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 \(<X_1,t_1>, \ldots, <X_n,t_n>\) where each \(X_i\) is a variable and each \(t_i\) is a term. A variable \(X\) is defined in a substitution \(\sigma\) iff for some term \(s\), \(<X,s>\) occurs in \(\sigma\). Let \(\sigma\) be a substitution and \(E\) be a term, possibly containing variables. Then \(E\sigma\) represents the result of applying \(\sigma\) to \(E\).

**Reduction Rules.** A reduction rule is of the form:

\[
\text{LHS} \Rightarrow \text{RHS}
\]

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 \(c(t_1, \ldots, t_n)\) where \(c\) is an n-ary constructor symbol, \(n \geq 0\).

(c) If LHS = \(f(t_1, t_2, \ldots, t_n)\), \(n \geq 0\), each \(t_i\) is a variable or a term of the form \(c(X_1, \ldots, X_n)\) where \(c\) is an n-ary constructor symbol, \(n \geq 0\), and each \(X_i\) 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:

\[
\text{append([],X)} \Rightarrow X
\]
\[
\text{append([U|V],W) \Rightarrow [U|append(V,W)]}
\]

\[
\text{if(true,X,Y)} \Rightarrow X.
\]
\[
\text{if(false,X,Y)} \Rightarrow Y.
\]
\[
\text{not(true)} \Rightarrow \text{false}.
\]
\[
\text{not(false)} \Rightarrow \text{true}.
\]

\[
\text{lesseq(0,X)} \Rightarrow \text{true}.
\]
\[
\text{lesseq(s(X),s(Y))} \Rightarrow \text{lesseq(X,Y)}.
\]
\[
\text{lesseq(s(X),0)} \Rightarrow \text{false}.
\]
\[
\text{greater(X,Y)} \Rightarrow \text{not(lesseq(X,Y))}.
\]
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 =>_E1. Let P be an F* program and E and E1 be terms. We say E =>_PE1 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 =>_PE1 then, by restriction (e) E1 is also ground. If P is clear from the context we write E =>_PE1 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 =>_H E. Let E1 be the result of substituting H for G in E. Then we say E =>_PE1. Note that if E =>_PE1 then E unifies with the left hand side of some rule in P. If E =>_PE1 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 =>_PE1 or E =>_p 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 =>_PEi+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\geq 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\geq 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 $\Rightarrow$ rule in P apply to the whole of $E$. Where $f(t_1, \ldots, t_n)$ is a term, $n \geq 0$ the relation $select_P$ is:

\[
select_P(f(t_1, \ldots, t_n), f(t_1, \ldots, t_n)) \text{ if } f(t_1, \ldots, t_n) \Rightarrow_P X.
\]

\[
select_P(f(t_1, \ldots, t_i, \ldots, t_n), X) \text{ if }
\]

\[
\begin{align*}
\text{there is a rule } f(L_1, \ldots, L_i, \ldots, L_n) \Rightarrow \text{RHS in P, and} \\
\text{ti does not unify with Li, and} \\
select_P(t_i, X).
\end{align*}
\]

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 $\Rightarrow$ 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 $\hat{c}(t_1, \ldots, t_n) \Rightarrow \text{RHS}$, 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 $E = [1]\{1\}$ 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 $E_i$ be the result of replacing G by $H$ in $E$. Then we say that $E$ reduces to $E_i$ 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 $E_1, E_2, \ldots$ in P such that for each $i$ when $E_i$ and $E_{i+1}$ both exist, $E_i$ reduces to $E_{i+1}$ 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). Similarly, f(N₁,..,Nₖ)σ=f(N₁σ,..,Nₖσ), 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. ti=<{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 \(<X_l,s_l>,..,<X_m,s_m>\) appear in \(\sigma\).

If \(t_i\) is identical with \(t_i^*\), \(t_i^*\) also unifies with \(L_i\) with m.g.u. \(t_i = \langle X_l, s_l >,..,\langle X_m, s_m >\). Of course, for every \(i\), \(R(G,H,s_i,s_i)\).

If \(t_i\) is not identical with \(t_i^*\) then since \(R(G,H,t_i,t_i^*)\), \(t_i\) contains at least one occurrence of \(G\) and \(G=\geq H\). Since \(t_i = c(s_1,..,s_m)\), \(c\) a constructor symbol, by restriction (b) \(t_i=\geq G\). Hence \(t_i^* = c(s_1^*,..,s_m^*)\) each \(s_i^*\) a term and for every \(i\) \(R(G,H,s_i,s_i^*)\). Hence \(t_i^*\) unifies with \(L_i\) with m.g.u. \(t_i = \langle X_l, s_l^* >,..,\langle X_m, s_m^* >\).

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

Since \(\tau\) is an m.g.u. of \(P_l\) 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 \(\tau\). Otherwise \(B\) would contain \(X\). Thus \(\tau\) defines only those variables which occur in \(B\). Similarly for \(\sigma\). Thus \(\sigma\) and \(\tau\) define exactly the same variables. Thus (2).

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

**Lemma 4.** Let \(P\) be an \(F^*\) program. If:
1. \(f(t_1,..,t_i,..,t_n)\) is a term, and
2. \(f(L_1,..,L_{i-1},c(X_1,..,X_m),L_{i+1},..,L_n)\Rightarrow RHS\) is a rule in \(P\), and
3. \(t_i=d_1,d_2,d_3,..,d_r, r>0, \) is an \(N\)-reduction.

Then:
\(f(t_1,..,t_{i-1},d_1,t_i+1,..,t_n), f(t_1,..,t_{i-1},d_2,t_i+1,..,t_n),..,\)
\(f(t_1,..,t_{i-1},d_r,t_i+1,..,t_n)\) is also an \(N\)-reduction.

**Proof.** Let \(L_i=c(X_1,..,X_m)\). Since \(f(L_1,..,L_{i-1},L_n)\Rightarrow RHS\) 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 \(d_1,d_2,d_3,..,d_r\) can be in simplified form. Hence, since \(L_i=c(X_1,..,X_m)\), none of the \(d_i\), \(0<i<r\) unify with \(L_i\).

We now show that for all \(j\), \(0<j<r\), \(f(t_1,..,t_{i-1},d_j,t_i+1,..,t_n)\) reduces to \(f(t_1,..,t_{i-1},d_j+1,t_i+1,..,t_n)\) in an \(N\)-step. Since \(d_j\) is not simplified, it does not unify with \(L_i\). Hence, by definition of select, for every \(X\) select \(f(t_1,..,t_{i-1},d_j,t_i+1,..,t_n), X\) if select \(d_j,X\).

Since \(d_j\) reduces to \(d_j+1\) in an \(N\)-step there are terms \(p_j\) and \(q_j\) such that select \(p_j(d_j,p_j), p_j\Rightarrow q_j\) and \(d_j+1\) is the result of replacing \(p_j\) by \(q_j\) in \(d_j\). Then \(f(t_1,..,t_{i-1},d_j,t_i+1,..,t_n)\) reduces to \(f(t_1,..,t_{i-1},d_j+1,t_i+1,..,t_n)\) in an \(N\)-step.
Hence, \( f(t_1, \ldots, t_{i-1}, d_1, t_i+1, \ldots, t_n) \), \( f(t_1, \ldots, t_{i-1}, d_2, t_i+1, \ldots, t_n) \), \( \ldots \), \( f(t_1, \ldots, t_{i-1}, d_r, t_i+1, \ldots, t_n) \) is an N-reduction. QED.

Theorem 1.

Let \( P \) be an \( F^* \) program. Let \( E_1, F_1, F_2, G, H \) be terms such that
(1) \( E_1 \) is not simplified, and
(2) \( R(G, H, E_1, F_1) \), and
(3) \( F_1 \) reduces to \( F_2 \) in an N-step

Then there is an N-reduction \( E_1, \ldots, E_2 \) in \( P \) such that \( R(G, H, E_2, F_2) \).

Proof. It is helpful to draw the following diagram:

\[
\begin{array}{c}
E_1 & F_1 & G \text{ occurs 0 or more times in } E_1 \\
\mid & \mid & G \Rightarrow H \\
* & \text{N-step} & \\
\mid & \mid & \\
v & v & \\
E_2 & F_2
\end{array}
\]

We have to show that \( R(G, H, E_2, F_2) \).

We proceed by induction on length of \( E_1 \). 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.

\[
\begin{array}{c}
E_1=0\text{-ary fn symbol?} \\
/ \ \\no, \ E_1=f(t_1, \ldots, t_n) \\
/ \\
E_1=F_1? \\
/ \\
yes \no \\
/ \\
\begin{array}{c}
= \ E_1=G? \\
/ \\
yes \no \\
/ \\
F_1 \text{ reduces to } F_2 \text{ by some rule } R \text{ in } P? \\
/ \\
yes \no \\
/ \\
E_1 \text{ reduces to } E_2 \text{ by } R? \\
+ \\
/ \\
yes \no \\
+ \\
+
\end{array}
\end{array}
\]

Suppose \( E_1 \) is a 0-ary function symbol. If \( E_1=F_1 \) then \( E_1, F_2 \) is an
N-reduction and \( R(G,H,F_2,F_2) \). If \( E_1=\neq F_1 \) then since \( R(G,H,E_1,F_1) \) \( G \) must occur in \( E_1 \), and \( F_1 \) is the result of replacing \( G \) in \( E_1 \) by \( H \). So, \( E_1=G \) and \( E_1=\Rightarrow F_1 \), there is an \( N \)-reduction \( E_1,F_1,F_2 \) and \( R(G,H,F_2,F_2) \). That is, putting \( E_2=F_2 \) satisfies the theorem.

Otherwise, since we are given that \( E_1 \) is not simplified, \( E_1=\neg f(t_1,\ldots,t_n) \), \( n\geq 0 \), \( f \) not a constructor symbol. Assume the theorem for every term whose length is less than that of \( f(t_1,\ldots,t_n) \).

If \( E_1=F_1 \) then \( E_1,F_2 \) is an \( N \)-reduction and \( R(G,H,F_2,F_2) \), so putting \( E_2=F_2 \) satisfies the theorem. Otherwise \( E_1=\neq F_1 \). If \( \neg G \) then since \( R(G,H,E_1,F_1) \), \( E_1=\Rightarrow F_1 \), and \( E_1,F_1,F_2 \) is an \( N \)-reduction and \( R(G,H,E_2,F_2) \). Again, that is, putting \( E_2=F_2 \) satisfies the theorem.

Having considered the easy cases, we arrive at the interesting cases, with \( E_1=\neq F_1 \) and \( G \) occurs in \( E_1 \) but \( G=\neq E_1 \). Hence \( F_1=\neg f(t_1^*,\ldots,t_n^*) \) where for every \( i \), \( R(G,H,t_i,t_i^*) \). We now consider the following subcases:

1. \( F_1=\Rightarrow F_2 \). Then there is a rule \( f(L_1,\ldots,L_n)=\neg RHS \) in \( P \) such that \( F_1 \) and \( f(L_1,\ldots,L_n) \) unify with m.g.u. \( \tau \) and \( F_2=RHS^\tau \).

1-1. \( E_1 \) and \( f(L_1,\ldots,L_n) \) do unify. Let the m.g.u. be \( \sigma \). By Lemma 3, \( F_1 \) and \( f(L_1,\ldots,L_n) \) also unify with some m.g.u. \( \beta \). Since \( F_1 \) already unifies with \( f(L_1,\ldots,L_n) \) with m.g.u. \( \tau \), \( \tau=\beta \).

\( E_1=RHS^\sigma \) and so \( E_2=RHS^\sigma \). The \( N \)-reduction is \( E_1,E_2 \). Of course \( F_2=RHS^\tau \). By Lemma 3, \( \sigma \) and \( \tau \) define exactly the same variables, and if \( <X,s> \) occurs in \( \sigma \) and \( <X,s^*> \) appears in \( \tau \) then \( R(G,H,s,s^*) \). Hence, by Lemma 2, \( R(G,H,E_2,F_2) \).

1-2. \( E_1 \) and \( f(L_1,\ldots,L_n) \) do not unify. Then, since \( E_1 \) is ground and each variable occurs at most once in \( f(L_1,\ldots,L_n) \), there is at least one \( L_i \) in \( L_1,\ldots,L_n \) such that \( ti \) does not unify with \( L_i \). Hence \( L_i \) is not a variable and so \( L_i=c(X_1,\ldots,X_m) \), \( c \) a constructor symbol and each \( X_i \) a variable.

Since \( R(G,H,t_i,t_i^*) \), \( ti \) does not unify with \( L_i \), \( ti \) is not simplified. Suppose \( ti \) were simplified. Either \( ti=c(s_1,\ldots,s_m) \), each \( s_i \) a term. But then \( ti \) must unify with \( L_i \). Contradiction. Or, \( ti=d(s_1,\ldots,s_m) \), \( d \) a constructor symbol, \( d=\neq c \), each \( s_i \) a term. Since \( R(G,H,t_i,t_i^*) \), \( ti=G \) and \( ti^*=H \). By restriction (b) this is impossible.

Since \( F_1 \) unifies with \( f(L_1,\ldots,L_n) \), \( ti^* \) unifies with \( L_i \), and so \( ti^* \) is simplified. Since \( ti \) is not simplified, and \( R(G,H,t_i,t_i^*) \), \( ti=ti^* \). Thus select \( E_1,ti \). Hence \( f(t_1,\ldots,ti,\ldots,t_n) \) reduces to \( f(t_1,\ldots,ti^*,\ldots,t_n) \) in an \( N \)-step.

Hence there exists an \( N \)-reduction \( E_1=P_1,P_2,P_3,\ldots \) where for each \( i \), \( P_i=f(s_1,\ldots,s_n) \), \( sk=tk \) or \( sk=tk^* \), and if \( sk \) does not unify with \( L_k \) then \( P_i+1 \) is derived from \( P_i \) by replacing \( sk \) by \( sk^* \) such that \( sk^*=tk^* \). We also have for each \( i \), \( R(G,H,P_i,F_1) \). This reduction cannot be infinite.
since \( F_1 = f(t_{1*}, \ldots, t_{n*}) \) unifies with \( f(l_1, \ldots, l_n) \). Let the last term be Pm. Let the m.g.u. of Pm and \( f(l_1, \ldots, l_n) \) be \( \sigma \). Then \( Pm \Rightarrow RHS\sigma \). Hence we have the N-reduction \( E_1, P_2, F_3, \ldots, P_m, RHS\sigma \). By Lemma 3, there is an m.g.u. of F1 and \( f(l_1, \ldots, l_n) \) and clearly this is \( \tau \). Already, \( F_2 = RHS\tau \). By Lemma 2, \( R(G, H, RHS\sigma, F_2) \).

2. There is no \( F_2 \) such that \( F_1 \Rightarrow F_2 \), i.e. \( \text{select}(F_1, F_1) \) is not true. Or, \( F_1 \) does not unify with the head of any reduction rule in \( P \). Hence \( \text{select}(E_1, E_1) \) is not true. If it were, there would be a contradiction with Lemma 1. We are given that \( F_1 \) reduces to \( F_2 \) by an N-step. We now have to show that there is an N-reduction \( E_1, \ldots, E_2 \) such that \( R(G, H, E_2, F_2) \).

Suppose \( \text{select}(F_1, u) \). Then \( u \) occurs in some \( t_{i*} \). That is, there is some \( i \) such that \( \text{select}(t_{i*}, u) \). Let \( u \Rightarrow v \) and let \( t_{i**} \) be the result of replacing \( u \) in \( t_{i*} \) by \( v \). Hence \( t_{i*} \) reduces to \( t_{i**} \) in an N-step, and also \( F_2 = f(t_{1*}, \ldots, t_{i**}, \ldots, t_{n*}) \). By definition of select, there is a rule \( f(l_1, \ldots, l_i, \ldots, l_n) \Rightarrow RHS \) in \( P \) such that \( t_{i*} \) does not unify with \( l_i \). Hence \( L_i = c(x_1, \ldots, x_m) \), \( m \geq 0 \), where \( c \) is a constructor symbol and each \( x_i \) is a variable.

Clearly, \( t_{i*} \) is not simplified. So, by restriction (b) \( t_{i*} \) is also not simplified. \( t_{i*} \) reduces to \( t_{i**} \) in an N-step. We already have \( R(G, H, t_{i*}, t_{i**}) \). Since the length of \( t_{i*} \) is less than \( f(t_{1}, \ldots, t_{i}, \ldots, t_{n}) \), by induction hypothesis there is an N-reduction \( t_i = d_1, d_2, \ldots, d_r, r > 1 \), such that \( R(G, H, d_r, t_{i**}) \). By Lemma 4, the sequence \( f(t_{1}, \ldots, t_{i-1}, t_{i}, t_{i+1}, \ldots, t_{n}) \), \( f(t_{1}, \ldots, t_{i-1}, d_2, t_{i+1}, \ldots, t_{n}) \), \( f(t_{1}, \ldots, t_{i-1}, d_1, t_{i+1}, \ldots, t_{n}) \) is an N-reduction. We already have \( F_2 = f(t_{1*}, \ldots, t_{i**}, \ldots, t_{n*}) \) and for each \( k \) \( R(G, H, t_k, t_k) \). Hence \( R(G, H, f(t_{1}, \ldots, t_{i-1}, d_1, t_{i+1}, \ldots, t_{n}), f(t_{1}, \ldots, t_{i-1}*, t_{i**}, t_{i+1}*, \ldots, t_{n}) \).

QED

**Theorem 2. The reduction-completeness of \( F^* \)**

Let \( P \) be an \( F^* \) program and \( D_0 \) be a ground term. Let \( D_0, D_1, \ldots, D_n = c(t_1, \ldots, t_x) \), be a successful reduction in \( P \) for some ground terms \( D_1, \ldots, D_n \) and \( t_1, \ldots, t_x \) and some constructor symbol \( c \). Then there is a successful N-reduction \( D_0, Q_1, \ldots, Q_p = c(s_1, \ldots, s_x) \) in \( P \) for some ground terms \( Q_1, \ldots, Q_p \) and \( s_1, \ldots, s_x \).

**Proof.** By induction on length \( n \) of successful reduction \( D_0, D_1, \ldots, D_n \).

If \( n = 0 \), then \( D_0 \) is already simplified and \( D_0 \) is the successful reduction and \( D_0 = c(t_1, \ldots, t_x) \).

If \( n = 1 \) then \( D_0 \Rightarrow D_1 \). Hence, \( \text{select}(D_0, D_0) \), so we have the successful N-reduction \( D_0, Q_1 = D_1 \). Again, \( Q_1 = c(t_1, \ldots, t_x) \).

Assume the theorem holds for \( n = k - 1 \) i.e. for the successful reduction \( D_1, \ldots, D_k = c(t_1, \ldots, t_x) \). We now show that it holds for the successful reduction \( D_0, D_1, \ldots, D_k \). By induction hypothesis, \( D_1 \) has a successful
N-reduction, say \( D_1, F_2, F_3, F_4, \ldots, F_m = c(p_1, \ldots, p_x) \). Of course, all terms in this sequence, except \( F_m \), are unsimplified.

Since \( D_0 \rightarrow D_1 \), there are terms \( G, H \) such that \( G \) occurs in \( D_0 \), \( G \rightarrow H \) and \( D_1 \) is the result of replacing \( G \) by \( H \) in \( D_0 \). Hence \( R(G, H, D_0, D_1) \).

Hence by theorem 1 there is an N-reduction \( D_0, \ldots, E_2 \) such that \( R(G, H, E_2, F_2) \). If \( F_2 \) is not simplified, then since \( R(G, H, E_2, F_2) \), by restriction (b) \( E_2 \) is also not simplified.

By repeatedly applying theorem 1 we have the N-reductions \( D_0, \ldots, E_2, \ldots, E_3, \ldots \) and \( E_{m-1}, \ldots, E_m \) for some finite \( m \geq 2 \), such that for each \( i, 2 \leq i \leq m \), \( R(G, H, E_i, F_i) \) and at most \( E_m \) is simplified. The resulting situation can be laid out in the following diagram:

```
    D0    D1    R(G, H, D0, D1)
    |      |      |
    *      |      |
    v      v      v
    E2    F2    R(G, H, E2, F2)
    |      |      |
    *      |      |
    v      v      v
    E3    F3    R(G, H, E3, F3)
    ... ...
    E_{m-1} F_{m-1} R(G, H, E_{m-1}, F_{m-1})
    |      |      |
    *      |      |
    v      v      v
    E_{m} F_{m} R(G, H, E_{m}, F_{m})
```

Since at most \( E_m \) is simplified let \( S \) be the reduction \( D_0, \ldots, E_2, \ldots, E_3, \ldots, E_{m-1}, \ldots, E_{m} \). Clearly, \( S \) is an N-reduction.

If \( E_m \) is simplified then since \( R(G, H, E_m, F_m) \) and \( F_m = c(p_1, \ldots, p_x) \), \( E_m = c(s_1, \ldots, s_x) \) and for each \( i \), \( R(G, H, s_i, t_i) \). Then, \( S \) is the required N-reduction.

Otherwise, since \( F_m \) is simplified, \( R(G, H, E_m, F_m) \), \( G \rightarrow H \), we have \( E_m = G \) and \( F_m = H \), i.e. \( E_m \rightarrow F_m \). Hence, select \( (E_m, E_m) \), and so we have the N-step \( E_m, F_m \). The required N-reduction is then \( S, F_m \) which is \( S, c(p_1, \ldots, p_x) \). 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:

\[ \text{reduce}(c(X_1, \ldots, X_n), c(X_1, \ldots, X_n)) \]

Stage 2. Let \( f(t_1, \ldots, t_n) \Rightarrow \text{RHS} \) be a rule in P where \( f \) is an n-ary, \( n \geq 0 \), non-constructor function symbol and each of RHS and \( t_1, \ldots, t_n \) is a term, possibly containing variables. For each such rule perform the following steps:

(a) Let \( A_1, \ldots, A_n \) be n distinct Prolog variables none of which occur in the rule. If \( t_i \) is a variable generate the predication \( A_i=t_i \). If \( t_i \) is \( c(X_1, \ldots, X_n) \) where \( c \) is a constructor symbol and each \( X_i \) a variable, generate the predication \( \text{reduce}(A_i, c(X_1, \ldots, X_n)) \). Let \( \text{LHS\_CONDS} \) be the set of predications so generated.

(b) Let Out be a Prolog variable not occurring in the rule and different from \( A_1, \ldots, A_n \). Generate the predication \( \text{reduce}(\text{RHS}, \text{Out}) \).

(c) Generate the clause

\[ \text{reduce}(f(A_1, \ldots, A_n), \text{Out}) :\neg \text{LHS\_CONDS} \cup \{\text{reduce}(\text{RHS}, \text{Out})\} \]

For example, the F* rules:

\begin{align*}
\text{append}([], X) & \Rightarrow X \\
\text{append}([U|V], W) & \Rightarrow [U|\text{append}(V, W)]
\end{align*}

are compiled into:

\begin{align*}
\text{reduce}([], []). \\
\text{reduce}([U|V], [U|V]).
\end{align*}

\begin{align*}
\text{reduce}(\text{append}(A_1, A_2), \text{Out}) :\neg \text{reduce}(A_1, []), A_2=\text{X}, \text{reduce}(\text{X}, \text{Out}). \\
\text{reduce}(\text{append}(A_1, A_2), \text{Out}) :\neg \text{reduce}(A_1, [U|V]), A_2=W, \text{reduce}([U|\text{append}(V, W)], \text{Out}).
\end{align*}

4.2 Correctness of translation of F*

Lemma 5. Let P be an F* program. If:

1. \( E_0=f(t_1, \ldots, t_i, \ldots, t_m), \) and
2. \( E_k=f(s_1, \ldots, s_i, \ldots, s_m), \) and
3. si is simplified, and
4. \( E_0, \ldots, E_k, k \geq 0, \) is an N-reduction such that for no \( i \) \( E_i \Rightarrow E_{i+1} \).

Then there is a successful N-reduction \( t_i, \ldots, s_i \) of length less than or
equal to the length \( k \) of \( E_0, E_1, \ldots, E_k \).

**Proof:** By induction on length of \( N \)-reduction \( E_0, \ldots, E_k \). Suppose \( k = 0 \). Since \( E_0 = f(t_1, \ldots, t_i, \ldots, t_m) \), \( t_i = s_i \). The successful \( N \)-reduction is simply \( t_i \) whose length is 0.

Suppose \( k > 0 \). Assume that the lemma holds for all \( N \)-reductions of length \( k-1 \). Consider the \( N \)-reduction \( E_1, \ldots, E_k \), \( k > 1 \), of length \( k-1 \). Since there is no \( i \) such that \( E_i \rightarrow E_i+1 \), \( E_i = f(u_1, \ldots, u_i, \ldots, u_m) \) for terms \( u_1, \ldots, u_m \). By induction hypothesis, there is an \( N \)-reduction \( u_i, \ldots, s_i \), whose length is less than or equal to \( k-1 \).

If \( t_i = u_i \) then there is a successful \( N \)-reduction \( t_i, \ldots, s_i \), whose length is less than or equal to \( k-1 \) and so less than or equal to \( k \).

If \( t_i \neq u_i \) then since \( E_0 \) reduces to \( E_1 \) by an \( N \)-step, by definition of select, \( t_i \) reduces to \( u_i \) in an \( N \)-step. We now have the successful \( N \)-reduction \( t_i, u_i, \ldots, s_i \) 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 \( \text{reduce}(A, B) \) succeeds with answer substitution \( \sigma \). Then \( B\sigma \) is ground.

**Proof:** By induction on length \( n \) of successful \( SLD \)-derivation \( \text{reduce}(A, B), G_1, \ldots, G_n = \varnothing \). If \( n = 1 \) then \( A = c(t_1, \ldots, t_m) \), \( c \) a constructor symbol each \( t_i \) a term, \( m \geq 0 \). The query \( \text{reduce}(A, B) \) will succeed by matching the head of the clause \( \text{reduce}(c(X_1, \ldots, X_m), c(X_1, \ldots, X_m)) \). The answer substitution \( \sigma \) will be such that \( B\sigma = A \). Clearly \( B\sigma \) is ground.

Assume lemma for successful \( SLD \)-derivations of length less than \( n \). Let the successful derivation starting at \( \text{reduce}(A, B) \) be of length \( n > 1 \). Then \( A = f(t_1, \ldots, t_m) \), \( m \geq 0 \), where \( f \) is a function symbol, but not a constructor symbol, and each \( t_i \) is a ground term. Then there is a clause:

\[
\text{reduce}(f(X_1, \ldots, X_m), Z) :- Q, \text{reduce}(RHS, Z).
\]

such that (a) this clause is the compilation of a rule \( f(L_1, \ldots, L_m) \rightarrow RHS \) (b) each of \( X_1, \ldots, X_m, Z \) is a distinct variable not appearing in \( f(L_1, \ldots, L_m) \rightarrow RHS \) (c) if \( L_i \) is a variable then \( X_i = L_i \) appears in \( Q \). Otherwise \( \text{reduce}(X_i, L_i) \) appears in \( Q \) (d) \( \text{reduce}(f(t_1, \ldots, t_m), B) \) unifies with the head of this clause with some m.g.u. \( \tau \) and its immediate descendant \( (Q, \text{reduce}(RHS, Z)) \tau \) has a successful \( SLD \)-derivation of length \( n-1 \). Clearly, \( \tau = \{ <X_1, t_1>, \ldots, <X_m, t_m>, <Z, B> \} \) and so \( Z\tau = B \). Also, since \( RHS \) does not contain any of the \( X_i \), \( RHSt = RHS \).

If \( Q \) is empty then \( m = 0 \), so, by restriction (e) \( RHSt \) is ground. By induction hypothesis, \( \text{reduce}(RHSt, B) \) succeeds with answer substitution \( \sigma \) such that \( B\sigma \) is ground. So, \( \text{reduce}(A, B) \) succeeds with answer substitution \( \sigma \) such that \( B\sigma \) is ground.
Assume Q is non-empty. Let Q1,..,Qm be the members of Q. If Qi is Xi=Li
then Qi= (ti=Li) and succeeds with answer substitution \( \sigma_i = \{ <Li,ti> \} \).
If Qi is reduce(Xi,Li) then Qi= reduce(ti,Li) and has a successful
SLD-derivation of length less than or equal to n-1. Hence, by induction
hypothesis, Qi succeeds with answer substitution \( \sigma_i \) such that
Lid is ground.

By restriction (e) all variables of RHS occur in L1,..,Lm. Hence, since
each Lid is ground, RHS_\sigma_1,..,\sigma_m is ground. Already Zt=B.
Since B does not contain any variables in L1,..,Lm, B\sigma_1,..,\sigma_m=B.
Hence reduce(RHS, Z)\sigma_1,..,\sigma_m = reduce(RHS_\sigma_1,..,\sigma_m, B).
By
induction hypothesis, this succeeds with answer substitution \( \sigma \) such
that B\sigma is ground. So, reduce(A, B) succeeds with answer substitution
\( \sigma \) such that B\sigma 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 \( \alpha \),
D\sigma_\alpha=B. Then reduce(A, D) succeeds with answer substitution \( \alpha \).

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 \( \alpha \).

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:

\[
\text{reduce}(f(X1,..,Xm),Z) :- \text{QU}\{\text{reduce}(\text{RHS},Z)\}
\]

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.
t=\{X1,t1,..,Xn,tn,<Z,B>\} and its immediate descendant is
(QU\{reduce(\text{RHS},Z)\})t. Since RHS does not contain any of the X1, this
is QU\{reduce(\text{RHS},B)\}. It has a successful derivation of length n-1.

If Q is empty, by restriction (e) RHS\sigma is ground. Otherwise let
Q1,..,Qm be the members of Q. Consider some Qi. If Qi is Xi=Li, then
Qi= (ti=Li) which succeeds with answer substitution \( \sigma_i = \{ <Li,ti> \} \).
Otherwise Qi= reduce(Xi,Li), so Qi= reduce(ti,Li). By Lemma 6
reduce(ti,Li) succeeds with answer substitution \( \sigma_i \) such that Lid is
ground. Since all the variables of RHS are in L1,..,Lm, RHS_\sigma_1,..,\sigma_m
is again ground.

Since reduce(RHS_\sigma_1,..,\sigma_m, B) succeeds, by induction hypothesis,
reduce(RHS_\sigma_1,..,\sigma_m, D) succeeds with answer substitution \( \alpha \). Now
consider the query reduce(A, D). Again, by reasoning as above,
reduce(RHS(\ast s1..\sigma_m, D) appears in an SLD-derivation of reduce(A, D).
Hence reduce(A,D) also succeeds with answer substitution $\alpha$. QED.

**Lemma 8.** Let $P$ be an $F^*$ program. Let $PC$ be the compiled version of $P$. Let $E_0,\ldots,E_n$ be a successful $N$-reduction. Then reduce$(E_0,E_n)$ succeeds (in the sense of SLD-resolution) in the presence of $PC$.

**Plan of Proof:** By induction on length of successful $N$-reduction $E_0,\ldots,E_n$. We show that there is some $E_j$ in $E_0,\ldots,E_n$ such that an SLD-derivation of reduce$(E_0,E_n)$ contains the goal reduce$(E_j,E_n)$. Since $E_j,\ldots,E_n$ is also a successful $N$-reduction, by induction hypothesis, reduce$(E_j,E_n)$ succeeds. Hence reduce$(E_0,E_n)$ succeeds.

**Proof:** By induction on length $n$ of successful reduction $E_0,\ldots,E_n$.
If $n=0$ then $E_0$ is already simplified. In particular, $E_0=c(t_1,\ldots,t_m)$ where $c$ is an $m$-ary constructor symbol, $m>0$, and $t_1,\ldots,t_m$ are terms. There is a clause in $PC$ reduce$(c(X_1,\ldots,X_m),c(X_1,\ldots,X_m))$ where each $X_i$ is a variable. Clearly reduce$(E_0,E_0)$ succeeds.

Let $n>0$ and $E_0=f(t_1,\ldots,t_m)$, $f$ not a constructor symbol, each $t_i$ a term and $m>0$. Assume theorem holds for all successful reductions of length less than $n$.

Since $E_0$ is not simplified, the $N$-reduction is of the form $E_0,\ldots,E_{k-1},E_k,\ldots,E_n$, $0<k<n$, such that $E_{k-1}\rightarrow E_k$, but for each $i$, $0<i<k-1$, not$(E_i\rightarrow E_{i+1})$. Hence, $E_{k-1}=f(s_1,\ldots,s_m)$ for some terms $s_1,\ldots,s_m$. Since $E_{k-1}\rightarrow E_k$, there is some rule $f(L_1,\ldots,L_m)\rightarrow$ RHS such that $E_{k-1}$ unifies with $f(L_1,\ldots,L_m)$ with m.g.u. $\sigma$ and $E_k=$ RHS$\sigma$. Since none of the $L_i$ share any variables $\sigma$ is the union of $\sigma_1,\ldots,\sigma_m$ such that $L_i$ and $s_i$ unify with m.g.u. $\sigma_i$.

For each $i$, if $L_i$ is not a variable, then since $L_i$ and $s_i$ unify, $s_i$ is in simplified form. For such $i$, there is, by Lemma 5, a successful $N$-reduction $t_i,\ldots,s_i$ of length less than or equal to $k-1$.

The rule $f(L_1,\ldots,L_m)\rightarrow$ RHS is compiled into the Horn clause

$$\text{reduce}(f(X_1,\ldots,X_m),Z) :- \text{QU}(\text{reduce}(\text{RHS},Z))$$

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

Consider the query reduce$(E_0,E_n)$, i.e. reduce$(f(t_1,\ldots,t_n),E_n)$. It unifies with reduce$(f(X_1,\ldots,X_m),E_n)$ with m.g.u. $\tau=\{<X_1,t_1>,\ldots,<X_n,t_n>,<Z,E_n>\}$ and its immediate descendant is $\text{QU}(\text{reduce}(\text{RHS},Z))\tau$. Since RHS does not contain any of the $X_i$, this is $\text{QU}(\text{reduce}(\text{RHS},E_n))$.

Let $Q_1,\ldots,Q_m$ be the members of $Q$. Consider some $Q_i$. If $Q_i$ is $X_i=L_i$, then $Q_i\tau=(t_i=L_i)$ which succeeds with answer substitution $\sigma_i=\{<L_i,t_i>\}$.

Otherwise $Q_i=$ reduce$(X_i,L_i)$, so $Q_i\tau=$ reduce$(t_i,L_i)$. Since there is a successful $N$-reduction $t_i,\ldots,s_i$ of length less than or equal to $k-1$, by
induction hypothesis, reduce(t_i,s_i) succeeds. Since Liσ_i=s_i, by Lemma 7 reduce(t_i,L_i) also succeeds with answer substitution σ_i.

By repeating the same argument for each Q_i, we see that an SLD-derivation starting at reduce(E_0,En) contains reduce(RHS_0,...,n,En) as a member. Since σ is the union of σ_i and no variable is defined in more than one σ_i, RHS_0,...,n= RHS_0. But RHS_0=E_k. Hence the SLD-derivation starting at reduce(E_0,En) contains reduce(E_k,En). Since the length of the successful reduction E_k,...,En is less than n, by induction hypothesis, reduce(E_k,En) succeeds. Thus, the query reduce(E_0,En) succeeds. QED.

**Lemma 9.** Let P be an F* program. Let PC be the compiled version of P. Let E_0 and En be terms such that reduce(E_0,En) succeeds (in the sense of SLD-resolution) in the presence of PC. Then there is a successful N-reduction E_0,...,En.

**Plan of Proof:** By induction on length of successful SLD-derivation reduce(E_0,En),...,,1. We show that there is some goal reduce(E_j,En) in this derivation such that there is an N-reduction E_0,...,E_j. Since reduce(E_j,En) succeeds, by induction hypothesis, there is a successful N-reduction E_j,...,En. So there is a successful N-reduction E_0,...,E_j,...,En.

**Proof:** By induction on length n of successful SLD-derivation starting at reduce(E_0,En). If n=1 then there is a clause reduce(c(X_1,...,X_m),c(X_1,...,X_m)) in PC such that reduce(E_0,En) unifies with the head of this clause. Clearly, then, E_0=En, En is simplified and the required N-reduction is simply E_0.

Let n>0. Assume lemma for all successful derivations of length less than n. Assume E_0=f(t_1,...,t_m) for some non-constructor function symbol f and terms t_1,...,t_m. Since reduce(E_0,En) succeeds there is a clause in PC:

\[
\begin{align*}
\text{reduce}(f(X_1,...,X_m),Z) &:= Q \cup \{\text{reduce}(\text{RHS},Z)\}
\end{align*}
\]

such that it is the compilation of a rule f(L_1,...,L_m)->RHS in P. Moreover, reduce(f(t_1,...,t_m),En) unifies with the head of the above clause with m.g.u. τ=(<X_1,t_1>,...,<X_m,t_m>,<Z,En>) and Qτ U {reduce(\text{RHS},Z)}τ has a successful derivation of length n-1. Moreover, RHS=\text{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. E_0 unifies with f(L_1,...,L_m) and so E_0->RHS. Hence E_0,RHS,...,En is a successful N-reduction.

Suppose Q is non-empty. Let Q_1,...,Q_m be the members of Q. Consider Q_i. If Q_i=(X_i=L_i) then t_i unifies with L_i with substitution σ_i=(<L_i,t_i>). Construct the singleton sequence f(t_1,...,t_i,...,t_m). This sequence is an N-reduction.
If \( Q_i \)-reduce(\( X_i, L_i \)) then \( L_i \)-c(\( U_1, \ldots, U_k \)) for some constructor symbol \( c \) and variables \( U_1, \ldots, U_k \). Also \( Q_i \)-reduce(\( t_i, L_i \)). Clearly, reduce(\( t_i, L_i \)) succeeds. Let the answer substitution be \( \sigma_i \). By Lemma 6 \( L_i \sigma_i \) is ground. Then reduce(\( t_i, L_i \sigma_i \)) also succeeds. The successful derivation of reduce(\( t_i, L_i \sigma_i \)) is the same as that of reduce(\( t_i, L_i \)) with \( L_i \) replaced by \( L_i \sigma_i \). Moreover, the length of this derivation is also less than \( n \). By induction hypothesis, there is a successful N-reduction \( t_i, \ldots, L_i \sigma_i \). By Lemma 4, the sequence \( f(t_1, \ldots, t_i, \ldots, t_m), \ldots, f(t_1, \ldots, L_i \sigma_i, \ldots, t_m) \) is an N-reduction.

Hence we obtain the N-reductions \( f(t_1, \ldots, t_m), \ldots, f(L_i \sigma_1, \ldots, t_m) \) and \( f(L_i \sigma_1, t_2, \ldots, t_m), \ldots, f(L_i \sigma_1, L_i \sigma_2, \ldots, s_m) \) and \( f(L_i \sigma_1, L_i \sigma_2, \ldots, t_m), \ldots, f(L_i \sigma_1, L_i \sigma_2, \ldots, L_i \sigma_m) \).

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, \( f(t_1, \ldots, t_m), \ldots, f(L_i \sigma_1, \ldots, L_i \sigma_m-1, t_m) \) is an N-reduction. If \( L_i \sigma_m \) is a variable, \( L_i \sigma_m=t_m \). Hence, the reduction is not extended. If \( L_i \sigma_m \) is not a variable, then if \( t_m \) unifies with \( L_i \sigma_m \), then again the reduction is not extended. Otherwise select \( f(L_i \sigma_1, \ldots, L_i \sigma_m-1, t_m) \) and the reduction \( f(t_1, \ldots, t_m), \ldots, f(L_i \sigma_1, \ldots, L_i \sigma_m-1, t_m), \ldots, f(L_i \sigma_1, \ldots, L_i \sigma_m-1, L_i \sigma_m) \) is also an N-reduction.

Since none of the \( L_i \) share any variables, \( f(L_i \sigma_1, \ldots, L_i \sigma_m) \) unifies with \( f(L_i, \ldots, L_i) \). Moreover, the m.g.u. is the union of \( \sigma_1, \ldots, \sigma_m \). Let \( \sigma \) be this union. Hence \( f(L_i \sigma_1, \ldots, L_i \sigma_m) \Rightarrow RHSG \). Since all the variables of RHS are in \( L_i, \ldots, L_i \sigma_m \) and for each \( \sigma_i \), \( L_i \sigma_i \) is ground, \( RHSG \) is ground.

The predication \( \text{reduce}(RHSG, En) \) succeeds and the length of the associated successful derivation is less than \( n \). By induction hypothesis, there is a successful N-reduction \( RHSG, \ldots, En \). Hence there is a successful N-reduction \( f(t_1, \ldots, t_n), \ldots, f(L_i \sigma_1, \ldots, L_i \sigma_m), RHSG, \ldots, En \).

QED.

**Theorem 3. The correctness of the compilation of F*.** Let \( P \) be an \( F^* \) program and \( PC \) be its compilation. Let \( E_0 \) and \( En \) be ground terms. Then there is a successful N-reduction beginning with \( E_0 \) and ending with \( En \) iff \( PC \mid \text{reduce}(E_0, En) \).

**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 \( E_0 \) and \( En \) be ground terms such that there is a successful N-reduction \( E_0, \ldots, En \). Then \( \text{reduce}(E_0, Z), Z \) 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 \( E_0, \ldots, En \). 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 \(F^*\):

\[
\begin{align*}
\text{first}(0, X) & \to [\]. \\
\text{first}(s(A), [U|V]) & \to [U|\text{first}(A, V)]. \\
\text{intfrom}(N) & \to [N|\text{intfrom}(s(N))].
\end{align*}
\]

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 \(\text{intfrom}(0)\) can be thought of as denoting the infinite list of integers \(0, 1, 2, \ldots\). However, \(\text{select}\), if given the term \(\text{first}(s(s(0)), \text{intfrom}(0))\), will simplify it to \([0|\text{first}(s(0), \text{intfrom}(s(0)))]\). 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:

\[
\begin{align*}
\text{reduce}(0, 0). \\
\text{reduce}(s(X), s(X)). \\
\text{reduce}([], []). \\
\text{reduce}([U|V], [U|V]). \\
\text{reduce}(\text{first}(A1, A2), \text{Out}) & \leftarrow \text{reduce}(A1, 0), A2 = X, \text{reduce}([], \text{Out}). \\
\text{reduce}(\text{first}(A1, A2), \text{Out}) & \leftarrow \text{reduce}(A1, s(A)), \text{reduce}(A2, [U|V]), \\
& \quad \text{reduce}([U|\text{first}(A, V)], \text{Out}). \\
\text{reduce}(\text{intfrom}(A1), \text{Out}) & \leftarrow A1 = N, \text{reduce}([N|\text{intfrom}(s(N))], \text{Out}).
\end{align*}
\]

Now the goal \(\text{reduce}(\text{first}(s(s(0)), \text{intfrom}(0)), Z)\) succeeds with \(Z = [0|\text{first}(s(0), \text{intfrom}(s(0)))]\). 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 it also satisfies the property of minimality. That is, it simplifies terms in a minimum number of steps.

(b) Investigating properties of an F* program satisfying 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


