ENUMERATION OPERATORS AND MODULAR LOGIC PROGRAMMING

MELVIN FITTING*

▷ An operational and a minimal model semantics for logic programming modules is introduced. It is shown that this semantics corresponds to the recursion theoretic notion of enumeration operator. Basic operations on modules, such as composition and recursion, are discussed. The adequacy of these operations is established by showing that all logic programming can be done, in principle, by combining certain elementary modules using these basic operations. ◁

1. INTRODUCTION

One problem with conventional logic programming is its lack of modularity. Programs conceptually are "flat" objects; there is no notion of subprogram. Consequently to construct a big program out of prewritten, pretested pieces, the programmer must impose structure that is not naturally inherent in the language itself. One finds oneself in the position of the user of a simple dialect of BASIC, imposing a PASCAL-like structure from the outside.

O'Keefe [5] recognizes the problem and deals with it head on. A natural modular structure is proposed, and the beginnings of a semantics for it are sketched. Also, Lassez and Maher [4] propose the separation of a logic program into rules and facts. One thinks of the rules as defining an operator with the facts as input. Though their motivation is somewhat different, the result is closely related to O'Keefe's.

What seems to have gone unremarked in the literature is that the mathematical basis for a semantics of modular logic programming already exists. It is well known that the relations definable using Horn clause programs are the recursively enumer-

Address correspondence to Melvin Fitting, Department of Mathematics and Computer Science, Herbert H. Lehman College, Bedford Park Boulevard West, Bronx, New York 10468.

* Partial support received from NSF grant DCR 8504825.

able relations. This generalizes: the operators (modules) definable using Horn clause programs are the *enumeration operators*. These operators have been studied for some 20 years by recursion theorists.

Since many of the results stated here have already appeared in print, we generally omit proofs. Primarily, what we do is collect things together and present them with logic programming applications in mind.

2. ENUMERATION OPERATORS—ORIGINAL DEFINITION

Recursion theorists tend to work with numbers; logic programmers tend to work with formal terms built up from constant symbols using function symbols. This difference is a minor annoyance, but nothing serious. Everything we say applies equally well to either domain, via familiar coding tricks, or because of things mentioned in Section 3 concerning pairing functions. For uniformity and simplicity, we will take numbers as basic, and when we are discussing logic programming we will assume the arithmetic operator SUCCESSOR is available (as a relation) and can be used in axiom bodies, but not heads. Other arithmetic operations like addition and multiplication are then easily programmable.

Definition. ω is the set $\{0,1,2,\dots\}$. An *operator* is a mapping from subsets of ω to subsets of ω. An operator Φ is *monotone* if $P \subseteq Q \subseteq \omega \Rightarrow \Phi(P) \subseteq \Phi(Q)$. An operator Φ is *compact* if $n \in \Phi(P) \Rightarrow n \in \Phi(F)$ for some finite $F \subseteq P$.

Continuity is usually discussed, rather than compactness, but they are equivalent in the present of monotonicity. For our purposes, compactness is the more useful version.

A compact, monotone operator is completely determined by what it does on finite sets. This means a compact, monotone operator can be specified by giving a relation R between finite sets and numbers, where $R(F, n)$ is to mean that n is in the operator output provided F is (part of) the input. Using standard devices from elementary recursion theory, such a relation can be coded as a set of numbers.

Definition. $J(x, y) = [(x+y)^2 + 3x + y]/2$ is the standard recursive *pairing function*, mapping $\omega \times \omega$ in a 1-1, onto fashion to ω. D is the recursive *finite coding function*, mapping the collection of finite subsets of ω in a 1-1, onto fashion to ω as follows. For a finite set F of numbers, $D(F)$ is the number whose binary representation has 1 in the 2^n place if $n \in F$, and has 0 in the 2^n place otherwise.

Definition. Let R be a set of numbers. Associate with R an operator Φ_R as follows. For $P \subseteq \omega$,

$$\Phi_R(P) = \{n \mid \text{for some finite } F \subseteq P,\ J(D(F), n) \in R\}.$$

Φ_R is always monotone and compact, and every monotone and compact operator is of this form. Now the original definition of enumeration operator, from [7], can be given easily.

Definition. An operator is an *enumeration operator* if it is Φ_R for some recursively enumerable set R.

The family of monotone, compact operators provides a model for the lambda calculus [6]. For two sets P and Q, the *application* of P to Q is taken to be $\Phi_p(Q)$. Likewise, using enumeration operators and restricting input sets to be recursively enumerable provides another model. Details are sketched in [8], but would take us too far afield here.

3. PROGRAMS—OPERATIONAL SEMANTICS

Enumeration operators can be characterized naturally using logic programs and their operational semantics, as defined in [9]. We do so in this section.

Definition. A *program* is a finite set of Horn clauses. An *extended program* is a set of Horn clauses that need not be finite.

Definition. For an extended program E, the *derivation set* of E is the smallest set E^* (of variable free formulas) containing all substitution instances (over ω) of members of E, containing all statements $\text{SUCCESSOR}(a, b)$ where $b = a + 1$, and closed under the following rule: $A \leftarrow B_1, \ldots, B_n \in E^*$ and $B_1, \ldots, B_n \in E^*$ imply $A \in E^*$. A variable free formula is said to be *derivable from* E if it is a member of E^*.

This is the basis for the usual operational semantics for Horn clause programs. Adding machinery for inputs is a simple matter.

Definition. Suppose $\mathbf{P}$ is a program in which the one place predicate symbol $\mathbf{I}$ (intended to represent input) does not occur in the head of any axiom. Also, let $\mathbf{O}$ (intended to represent output) be another one place predicate symbol. We use $[\mathbf{P}^{\mathbf{I}}_{\mathbf{O}}]$ to denote the operator whose (operational) behavior is specified as follows. For $S \subseteq \omega$, $[\mathbf{P}^{\mathbf{I}}_{\mathbf{O}}](S)$ is the set of numbers n such that $\mathbf{O}(n)$ is derivable from the extended program $\mathbf{P} \cup \{\mathbf{I}(k) | k \in S\}$.

In short, add to $\mathbf{P}$ axioms saying members of S are inputs, and see what outputs are derivable. The notion of operator above differs slightly from that considered by Lassez and Maher [4] in that we allow axioms with empty bodies to be members of $\mathbf{P}$, while they do not. The restriction that $\mathbf{I}$ may not appear in axiom heads says intuitively that input is not computed: it must be supplied from the outside. The restriction plays a technical role in Section 6.

It is immediate from this definition that $[\mathbf{P}^{\mathbf{I}}_{\mathbf{O}}]$ is a monotone operator, because increasing a set of axioms does not decrease the corresponding derivation set. Also $[\mathbf{P}^{\mathbf{I}}_{\mathbf{O}}]$ is compact, essentially because derivations are finite.

Proposition 3.1. The enumeration operators in Rogers's sense, Φ_R for a recursively enumerable relation R, and the operators defined above, $[\mathbf{P}^{\mathbf{I}}_{\mathbf{O}}]$ for a program $\mathbf{P}$, are the same class of operators.

This is proved in [2] in a more general setting (Theorem 8.7.3). In that work the characterization of the present section was used as the definition of enumeration operator. We remark that an analogous result holds for a wide variety of domains, not just ω or the usual logic programming setting. The key item needed is the existence of a pairing function on the domain that is computable in a suitable sense. Details can be found in [2].

4. PROGRAMS—MINIMAL MODEL SEMANTICS

The familiar minimal model semantics of van Emden and Kowalski [9] for logic programs without input extends readily to take inputs into account. The basic ideas are easily sketched. We assume the usual notions of *model* and *minimal model* are known, along with their general properties. The only modifications we assume are: the domain now is ω, not the usual Herbrand base, and SUCCESSOR represents the successor relation on ω in every model, as part of the criteria for being a model.

For this section, **P** is a program in which the one place predicate symbol **I** does not occur in the head of any axiom, and **O** is another one place predicate symbol. Then $[\mathbf{P}_\mathbf{O}^\mathbf{I}]$ is an enumeration operator, as characterized in Section 3.

Definition. Let S be some subset of ω. We call **M** an *S input model* for **P** if **M** is a model in which the axioms of **P** are true, and in addition,

$$n \in S \quad \Rightarrow \quad \mathbf{I}(n) \in \mathbf{M}.$$

In other words, the interpretation of **I** in **M** extends S. Note that we have $\Rightarrow$ in the definition above, not $\Leftrightarrow$. The stronger requirement of $\Leftrightarrow$ would make it nontrivial to show there are any S input models for **P**, while the weaker $\Rightarrow$ admits the usual "universal" model in which every atom is taken as true (except for the ones involving SUCCESSOR). Thus S input models for **P** exist. Also the usual "intersection property" holds: the intersection of any nonempty family of S input models for **P** is another such. Consequently there will be a *minimal* S input model for **P**. The main result concerning it is the following, which gives us the $\Leftrightarrow$ version after all. It is established by a straightforward extension of the usual van Emden–Kowalski argument.

Proposition 4.1. Let **M** *be the minimal S input model for* **P**. *Then*

(1) $S = \{n | \mathbf{I}(n) \in \mathbf{M},\}$

(2) $[\mathbf{P}_\mathbf{O}^\mathbf{I}](S) = \{n | \mathbf{O}(n) \in \mathbf{M}\}$.

Corollary 4.2. $n \in [\mathbf{P}_\mathbf{O}^\mathbf{I}](S) \Leftrightarrow \mathbf{O}(n) \in \mathbf{N}$ *for every S input model* **N** *for* **P**.

Models can be used to show enumeration operators are monotone and compact, as an alternative to the operational semantic arguments sketched in Section 3. In brief, the ideas are these.

For monotonicity: Suppose $S_1 \subseteq S_2 \subseteq \omega$. Then any S_2 input model for **P** is also an S_1 input model for **P**. By Corollary 4.2, $[\mathbf{P}_\mathbf{O}^\mathbf{I}](S_1)$ is included in the interpretation of **O** in any S_1 input model for **P**. But the minimal S_2 input model for **P** is an S_1 input model for **P**, and the interpretation of **O** in it is $[\mathbf{P}_\mathbf{O}^\mathbf{I}](S_2)$.

For compactness: Although the intersection of a family of models for **P** is another one, the same is not true for union. But it is easy to show the union of a *directed* family of models for **P** yields another model for **P**. (A family **F** of models is directed if, for any $\mathbf{M}_1, \mathbf{M}_2 \in \mathbf{F}$, there is an $\mathbf{M}_3 \in \mathbf{F}$ with $\mathbf{M}_1 \subseteq \mathbf{M}_3$ and $\mathbf{M}_2 \subseteq \mathbf{M}_3$.)

Now suppose $n \in [\mathbf{P}_\mathbf{O}^\mathbf{I}](S)$; we show $n \in [\mathbf{P}_\mathbf{O}^\mathbf{I}](F_0)$ for some finite $F_0 \subseteq S$.

For each finite subset F of S, let $\mathbf{M}(F)$ be the minimal F input model for **P**. And let **F** be $\{\mathbf{M}(F) | F \text{ a finite subset of } S\}$. **F** is a directed family of models for **P**; hence $\bigcup\mathbf{F}$ is a model for **P**. Also $\bigcup\mathbf{F}$ is an S input model; hence $\mathbf{O}(n) \in \bigcup\mathbf{F}$. But then $\mathbf{O}(n) \in \mathbf{M}(F_0)$ for some finite $F_0 \subseteq S$, so $n \in [\mathbf{P}_\mathbf{O}^\mathbf{I}](F_0)$.

The proofs that are sketched (or omitted) in this section can be found in detail in [Fitting 1986].

5. GENERALIZATION

Up to now we have only considered enumeration operators that took sets (one place relations) as inputs and returned sets as outputs. This is a concession to history: it is what enumeration operators were defined to be in [7]. Also, extending the original definition to anything more complicated involves layers of unpleasant coding. But the characterization in terms of logic programming generalizes easily.

Let **P** be a program. Let $\mathbf{I}_1, \ldots, \mathbf{I}_k$ be predicate symbols, none of which occur in the head of any axiom in **P**. Say $\mathbf{I}_1$ is n_1-place,..., $\mathbf{I}_k$ is n_k-place. Finally, let **O** be an m-place predicate symbol. We extend the notation of previous sections to allow $[\mathbf{P}_\mathbf{O}^{\mathbf{I}_1, \ldots, \mathbf{I}_k}]$ to denote a (generalized) enumeration operator. It takes as input an n_1-place relation,..., an n_k-place relation, and produces as output an m-place relation. The behavior of it is characterized by directly extending the semantics of Sections 3 and 4 as follows.

Operationally, $[\mathbf{P}_\mathbf{O}^{\mathbf{I}_1, \ldots, \mathbf{I}_k}](S_1, \ldots, S_k)$ is the set of m-tuples $\langle x_1, \ldots, x_m \rangle$ such that $\mathbf{O}(x_1, \ldots, x_m)$ is derivable from the extended program $\mathbf{P} \cup \{\mathbf{I}_1(y_1, \ldots, y_{n_1}) | \langle y_1, \ldots, y_{n_1} \rangle \in S_1\} \cup \cdots \cup \{\mathbf{I}_k(y_1, \ldots, y_{n_k}) | \langle y_1, \ldots, y_{n_k} \rangle \in S_k\}$.

For a model theoretic characterization, define an $S_1, \ldots, S_k$ *input model* for **P** to be a model **M** in which the axioms of **P** are true, and in which the interpretation of $\mathbf{I}_1$ extends S_1,..., the interpretation of $\mathbf{I}_k$ extends S_k. Then Proposition 4.1 extends readily to: $[\mathbf{P}_\mathbf{O}^{\mathbf{I}_1, \ldots, \mathbf{I}_k}](S_1, \ldots, S_k)$ is the interpretation of **O** in the minimal $S_1, \ldots, S_k$ input model for **P**.

Monotonicity and compactness results are easily established, using the same arguments that were given in earlier sections for the restricted case. From now on we use the terms operator and enumeration operator in the extended sense given above.

6. CLOSURE OPERATIONS

If we think of enumeration operators as the modules of logic programming, then what we need next is mechanisms for building more elaborate modules from simpler ones. In this section we consider several, ranging from composition to minimalization or recursion. Not only is the family of enumeration operators closed under the operations considered here, it is *effectively* closed, and the closure is *low overhead*. That is, if you supply me with the axioms for the operators that one of the operations below is to act on, I can return the axioms for the operator that results, and this axiom set will be only a little more complicated than those of the original operators. In the next section we give an adequacy result, showing the operations considered here are sufficient in a reasonable sense.

The results of this section are, for the most part, taken from [2], though the proof of the first recursion theorem generalization is from [3].

Composition or Substitution

We begin with the most elementary operation, feeding the output of one operator to another as input. To keep notation down we only discuss simple cases, but the method is general.

Suppose $[\mathbf{P}_{\mathbf{J}}^{\mathbf{I}}]$ and $[\mathbf{Q}_{\mathbf{L}}^{\mathbf{K}}]$ are enumeration operators, with both **L** and **I** being n-place predicate symbols. We can define an operator $\mathbf{\Phi}$ by

$$\mathbf{\Phi}(S) = [\mathbf{P}_{\mathbf{J}}^{\mathbf{I}}]\big([\mathbf{Q}_{\mathbf{L}}^{\mathbf{K}}](S)\big).$$

We wish to show $\mathbf{\Phi}$ is an *enumeration* operator. The idea is to convert the axioms for **P** and **Q** into an axiom system for the composition by adding a linking axiom saying: feed output of **Q** to **P** as input.

We say **P** and **Q** are *disjoint* if the only predicate symbol they have in common is SUCCESSOR. Since predicate symbols can be renamed without affecting behavior, **P** and **Q** can always be made disjoint. Assume this has been done. Now, very simply, let **R** consist of

axioms of **P**,

axioms of **Q**,

$\mathbf{I}(x_1,\ldots,x_n) \leftarrow \mathbf{L}(x_1,\ldots,x_n)$.

It is easy to see, intuitively, that $\mathbf{\Phi} = [\mathbf{R}_{\mathbf{J}}^{\mathbf{K}}]$ and hence is an enumeration operator. Or more formally, either the operational or the minimal model semantics can be used to give a proof that $\mathbf{\Phi} = [\mathbf{R}_{\mathbf{J}}^{\mathbf{K}}]$.

The "linking" technique above easily extends to more general cases. Suppose, for instance, that $[\mathbf{P}_{\mathbf{C}}^{\mathbf{A},\mathbf{B}}]$, $[\mathbf{Q}_{\mathbf{F}}^{\mathbf{D},\mathbf{E}}]$, and $[\mathbf{R}_{\mathbf{H}}^{\mathbf{G}}]$ are enumeration operators, with **F** and **A** both n-place, and **H** and **B** both k-place. Then the following is an enumeration operator:

$$(\lambda S_1, S_2, S_3)[\mathbf{P}_{\mathbf{C}}^{\mathbf{A},\mathbf{B}}]\big([\mathbf{Q}_{\mathbf{F}}^{\mathbf{D},\mathbf{E}}](S_1, S_2), [\mathbf{R}_{\mathbf{H}}^{\mathbf{G}}](S_3)\big).$$

It is $[\mathbf{T}_{\mathbf{C}}^{\mathbf{D},\mathbf{E},\mathbf{G}}]$, where **T** consists of (assuming disjointness of **P**, **Q**, and **R**):

axioms of **P**,

axioms of **Q**,

axioms of **R**,

$\mathbf{A}(x_1,\ldots,x_n) \leftarrow \mathbf{F}(x_1,\ldots,x_n)$,

$\mathbf{B}(x_1,\ldots,x_k) \leftarrow \mathbf{H}(x_1,\ldots,x_k)$.

Intersection and Union

Suppose $[\mathbf{P}_{\mathbf{B}}^{\mathbf{A}_1,\ldots,\mathbf{A}_n}]$ and $[\mathbf{Q}_{\mathbf{D}}^{\mathbf{C}_1,\ldots,\mathbf{C}_n}]$ are enumeration operators, where corresponding predicate symbols have the same arity, that is, $\mathbf{A}_i$ and $\mathbf{C}_i$ are both n_i-place, and **B** and **D** are both k-place. We define the *intersection* and *union* of these operators pointwise, as follows:

$$[\mathbf{P}_{\mathbf{B}}^{\mathbf{A}_1,\ldots,\mathbf{A}_n}] \cap [\mathbf{Q}_{\mathbf{D}}^{\mathbf{C}_1,\ldots,\mathbf{C}_n}]$$
$$= (\lambda S_1,\ldots,S_n)\{[\mathbf{P}_{\mathbf{B}}^{\mathbf{A}_1,\ldots,\mathbf{A}_n}](S_1,\ldots,S_n) \cap [\mathbf{Q}_{\mathbf{D}}^{\mathbf{C}_1,\ldots,\mathbf{C}_n}](S_1,\ldots,S_n)\},$$
$$[\mathbf{P}_{\mathbf{B}}^{\mathbf{A}_1,\ldots,\mathbf{A}_n}] \cup [\mathbf{Q}_{\mathbf{D}}^{\mathbf{C}_1,\ldots,\mathbf{C}_n}]$$
$$= (\lambda S_1,\ldots,S_n)\{[\mathbf{P}_{\mathbf{B}}^{\mathbf{A}_1,\ldots,\mathbf{A}_n}](S_1,\ldots,S_n) \cup [\mathbf{Q}_{\mathbf{D}}^{\mathbf{C}_1,\ldots,\mathbf{C}_n}](S_1,\ldots,S_n)\}.$$

The claim is, both $\cap$ and $\cup$ turn enumeration operators into enumeration operators. And again, the ideas are simple. First, make **P** and **Q** disjoint. Then, the intersection operator is $[\mathbf{M}_{\mathbf{O}}^{\mathbf{I}_1,\ldots,\mathbf{I}_n}]$, where **M** consists of (using obvious vector notation)

axioms of **P**,

axioms of **Q**,

$\mathbf{A}_1(\mathbf{x}) \leftarrow \mathbf{I}_1(\mathbf{x})$,

$\ldots,$

$\mathbf{A}_n(\mathbf{x}) \leftarrow \mathbf{I}_n(\mathbf{x})$,

$\mathbf{C}_1(x) \leftarrow \mathbf{I}_1(\mathbf{x})$,

$\ldots,$

$\mathbf{C}_n(\mathbf{x}) \leftarrow \mathbf{I}_n(\mathbf{x})$,

$\mathbf{O}(\mathbf{y}) \leftarrow \mathbf{B}(\mathbf{y}), \mathbf{D}(\mathbf{y})$.

And union is $[\mathbf{J}_{\mathbf{O}}^{\mathbf{I}_1,\ldots,\mathbf{I}_n}]$, where the axioms for **J** are those of **M**, except that the last one is replaced by

$\mathbf{O}(\mathbf{y}) \leftarrow \mathbf{B}(\mathbf{y})$,

$\mathbf{O}(\mathbf{y}) \leftarrow \mathbf{D}(\mathbf{y})$.

Cartesian Product

This time suppose $[\mathbf{P}_{\mathbf{B}}^{\mathbf{A}_1,\ldots,\mathbf{A}_n}]$ and $[\mathbf{Q}_{\mathbf{D}}^{\mathbf{C}_1,\ldots,\mathbf{C}_n}]$ are enumeration operators with $\mathbf{A}_i$ and $\mathbf{C}_i$ both n_i-place predicate symbols, **B** b-place, and **D** d-place (where b and d may be different). Then

$$[\mathbf{P}_{\mathbf{B}}^{\mathbf{A}_1,\ldots,\mathbf{A}_n}] \times [\mathbf{Q}_{\mathbf{D}}^{\mathbf{C}_1,\ldots,\mathbf{C}_n}]$$
$$= (\lambda S_1,\ldots,S_n)\{[\mathbf{P}_{\mathbf{B}}^{\mathbf{A}_1,\ldots,\mathbf{A}_n}](S_1,\ldots,S_n) \times [\mathbf{Q}_{\mathbf{D}}^{\mathbf{C}_1,\ldots,\mathbf{C}_n}](S_1,\ldots,S_n)\}.$$

It is straightforward to show enumeration operators are closed under $\times$.

Minimalization

Suppose $[\mathbf{P}_{\mathbf{O}}^{\mathbf{I}_1,\ldots,\mathbf{I}_n}]$ is an enumeration operator, and $\mathbf{I}_i$ and **O** are both k-place predicate symbols. Let $S_1,\ldots,S_{i-1},S_{i+1},\ldots,S_n$ be fixed input relations, and consider the operator as a function of the ith input alone:

$$\Phi = (\lambda S)[\mathbf{P}_{\mathbf{O}}^{\mathbf{I}_1,\ldots,\mathbf{I}_n}](S_1,\ldots,S_{i-1},S,S_{i+1},\ldots,S_n).$$

Φ is itself monotone, and hence has a least fixed point, a k-place relation. We take $\mu_i[\mathbf{P}_{\mathbf{O}}^{\mathbf{I}_1,\ldots,\mathbf{I}_n}]$ to be the operator mapping $S_1,\ldots,S_{i-1},S_{i+1},\ldots,S_n$ to that fixed point. More precisely,

$$\mu_i[\mathbf{P}_{\mathbf{O}}^{\mathbf{I}_1,\ldots,\mathbf{I}_n}] = (\lambda S_1,\ldots,S_{i-1},S_{i+1},\ldots,S_n)\{\text{the least fixed point of } (\lambda S)[\mathbf{P}_{\mathbf{O}}^{\mathbf{I}_1,\ldots,\mathbf{I}_n}](S_1,\ldots,S_{i-1},S,S_{i+1},\ldots,S_n)\}.$$

The family of enumeration operators is closed under the application of μ_i, minimalization on the ith input, and again the ideas are simple. To keep notation down, we deal with the following special case: we have the enumeration operator $[\mathbf{P}_{\mathbf{J}}^{\mathbf{I},\mathbf{K}}]$, where $\mathbf{I}$ and $\mathbf{J}$ are n-place and $\mathbf{K}$ is k-place. We wish to show $\mu_1[\mathbf{P}_{\mathbf{J}}^{\mathbf{I},\mathbf{K}}]$ is an enumeration operator. The intuition is this. In general, to produce the least fixed point of a continuous operator, keep recycling output as input. This idea leads to the following representation for $\mu_1[\mathbf{P}_{\mathbf{J}}^{\mathbf{I},\mathbf{K}}]$. It is the enumeration operator $[\mathbf{Q}_{\mathbf{J}}^{\mathbf{K}}]$, where $\mathbf{Q}$ consists of

axioms of $\mathbf{P}$,

$\mathbf{I}(\mathbf{x}) \leftarrow \mathbf{J}(\mathbf{x})$.

This time we sketch the argument that the enumeration operator we propose is correct. Using standard facts about fixed points in complete lattices, it is enough to show the following. Suppose S is a k-place relation, and let $T = [\mathbf{Q}_{\mathbf{J}}^{\mathbf{K}}](S)$. Then

(1) $[\mathbf{P}_{\mathbf{J}}^{\mathbf{I},\mathbf{K}}](T,S) \subseteq T$;

(2) if $[\mathbf{P}_{\mathbf{J}}^{\mathbf{I},\mathbf{K}}](R,S) \subseteq R$ then $T \subseteq R$.

For (1) we argue as follows. Let $\mathbf{M}$ be the minimal S input model for $\mathbf{Q}$. Then S is the interpretation of $\mathbf{K}$ in $\mathbf{M}$, and T is the interpretation of $\mathbf{J}$. Since $\mathbf{I}$ represented an input to $\mathbf{P}$, it could not occur in any axiom head of $\mathbf{P}$. Consequently $\mathbf{I}(\mathbf{x}) \leftarrow \mathbf{J}(\mathbf{x})$ is the only axiom for $\mathbf{I}$ in $\mathbf{Q}$, so $\mathbf{I}$ and $\mathbf{J}$ represent the same relation in $\mathbf{M}$, namely T. This means $\mathbf{M}$ is also a T, S input model for $\mathbf{P}$; hence by Corollary 4.2, $[\mathbf{P}_{\mathbf{J}}^{\mathbf{I},\mathbf{K}}](T,S) \subseteq$ (interpretation of $\mathbf{J}$ in $\mathbf{M}$) $= T$.

For (2), suppose $[\mathbf{P}_{\mathbf{J}}^{\mathbf{I},\mathbf{K}}](R,S) \subseteq R$; we show $T \subseteq R$. This time let $\mathbf{M}$ be the minimal R, S input model for $\mathbf{P}$. Then in $\mathbf{M}$ the interpretation of $\mathbf{I}$ is R, the interpretation of $\mathbf{K}$ is S, and the interpretation of $\mathbf{J}$ is $[\mathbf{P}_{\mathbf{J}}^{\mathbf{I},\mathbf{K}}](R,S)$, and hence is a subset of R. It follows that $\mathbf{I}(\mathbf{x}) \leftarrow \mathbf{J}(\mathbf{x})$ is true in $\mathbf{M}$, and hence all axioms of $\mathbf{Q}$ are true in $\mathbf{M}$. Then $\mathbf{M}$ is also an S input model for $\mathbf{Q}$, so by Corollary 4.2 again, $T = [\mathbf{Q}_{\mathbf{J}}^{\mathbf{K}}](S) \subseteq$ (the interpretation of $\mathbf{J}$ in $\mathbf{M}$) $\subseteq R$.

The result above is a generalization of the Kleene first recursion theorem, which is the special case in which one starts with a single input operator, producing a zero input operator, a recursively enumerable relation, as a result. The proof presented is taken from [3].

7. A REPRESENTATION THEOREM

In practice, if it ever comes to that, one would use the operations of the previous section to combine operators of some complexity. To establish their sufficiency, we state a result that says, if we start with a small set of very simple enumeration operators, and close under the operations of Section 6, we get all enumeration operators. We begin by laying out the elementary operators we start with. In each case, it is trivial that the operators in question really are *enumeration* operators.

Structural Operators

(1) *Transposition.* For each $1 \leq i, j \leq n$, let $T^n_{i,j}$ be the operator that takes an n-place relation R as input and returns R with "columns" i and j transposed.

(2) *Projection, or existential quantification.* For each $n \geq 2$, let P^n be the operator that takes an n-place relation R as input and returns the $(n-1)$-place relation $(\exists x_1)R(x_1, \ldots, x_n)$.

We also want some "constant" operators. We can take these to be enumeration operators with zero inputs, $[\mathbf{P}_O]$, which are just the recursively enumerable relations.

General Constants

(3) *Null.* The constant whose value is $\varnothing$.

(4) *Equality.* The constant whose value is $\{\langle x, x\rangle | x \in \omega\}$.

Domain Specific Constants

(5) *Zero.* The constant whose value is $\{0\}$.

(6) *Successor.* The constant whose value is $\{\langle n, n+1\rangle | n \in \omega\}$.

Proposition 7.1. The enumeration operators are exactly those operators generated from the transposition, projection, null, equality, zero, and successor operators using the operations of substitution, intersection, union, product, and minimalization. In fact, each enumeration operator can be obtained with a single use of minimalization.

This is a mild generalization (to allow multiple input operators) of Corollary 2.7.3 from [2] and is not re-proved here. The essential idea is: rewrite the program defining an enumeration operator using the machinery of first order logic as one does when constructing a completed data base; then simulate the resulting logic formula using the machinery allowed by the proposition. Further, two different predicate symbols, say $P(\mathbf{x})$ and $Q(\mathbf{x})$, can be replaced by one, provided we use constants to distinguish cases: say $R(0,\mathbf{x})$ for $P(\mathbf{x})$, and $R(1,\mathbf{x})$ for $Q(\mathbf{x})$. It is by such a trick, replacing all predicate symbols except for SUCCESSOR by a single predicate symbol, that the restriction to a single use of minimalization can be achieved.

We close with two examples, one easy, one a little less so.

Example 1. Consider the following operator, where R and S are 1-place: *If R is nonempty then S.* (A nontrivial else case is not possible here, because R could be recursively enumerable but not recursive.) This is an enumeration operator, as is easy to see. It can be represented using the machinery of Proposition 7.1 very simply, as follows: $P^2(R \times S)$.

Example 2. Let $\mathbf{\Phi}(\mathbf{G},\mathbf{R},\mathbf{S})$ be the operator that, when supplied with a semigroup multiplication table $\mathbf{G}$ (representing the semigroup members by numbers), and two sets $\mathbf{R}$ and $\mathbf{S}$ of semigroups members, returns the set of members common to the semigroups generated by $\mathbf{R}$ and $\mathbf{S}$. $\mathbf{\Phi}$ is an enumeration operator. This is easy to see directly, but we wish to illustrate the use of the machinery discussed here. First we construct an auxiliary enumeration operator $[\mathbf{P}_{\mathbf{O}}^{\mathbf{A},\mathbf{B},\mathbf{I}}]$ with axioms as follows:

$$\mathbf{O}(z) \leftarrow \mathbf{B}(z),$$
$$\mathbf{O}(z) \leftarrow \mathbf{B}(x), \mathbf{I}(y), \mathbf{A}(x,y,z),$$
$$\mathbf{O}(z) \leftarrow \mathbf{B}(x), \mathbf{I}(y), \mathbf{A}(y,x,z).$$

The idea is, if $\mathbf{G}$ is a semigroup multiplication table, $\mathbf{R}$ is a subset of the semigroup, and $\mathbf{T}$ is part of the semigroup generated by $\mathbf{R}$, then $[\mathbf{P}_{\mathbf{O}}^{\mathbf{A},\mathbf{B},\mathbf{I}}](\mathbf{G},\mathbf{R},\mathbf{T})$ will be more of that semigroup. Then it is easy to see that

$$\mathbf{\Phi} = (\lambda \mathbf{G},\mathbf{R},\mathbf{S})\big(\mu_3[\mathbf{P}_{\mathbf{O}}^{\mathbf{A},\mathbf{B},\mathbf{I}}]\big)(\mathbf{G},\mathbf{R}) \cap \big(\mu_3[\mathbf{P}_{\mathbf{O}}^{\mathbf{A},\mathbf{B},\mathbf{I}}]\big)(\mathbf{G},\mathbf{S}).$$

Finally, the axioms of $\mathbf{P}$ convert to the following logic expression:

$$\begin{aligned}\mathbf{O}(z) \leftarrow \mathbf{B}(z) & \\ & \vee (\exists y)(\exists x)[\mathbf{B}(x) \wedge \mathbf{I}(y) \wedge \mathbf{A}(x,y,z)] \\ & \vee (\exists y)(\exists x)[\mathbf{B}(x) \wedge \mathbf{I}(y) \wedge \mathbf{A}(y,x,z)],\end{aligned}$$

and hence $[\mathbf{P}_{\mathbf{O}}^{\mathbf{A},\mathbf{B},\mathbf{I}}](\mathbf{G},\mathbf{R},\mathbf{I})$ itself can be represented as

$$\begin{aligned}&\mathbf{R} \cup P^2P^3[(\mathbf{R}\times\omega\times\omega)\cap(\omega\times\mathbf{I}\times\omega)\cap\mathbf{G}] \\ &\quad \cup P^2P^3[(\mathbf{R}\times\omega\times\omega)\cap(\omega\times\mathbf{I}\times\omega)\cap T^3_{1,2}(\mathbf{G})],\end{aligned}$$

where ω is the constant relation $P^2(\mathit{equality})$.

8. CONCLUSION

Enumeration operators are functions, not at ground level, but at a level up. Proposition 7.1 then says we can do logic programming in a functional programming style provided we work at this level, using relations as inputs and outputs.

Still, a functional approach is not customary for logic programs which are open to query about many relations, not just a single designated output relation. But we can generalize beyond Section 7 to allow multiple outputs as well as multiple inputs, $[\mathbf{P}_{\mathbf{O}_1,\ldots,\mathbf{O}_k}^{\mathbf{I}_1,\ldots,\mathbf{I}_n}]$. One can think of this as a family of operators in the old sense, $[\mathbf{P}_{\mathbf{O}_i}^{\mathbf{I}_1,\ldots,\mathbf{I}_n}]$ for $i = 1,2,\ldots,k$, all having the same set of axioms. The operational and minimal model semantics extend in the obvious way. Likewise the closure operations of Section 6 generalize naturally. The result is very near to that proposed by [5].

An "operator" $[\mathbf{P}_{\mathbf{O}_1,\ldots,\mathbf{O}_k}^{\mathbf{I}_1,\ldots,\mathbf{I}_n}]$ is the formal counterpart of a module with axioms $\mathbf{P}$, having predicates $\mathbf{O}_1,\ldots,\mathbf{O}_k$ which can be queried, being able to issue queries itself concerning $\mathbf{I}_1,\ldots,\mathbf{I}_n$, and with all other machinery purely internal and closed off from the outside world. They are similar to micro-Prolog's notion of module, with its export list and its import list, except, of course, for ignoring Prolog's deterministic control structure (Clark and McCabe [1]). These multiple input, multiple output "operators" are, we think, the right candidates for the semantic understanding of logic program modules.

REFERENCES

1. Clark, K. and McCabe, F., *micro-Prolog: Programming in Logic*, Prentice-Hall, Englewood Cliffs, N.J., 1984.
2. Fitting, M., *Fundamentals of Generalized Recursion Theory*, North-Holland, Amsterdam, 1981.
3. Fitting, M., *Computability Theory, Semantics and Logic Programming*, Oxford U.P., to appear.
4. Lassez, J. and Maher, M., Closure and Fairness in the Semantics of Logic Programming, *Theoret. Comput. Sci.* 29:167–184 (1984).
5. O'Keefe, R., Towards an Algebra for Constructing Logic Problems, in *IEEE 1985 Symoposium on Logic Programming*, 1985 pp. 152–160.
6. Plotkin, G., A Set-Theoretical Definition of Application, Memo MIP-R-95, School of Artificial Intelligence, Edinburgh, 1972.
7. Rogers, H., *Theory of Recursive Functions and Effective Computability*, McGraw-Hill, New York, 1967.
8. Scott, D., Data Types as Lattices, *SIAM J. Comput.* 5:522–587 (1976).
9. Van Emden, M. and Kowalski R., The Semantics of Predicate Logic as a Programming Language, *J. Assoc. Comput. Mech.* 23:733–742 (1976).