LOG(F) A NEW SCHEME FOR INTEGRATING REWRITE RULES, LOGIC PROGRAMMING AND LAZY EVALUATION

Sanjai Narain

July 1987
CSD-870027

LOG(F): A New Scheme for Integrating Rewrite Rules, Logic Programming and Lazy Evaluation

Sanjai Narain

ABSTRACT

We present LOG(F), a new scheme for integrating rewrite rules logic programming and lazy evaluation. First, we develop a simple yet expressive rewrite rule system F* for representing functions. F* is non-Noetherian, i.e. an F* program *can* admit infinite reductions. For this system, we develop a reduction strategy called select and show that it possesses the property of reduction-completeness. Because of this property, select exhibits a weak form of lazy evaluation.

We then show how to implement F* in Prolog. Specifically, we compile rewrite rules of F* into Prolog clauses in such a way that when Prolog interprets these clauses it directly simulates the behavior of select. In particular, Prolog behaves lazily. Since it is not necessary to change Prolog it is possible to do lazy evaluation efficiently. Since Prolog is already a logic programming system, a combination of rewrite rules, logic programming and lazy evaluation is achieved.

1.0 DEFINITION OF F*

Variables. There is a countably infinite list of variables.

Function symbols. There is a countably infinite list of 0-ary function symbols. In particular, [], 0, true, false, are 0-ary function symbols. There is a countably infinite list of 1-ary function symbols. In particular, s is a 1-ary function symbol. There is a countably infinite list of 2-ary function symbols. In particular, | is a 2-ary function symbol. And so on, for all other arities.

Connectives. The connectives are =>, (,), ',',.

Constructor Symbols. There is an infinite subset of the function symbols called Constructors. Each element of Constructors is called a constructor symbol. For each n, n>=0, Constructors contains an infinite number of n-ary function symbols. In particular, 0, true, false, [] and | are constructor symbols.

Terms. A term is either a variable, a 0-ary function symbol or an expression of the form f(t1,..tn) where f is an n-ary function symbol, n>0, and each ti is a term. A term is called ground if it contains no variables. **However, unless explicitly stated otherwise, by a term we mean a ground term.**

Subterms. Let E be a term. Then E is a subterm of E. Also, if E=f(t1,..,tn), n>0, then X is a subterm of E if X is a subterm of ti. If

X is a subterm of E, X is said to occur in E.

Abbreviations. The symbols 1,2,3,... are, respectively, abbreviations for s(0), s(s(0)), s(s(s(0))),.....

Substitutions. A substitution is a set {<X1,t1>,..,<Xn,tn>} where each Xi is a variable and each ti is a term. A variable X is defined in a substitution σ iff for some term s, <X,s> occurs in σ. Let σ be a substitution and E be a term, possibly containing variables. Then Eσ represents the result of applying σ to E.

Reduction Rules. A reduction rule is of the form:

```
LHS=>RHS
```

where LHS and RHS are terms. LHS is called the head of the rule. The following restrictions are placed on LHS and RHS:

(a) LHS is not a variable.

(b) LHS is not of the form c(t1,..,tn) where c is an n-ary constructor symbol, n>=0.

(c) If LHS=f(t1,t2,..,tn), n>=0, each ti is a variable or a term of the form c(X1,..,Xn) where c is an n-ary constructor symbol, n>=0, and each Xi is a variable.

(d) There is at most one occurrence of any variable in LHS.

(e) All variables of RHS appear in LHS.

These restrictions are not very limiting. As can be seen from the examples below, many common functions can be defined in F*. However, these restrictions enable F* to possess many useful properties.

F* programs. An F* program consists of a set of reduction rules. Some examples of F* programs are:

```
append([],X)=>X
append([U|V],W)=>[U|append(V,W)]

if(true,X,Y)=>X.
if(false,X,Y)=>Y.
not(true)=>false.
not(false)=>true.

lesseq(0,X)=>true.
lesseq(s(X),s(Y))=>lesseq(X,Y).
lesseq(s(X),0)=>false.
greater(X,Y)=>not(lesseq(X,Y)).
```

```
merge([A|B],[C|D])=>
      if(lesseq(A,C),[A|merge(B,[C|D]),[C|merge([A|B],D)]).

int(N)=>[N|int(s(N))].

partition(U,[A|B],L,R)=>if(lesseq(U,A),partition(U,B,[A|L],R),
                                       partition(U,B,L,[A|R])).
partition(U,[],L,R)=>t(L,R).

quicksort([])=>[].
quicksort([A|B])=>quicksort1(A,partition(A,B,[],[])).
quicksort1(A,t(L,R))=>append(quicksort(L),append([A],quicksort(R))).
```

2.0 REDUCTIONS

We now consider the reduction of **terms**. Again, unless explicitly stated, by a term we mean a ground term.

E=>$_P$E1. Let P be an F* program and E and E1 be terms. We say E=>$_P$E1 if there is a rule LHS=>RHS in P such that LHS and E unify with m.g.u. σ and E1 is RHSσ. We also say that E reduces to E1 by the rule LHS=>RHS, or that the rule applies to the whole of E. Note that if E is ground and E=>$_P$E1 then, by restriction (e) E1 is also ground. If P is clear from the context we write E=>E1 in place of E=>$_P$E1.

E->$_P$E1,E-*>$_P$E1. Let P be an F* program and E be a term. Let G be a subterm of E such that G=>$_P$H. Let E1 be the result of substituting H for G in E. Then we say that E->$_P$E1. Note that if E=>$_P$E1 then E unifies with the left hand side of some rule in P. If E->$_P$E1 then some subexpression of E, including possibly E, unifies with the left hand side of some rule in P. We define -*>$_P$ to be the reflexive transitive closure of ->$_P$. Again, if P is clear from context we write E->E1 or E-*>E1 in place of E->$_P$E1 or E-*>$_P$E1.

Reductions. Let P be an F* program. A reduction in P is a sequence E1,E2,... such that for each i, when Ei and Ei+1 both exist, Ei->$_P$Ei+1.

Simplified forms. A term is said to be in simplified form or simplified if it is of the form c(t1,..,tn) where c is an n-ary constructor symbol, n>=0, and each ti is a term.

Successful reductions. Let P be an F* program. A successful reduction in P is a reduction E1,..,En, n>0, in P, such that for each i if i<n then Ei is not simplified, and, if i=n then Ei is simplified.

R_P(G,H,A,B). Let P be an F* program. Where G,H,A,B are terms, R_P(G,H,A,B) is defined as follows:

R_P(G,H,A,B) if (a) G=>H, and
(b) B is identical with A except that zero or more occurrences of G in A are replaced by H.

Note that A and G can be identical. Again, if P is clear from context we omit the prefix from R_P.

Reduction strategy. Let P be an F* program. A reduction strategy for P takes as input a term E and selects a subterm G of E such that there exists a term H such that $G \Rightarrow_P H$.

A special reduction strategy. Let P be an F* program. We now define a reduction strategy, $select_P$ for P. Informally, given a term E it will select that subterm of E whose reduction is necessary in order that some => rule in P apply to the whole of E. Where f(t1,..,tn) is a term, n>=0 the relation $select_P$ is:

$select_P$(f(t1,..,tn),f(t1,..,tn)) if f(t1,..,tn)$\Rightarrow_P$X.
$select_P$(f(t1,..,ti,..,tn),X) if
    there is a rule f(L1,..,Li,..,Ln)=>RHS in P, and
    ti does not unify with Li, and
    $select_P$(ti,X).

Again, if P is clear from context the subscript P on $select_P$ is omitted. Note the following: (1) when $select_P$ takes as input E and returns G, it also, implicitly, returns an occurrence of G in E. This occurrence can be obtained from the proof of $select_P$(E,G) (2) if $select_P$(E,G) then there is a term H such that $G \Rightarrow_P H$ (3) if there is more than one => rule in P, then there could be more than one G such that $select_P$(E,G) (4) since, by restriction (b) there is no rule in P of the form c(t1,..,tn)=>RHS, where c is a constructor symbol, if E is simplified, $select_P$ is undefined for E. For example, where P is the set of reduction rules which appear above, we have the following:

select(merge(int(1),int(2)),int(1)).
select(merge(int(1),int(2)),int(2)).
select(merge([1,3],int(2)),int(2)).
select(merge([1,2],[3,4]),merge([1,2],[3,4])).
If E=[1|merge(int(1),int(2))] then select is undefined for E.

N-step. Let P be an F* program and E,G,H be terms. Suppose $select_P$(E,G) and $G \Rightarrow_P H$. Let E1 be the result of replacing G by H in E. Then we say that E reduces to E1 in an N-step in P. The qualification "in P" is omitted when P is clear from context. It should be noted that there may be many occurrences of G in E. However, the specific occurrence in E to be replaced by H is the occurrence returned by $select_P$. The prefix N in N-step is intended to connote normal order.

N-reduction. Let P be an F* program. An N-reduction in P is a reduction E1,E2,.... in P such that for each i when Ei and Ei+1 both exist, Ei reduces to Ei+1 in an N-step in P. In particular, the sequence E where E is a term, is an N-reduction in P. The qualification "in P" is omitted when P is clear from the context.

3.0 REDUCTION-COMPLETENESS OF select

Lemma 1. Let P be an F* program. If A->B and B is simplified but A is not, then A=>B.

Proof. Since A is not simplified, A=f(t1,..,tn) where f is not a constructor symbol and each ti is a term. Since the reduction of A to B replaces this symbol, it follows that A must reduce as a whole to B. Thus A=>B.

Lemma 2. Let P be an F* program. Let X1,..,Xn be variables, G,H,t1,..,tn,t1*,..,tn* be terms such that for each i R(G,H,ti,ti*). Let σ={<X1,t1>,..,<Xn,tn>} and τ={<X1,t1*>,..,<Xn,tn*>} be substitutions. Suppose M is a term, possibly containing variables, whose variables are a subset of {X1,..,Xn}. Then R(G,H,Mσ,Mτ).

Proof. By induction on length of M. Since M is a term, possibly containing variables, it is either a variable, a 0-ary function symbol or of the form f(N1,..,Nk) where f is an n-ary function symbol and each Ni is a term, possibly containing variables.

If M is a variable Xi, then Mσ=ti and Mτ=ti* and so clearly R(G,H,Mσ,Mτ). If M is a 0-ary function symbol then Mσ=M and Mτ=M and obviously R(G,H,M,M). Let M=f(N1,..,Nk). Assume the lemma holds for N1,..,Nk, i.e., for all i, R(G,H,Niσ,Niτ). f(N1,..,Nk)σ=f(N1σ,..,Nkσ). Similarly, f(N1,..,Nk)τ=f(N1τ,..,Nkτ), and hence R(G,H,Mσ,Mτ).

Lemma 3. Let P be an F* program. If:
(1) G, H, E1=f(t1,..,tn) and F1=f(t1*,..,tn*) are terms, and
(2) R(G,H,ti,ti*) for every i in 1,..,n.
(3) B=f(L1,..,Ln) is the head of some rule in P, and
(4) E1 unifies with B with m.g.u. σ

Then:
(1) F1 unifies with B with m.g.u. τ, and
(2) σ and τ define exactly the same variables, i.e. only those occurring in B, and
(3) If pair <X,s> occurs in σ and <X,s*> occurs in τ then R(G,H,s,s*).

Proof. Since by restriction (d) a variable occurs at most once in B=f(L1,..,Ln), a term f(d1,..,dn) unifies with B iff for each i, di unifies with Li with m.g.u. σi. So, the union of the σi is a unifier of f(d1,..,dn) and B. Consider some Li in L1,..,Ln. By restriction (c) there are the following cases.

Case 1. Li is a variable. Then Li unifies with ti* with m.g.u. τi={<Li,ti*>}. Also, the pair <Li,ti> appears in σ. By assumption, R(G,H,ti,ti*).

Case 2. Li=c(X1,..,Xm), m>=0, c a constructor symbol and each Xj a variable. Then since ti unifies with Li, ti=c(s1,..,sm) where each si is a term. Thus the pairs {<X1,s1>,..,<Xm,sm>} appear in σ.

If ti is identical with ti*, ti* also unifies with Li with m.g.u. τi={<X1,s1>,..,<Xm,sm>}. Of course, for every i, R(G,H,si,si).

If ti is not identical with ti* then since R(G,H,ti,ti*), ti contains at least one occurrence of G and G=>H. Since ti=c(s1,..,sm), c a constructor symbol, by restriction (b) ti=/=G. Hence ti*=c(s1*,..,sm*) each si* a term and for every i R(G,H,si,si*). Hence ti* unifies with Li with m.g.u. τi={<X1,s1*>,..,<Xm,sm*>}.

The same argument can be repeated for every other Li. Let τ be the union of the τi. Then τ is a unifier of B and F1. Since for each pair <X,d> in τ, d is ground, τ is most general. Thus (1).

Since τ is an m.g.u. of F1 and B, it contains only pairs <X,d> such that X is a variable of B. For the same reason, if X is a variable of B then some pair <X,d>, where d is a term, occurs in τ. Otherwise Bτ would contain X. Thus τ defines only those variables which occur in B. Similarly for σ. Thus σ and τ define exactly the same variables. Thus (2).

If some pair <X,d*> appears in τ, then, by the above discussion <X,d> appears in σ and R(G,H,d,d*). Thus (3). QED.

Lemma 4. Let P be an F* program. If:
(1) f(t1,..,ti,..,tn) is a term, and
(2) f(L1,..,Li-1,c(X1,..,Xm),Li+1,..,Ln)=>RHS is a rule in P, and
(3) ti=d1,d2,d3,..,dr, r>0, is an N-reduction.

Then:
f(t1,..,ti-1,d1,ti+1,..,tn), f(t1,..,ti-1,d2,ti+1,..,tn), .., f(t1,..,ti-1,dr,ti+1,..,tn) is also an N-reduction.

Proof. Let Li=c(X1,..,Xm). Since f(L1,..,Li,..,Ln)=>RHS is a rule, by restriction (b) f is not a constructor symbol. If r=1 then, by definition of N-reduction, the lemma is obvious. So, assume r>1.

By definition of N-reduction, at most the last member of the sequence d1,d2,d3,..,dr can be in simplified form. Hence, since Li=c(X1,..,Xm), none of the di, 0<i<r unify with Li.

We now show that for all j, 0<j<r, f(t1,..,ti-1,dj,ti+1,..,tn) reduces to f(t1,..,ti-1,dj+1,ti+1,..,tn) in an N-step. Since dj is not simplified, it does not unify with Li. Hence, by definition of select, for every X select(f(t1,..,ti-1,dj,ti+1,..,tn),X) if select(dj,X).

Since dj reduces to dj+1 in an N-step there are terms pj and qj such that select$_p$(dj,pj), pj=>qj and dj+1 is the result of replacing pj by qj in dj. Then f(t1,..,ti-1,dj,ti+1,..,tn) reduces to f(t1,..,ti-1,dj+1,ti+1,..,tn) in an N-step.

Hence, f(t1,..,ti-1,d1,ti+1,..,tn), f(t1,..,ti-1,d2,ti+1,..,tn), .., f(t1,..,ti-1,dr,ti+1,..,tn) is an N-reduction. QED.

Theorem 1.

Let P be an F* program. Let E1,F1,F2,G,H be terms such that
(1) E1 is not simplified, and
(2) R(G,H,E1,F1), and
(3) F1 reduces to F2 in an N-step

Then there is an N-reduction E1,..,E2 in P such that R(G,H,E2,F2).

Proof. It is helpful to draw the following diagram:

```
E1  F1     G occurs 0 or more times in E1
|   |      G=>H
*  N-step
|   |
v   v
E2  F2
```

We have to show that R(G,H,E2,F2).

We proceed by induction on length of E1. The cases we have to consider in the proof can be laid out as below. Here, if a case is annotated with = it is easy to deal with, while if annotated with +, it requires some consideration.

```
           E1=0-ary fn symbol?
              /        \
            yes        no, E1=f(t1,..,tn)
            /             \
        E1=F1?           E1=F1?
         / \              / \
       yes  no          yes  no
       /     \          /     \
      =       =        =     E1=G?
                              / \
                            yes no
                            /     \
                           =    F1 reduces to F2 by some rule R in P?
                                   / \
                                 yes  no
                                 /     \
             E1 reduces to E2 by R?   +
                              /   \
                            yes    no
                            /       \
                           +         +
```

Suppose E1 is a 0-ary function symbol. If E1=F1 then E1,F2 is an

N-reduction and R(G,H,F2,F2). If E1=/=F1 then since R(G,H,E1,F1) G must occur in E1, and F1 is the result of replacing G in E1 by H. So, E1=G and E1=>F1, there is an N-reduction E1,F1,F2 and R(G,H,F2,F2). That is, putting E2=F2 satisfies the theorem.

Otherwise, since we are given that E1 is not simplified, E1=f(t1,..,tn), n>=0, f not a constructor symbol. Assume the theorem for every term whose length is less than that of f(t1,..,tn).

If E1=F1 then E1,F2 is an N-reduction and R(G,H,F2,F2), so putting E2=F2 satisfies the theorem. Otherwise E1=/=F1. If E1=G then since R(G,H,E1,F1), E1=>F1, and E1,F1,F2 is an N-reduction and R(G,H,E2,F2). Again, that is, putting E2=F2 satisfies the theorem.

Having considered the easy cases, we arrive at the interesting cases, with E1=/=F1 and G occurs in E1 but G=/=E1. Hence F1=f(t1*,..,tn*) where for every i, R(G,H,ti,ti*). We now consider the following subcases:

1. F1=>F2. Then there is a rule f(L1,..,Ln)=>RHS in P such that F1 and f(L1,..,Ln) unify with m.g.u. τ and F2=RHSτ.

1-1. E1 and f(L1,..,Ln) do unify. Let the m.g.u. be σ. By Lemma 3, F1 and f(L1,..,Ln) also unify with some m.g.u. β. Since F1 already unifies with f(L1,..,Ln) with m.g.u. τ, τ=β.

E1=>RHSσ and so E2=RHSσ. The N-reduction is E1,E2. Of course F2=RHSτ. By Lemma 3, σ and τ define exactly the same variables, and if <X,s> occurs in σ and <X,s*> appears in τ then R(G,H,s,s*). Hence, by Lemma 2, R(G,H,E2,F2).

1-2. E1 and f(L1,..,Ln) do not unify. Then, since E1 is ground and each variable occurs at most once in f(L1,..,Ln), there is at least one Li in L1,..,Ln such that ti does not unify with Li. Hence Li is not a variable and so Li=c(X1,..,Xm), c a constuctor symbol and each Xi a variable.

Since R(G,H,ti,ti*), and ti does not unify with Li, ti is not simplified. Suppose ti were simplified. Either ti=c(s1,..,sm), each si a term. But then ti must unify with Li. Contradiction. Or, ti=d(s1,..,sm), d a constructor symbol, d=/=c, each si a term. Since R(G,H,ti,ti*), ti=G and ti*=H. By restriction (b) this is impossible.

Since F1 unifies with f(L1,..,Ln), ti* unifies with Li, and so ti* is simplified. Since ti is not simplified, and R(G,H,ti,ti*), ti=>ti*. Thus select(E1,ti). Hence f(t1,..,ti,..,tn) reduces to f(t1,..,ti*,..,tn) in an N-step.

Hence there exists an N-reduction E1=P1,P2,P3,.. where for each i Pi=f(s1,..,sn), sk=tk or sk=tk*, and if sk does not unify with Lk then Pi+1 is derived from Pi by replacing sk by sk* such that sk=>tk*. We also have for each i R(G,H,Pi,F1). This reduction cannot be infinite since F1=f(t1*,..,tn*) unifies with f(L1,..,Ln). Let the last term be Pm. Let the m.g.u. of Pm and f(L1,..,Ln) be σ. Then Pm=>RHSσ. Hence we have the N-reduction E1,P2,P3,..,Pm,RHSσ. By Lemma 3, there is an m.g.u. of F1 and f(L1,..,Ln) and clearly this is τ. Already, F2=RHSτ. By Lemma 2, R(G,H,RHSσ,F2).

2. There is no F2 such that F1=>F2, i.e. select(F1,F1) is not true. Or, F1 does not unify with the head of any reduction rule in P. Hence select(E1,E1) is not true. If it were, there would be a contradiction with Lemma 1. We are given that F1 reduces to F2 by an N-step. We now have to show that there is an N-reduction E1,..,E2 such that R(G,H,E2,F2).

   Suppose select(F1,u). Then u occurs in some ti*. That is, there is some i such that select(ti*,u). Let u=>v and let ti** be the result of replacing u in ti* by v. Hence ti* reduces to ti** in an N-step, and also F2=f(t1*,..,ti**,..,tn*). By definition of select, there is a rule f(L1,..,Li,..,Ln)=>RHS in P such that ti* does not unify with Li. Hence Li=c(X1,..,Xm), m>=0, where c is a constructor symbol and each Xi is a variable.

   Clearly, ti* is not simplified. So, by restriction (b) ti is also not simplified. ti* reduces to ti** in an N-step. We already have R(G,H,ti,ti*). Since the length of ti is less than f(t1,..,ti,..,tn), by induction hypothesis there is an N-reduction ti=d1,d2,..,dr, r>=1, such that R(G,H,dr,ti**). By Lemma 4, the sequence f(t1,..,ti-1,ti,ti+1,..,tn), f(t1,..,ti-1,d2,ti+1,..,tn),.., f(t1,..,ti-1,dr,ti+1..,tn) is an N-reduction. We already have F2=f(t1*,..,ti**,..,tn*) and for each k R(G,H,tk,tk*). Hence R(G,H,f(t1,..,ti-1,dr,ti+1,..,tn), f(t1,..,ti-1*,ti**,ti+1*..,tn).

QED

Theorem 2. The reduction-completeness of F*

Let P be an F* program and D0 be a ground term. Let D0,D1,..,Dn=c(t1,..,tx), be a successful reduction in P for some ground terms D1,..,Dn and t1,..,tx and some constructor symbol c. Then there is a successful N-reduction D0,Q1,..,Qp=c(s1,..,sx) in P for some ground terms Q1,..,Qp and s1,..,sx.

Proof. By induction on length n of successful reduction D0,D1,..,Dn.
If n=0, then D0 is already simplified and D0 is the successful reduction and D0=c(t1,..,tx).

If n=1 then D0=>D1. Hence, select(D0,D0), so we have the successful N-reduction D0,Q1=D1. Again, Q1=c(t1,..,tx).

Assume the theorem holds for n=k-1 i.e. for the successful reduction D1,..,Dk=c(t1,..,tx). We now show that it holds for the successful reduction D0,D1,..,Dk. By induction hypothesis, D1 has a successful

N-reduction, say D1,F2,F3,F4,..,Fm=c(p1,..,px). Of course, all terms in this sequence, except Fm, are unsimplified.

Since D0->D1, there are terms G,H such that G occurs in D0, G=>H and D1 is the result of replacing G by H in D0. Hence R(G,H,D0,D1).

Hence by theorem 1 there is an N-reduction D0,..,E2 such that R(G,H,E2,F2). If F2 is not simplified, then since R(G,H,E2,F2), by restriction (b) E2 is also not simplified.

By repeatedly applying theorem 1 we have the N-reductions D0,..,E2, and E2,..,E3, ... and Em-1,..,Em for some finite m>=2, such that for each i, 2=<i=<m R(G,H,Ei,Fi) and at most Em is simplified. The resulting situation can be laid out in the following diagram:

```
D0       D1       R(G,H,D0,D1)
|        |
|       N-step
*        |
v        v
E2       F2       R(G,H,E2,F2)
|        |
|       N-step
*        |
v        v
E3       F3       R(G,H,E3,F3)
...      ...
Em-1    Fm-1      R(G,H,Em-1,Fm-1)
|        |
|       N-step
*        |
v        v
Em       Fm       R(G,H,Em,Fm)
```

Since at most Em is simplified let S be the reduction D0,..,E2,..,E3,..,...,Em-1,..,Em. Clearly, S is an N-reduction.

If Em is simplified then since R(G,H,Em,Fm) and Fm=c(p1,..,px), Em=c(s1,..,sx) and for each i, R(G,H,si,ti). Then, S is the required N-reduction.

Otherwise, since Fm is simplified, R(G,H,Em,Fm), G=>H, we have Em=G and Fm=H, i.e. Em=>Fm. Hence, select(Em,Em), and so we have the N-step Em,Fm. The required N-reduction is then S,Fm which is S,c(p1,..,px). QED.

4.0 COMPILATION OF F* INTO PROLOG AND ITS CORRECTNESS

4.1 Compilation of F* into Prolog

Let P be an F* program. The translation of P into Prolog proceeds in two stages.

Stage 1. For each n-ary constructor symbol c in P generate the clause:

```
reduce(c(X1,..,Xn),c(X1,..,Xn))
```

Stage 2. Let f(t1,..,tn)=>RHS be a rule in P where f is an n-ary, n>=0, non-constructor function symbol and each of RHS and t1,..,tn is a term, possibly containing variables. For each such rule perform the following steps:

(a) Let A1,..,An be n distinct Prolog variables none of which occur in the rule. If ti is a variable generate the predication Ai=ti. If ti is c(X1,..,Xn) where c is a constructor symbol and each Xi a variable, generate the predication reduce(Ai,c(X1,..,Xn)). Let LHS_CONDS be the set of predications so generated.

(b) Let Out be a Prolog variable not occurring in the rule and different from A1,..,An. Generate the predication reduce(RHS,Out).

(c) Generate the clause

```
reduce(f(A1,..,An),Out):-LHS_CONDS U {reduce(RHS,Out)}
```

For example, the F* rules:

```
append([],X)=>X
append([U|V],W)=>[U|append(V,W)]
```

are compiled into:

```
reduce([],[]).
reduce([U|V],[U|V]).

reduce(append(A1,A2),Out):-reduce(A1,[]),A2=X,reduce(X,Out).
reduce(append(A1,A2),Out):-reduce(A1,[U|V]),A2=W,reduce([U|append(V,W)],Out).
```

4.2 Correctness of translation of F*

Lemma 5. Let P be an F* program. If:
(1) E0=f(t1,..,ti,..,tm), and
(2) Ek=f(s1,..,si,..,sm), and
(3) si is simplified, and
(4) E0,..,Ek, k>=0, is an N-reduction such that for no i Ei=>Ei+1.

Then there is a successful N-reduction ti,..,si of length less than or equal to the length k of E0,E1,..,Ek.

Proof: By induction on length of N-reduction E0,..,Ek. Suppose k=0. Since E0=f(t1,..,ti,..,tm), ti=si. The successful N-reduction is simply ti whose length is 0.

Suppose k>0. Assume that the lemma holds for all N-reductions of length k-1. Consider the N-reduction E1,..,Ek, k>=1, of length k-1. Since there is no i such that Ei=>Ei+1, E1=f(u1,..,ui,..,um) for terms u1,..,um. By induction hypothesis, there is an N-reduction ui,..,si whose length is less than or equal to k-1.

If ti=ui then there is a successful N-reduction ti,..,si, whose length is less than or equal to k-1 and so less than or equal to k.

If ti=/=ui then since E0 reduces to E1 by an N-step, by definition of select, ti reduces to ui in an N-step. We now have the successful N-reduction ti,ui,..,si of length less than or equal to k. QED.

Lemma 6. Let P be an F* program and PC its compiled version. Let A be a ground term and B a term, possibly containing variables, such that reduce(A,B) succeeds with answer substitution σ. Then Bσ is ground.

Proof: By induction on length n of successful SLD-derivation reduce(A,B),G1,..,Gn=□. If n=1 then A=c(t1,..,tm), c a constructor symbol each ti a term, m>=0. The query reduce(A,B) will succeed by matching the head of the clause reduce(c(X1,..,Xm),c(X1,..,Xm)). The answer substitution σ will be such that Bσ=A. Clearly Bσ is ground.

Assume lemma for successful SLD-derivations of length less than n. Let the successsful derivation starting at reduce(A,B) be of length n, n>1. Then A=f(t1,..,tm), m>=0, where f is a function symbol, but not a constructor symbol, and each ti is a ground term. Then there is a clause:

```
reduce(f(X1,..,Xm),Z):-Q,reduce(RHS,Z).
```

such that (a) this clause is the compilation of a rule f(L1,..,Lm)=>RHS (b) each of X1,..,Xm,Z is a distinct variable not appearing in f(L1,..,Lm)=>RHS (c) if Li is a variable then Xi=Li appears in Q. Otherwise reduce(Xi,Li) appears in Q (d) reduce(f(t1,..,tm),B) unifies with the head of this clause with some m.g.u. τ and its immediate descendant (Q,reduce(RHS,Z))τ has a successful SLD-derivation of length n-1. Clearly, τ={<X1,t1>,..,<Xm,tm>,<Z,B>} and so Zτ=B. Also, since RHS does not contain any of the Xi, RHSτ=RHS.

If Q is empty then m=0, so, by restriction (e) RHSτ is ground. By induction hypothesis, reduce(RHSτ,B) succeeds with answer substitution σ such that Bσ is ground. So, reduce(A,B) succeeds with answer substitution σ such that Bσ is ground.

Assume Q is non-empty. Let Q1,..,Qm be the members of Q. If Qi is Xi=Li then Qiτ=(ti=Li) and succeeds with answer substitution σi={<Li,ti>}. If Qi is reduce(Xi,Li) then Qiτ=reduce(ti,Li) and has a successful SLD-derivation of length less than or equal to n-1. Hence, by induction hypothesis, Qiτ succeeds with answer substitution σi such that Liσi is ground.

By restriction (e) all variables of RHS occur in L1,..,Lm. Hence, since each Liσi is ground, RHSτσ1,..,σm is ground. Already Zτ=B. Since B does not contain any variables in L1,..,Lm, Bσ1,..,σm=B. Hence reduce(RHS,Z)τσ1,..,σm = reduce(RHSσ1,..,σm,B). By induction hypothesis, this succeeds with answer substitution σ such that Bσ is ground. So, reduce(A,B) succeeds with answer substitution σ such that Bσ is ground. QED.

Lemma 7. Let P be an F* program and PC its compiled version. Let A and B be ground terms such that reduce(A,B) succeeds. Let D be a term possibly containing variables such that for some substitution α, Dα=B. Then reduce(A,D) succeeds with answer substitution α.

Proof: By induction on length n of successful SLD-derivation starting at reduce(A,B). If n=1 then A=B=c(t1,..,tm), c a constructor symbol each ti a term, m>=0. The query reduce(A,D) will succeed with answer substitution which is the m.g.u. of B and D. Since B is ground this m.g.u. is α.

Assume lemma for successful derivations of length less than n. Let the successful derivation starting at reduce(A,B) be of length n, n>1. Then A=f(t1,..,tm) where f is a function symbol, but not a constructor symbol, and each ti is a term. Then there is a clause:

```
reduce(f(X1,..,Xm),Z):-QU{reduce(RHS,Z)}
```

which is the translation of some rule in P. Also, reduce(f(t1,..,tm),B) unifies with the head of this clause with some m.g.u. τ={<X1,t1>,..,<Xn,tn>,<Z,B>} and its immediate descendant is (QU{reduce(RHS,Z)})τ. Since RHS does not contain any of the Xi, this is QτU{reduce(RHS,B)}. It has a successful derivation of length n-1.

If Q is empty, by restriction (e) RHSτ is ground. Otherwise let Q1,..,Qm be the members of Q. Consider some Qi. If Qi is Xi=Li, then Qiτ=(ti=Li) which succeeds with answer substitution σi={<Li,ti>}. Otherwise Qi=reduce(Xi,Li), so Qiτ=reduce(ti,Li). By Lemma 6 reduce(ti,Li) succeeds with answer substitution σi such that Liσi is ground. Since all the variables of RHS are in L1,..,Lm, RHSσ1,..,σm is again ground.

Since reduce(RHSσ1,..,σm,B) succeeds, by induction hypothesis, reduce(RHSσ1,..,σm,D) succeeds with answer substitution α. Now consider the query reduce(A,D). Again, by reasoning as above, reduce(RHS(*s1..σm,D) appears in an SLD-derivation of reduce(A,D).

Hence reduce(A,D) also succeeds with answer substitution α. QED.

Lemma 8. Let P be an F* program. Let PC be the compiled version of P. Let E0,..,En be a successful N-reduction. Then reduce(E0,En) succeeds (in the sense of SLD-resolution) in the presence of PC.

Plan of Proof: By induction on length of successful N-reduction E0,..,En. We show that there is some Ej in E0,..,En such that an SLD-derivation of reduce(E0,En) contains the goal reduce(Ej,En). Since Ej,..,En is also a successful N-reduction, by induction hypothesis, reduce(Ej,En) succeeds. Hence reduce(E0,En) succeeds.

Proof: By induction on length n of successful reduction E0,..,En. If n=0 then E0 is already simplified. In particular, E0=c(t1,..,tm) where c is an m-ary constructor symbol, m>=0, and t1,..,tm are terms. There is a clause in PC reduce(c(X1,..,Xm),c(X1,..,Xm)) where each Xi is a variable. Clearly reduce(E0,E0) succeeds.

Let n>0 and E0=f(t1,..,tm), f not a constructor symbol, each ti a term and m>=0. Assume theorem holds for all successful reductions of length less than n.

Since E0 is not simplified, the N-reduction is of the form E0,..,Ek-1,Ek,..,En, 0<k=<n, such that Ek-1=>Ek, but for each i, 0=<i<k-1, not(Ei=>Ei+1). Hence, Ek-1=f(s1,..,sm) for some terms s1,..,sm. Since Ek-1=>Ek, there is some rule f(L1,..,Lm)=>RHS such that Ek-1 unifies with f(L1,..,Lm) with m.g.u. σ and Ek=RHSσ. Since none of the Li share any variables σ is the union of σ1,..,σm such that Li and si unify with m.g.u. σi.

For each i, if Li is not a variable, then since Li and si unify, si is in simplified form. For such i, there is, by Lemma 5, a successful N-reduction ti,..,si of length less than or equal to k-1.

The rule f(L1,..,Lm)=>RHS is compiled into the Horn clause

```
reduce(f(X1,..,Xm),Z):- QU{reduce(RHS,Z)}
```

in accordance with the compilation rules stated above. This clause is contained in PC.

Consider the query reduce(E0,En), i.e. reduce(f(t1,..,tn),En). It unifies with reduce(f(X1,..,Xm),En) with m.g.u. τ= {<X1,t1>,..,<Xn,tn>,<Z,En>} and its immediate descendant is (QU{reduce(RHS,Z)})τ. Since RHS does not contain any of the Xi, this is QτU{reduce(RHS,En)}.

Let Q1,..,Qm be the members of Q. Consider some Qi. If Qi is Xi=Li, then Qiτ=(ti=Li) which succeeds with answer substitution σi={<Li,ti>}.

Otherwise Qi=reduce(Xi,Li), so Qiτ=reduce(ti,Li). Since there is a successful N-reduction ti,..,si of length less than or equal to k-1, by induction hypothesis, reduce(ti,si) succeeds. Since Liσi=si, by Lemma 7 reduce(ti,Li) also succeeds with answer substitution σi.

By repeating the same argument for each Qi, we see that an SLD-derivation starting at reduce(E0,En) contains reduce(RHSσ1,..,σn,En) as a member. Since σ is the union of σi and no variable is defined in more than one si, RHSσ1,..,σn= RHSσ. But RHSσ=Ek. Hence the SLD-derivation starting at reduce(E0,En) contains reduce(Ek,En). Since the length of the successful reduction Ek,..,En is less than n, by induction hypothesis, reduce(Ek,En) succeeds. Thus, the query reduce(E0,En) succeeds. QED.

Lemma 9. Let P be an F* program. Let PC be the compiled version of P. Let E0 and En be terms such that reduce(E0,En) succeeds (in the sense of SLD-resolution) in the presence of PC. Then there is a successful N-reduction E0,..,En.

Plan of Proof: By induction on length of successful SLD-derivation reduce(E0,En),..,□. We show that there is some goal reduce(Ej,En) in this derivation such that there is an N-reduction E0,..,Ej. Since reduce(Ej,En) succeeds, by induction hypothesis, there is a successful N-reduction Ej,..,En. So there is a successful N-reduction E0,..,Ej,..,En.

Proof: By induction on length n of successful SLD-derivation starting at reduce(E0,En). If n=1 then there is a clause reduce(c(X1,..,Xm),c(X1,..,Xm)) in PC such that reduce(E0,En) unifies with the head of this clause. Clearly, then, E0=En, En is simplified and the required N-reduction is simply E0.

Let n>0. Assume lemma for all successful derivations of length less than n. Assume E0=f(t1,..,tm) for some non-constructor function symbol f and terms t1,..,tm. Since reduce(E0,En) succeeds there is a clause in PC:

    reduce(f(X1,..,Xm),Z):-Q U {reduce(RHS,Z)}

such that it is the compilation of a rule f(L1,..,Lm)=>RHS in P. Moreover, reduce(f(t1,..,tm),En) unifies with the head of the above clause with m.g.u. τ={<X1,t1>,..,<Xm,tm>,<Z,En>} and Qτ U {reduce(RHS,Z)}τ has a successful derivation of length n-1. Moreover, RHSτ=RHS and Zτ=En.

If Q is empty, m=0. So, by restriction (e) RHS is ground. By induction hypothesis there is a successful N-reduction RHS,..,En. E0 unifies with f(L1,..,Lm) and so E0=>RHS. Hence E0,RHS,..,En is a successful N-reduction.

Suppose Q is non-empty. Let Q1,..,Qm be the members of Q. Consider Qi. If Qi=(Xi=Li) then ti unifies with Li with substitution σi={<Li,ti>}. Construct the singleton sequence f(t1,..,ti,..,tm). This sequence is an N-reduction.

If Qi=reduce(Xi,Li) then Li=c(U1,..,Uk) for some constructor symbol c and variables U1,..,Uk. Also Qiτ=reduce(ti,Li). Clearly, reduce(ti,Li) succeeds. Let the answer substitution be σi. By Lemma 6 Liσi is ground. Then reduce(ti,Liσi) also succeeds. The successful derivation of reduce(ti,Liσi) is the same as that of reduce(ti,Li) with Li replaced by Liσi. Moreover, the length of this derivation is also less than n. By induction hypothesis, there is a successful N-reduction ti,..,Liσi. By Lemma 4, the sequence f(t1,..,ti,..,tm),..,f(t1,..,Liσi,..,tm) is an N-reduction.

Hence we obtain the N-reductions f(t1,..,tm),..,f(L1σ1,..,tm) and f(L1σ1,t2,..,tm),..,f(L1σ1,L2σ2,..,sm) and .. f(L1σ1,L2σ2,..,tm),..,f(L1σ1,L2σ2,..,Lmσm).

The concatenation of these reductions is itself an N-reduction. If m=1 this is clear. If m>1, assume assertion for m-1. That is, f(t1,..,tm),..,f(L1σ1,..,Lm-1σm-1,tm) is an N-reduction. If Lm is a variable, Lmσm=tm. Hence, the reduction is not extended. If Lm is not a variable, then if tm unifies with Lm, then again the reduction is not extended. Otherwise select(f(L1σ1,..,Lm-1σm-1,tm)=tm and the reduction f(t1,..,tm),..,f(L1σ1,..,Lm-1σm-1,tm),.., f(L1σ1,..,Lm-1σm-1,Lmσm) is also an N-reduction.

Since none of the Li share any variables, f(L1σ1,..,Lmσm) unifies with f(L1,..,Lm). Moreover, the m.g.u. is the union of σ1,..,σm. Let σ be this union. Hence f(L1σ1,..,Lmσm)=>RHSσ. Since all the variables of RHS are in L1,..,Lm and for each σi, Liσi is ground, RHSσ is ground.

The predication reduce(RHSσ,En) succeeds and the length of the associated successful derivation is less than n. By induction hypothesis, there is a successful N-reduction RHSσ,..,En. Hence there is a successful N-reduction f(t1,..,tn),..,f(L1σ1,..,Lmσm),RHSσ,..,En.
QED.

Theorem 3. The correctness of the compilation of F*. Let P be an F* program and PC be its compilation. Let E0 and En be ground terms. Then there is a successful N-reduction beginning with E0 and ending with En iff PC|-reduce(E0,En).

Proof: Lemmas 8 and 9 state, respectively, the if and only if parts of the theorem. By their proofs, we obtain the proof of the theorem.
QED.

Theorem 4. Simplification theorem. Let P be an F* program and PC its compilation. Let E0 and En be ground terms such that there is a successful N-reduction E0,..,En. Then reduce(E0,Z), Z a variable, succeeds in the presence of PC, with answer substitution {<Z,En>}.

Proof: Let En be a term such that there is a successful N-reduction E0,..,En. By Theorem 3, there is a successful SLD-derivation starting at reduce(E0,En). A simple induction on the length of this derivation establishes that reduce(E0,Z) succeeds with answer substitution {<Z,En>}. QED.

5.0 LAZY EVALUATION

The completeness property of the reduction strategy select enables it to exhibit a weak form of lazy evaluation. That is, it simplifies terms whenever it is possible to do so. In particular, it can simplify terms even if they contain subterms denoting infinite structures. For example, suppose we define in F*:

```
first(0,X)=>[].
first(s(A),[U|V])=>[U|first(A,V)].

intfrom(N)=>[N|intfrom(s(N))].
```

The first set of rules defines the function for computing an initial segment of a list whose length is some specified number. The second rule defines the function for computing the infinite list of integers starting at some integer.

The term intfrom(0) can be thought of as denoting the infinite list of integers 0,1,2,... However, select, if given the term first(s(s(0)),intfrom(0)), will simplify it to [0|first(s(0),intfrom(s(0)))]. If the above functions are defined in the usual way in say, Lisp, and the above term is evaluated, a non-terminating computation will occur.

To perform reductions in Prolog we first compile the above rules into Prolog:

```
reduce(0,0).
reduce(s(X),s(X)).

reduce([],[]).
reduce([U|V],[U|V]).

reduce(first(A1,A2),Out):-reduce(A1,0),A2=X,reduce([],Out).
reduce(first(A1,A2),Out):-reduce(A1,s(A)),reduce(A2,[U|V]),
                          reduce([U|first(A,V)],Out).

reduce(intfrom(A1),Out):-A1=N,reduce([N|intfrom(s(N))],Out).
```

Now the goal reduce(first(s(s(0)),intfrom(0)),Z) succeeds with Z=[0|first(s(0),intfrom(s(0)))]. Thus, Prolog also exhibits the above weak form of lazy evaluation, as intended.

6.0 FUTURE DIRECTIONS

In future, we intend to accomplish the following goals:

(a) Showing that if an F* program satisfies certain conditions, then select also satisfies the property of minimality. That is, it simplifies terms in a minimum number of steps.

(b) Investigating properties of an F* program satisfiying these conditions, e.g. a simple test for confluence.

(c) Precisely defining the notion of lazy evaluation. It appears there are two notions: a weak one which is a consequence of reduction-completeness, and a strong one which is a consequence of minimality.

(d) Implementing the above conditions in Prolog, so that Prolog also simplifies terms in a minimum number of steps.

(e) Identifying how to let the user specify which functions to be evaluated eagerly. These functions can be implemented as Prolog relations. For example, arithmetic could be done this way. The dominant mode of evaluation in usual languages is eager. Sometimes they allow the user to specify which functions are to be evaluated lazily. In F*, the situation would be reversed.

(f) Reasoning why the Prolog implementation of F* is "efficient", especially compared to previous approaches for combining rewrite rules and logic programming, or for realizing lazy evaluation.

(g) Identifying advantages of a combined functional/logic programming systems and lazy evaluation. One advantage is that we can efficiently simulate a special case of the following axioms of equality: x=y & q(x)->q(y). Another advantage is that we can compute with infinite structures.

ACKNOWLEDGEMENTS

I am very grateful to Professor D. Stott Parker for his many insightful comments on this report.

REFERENCES

Henderson, P. [1980]. Functional Programming: Application and Implementation. Prentice Hall International, New Jersey. 1980

Lloyd, J. [1984]. Foundations of logic programming. Springer Verlag, New York.

Narain, S. [1986]. A technique for doing lazy evaluation in logic. Journal of logic programming, vol. 3 no. 3. Elsevier North-Holland.

Vuillemin, J. [1974]. Correct and optimal implementations of recursion in a simple programming language. Journal of computer and systems sciences, 9, 332-354. Academic Press.