A DETERMINISTIC PROLOG FIXPOINT SEMANTICS*

MELVIN FITTING

▷ A fixpoint semantics for PROLOG is presented that takes into account PROLOG's deterministic control structure as well as its logic features. It is applied to prove the correctness of a prime-number program based on the sieve of Eratosthenes, a program whose behavior depends critically on the sequential nature of PROLOG. ◁

1. INTRODUCTION:

PROLOG is the best-known of the logic programming languages. But it still shares an important feature with imperative languages: the order in which things are done is determined by the order in which statements are written. PROLOG is implemented in a deterministic way that causes conditions to be checked from left to right, and axioms to be tried from top down. As a consequence, the following two PROLOG procedures behave quite differently when queried with ← even(2).:

$$\text{even}(0).$$

$$\text{even}(X) \leftarrow \text{diff}(X, 2, Y) \& \text{even}(Y).$$

and

$$\text{even}(X) \leftarrow \text{diff}(X, 2, Y) \& \text{even}(Y).$$

$$\text{even}(0).$$

The first succeeds with the query, while the second gets stuck in an infinite computation.

PROLOG has been provided with an elegant, simple fixpoint semantics [4], essentially deriving from first-order model theory. But it only applies to PROLOG as

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

*The research for this paper was partially supported by NSF grant MCS83-04959.

52 Vanderbilt Ave., New York, NY 10017 0743-1066/85/$03.30

a "pure" nondeterministic logic programming language. It does not take control, implicitly specified by statement order, into account. As a consequence, both programs above would be assigned the same semantical meaning. In practice, however, issues of control are often vital. In the example above it means the difference between output and no output.

What we propose here is a fixpoint semantics for PROLOG that is sensitive to control as well as to logic. Not only will relations have instances, but those instances will be "discovered" in a particular order, which must be taken into account. The result is a semantics that has much in common with a deterministic imperative language semantics. We note that Francez et al. [1] have also found it useful to bring "conventional" methods into the PROLOG context, to prove program termination. On the other hand, if we choose to ignore issues of order, the semantics presented here essentially reverts to that of [4].

In the example above, choice of statement order had a blunt consequence: output or no output. The following is a more subtle example, and we follow it through this paper. The issue now is right answer or wrong answer. Of course one can argue that if the response can be changed from right to wrong by a change in control only, we simply have a badly written logic program. We do not argue with that. We do, however, have a correctly written PROLOG program, with a precise behavior which we wish to characterize. We want a semantics for PROLOG as it is.

We refer to the following program as SIEVE throughout this paper:

$\text{prime}(1, 2).$

$\text{prime}(N, X) \leftarrow \text{diff}(N, 1, I)\&\text{pass}(I, X).$

$\text{pass}(1, 3).$

$\text{pass}(1, X) \leftarrow \text{pass}(1, Y)\&\text{sum}(Y, 2, X).$

$\text{pass}(N, X) \leftarrow \text{prime}(N, P)\&\text{diff}(N, 1, I)\&$
$\text{pass}(I, X)\&\text{notdivide}(P, X).$

$\text{notdivide}(B, A) \leftarrow \text{quot}(A, B, C)\&\text{prod}(B, C, D)\&\text{ne}(D, A).$

The intention is: for a positive integer n, the X's for which pass(n, X) are the X's left undeleted in the nth pass of the sieve of Eratothsenes, and prime(n, P) is to mean P is the nth prime number. In fact, when given the initial goal $\leftarrow$ prime(n, P), for a particular integer n, the SIEVE program will determine the nth prime as its value for P, *provided* PROLOG has been given its standard deterministic implementation. A pure logic-programming-based semantics is not adequate to account for the behavior of SIEVE. Indeed, it will allow both prime$(2, 3)$ and prime$(3, 3)$ to be part of the "meaning" of prime. We leave it to the reader to verify this.

The semantics we present, starting in the next section, is still a fixpoint semantics, though on a more complex space than that used in [4]. To illustrate its uses, we apply it to proving the behavior of SIEVE. We do not consider any of the control- or logic-altering PROLOG features like cut or addition and deletion of axioms.

REMARK. The SIEVE program above is hopelessly inefficient and is not intended to be a paradigm of how to determine primes using PROLOG.

2. WHAT WE ASSUME

We assume a general familiarity with PROLOG; we present no detailed syntax, nor any discussion of unification, backtracking, or meeting a goal. We use one nonstandard piece of terminology. Suppose we query a PROLOG program by setting it an initial goal. It responds with a value. We reject the value and ask again, setting it a new goal, like the original one, but with the extra condition: not the first value. When we get a response, we reject it and ask again, and so on. We call this a *repeated query*; the response is a sequence of outputs. When necessary for emphasis, we will refer to a conventional initial goal setting as a *simple query*.

For example, the reader should be able to predict the output of the following when $\leftarrow b(X)$ is set as a repeated query:

$$a(1).$$

$$a(7).$$

$$b(0).$$

$$b(X) \leftarrow \text{diff}(X, 1, Y) \& a(Y). \tag{1}$$

The output will be 0, 2, 8, in that order. It is the same response we would get if we added the clause

$$\text{print} \leftarrow b(X) \& \text{write}(X) \& \text{fail}.$$

and we set the initial goal $\leftarrow$ print.

By a *goal set* we mean, simply, a set of goals, not necessarily finite. We do not distinguish between two goal sets that differ only by a variable renaming. For example, $\{\text{prime}(N, P), \text{pass}(N, X)\}$ and $\{\text{prime}(M, Z), \text{pass}(N, X)\}$ are goal sets we will identify.

If G is a goal, by a *G-sequence* we mean a finite or infinite sequence of tuples that can serve as "values" for G. We also allow $\perp$, undefined, as a G-sequence. For example, if G is the goal prime(2, X), the following are G-sequences: $\perp$; $\langle\,\rangle$ (empty sequence); $\langle(2,3),(2,3),(2,5)\rangle$; and $\langle(2,3),(2,4),(2,5),\ldots\rangle$.

For two G-sequences s_1 and s_2 we set $s_1 \leq s_2$ if s_1 is an initial segment (not necessarily proper) of s_2, or if s_1 is $\perp$.

If $\boldsymbol{C}$ is a goal set, then a *C-state function*, or just *C-state*, is a function f mapping each member G of $\boldsymbol{C}$ to a G-sequence $f(G)$. For example, let C be the goal set given above, and let f be the function such that $f(\text{prime}(N, P))$ is the sequence

$$\langle(1,2),(2,3),(3,5),\ldots,(n,p_n),\ldots\rangle,$$

where p_n is the nth prime, and $f(\text{pass}(N, X))$ is the sequence

$$\langle(1,3),(1,5),(1,7),\ldots,(1,2n+1),\ldots\rangle.$$

Suppose P is a PROLOG program, $\boldsymbol{C}$ is a goal set, and f is a $\boldsymbol{C}$-state. We say f is a *P-model* provided, for each $G \in \boldsymbol{C}$, $f(G)$ is the sequence of values resulting from a repeated query of P with G. For example, let P be the program (1) given above, let $\boldsymbol{C}$ be the goal set $\{a(X), b(X)\}$, and let f be the function

$$f(a(X)) = \langle 1, 7\rangle,$$

$$f(b(X)) = \langle 0, 2, 8\rangle.$$

Then f is a P-model.

Note that P-models depend on the goal set C as well as on P; what constitutes a model is sensitive to what questions we intend to ask. However, for a given C there is only one P-model.

Let C be a fixed goal set for the time being. By $|C|$ we mean the set of all C-states. We define an ordering on $|C|$ as follows. For $f, g \in C$, $f \leq_C g$ provided as sequences, $f(G) \leq g(G)$ for each $G \in C$. By *C-space* we mean $\langle |C|, \leq_C \rangle$. For convenience we call f a member of C-space if $f \in |C|$. Also we leave the subscript off $\leq$, as the set C in question can usually be determined from context.

It is easy to check that a C-space is always a partially ordered set with a unique smallest element, which we also denote $\perp$. Further, the inf of any nonempty subset always exists. On the other hand, we do not have closure under sup, so a C-space is not a complete lattice. We do have the following useful weaker alternative, however, whose verification we leave to the reader. Call a nonempty subset S of $|C|$ *directed* if for any $f, g \in S$ there is some $h \in S$ with $f \leq h$ and $g \leq h$. Then, every C-space is closed under the taking of sups of directed sets. In particular, if $f_1, f_2, \ldots$ is a *chain* in $|C|$, that is, if $f_1 \leq f_2 \leq \cdots$, then it is a directed set and so will have a sup, or limit.

Suppose Φ is a mapping on $|C|$. Then Φ is *monotone* provided $f \leq g$ implies $\Phi(f) \leq \Phi(g)$. A classic result of Knaster [2] and Tarski [3] says that monotone maps on complete lattices always have smallest (and largest) fixed points. We observed above that a C-space is not a complete lattice, but we still have enough machinery to establish a fixed-point result. We state it and present the proof.

Theorem. Let Φ be a monotone map on C-space. Then Φ has a smallest fixed point, denoted $\mu\Phi$. In fact, if $\Phi(f) \leq f$ then $\mu\Phi \leq f$.

PROOF. Let L be $\{ f \in |C| \mid \Phi(f) \leq f \}$. If we had a complete lattice, there would be a largest member $T \in |C|$. Trivially $\Phi(T) \leq T$, so L would be nonempty. Then $\inf(L)$ would exist, and the traditional Tarski argument would establish that it is the smallest fixed point of Φ, and meets our other condition. But we do not have a complete lattice; in particular there is no largest member, so this argument breaks down. Still, if we establish that L is nonempty, everything else goes through. That can be done, but by a more compliçated argument.

Let U be $\{ f \in |C| \mid f \leq \Phi(f) \}$. $|C|$ has a smallest member $\perp$, and certainly $\perp \leq f(\perp)$, so U is not empty. Let $f_1, f_2, \ldots$ be a chain in U. The chain has a limit, say g, in $|C|$. For each i, $f_i \leq g$, so $f_i \leq \Phi(f_i) \leq \Phi(g)$. It follows that $g \leq \Phi(g)$, so $g \in U$. Thus U is closed under chain limits. Using Zorn's lemma, U has a maximal member, say m. Since $m \in U$, $m \leq \Phi(m)$. By monotonicity, $\Phi(m) \leq \Phi(\Phi(m))$, so $\Phi(m) \in U$. If $m \neq \Phi(m)$, then $\Phi(m)$ would dominate a maximal member of U, which is impossible. Hence $\Phi(m) = m$, and Φ has at least one fixed point, and so L is not empty. □

Given a PROLOG program P and a goal space C, our task essentially is to identify which member of C-space is the model for P. Anticipating the next section, it will turn out to be the least fixed point of a monotone function on C-space that is associated with P in a straightforward way.

3. STATE MAPPINGS

Suppose P is a PROLOG program. Some of the identifiers appearing in P may be among the built-ins of the language, with known behavior. Others may be of our creation, but we may have established their behavior earlier. We refer to these kinds of identifiers as *background* identifiers. We are not concerned with them; they are known quantities.

To each identifier of P that is not a background one, we associate two distinct new identifiers in the following simple way. With the identifier ident we associate the identifiers inident and outident. Finally, we associate with the program P a new program P^0 as follows. Each nonbackground identifier ident in a P clause is replaced by outident on the head ends of arrows, and by inident on the body sides of arrows, to give us program P^0.

For example, in the SIEVE program from Section 1, we can take diff, sum, prod, quot, ne, and notdivide as background identifiers. Then SIEVE^0 is the program

outprime(*1*, *2*).

outprime(N, X) ← diff(N, *1*, *I*)&inpass(I, X).

outpass(*1*, *3*).

outpass(*1*, X) ← inpass(*1*, Y)&sum(Y, *2*, X).

outpass(N, X) ← inprime(N, P)&diff(N, *1*, *I*)&
inpass(I, X)&notdivide(P, X).

notdivide(B, A) ← quot(A, B, C)&prod(B, C, D)&ne(D, A).

Notice that P^0 will involve no recursions (except possibly on background identifiers). Not surprisingly, the intent of the present semantics is to account for the behavior of recursive programs provided the behavior of nonrecursive programs is understood.

Let C be a goal set. We call C *P-closed* provided, if we choose any member of C, say G, and we issue to P^0 the repeated query ← out$G$, in the course of program execution any further goal to be proved (other than those involving background identifiers) will be of the form in$H$, where $H \in C$.

For example, the goal set {prime(N, P), pass(N, X)} is not SIEVE-closed, because if we issue the goal ← outpass(N, X) and reject the first response, which will be outpass(1, 3), then the goal ← inpass(1, Y) will arise, and pass(1, Y) is not in the goal set. On the other hand, the goal set {prime(1, P), prime(2, P), ..., pass(1, X), pass(2, X), ...} is SIEVE-closed, as can be easily verified.

Now let P be a fixed PROLOG program and let C be a goal set that is P-closed. We associate with P a mapping Φ_P on C-space. Let $f \in |C|$; $\Phi_P(f)$ is to be the state function specified as follows. Let G be a goal in C. Then $(\Phi_P(f))(G)$ is the G-sequence whose terms are the result of issuing the repeated query ← out$G$ to the program P^0, but with the understanding that, if a call on, say, in$H$ is generated, values are supplied using the sequence $f(H)$, beginning with the first term, and moving to successive terms if backtracking comes up.

For example, let P be the program

$$\text{even}(X) \leftarrow \text{diff}(X, 2, Y)\&\text{even}(Y).$$

$$\text{even}(0).$$

and let C be the goal set $\{\text{even}(X)\}$, which is P-closed. Suppose we have state functions given by $f(\text{even}(X)) = \langle 1, 3\rangle$; $g(\text{even}(X)) = \langle\,\rangle$; $h(\text{even}(X)) = \bot$. Then $(\Phi_P(f))(\text{even}(X)) = \langle 3, 5, 0\rangle$; $(\Phi_P(g))(\text{even}(X)) = \langle 0\rangle$; $(\Phi_P(h))(\text{even}(X)) = \bot$. (Thus $\bot$ must be the smallest fixed point of Φ_P.)

As a more elaborate example, let C be the goal set $\{\text{prime}(1, P), \text{prime}(2, P), \ldots, \text{pass}(1, X), \text{pass}(2, X), \ldots\}$. Let f be the state given by

$$f(\text{prime}(1, P)) = \langle (1, 2)\rangle,$$

$$f(\text{prime}(n, P)) = \bot \quad \text{if} \quad n > 1,$$

$$f(\text{pass}(1, X)) = \langle (1, 3), (1, 5), (1, 7), \ldots\rangle,$$

$$f(\text{pass}(n, X)) = \bot \quad \text{if} \quad n > 1.$$

Then $\Phi_{\text{SIEVE}}(f) = g$, where

$$g(\text{prime}(1, P)) = \langle (1, 2)\rangle,$$

$$g(\text{prime}(2, P)) = \langle (2, 3), (2, 5), (2, 7), \ldots\rangle,$$

$$g(\text{prime}(n, P)) = \bot \quad \text{if} \quad n > 2,$$

$$g(\text{pass}(1, X)) = \langle (1, 3), (1, 5), (1, 7), \ldots\rangle,$$

$$g(\text{pass}(n, X)) = \bot \quad \text{if} \quad n > 1.$$

It is easy to see that Φ_P will always be a *monotone* map on C-space, and so will have a smallest fixed point. In fact, it will be a P-model.

It is also the case that Φ_P is *continuous*. That is, if $f_1, f_2, \ldots$ is a chain in C-space, then $\Phi_P(\lim f_i) = \lim \Phi_P(f_i)$. Essentially the reason is that any item of P^0 output that went into $\Phi_P(\lim f_i)$ is the result of a successful computation, which terminated in a finite time and hence used only a finite amount of the information contained in $\lim f_i$, and that information must already be available in some member of the chain $f_1, f_2, \ldots$. It follows that we can approximate to the least fixed point of Φ_P in the familiar way by considering the chain $f_1, f_2, \ldots$, where

$$f_1 = \bot,$$

$$f_{n+1} = \Phi_P(f_n).$$

Finally, we state the main result, and briefly sketch a proof.

Theorem. Let P be a PROLOG *program and let C be a P-closed goal set. The least fixed point of Φ_P is a P-model.*

SKETCH OF PROOF.

Part 1. Let $m \in C$ be the P-model. If one issues a repeated query $\leftarrow \text{out}G$ to $\Phi_P(m)$, it is not hard to see that the sequence of responses will be the same

ones that arise when issuing the repeated query $\leftarrow G$ to m. Then $\Phi_P(m) = m$, so $\mu\Phi_P \leq m$.

Part 2. Suppose (induction hypothesis) that for any repeated query in $\boldsymbol{C}$ that P will answer in fewer than k computational steps, the response (which by definition is embodied in the model m) agrees with $\mu\Phi_P$. Now issue a query Q that requires k computational steps to answer. After an appropriate axiom has been selected and its head unified with, we are left with several subgoals, responses to which will require fewer than k computational steps. Making use of the induction hypothesis, P's response to Q will be the same as the response of $\Phi_P(\mu\Phi_P)$ to Q. But since $\mu\Phi_P$ is a fixed point, P's response to Q will agree with $\mu\Phi_P$. It follows by induction that $m \leq \mu\Phi_P$. □

4. THE SIEVE EXAMPLE CONCLUDED

Once again let $\boldsymbol{C}$ be the goal set {prime(1, P), prime(2, P), ..., pass(1, X), pass(2, X), ...}, and let m be the $\boldsymbol{C}$-state specified as follows:

$$m(\text{pass}(1, X)) = \langle (1,3), (1,5), (1,7), (1,9), \ldots \rangle,$$

$$m(\text{pass}(2, X)) = \langle (2,5), (2,7), (2,11), (2,13), \ldots \rangle,$$

and in general, m(pass(n, X)) is the sequence $\langle (n, a_1), (n, a_2), \ldots \rangle$, where $a_1, a_2, \ldots$ is the sequence of numbers left after the nth pass of the sieve of Eratothsenes. Further,

$$m(\text{prime}(1, P)) = \langle (1,2) \rangle,$$

and for $n > 1$,

$$m(\text{prime}(n, P)) = \langle (n, a_i) : i = 1, 2, \ldots \rangle,$$

where $m(\text{pass}(n-1, X)) = \langle (n-1, a_i) : i = 1, 2, \ldots \rangle$.

We show $m = \mu\Phi_{\text{SIEVE}}$, which will establish that a *simple* query of SIEVE with $\leftarrow$ prime(n, P) will get us the nth prime as the value of P.

First we argue that $\Phi_{\text{SIEVE}}(m) \leq m$. But this is straightforward. Let $g = \Phi_{\text{SIEVE}}(m)$ for convenience, and consider g(pass(n, X)), where $n > 1$, as an example. If we give SIEVE^0 the repeated query $\leftarrow$ outpass(n, X), using m as input, then since $n > 1$, the fifth clause of SIEVE^0 will be involved and we will have essentially

$$\text{outpass}(n, X) \leftarrow \text{inprime}(n, P) \& \text{inpass}(n-1, X) \& \text{notdivide}(P, X).$$

By construction, m(prime(n, P)) must begin with (n, p_n), where p_n is the nth prime. Repeatedly issuing the query $\leftarrow$ outpass(n, X) will cause backtracking only as far as inpass($n-1$, X), and by definition m(pass($n-1$, X)) involves the sequence remaining after pass $n-1$ of the prime sieve. Then it follows easily that the sequence of responses to our repeated query $\leftarrow$ outpass(n, X) must be that sequence with multiples of p_n removed; but this is exactly the sequence corresponding to the nth pass of the prime sieve. That is, the value of g(pass(n, X)) must be m(pass(n, X)).

Other cases are similar. We thus have $\Phi_{\text{SIEVE}}(m) = m$, and so $\mu\Phi_{\text{SIEVE}} \leq m$. Define states s_k and t_k as follows:

$$s_k(\text{prime}(n, P)) = \begin{cases} m(\text{prime}(n, P)), & 1 \leq n \leq k, \\ \bot, & n > k, \end{cases}$$

$$s_k(\text{pass}(n, X)) = \begin{cases} m(\text{pass}(n, X)), & 1 \leq n \leq k, \\ \bot, & n > k, \end{cases}$$

$$t_k(\text{prime}(n, P) = \begin{cases} m(\text{prime}(n, P)), & 1 \leq n \leq k+1, \\ \bot, & n > k+1, \end{cases}$$

$$t_k(\text{pass}(n, X)) = \begin{cases} m(\text{pass}(n, X)), & 1 \leq n \leq k, \\ \bot, & n > k. \end{cases}$$

It is trivial that $s_1, t_1, s_2, t_2, \ldots$ is a chain, with m as its limit. Further, examination of the program SIEVE^0 shows that $\Phi_{\text{SIEVE}}(s_k) = t_k$ and $\Phi_{\text{SIEVE}}(t_k) = s_{k+1}$. Finally a straightforward induction argument shows that $s_1 \leq f$, where f is any fixed point of Φ_{SIEVE}. It follows using monotonicity that $s_i \leq f$ and $t_i \leq f$ for each i and any fixed point f. Then $m \leq \mu\Phi_{\text{SIEVE}}$.

5. CONCLUSION

There are two more points we wish to make. First, we did not have much to say about the behavior of PROLOG on nonrecursive programs. We assumed that understood, and only considered the effects of recursion. Consequently, our fixpoint analysis will still apply if PROLOG's method of goal selection is subjected to "reasonably straightforward" modification. We do not elaborate on this.

Our second point is that not much modification is necessary to apply a similar treatment to PROLOG implemented in a truly nondeterministic manner. Redefine the state function so that $f(G)$, say, is not a *sequence* of tuples, but a *set* of tuples. Redefine the order relation $\leq_C$ so that $f \leq_C g$ means, for each $G \in \mathbf{C}$, $f(G) \subseteq g(G)$. Essentially, no other changes are necessary. Mathematically things are nicer because we now have a complete lattice. The resulting semantics should be compared with that in [4]. They are essentially the same.

REFERENCES

1. Francez, N., Grumberg, O., Katz, S., and Pnueli, A., Proving termination of PROLOG programs. *Logic of Programs*, Springer Lecture Notes, Springer, New York, 1985, pp. 89-105.
2. Knaster, B., Un théorème sur les fonctions d'ensembles, *Ann. Soc. Polon. Math.* 6:133-134 (1928).
3. Tarski, A., A Lattice-theoretical Fixpoint Theorem and Its Applications, *Pacific J. Math.* 5:285-309 (1955).
4. Van Emden, M. and Kowalski, R., The Semantics of Predicate Logic as a Programming Language, *J. Assoc. Comput. Mach.* 23:733-742 (1976).