2 \vee y_1 \neq y_2) \rightarrow f(x_1,y_1) \neq f(x_2,y_2)] \\
\forall x \;\; & [g(x) \neq a] \\
\forall xy \;\; & [f(x,y) \neq a] \\
\forall xyz \;\; & [f(x,y) \neq g(z)]
\end{aligned}$$

   For a finite vocabulary, this constraint consists of a finite number of axioms.

2. The *acyclicity constraint* consists of axioms that prevent any term being equal to a proper subterm of itself. This constraint is usually presented as an infinite set of inequalities, even for finite vocabularies [9, 26]:

$$\begin{array}{ll}
\forall x \;\; [x \neq g(x)] & \forall xy \;\; [x \neq f(x,y) \;\&\; x \neq f(y,x)] \\
\forall x \;\; [x \neq g(g(x))] & \forall xy \;\; [x \neq f(g(x),y) \;\&\; x \neq f(y,g(x))] \\
\forall xy \;\; [x \neq g(f(x,y)) \;\&\; x \neq g(f(y,x))] & \forall xyz \;\; [x \neq f(f(x,y),z) \;\&\; x \neq f(f(y,x),z) \\
 & \quad \;\&\; x \neq f(z,f(x,y)) \;\&\; x \neq f(z,f(y,x))] \\
\quad \ldots & \quad \ldots
\end{array}$$

It will be instructive later (Section 6) to study a finite presentation of the acyclicity constraint.

The generalization to arbitrary vocabularies should be obvious. Unmodified, the abbreviation CEA denotes the set of Clark equality axioms for L_P, the functional vocabulary of the program under consideration. We shall have occasion to phrase CEA using different vocabularies; for $L_{\#}$ we use CEA#, for L_*, CEA$*$, for L_1, CEA$_1$, etc. □

Note that if L_1, L_2 are two vocabularies whose Clark Equality axioms are CEA$_1$ and CEA$_2$, respectively, then $L_1 \subseteq L_2$ if and only if CEA$_1 \subseteq$ CEA$_2$.

Definition 3.3: The *dualized program* associated with a given logic program $\mathbf{P}$, is denoted by $(\mathbf{P}, \tilde{\mathbf{P}}, \mathrm{CEA})$; it is the set of formulas consisting of the positive and tilde rules, together with the Clark equality axioms. □

Example 3.2: The original rules are:

$$\begin{aligned} p(x) &\leftarrow e(x) \;\&\; \exists y\, d(x,y) \\ d(x,y) &\leftarrow \neg(x = y) \\ e(x) &\leftarrow (x = a) \end{aligned}$$

Intuitively, d is intended to express the property that x and y are distinct. The dualized program consists of:

$$\begin{aligned} p(x) &\leftarrow e(x) \;\&\; \exists y\, d(x,y) \\ d(x,y) &\leftarrow (x \neq y) \\ e(x) &\leftarrow (x = a) \end{aligned}$$

$$\begin{aligned} \tilde{p}(x) &\leftarrow \tilde{e}(x) \vee \forall y\, \tilde{d}(x,y) \\ \tilde{d}(x,y) &\leftarrow (x = y) \\ \tilde{e}(x) &\leftarrow (x \neq a) \end{aligned}$$

plus CEA.

If the dualized program is interpreted in a universe of one element, then $\tilde{p}(a)$ must be true; whereas if it is interpreted in a larger universe, then $p(a)$ must be true. Note that neither $p(a)$ nor $\tilde{p}(a)$ is a logical consequence of the dualized program. This program is discussed further in later examples. □

Although all rule bodies are positive formulas in the dualized program, $(\mathbf{P}, \tilde{\mathbf{P}})$ is not generally a Horn program due to the presence of universal quantification. However, it *is* an inductive system as studied by Moschovakis [32]. The familiar *immediate consequence* operator associated with the dualized program may be defined using the formula named $\mathbf{T}_{P\tilde{P}}$ below. Since $\mathbf{T}_{P\tilde{P}}$ is defined on the dualized program, it is the analogue of Fitting's three-valued immediate consequence operator Φ_P [11], but stated in the language of dualized programs. We discuss Fitting's semantics in Section 4. With some abuse of notation we use the same symbol for the operator as for its defining formula. This operator is monotonic and has a least fixpoint, denoted by $\mathbf{T}^{\infty}_{P\tilde{P}}$, in any structure for $(\mathbf{P}, \tilde{\mathbf{P}}, \mathrm{CEA})$.

Definition 3.4: Let $\mathbf{P}$ be a program given by the rules $q(x) \leftarrow \phi(x)$, where q is an n-vector of predicate symbols. (Recall that x is a vector $(x_1, \ldots, x_n)$ of vectors of individual variables; each x_i has the arity of q_i.) Let y have the same arities as x, and let R and $\tilde{R}$ (n-vectors of predicate variables) have the same arities as q. Then,

$$\mathbf{T}_{P\tilde{P}}(R, \tilde{R})(x, y) \stackrel{\text{def}}{=} (\phi[q/R, \tilde{q}/\tilde{R}](x),\ \tilde{\phi}[q/R, \tilde{q}/\tilde{R}](y))$$

that is, the vector of second order formulas:

$$\begin{array}{l}
\phi_1[q/R, \tilde{q}/\tilde{R}](x_1) \\
\ldots \\
\phi_n[q/R, \tilde{q}/\tilde{R}](x_n) \\
\tilde{\phi}_1[q/R, \tilde{q}/\tilde{R}](y_1) \\
\ldots \\
\tilde{\phi}_n[q/R, \tilde{q}/\tilde{R}](y_n)
\end{array}$$

The *immediate consequence* operator for $(\mathbf{P}, \tilde{\mathbf{P}})$ is defined by this formula in the obvious way: For input relations $(r, \tilde{r})$ of the same arities as q over some structure, the output of $\mathbf{T}_{P\tilde{P}}$ (viewed as an operator) is the $2n$-vector of relations, the sets of all tuples that satisfy $\phi_1, \ldots, \phi_n, \tilde{\phi}_1, \ldots, \tilde{\phi}_n$ when R are interpreted by r and $\tilde{R}$ are interpreted by $\tilde{r}$. $\square$

Example 3.3: If $\mathbf{P}$ is the transitive closure program of Example 2.1, whose dualized rules were shown in Example 3.1, then the formula $\mathbf{T}_{P\tilde{P}}$ is built as follows. With some renaming of free variables, the body of the rule for p is:

$$\phi_1(x_1^1, x_1^2) = e(x_1^1, x_1^2) \vee \exists z\, [p(x_1^1, z)\ \&\ p(z, x_1^2)]$$

Thus,

$$\phi_1[p/R_1, e/R_2, \tilde{p}/\tilde{R}_1, \tilde{e}/\tilde{R}_2](x_1^1, x_1^2) = R_2(x_1^1, x_1^2) \vee \exists z\, [R_1(x_1^1, z)\ \&\ R_1(z, x_1^2)]$$

Repeating this substitution for ϕ_2 (the body of the rule for e), $\tilde{\phi}_1$ and $\tilde{\phi}_2$, we obtain:

$$\mathbf{T}_{P\tilde{P}}(R_1, R_2, \tilde{R}_1, \tilde{R}_2)(x_1^1, x_1^2, x_2^1, x_2^2, y_1^1, y_1^2, y_2^1, y_2^2) =$$
$$\left(\begin{array}{c}
R_2(x_1^1, x_1^2) \vee \exists z\, [R_1(x_1^1, z)\ \&\ R_1(z, x_1^2)] \\
[(x_2^1 = a)\ \&\ (x_2^2 = b)] \vee [(x_2^1 = b)\ \&\ (x_2^2 = a)] \vee [(x_2^1 = c)\ \&\ (x_2^2 = c)] \\
\tilde{R}_2(y_1^1, y_1^2)\ \&\ \forall z\, [\tilde{R}_1(y_1^1, z) \vee \tilde{R}_1(z, y_1^2)] \\
[(y_2^1 \neq a) \vee (y_2^2 \neq b)]\ \&\ [(y_2^1 \neq b) \vee (y_2^2 \neq a)]\ \&\ [(y_2^1 \neq c) \vee (y_2^2 \neq c)]
\end{array}\right)$$

The free variables of each formula have the same name except for superscripts. We hope this example illustrates our motivation to streamline the notation!

The associated immediate consequence operator transforms a 4-vector of binary relations into another 4-vector of binary relations. To evaluate the operator, interpret $(R_1, R_2, \tilde{R}_1, \tilde{R}_2)$ as the input 4-vector. These 4-vectors intuitively correspond to $(p, e, \tilde{p}, \tilde{e})$. □

4 Program Completion Semantics

From the dualized program we can re-establish the connection between duals and negation with the aid of two additional first order axioms.

Definition 4.1: The *disjointness* and *totality* axioms are

$$\begin{array}{rcl} \mathit{disjoint} & \stackrel{\text{def}}{=} & \neg\exists x\,[q(x)\ \&\ \tilde{q}(x)] \\ & \equiv & (q \cap \tilde{q} = \emptyset) \\ \mathit{total} & \stackrel{\text{def}}{=} & \forall x\,[q(x) \vee \tilde{q}(x)] \end{array}$$

We remind the reader that the axioms are presented in abbreviated form for programs with several predicates; the unabbreviated forms would have the appropriate conjunctions and subscripts. □

Clark proposed a semantics based on the *completed program*, which in our notation, adds *disjoint* and *total* to the dualized program:

$$\mathit{comp}(\mathbf{P}) \stackrel{\text{def}}{=} (\mathbf{P}, \tilde{\mathbf{P}}, \text{CEA}, \mathit{disjoint}, \mathit{total})$$

Sentences that are *logical consequences* of the completed program are regarded as true. For query-answering purposes, associate $\tilde{q}$ with $\neg q$. Recall that a formula is a *logical consequence* of a set of axioms if and only if it is true in *all* models of those axioms (not just Herbrand models).

One problem with the completed program is that it might be inconsistent. While the examples of this phenomenon, such as $p \leftarrow \neg p$, might appear silly as programs, inconsistency can arise in quite reasonable programs. For this single rule, the dualized rules are $p \leftarrow \tilde{p}$ and $\tilde{p} \leftarrow p$. The dualized program has two models, but $\mathit{comp}(\mathbf{P})$ is inconsistent.

However, it is straightforward to show that *disjoint* can cause inconsistency only in conjunction with *total*; that is, an easy induction argument shows that the least fixpoint of $\mathbf{T}_{P\tilde{P}}$ on any structure that satisfies CEA is a model of $(\mathbf{P}, \tilde{\mathbf{P}}, \text{CEA}, \mathit{disjoint})$.

Fitting and Kunen considered variations of program completion that essentially discarded *total*, although their work was presented in terms of three-valued logic. With *total* gone, a simple argument shows that the addition of *disjoint* to the dualized program does not constrain the logical consequences: the least fixpoint of $\mathbf{T}_{P\tilde{P}}$ on the Herbrand universe is always a model of the dualized program that satisfies *disjoint*. So *disjoint* may be ignored also, and we are back to the dualized program $(\mathbf{P}, \tilde{\mathbf{P}}, \text{CEA})$.

It is easy to see that $\mathbf{T}_{P\tilde{P}}$ as an operator corresponds to Fitting's three-valued immediate consequence operator Φ_P with just a change of terminology. Just associate atoms $\tilde{q}$ here with negative literals $\neg q$, and observe that $\neg q$ belongs to the output of Φ_P just when $\tilde{q}$ belongs to the output of $\mathbf{T}_{P\tilde{P}}$. There are two important differences between the Fitting and Kunen semantics, one of which is quite subtle.

4.1 Domain Closure Axioms

The *Fitting semantics* essentially limits the domain of interpretation to the Herbrand universe; it specifies that the true facts are precisely those in $\mathbf{T}^{\infty}_{P\tilde{P}}$, the least fixpoint of the immediate consequence operator, on that universe [11]. We now describe this semantics in terms of logical consequences of a second order formula.

Membership in the Herbrand universe can be defined inductively. For each finite functional vocabulary L there is a rule $h(x) \leftarrow \phi_H(x)$ whose least fixpoint defines the Herbrand universe (on structures that satisfy CEA, up to isomorphism).

Definition 4.2: The construction of ϕ_H depends on the set of symbols of L, but is purely mechanical. For example, if the program has just a constant a, a unary function g and binary function f:

$$\phi_H(x) \stackrel{\text{def}}{=} (x = a) \vee \exists y\,[(x = g(y)) \;\&\; h(y)] \vee \exists yz\,[(x = f(y,z)) \;\&\; h(y) \;\&\; h(z)]$$

□

As is well-known, the inductive closure of $h(x) \leftarrow \phi_H(x)$ is not first order definable, but *is* expressed by the second order formula

$$\forall x\,[h(x) \leftrightarrow \phi_H(x)] \;\&\; \forall R\,[(\phi_H[h/R] \subseteq R) \rightarrow h \subseteq R]$$

The first conjunct states that h is *a* fixpoint, and the second conjunct (the second order part of the formula) constrains h to be contained in *any* fixpoint.

The goal of a domain closure axiom is to force the any model of the above rule to make h true for the whole universe. First, notice that the weaker requirement that the universe be *a* fixpoint of the rule is expressed by $\forall x\,\phi_H[h/\mathbf{true}](x)$, which axiom is considered later under the name dca_{fo}. However, to require that the universe be the *least* fixpoint requires a second order axiom, such as the following.

Definition 4.3: With ϕ_H as in Definition 4.2, for a given functional vocabulary L, the *domain closure axiom* is:

$$dca \stackrel{\text{def}}{=} \forall R\,[(\phi_H[h/R] \subseteq R) \rightarrow \forall y\,R(y)]$$

This axiom states that no proper subset of the universe both contains (the interpretations of) the constant symbols and is closed under (the interpretations of) the function symbols of L. □

Recall that the operator $\mathbf{T}_{P\tilde{P}}$ is the analogue of Fitting's Φ_P operator. It follows that the Fitting semantics can be defined as the logical consequences of the dualized program conjoined with the above domain closure axiom: $(\mathbf{P}, \tilde{\mathbf{P}}, \mathrm{CEA}, dca)$.

4.2 Kunen's Logical Consequence Semantics

Kunen has also proposed a semantics based on Fitting's Φ_P operator ($\mathbf{T}_{P\tilde{P}}$ in this paper's notation) [19]. The well-recognized difference between the Fitting and Kunen semantics is that Fitting uses $\mathbf{T}^{\infty}_{P\tilde{P}}$, the least fixpoint, whereas Kunen uses $\mathbf{T}^{\omega}_{P\tilde{P}}$. There is another subtle difference: Kunen requires the logic program to be expressed in a vocabulary L_* with a countably infinite set of function symbols of each arity. This has the effect of preventing the (finite) definition of domain closure!

The infinite set of function symbols in L_* can be introduced by using CEA$*$, an extended set of equality axioms that mentions these function symbols as well as those that occur in the program. The *Kunen semantics* is defined as the logical consequences of $(\mathbf{P}, \tilde{\mathbf{P}}, \mathrm{CEA}*)$. One of Kunen's main theorems [19, Theorem 6.3] states that this semantics corresponds to $\mathbf{T}^{\omega}_{P\tilde{P}}$ (which may not be a fixpoint!) on the Herbrand universe of L_*:

Theorem 4.1: (Kunen) Let $\mathbf{P}$, $\tilde{\mathbf{P}}$, $\mathbf{T}_{P\tilde{P}}$, L_* and CEA$*$ be as defined above, and let I be the Herbrand interpretation in which the universe is generated by L_* and the relations are specified by $\mathbf{T}^{\omega}_{P\tilde{P}}$. Then sentence ψ is a logical consequence of $(\mathbf{P}, \tilde{\mathbf{P}}, \mathrm{CEA}*)$ if and only if ψ evaluates to true in I. $\Box$

The next example [50] shows that the Fitting and Kunen semantics are actually incomparable; neither is contained in the other.

Example 4.1: Recall the dualized program of Example 3.2.

$$\begin{aligned}
p(x) &\leftarrow e(x) \mathbin{\&} \exists y\, d(x,y) \\
d(x,y) &\leftarrow (x \neq y) \\
e(x) &\leftarrow (x = a) \\[4pt]
\tilde{p}(x) &\leftarrow \tilde{e}(x) \vee \forall y\, \tilde{d}(x,y) \\
\tilde{d}(x,y) &\leftarrow (x = y) \\
\tilde{e}(x) &\leftarrow (x \neq a)
\end{aligned}$$

plus CEA. First, note that neither $p(a)$ nor $\tilde{p}(a)$ is a logical consequence of $(\mathbf{P}, \tilde{\mathbf{P}}, \mathrm{CEA})$. The Fitting semantics and Kunen semantics further constrain the models in different ways, leading to different results.

The atom $\tilde{p}(a)$ is true in all Herbrand models, hence is true in the Fitting semantics, as it is a logical consequence of $(\mathbf{P}, \tilde{\mathbf{P}}, \mathrm{CEA}, dca)$. However, $p(a)$ is true in all models on infinite universes, hence is true in Kunen semantics, as it is a logical consequence of $(\mathbf{P}, \tilde{\mathbf{P}}, \mathrm{CEA}*)$. As both semantics respect *disjoint*, they are incomparable on this program.

Finally, since $(\mathbf{P}, \tilde{\mathbf{P}}, \mathrm{CEA})$ is a subset of both $(\mathbf{P}, \tilde{\mathbf{P}}, \mathrm{CEA}, dca)$ and $(\mathbf{P}, \tilde{\mathbf{P}}, \mathrm{CEA}*)$, its set of consequences is (always) a subset of sets of consequences of both the preceding two and (in this case) is a proper subset of them both. □

It is not always clear whether requiring the program to be interpreted in a universe with infinitely many "unknown" objects agrees with the user's intentions. This question is discussed in Section 7.

4.3 A Limitation of Program Completion

None of the program completion semantics capture the complement of an inductive closure, even over a finite set, in a natural way. The well worn example is the complement of transitive closure on a finite directed graph [48, 20, 50], as may be seen from Example 3.1. This fact is perhaps the primary motivation for exploring other semantics. We examine some recent proposals in Section 5.

5 A "Common Sense" Axiom

McCarthy observed that in everyday "common sense" reasoning, people treat a statement as false if there is no foundation for believing it to be true [28]. Thus from the rule "healthy birds can fly, except penguins" and the fact "Tweety is a healthy bird", the common sense conclusion is that Tweety can fly, because there is *no reason to believe* that Tweety is a penguin. He formalized this practice by adding an axiom requiring models to be minimal with respect to certain predicates.

For logic programs, we propose a different axiom, one that goes more directly to the point that there is "no reason to believe" something. Informally, a set of facts that we have "no reason to believe" are called *unsupported*. We intend to accept models only if they satisfy a *well-supported set axiom*, which requires all sets of unsupported facts about the *original* predicates of the program have dualized facts that are true in the model. That is, if $p(a)$ is in an unsupported set, then the well-supported set axiom requires that $\tilde{p}(a)$ must be in the model. This is the "common sense" axiom that we believe captures the spirit of circumscription more accurately than the usual relation-minimization axioms.

5.1 Unsupported Set Axiom

The unsupported set axiom is essentially a generalization of the notion of unfounded set [50] to arbitrary domains. The reader is advised to study the example that follows to see the motivation and get a feeling for how the definition works. Various technical meanings for "supported" and "unsupported" appear in the literature; care must be taken not to confuse them with the one that follows.

Definition 5.1: Let $\mathbf{P}$ be a given logic program with generic rule $q(x) \leftarrow \phi(x)$. The *unsupported set axiom* for U with respect to S and $\tilde{S}$ is:

$$unsup(U, S, \tilde{S}) \ \stackrel{\mathrm{def}}{=}\ U \subseteq \tilde{\phi}[q/S, \tilde{q}/(\tilde{S} \vee U)]$$

Here we use the substitution notation from Section 2.2; an occurrence $\tilde{q}(t)$ is replaced by $(\tilde{S}(t) \vee U(t))$, where t is a vector of terms. Observe that *unsup* depends implicitly on $\mathbf{P}$ through q, $\tilde{q}$ and $\tilde{\phi}$. Intuitively, given a model with $(S, \tilde{S})$ true, *unsup* states that U is able to "rederive itself" as additional tilde atoms. □

Example 5.1: The dualized transitive closure program of Example 3.1 was:

$$
\begin{aligned}
p(x,y) &\leftarrow e(x,y) \vee \exists z\,[p(x,z) \,\&\, p(z,y)] \\
e(x,y) &\leftarrow ((x=a) \,\&\, (y=b)) \vee ((x=b) \,\&\, (y=a)) \vee ((x=c) \,\&\, (y=c))
\end{aligned}
$$

$$
\begin{aligned}
\tilde{p}(x,y) &\leftarrow \tilde{e}(x,y) \,\&\, \forall z\,[\tilde{p}(x,z) \vee \tilde{p}(z,y)] \\
\tilde{e}(x,y) &\leftarrow ((x \neq a) \vee (y \neq b)) \,\&\, ((x \neq b) \vee (y \neq a)) \,\&\, ((x \neq c) \vee (y \neq c))
\end{aligned}
$$

As mentioned before, the rule for $\tilde{p}$ is recursive with no base case; consequently no facts for $\tilde{p}$ can be derived. However, (a,c) and (b,c), among other tuples, are in the complement of the transitive closure of e.

Let us use the notation $q = (p,e)$, $\phi = (\phi_p, \phi_e)$, and $U = (U_p, U_e)$ to be specific. Let the universe be $(a, b, c, a_{\#})$. Only one disjoint pair of relations $(e_0, \tilde{e}_0)$ on this universe satisfies the rules for e and $\tilde{e}$, namely those that interpret $\phi_e(x,y)$ and $\tilde{\phi}_e(x,y)$, respectively.

Consider (an interpretation for) U whose components are defined by $U_p = \{(a,c), (a,a_{\#}), (b,c), (b,a_{\#})\}$ and $U_e = \emptyset$. Then

$$
\begin{aligned}
\mathit{unsup}(U, (\emptyset, e_0), (\emptyset, \tilde{e}_0)) \quad &\stackrel{\text{def}}{=} \quad (U_p \subseteq \tilde{\phi}_p[p/\emptyset, e/e_0, \tilde{p}/U_p, \tilde{e}/(\tilde{e}_0 \cup U_e)]) \\
&\& \quad (U_e \subseteq \tilde{\phi}_e[p/\emptyset, e/e_0, \tilde{p}/U_p, \tilde{e}/(\tilde{e}_0 \cup U_e)])
\end{aligned}
$$

Of course, the second conjunct is satisfied trivially. The first conjunct written in more detail becomes

$$\forall xy\,[U_p(x,y) \rightarrow (\tilde{e}_0(x,y) \,\&\, \forall z\,[U_p(x,z) \vee U_p(z,y)])]$$

This conjunct also is true for U_p as given. Thus the unsupported set axiom, $\mathit{unsup}(U, (\emptyset, e_0), (\emptyset, \tilde{e}_0))$, is satisfied by $U = (U_p, \emptyset)$. By monotonicity of $(\phi, \tilde{\phi})$, $\mathit{unsup}(U, S, \tilde{S})$ holds for any $(S, \tilde{S})$ that contains $(e_0, \tilde{e}_0)$.

Note that U_p is contained in the complement of the transitive closure of e in the given universe. If U_p were any smaller, and nonempty, the axiom would not hold. However, it could be enlarged to the entire complement of the transitive closure, and the axiom would hold, and this is true for all universes that contain the Herbrand universe. (View $a_{\#}$ as a generic extra term, and the details are straightforward.)

It is important that $\mathit{unsup}(U, (\emptyset, e_0), (\emptyset, \tilde{e}_0))$ holds only for U_p that are contained in the complement of the transitive closure. To see that this is indeed the case, first note that U_p must be a subset of $\tilde{e}_0$ for the axiom to have a chance, so assume this is the case. Now if $(s_1, t) \in U_p$ and is in the transitive closure, there is an edge (s_1, s_2) that is part of a simple path of length $n > 1$ from s_1 to t. Thus $U_p(s_1, s_2)$ must be false, and it is necessary that $U_p(s_2, t)$ be true for the axiom to hold. But there is a path from s_2 to t of length $n-1$, so there is an edge (s_2, s_3) that forces $U_p(s_3, t)$ to be true, and so on. Eventually, (s_n, t) is forced to be in U_p, but is also an edge, contradicting the assumption that U_p is contained in $\tilde{e}_0$.

In summary, the complement of the transitive closure is the maximum unsupported set with respect to $(e_0, \tilde{e}_0)$. $\square$

The observation in the previous example can be generalized to all positive existential inductive definitions (Horn programs), as well as to stratified programs. The arguments are omitted, as they have been given elsewhere in connection with the well-founded semantics [50, Theorems 3.7 and 6.1]. Although those proofs were phrased in terms of unfounded sets, rather than unsupported sets, no new ideas are involved. The connection between unsupported sets and unfounded sets is discussed next.

Unfounded sets were defined as part of the well-founded semantics [50]. They were defined within the Herbrand universe, although the definition can be applied in other universes. (In this section, a, b, and c will denote ground (variable-free) terms, not necessarily constant symbols.) The definition is given here in terms of the dualized program. The cited paper should be consulted for additional details and motivation. The relationship to unsupported sets is illustrated in the lemma that follows the definitions.

Definition 5.2: Without loss of generality, we assume that each rule body is in the form

$$\phi_i(x) \stackrel{\text{def}}{=} \bigvee_j \exists y_j \mathop{\&}_k L_{jk}(x, y_j)$$

where the L_{jk} are atoms. A vector of relations U_i, $1 \leq i \leq n$, is an *unfounded set* with respect to $(S, \tilde{S})$ if for each ground Herbrand rule instantiation

$$\phi_i(a) = \bigvee_j \mathop{\&}_k L_{jk}(a, b_j)$$

such that $a \in U_i$, for each disjunct j there is a so-called *witness of unusability* for some $L_{jk}(a, b_j)$ with one of these properties, where q_m denotes the predicate symbol of L_{jk} and c abbreviates (a, b_j):

1. If $L_{jk}(a, b_j)$ is the positive atom $q_m(c)$ and $c \in \tilde{S}_m$, then $\tilde{q}_m(c)$ is a witness of unusability; similarly, if $L_{jk}(a, b_j)$ is the tilde atom $\tilde{q}_m(c)$ and $c \in S_m$, then $q_m(c)$ is a witness of unusability.

2. If $L_{jk}(a, b_j)$ is the positive atom $q_m(c)$ and $c \in U_m$, then $q_m(c)$ is a witness of unusability.

Intuitively, for fixed i and a, the witnesses of unusability for all j collectively demonstrate that $q_i(a)$ cannot be the *first* atom in U to be derived starting from $(S, \tilde{S})$.

The mapping $\tilde{\mathbf{U}}(S, \tilde{S})$ is defined by

$$\begin{aligned} \tilde{\mathbf{U}}(S, \tilde{S}) \stackrel{\text{def}}{=} \{\tilde{q}_i(a) \mid a \in U_i \text{ such that} \\ U \text{ is unfounded w.r.t. } (S, \tilde{S})\} \end{aligned}$$

Note that this maps a pair of vectors of relations into a vector of tilde relations.

The *well-founded transformation* $\mathbf{W}(S, \tilde{S})$ combines the positive immediate consequences with $\tilde{\mathbf{U}}$:

$$\mathbf{W}(S, \tilde{S}) \stackrel{\text{def}}{=} (\phi[q/S, \tilde{q}/\tilde{S}],\ \tilde{\mathbf{U}}(S, \tilde{S}))$$

and its least fixpoint gives the *well-founded dualized model* [50]. □

Observe that, if U is unfounded w.r.t. $(S, \tilde{S})$, then no element of U can be the *first* element of U to be derived in a positive relation as long as the positive and tilde relations that represent the current "set of beliefs" remain disjoint and are supersets of $(S, \tilde{S})$. Furthermore, this remains true even after the tuples of U are added to the tilde relations. In this sense, it is "safe" to put U's tuples in the tilde relations.

What is called the "well-founded partial model" in the original nomenclature is called the "well-founded dualized model" here because it is actually a *two-valued* model of the dualized program. However, it may not satisfy the totality axiom, *total*. One theorem we shall use is that stable models are precisely the fixpoints of **W** that do satisfy *total* [50, Theorem 5.4].

The next lemma shows that unsupported sets are essentially a generalization of unfounded sets to arbitrary domains.

Lemma 5.1: Let **P** be a logic program and let L be a functional vocabulary containing at least the functional vocabulary of **P**. Define CEA and *dca* with respect to L. In structures satisfying CEA and *dca* (essentially the Herbrand universe for L), U is an unfounded set with respect to $(S, \tilde{S})$ if and only if U is an unsupported set with respect to $(S, \tilde{S})$.

Proof: This is a matter of checking the definitions. Let a rule body be

$$\phi_i(x) = \bigvee_j \exists y_j \mathop{\&}_k L_{jk}(x, y_j)$$

Its dual is

$$\tilde{\phi}_i(x) = \mathop{\&}_j \forall y_j \bigvee_k \tilde{L}_{jk}(x, y_j)$$

Suppose U is unfounded. The domain closure axiom forces $\forall y_j$ in $\tilde{\phi}_i$ to range only over ground Herbrand terms. For $x = a$ and $y_j = b_j$, the witness of unusability causes some $\tilde{L}_{jk}[q/S, \tilde{q}/(\tilde{S} \vee U)]$ to evaluate to true, and there is one witness for each conjunct j.

Suppose U is unsupported. Then for each $q_i(a) \in U$, the body $\tilde{\phi}_i(a)$ must be true, so each conjunct j must be true, so for all terms b_j some literal $\tilde{L}_{jk}[q/S, \tilde{q}/(\tilde{S} \vee U)](a, b_j)$ must be true. Whichever literal is true is the witness of unusability. For example, if the literal is $\tilde{q}_m(c)$ and $c \in U_m$, it meets condition 2; other cases meet condition 1. ∎

Several different fixpoint constructions of the well-founded semantics have appeared in the literature. Przymusinski has given one that involves the *greatest* fixpoint of an operator based on $\tilde{\phi}$ [35], and has shown (see Theorem 3.2 there) that it defines the well-founded semantics. Although the notation is rather different, we shall use essentially the same idea to characterize the greatest unsupported set.

Greatest fixpoints are not as widely used as least fixpoints. We shall use the following standard properties that are analogs of least fixpoint properties [32, 2]:

Fact A A monotone operator on a lattice has a greatest fixpoint.

Fact B The greatest fixpoint of a universally quantified positive induction (i.e., the carrier appears only positively, and not under an existential quantifier) closes within ω stages.

Fact C If G is the greatest fixpoint of the monotonic transformation T, where "$\subseteq$" is the partial order on the lattice, and $R \subseteq T(R)$, then $R \subseteq G$.

Recall that a set U is unsupported with respect to S and $\tilde{S}$ if it satisfies: $U \subseteq \tilde{\phi}[q/S, \tilde{q}/(\tilde{S} \vee U))]$ (Definition 5.1). Now add to the language a vector u of extra predicates for the sets U, and consider the formula $\tilde{\phi}[q, \tilde{q}/(\tilde{q} \vee u)]$. By construction, u occurs only positively in this formula. Moreover, the formula has only universal quantification (if any), because ϕ, as a normal rule body, may contain only existential quantification.

Hence, for any structure $|\mathcal{M}|$, with fixed interpretations $(S, \tilde{S})$ of $(q, \tilde{q})$, there is a greatest fixed point G of the formula; that is, G is the greatest set (vector of relations, actually) U that satisfies $U = \tilde{\phi}[q/S, \tilde{q}/(\tilde{S} \vee U)]$.

Lemma 5.2: The greatest fixpoint G just described is the union of all sets U that satisfy

$$U \subseteq \tilde{\phi}[q/S, \tilde{q}/(\tilde{S} \vee U)]$$

on $|\mathcal{M}|$. Thus it is the *greatest unsupported set* with respect to S and $\tilde{S}$. Moreover, G is definable by an induction that closes within ω stages.

Proof: By Fact C above, any unsupported set is contained in G. But G, being a fixpoint the formula, is itself an unsupported set. Closure within ω follows from Fact B above. ▮

The greatest fixed point G above can be constructed in universe $|\mathcal{M}|$ for given $(S, \tilde{S})$ by induction, in the standard fashion:

- $U_0 = |\mathcal{M}|^k$ (for whatever the appropriate arity k is).
- $U_{n+1} = \tilde{\phi}[q/S, \tilde{q}/(\tilde{S} \vee U_n)]$.
- $U_\omega = \bigcap_m U_m$.

This U_ω is the desired greatest unsupported set. The finite stages of the above constructions can be captured by finitary formulas: $\chi_0(x) = \textbf{true}$, and $\chi_{n+1}(x) = \tilde{\phi}[q/S, \tilde{q}/(\tilde{S} \vee \chi_n)]$.

5.2 Well-Supported Set Axiom

We now formulate an axiom to require that all unsupported sets are contained in corresponding tilde relations. This is the "common sense" axiom that we believe captures the spirit of circumscription more accurately than the usual relation-minimization axioms.

Definition 5.3: Let $\mathbf{P}$ be a given logic program with generic rule $q(x) \leftarrow \phi(x)$. The *well-supported set axiom* is:

$$wellsup(R, \tilde{R}) \stackrel{\text{def}}{=} \forall U\, [unsup(U, R, \tilde{R}) \rightarrow (U \subseteq \tilde{R})]$$

Observe that *wellsup* depends implicitly on $\mathbf{P}$ through *unsup*.

Intuitively, for a "set of beliefs" $(R, \tilde{R})$ to be well-supported, any unsupported set U with respect to $(R, \tilde{R})$ must be contained in $\tilde{R}$. □

Certain relationships between the *wellsup* axiom and stable and well-founded models can be shown. These relationships apply to so-called "augmented" programs as well as the original programs. A simple way to incorporate "unknown" objects is to form the augmented program [50]:

Definition 5.4: Given a dualized set of rules $(\mathbf{P}, \tilde{\mathbf{P}})$, the associated *augmented program* is obtained by first adding the rule:

$$aug \stackrel{\text{def}}{=} p_{\#}(g_{\#}(a_{\#})) \leftarrow p_{\#}(g_{\#}(a_{\#}))$$

to the program, where $p_{\#}$, $g_{\#}$, and $a_{\#}$, are new symbols, not in $\mathbf{P}$. This produces a new functional vocabulary, $L_{\#} = L_P \cup \{a_{\#}, g_{\#}\}$, with corresponding Clark equality axioms, denoted as CEA#. Thus the augmented program is $(\mathbf{P}, \tilde{\mathbf{P}}, aug, \text{CEA}\#)$. Predicate $p_{\#}$ is considered neither a positive nor a tilde predicate; it just functions as a "carrier" for $g_{\#}$ and $a_{\#}$. Clearly it can always be interpreted as the empty relation.

If domain closure (Definition 4.3) is to be used, it is defined with respect to $L_{\#}$, and is denoted as *dca#*. □

Theorem 5.3: Let a dualized (possibly augmented) program be given, and append the domain closure axiom, and the axiom $wellsup(q, \tilde{q})$. The logical consequences of the resulting set of axioms, which is $(\mathbf{P}, \tilde{\mathbf{P}}, \text{CEA}, wellsup(q, \tilde{q}), dca)$ (for the non-augmented program) or $(\mathbf{P}, \tilde{\mathbf{P}}, aug, \text{CEA}\#, wellsup(q, \tilde{q}), dca\#)$ (for the augmented program), agree with the well-founded dualized model in the sense that a ground atom is in the well-founded dualized model if and only if it is a logical consequence of the axioms.

Proof: The domain closure axiom allows us to restrict attention to Herbrand models. Suppose $(R, \tilde{R})$ is the well-founded dualized model. By its definition [50, Section 3], $\tilde{R}$ is the union of all sets U that are unfounded with respect to $(R, \tilde{R})$. By Lemma 5.1 all such U are unsupported, so the axiom $wellsup(R, \tilde{R})$ holds. Thus the well-founded dualized model is an upper bound on the set of atoms that are logical consequences. Let $(S, \tilde{S})$ be any Herbrand model that satisfies $wellsup(q, \tilde{q})$. Being a model, we have $\phi(S, \tilde{S}) \subseteq S$, and by the *wellsup* axiom $\tilde{\mathbf{U}}(S, \tilde{S}) \subseteq \tilde{S}$ (recall Definition 5.2 for $\tilde{\mathbf{U}}$ and $\mathbf{W}$). Thus $(S, \tilde{S})$ is a pre-fixpoint of $\mathbf{W}$, so is a superset of the well-founded dualized model. ∎

Note that we did not append the disjointness axiom to the list of assumptions above. Just as was the case without the axiom *wellsup*, it is straightforward to show that, for any dualized program $(\mathbf{P}, \tilde{\mathbf{P}}, \text{CEA})$, with any combination of the axioms *aug*, *dca*, and *wellsup* (all expressed in appropriate vocabularies), the logical consequences are the same with or without the *disjoint* axiom. In each case, over any fixed universe, the set of consequences can be built up by transfinite induction, and it is routine to show that there can be no first step where *disjoint* is violated. However, this axiom, as well as *total* are needed to obtain *stable* models.

Theorem 5.4: Let a dualized (possibly augmented) program be given, and append the domain closure axiom, the totality axiom, the disjointness axiom, and the axiom $wellsup(q, \tilde{q})$. The models of the resulting set of axioms, which is $(\mathbf{P}, \tilde{\mathbf{P}}, \mathrm{CEA}, disjoint, total, wellsup(q, \tilde{q}), dca)$ (for the non-augmented program) or $(\mathbf{P}, \tilde{\mathbf{P}}, aug, \mathrm{CEA}\#, disjoint, total, wellsup(q, \tilde{q}), dca\#)$ (for the augmented program), are precisely the stable models of the original (possibly augmented) program, with $\tilde{q}$ corresponding to $\neg q$.

Proof: It is sufficient to prove that all models are fixpoints of $\mathbf{W}$, and all fixpoints of $\mathbf{W}$ that satisfy totality and disjointness are models [50, Theorem 5.4]. If $(R, \tilde{R})$ is a fixpoint of $\mathbf{W}$, the *wellsup* axiom holds, as argued in the previous theorem. It easily follows that $(R, \tilde{R})$ is a model of the whole formula.

Now suppose $(R, \tilde{R})$ is a model, implying that $\phi[q/R, \tilde{q}/\tilde{R}] \subseteq R$. If $U \stackrel{\mathrm{def}}{=} R - \phi[q/R, \tilde{q}/\tilde{R}]$ is nonempty, then we claim it is an unsupported set w.r.t. $(R, \tilde{R})$. By totality and disjointness, $\tilde{\phi}$ is true just where ϕ is false (and *vice versa*), so $U \subseteq \tilde{\phi}[q/R, \tilde{q}/\tilde{R}]$. So Definition 5.1 holds by monotonicity of $\tilde{\phi}$, and the claim follows. Consequently, $\phi[q/R, \tilde{q}/\tilde{R}] = R$ (or disjointness would be violated). Similarly, $wellsup(R, \tilde{R})$ implies that $\tilde{\mathbf{U}}(R, \tilde{R}) \subseteq \tilde{R}$. If it is a proper subset, there is a nonempty set $S \subseteq \tilde{R}$ such that $S \subseteq \phi[q/R, \tilde{q}/\tilde{R}] = R$ as well, again violating disjointness. So $\tilde{\mathbf{U}}(R, \tilde{R}) = \tilde{R}$, and $(R, \tilde{R})$ is a fixpoint of $\mathbf{W}$. ▮

One issue that arises with second order axioms is whether countable models exist. In conjunction with *dca* and CEA, all models must be countable, as their universes are isomorphic copies of the Herbrand universe. However, one motive for taking the axiomatic approach was to get away from fixed universes. It is noteworthy that *wellsup* has countable models in the sense given next, without the coercion of *dca*.

Recall that $\mathcal{M}$ is an *elementary submodel* of an arbitrary infinite model $\mathcal{N}$ if its universe $|\mathcal{M}|$ is a subset of the universe of $\mathcal{N}$, and every *first order* formula with k free variables is true in $\mathcal{M}$ for tuple $(m_1, \ldots, m_k) \in |\mathcal{M}|^k$ if and only if it is true in $\mathcal{N}$ for the same tuple. By the downward Löwenheim-Skolem theorem, every infinite model $\mathcal{N}$ (over a finite or countable vocabulary) has a countable elementary submodel $\mathcal{M}$. However, since $wellsup(q, \tilde{q})$ is stated in second order logic, we cannot use that result to conclude immediately that, if a set of sentences including $wellsup(q, \tilde{q})$ has a model, it has a countable model. However, this property can be proved with the aid of Lemma 5.2 and the construction following it.

Theorem 5.5: Let ψ be a first order sentence involving just the symbols of the program $(\mathbf{P}, \tilde{\mathbf{P}})$. If $\mathcal{N}$ is any infinite model of $(\mathbf{P}, \tilde{\mathbf{P}}, aug, \mathrm{CEA}\#, wellsup(q, \tilde{q})) \cup \{\psi\}$, and $\mathcal{M}$ is a countable elementary model of $\mathcal{N}$, then $\mathcal{M}$ is also a model of $(\mathbf{P}, \tilde{\mathbf{P}}, aug, \mathrm{CEA}\#, wellsup(q, \tilde{q})) \cup \{\psi\}$.

Proof: Suppose $\mathcal{N}$ is an infinite model of $(\mathbf{P}, \tilde{\mathbf{P}}, aug, \text{CEA}\#, wellsup(q, \tilde{q})) \cup \{\psi\}$ with universe $|\mathcal{N}|$ and interpretations $(S, \tilde{S})$ of $(q, \tilde{q})$. Let $\mathcal{M}$ be a countable elementary submodel of $\mathcal{N}$, with interpretations $(R, \tilde{R})$ of $(q, \tilde{q})$.

Since $\mathcal{M}$ is an elementary submodel of $\mathcal{N}$, $\mathcal{M}$ is a model of the first order axioms $(\mathbf{P}, \tilde{\mathbf{P}}, aug, \text{CEA}\#) \cup \{\psi\}$. So it remains only to prove that $\mathcal{M}$ is a model of $wellsup(q, \tilde{q})$.

To prove $wellsup(R, \tilde{R})$ is satisfied, we must show that the greatest unsupported set G_M w.r.t $(R, \tilde{R})$, calculated in $\mathcal{M}$, is a subset of $\tilde{R}$, the interpretation of $\tilde{q}$ in $\mathcal{M}$. Suppose not: some tuple $x \in (G_M - \tilde{R})$; such an x cannot be in $\tilde{S}$, as $\tilde{R} = \tilde{S}$ restricted to $\mathcal{M}$. Recall that G_M is the intersection of the interpretations of the finitary first order formulas χ_n given following Lemma 5.2. Since $x \in G_M$, every $\chi_n(x)$ is true in $\mathcal{M}$. Hence every $\chi_n(x)$ is also true in $\mathcal{N}$, by the elementary submodel relationship, so x must be in the greatest unsupported set w.r.t $(S, \tilde{S})$ in $\mathcal{N}$. But $x \notin \tilde{S}$, so $wellsup(q, \tilde{q})$ in $\mathcal{N}$ is violated. Thus the assumption that $G_M \not\subseteq \tilde{R}$ is inconsistent, so the theorem is proved. ▮

6 Finite Presentations

Issues of finite axiomatization might seem like abstract theoretical points, but our study is motivated by quite practical considerations. If the semantics of a logic program can be stated with a finite vocabulary, that should be a great convenience for the development of compilers, interpreters, and automated verification tools.

In this section we give alternative finite formulations of CEA and of Kunen's semantics. To simplify the presentation, we sometimes assume that the program contains just one constant a, one unary function symbol g, and one binary function symbol f. This is without loss of generality, as larger vocabularies can be encoded with just the symbols a and f.

6.1 Finite Presentation of CEA

For a finite program, the *global injectivity constraint* can be stated finitely (see Definition 3.2). Only the *acyclicity constraint*, which prevents a term being equal to a proper subterm of itself, requires an infinite number of inequalities. However, an (essentially) equivalent constraint can be stated finitely in first-order logic, with the aid of an auxiliary predicate. Similar constructions have been used elsewhere in logical literature.

Definition 6.1: Let L be a given finite functional vocabulary. The following formula, denoted $\mathbf{F}_L(x, y)$, defines *functional edges*. For illustration, we assume L contains one unary function symbol g one binary function symbol f, plus constant symbol a (constants do not affect this definition). The general case is the obvious extension, including k disjuncts for each k-ary function symbol of L.

$$\mathbf{F}_L(x, y) \stackrel{\text{def}}{=} y = g(x) \vee \exists z\, [y = f(x, z)] \vee \exists z\, [y = f(z, x)]$$

For functional vocabulary L_i, the associated formula is written $\mathbf{F}_i$, to avoid double subscripts.

In any structure for L with fixed interpretations of the symbols of L, the *functional graph* is the set of functional edges in the structure–that is, ordered pairs of elements (m_1, m_2) such that $\mathbf{F}_L(m_1, m_2)$ is satisfied. (Because $\mathbf{F}_L$ contains only the equality predicate, there is no need to distinguish between the formula and its interpretation.) □

For infinite vocabularies, the definition of $\mathbf{F}_L$ becomes an infinite length disjunction. When we use $\mathbf{F}_L$ in definitions of concepts, that is harmless. In other contexts, such as in the statement of the $\sqsubset$-acyclicity axioms below, it is possible to get an equivalent definition by replacing the one axiom involving $\mathbf{F}_L$ with an infinite collection of (finite-length) axioms, one for each disjunct in the definition of $\mathbf{F}_L$ above.

Clearly, if $\mathbf{F}(t_1, t_2)$ holds, then t_1 is a proper subterm of t_2. However, to identify all proper subterms of t_2, transitivity is needed.

Definition 6.2: Let $\sqsubset$ be a predicate symbol not occurring in the program. We shall use it as an infix binary predicate, with the intuitive meaning, "is a proper subterm of". Let L be a given functional vocabulary. Now the $\sqsubset$-*acyclicity constraint* is given by a set of axioms in the following form.

$$\begin{aligned}
&\forall xy\ [x \sqsubset y \leftarrow \mathbf{F}_L(x, y)] \\
&\forall xyz\ [x \sqsubset z \leftarrow (x \sqsubset y\ \&\ y \sqsubset z)] \\
&\neg \exists x\ [x \sqsubset x]
\end{aligned}$$

The last two lines assert that $\sqsubset$ is transitive and anti-reflexive–that is, it is a strict partial order. Clearly, if term t_2 is formed by any finite number of function applications from term t_1 and other terms, then $t_1 \sqsubset t_2$ must hold.

Note that this definition depends implicitly on the vocabulary; when necessary, the intended vocabulary is made explicit in the discussion. However, subscripts to the "$\sqsubset$" symbol denote more than this; see Definition 6.3.

For an infinite vocabulary, an infinite set of axioms replaces the first line.

A $\sqsubset$-*minimum* model is one that interprets $\sqsubset$ by its minimum model. □

Because the axioms for $\sqsubset$ can be written as Horn clauses (including one negative clause) they have the model intersection property. If they have any model in a given structure, they have a minimum model in that structure, namely the transitive closure of the functional graph. The transitive closure is defined by the well-known least fixpoint, and every tuple in it is "derived" at some finite stage. Thus "$\sqsubset$-minimum" is well defined above.

However, we do not in general require $\sqsubset$ to be interpreted by its least fixpoint; that would require a second order axiom. It is sufficient for our purposes that any model *contains* the least fixpoint. For each such model, there is a "$\sqsubset$-minimum" model, which interprets all relations as in the original model except that it interprets $\sqsubset$ by its least fixpoint. Although the axioms do not restrict models to be $\sqsubset$-minimum, certain properties of $\sqsubset$-minimum models will be useful.

Definition 6.3: Let $\mathcal{M}$ be a structure for vocabulary L and predicate symbol $\sqsubset$, as well as the symbols of program **P**. The minimum relation satisfying Definition 6.2 is denoted $\sqsubset_L$. To avoid double subscripting, for L_i the minimum relation is denoted $\sqsubset_i$, rather than $\sqsubset_{L_i}$. In particular: $\sqsubset_P$ is associated with L_P the vocabulary of the original program in question; $\sqsubset_\#$ is associated with $L_\#$; $\sqsubset_*$ is associated with L_*; $\sqsubset_1$ is associated with L_1, etc. $\square$

Lemma 6.1: Every model of the $\sqsubset$-acyclicity constraint satisfies the acyclicity constraint of Definition 3.2. Moreover, any model (without a $\sqsubset$ relation) that satisfies that acyclicity constraint can be extended to a "$\sqsubset$-minimum" model.

Proof: (1) Start with a model satisfying the $\sqsubset$-acyclicity constraint. Clearly, the interpretation of $\sqsubset$ must contain the transitive closure of the functional graph (Definition 6.1). Now if any acyclicity axiom of Definition 3.2, say $x \neq t(x)$, were violated, it would produce a sequence of functional edges from the occurrence of x in t to the root of t that constitute a cycle. Thus $x \sqsubset x$ would be satisfied, contradicting the hypothesis.

(2) The transitive closure of the functional edge relation satisfies the first two requirements for $\sqsubset$. If there were a finite sequence of functional edges from any x to itself, we could read off a violation of the acyclicity constraint of Definition 3.2 from the cycle, so the transitive closure must be acyclic. Finally, as noted above, the transitive closure of the functional edge relation must be the minimal such $\sqsubset$ relation. ▮

Therefore, whether CEA is interpreted to contain the original acyclicity constraint or the $\sqsubset$-acyclicity constraint, the logical consequences of the program are not affected as far as symbols other than $\sqsubset$ are concerned.

The axiom set CEA$*$ used for the Kunen semantics has no finite presentation, as it covers an infinite set of symbols. We shall show that the Kunen semantics can nevertheless be finitely presented.

6.2 Technical Lemmas

As tools for demonstrating a finite presentation of Kunen's semantics (Theorem 6.7), we look briefly at the structure of models of CEA and an additional binary relation $\prec$ or $\sqsubset$, on functional vocabularies that include all symbols of **P** and possibly additional symbols. The relation $\prec$ or $\sqsubset$, written as an infix operator, helps to identify elements of the universe that are not built up from the functional vocabulary of **P** alone. Essentially, these elements will enable us to transform models on infinite vocabularies to models on finite vocabularies. We shall use $x \preceq y$ and $x \sqsubseteq y$ as abbreviations for $(x = y \vee x \prec y)$ and $(x = y \vee x \sqsubset y)$, respectively.

In particular, we shall use the symbol $\sqsubset_i$ to indicate the transitive closure of the functional edge relation of vocabulary L_i. We shall generally use $\prec$ for intermediate stages in the construction of such $\sqsubset$ relations.

Definition 6.4: Let $\mathcal{M}$ be a structure for functional vocabulary L, with universe $|\mathcal{M}|$. Recall that $\sqsubset_L$ denotes the minimum relation satisfying Definition 6.2 for L.

Call x a $\sqsubset_L$*-minimal element* of $|\mathcal{M}|$ if there is no $z \in |\mathcal{M}|$ such that $z \sqsubset_L x$. Note that these are just the elements that are not in the range of (the interpretations of) any function of L.

An element of $|\mathcal{M}|$ is called *unnameable in* L (or just *unnameable* when L is understood) if it is not in the range of (the interpretations of) any function or constant of L. $\square$

Definition 6.5: Let L be any functional vocabulary. An *expanding vocabulary* for L is a countable (i.e., finite or countably infinite) vocabulary $L_1 \supseteq L$ which contains at least one function symbol (of positive arity) not in L.

Note that, for any program $\mathbf{P}$, $L_\#$ $(= L_P \cup \{a_\#, g_\#\}$ per Definition 5.4) and L_* (which consists of L_P plus infinitely many new function symbols of each arity, including 0, per Section 4.2) are both expanding vocabularies for L_P.

The import of the above definition is that, if L_1 is an expanding vocabulary for L_P, then the corresponding CEA_1 forces the existence of infinitely many objects not forced to exist by CEA.

We conjecture that the definition of "expanding vocabulary" can be generalized to include the case where L_1 contains infinitely many constants not in L_P, but no new functions. This case might be of interest to deductive database theory. However, the proof of Lemma 6.2 would break down, and this is needed for Lemma 6.5. Overcoming the technical difficulties appears to require advanced techniques of recursive function theory and recursively saturated models [5], which are beyond the scope of this paper.

Lemma 6.2: Let $\mathbf{P}$ be a logic program with vocabulary L_P. Let L_1 be any expanding vocabulary for L_P. Let $\mathcal{M}_1$ be any model for $(\mathbf{P}, \tilde{\mathbf{P}}, \mathrm{CEA}_1)$, with universe $|\mathcal{M}|$. For each constant symbol a or function symbol f of L_1, let $|a|$ or $|f|$ denote its interpretation in $\mathcal{M}_1$. Define $\sqsubset_P$ as in Definition 6.3.

1. For each $a \in L_1 - L_P$, $|a|$ is unnameable in L_P.

2. For each k-ary $f \in L_1 - L_P$ and each k-tuple m of elements of elements $|\mathcal{M}|$, $|f|(m)$ is unnameable in L_P.

3. Infinitely many elements of $|\mathcal{M}|$ are unnameable in L_P.

4. Every element of $|\mathcal{M}|$ that is unnameable in L_P is $\sqsubset_P$-minimal.

Proof: (1–3) are a straightforward consequences of CEA_1. For (4), by the fact that an element that is unnameable in L_P cannot have a functional edge of $\mathbf{F}_P$ entering it, it is $\sqsubset_P$-minimal. ∎

We now introduce the notions of "weakly dominated by" and "dispersed minimal set", which are purely technical tools for the proofs that follow.

Definition 6.6: Let $\prec$ be any partial order on set S. Call x a *$\prec$-minimal element* of S if there is no $z \in S$ such that $z \prec x$.

Let U and V be subsets of S. We say U is *weakly dominated by* V (under $\prec$) if, for each $u \in U$ there is a $v \in V$ such that $u \preceq v$.

We call U a *dispersed minimal set for* $\prec$ if U is a subset of $\prec$-minimal elements of S that is not weakly dominated under $\prec$ by any finite subset of S. $\square$

Observe that any dispersed minimal set must be infinite, because it is weakly dominated by itself.

Lemma 6.3: Let $\mathbf{P}$ be a logic program with vocabulary L_P. Let L_1 be an expanding language for L_P. Let $\mathcal{M}_1$ be any model for $(\mathbf{P}, \tilde{\mathbf{P}}, \mathrm{CEA}_1)$, with universe $|\mathcal{M}|$. Define $\sqsubset_P$ as in Definition 6.3. Let $C = \{m_1, \ldots, m_k\}$ be any nonempty finite subset of $|\mathcal{M}|$. Then there is an element $m' \in |\mathcal{M}|$ such that:

1. m' is unnameable in L_P;

2. m' is not the interpretation of any constant symbol in L_1;

3. $m' \not\sqsubseteq_P m_i$ for any $m_i \in C$.

Therefore, the subset of $|\mathcal{M}|$ that is unnameable in $\mathbf{P}$ is a dispersed minimal set for $\sqsubset_P$.

Proof: Let $\sqsubset_1$ be the minimum $\sqsubset$ relation for L_1 (Definition 6.3), and let f be a function in $L_1 - L_P$. For notational simplicity, we assume f is binary. Let $|f|$ be the interpretation of f. Let m_j be a maximal element of C with respect to $\sqsubset_1$ (which must exist since C is finite and nonempty). Set $m' = |f|(m_j, m_j)$. If $m' \sqsubseteq_P m_i$ is true for any $m_i \in C$, then $m_j \sqsubset_1 m_i$, contradicting maximality.

The subset of $|\mathcal{M}|$ that is unnameable in $\mathbf{P}$ (call it U) is $\sqsubset_P$-minimal by Lemma 6.3. For any nonempty, finite $C \subseteq |\mathcal{M}|$, an $m' \in U$ can be chosen as above to demonstrate that U is not weakly dominated by C. ▮

Example 6.1: Let S be the set of all pairs (i, j), (ω, j), (i, ω), where i and j are natural numbers and ω is the first infinite ordinal. Let $(u, v) \prec (x, y)$ hold when $u < x$ and $v < y$ in the standard order. Then $\{(0, 2), (1, 1)\}$ is weakly dominated by $\{(2, 3)\}$ and $\{(0, 2), (1, 1), (2, 0)\}$ is weakly dominated by $\{(2, 3), (2, 0)\}$.

Here $\prec$-minimal elements are those of the forms $(0, x)$ and $(x, 0)$. For U to be a dispersed minimal set, it must be the union of infinite subsets of both minimal forms; for example, if only a finite number of elements of the form $(0, x)$ are in U, let the largest such finite x be less than the integer n ($(0, \omega)$ may also be present). But now U is weakly dominated by the finite set $\{(0, \omega), (\omega, n)\}$. $\square$

Lemma 6.4: Let $\prec_{i-1}$ be a partial order on an infinite set S and let U_{i-1} be a dispersed minimal set for $\prec_{i-1}$. Let $C_{i-1} = \{r_1, \ldots, r_k\}$ be a finite subset of S.

Suppose $r \in U_{i-1}$ is such that $r \not\preceq_{i-1} r_i$ for all $r_i \in C_{i-1}$. Let $\prec_i$ be the transitive closure of $\prec_{i-1} \cup\{(r_1 \prec r), \ldots, (r_k \prec r)\}$. Let U_i be $U_{i-1} - \{r\}$. Then $\prec_i$ is a strict partial order on S and U_i is a dispersed minimal set for $\prec_i$.

Proof: First, we claim that $\prec_i$ is a strict partial order. It is transitive by definition. If there is a cycle $s \prec_i s$ for any $s \in S$, it must involve an edge $(r_i \prec r)$ for some i. Follow the cycle from r over edges of $\prec_{i-1}$ until it reaches some r_j (possibly $j = i$) such that the next edge in the cycle is $(r_j \prec r)$. This sequence of edges shows that $r \prec_{i-1} r_j$, contradicting the choice of r. Thus $\prec_i$ is also acyclic, proving the claim.

Clearly, U_i is infinite and consists of $\prec_{i-1}$-minimal elements of S. All of these except r are also $\prec_i$-minimal.

Finally, suppose that U_i were weakly dominated under $\prec_i$ by some finite subset C_i. Then U_{i-1} would be weakly dominated under $\prec_{i-1}$ by $(C_i \cup C_{i-1})$, contradicting the hypothesis of the lemma. ▮

Example 6.2: Continuing Example 6.1, let $\prec_0$ be $\prec$ from that example, and let U_0 be the set of all elements of the form $(0, x)$ or $(x, 0)$. Let $C_0 = \{(2, 2)\}$. Then a possible choice of r is $(0, 3)$, which leads to $\prec_1$ having the new relationships: $(2, 2) \prec (0, 3)$, $(2, 2) \prec (1, k)$, $(2, 2) \prec (2, k)$, for all $k \geq 4$. □

6.3 Finite Presentation of Kunen Semantics

We have considered two ways to add to the functional vocabulary of the program, finitely with CEA# and infinitely with CEA$*$. This section shows that models can be transformed between the two vocabularies without changing the interpretation of any symbol in $\mathbf{P}$. While this seems quite plausible, there are technical difficulties due to the fact that the functional graph (Definition 6.2) need not be well-founded, so straightforward inductive constructions do not work.

Recall that $L_\#$ denotes the vocabulary of $\mathbf{P}$ with the additional constant $a_\#$ and unary function $g_\#$, that CEA# is defined over this vocabulary, and that elements of a universe are called *nameable in* $L_\#$ if they are in the range of (the interpretation of) some constant or function of $L_\#$.

Lemma 6.5: Let ψ be a first order sentence involving just the symbols of the program $(\mathbf{P}, \tilde{\mathbf{P}})$. Let L_P be the vocabulary of $\mathbf{P}$, and let $L_\#$ be that vocabulary augmented with constant $a_\#$ and unary function $g_\#$. Let L_1 be any expanding vocabulary for L_P, and let CEA_1 be based on L_1. If $(\mathbf{P}, \tilde{\mathbf{P}}, \text{CEA}_1) \cup \{\neg\psi\}$ has a model $\mathcal{M}_1$ with universe $|\mathcal{M}|$, then $(\mathbf{P}, \tilde{\mathbf{P}}, \mathit{aug}, \text{CEA\#}) \cup \{\neg\psi\}$ has a model. Further, if $|\mathcal{M}|$ is countable, the new model can be chosen so that it has the same universe, and every element of the universe is nameable in $L_\#$.

Proof: By the downward Löwenheim-Skolem theorem, it suffices to restrict attention to countable models. We shall construct an interpretation $|g_\#|$ for $g_\#$ and form a new structure $\mathcal{M}\#$ by discarding the interpretations of symbols in CEA_1 that are not in $\mathbf{P}$, adding $|a_\#|$ and $|g_\#|$, interpreting $p_\#$ as the empty relation, and leaving everything else as in $\mathcal{M}_1$. It follows immediately that $\mathcal{M}\#$ is a model of $(\mathbf{P}, \tilde{\mathbf{P}}, aug)$ and $\neg\psi$.

We shall construct the ranges of $|a_\#|$ and $|g_\#|$ to be exactly the set of elements of $|\mathcal{M}|$ that are unnameable in L_P. Because CEA_1 is an expanding language for L_P, that set is infinite. We shall also construct $|g_\#|$ to be 1-1; this will satisfy the global injectivity constraints of the CEA#. Finally, as we construct $|g_\#|$, we shall also construct a partial order $\sqsubset_\#$ which will show the acyclicity constraint of CEA# to be satisfied.

Enumerate the elements of $|\mathcal{M}|$ as $m_1, m_2, \ldots$, in any order. Let $u_0, u_1, u_2, \ldots$ be the subsequence of this sequence consisting of all elements of $|\mathcal{M}|$ that are unnameable in L_P. (The interpretation of L_P is retained from $\mathcal{M}_1$.) Choose u_0 as the interpretation of $a_\#$.

Let U_0 be the set $\{u_1, u_2, \ldots\}$; we need to define $|g_\#|$ to be a 1-1 function from $|\mathcal{M}|$ onto U_0. Define $\prec_0 = \sqsubset_P$ (see Definition 6.3). We shall extend $\prec_0$ to a new strict partial order $\sqsubset_\#$ that satisfies Definition 6.2 for $L_\#$, thereby showing that $\mathcal{M}\#$ satisfies the acyclicity constraint of CEA#.

We construct $|g_\#|$ inductively, for stages α ranging over the natural numbers, with different cases for α even and α odd. For stage 0, $|g_\#|$ is totally undefined. We preserve the following invariant properties:

1. U_α consists of all elements of U_0 which are not in the range of the part of $|g_\#|$ constructed by the end of stage α;
2. $\prec_\alpha$ is a strict partial order on $|\mathcal{M}|$;
3. U_α is a dispersed minimal set for $\prec_\alpha$ (see Definition 6.6).
4. for $j < i \leq \alpha$, $U_j \supseteq U_i$ and $\prec_j \subseteq \prec_i$;
5. for as many of the elements:
$$u_0, |g_\#|(u_0), |g_\#|^2(u_0), \ldots$$
as are currently defined (where superscripts denote function composition), for all elements $m \in |\mathcal{M}|$ such that $m \prec_\alpha |g_\#|^i(u_0)$, either $m = u_0$ or $m = |g_\#|^j(u_0)$ for some $j < i$.

For step 0, invariant 3 holds because $U_0 \cup \{u_0\}$ is dispersed minimal by Lemma 6.3 and removing a finite number of elements leaves a dispersed minimal set. Also for step 0, invariant 4 holds because u_0 is $\prec_0$-minimal and $|g_\#|$ is totally undefined. The other invariants hold at step 0 vacuously or by construction.

At stage $\alpha = 2n-1$: If $|g_\#|(m_n)$ is not already defined, define it as follows. Pick $u_i \in U_{2n-2}$ of least index such that $u_i \not\preceq_{2n-2} m_n$; such a u_i must exist since U_{2n-2} is dispersed minimal. Set $|g_\#|(m_n) = u_i$. Set $U_{2n-1} = U_{2n-2} - \{u_i\}$ and $\prec_{2n-1}$ to be the transitive closure of $\prec_{2n-2} \cup \{m_n \prec u_i\}$ (If $|g_\#|(m_n)$ was already defined, set $U_{2n-1} = U_{2n-2}$ and $\prec_{2n-1} = \prec_{2n-2}$.) Note that property (5) is trivially preserved under this definition.

At stage $\alpha = 2n$: If $|g_\#|^{-1}(u_n)$ is not already defined, define u_n to be the first of the following expressions which is currently undefined:

$$|g_\#|(u_0), |g_\#|^2(u_0), |g_\#|^3(u_0), \ldots$$

Thus, for some i, $|g_\#|^i(u_0) = m'$ is defined before stage $2n$ but $|g_\#|(m')$ is not yet defined; define $u_n = |g_\#|(m')$. We need to show that $u_n \not\prec_{2n-1} m'$; if it were, then u_n would already have been defined to be some $|g_\#|^i(u_0)$, and thus $|g_\#|^{-1}(u_n)$ would already have been defined, contradicting the assumption. Set $U_{2n} = U_{2n-1} - \{u_n\}$ and $\prec_{2n}$ to be the transitive closure of $\prec_{2n-1} \cup \{m' \prec u_n\}$. We need to show that it is still true that if some $m \prec_{2n} |g_\#|^j(u_0)$, then for some $k < j$, $m = |g_\#|^k(u_0)$; this is routine. (If $|g_\#|^{-1}(u_n)$ was already defined, set $U_{2n} = U_{2n-1}$ and $\prec_{2n} = \prec_{2n-1}$.)

By Lemma 6.4, $\prec_\alpha$ is a strict partial order on $|\mathcal{M}|$, and U_α is a dispersed minimal set for $\prec_\alpha$.

Let $|g_\#|$ and $\mathcal{M}\#$ be constructed as shown. Now let $\sqsubset_\# = \cup_{\alpha<\omega}\{\prec_\alpha\}$. Then $\sqsubset_\#$ is a strict partial order and demonstrates that $\mathcal{M}\#$ satisfies the acyclicity constraints. ▮

The differences between the previous lemma and the next are (1) the new vocabulary may be infinite, and (2) the new model may contain unnameable elements.

Lemma 6.6: Let ψ be a first order sentence involving just the symbols of the program $(\mathbf{P}, \tilde{\mathbf{P}})$. Let L_P be the vocabulary of $\mathbf{P}$, and let $L_\#$ be that vocabulary augmented with constant $a_\#$ and unary function $g_\#$. Let L_2 be any (at most countable) vocabulary containing at least L_P, and let CEA_2 be based on L_2. If $\neg\psi$ is consistent with $(\mathbf{P}, \tilde{\mathbf{P}}, aug, \text{CEA}\#)$ (that is, $(\mathbf{P}, \tilde{\mathbf{P}}, aug, \text{CEA}\#) \cup \{\neg\psi\}$ has a model), then it is consistent with $(\mathbf{P}, \tilde{\mathbf{P}}, \text{CEA}_2)$.

Proof: By the downward Löwenheim-Skolem theorem, it suffices to restrict attention to countable models. Let $\mathcal{M}\#$ be a countable model of $(\mathbf{P}, \tilde{\mathbf{P}}, aug, \text{CEA}\#) \cup \{\neg\psi\}$ with universe $|\mathcal{M}|$. We shall define $\mathcal{M}_2$ to be an interpretation of $(\mathbf{P}, \tilde{\mathbf{P}}, \text{CEA}_2)$ with the same universe and the same interpretations for the constants, functions, and predicates of $(\mathbf{P}, \tilde{\mathbf{P}})$ as $\mathcal{M}\#$. It follows immediately that $\mathcal{M}_2$ is a model of $(\mathbf{P}, \tilde{\mathbf{P}}) \cup \{\neg\psi\}$, regardless of the interpretations of functions and constants (if any) that do not appear in $\mathbf{P}$. It remains to define interpretations of those symbols that appear only in CEA_2 in such a way that CEA_2 is satisfied.

We define the interpretations of the functions appearing only in CEA_2, not all at once, but one tuple at a time; that is, we shall build nested sequences of partial functions whose unions are functions as desired. Constants can be treated as 0-ary functions in this construction.

Denote by g_{*j} function symbols that appear in CEA_2 but not in $\mathbf{P}$; let $|g_{*j}|$ denote their interpretations in $\mathcal{M}_2$, which we shall specify. Enumerate in any order all $*$-*expressions*: "expressions" of the form $|g_{*j}|(m_1, \ldots, m_k)$, where each $m_n \in |\mathcal{M}|$ for $1 \le n \le k$. Call these $*$-expressions e_i, for $i \ge 1$. Note that j, k, and $m_1, \ldots, m_k$ depend implicitly on i in this notation, and k may be 0.

Define $\prec_0 = \sqsubset_P$ (see Definition 6.3), which is a strict partial order on $|\mathcal{M}|$. Let U_0 be the set of elements of $|\mathcal{M}|$ that are unnameable in L_P. By Lemma 6.3, U_0 is a dispersed minimal set for $\prec_0$. For each expression e_i we will pick a distinct element of U_0 to be its value. This will force all the constants and functions in CEA_2 to be 1-1 and to have disjoint ranges.

We have left to ensure that $\mathcal{M}_2$ satisfies the acyclicity constraint of the Clark equality axioms. To do this, as we build the extension, we build (through stages $\prec_i$) the relation $\sqsubset_2$, as in Definition 6.2, which demonstrates that $\mathcal{M}_2$ satisfies the acyclicity constraints. The following invariants are maintained after each stage i of the construction:

1. Values for $e_1, \ldots, e_i$ have been chosen;
2. $\sqsubset_P \subseteq \prec_i$, and $\prec_i$ is a partial order on $|\mathcal{M}|$;
3. $U_i \subseteq U_0$, and U_i is a dispersed minimal set for $\prec_i$.

They clearly hold for $i = 0$.

For $i > 0$, suppose the invariants hold for $i-1$. We need to assign a value to the $*$-expression $e_i = |g_{*j}|(m_1, \ldots, m_k)$. (Doing so will extend the current partial interpretation $|g_{*j}|$ to include one more tuple in its domain.) Let $C = \{a, m_1, \ldots, m_k\}$. Since U_{i-1} is not weakly dominated under $\prec_{i-1}$ by C, there is a $\prec_{i-1}$-minimal element $m' \in U_{i-1}$ where $m' \not\preceq_{i-1} m_h$ for any $m_h \in C$. Pick one such m' as the value for e_i; set $\prec_i$ to be the transitive closure of $\prec_{i-1} \cup \{(m_1 \prec m'), \ldots, (m_k \prec m')\}$ and set U_i to be $U_{i-1} - \{m'\}$. By Lemma 6.4, $\prec_i$ is a strict partial order on $|\mathcal{M}|$, and U_i is a dispersed minimal set for $\prec_i$.

Let $\sqsubset_2 = \bigcup_{i \geq 0} \prec_i$. It is routine to check that $\sqsubset_2$ is strict partial order and satisfies the criterion given in Definition 6.2 for the model $\mathcal{M}_2$. Thus $\mathcal{M}_2$ satisfies the acyclicity constraints and hence is a model of CEA_2. ∎

Theorem 6.7: Let ψ be a first order sentence involving just the symbols of the program $(\mathbf{P}, \tilde{\mathbf{P}})$. Then ψ is true in the Kunen semantics (that is, it is a logical consequence of $(\mathbf{P}, \tilde{\mathbf{P}}, \mathrm{CEA}*)$) if and only if it is a logical consequence of the augmented program, $(\mathbf{P}, \tilde{\mathbf{P}}, aug, \mathrm{CEA}\#)$. The conclusion also holds if CEA$*$ and CEA# are replaced by any CEA_1 and CEA_2 for L_1 and L_2 being any two expanding vocabularies of L_P.

Proof: The first part of the theorem follows from the contrapositive forms of the two preceding lemmas, instantiating L_1 and L_2 in turn to L_*. The second part follows by transforming from L_1 through $L_\#$ to L_2, using the same lemmas. ∎

7 Structural Constraints on the Domain

In logic programming, the tradition of assuming the universe is an Herbrand model—usually the Herbrand model of the symbols in program—has become fairly well established. For definite clause programs this causes no loss of generality: an atom holds in all models of a definite clause program if and only if it holds in all Herbrand models of the program. But this equivalence breaks down for most semantics of non-definite programs—and even the definitions of the semantics must often be generalized to cover non-Herbrand universes.

It is interesting to observe that Theorems 5.3–5.4 of Section 5 use the domain closure axiom *dca* to ensure the connection between the logical consequences and the existing model-based definitions in the literature. Now *dca* at first seems to be unrelated to the common sense and autoepistemic properties that the *wellsup* axiom is attempting to enforce, so it is natural to wonder if it is necessary or even desirable. Recall that Kunen's semantics does not require such an axiom, yet it has both a logical and a computational formulation (Theorem 4.1), and the logical formulation can be finitely presented (Theorem 6.7).

It is not always clear whether permitting or requiring the program to be interpreted in a universe with infinitely many "unknown" objects agrees with the user's intentions. This question, called the *universal query problem*, is discussed by Przymusinska and Przymusinski [33]. In the context of normal software, it seems clear that this *is* the right idea, and that the domain closure axiom has no place. We want "procedures" to be as independent as possible of their environments. Certainly, we do not want their behavior to change when unrelated procedures, containing new symbols, are added to the software system. By assuming the existence of unknown objects, the semantics essentially "foresees" the addition of the new, unrelated, symbols.

In this section we investigate the impact of constraints on the domain of interpretation that go beyond CEA, but stop short of domain closure. First, we note that one weakened form of *dca* that has been proposed has no effect on augmented programs. Then we formulate a modified version of *dca* that seems to be more natural for logic programs, which we call the *domain foundation axiom*, and show that it does have an effect.

7.1 First Order Approximation

Maher has attributed to "folk lore" the idea of using a first order approximation to the domain closure axiom, which we shall call dca_{fo}. One of the important properties of Kunen's semantics is that its axioms can be expressed in first order logic, and dca_{fo} stays within this framework. Essentially, dca_{fo} states that every element is *nameable* in L_P, i.e., it is in the range of the interpretation of some constant or function symbol of $\mathbf{P}$.

$$dca_{fo} \stackrel{\text{def}}{=} \forall x\, \phi_H[h/true](x)$$

where ϕ_H is given by Definition 4.2. For the vocabulary of a, g, and f that we have been using, dca_{fo} reduces to:

$$\forall x\, ((x = a) \vee \exists y\, [x = g(y)] \vee \exists yz\, [x = f(y, z)])$$

and thus says that every element of the universe either is the interpretation of constant symbol a or is in the range of the interpretation of one of the functions f or g. The variant $dca_{fo}\#$ is defined analogously over the vocabulary that also includes $a_{\#}$ and $g_{\#}$.

The axiom *dca* logically implies dca_{fo}, but the latter does not imply the former. In particular, dca_{fo} permits models containing infinite descending chains (see Section 7.2), such as

$$\ldots \quad z_{-1} = g(z_{-2}), \quad z_0 = g(z_{-1}), \quad z_1 = g(z_0), \quad z_2 = g(z_1), \quad \ldots$$

that are disjoint from the Herbrand universe. However, *dca* disallows such a model.

Theorem 7.1: Let ψ be a first order sentence involving just the symbols of the program $(\mathbf{P}, \tilde{\mathbf{P}})$. Then ψ is a logical consequence of the augmented program, $(\mathbf{P}, \tilde{\mathbf{P}}, aug, \mathrm{CEA}\#)$, if and only if it is a logical consequence of $(\mathbf{P}, \tilde{\mathbf{P}}, aug, \mathrm{CEA}\#, dca_{fo}\#)$.

Proof: The "if" direction is trivial, and the "only if" direction is a corollary of Lemma 6.5. ∎

Another variant is $(\mathbf{P}, \tilde{\mathbf{P}}, aug, \mathrm{CEA}\#, dca\#)$. This can be shown to have different consequences, for appropriate $\mathbf{P}$'s, both from $(\mathbf{P}, \tilde{\mathbf{P}}, \mathrm{CEA}, dca)$, which is Fitting's semantics (see Example 4.1), and from $(\mathbf{P}, \tilde{\mathbf{P}}, aug, \mathrm{CEA}\#)$, which was shown to be the same as Kunen's semantics. In the latter case, the import of $dca\#$ is that it prevents infinite descending chains. This issue is discussed further in Section 7.2.

7.2 Infinite Descending Chains

On many augmented programs (perhaps most realistic ones), *dca* does not affect the logical consequences. That is, $(\mathbf{P}, \tilde{\mathbf{P}}, aug, \mathrm{CEA}\#, wellsup)$ has the same logical consequences with and without *dca*. The next example shows that this is not true in all cases. The fundamental issue is whether a model can have "infinite descending chains", as defined below. Without *dca* such chains are possible.

Definition 7.1: Let $|\mathcal{M}|$ be a structure for the vocabulary L, which contains unary function g and binary function f. Let their interpretations be $|g|$ and $|f|$. An *infinite descending chain* is a sequence $(z_0, z_1, z_2, \ldots)$ of elements of $|\mathcal{M}|$ such that, for all natural numbers k: either $z_k = |f|(z_{k+1}, x)$ for some $x \in |\mathcal{M}|$, or $z_k = |f|(x, z_{k+1})$ for some $x \in |\mathcal{M}|$, or $z_k = |g|(z_{k+1})$. (In a general vocabulary, the disjunction covers all functions and z_{k+1} appears in all of their argument positions. This definition also includes the case where there are an infinite number of function symbols, although that case cannot be stated by a finite-length formula in formal logic.) Note that the z_k need not be distinct. □

Example 7.1: Consider the following program $\mathbf{P}$, presented in the usual logic program notation:

$$\begin{array}{rcl} p(a) & & \\ p(x) & \leftarrow & \neg q(x) \\ q(s(y)) & \leftarrow & \neg p(y) \\ n & \leftarrow & q(x) \end{array}$$

Intuitively, $p(x)$ is intended to mean that there is no infinite descending s-chain from x, while $q(x)$ means the opposite. (The q rule may be read, "Term $s(y)$ has an infinitely descending s-chain if y does not lack one." Note that eliminating q would introduce a universal quantifier in the rule for p.) The dualized version, $(\mathbf{P}, \tilde{\mathbf{P}})$, is:

$$
\begin{array}{rcl}
p(x) & \leftarrow & x = a \vee \tilde{q}(x) \\
q(x) & \leftarrow & \exists y\,[x = s(y)\ \&\ \tilde{p}(y)] \\
n & \leftarrow & \exists x\, q(x) \\
\tilde{p}(x) & \leftarrow & x \neq a\ \&\ q(x) \\
\tilde{q}(x) & \leftarrow & \forall y\,[x \neq s(y) \vee p(y)] \\
\tilde{n} & \leftarrow & \forall x\, \tilde{q}(x)
\end{array}
$$

From $(\mathbf{P}, \tilde{\mathbf{P}}, aug, \text{CEA}, wellsup(\mathbf{P}, \tilde{\mathbf{P}}), dca\#)$ one can infer $p(x)$ and $\tilde{q}(x)$ for each x of the form $s^k(y)$, where y is not in the range of s and $k \geq 0$ is a natural number. But, according to $dca\#$, all elements in the range of s are of that form; hence one can conclude $\tilde{n}$.

On the other hand, without $dca\#$ the rest of the axioms have models in which $\tilde{n}$ is false. For example, let the domain consist of the Herbrand universe of the augmented program, together with $\mathbf{Z}$, the integers (positive and negative). In $\mathbf{Z}$, we interpret s as the usual integer successor, while in the Herbrand universe we interpret s "as itself". Finally, we interpret p and $\tilde{q}$ to be true only in the Herbrand universe, and interpret $\tilde{p}$ and q to be true in $\mathbf{Z}$. Now $\tilde{n}$ may be false.

Looking closely, we see that the axiom *wellsup* has no effect on this program, because none of the (original, normal) rules has a positive recursive subgoal. This phenomenon has been observed before [43], and can be seen by observing that the *unsup* axiom simplifies to $U \subset \tilde{\phi}[q/S]$, as $\tilde{\phi}$ has no tilde atoms. But then $\tilde{\phi}[q/S] = \tilde{S}$ at any fixpoint, and *wellsup* is trivially satisfied.

Thus this example makes the same point for semantics not based on *wellsup*: Even applying domain closure axioms to an augmented vocabulary, which contains additional functions not in $\mathbf{P}$, can change the truth of sentences containing only symbols of $\mathbf{P}$. □

7.3 Domain Foundation Axiom

Having seen that $dca\#$, the domain closure axiom, affects the logical consequences of augmented programs, even with *wellsup* present, it is natural to ask whether it is a "bug" or a "feature". Rules might be interpreted in nonmonotonic logic in many contexts, so there is probably not one correct answer.

In the context of logic programs, uninterpreted function symbols play the role of record constructors, and their names can often be thought of as data types. Conceptually, data structures must be constructed by computation; they have no independent existence (as do the integers). Therefore it seems reasonable to require that any data structure of a known type must be well-founded, that is, must not have any infinitely descending chains.

However, it does not seem necessary or reasonable to require that the constructor can only be applied to objects currently in the program, for then the data type changes when the program changes. Reasoning about data types becomes difficult and of transient value at best in this case.

With these observations as motivation, we propose a new axiom, which we call the *domain foundation axiom* (*dfa*) whose effect is to prevent infinite descending chains, while permitting the universe to contain unnameable elements.

Definition 7.2: Again, assume the program has just a constant a, unary function g, and binary function f. Define

$$\phi_w(x) \stackrel{\text{def}}{=} \forall yz\,[(x = f(y,z)) \rightarrow (w(y)\ \&\ w(z))]\ \&\ \forall y\,[(x = g(y)) \rightarrow w(y)]$$

(In general, there is a conjunction over all nonconstant function symbols of $\mathbf{P}$.) Note that $\phi_w(x)$ is vacuously true if x is not in the range of f or g. $\square$

Recall that the inductive closure of the rule $w(x) \leftarrow \phi_w(x)$ is expressed by the second order formula

$$\forall x\,[w(x) \leftrightarrow \phi_w(x)]\ \&\ \forall R\,[(\phi_w[w/R] \subseteq R) \rightarrow (w \subseteq R)]$$

The second order part of the formula constrains w to be the least fixpoint of (the operator defined by) ϕ_w. This leads to a domain foundation axiom, which effectively forces "known data types" to be well-founded.

Definition 7.3: With ϕ_w as defined in Definition 7.2, the *domain foundation axiom* is:

$$dfa \stackrel{\text{def}}{=} \forall R\,[(\phi_w[w/R] \subseteq R) \rightarrow \forall x\, R(x)]$$

(Thus, like *dca*, this axiom can be stated formally only for a finite number of symbols.) We also denote this axiom by $dfa\#$ when it is defined over the augmented vocabulary $L_\#$. $\square$

Lemma 7.2: A structure for any vocabulary L satisfies *dfa* iff it has no infinite descending chains (see Definition 7.1).

Proof: To simplify notation, assume the functional vocabulary consists of just one unary function symbol g and one binary function symbol f. Suppose that $\mathcal{M}$ is a model of CEA with an infinite descending chain $(z_0, z_1, \ldots)$. Let U be the least fixpoint of ϕ_w (see Definition 7.2). The least fixed point U can be constructed by transfinite induction:

$$U_\alpha = \{x : \forall yz\,[(x = f(y,z) \vee x = g(y)) \rightarrow \exists \eta\,(\eta < \alpha\ \&\ y \in U_\eta\ \&\ z \in U_\eta)]\}$$

for all ordinals α. Now by induction on α, no $z_i \in U_\alpha$. Thus no $z_i \in U$, contradicting *dfa*.

Conversely, suppose $\mathcal{M}$ is not a model of *dfa*. Let U be the least fixed point of ϕ_w (see Definition 7.2). So some element $z_0 = f(x,y)$ of $|\mathcal{M}|$ is not in U. Since $z_0 \notin U$, $x \notin U$ or $y \notin U$; pick z_1 to be either x or y so that $z_1 \notin U$. Continue this way inductively, defining z_{n+1} from z_n as z_1 was defined from z_0. This produces an infinite descending chain. ▮

The domain foundation axiom subsumes the acyclicity constraint of CEA, as shown next. The converse is not true, as was shown by Example 7.1.

Theorem 7.3: A structure that satisfies *dfa* must also satisfy the acyclicity constraint of CEA.

Proof: If $x = t$ is true, where t is a term containing x, then an infinite descending chain can be built by repeatedly traversing the term t (viewed as a tree) from its root to the place where x occurs. ▮

Lemma 7.4: Suppose a structure $\mathcal{M}$ for functional vocabulary L satisfies CEA. Then $\mathcal{M}$ satisfies *dca* if and only if it satisfies *dfa* and dca_{fo}.

Proof: First suppose it satisfies *dca*. Then it must be an isomorphic copy of the Herbrand universe, and by construction the Herbrand universe satisfies dca_{fo} and has no descending chains. By Lemma 7.2 it satisfies *dfa*.

Conversely, suppose $\mathcal{M}$ satisfies *dfa* and dca_{fo}. Since $\mathcal{M}$ satisfies CEA, an isomorphic copy $\mathcal{H}$ of the Herbrand universe is a subset of $|\mathcal{M}|$; the goal is to show that $\mathcal{H}$ is all of $|\mathcal{M}|$. Suppose $x_0 \in |\mathcal{M}| - \mathcal{H}$. Then x_0 cannot be the interpretation of a constant (that would be in $\mathcal{H}$), so, by dca_{fo}, $x_0 = f(y, z)$ for some function symbol f and for some y, z. If both $y, z \in \mathcal{H}$, then $x_0 \in \mathcal{H}$, so either $y \notin \mathcal{H}$ or $z \notin \mathcal{H}$. Set x_1 to be either y or z so that $x_1 \notin \mathcal{H}$. Now repeat the construction – to get $x_2, x_3, \ldots$. This is an infinite descending chain, contradicting Lemma 7.2. ▮

Analogous to Theorem 7.1 is the following Theorem 7.5, which clarifies to what extent the domain closure assumption affects the semantics discussed in this paper, in the presence of *aug* and CEA#. The proof uses the same ideas as the proof of Lemma 6.5.

Theorem 7.5: Let ψ be a first order sentence involving just the symbols of the program $(\mathbf{P}, \tilde{\mathbf{P}})$. Then ψ is true in all models of $(\mathbf{P}, \tilde{\mathbf{P}}, aug, \text{CEA\#}, wellsup(q, \tilde{q}), dfa\#)$ if and only if ψ is true in all models of $(\mathbf{P}, \tilde{\mathbf{P}}, aug, \text{CEA\#}, wellsup(q, \tilde{q}), dca\#)$. Also, ψ is true in all models of $(\mathbf{P}, \tilde{\mathbf{P}}, aug, \text{CEA\#}, dfa\#)$ if and only if ψ is true in all models of $(\mathbf{P}, \tilde{\mathbf{P}}, aug, \text{CEA\#}, dca\#)$.

Proof: The proofs of the two statements are essentially identical; we prove the first. The "if" direction is now immediate by Lemma 7.4. For the "only if" direction, we shall prove the contrapositive form: If $(\mathbf{P}, \tilde{\mathbf{P}}, aug, \text{CEA\#}, wellsup(q, \tilde{q}), dfa\#) \cup \{\neg\psi\}$ has any model, then so does

$$(\mathbf{P}, \tilde{\mathbf{P}}, aug, \text{CEA\#}, wellsup(q, \tilde{q}), dca\#) \cup \{\neg\psi\}$$

Suppose $(\mathbf{P}, \tilde{\mathbf{P}}, aug, \text{CEA\#}, wellsup(q, \tilde{q}), dfa\#) \cup \{\neg\psi\}$ has a model $\mathcal{N}$. Then by Theorem 5.5, $(\mathbf{P}, \tilde{\mathbf{P}}, aug, \text{CEA\#}, wellsup(q, \tilde{q})) \cup \{\neg\psi\}$ has a countable elementary submodel $\mathcal{M}$, with universe $|\mathcal{M}|$.

Moreover, $\mathcal{M}$ is a model of *dfa*#, since any infinite descending chain in $\mathcal{M}$ would also be an infinite descending chain in $\mathcal{N}$, contradicting *dfa*# in $\mathcal{N}$.

By CEA#, infinitely many elements of $|\mathcal{M}|$ are in the ranges of the interpretations of $a_\#$ and $g_\#$, and hence unnameable in L_P. We shall discard the old interpretations of these augmentation symbols and construct new interpretations, $|a_\#|$ and $|g_\#|$, that "cover" *all* unnameable (in L_P) elements. Doing so causes $dca_{fo}\#$ to be satisfied, that is, it makes all elements of $|\mathcal{M}|$ become nameable in $L_\#$.

In the new model, all other symbols will be interpreted as in $\mathcal{M}$; it will be necessary only to make sure that (CEA#, *dfa*#, $dca_{fo}\#$) is satisfied.

Since $|\mathcal{M}|$ contains no infinite descending chains, for each m in $\mathcal{M}$ there are only finitely many m' in $\mathcal{M}$ where $m' \sqsubset_P m$ (by König's Theorem on finitely branching trees). Hence an enumeration $m_1, m_2, \ldots$ of $|\mathcal{M}|$ can be chosen so that if $m_i \sqsubset_P m_j$ (see Definition 6.3), then $i < j$. That is, $\sqsubset_P$ respects subscript order.

As in Lemma 6.5, let $u_0, u_1, u_2, \ldots$ be the subsequence of $m_1, m_2, \ldots$ consisting of elements that are unnameable in L_P. Thus $u_i = m_j$ for some $j \geq i$. Now simply define $|a_\#| = u_0$ and $|g_\#|(m_n) = u_n$ for $n \geq 1$. Beginning with the partial order $\prec = \sqsubset_P$, and adding the relationships $m_n \prec u_n$ maintains the property that $\prec$ respects subscript order. It follows easily that the transitive closure of $\prec$ is a partial order, and has no infinitely descending chains, so *dfa*# is satisfied. By construction, the global injectivity part of CEA# is satisfied and all of $|\mathcal{M}|$ is nameable in $L_\#$, so $dca_{fo}\#$ is satisfied. By Lemma 7.4, *dca*# is satisfied. ∎

8 Conclusion

We have addressed two primary issues is this paper. First, using the tool of the dualized program, we have provided what we feel is a more natural axiomatization of the well-founded and stable semantics. In addition, we have presented finite (albeit sometimes second-order) axiomatizations for several other semantics.

Second, as we noted, the restriction to Herbrand universes in logic programming is becoming suspect. We have examined various weakenings of that restriction, *dca*#, *dfa*, and dca_{fo}. We have compared the formulations of program-completion, stable, and well-founded semantics under differing domain axioms. The relationships are illustrated in Figure 1. Again, we have also provided finite (albeit sometimes second-order) axiomatizations of domain assumptions. In particular, *dfa*, proposed here, seems to merit consideration as the proper domain assumption for logic programming.

8.1 Future Work

The natural (and open) questions concern variations of well-founded and stable semantics that are less closely tied to the Herbrand universe:

The Figure to go here appears at end of manuscript

The caption appears both places; print caption only once

Figure 1: Relationships among the semantics studied; those in the same box have the same logical consequences for first-order sentences in L_P. All semantics include $(\mathbf{P}, \tilde{\mathbf{P}}, \mathrm{CEA})$. An arrow $A \rightarrow B$ indicates that B's models are a subset of A's, but not necessarily that B has greater expressive power. WF-aug abbreviates the augmented well-founded semantics (on $L_{\#}$) and WF-P denotes the well-founded semantics on L_P.

---

1. What sort of semantics are given by the logical consequences of the dualized program plus $wellsup(q, \tilde{q})$ without the additional restriction enforced by *dca* or *dfa*? In most practical cases, we expect this to be the same as the well-founded semantics.

2. Removal of the acyclicity constraint is another possibility, which has been extensively studied, but not in conjunction with $wellsup(q, \tilde{q})$.

3. The corresponding questions about stable semantics.

4. In general, semantics with second order axioms are not recursively enumerable. Can reasonable classes of programs be identified that allow the second order axiom to be replaced by a first order axiom, and thereby drop the complexity to r.e.? This question has been studied in connection with traditional circumscription [22, 18, 17]. A major goal of logic programming is to write programs for computer generation of inferences, and only when the set of inferences is r.e. can all inferences be generated.

5. Procedurally, it seems necessary to label atoms "undefined" in the course of query answering in the well-founded semantics [38]. Should the language of logic programming be extended to allow the user to state $undefined(p(x))$ explicitly as a goal?

Acknowledgments

This attempt to express logic programming semantics in terms of classical logic was motivated by Robert Kowalski's expressed reservations about nonstandard logics and procedural definitions. (However, he has in no way endorsed this effort.) Discussions with Phokion Kolaitis have been helpful to the authors. The anonymous referees made many helpful comments and many suggestions to improve the the presentation of the paper. The first author's research was supported in part by NSF grants CCR-8958590 and IRI-9102513. The second author's research was supported in part by NSF grant IRI-8905166.

References

[1] S. Abiteboul and V. Vianu. Procedural and declarative database update languages. In *ACM Symposium on Principles of Database Systems*, pages 240–250, 1988.

[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] 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.

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

[5] J. Barwise and J. S. Schlipf. An introduction to recursively saturated and resplendent models. *Journal of Symbolic Logic*, 41(2):531–536, 1976.

[6] N. Bidoit and C. Froidevaux. Negation by default and unstratifiable logic programs. *Theoretical Computer Science*, 78(1):85–112, 1991.

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

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

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

[10] S. S. Cosmadakis, H. Gaifman, P. C. Kanellakis, and M. Y. Vardi. Decidable optimization problems for database logic programs. In *ACM Symposium on Theory of Computing*, 1988.

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

[12] 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.

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

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

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

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

[17] P. G. Kolaitis and C. H. Papadimitriou. Some computational aspects of circumscription. In *AAAI Conference on Artificial Intelligence*, pages 465–469, 1988.

[18] T. Krishnaprasad. On the computability of circumscription. *Information Processing Letters*, 27:237–243, 1988.

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

[20] 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).

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

[22] V. Lifschitz. Computing circumscription. In *Ninth Int'l Joint Conference on AI*, pages 121–127, 1985.

[23] V. Lifschitz. Pointwise circumscription: preliminary report. In *AAAI Conference on Artificial Intelligence*, pages 406–410, 1986.

[24] 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.

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

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

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

[28] J. McCarthy. Circumscription – a form of non-monotonic reasoning. *Artificial Intelligence*, 13(1):27–39, 1980.

[29] J. McCarthy. Applications of circumscription to formalizing common sense knowledge. In *AAAI Workshop on Non-Monotonic Reasoning*, pages 295–323, 1984.

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

[31] J. Minker and D. Perlis. Computing protected circumscription. *Journal of Logic Programming*, 2(4):235–249, 1985.

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

[33] H. Przymusinska and T. C. 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.

[34] 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.

[35] 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.

[36] A. Rajasekar and J. Minker. A stratification semantics for general disjunctive programs. In *North American Conference on Logic Programming*, pages 573–586, 1989.

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

[38] 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.

[39] K. A. Ross. The well-founded semantics for disjunctive logic programs. In *First Int'l Conference on Deductive and Object-Oriented Databases*, 1989.

[40] 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.

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

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

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

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

[46] 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.

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

[48] 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.

[49] A. Van Gelder. The alternating fixpoint of logic programs with negation. *Journal of Computer and System Sciences*, 1993. (to appear).

[50] A. Van Gelder, K. A. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.

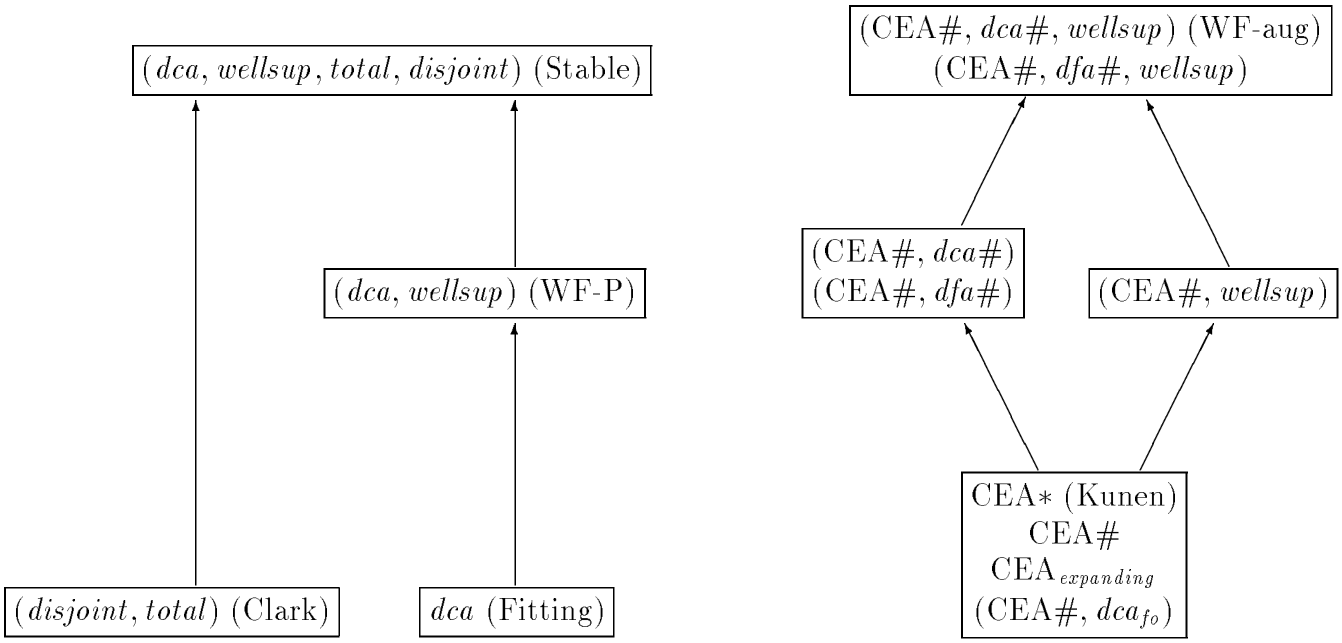

Figure 1. Relationships among the semantics studied; those in the same box have the same logical consequences for first-order sentences in L_P. All semantics include $(\mathbf{P}, \tilde{\mathbf{P}}, \mathrm{CEA})$. An arrow $A \rightarrow B$ indicates that B's models are a subset of A's, but not necessarily that B has greater expressive power. WF-aug abbreviates the augmented well-founded semantics (on $L_{\#}$) and WF-P denotes the well-founded semantics on L_P.