LOGIC PROGRAMMING SEMANTICS USING A COMPACT DATA STRUCTURE

Melvin Fitting
The Graduate School and University Center (CUNY), and
Herbert H. Lehman College (CUNY)
Department of Mathematics and Computer Science
Bedford Park Boulevard West
Bronx, New York 10468

ABSTRACT: A fixpoint semantics is given for logic programming using domain theory, with undefined as one truth value, allowing negation, and arbitrary data structures. This generalizes the conventional semantics, and agrees with it on 'successes' for Horn clause programs. Consequences of requiring the data structure to be a compact topological space and the given relations to be continuous are investigated, extending results in the last chapter of Lloyd's book.

KEYWORDS: logic programming, fixpoint, semantics, compact, Scott topology.

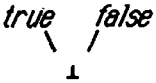
S1 Introduction

In the last chapter of [10], based on [1], a theory of logic programming "perpetual processes" (such as operating systems) is presented. Think of a perpetual process as computing an infinite term via a sequence of finite approximations. But the development in [10] is based on two-valued logic, which in many ways is far from ideal. A program like $P \leftarrow P$ can not be thought of as assigning either *true* or *false* as a value for P; $\perp$ (undefined) is the only thing that makes sense because of infinite regress. A break from the two-valued position is found in [9] and [11]. In [4] we argued for a semantics allowing $\perp$ as a value, showing that it yielded a smooth treatment of negation. Here we extend this to perpetual proceses. We use machinery from so-called domain theory, thus making logic programming semantics a part of program semantics generaly. We also generalize beyond the terms of formal logic. Commercial Prologs make available other data structures, such as integers, finite decimals and character strings. These can be identified with certain terms since the family of terms constitutes a "universal" data structure. But such an identification leaves no place for typed predicates, or for issues of implementation. Primarily, though, a restriction of theory to formal terms is entirely unnecessary. Logic programming semantics develops as nicely as ever when applied to arbitrary data structures. But then, what plays the role of infinite terms for a theory of perpetual processes? We show that any compact data structure (definition in S5) will do quite well. Infinite terms provide one example, but there are others, equally natural, such as infinite words, or real numbers.

I want to thank Paul Meyer for help with the topological aspects of this paper, and in particular for supplying the lemma for Proposition 1, and its proof.

S2 The class of truth values, background

By *Bool* we mean the partial ordering



Bool, simple as it is, has a rich algebraic and topological life. It is a complete partial ordering (cpo): there is a least member, and every non-empty directed family has a supremum. It is a domain in the sense of domain theory. Suppose **A** is a set and **C** is a partial ordering; let $\mathbf{C}^{\mathbf{A}}$ be the family of all functions from **A** to **C**, with the pointwise ordering ($f \leq g$ if $f(x) \leq g(x)$ for every $x \in \mathbf{A}$). Then $\mathbf{C}^{\mathbf{A}}$ is a partial ordering, and $\mathbf{C}^{\mathbf{A}}$ is a cpo if **C** is. In particular, this is the case with $\mathit{Bool}^{\mathbf{A}}$. A function $T:\mathbf{C} \to \mathbf{C}$ is monotone provided $x \leq y$ implies $T(x) \leq T(y)$, for all $x,y \in \mathbf{C}$. Monotone functions on cpos always have smallest fixed points. Proofs of these and related results may be found in [6] and [4]. Every cpo has an associated topology, the Scott topology, chosen to mesh well with the algebraic structure. We won't need the general definition here. For *Bool*, the Scott topology has as open sets the upward closed ones: $\emptyset$, {*true*}, {*false*}, {*true, false*}, and *Bool*.

The truth functions we consider on *Bool* are those of Kleene's strong three-valued logic [7]. These have played roles in philosophy [8] and in logic programming [4], [5]. Informally, for $P \wedge Q$ to be true, both P and Q must be true, and for $P \wedge Q$ to be false, one of P or Q must be false; if P is true but the truth value of Q is unknown, the truth value of $P \wedge Q$ is unknown. Formally, the operations $\wedge$, $\vee$, $\neg$ and $\supset$ are defined by the following tables.

$\wedge$	*true*	*false*	$\perp$
true	*true*	*false*	$\perp$
false	*false*	*false*	*false*
$\perp$	$\perp$	*false*	$\perp$

$\vee$	*true*	*false*	$\perp$
true	*true*	*true*	*true*
false	*true*	*false*	$\perp$
$\perp$	*true*	$\perp$	$\perp$

$\supset$	*true*	*false*	$\perp$
true	*true*	*false*	$\perp$
false	*true*	*true*	*true*
$\perp$	*true*	$\perp$	$\perp$

	$\neg$
true	*false*
false	*true*
$\perp$	$\perp$

Note that $\wedge$ is not the infimum, nor is $\vee$ the supremum operation (though they are using the ordering *false* $<$ $\perp$ $<$ *true*). Each truth function is monotone. For example, if $p,q,r \in \mathit{Bool}$ and $p \leq q$, then $p \wedge r \leq q \wedge r$. In the Scott topology, each truth function is continuous in the appropriate sense. For example, $\wedge: \mathit{Bool}^2 \to \mathit{Bool}$ is continuous, where Bool^2 has the product topology.

Suppose **D** is some non-empty set, and $f:\mathbf{D}^{n+1} \to \mathit{Bool}$. For each $i = 1,2,\ldots, n+1$ we define an i^{th} existential quantifier $(\exists_i f)$ to be a function from $\mathbf{D}^n$ to *Bool*. For notational convenience we give the definition for $i = 1$; the general case is similar.

$$(\exists_1 f)(x_2,\ldots,x_{n+1}) = \begin{cases} \mathit{true} & \text{if } (\exists x_1 \in \mathbf{D})\ f(x_1,x_2,\ldots,x_{n+1}) = \mathit{true} \\ \mathit{false} & \text{if } (\forall x_1 \in \mathbf{D})\ f(x_1,x_2,\ldots,x_{n+1}) = \mathit{false} \\ \perp & \text{otherwise} \end{cases}$$

Existential quantifiers preserve monotonicity: if $f(\mathbf{x}) \leq g(\mathbf{x})$ for every $\mathbf{x} \in \mathbf{D}^{n+1}$, then $(\exists_i f)(\mathbf{x}) \leq (\exists_i g)(\mathbf{x})$ for every $\mathbf{x} \in \mathbf{D}^n$.

Lemma: Let **D** be a topological space and let P_i: $\mathbf{D}^{n+1} \to \mathbf{D}^n$ be defined by $P_i(\langle x_1,\ldots,x_{n+1}\rangle) = \langle x_1,\ldots,x_{i-1},x_{i+1},\ldots,x_{n+1}\rangle$. Then (using the product topologies) P_i is an open map, and also a closed map provided **D** is compact. (An open map takes open sets to open sets; similarly for closed map.)

Proof: It is straightforward that P_i is an open map, and we omit the proof. Now suppose **D** is compact; we show P_i is a closed map. Without loss of generality, take i to be 1; thus P_1: $\mathbf{D}\times\mathbf{D}^n \to \mathbf{D}^n$. Let **B** be a closed subset of $\mathbf{D}\times\mathbf{D}^n$; we show the complement of $P_1\mathbf{B}$ is open in $\mathbf{D}^n$. Let $c \in \mathbf{D}^n - P_1\mathbf{B}$. For each $y \in \mathbf{D}$, $\langle y,c\rangle \notin \mathbf{B}$ and since **B** is closed, there is a basic neighborhood $\mathbf{U}_y$ of $\langle y,c\rangle$

disjoint from $\mathbf{B}$. $\mathbf{D}$ is compact so a finite set $\mathbf{U}_{y_1},\ldots,\mathbf{U}_{y_n}$ of these neighborhoods covers $\mathbf{D}\times\{c\}$. P_1 is an open map so each $P_1\mathbf{U}_{y_n}$ is open in $\mathbf{D}^n$. Let $\mathbf{V} = \cap\{P_1\mathbf{U}_{y_1},\ldots,P_1\mathbf{U}_{y_n}\}$. This is a neighborhood of c. Finally, $\mathbf{D}\times\mathbf{V} \subseteq \cup\{\mathbf{U}_{y_1},\ldots,\mathbf{U}_{y_n}\}$ which is disjoint from $\mathbf{B}$, hence $\mathbf{V}$ is disjoint from $P_1\mathbf{B}$.

Proposition 1: Suppose $\mathbf{D}$ is a compact topological space and $f:\mathbf{D}^{n+1} \to \mathit{Bool}$ is continuous, where $\mathbf{D}^{n+1}$ has the product topology and *Bool* has the Scott topology. Then $(\exists_i f)$: $\mathbf{D}^n \to \mathit{Bool}$ is also continuous.

Proof: We show the inverse images of $\{\mathit{true}\}$ and $\{\mathit{false}\}$ under $(\exists_i f)$ are open in $\mathbf{D}^n$. Let P_i: $\mathbf{D}^{n+1} \to \mathbf{D}^n$ be as above. Then $(\exists_i f)^{-1}(\{\mathit{true}\}) = P_i f^{-1}(\{\mathit{true}\})$ and this is open, using the continuity of f and the fact that P_i is an open map. Also $(\exists_i f)^{-1}(\{\mathit{true}, \perp\}) = P_i f^{-1}(\{\mathit{true}, \perp\})$. Since $\{\mathit{true}, \perp\}$ is closed in *Bool*, this set is closed in $\mathbf{D}^n$, using the continuity of f and the fact that P_i is a closed map. Then the complement in $\mathbf{D}^n$ is open, but this is $(\exists_i f)^{-1}(\{\mathit{false}\})$.

S3 Logic Programming, syntax

By a <u>data structure</u> we mean $\langle \mathbf{D}; \mathbf{R}_1,\ldots, \mathbf{R}_k\rangle$ where $\mathbf{D}$ is a non-empty set, called the <u>domain</u>, and $\mathbf{R}_1,\ldots, \mathbf{R}_k$ are partial relations on $\mathbf{D}$, called <u>given relations</u> (functions on $\mathbf{D}$ are identified with their graphs). By <u>partial relations</u> we informally mean relations whose truth value is allowed to be undefined in some cases. Formally, an n-ary partial relation on $\mathbf{D}$ is a mapping from $\mathbf{D}^n$ to *Bool*. The examples in this section have given partial relations that never take on the value $\perp$, but relations that do will be used in S5.

Examples:
1) Domain: L^*, all finite words over the finite alphabet L. Given relation: concatenation.
2) Domain: finite decimals. Given relations: addition, multiplication.
3) Domain: all closed terms built up from a finite list of constant and function symbols. Given relations: equality and, for each n-place function symbol f, the n+1 place relation $\mathbf{R}_f$ such that $\mathbf{R}_f(t_1,\ldots,t_n,t_{n+1})$ just when $f(t_1,\ldots,t_n) = t_{n+1}$.

In example 1 equality can be defined from concatenation and the empty word. Similarly for example 2, using addition and 0. Example 3 is the usual setting for logic programming theory.

Fix a data structure $\langle \mathbf{D}; \mathbf{R}_1,\ldots, \mathbf{R}_k\rangle$ for the time being. We describe a logic programming language for it. As usual, we suppose available an unlimited supply of <u>variables</u>, and an unlimited supply of 1-place, 2-place,... <u>relation symbols</u>. Once and for all, certain relation symbols, $R_1,\ldots,R_k$ are set aside to represent the given relations of $\langle \mathbf{D}; \mathbf{R}_1,\ldots, \mathbf{R}_k\rangle$. We call these <u>reserved</u>. They must be distinct, and R_i must be an n-place relation symbol if $\mathbf{R}_i$ is an n-place relation. <u>Atomic formulas</u> are expressions of the form $R(u_1,\ldots,u_n)$ where R is a relation symbol and each u_i is a variable or a member of $\mathbf{D}$. We also allow *true* and *false* as atomic formulas. <u>Formulas</u> are built up using the connectives $\wedge, \vee, \neg, \supset$ and the existential quantifier $\exists$. <u>Free variable occurrences</u> are defined in the usual way. We write $\varphi(x_1,\ldots,x_n)$ to indicate that φ is a formula with all its free variables among $x_1,\ldots,x_n$. Then by $\varphi(t_1,\ldots,t_n)$ we mean the result of substituting occurrences of t_i for all free occurrences of x_i in φ $(i = 1,2,\ldots,n)$. Let R be an n-place relation symbol: a <u>definition</u> for R

is an expression of the form $R(x_1,...,x_n) \leftarrow \varphi(x_1,...,x_n)$, where φ is some formula, called the body of the definition. A program is a finite set of definitions such that: 1) no definition is for a reserved relation symbol; 2) no relation symbol has two definitions; and 3) every non-reserved relation symbol occurring in any definition body itself has a definition.

S4 Logic Programming, semantics

Let $\langle \mathbf{D}; \mathbf{R}_1,..., \mathbf{R}_k \rangle$ be a data structure, fixed for this section. Then we have a logic programming language for it as described in the previous section (with $R_1,..., R_k$ as the reserved relation symbols associated with the given partial relations $\mathbf{R}_1,..., \mathbf{R}_k$). Let $\mathbf{P}$ be a program, also fixed for this section. We sketch a semantics for $\mathbf{P}$.

Let $F_1, F_2,..., F_t$ be the relation symbols of $\mathbf{P}$ (including all reserved ones), where F_i is n(i) place. By interpretation space we mean $\mathit{Int} = \mathbf{D}^{n(1)} \oplus \mathbf{D}^{n(2)} \oplus ... \oplus \mathbf{D}^{n(t)}$ where $\mathbf{D}^{n(i)}$ is $\mathbf{D} \times \mathbf{D} \times ... \times \mathbf{D}$ with n(i) components, and $\oplus$ represents disjoint union, the union of disjoint copies. Thus *Int* has one component for each relation symbol $F_1, F_2,..., F_t$. In a mild abuse of notation we will act as if $\mathbf{D}^{n(i)}$ and the corresponding component of *Int* are the same, though technically the component of *Int* is an isomorphic copy of $\mathbf{D}^{n(i)}$, chosen so that no two components of *Int* overlap. If $\langle d_1,...,d_{n(i)} \rangle \in \mathbf{D}^{n(i)}$ we will say $\langle d_1,...,d_{n(i)} \rangle$ is in the F_i component of *Int,* and assume no misunderstanding will result from this. An interpretation is simply a mapping $\mathbf{I}: \mathit{Int} \rightarrow \mathit{Bool}$. An interpretation $\mathbf{I}$ is in the data structure $\langle \mathbf{D}; \mathbf{R}_1,..., \mathbf{R}_k \rangle$ provided, for each given relation $\mathbf{R}_i$, the behavior of $\mathbf{I}$ on the R_i component of *Int* agrees with the map $\mathbf{R}_i$. The family of all interpretations in $\langle \mathbf{D}; \mathbf{R}_1,..., \mathbf{R}_k \rangle$ with the pointwise ordering is a cpo because *Bool* is. Let $\varphi(x_1,...,x_n)$ be a constant-free formula built up from the relation symbols $F_1, F_2,..., F_t$, with free variables $x_1,...,x_n$, and let $\mathbf{I}$ be an interpretation in $\langle \mathbf{D}; \mathbf{R}_1,..., \mathbf{R}_k \rangle$; we define a function $\varphi^{\mathbf{I}}: \mathbf{D}^n \rightarrow \mathit{Bool}$ inductively as follows. 1) φ is atomic, say $\varphi(x_1,...,x_n)$ is $F(x_1,...,x_n)$ where F is an n place relation symbol. Then, for $d_1,...,d_n \in \mathbf{D}$, $\varphi^{\mathbf{I}}(d_1,...,d_n)$ is $\mathbf{I}(\langle d_1,...,d_n \rangle)$, where $\langle d_1,...,d_n \rangle$ is taken from the F component of *Int.* 2) φ is $\alpha \wedge \beta$. Then $\varphi^{\mathbf{I}} = \wedge(\alpha^{\mathbf{I}}, \beta^{\mathbf{I}})$, where $\wedge$ is the function on *Bool* given in S2. Similarly for $\vee$, $\neg$ and $\supset$. 3) φ is $(\exists x_i)\alpha$. Then $\varphi^{\mathbf{I}} = (\exists_i \alpha^{\mathbf{I}})$, where $\exists_i$ was also defined in S2. Since the propositional connectives are monotone on *Bool,* and existential quantifiers preserve monotonicity, it follows that $\mathbf{I} \leq \mathbf{J}$ implies $\varphi^{\mathbf{I}} \leq \varphi^{\mathbf{J}}$.

Now, we associate with program $\mathbf{P}$ an operator $T_{\mathbf{P}}$ on interpretations in $\langle \mathbf{D}; \mathbf{R}_1,..., \mathbf{R}_k \rangle$. Let $\mathbf{I}$ be an interpretation in $\langle \mathbf{D}; \mathbf{R}_1,..., \mathbf{R}_k \rangle$; $T_{\mathbf{P}}(\mathbf{I})$ is the interpretation $\mathbf{J}$ characterized as follows. For $\langle d_1,...,d_{n(i)} \rangle$ in the F_i component of *Int,* 1) if F_i is a reserved relation symbol, say R_k, then $\mathbf{J}(\langle d_1,...,d_{n(i)} \rangle)$ is $R_k^{\mathbf{I}}(d_1,...,d_{n(i)})$, that is, $\mathbf{R}_k(d_1,...,d_{n(i)})$. 2) Otherwise, F_i is a defined relation symbol, say its definition in $\mathbf{P}$ is $F_i(x_1,...,x_{n(i)}) \leftarrow \varphi(x_1,...,x_{n(i)})$. Then $\mathbf{J}(\langle d_1,...,d_{n(i)} \rangle)$ is $\varphi^{\mathbf{I}}(d_1,...,d_{n(i)})$. $T_{\mathbf{P}}$ maps interpretations in $\langle \mathbf{D}; \mathbf{R}_1,..., \mathbf{R}_k \rangle$ to interpretations in $\langle \mathbf{D}; \mathbf{R}_1,..., \mathbf{R}_k \rangle$, and is easily seen to be monotone. Then $T_{\mathbf{P}}$ has a smallest fixed point $\mathbf{I}$ in the space of interpretations in $\langle \mathbf{D}; \mathbf{R}_1,..., \mathbf{R}_k \rangle$. We take this smallest fixed point as the semantical meaning of program $\mathbf{P}$.

There is a direct connection between this approach and the more familiar minimal model semantics as found in [12], [2] and [10]. Suppose we use the data structure given as example 3 in S3,

which can be identified with the customary logic programming setting. Let **P** be a conventional Horn clause program (in particular, no negations are used). **P** can be converted to a program in our sense in standard ways; call the resulting program **Q**. Then, for each relation symbol R and closed terms $t_1,...,t_n$, $R(t_1,...,t_n)$ is assigned the value *true* in the conventional minimal model semantics if and only if $R^I(t_1,...,t_n)$ = *true* where **I** is the smallest fixed point of the operator T_Q defined as above. This extends to relate the failure (not finite failure) set of the conventional semantics for **P** with the set mapping to *false* using the smallest fixed point of T_Q. A proof of this can be found in [4], after noting that the semantics presented here is equivalent to the semantics of that paper.

S5 Introducing compact data structures

For each data structure $\langle \mathbf{D}; \mathbf{R}_1,..., \mathbf{R}_k \rangle$ and each program **P** a mapping T_P on interpretations has been defined. Let I^P_0 be the smallest interpretation in $\langle \mathbf{D}; \mathbf{R}_1,..., \mathbf{R}_k \rangle$, behaving like the given relations of $\langle \mathbf{D}; \mathbf{R}_1,..., \mathbf{R}_k \rangle$ on the components of *Int* corresponding to the reserved relation symbols, identically $\perp$ on all other components. Let $I^P_{n+1} = T_P(I^P_n)$. Then $I^P_0 \leq I^P_1 \leq I^P_2 \leq ...$ Let I^P_ω be the pointwise limit of this sequence. One would like I^P_ω to be the least fixed point of T_P, but this is not always the case. The problem is not due to our use of arbitrary data structures, or of *Bool* instead of the conventional two valued setting. In [2] an example is given which shows their T operator need not reach a fixed point when "coming down from above" in ω steps. By work in [4], this provides an example of a program **P** for which I^P_ω is not the least fixed point of T_P. [3] makes it clear that we may have to go very much further than I^P_ω to reach a fixed point.

We call a data structure $\langle \mathbf{D}; \mathbf{R}_1,..., \mathbf{R}_k \rangle$ <u>compact</u> if: 1) **D** has a compact topology, and 2) each given (n-place) relation $\mathbf{R}_i$, thought of as a mapping from $\mathbf{D}^n$ with the product topology, to *Bool* with the Scott topology, is a continuous function.

In the next section we show that if we have a compact data structure, then the least fixed point of T_P will be I^P_ω. We conclude this section with some examples of compact data structures.

Example 4: Domain **D**: all words, finite or infinite, over the finite alphabet L. A distance between words is defined as follows. If w_1 and w_2 are identical, the distance between them is 0. Otherwise, the distance between them is $1/2^n$ where w_1 and w_2 first differ at position n. (For this purpose words are taken as having the same letter at position i if both words are shorter in length than i, and as having different letters at position i if one word is shorter than i but the other is not.) This distance provides us with a compact metric space, but given relations are a little subtle. The equality relation, thought of as the function f on $\mathbf{D}^2$ such that f(x,y) = *true* if x = y and f(x,y) = *false* if x ≠ y, is <u>not</u> continuous. Nor is it desirable computationally. If x and y are different words, that fact can be discovered. Starting at the left ends and working through a letter at a time, we must come on a position where x and y differ. Likewise, if x and y are identical and finite, their identity can be determined since we can examine each position in each word. But if x and y are identical and infinite, an inspection procedure will never terminate. This suggests we use the following.

$$equal(x,y) = \begin{cases} true & \text{if } x = y \text{ and } x \text{ and } y \text{ are finite words} \\ false & \text{if } x \neq y \\ \perp & \text{otherwise} \end{cases}$$

Then *equal*:$\mathbf{D}^2 \to$ *Bool* is a continuous function. Similar reasoning applies to a candidate for the concatenation relation.

Example 5: Domain **D**: the real interval [0,1] with standard distance function. Equality is as inappropriate here as it was in example 4. A "nearness" relation is reasonable, however.

$$\mathit{within}(x,y,z) = \begin{cases} \mathit{true} & \text{if } |x-y| < z \\ \mathit{false} & \text{if } |x-y| > z \\ \perp & \text{otherwise} \end{cases}$$

This is a continuous function from $\mathbf{D}^3$ to *Bool*, as desired. Likewise, instead of "ordinary" addition, we can use an "approximate" addition relation: x + y is within z of w, correcting appropriately for the possibility of going out of the range [0,1].

Example 6: The space of finite and infinite terms made up from a finite list of constant and function symbols, as presented in Chapter 4 of [10] is easily made into an example in our sense.

§6 Results about compact data structures

If **D** is a compact topological space, the product space $\mathbf{D}^n$, using the standard product topology, is compact by Tychonoff's Theorem. If $\mathbf{D}_1,\ldots,\mathbf{D}_n$ are topological spaces, we give the disjoint union $\mathbf{D}_1 \oplus \ldots \oplus \mathbf{D}_n$ a topology with basic open sets (copies of) the open sets of $\mathbf{D}_1, \ldots, \mathbf{D}_n$. In this topology a function on $\mathbf{D}_1 \oplus \ldots \oplus \mathbf{D}_n$ will be continuous just when it is continuous on each component separately. And finally, $\mathbf{D}_1 \oplus \ldots \oplus \mathbf{D}_n$ is compact if and only if each of $\mathbf{D}_1, \ldots, \mathbf{D}_n$ is compact. For the rest of this section, $\langle \mathbf{D}; \mathbf{R}_1,\ldots, \mathbf{R}_k \rangle$ is a compact data structure and **P** is a program in its logic programming language. We assume the relation symbols of **P** are $F_1, F_2,\ldots, F_t$. Thent *Int* = $\mathbf{D}^{n(1)} \oplus \mathbf{D}^{n(2)} \oplus \ldots \oplus \mathbf{D}^{n(t)}$ is also a compact space.

Proposition 2: Suppose the interpretation **I** : *Int* $\to$ *Bool* is a continuous function. Then so is $T_{\mathbf{P}}(\mathbf{I})$: *Int* $\to$ *Bool*.

Proof: $T_{\mathbf{P}}(\mathbf{I})$ will be continuous on *Int* = $\mathbf{D}^{n(1)} \oplus \mathbf{D}^{n(2)} \oplus \ldots \oplus \mathbf{D}^{n(t)}$ if it is continuous on each component. A component corresponding to a reserved relation symbol gives no trouble since the given relations are required to be continuous. For a non-reserved relation symbol F_i, say the corresponding definition in program **P** is $F_i(x_1,\ldots,x_{n(i)}) \leftarrow \varphi(x_1,\ldots,x_{n(i)})$. The behavior of $T_{\mathbf{P}}(\mathbf{I})$ on the F_i component is defined to be that of $\varphi^{\mathbf{I}}(x_1,\ldots,x_{n(i)})$. Continuity follows easily from the continuity of **I**, the continuity of the propositional connectives and Proposition 1.

As in the previous section, we can define an "approximation sequence" of interpretations $\mathbf{I}^{\mathbf{P}}{}_0$, $\mathbf{I}^{\mathbf{P}}{}_1$, $\mathbf{I}^{\mathbf{P}}{}_2,\ldots$. $\mathbf{I}^{\mathbf{P}}{}_0$ is continuous because it agrees with the continuous given relations of $\langle \mathbf{D}; \mathbf{R}_1,\ldots, \mathbf{R}_k \rangle$ on components of *Int* corresponding to reserved relation symbols, and is identically $\perp$ on all other components of *Int* (obviously a continuous function on each component). Then each $\mathbf{I}^{\mathbf{P}}{}_n$ must be continuous by Proposition 2. The following extends this to $\mathbf{I}^{\mathbf{P}}{}_\omega$.

Proposition 3: Let $\mathbf{I}_0 \leq \mathbf{I}_1 \leq \mathbf{I}_2 \leq \ldots$ be a sequence of interpretations, monotone under the pointwise ordering. And let $\mathbf{I} = \sup_n \mathbf{I}_n$ be the (pointwise) limit. If each $\mathbf{I}_n$ is continuous, so is $\mathbf{I}$.

Proof: The open sets in *Bool* are upward closed in that, if O is open, $a \in O$, and $a \leq b$, then $b \in O$. Then, for an open set O in *Bool*, if $\mathbf{I}_n(x) \in O$ and $n < k$, then $\mathbf{I}_k(x) \in O$ because of the pointwise ordering used on interpretations. It follows that for any open set O in *Bool*, $\mathbf{I}^{-1}(O) = \cup_n \mathbf{I}_n^{-1}(O)$ and this is open because each $\mathbf{I}_n$ is continuous and the open sets are closed under union.

Lemma: Let $\mathbf{I}_0 \leq \mathbf{I}_1 \leq \mathbf{I}_2 \leq \ldots$ be a monotone sequence of continuous interpretations with $\mathbf{I}$ as the pointwise limit. Also let $\varphi(x_1,\ldots,x_n)$ be a formula with no constants (members of **D**), built up from the relation symbols $F_1,\ldots, F_t$ of **P**. Then, for each $d_1,\ldots, d_n \in \mathbf{D}$, there is some m such that $\varphi^{\mathbf{I}_m}(d_1,\ldots, d_n) = \varphi^{\mathbf{I}}(d_1,\ldots, d_n)$.

Proof: By induction on the degree of φ. If φ is atomic the result is immediate from the definition of $\mathbf{I}$. The propositional connective cases are straightforward. For the quantifier case, $\varphi(x_1,\ldots,x_n)$ is $(\exists y)\theta(y,x_1,\ldots,x_n)$, the subcases where $\varphi^{\mathbf{I}}(d_1,\ldots, d_n)$ is *true* or $\perp$ are unproblematic. Suppose now that the result is known for θ, $\varphi(x_1,\ldots,x_n)$ is $(\exists y)\theta(y,x_1,\ldots,x_n)$, and $\varphi^{\mathbf{I}}(d_1,\ldots, d_n) =$ *false*. Each $\mathbf{I}_k$ is continuous and it follows that $\theta^{\mathbf{I}_m}: \mathbf{D}^{n+1} \to$ *Bool* is a continuous function. For notational convenience, let $H_k(y) = \theta^{\mathbf{I}_m}(y,d_1,\ldots,d_n)$. Then each $H_k : \mathbf{D} \to$ *Bool* is also continuous. Since $\varphi^{\mathbf{I}}(d_1,\ldots, d_n) =$ *false*, for each $a \in \mathbf{D}$, $\theta^{\mathbf{I}}(a,d_1,\ldots, d_n) =$ *false*. By the induction hypothesis, for each $a \in \mathbf{D}$ there is some k such that $H_k(a) =$ *false*. Since $\{$*false*$\}$ is open in *Bool*, $H_k^{-1}(\{$*false*$\})$ is open in **D**. Then $\{ H_0^{-1}(\{$*false*$\}), H_1^{-1}(\{$*false*$\}), \ldots \}$ is an open cover of **D**. Compactness yields a finite subcover, $\{ H_a^{-1}(\{$*false*$\}),\ldots, H_m^{-1}(\{$*false*$\}) \}$ where m is the biggest index appearing. It follows from the pointwise ordering of the sequence $\mathbf{I}_0, \mathbf{I}_1, \mathbf{I}_2,\ldots$ that $\theta^{\mathbf{I}_m}(a,d_1,\ldots,d_n) =$ *false* for every $a \in \mathbf{D}$, and hence $\varphi^{\mathbf{I}_m}(d_1,\ldots,d_n) =$ *false*.

Proposition 4: For the compact data structure $\langle \mathbf{D}; \mathbf{R}_1,\ldots, \mathbf{R}_k \rangle$ and program **P**, the associated mapping $T_{\mathbf{P}}$ has $\mathbf{I}^{\mathbf{P}}{}_{\omega}$ as its least fixed point.

Proof: Let F be the least fixed point of $T_{\mathbf{P}}$. Trivially, $\mathbf{I}^{\mathbf{P}}{}_0 \leq F$. If $\mathbf{I}^{\mathbf{P}}{}_n \leq F$, using monotonicity, $\mathbf{I}^{\mathbf{P}}{}_{n+1} = T_{\mathbf{P}}(\mathbf{I}^{\mathbf{P}}{}_n) \leq T_{\mathbf{P}}(F) = F$. It follows that $\mathbf{I}^{\mathbf{P}}{}_{\omega} \leq F$. Consequently $\mathbf{I}^{\mathbf{P}}{}_{\omega}$ will be <u>least</u> if it is a fixed point at all. For each n, $\mathbf{I}^{\mathbf{P}}{}_n \leq \mathbf{I}^{\mathbf{P}}{}_{\omega}$, hence $\mathbf{I}^{\mathbf{P}}{}_{n+1} = T_{\mathbf{P}}(\mathbf{I}^{\mathbf{P}}{}_n) \leq T_{\mathbf{P}}(\mathbf{I}^{\mathbf{P}}{}_{\omega})$. It follows that $\mathbf{I}^{\mathbf{P}}{}_{\omega} \leq T_{\mathbf{P}}(\mathbf{I}^{\mathbf{P}}{}_{\omega})$. Finally to show $T_{\mathbf{P}}(\mathbf{I}^{\mathbf{P}}{}_{\omega}) \leq \mathbf{I}^{\mathbf{P}}{}_{\omega}$ it suffices to show that for each atomic formula $F_i(d_1,\ldots,d_{n(i)})$ without variables, its truth value under $T_{\mathbf{P}}(\mathbf{I}^{\mathbf{P}}{}_{\omega})$ is $\leq$ its truth value under $\mathbf{I}^{\mathbf{P}}{}_{\omega}$. If F_i is reserved, the result is immediate. If F_i is not reserved, there is a definition for it in **P**, say $F_i(x_1,\ldots,x_{n(i)}) \leftarrow \varphi(x_1,\ldots,x_{n(i)})$. The truth value of $F_i(d_1,\ldots,d_{n(i)})$ under $T_{\mathbf{P}}(\mathbf{I}^{\mathbf{P}}{}_{\omega})$ is that of $\varphi(d_1,\ldots,d_{n(i)})$ under $\mathbf{I}^{\mathbf{P}}{}_{\omega}$. By the Lemma, for some j, $\varphi(d_1,\ldots,d_{n(i)})$ has the same truth value

under I^P_ω and I^P_j. Then $F_i(d_1,\ldots,d_{n(i)})$ has the same truth value under $T_P(I^P_\omega)$ and $T_P(I^P_j) = I^P_{j+1}$. But $I^P_{j+1} \leq I^P_\omega$, which concludes the proof.

Finally, compact data structures of interest tend to have a domain with a dense subset of "concrete" objects. The reals have the rationals; the family of all words has the family of finite words; the family of all terms has the family of finite terms. We conclude with a result relating the semantics for a compact data structure with that for a dense substructure. We use the following notation. As usual in this section, $\langle D; R_1,\ldots, R_k\rangle$ is a compact data structure, P is a program relative to this data structure, and T_P is the corresponding mapping on the interpretation space *Int*. Now D^0 is a dense subset of D, and $R_1{}^0,\ldots, R_k{}^0$ are the relations $R_1,\ldots, R_k$ restricted to D^0. *Int*^{0} is the interpretation space, like *Int*, except that each component is a power of D^0 rather than of D. Finally, τ_P is the mapping on *Int*^{0} associated with program P, using the data structure $\langle D^0; R_1{}^0,\ldots, R_k{}^0\rangle$, just as T_P is associated with P using $\langle D; R_1,\ldots, R_k\rangle$. All the results about monotonicity and fixed points from S4 apply to τ_P, since no special topological assumptions were made in S4. Finally, we defined a sequence of interpretations above, mapping *Int* to *Bool*: I^P_0, $I^P_1,\ldots$ and a limit I^P_ω, using the map T_P. In exactly the same way we have a sequence of interpretations, mapping *Int*^{0} to *Bool*: J^P_0, $J^P_1,\ldots$ and a limit J^P_ω, using the map τ_P. Since $\langle D^0; R_1{}^0,\ldots, R_k{}^0\rangle$ need not be compact, J^P_ω may not be the least fixed point of τ_P, but it must be $\leq$ the least fixed point.

Lemma: Suppose $\varphi(x_1,\ldots, x_n)$ is a formula with no constants, and I is an interpretation mapping *Int* to *Bool*, whose interpretation space has a component for every relation symbol of φ. Let J be I restricted to *Int*^{0}. Then, for $d_1,\ldots, d_n \in D^0$, $\varphi^I(d_1,\ldots, d_n) \leq \varphi^J(d_1,\ldots, d_n)$, where φ^I treats existential quantifiers as quantifying over D, while φ^J treats them as quantifying over D^0.

Proof: By induction on the degree of φ. If φ is atomic, we have $\varphi^I(d_1,\ldots, d_n) = \varphi^J(d_1,\ldots, d_n)$. The propositional cases are straightforward. Suppose $\varphi(x_1,\ldots,x_n)$ is $(\exists y)\Theta(y,x_1,\ldots,x_n)$, and the result is known for Θ. If $\varphi^I(d_1,\ldots, d_n) = \perp$, the conclusion is immediate. If $\varphi^I(d_1,\ldots, d_n) =$ *false* then for each $a \in D$, $\Theta^I(a, d_1,\ldots, d_n) =$ *false*. In particular, this happens for all $a \in D^0$. By the induction hypothesis, $\Theta^J(a, d_1,\ldots, d_n) =$ *false* for all $a \in D^0$, and so $\varphi^J(d_1,\ldots, d_n) =$ *false*. Finally, suppose $\varphi^I(d_1,\ldots, d_n) =$ *true*. Then for some $a \in D$, $\Theta^I(a, d_1,\ldots, d_n) =$ *true*. Since Θ^I is continuous and $\{$*true*$\}$ is open, there is an open set O in D with $a \in O$ such that for all $y \in O$, $\Theta^I(y, d_1,\ldots, d_n) =$ *true*. Since D^0 is dense in D, there is some $b \in D^0 \cap O$. Then $\Theta^I(b, d_1,\ldots, d_n) =$ *true*, and by the induction hypothesis, $\Theta^J(b, d_1,\ldots, d_n) =$ *true*, so $\varphi^J(d_1,\ldots, d_n) =$ *true*.

Our final result says that by working with a dense subset of D we loose no positive information, though some atomic formulas will be assigned *true* or *false* which should have been $\perp$.

Proposition 5: Let I^P_ω be the least fixed point of T_P. Then I^P_ω, restricted to *Int*^{0}, is $\leq$ the least fixed point of τ_P.

Proof: I^P_0, restricted to Int^o, $\leq J^P_0$. In fact, we have equality. Suppose I^P_n, restricted to Int^o, $\leq J^P_n$. Let $F_i(x_1,\ldots, x_{n(i)})$ be atomic, and $d_1,\ldots, d_{n(i)} \in D^o$. We show the truth value of $F_i(d_1,\ldots, d_{n(i)})$ under $I^P_{n+1} \leq$ the truth value of $F_i(d_1,\ldots, d_{n(i)})$ under J^P_{n+1}, which is enough to establish that I^P_{n+1}, restricted to Int^o, $\leq J^P_{n+1}$. If F_i is reserved, the result is immediate. Otherwise, say $F_i(x_1,\ldots, x_{n(i)}) \leftarrow \varphi(x_1,\ldots, x_{n(i)})$ is a definition in P. The truth value of $F_i(d_1,\ldots, d_{n(i)})$ under I^P_{n+1} is that of $\varphi(d_1,\ldots, d_{n(i)})$ under I^P_n. Using the lemma, the induction hypothesis, and the monotonicity of the connectives and quantifiers, this is $\leq$ the truth value of $\varphi(d_1,\ldots, d_{n(i)})$ under J^P_n, which in turn is the truth value of $F_i(d_1,\ldots, d_{n(i)})$ under J^P_{n+1}. Then, by induction, I^P_n, restricted to $Int^o \leq J^P_n$ for every integer n, and the theorem follows directly.

Bibliography

[1] Infinite-Term Semantics for Logic Programs, H. Andreka, M.H. van Emden, I. Nemeti and J. Tiuryn, manuscript, 1983.

[2] Contributions to the theory of logic programming, K. R. Apt, M. H. van Emden, J. Assoc. Comput. Mach., vol 29, pp. 841-862, 1982.

[3] The recursion-theoretic complexity of the semantics of predicate logic as a programming language, H. Blair, Information and Control, vol 54, pp. 25-47, 1982.

[4] A Kripke-Kleene Semantics for Logic Programs, M. Fitting, The Journal of Logic Programming, vol. 4, pp. 295-312, 1985.

[5] Partial models and logic programs, M. Fitting, manuscript, 1986.

[6] On the Existence of Optimal Fixpoints, J. H. Gallier, Math. Systems Theory, vol. 13, pp. 209-217, 1980.

[7] Introduction To Metamathematics, S. C. Kleene, Van Nostrand, New York, 1952.

[8] Outline of a Theory of Truth, S. Kripke, Journal of Philosophy, vol. 72, pp. 690-716, 1975.

[9] Optimal fixedpoints of logic programs, J. L. Lassez and M. Maher, Theoretical Computer Science, vol 39, 1985.

[10] Foundations of Logic Programming, J. W. Lloyd, Springer-Verlag, Berlin, 1984.

[11] Logic Programs and Many-Valued Logic, A. Mycroft, in: M. Fontet and K. Mehlhorn (eds.), STACS 84, Symposium of Theoretical Aspects of Computer Science, Proceedings, Springer Lecture Notes in Computer Science, 166, pp. 274-286, 1984.

[12] The semantics of predicate logic as a programming language, M. van Emden, R. Kowalski, J. Assoc. Comput. Mach., vol. 23, pp. 733-742, 1976.