TOWARD A DECLARATIVE SEMANTICS FOR INFINITE OBJECTS IN LOGIC PROGRAMMING*

WILLIAM G. GOLSON

A greatest fixed point characterization of the minimal infinite objects computed by a nonterminating logic program is presented, avoiding difficulties experienced by other attempts in the literature. A minimal infinite object is included in the denotation just when (1) it is successively finitely approximated by a fair infinite computation of the program and (2) any nonterminating computation which continually approximates this object in fact constructs it. Minimal objects are the most general constructible by nonterminating computations of the program.

0. INTRODUCTION

Logic programming has received increased attention over the last decade, due in part to its applicability to a wide variety of areas, including artificial intelligence, natural language processing, and knowledge engineering. More recently, the paradigm has been extended to concurrency with the appearance of a number of stream-oriented parallel logic programming languages, such as Concurrent PROLOG [10], Guarded Horn Clauses [12], and Parlog [2]. However, it has long been recognized that the usual semantic foundations of logic programming, as developed in [1] and [3], are not an appropriate basis for the modeling of concurrent systems. Many of the technical issues concerning the declarative semantics of the languages above, such as synchronization, committed (guarded command) choice, and infinite elements, have yet to be satisfactorily resolved.

This paper focuses on one such issue, the declarative treatment of infinite elements, or the assignment of meaning to nonterminating computations. The standard semantics of [3], which ties together proof-theoretic, model-theoretic, and denotational descriptions, is based on the notion of finite proof. Therefore, nontriv-
ial meanings are reserved for finite computations and all nonterminating computations are rendered meaningless. However, within the contexts of operating systems, where nontermination is not considered harmful in all cases, or lazy computations, where infinite structures abound, meaningful infinite elements arise quite naturally. As a typical example, consider the program

\[ \text{generate}(n, n^\ast l) \leftarrow \text{generate}(\text{suc}(n), l) \]

where \( ^\ast \) is the list constructor. Given the query \( \leftarrow \text{generate}(0, l) \), the resulting computation, despite not terminating, performs useful work, constructing the sequence of integers from 0. Conceptually \( l \) is bound to an infinite list and this computation should have a declarative meaning. The purpose of this paper is to establish a basis for providing such meanings.

We investigate a subclass of the finite and infinite elements constructible by nonterminating computations of a logic program, the minimal objects. (Notions discussed informally here will be made precise in later sections). An object represents a query, or a finite set of atoms, whose variable bindings may be conceptually infinite. Minimal objects are characterized as the most general objects constructible by nonterminating computations of a logic program. Intuitively, an object is minimal just if any derivation which continually approximates it actually computes the object rather than some proper approximation. An object is continually approximated if the initial query, when instantiated with the partial answer substitution constructed after any finite number of derivation steps, is at least as general as the object. For example, any nonterminating computation for the program

\[ p(f(x), y) \leftarrow p(x, y) \]

constructs an infinite term in the first argument position, while the term constructed in the second position depends on the original term in the second position of the query. The minimal object is constructed from the query \( \leftarrow p(x, y) \). Any nonterminating computation arising from \( \leftarrow p(x, t) \), for \( t \) a nonvariable, results in an nonminimal object, as the derivation associated with \( \leftarrow p(x, y) \) constructs a proper approximation.

The paper shows the minimal objects to be the most general ones in the set of objects representing the greatest fixed point of an appropriate functional. While the minimal objects are constructible, not all the other objects in the fixed point are. Those that are, however, appear to stand in a special relationship to the minimal ones, the details of which have yet to be worked out. We are primarily interested in the infinite minimal objects, the most general objects constructible by nonterminating computations which continually refine bindings to initial queries. We believe this class to have potential as a basis for characterizing all meaningful infinite objects in the denotation of a program.

Our approach differs significantly from published treatments of the declarative semantics of infinite objects. If an infinite object is in the denotation of a program, it is necessary that it have finite support, that is, it must be successively approximated by a nonterminating computation. The characterizations in [4], [5], and [9], for example, do not distinguish between an object which has such support and one which does not. Their approach gives the program \( \{ p(x) \leftarrow p(x) \} \) the denotation

\[ \{ p(t) \mid t \text{ is a finite or infinite ground term} \} \]

In our reading, this program, unlike \text{generate}, is performing no useful work and
therefore should be rendered meaningless. Even more telling are the programs

\[
P: \quad p(f(x), f^2(x)) \leftarrow p(x, f(x)) \quad Q: \quad p(f(x), f(x)) \leftarrow p(x, x)
\]

\[
q(x) \leftarrow p(x, f(x)) \quad q(x) \leftarrow p(x, f(x))
\]

The works [4], [5], and [9] do not distinguish between \( P \) and \( Q \), despite the fact that the query \( q(x) \) succeeds in \( P \) (computes an infinite object) and fails in \( Q \); the essential difficulty lies with the reliance on (completed) Herbrand bases. Our approach, which provides a richer representation for the denotations of logic programs, makes such distinctions.

The organization of the paper is as follows. Section 1 presents background material and the basic definitions. Section 2 defines a functional over interpretations and establishes its continuity. The third section contains the main result, namely the characterization of objects occurring in the greatest fixed point of the functional of Section 2. This leads to the definition and characterization of the minimal objects. The paper concludes in Section 4 with a discussion of previous and future work. The last section is an appendix for deferred proofs.

1. PRELIMINARIES

1.0 Background

Logic programs are constructed from the two connectives (\( \leftarrow \) and \( , \)) and a denumerable collection of symbols, denoting variables \( \{x, y, \ldots\} \), constructors \( \{f, g, a, b, \ldots\} \), and predicates \( \{p, q, \ldots\} \). The terms of the language are formed recursively from constructors and variables, and the atoms from the predicate symbols and terms, in the usual manner. Let \( \text{Atom} \) be the collection of atoms, ranged over by \( A, B, \ldots \). The class of programs considered in this paper, denoted by \( \text{Prog} \) and ranged over by \( P \) and \( Q \), consists of collections of definite Horn clauses (rules) subject to the following restrictions:

- the bodies of all rules are nonempty: no unconditional rules are permitted; and
- any variable in the body of a rule also appears in the head.

The first restriction is simply for convenience. This paper is concerned with nonterminating computations, and the language is restricted to focus on them. Finite computations could be accommodated within this framework by introducing a distinguished predicate, say \( \lambda \), denoting \textit{true}. Unconditional rules would be represented by \( A \leftarrow \lambda \), and the rule \( \lambda \leftarrow \lambda \) would be considered part of any program. The second restriction concerning variables is necessary to establish (downward) continuity. However this restriction does not reduce the computing power of the language, since Turing machine computations can be readily specified in \( \text{Prog} \) (with unconditional rules permitted) [11]. See [8] for a related result concerning continuity and this restriction.

Let \( \text{Sub} \) denote the set of substitutions, or mappings from variables to terms in which only a finite number of variables do not map to themselves. Lowercase Greek letters such as \( \sigma, \theta \) range over substitutions. The empty substitution, or the identity map over variables, is denoted by \( \epsilon \). The domain of \( \sigma \) is the finite set of variables
which are not mapped to themselves under \( \sigma \); the range of \( \sigma \) is the image of its domain. \( A\sigma \) denotes, as usual, the atom resulting from simultaneously replacing all variables in \( A \) with the terms prescribed by \( \sigma \). When explicitly describing substitutions, we may write, for example, \( \sigma = \{ x \mapsto t \} \), where \( x \) is different from \( t \), to indicate that \( \sigma(x) = t \) and is the identity elsewhere. Let

\[
\sigma \leq \tau \iff \exists \theta : \sigma \theta = \tau ,
\]

where \( \sigma \theta \), the composition of \( \sigma \) and \( \theta \), denotes the substitution resulting from first applying \( \sigma \) and then \( \theta \):

**Definition (Composition of substitutions).** Let \( \sigma = \{ x_1 \mapsto s_1, \ldots, x_n \mapsto s_n \} \) and \( \theta = \{ y_1 \mapsto t_1, \ldots, y_m \mapsto t_m \} \). Then \( \sigma \theta \) is the substitution obtained by removing elements of the form

\[
y_k \mapsto t_k \text{ where } y_k \text{ is the same variable as some } x_i, \text{ and}
\]

\[
x_i \mapsto x_i
\]

from the set

\[
\{ x_1 \mapsto s_1 \theta, \ldots, x_n \mapsto s_n \theta, y_1 \mapsto t_1, \ldots, y_m \mapsto t_m \}.
\]

For example, if \( \sigma = \{ x \mapsto f(u), y \mapsto g(z), z \mapsto x \} \) and \( \theta = \{ x \mapsto z, u \mapsto f(x), y \mapsto b \} \), then \( \sigma \theta = \{ x \mapsto f(x), y \mapsto g(z), u \mapsto f(x) \} \). Note that \( A(\sigma \theta) = (A\sigma)\theta \) and that substitution composition is associative.

Call a substitution a relabeling when its range consists only of variables and no two members of its domain are mapped to the same variable. Note that \( \sigma_1 \sigma_2 \sigma_3 \sigma_4 = \tau \) if and only if \( \exists \eta, \theta : \sigma_1 \sigma_2 = \eta, \sigma_3 \sigma_4 = \theta \).

In such a case \( \sigma \) and \( \tau \) are said to be \( \leq \)-equivalent. Let \( \text{Fin}(\text{Atom}) \) denote all finite subsets of \( \text{Atom} \), and let \( A, B, \ldots \) range over \( \text{Fin}(\text{Atom}) \). Let

\[
A\sigma = \{ A\sigma | A \in A \}.
\]

We define the substitution ordering over \( \text{Fin}(\text{Atom}) \) as follows:

\[
A \sqsubseteq B \iff \exists \sigma : A\sigma = B.
\]

This ordering, of course, is an extension of the well-known substitution preordering over \( \text{Atom} \), in which \( A \) approximates \( B \) just if \( A\sigma = B \) for some substitution \( \sigma \). We have that

\[
A \sqsubseteq B \sqsubseteq A \iff \text{there exist relabelings } \mu, \rho : A\mu = B \text{ and } A = B\rho ,
\]

that is, \( A \) and \( B \) differ only by an inessential renaming of variables. Here \( A \) and \( B \) are said to be \( \leq \)-equivalent.

### 1.1 Objects

In the standard approaches to declarative semantics, the meaning of a logic program is captured by a subset of either the program's Herbrand base \([3]\) in the finite case or the completion of its base \([4]\) in the infinite one. Our approach employs a richer description, modeling a program by, intuitively, the collection of queries it satisfies. These queries are not necessarily ground, and variables may be bound, conceptually, to infinite terms. As we choose not to explicitly denote infinite terms, a query is
represented by the collection of finite queries which approximate it. The central
notion of this representation is the object. An object represents a particular query, in
which some terms may be conceptually infinite. Recall that a set $S$ is directed with
respect to a preorder $<$ iff

$$x, y \in S \Rightarrow \exists z \in S: x < z \text{ and } y < z,$$

and that $S$ is downward (upward) closed with respect to $<$ iff

$$x \in S \text{ and } y < x \Rightarrow y \in S.$$

**Definition (Object).** An object is an ideal of $\text{Fin}(\text{Atom})$: a subset of $\text{Fin}(\text{Atom})$
which is both $\preceq$-directed and $\preceq$-downward closed. Let $\text{Obj}$ denote the set of
objects, and let $\alpha$ and $\beta$ range over $\text{Obj}$.

The directedness criterion guarantees existence: the object is identifying some
finite set of atoms. Downward closure insures uniqueness of representation (up to
$\preceq$-equivalence). If the object contains just a finite number of $\preceq$-equivalence classes,
then it is the $\preceq$-downward closure of some $A \in \text{Fin}(\text{Atom})$, and hence represents $A$
(or any $B \preceq$-equivalent to $A$). On the other hand, if the object contains an infinite
number of classes, the finite set of atoms it represents has at least one variable
whose binding is being continually refined, i.e., a variable which is bound to an
infinite term.

To better describe objects, we introduce $\text{DSub}$, the collection of nonempty
objects, $\preceq$-directed subsets of $\text{Sub}$. Uppercase Greek letters, such as $\Theta$ and $\Psi$,
range over $\text{DSub}$. For $A \in \text{Fin}(\text{Atom})$ and $\Theta \in \text{DSub}$, let

$$A\Theta = \{ B | \exists \sigma \in \Theta : B \preceq A\sigma \}.$$

One can readily verify that $A\Theta$ is an object; furthermore, every object is of this
form:

**Lemma (Representation of objects).**

$$\forall \alpha \in \text{Obj} \Rightarrow \exists A \in \text{Fin}(\text{Atom}) \exists \Theta \in \text{DSub} : \alpha = A\Theta.$$

**PROOF.** See Appendix. $\square$

Now let $x \in \text{variables}(A)$. $x$ is bound to an infinite term in the object $A\Theta$ just when

$$\{ [\theta_x] \preceq [\theta \in \Theta] \}$$

is an infinite set, where $\theta_x$ denotes the restriction of the domain of $\theta$ to $x$ and $[\ ] \preceq$
denotes an $\preceq$-equivalence class representative over $\text{Sub}$. Note that when (1) is
infinite, the binding to $x$ grows without limit:

$$\forall \sigma \in \Theta \exists \tau \in \Theta : \sigma_x \leq \tau_x \text{ and not } \tau_x \leq \sigma_x.$$

We say $\alpha$ is an infinite object just if $\alpha \models A\Theta$, where the cardinality of

$$\{ [\theta] \preceq [\theta \in \Theta] \}$$

is infinite, where we assume no $\theta \in \Theta$ has a domain variable not occurring in $A$. 

This definition is justified as

\[ (2) \text{ is infinite} \iff (1) \text{ is infinite for some } x \in \text{variables}(A). \]

We turn to some examples of objects.

\[ \alpha = \{ p(x) \} \{ \epsilon \}. \] This object represents \{ p(x) \}. \( \alpha \) itself is quite large, containing for each nonempty finite subset of variables \( V \) the element \{ p(x) \mid x \in V \}.

\[ \beta = \{ p(x) \} \Theta, \text{ where } \Theta = \bigcup_i \{ \sigma_i \} \text{ for } \sigma_i = \{ x \to f'(x) \}. \] Here \( x \) is bound to an infinite object which, if viewed as a tree, has a single infinite strand with each node labeled \( f \).

The representation for infinite objects contains information concerning the construction of infinite terms. For example, the objects \{ p(x, f(x)) \} \( \Theta \) and \{ p(x, x) \} \( \Theta \), for \( \Theta \) above, are distinct. Although one could argue that both represent the binary predicate \( p \) taking the same infinite term in both arguments, the objects should not be identified. Operational contexts distinguish these objects, such as the programs \( P \) and \( Q \) presented in the introduction.

1.2 Interpretations and Minimal Objects

Logic computations yield the most general solution possible. If an object is a solution, that is, represents a query which is satisfied by a program, then any object representing a further refinement or instantiation is a solution as well. An object \( \beta \) is more refined in this sense than an object \( \alpha \) just if every finite approximation to \( \alpha \) is one for \( \beta \) as well, that is, \( \alpha \subseteq \beta \). For the two examples closing the previous subsection we have \( \alpha \subsetneq \beta \).

We define an interpretation to be an \( \subseteq \)-upward closed set of objects: an interpretation is closed under refinement. Let \( \text{Int} \) denote the collection of interpretations, and let \( I \) range over \( \text{Int} \). Let \( \langle \text{Int}, \subseteq \rangle \) denote the complete lattice of interpretations, with bottom element \( \emptyset \) and top element \( T \), the set of all objects. For \( I \in \text{Int} \), let

\[ \text{min } I = \{ \alpha \in I \mid \forall \beta \in I : \beta \subseteq \alpha \implies \alpha = \beta \}. \]

If \( \alpha \in \text{min } I \), then \( \alpha \) is said to be a minimal object with respect to \( I \). Clearly \( \text{min } I \) is the smallest subset of \( I \) whose \( \subseteq \)-upward closure is \( I \).

Note that we do not use the term "interpretation" in the logical sense. However, the terminology is convenient, as \( \text{Int} \) plays a role in the denotational description analogous to that played by logical interpretations in [3].

1.3 Derivations and Rules

A query is a list of atoms, duplications permitted. \( G \) and \( H \) range over queries. The derivations or resolution proofs in this paper rely on a fair computation rule [6], that is, every atom in a query is eventually chosen for resolution. Fairness is essential; otherwise, for example, the program

\[ p(f(x)) \leftarrow p(x), c \]

will have a nonempty denotation rather than the empty one. For definiteness, a (fair) derivation step consists of simultaneously resolving every atom in the current
query. We write \( \langle G_i \rangle \) via \( \langle \theta_i \rangle \) to mean the possibly infinite sequence of fair derivations \( G_i \) from \( G_{i-1} \) using the unifier \( \theta_i \).

A rule of \( P \) is represented by \( A \leftarrow B \), where \( A \) is the single atom rule head and \( B \) is the body. (This representation is slightly abusive, as the body should be a list, not a set.) We say \( C \leftarrow D \) is a variant of \( A \leftarrow B \) if it is obtained from \( A \leftarrow B \) just by an inessential renaming of variables. Let \( \text{Var}(P) \) denote the variants of the rules of \( P \), and let \( \text{Fin}(\text{Var}(P)) \) be the finite subsets of \( \text{Var}(P) \). No two elements of \( \text{Var}(P) \) share the same variable. We assume that the variables in the rule variant used at a derivation step are disjoint from those in the query.

2. A TRANSFORMATION OVER INTERPRETATIONS

We come to the crucial definitions.

Definition. Let \( \vdash_p \subseteq \text{Int} \times \text{Obj} \) be defined by
\[
I \vdash_p \alpha \iff \exists \Theta \in \text{DS} \cup \{ A_i \leftarrow B_i \} \in \text{Fin}(\text{Var}(P)) : \alpha = \Theta A \land B \Theta \in I,
\]
where \( A = \bigcup \{ A_i \} \) and \( B = \bigcup B_i \).

The relation \( \vdash_p \) means that \( \alpha \) is deducible from an object in \( I \) in a single fair derivation step. For convenience, the set \( \bigcup \{ A_i \leftarrow B_i \} \) is always assumed to be nonempty.

Definition. Let \( S_p : \text{Int} \to \text{Int} \) by \( S_p(I) = \{ \alpha | I \vdash_p \alpha \} \).

The above are well defined. The subscript \( P \) may be omitted from \( \vdash_p \) and \( S_p \) when the context is clear. \( S_p \) operates over a richer notion of interpretation than the standard approaches, namely \( \subseteq \)-upward closed sets of objects rather than subsets of the Herbrand base, or, intuitively, sets of queries, not necessarily ground, rather than sets of ground atoms.

We turn to the downward continuity of \( S_p \). As an immediate corollary of the following theorem, we have that the greatest fixed point of \( S_p \), \( \text{gfp}(S_p) \), exists and is equal to \( \cap S_p^i(\text{T}) \).

Theorem (Continuity of \( S_p \)). \( S_p \) is downward continuous in \( \langle \text{Int}, \subseteq \rangle \).

Proof. We have for a \( \supseteq \)-chain \( \langle I_k \rangle \)
\[
\alpha \in S_p(\bigcap I_k) \iff \bigcap I_k \vdash \alpha 
\iff \exists \Theta \exists \bigcup \{ A_i \leftarrow B_i \} : A \Theta = \alpha \land B \Theta \in \bigcap I_k
\iff \exists \Theta \exists \bigcup \{ A_i \leftarrow B_i \} \forall k : A \Theta = \alpha \land B \Theta \in I_k
\]
where \( A = \bigcup \{ A_i \} \) and \( B = \bigcup B_i \), and
\[
\alpha \in \bigcap S_p(I_k) \iff \forall k : \alpha \in S_p(I_k) \iff \forall k : I_k \vdash \alpha
\iff \forall k \exists \Theta_k \exists \bigcup \{ A_{k,i} \leftarrow B_{k,i} \} : A_k \Theta_k = \alpha \land B_k \Theta_k \in I_k,
\]
where \( A_k = \bigcup \{ A_{k,i} \} \) and \( B_k = \bigcup B_{k,i} \).

We show the equivalence of (1) and (2). The proof of (1) \( \Rightarrow \) (2) is immediate. To prove (2) \( \Rightarrow \) (1) we make uniform choices for the rule variants and substitutions. For
the rule variants, we first show the number of variants required in (2) can be bounded independent of \( k \). Partition \( A_k \) into atoms identified by \( \Theta_k \), that is, atoms \( A \) and \( A' \) of \( A_k \) are in the same partition just if \( \{ A \} \Theta_k = \{ A' \} \Theta_k \). Clearly the number of such partitions is determined by \( \alpha \), independent of a particular choice for \( A_k \) and \( \Theta_k \). Therefore the number of rule variants used for each \( k \) can be assumed to be uniformly bounded, since only one variant is necessary to represent each partition.

Furthermore, since the number of rules of \( P \) is finite, we can assume infinitely many \( k \) utilize the same set of rule variants, up to an inessential renaming of variables. Assume without loss of generality that this set is used for every \( k \). Let \( A \) and \( B \) denote the usual values with respect to this set of rule variants.

Turning to choice of \( \Theta \), we claim that any \( \Theta_k \) will suffice. Certainly \( A \Theta \) is the same for every \( k \) (each one is equal to \( \alpha \)). \( B \Theta \) is the same for every \( k \) also, due to the language restriction that variables in the body of a rule appear in the head as well. Hence we are done, as \( B \Theta_k \) belongs to every member of the \( \sqsupset \)-chain. \( \Box \)

The following illustrates that the language restriction concerning variables in rule bodies is essential for continuity. Let

\[
P = \{ p(x) \leftarrow q(x, y) \} \quad \text{and} \quad \alpha_k = \{ q(b, f^k(a)) \{ \epsilon \} ,
\]

where the terms \( a \) and \( b \) are ground. Note that \( P \) does not honor the language restriction. Since \( \alpha_k \) represents a finite ground term, no objects are larger than \( \alpha_k \) under \( \subseteq \). Therefore the set

\[
I_k = \{ \alpha_i | i \geq k \}
\]

is an interpretation for every \( k \), and \( \langle I_k \rangle \) forms a \( \sqsupset \)-chain in \( \text{Int} \). Clearly \( \cap I_k = \emptyset \), and so \( S_\rho(\cap I_k) = \emptyset \) as well, However, it is easily verified that

\[
\forall k: S_\rho(I_k) = \{ \{ p(b) \{ \epsilon \} \}
\]

and so \( \cap S_\rho(I_k) \) is nonempty.

3. CHARACTERIZATION OF MINIMAL OBJECTS

In this section we present the main result, a characterization of the minimal objects. The initial task is to characterize when an object is in the greatest fixed point of \( S_\rho \), and the first subsection presents a preliminary lemma and a theorem to achieve this. The minimal objects are characterized in the next subsection. They turn out to be those objects constructed by nonterminating computations from queries consisting of rule head variants or, more generally, queries without constructors or duplicate variables. Furthermore, any derivation continually \( \subseteq \)-approximating a minimal object constructs it rather than some proper approximation. The last subsection presents some examples.

3.0 Objects in the Greatest Fixed Point of \( S_\rho \)

Below, when we write \( \sigma_1 \cdots \sigma_k \) we mean the substitution resulting from the composition of \( \sigma_1 \) through \( \sigma_k \). For a query \( G \), let \( G\downarrow \) be the set consisting of the members of \( G \). By \( G\sigma \), we mean that list resulting from applying \( \sigma \) to each member. For \( A \in \text{Fin(Atom)} \), let \( G(A) \) be a query obtained by turning \( A \) into a list. By an expansion of \( G \), we mean a list \( G' \) obtained from \( G \) by duplication of elements.
To establish when an object is in the greatest fixed point of \( S_p \), we first determine when it is in \( S^k_p(T) \):

**Lemma.** \( \alpha \in S^{k+1}_p(T) \iff \exists \langle G_r \rangle \text{ via } \langle \sigma_r \rangle, \) a most general derivation of length \( k + 1 \), such that \( G_{r+1} \Phi \subseteq \alpha \), where \( \Phi = \{ \sigma_1 \cdots \sigma_{k+1} \} \). Furthermore, every member of \( G_0 \) is a distinct rule head variant of \( P \).

**Proof.** The issues of a most general derivation and the composition of \( G_0 \) are faced at the end of the proof. In the meantime, no such constraints are placed upon the derivations.

\( \Rightarrow \): We proceed by induction over \( k \).

**Case** \( k = 0 \):

\[ \alpha \in S(T) \iff T \vdash \alpha \]

\[ \exists \Theta \exists \mathcal{U} \{ A_i \leftarrow B_i \} : \alpha = \Theta \mathcal{U} \]

where \( \mathcal{A} = \mathcal{U} \{ A_i \} \). Use \( \langle G(A), G(B) \rangle \text{ via } \langle \epsilon \rangle \), where \( B = \mathcal{U} B_i \).

**Case** \( k \):

\[ \alpha \in S^{k+1}(T) \iff S^k(T) \vdash \alpha \]

\[ \exists \Theta \exists \mathcal{U} \{ A_i \leftarrow B_i \} : \alpha = \mathcal{A} \Theta \text{ and } \Theta \mathcal{B} \in S^k(T) \]

where \( \mathcal{A} = \mathcal{U} \{ A_i \} \) and \( \mathcal{B} = \mathcal{U} B_i \). By the induction hypothesis,

\[ \Theta \mathcal{B} \in S^k(T) \Rightarrow \exists \langle H_r \rangle \text{ via } \langle \tau_r \rangle \text{ of length } k : H_0 \downarrow \Phi \subseteq \Theta \mathcal{B}, \quad (*) \]

where \( \Phi = \{ \tau_1 \cdots \tau_k \} \).

Let \( \mu_i = \tau_i \cdots \tau_k \). Since \( H_0 \downarrow \Phi \subseteq \Theta \mathcal{B} \), we have \( H_0 \downarrow \mu_1 \subseteq \Theta \mathcal{B} \). The desired derivation is constructed as follows. Note that

\[ \langle H_0 \mu_1, H_1 \mu_2, \ldots, H_k \rangle \text{ via } \langle \mu_1, \ldots, \mu_k \rangle \]

is a derivation which yields the same bindings to the variables in \( H_0 \) as the one in \((*)\). In addition, unification is one-way: no variables are bound in any query during the derivation; all bindings are made to clause head variables. Therefore, for any \( \phi \),

\[ \langle H_0 \mu_1 \phi, \ldots, H_k \phi \rangle \text{ via } \langle \mu_1 \phi, \ldots, \mu_k \phi \rangle \]

is a derivation as well.

Choose \( \phi \) such that \( H_0 \downarrow \mu_1 \phi = \Theta \mathcal{B} \cdot \sigma \). Let \( H'_0 \) be \( H_0 \mu_1 \phi \) expanded, by duplication of elements, to include an integral number of copies of \( G(B) \). We have

\[ \langle H'_0, H'_1 \mu_2 \phi, \ldots, H'_k \phi \rangle \text{ via } \langle \mu_1 \phi, \ldots, \mu_k \phi \rangle, \]

where \( H'_i \) is \( H_i \) expanded as necessary to accommodate \( H'_0 \). Let \( G \) be the expansion of \( G(A) \) such that

\[ \langle G, H'_0, \ldots, H'_k \phi \rangle \text{ via } \langle \sigma, \mu_1 \phi, \ldots, \mu_k \phi \rangle. \]

To show this is the derivation we seek, we verify that \( G \downarrow \{ \sigma \mu_1 \phi \cdots \mu_k \phi \} \subseteq \alpha \). Note that bindings to variables in \( G \) are made in \( \sigma \) only, as the rest of the computation makes bindings only to variables in rule heads. Therefore, \( G \downarrow \{ \sigma \mu_1 \phi \cdots \mu_k \phi \} = G \downarrow \{ \sigma \} = A \{ \sigma \} \). As \( \sigma \in \Theta \), it follows \( A \sigma \in A \Theta = \alpha \), and we are done with the proof from left to right.

\( \Leftarrow \): Let \( \langle G_r \rangle \text{ via } \langle \sigma_r \rangle \) be a derivation of length \( k + 1 \), and let \( \Phi = \{ \sigma_1 \cdots \sigma_{k+1} \} \).

We show that \( G_0 \downarrow \Phi \in S^{k+1}(T) \), which is sufficient as \( S^{k+1}(T) \) is upward closed.
under \( \subseteq \). By definition,
\[
G_0 \downarrow \Phi \in S^{k+1}(T) \iff S^k(T) \vdash G_0 \downarrow \Phi \\
\iff \exists \Theta \exists \{ A_i \leftarrow B_i \} : G_0 \downarrow \Phi = A \Theta \text{ and } B \Theta \in S^k(T),
\]
where \( A = \bigcup \{ A_i \} \) and \( B = \bigcup B_i \). The proof continues by induction over \( k \).

**Case** \( k = 0 \): Since \( \langle G_0, G_1 \rangle \) via \( \langle \alpha \rangle \), all that is required is that \( B \{\sigma_1\} \in T \), which is immediate.

**Case** \( k \): For this case, take the derivation beginning with \( G_1 \) and apply the induction hypothesis, obtaining \( G_1 \downarrow \Psi \in S^k(T) \) for \( \Psi = \{ \sigma_2 \cdots \sigma_{k+1} \} \). Since \( \langle G_0, G_1 \rangle \) via \( \langle \sigma_1 \rangle \), select an appropriate set of rule variants and choose \( \Phi \) for \( \Theta \). Then as \( B \sigma_1 = G_1 \downarrow \), we have
\[
B \Theta = B \Phi = (B \sigma_1) \Psi = G_1 \downarrow \Psi \in S^k(T),
\]
completing the proof from right to left.

Now we establish that the derivations constructed can be assumed to be such that

- the initial query is a set of rule head variants, and
- the unifiers are most general.

Let \( \langle G_i \rangle \) via \( \langle \sigma_i \rangle \) be the derivation of length \( k + 1 \) constructed from a given \( \alpha \). Let \( \Phi = \{ \sigma_1 \cdots \sigma_{k+1} \} \), so that we have \( G_0 \downarrow \Phi \subseteq \alpha \). Let \( A \) be the set of rule head variants used in the first step. As \( G(A) \sigma_1 = G_0 \sigma_1 \), we can construct \( \langle G(A), G_1, \cdots \rangle \) via \( \langle \sigma_1 \rangle \), a derivation whose initial query contains just distinct rule head variants of \( P \). Furthermore \( A \Phi \subseteq \alpha \).

Next, as demonstrated in Lemma 8.1 of [7], one can construct from an arbitrary derivation a most general one such that

- the initial queries of both are identical; and
- if \( \langle \sigma_i \rangle \) and \( \langle \tau_i \rangle \) are the initial and most general unifiers respectively, then
  \[
  \forall k : \tau_1 \cdots \tau_k \leq \sigma_1 \cdots \sigma_k.
  \]

For the derivation \( \langle G(A), G_1, \cdots \rangle \), let \( \langle H_i \rangle \) via \( \langle \tau_i \rangle \) be the associated most general one, where \( H_0 = G(A) \). For this new derivation, we must show that
\[
H_0 \downarrow \Psi \subseteq \alpha, \text{ where } \Psi = \{ \tau_1 \cdots \tau_{k+1} \}. \text{ Using } \tau_1 \cdots \tau_{k+1} \leq \sigma_1 \cdots \sigma_{k+1}, \text{ we have}
\]
\[
H_0 \downarrow \Psi = A \Psi \subseteq A \Phi \subseteq \alpha
\]
as desired. \( \square \)

Now we can establish the

**Theorem.** \( \alpha \in gfp(S_p) \Leftrightarrow \) there is a most general derivation \( \langle G_i \rangle \) via \( \langle \sigma_i \rangle \), such that
\[
G_0 \downarrow \Phi \subseteq \alpha, \text{ where } \Phi = \bigcup \{ \sigma_1 \cdots \sigma_i \}. \text{ Furthermore, } G_0 \text{ is a collection of distinct rule head variants of } P.
\]

**Proof**

\[
\alpha \in gfp(S_p) \iff \alpha \in \cap S^k(T) \\
\iff \forall k > 0 : \alpha \in S^k(T) \ (k > 0, \text{ as } \alpha \in T \text{ always}) \\
\iff \forall k > 0 \exists \langle H_k, \rangle \text{ via } \langle \sigma_{k,i} \rangle \text{ (most general) of length } k : H_{k,0} \downarrow \Phi_k \subseteq \alpha \text{ where } \Phi_k = \{ \sigma_{k,1} \cdots \sigma_{k,k} \}, \text{ and } H_{k,0} \text{ consists of distinct rule head variants.}
\]
The last equivalence follows, of course, by application of the lemma. The proof of the theorem from right to left is now immediate.

As for the other direction, first we claim that the size of the initial queries of each of the derivations \( \langle H_{k,i} \rangle \) can be assumed to be uniformly bounded, depending only on \( \alpha \). Represent \( \alpha \) by \( A\theta \), where \( A \) is as constructed in the proof of the representation lemma. In particular, the cardinality of \( A \) is minimal over all other representations for \( \alpha \):

\[
\alpha = \mathbf{B} \Psi \quad \Rightarrow \quad \text{card}(A) \leq \text{card}(B).
\]

Let \( D\Phi \subseteq \alpha \), where \( D\Phi \) is the result of a computation approximating \( \alpha \). We want to construct \( C \subseteq D \) such that \( \text{card}(C) = \text{card}(A) \) and \( C\Phi \subseteq A\theta \). This will satisfy the claim, as a most general derivation from \( C \) approximating \( \alpha \) can readily be constructed, as evidenced in the proof of the previous lemma. For each \( \phi \in \Phi \) let

\[
\text{candidates}(D, \phi) = \{ C \subseteq D \mid \text{card}(C) = \text{card}(A) \text{ and } C\phi \subseteq A\theta \text{ for some } \theta \in \Theta \}.
\]

\( \text{Candidates}(D, \phi) \) is nonempty: \( D \Phi \subseteq A\theta \) plus \( \text{card}(A\theta) = \text{card}(A) \) readily gives us some \( C \). Assume \( \Theta \) is an infinite set. Then some \( C \) occurs in infinitely many of the candidate sets. It follows easily that \( C\Phi \subseteq A\Theta \). If \( \Phi \) is finite, let \( \psi \in \Phi \) be its upper bound. Any member of \( \text{candidates}(D, \psi) \) will suffice in this case.

Therefore, a uniform bound on the initial query sizes of each of the derivations \( \langle H_{k,i} \rangle \). Collect these derivations \( \langle H_{k,i} \rangle \) into \( \Sigma \). We construct an infinite derivation with the desired properties from the derivations in \( \Sigma \).

**Step 0:** Due to the uniform bound on initial query sizes and the fact that the number of rules of \( P \) is finite, some set of rule head variants (up to an inessential renaming of variables) appears as the initial query in infinitely many of the derivations in \( \Sigma \). Let \( G_0 \) be this list of rule head variants, and let \( \Sigma_0 \) be the collection of derivations in \( \Sigma \) beginning with \( G_0 \).

**Step n + 1:** Let \( \langle G_n \rangle \) via an appropriate mgu sequence be the derivation constructed so far, one which is followed by every derivation in the infinite set \( \Sigma_n \). As all derivations use only most general unifiers and as the number of rules of \( P \) is finite, one query, say \( G_{n+1} \), appears at the \( n + 2 \)nd position in infinitely many of the derivations in \( \Sigma_n \). Let \( \Sigma_{n+1} \) be just these derivations.

In this fashion, an infinite most general computation, \( \langle G_i \rangle \) via \( \langle \sigma_i \rangle \), is obtained, where \( G_0 \) consists of just rule head variants. This is the desired derivation: since by construction \( G_0 \vdash \{ \sigma_1 \cdots \sigma_k \} \subseteq \alpha \) for every \( k \), we have \( G_0 \vdash \Phi \subseteq \alpha \), as required, where \( \Phi = \bigcup\{ \sigma_1 \cdots \sigma_i \} \). □

### 3.1 Minimal Objects in the Greatest Fixed Point of \( S_P \)

Recall that

\[
\text{min } I = \{ \alpha \in I \mid \forall \beta \in I : \beta \subseteq \alpha \rightarrow \alpha = \beta \}.
\]

The minimal objects with respect to \( \text{gfp}(S_P) \) are characterized as follows:

**Corollary.** \( \alpha \in \text{min } \text{gfp}(S_P) \iff \exists \langle G_i \rangle \text{ via } \langle \sigma_i \rangle \text{ (most general): } G_0 \vdash \Phi = \alpha , \text{ where } \Phi = \bigcup\{ \sigma_1 \cdots \sigma_i \} \text{ and } G_0 \text{ is a collection of distinct rule head variants of } P. \text{ Furthermore, any derivation which } \subseteq \text{-approximates } \alpha \text{ actually computes it (rather than some proper approximation).} \)
PROOF. ⇒: Let \( \alpha \in \text{min} \text{gfp}(S_p) \). Then certainly \( \alpha \in \text{gfp}(S_p) \) and so, by the theorem, for some appropriate derivation we have \( G_0 \downarrow \Phi \subseteq \alpha \). Therefore \( G_0 \downarrow \Phi \subseteq \text{gfp}(S_p) \) by the theorem again, and equality follows by minimality. Finally suppose a derivation computes \( \beta \subseteq \alpha \). By the theorem, \( \beta \in \text{gfp}(S_p) \) and we have \( \beta = \alpha \) by minimality.

⇐: Given \( G_0 \downarrow \Phi = \alpha \), we have \( \alpha \in \text{gfp}(S_p) \) by the theorem. If \( \alpha \) is not minimal, then there is some \( \beta \) and associated derivation \( \langle H, \psi_i \rangle \) via \( \langle J, \xi \rangle \) such that \( H_0 \downarrow \Psi \subseteq \beta \subseteq \alpha \) for \( \Psi = \bigcup \{ \psi_1 \cdots \psi_i \} \). By assumption, however, \( H_0 \downarrow \Psi = \alpha \) and so \( \alpha \) is minimal.

The \textit{minimal infinite objects} are those infinite objects in \( \text{min} \text{gfp}(S_p) \). These are the most general objects constructible by nonterminating computations which continually refine bindings to initial queries.

3.2 Examples

For the program \( p(x) \leftarrow p(x) \), we have
\[
\{ p(x) \} \{ \epsilon \} \in S^k(T) \quad \text{for every } k
\]
and so
\[
\{ \{ p(x) \} \{ \epsilon \} \} = \text{min} \cap S^k(T),
\]
and hence no infinite objects are associated with the program.

For the program \( p(f(x)) \leftarrow p(x) \), we have
\[
\{ p(x) \} \{ \sigma_i \} \in S^k(T) \quad \iff \quad i \geq k
\]
where \( \sigma_i(x) = f^i(x) \) and is the identity elsewhere. Let \( \Theta = \bigcup \{ \sigma_i \} \). Therefore
\[
\{ p(x) \} \Theta \in S^k(T) \quad \text{for every } k
\]
and so
\[
\{ \{ p(x) \} \Theta \} = \text{min} \cap S^k(T).
\]
Therefore, \( \{ p(x) \} \Theta \) is computed at infinity, intuitively \( p \) with infinite argument "fff....".

For the program \( P \) of the introduction, we have, where \( \sigma_i \) and \( \Theta \) are as above,
\[
\{ p(x, f(x)) \} \{ \sigma_i \} \in S^k(T) \quad \iff \quad i \geq k
\]
and so
\[
\{ q(x) \} \{ \sigma_i \} \in S^{k+1}(T) \quad \iff \quad i \geq k.
\]
Both \( \{ p(x, f(x)) \} \Theta \) and \( \{ q(x) \} \Theta \) are computed at infinity.

For the program \( Q \) of the introduction, we have that \( \{ p(f(x), f(x)) \} \Theta \) is computed at infinity. However, the atom \( q \) does not appear in the greatest fixed point.

4. PREVIOUS AND FUTURE WORK

In previous work, the companion papers of [4] and [9] extend the semantics of logic programming to infinite computations via topological methods, representing atomic formulae as trees and endowing the space of trees with a metric measuring the depth at which two trees differ. The infinite fair computations are characterized as the
greatest fixed point of a functional $T_p$, similar to the one introduced in [3].

$$T_p(I) = \{ A \in \mathbb{B}_p^\infty \mid A \leftarrow B_1, \ldots, B_n \text{ is a ground instance} \}
\quad \text{of a clause in } P \text{ and } \forall i \in I \},$$

where $\mathbb{B}_p^\infty$ represents the completion of the Herbrand base of $P$ to include ground atomic formulae with infinite terms. However, as mentioned in the introduction, this characterization does not distinguish between an infinite object which has support (that is, an infinite derivation which successively approximates the object) and one which does not. In other cases, such as programs $P$ and $Q$ of the introduction, distinctions are not made which should be. In [7], a notion of computation at infinity is introduced and the infinite computations are characterized utilizing the same domain as [4]. However, as the author indicates, a disparity exists between this characterization of infinite computation and the fixed points of $T_p$. We believe the root of the difficulty lies not with the notion of computation at infinity, which is similar in spirit to ours, but rather with the denotational setting. In summary, these previous results suggest that a representation richer than the Herbrand base is appropriate when modeling infinite objects.

The fixed point semantics for infinite terms in [5] is an approach in an entirely different vein. The authors provide a semantics for lazy evaluation through a language containing both finite and infinite sorts. For each infinite sort a new constant representing the "rest of the computation" is introduced. Programs are augmented with terminal clauses containing this constant to permit the suspension (termination) of computations. While intriguing, the semantics of [5] utilizes Herbrand bases and does not distinguish between the programs $P$ and $Q$ of the introduction, recast in their setting.

As to future work, we wish to strengthen our results by characterizing all infinite object computed by a logic program. The minimal ones should form a basis from which the more general ones can be deduced. For example, consider the program

$$p(f(x), y) \leftarrow p(x, y).$$

In the following, by "$p(f^\omega, y)$" we mean an object such as $\{ p(x, y) \} \Theta$ for $\Theta = \{ s \}$ and $s(x) = f(x)$; other such "atoms" should be similarly interpreted. The object "$p(f^\omega, y)$" is minimal in the greatest fixed point, and certainly all derivations from the query $\leftarrow p(x, y)$ compute this object. However, "$p(f^\omega, a)$" should also be considered an infinite object computed by the program, as evidenced by derivations from the query $\leftarrow p(x, a)$. In fact, "$p(f^\omega, f^\omega)$" is computed as well, evidenced by $\leftarrow p(x, x)$. However, note that characterizing all objects constructible by a program is not just a matter of closing $\text{mingfp}(S_p)$ up under arbitrary instantiation: for example, "$p(f^\omega, g^\omega)$" is certainly not constructible by the program above. This question is currently under investigation.

5. APPENDIX

We present the proof of the representation lemma, deferred from a previous section.

Lemma (Representation of objects).

$$\alpha \in \text{Obj} \iff \exists A \in \text{Fin(Atom)} \exists \Theta \subseteq \text{DSub} : \alpha = A\Theta.$$ 

Proof. $\Leftarrow$: straightforward.

$\Rightarrow$: Let $\alpha \in \text{Obj}$. Note that the collection of predicate symbols which occur in every $B \in \alpha$ is the same, say $p_1, \ldots, p_n$. Let $\text{min}_\alpha$ represent the minimal number of
occurrences of the predicate symbol \( p_i \) in any \( B \in \alpha \). Define \( A \) to be an element of \( \text{Fin}(\text{Atom}) \) which contains, for each \( i \), exactly \( \min_i \) instances of the symbol \( p_i \) of the form \( p_i(x_1, \ldots, x_k) \), where the arity of \( p_i \) is \( k \). All variables appearing in \( A \) are unique.

Let \( \text{minsym} \subseteq \alpha \) be the set of elements which contain exactly the minimal number of predicate symbols for each \( i \). Note that for \( A \) above,

\[ \forall B \in \alpha : A \sqsubseteq B \iff B \in \text{minsym}. \]

We claim \( \text{minsym} \) is both nonempty and \( \sqsubseteq \)-directed. There exists \( U\{C_i\} \subseteq \alpha \) such that each \( C_i \) contains the minimal number of occurrences of the predicate symbols \( p_i \). As \( \alpha \) is \( \sqsubseteq \)-directed, \( U\{C_i\} \) has an \( \sqsubseteq \)-upper bound in \( \alpha \). As any upper bound of \( U\{C_i\} \) by necessity contains the minimal number of each predicate symbol, \( \text{minsym} \) is nonempty. Furthermore, as any upper bound of an element in \( \text{minsym} \) is in \( \text{minsym} \) as well, it is clear that \( \text{minsym} \) is \( \sqsubseteq \)-directed. Define

\[ \Theta = \{ a \mid \exists B \in \text{minsym} : Aa = B \}, \]

where the domain of each \( a \) is restricted to the variables of \( A \). As \( \text{minsym} \) is nonempty and directed, \( \Theta \) is also (with respect to \( \subseteq \)), and therefore is a member of \( D\text{Sub} \). We have

\[ A\Theta = \{ C \mid \exists a \in \Theta : C \subseteq Aa \} = \{ C \mid \exists B \in \text{minsym} : C \subseteq B \}. \]

We show \( A\Theta = \alpha \). \( A\Theta \subseteq \alpha \) is immediate. Now let \( C \in \alpha \). Let \( D \in \text{minsym} \), and let \( B \in \alpha \) be an \( \subseteq \)-upper bound of \( C \) and \( D \). Therefore \( B \in \text{minsym} \), and we have \( C \in A\Theta \). \( \Box \)

REFERENCES