ORIENTED EQUATIONAL CLAUSES AS A PROGRAMMING LANGUAGE

LAURENT FRIBOURG

▷ In the Prolog language, Horn clauses of first-order logic are regarded as programs, and the resolution procedure is used as an interpreter. In this paper, we present the formalism of Horn oriented equational clauses (Horn clauses with a rewrite rule as the head part, and a list of equations as the body part). We show that such a formalism can be interpreted as a logic language with built-in equality, and that a procedure based on clausal superposition can be used as an interpreter. We define the operational, model-theoretic and fixpoint semantics of the language, and prove their equivalence. Then we point out the advantages of such a programming language: embodying Prolog, mixing functional and relational features and, handling the equality relation. Lastly, we present experiments performed with an implemented interpreter. ◁

1. INTRODUCTION

Van Emden and Kowalski have shown that sentences of Predicate Logic can be regarded as programs [5]. This provides a theoretical model of the Prolog language [2, 3]. The Prolog language is based on Horn clause resolution. Our concern in this paper is to cover up two missing points of standard Prolog: the handling of functions and the handling of the equality relation. To reach such goals, several theoretical models were proposed [1, 9] and recently an extension of Prolog by inclusion of assertions about equality has been implemented [12].

In this paper, we propose an alternative approach which basically consists of performing the computations through the rule of clausal superposition. Clausal superposition indeed allows at once the replacement of an equal by an equal and the derivation of resolvents [7].

The statements handled by clausal superposition are Horn oriented equational clauses (Horn clauses with a rewrite rule as the head part, and a list of equations as

Address correspondence to L. Fribourg, Laboratoires de Marcoussis-C.G.E., 91460 Marcoussis, France.
Received 9 February 1984; accepted 31 May 1984.

52 Vanderbilt Ave., New York, NY 10017 0743-1066/84/$03.00

the body part). The computation procedure hereafter presented combines some aspects of the resolution procedure [14] (as used in Prolog) with some aspects of the rewrite system completion (used for refutation and computation purposes in [10, 4]).

The theoretical model of our programming language made of equational clauses is given in the framework of Predicate Logic with Equality. In keeping with [5], we define the operational, model-theoretic, and fixpoint semantics and we prove their equivalence. We then point out the advantages of the language:

1. handling the equality relation,
2. embodying Prolog programs,
3. mixing functional and relational features.

2. EQUATIONAL LOGIC PROGRAMS

2.1. Equational clauses

Definition 2.1.1. An *equational clause* is a first-order logic (with equality) formula of the form

$$L_1 = R_1, \ldots, L_p = R_p \leftarrow M_1 = N_1, \ldots, M_q = N_q,$$

where each L_i, R_i, M_j, N_j $(1 \leq i \leq p, 1 \leq j \leq q)$ is a term.

Definition 2.1.2. An *equational goal clause* is an equational clause of the form $\leftarrow M_1 = N_1, \ldots, M_q = N_q$.

Definition 2.1.3. An *equational definite clause* is an equational clause of the form:

$$L = R \leftarrow M_1 = N_1, \ldots, M_q = N_q.$$

An equational definite clause C is implicitly *oriented* from left to right: the leftmost equation $L = R$ must be viewed as the rewrite rule $L \rightarrow R$.

In the following, we assume that an *orientation rule* Δ is given, i.e., a function which maps any couple of terms $\langle M, N \rangle$ either into the equation $M = N$ or into $N = M$.

Definition 2.1.4. An *equational logic program* is a finite set of equational definite clauses.

Definition 2.1.5. An *equational Horn clause* is a clause which is either an equational definite clause or an equational goal clause.

Equational Horn clauses constitute the statements of our programming language. In the following, the computational use of equational Horn clauses will be referred to as the *equational logic* programming; it will be compared to the *classical logic* programming with standard Horn clauses.

2.2. *Operational Semantics of Equational Logic Programs*

In equational logic programming, computation is not performed with resolution, but with *clausal superposition* and *reflecting*. Clausal superposition, as defined in [7], is an oriented form of the rule of paramodulation [15], whereas reflecting is a form of resolution against the axiom $X = X$.

Definition 2.2.1. A *trivial* equation is an equation of the form $T = T$, where T is a term.

Definition 2.2.2. Let G be the goal $\leftarrow M_1 = N_1, \ldots, M_q = N_q$. The *trivial deletion* rule consists in removing from G all the equations $M_i = N_i$ $(1 \leq i \leq q)$, such that:

$M_i = N_i$ is trivial,

$M_j = N_j$ is trivial, for all $1 \leq j \leq i$.

Definition 2.2.3. Let G be the goal $\leftarrow M_1 = N_1, \ldots, M_q = N_q$, and let C be the definite clause $L = R \leftarrow L_1 = R_1, \ldots, L_r = R_r$. Then G' is a *goal-superposant* from C on G at occurrence t, using the most general unifier σ, if either:

M_1 has a subterm T_1 at occurrence t_1 unifiable with L, by m.g.u. σ $(T_1\sigma = L\sigma)$

G' is the goal obtained by trivial deletion from

$$\leftarrow \big(M_1[t_1 \leftarrow R] = N_1, L_1 = R_1, \ldots, L_r = R_r, M_2 = N_2, \ldots, M_q = N_q\big)\sigma$$

or:

N_1 has a subterm U_1 at occurrence u_1 unifiable with L, by m.g.u. σ $(U_1\sigma = L\sigma)$

G' is the goal obtained by trivial deletion from

$$\leftarrow \big(M_1 = N_1[u_1 \leftarrow R],\ L_1 = R_1, \ldots, L_r = R_r,\ M_2 = N_2, \ldots, M_q = N_q\big)\sigma.$$

If the subterm T_1 (resp. U_1) is nonvariable, the goal G' is said to be a *strict* goal-superposant of C on G.

Example:

C: $\text{rational}(X, Y) = \text{rational}(Z, W) \leftarrow X \times W = Z \times Y$

G: $\leftarrow \text{rational}(2, 3) \times Y = \text{rational}(X, 6) \times 4$

G': $\leftarrow \text{rational}(Z, W) \times Y = \text{rational}(X, 6) \times 4, \quad 2 \times W = Z \times 3$

Definition 2.2.4. Let C: $L = R \leftarrow L_1 = R_1, \ldots, L_r = R_r$ and C': $L' = R' \leftarrow L'_1 = R'_1, \ldots, L'_s = R'_s$ be two definite clauses. C'' is a *definite-superposant* of C on C' at occurrence t', using the m.g.u. σ, if:

L' has a subterm T' at occurrence t' unifiable with L $(T'\sigma = L\sigma)$

C'' is the definite clause:

$$P = Q \leftarrow \big(L_1 = R_1, \ldots, L_r = R_r, L'_1 = R'_1, \ldots, L'_s = R'_s\big)\sigma,$$

where $P = Q$ is the equation obtained by Δ-orienting the critical pair

$$\langle L'\sigma[t' \leftarrow R], R'\sigma \rangle.$$

If the subterm T' is nonvariable, then C'' is said to be a *strict* definite-superposant of C on C'.

Example:

C: rational$(X,1) = X \leftarrow$

C': rational(X,Y) = rational$(Z,W) \leftarrow X \times W = Z \times Y$

C'': rational$(Z,W) = X \leftarrow X \times W = Z \times 1$

Definition 2.2.5. Let G be the goal $\leftarrow M_1 = N_1, \ldots, M_q = N_q$. G' is a *reflectant* of G using the m.g.u. σ, if:

$M_1\sigma = N_1\sigma$

G' is the goal obtained by trivial deletion from $\leftarrow (M_2 = N_2, \ldots, M_q = N_q)\sigma$

Reflecting is the inference rule by which G yields G'.

Example:

G: $\leftarrow$ rational$(Z,W) \times Y$ = rational$(X,6) \times 4, \quad 2 \times W = Z \times 3$

G': $\leftarrow 2 \times 6 = X \times 3$

Definition 2.2.6. Let Q be an equational logic program, and G, G' two equational goals. A *linear RS-derivation* of G' from $Q \cup \{G\}$, denoted $Q \cup \{G\} \vdash_{RS} G'$, is a finite sequence $G_0, G_1, \ldots, G_n$ of goals such that:

1. G_0 is G, and G_n is G'
2. for all i, $1 \leq i \leq n$, G_i is either:
   (a) a reflectant of G_{i-1}, or
   (b) a (goal) superposant of a clause of Q on G_{i-1}.

A *linear RS-refutation* of $Q \cup \{G\}$ is a linear *RS*-derivation of the empty clause from $Q \cup \{G\}$.

Definition 2.2.7. Let P be a (equational logic) program. An *extension* Q of P is a program defined by a finite list $C_1, \ldots, C_m$ of definite clauses such that, for all i, $1 \leq i \leq m$, C_i is either:

(a) a member of P, or
(b) a (definite) superposant of C_j and C_k, for $j, k < i$.

Definition 2.2.8. Let P be a program, and G, G' two goals. An *RS-derivation* of G' from $P \cup \{G\}$, is a linear *RS*-derivation of G' obtained from $\{G\}$ and some extension Q of P. An *RS-refutation* of $P \cup \{G\}$ is an *RS*-derivation of the empty clause from $P \cup \{G\}$.

In equational logic programming, the computation rules are thus (clausal) superposition and reflecting. These computation rules, unlike the classical resolution rule, can produce new definite clauses, i.e., new statements of the program. An equational logic program P can thus be seen as a dynamic object. It can possibly be extended

without termination. Nevertheless, any computation consists in a linear derivation from a *finite* extension of P with the initial goal statement.

Another difference with classical logic programs is that the procedure invocation does not proceed by matching with the whole left-most atom but only with the left-hand side of this atom. Therefore, the left-hand side (and no longer the whole atom) must be interpreted as the procedure name, the remaining part of the clause standing for the procedure body. Thus equational logic languages have strong *functional* features, whereas standard Prolog is purely *relational.*

3. MODEL-THEORETIC SEMANTICS

The reader is assumed to be familiar with the interpretation, model, and logical consequence notions. These notions classically extend to first-order logic with equality (see [15]). The definitions given hereunder are slightly simpler because the only predicate involved is equality.

Definition 3.1. Given a set Q of equational clauses, the set of *axioms of equality* for Q is the set defined as:

$$EA(Q) = \{ X = X \leftarrow \} \cup \{ X = Y \leftarrow Y = X \} \cup \{ X = Z \leftarrow X = Y, Y = Z \}$$
$$\cup \{ F(X_1, X_2, \ldots, X_k) = F(Y_1, X_2, \ldots, X_k) \leftarrow X_1 = Y_1,$$

for all k-ary function symbol F occurring in Q and each argument of F }, where $X, Y, Z, X_1, X_2, \ldots, X_k, Y_1$ denote variables.

Definition 3.2. Let Q be a set of clauses, and let L and R be two terms. The equation $L = R$ is an *E-logical consequence* of Q (denoted $Q \models_E L = R$) iff it is a logical consequence of $Q \cup EA(Q)$ (i.e. $Q \cup EA(Q) \models L = R$).

Let P be an equational logic program.

The *Herbrand universe* of P is the set of all ground terms composed of the constant and function symbols appearing in P (in the case that P has no constants, add some constant, say α, to P).

The *Herbrand base* $B(P)$ of P is the set of all ground equations $M = N$, where M and N belong to the Herbrand universe $U(P)$ of P.

A *Herbrand interpretation* I of P is any subset of the Herbrand base of P.

In the following, the symbol $=_I$ denotes the congruence modulo I.

Let us now introduce the notion of Herbrand E-model.

Definition 3.3. Let I be a Herbrand interpretation and let C be the equational clause:

$$L_1 = R_1, \ldots, L_p = R_p \leftarrow M_1 = N_1, \ldots, M_q = N_q$$

C is *E-true* in I iff for every ground substitution η: $L_i\eta =_I R_i\eta$, for some i $(1 \le i \le p)$, or $\sim (M_j\eta =_I N_j\eta)$, for some j $(1 \le j \le q)$. C is *E-false* in I iff it is not E-true.

Definition 3.4. Let I be a Herbrand interpretation and Q a set of equational clauses. I is a *Herbrand E-model* of Q iff each clause in Q is E-true in I.

REMARK. Beware that our notion of Herbrand E-model is distinct from the one of R-interpretation defined in [15] (an R-interpretation of Q is an E-model of Q, but the converse is false in general).

Proposition 3.1. Let Q be a set of equational clauses. Q has a Herbrand E-model iff $Q \cup EA(Q)$ has a Herbrand model.

We now transpose the classical notion of the least Herbrand model of a logic program.

Definition 3.5. Let I and J be two interpretations of a set Q of equational clauses. The *E-intersection* of I and J (denoted $I\Omega J$) is the subset of the equations $M \simeq N$ of I such that $M =_J N$.

The definition of the operator Ω is not symmetric. However the congruences defined by $I\Omega J$ and $J\Omega I$ coincide ($M =_{I\Omega J} N$ iff $M =_I N$ and $M =_J N$). The definition of Ω naturally extends to a countable (ordered) set of interpretations.

Proposition 3.2. Let Q be a set of equational Horn clauses, and let L be a nonempty set of Herbrand E-models for Q. Then ΩL is a Herbrand E-model of Q.

The proof is exactly the same as the one given in Section 5 in [5], except that the membership relation of an atom A to an interpretation I is replaced by the congruence relation of the sides of an equation A modulo I.

Definition 3.6. A Herbrand E-model I of Q is a *least* Herbrand E-model of Q if, for any Herbrand E-model J of Q, $=_I \subseteq =_J$.

The congruences of all the least Herbrand E-models coincide, and define the so-called *least model congruence*.

Let P be an equational logic program. Let **EM(P)** be the nonempty set (supposed ordered) of all Herbrand E-models of P. By Proposition 3.2, the intersection Ω**EM(P)** of all the Herbrand E-models of P is a Herbrand E-model, and clearly is a least Herbrand E-model of P. The following proposition gives a characterization of the least model congruence.

Proposition 3.3. Let $L \simeq R$ be a member of the Herbrand base $B(P)$ of P.

$$L =_{\Omega \mathbf{EM(P)}} R \quad \text{iff} \quad P \models_E L \simeq R$$

PROOF

$P \models_E L \simeq R$.

iff $P \cup EA(P) \models L \simeq R$

iff $P \cup EA(P) \cup \{\leftarrow L \simeq R\}$ has no model

iff $P \cup EA(P) \cup \{\leftarrow L \simeq R\}$ has no Herbrand model (by Skolem-Lowenheim theorem)

iff $P \cup \{\leftarrow L \simeq R\}$ has no Herbrand E-model (by Proposition 3.1)

iff $(\leftarrow L = R)$ E-false in all the Herbrand E-models of P

iff $L =_I R$, for any Herbrand E-model I of P

iff $L =_{\Omega EM(P)} R$.

4. FIXPOINT SEMANTICS

Let P be an equational logic program and $B(P)$ the Herbrand base of P. Given a Herbrand interpretation I, the symbol $\rightarrow_I$ denotes the reduction relation by I, considered as a set of rewrite rules; $\overset{*}{\rightarrow}_I$ denotes the reflexive-transitive closure of $\rightarrow_I$ (see [11]).

In keeping with [5], we associate with the program P a mapping S_P of Herbrand interpretations, as follows:

Definition 4.1. For any Herbrand interpretation I, DS, GS_P, S_P are the transformations which respectively map I to:

$DS(I) = \{ L = R \in B(P) / L = R$ is a Δ-oriented critical pair of two members of $I \}$

$GS_P(I) = \{ L = R \in B(P) / (L = R \leftarrow M_1 = N_1, \ldots, M_q = N_q)$ is a ground instance of a clause in P, and for any i $(1 \leq i \leq q)$, there exists a term K_i such that $M_i \overset{*}{\rightarrow}_I K_i$ and $N_i \overset{*}{\rightarrow}_I K_i \}$

$S_P(I) = DS(I) \cup GS_P(I)$.

REMARKS. $DS(I)$ can be seen as the set of all the (definite) superposants of I, viewed as a set of definite clauses. $GS_P(I)$ is obtained from I by applying sequences of (goal) superposition. DS, GS_P, and S_P are monotonic (for the order of set inclusion $\subseteq$).

As usual, for any mapping T, we define the mappings $T^n(I)$ by:

$T^0(I) = I$, $T^{n+1}(I) = T(T^n(I))$

Definition 4.2. An interpretation I is an *E-fixpoint* of S_P (or I is E-closed under S_P) iff the congruences $=_I$ and $=_{S_P(I)}$ are identical.

REMARK. Since $I \subseteq S_P(I)$, I is an E-fixpoint iff $=_{S_P(I)} \subseteq =_I$.

Definition 4.3. An E-fixpoint I of S_P is a *least* E-fixpoint of S_P if, for any E-fixpoint J of S_P, $=_I \subseteq =_J$.

The congruences of all the least E-fixpoints coincide, and define the so-called *least fixpoint congruence.*

5. FIXPOINT AND MODEL-THEORETIC SEMANTICS

Let us show the equivalence between fixpoint and model-theoretic semantics.

Lemma 5.1. Let M and N be terms of the universe of Herbrand and I an interpretation. $M =_I N$ iff $M \overset{*}{\rightarrow}_{DS^n(I)} K$ and $N \overset{*}{\rightarrow}_{DS^n(I)} K$, for some term K and some integer n.

The proof is a direct transposition of the proof of theorem 4.1 of [7] (completeness of A-paramodulation) in the ground unit case. The result holds, independently of the chosen orientation rule Δ.

Theorem 5.1. Let P be an equational program and I be an interpretation of P. Then I is an E-model of P iff $=_{S_P(I)} \subseteq =_I$.

PROOF. ($\Rightarrow$) Let us suppose that I is an E-model of P. Let $L = R$ be an equation of $S_P(I)$. By definition, there exist either:

some ground instance of a clause in P, $L = R \leftarrow M_1 = N_1, \ldots, M_q = N_q$ with, for every i ($1 \leq i \leq q$), a term K_i such that $M_i \overset{*}{\rightarrow}_I K_i$ and $N_i \overset{*}{\rightarrow}_I K_i$, or

two equations of I which superpose themselves into $L = R$.

In the first case, $M_i =_I N_i$, for $1 \leq i \leq q$. Since I is an E-model of P, we have: $L =_I R$. In the second case, we have also: $L =_I R$. So in both cases if $L = R \in S_P(I)$, then $L =_I R$. Therefore, $=_{S_P(I)} \subseteq =_I$.

($\Leftarrow$) Let us suppose that I is not an E-model of P, and let us show that $=_{S_P(I)} \not\subseteq =_I$. Since I is not an E-model, I E-falsifies a ground instance of a clause in P of the form

$$L = R \leftarrow M_1 = N_1, \ldots, M_q = N_q.$$

So, we have: (1) $\sim (L =_I R)$ and (2) $M_i =_I N_i$, for $\leq i \leq q$. From (2) and Lemma 5.1: $M_i \overset{*}{\underset{DS^{ni}(I)}{\longrightarrow}} K_i$, and $N_i \overset{*}{\underset{DS^{ni}(I)}{\longrightarrow}} K_i$, for some integer ni and for some term K_i ($1 \leq i \leq q$). So, by monotonicity, $M_i \overset{*}{\underset{S_P^{ni}(I)}{\longrightarrow}} K_i$ and $N_i \overset{*}{\underset{S_P^{ni}(I)}{\longrightarrow}} K_i$, for $1 \leq i \leq q$. Therefore: $L = R \in GS_P(S_P^n(I))$, for $n = \max\{ni\}_{1 \leq i \leq q}$. Hence: (2′)

$$L = R \in S_P^{n+1}(I).$$

Clearly, from (1) and (2′), it follows: $=_{S_P(I)} \not\subseteq =_I$. □

Theorem 5.1 states that I is an E-model of P iff I is an E-fixpoint of S_P.

Let us now compare the least model congruence of P with the least fixpoint congruence of S_P.

Let **EC(P)** be the set of all the Herbrand interpretations E-closed under S_P. From Theorem 5.1, it easily follows that:

1. the congruences modulo Ω**EC(P)** and modulo Ω**EM(P)** are identical.
2. Ω**EC(P)** is E-closed under S_P.

Thus, the least model congruence and the least fixpoint congruence coincide. Hence, the *model-theoretic* and the *fixpoint* semantics coincide.

The following proposition gives a characterization of the least E-fixpoint congruence.

Proposition 5.1. Let P be a program, M and N two terms of the Herbrand universe $U(P)$. Let W abbreviate $\bigcup_{m=0}^{\infty} S_P^m(\phi)$. Then $M =_{\Omega EC(P)} N$ iff $M =_W N$.

The proof is similar to the proof given at Section 8 in [5], but makes use of the congruence relation and superposition rule instead of the membership relation and hyperresolution rule.

6. OPERATIONAL AND MODEL-THEORETIC SEMANTICS

Let us state the equivalence between the operational and model-theoretic semantics. The underlying result of this equivalence is the completeness of *RS*-deduction for first-order logic with equality. As in paramodulation (see [15]), the completeness proof requests the inclusion of the set of functional reflexive units.

Definition 6.1. For a given set Q of clauses, the set of the *functional reflexive units* is the set defined as:

$$Q^f = \{ F(X_1,\ldots,X_k) = F(X_1,\ldots,X_k) \leftarrow, \text{ for all } k\text{-ary function symbol } F \text{ occurring in } Q \}.$$

Theorem 6.1. Let P be an equational logic program, and M, N two terms of $U(P)$. Then $M =_{\Omega EM(P)} N$ iff $P \cup P^f \cup \{ \leftarrow M = N \}$ has an RS-refutation.

The proof cannot be given due to lack of space (the whole proof is in [8]).

Theorem 6.1 states that the *success set* of $P \cup P^f$ [i.e., the set of ground equations $M = N$ of $B(P)$ such that $P \cup P^f \cup \{\leftarrow M = N\}$ has an *RS*-refutation] coincide with the *least model congruence* of P. Thus, the operational and model-theoretic semantics coincide, modulo the inclusion of the functional reflexive units in P.

By analogy with paramodulation, we conjecture that Theorem 6.1 still holds when the set P^f of functional reflexive units is removed and when superposition is restricted to strict superposition.

When some functions are associative and/or commutative, the completeness theorem 6.1 is still valid if associative/commutative unification is used instead of ordinary unification, even though the corresponding axioms of associativity/commutativity are not added (see [13, 16], for completeness proofs of such extensions). The building-in of associativity-commutativity into *RS* computations through *AC*-unification is thus justified (see [6], for AC-unification algorithms).

7. FORMAL FEATURES OF EQUATIONAL LOGIC PROGRAMMING

7.1. A Mixed-Functional-Relational Language Embodying Prolog

Equational logic naturally has functional features (see Section 2.2). However, it can also behave as a relational language as well as Prolog.

Through the formalism of equational clauses indeed, we have until now dealt with formulas involving no predicate but equality. Yet it is easy to handle a general predicate of the form $R(T,U,V)$, simply by translating it into the equation $R(T,U,V) = \text{true}$, where "true" denotes a new constant symbol. An attractive feature of this coding is that (goal) superposition with such equations simulates the resolution rule [7]. For instance, the superposition of $R(T,U,V) = \text{true} \leftarrow$ on $\leftarrow R(T,U,V) = \text{true}$ gives the goal $\leftarrow \text{true} = \text{true}$, which becomes the empty clause after trivial deletion. Through this coding, any Prolog program P can be straightforwardly translated into an equational one P'. Given P' and an initial goal G', the *RS*-derivation procedure computes nothing else than the resolvents of P' against the goal statements, and thus behaves exactly as a Prolog interpreter. The relational program P' can indeed be refined by replacing predicates by functions. The example

given in the appendix illustrates this flexibility; moreover, it illustrates that the input-output reversing property of Prolog is maintained in equational logic programs.

7.2. A Prolog-like Language with Built-in Equality

Equational Horn clauses constitute a very convenient way to integrate the equality relation in a Prolog-like language. The handling of equations allows the replacement of an equal by an equal. Furthermore, the left-most equations are actually oriented, and the replacements are only applied from left to right. This discards useless paramodulants, and enables the language to efficiently handle the sensitive property of substitutivity.

The examples given in [12] have been successfully implemented. As an illustration, we give a small program which allows us to decide if a number (integer or rational) is equal to a list member:

$$\text{rational}(X, Y) = \text{rational}(Z, W) \leftarrow X \times W = Z \times Y$$

$$\text{rational}(X, 1) = X \leftarrow$$

$$\text{member}(X, X, Y) = \text{true} \leftarrow$$

$$\text{member}(X, Y, Z) = \text{true} \leftarrow \text{member}(X, Z) = \text{true}$$

For the following input:

$$\leftarrow \text{member}(\text{rational}(4, X), (2.(3.((Y.Z).(\text{rational}(2,7).NIL))))),$$

the computed answers are $\{X \leftarrow 2\}$ and $\{X \leftarrow 14\}$. Note that the computation succeeds, in spite of the permutativity of the head equation $\text{rational}(X, Y) = \text{rational}(Z, W)$.

8. IMPLEMENTATION

A prototype implementation, based on the theorem-proving program SEC [7], has been realized. SEC is an extension of the Knuth-Bendix algorithm implemented at INRIA within the FORMEL system. Unlike most Prolog interpreters, the search plan of SEC is complete (smallest components strategy). Relatively to the *RS*-derivation procedure described above, SEC presents two main differences:

- SEC computes the strict superposants only, and makes no use of the functional reflexive units.
- SEC normalizes the reducible terms, with the rewrite system made of the unit definite clauses.

An interpreter, called SLOG (LOGic with Superposition), is under development at Laboratoires de Marcoussis.

9. CONCLUSION

We have presented in this paper a programming language based on Horn equational clauses. This formalism allows the handling of equality and the combination of both relational and functional approaches. We have defined operational semantics by describing the computations performed with an interpreter of the language. The

major inference rule used by the interpreter, is the operation of clausal superposition which is a powerful inference rule for first-order logic with equality. Model-theoretic and fixpoint semantics have also been defined and have been shown equivalent to the operational ones. First experimental results confirm that the interpreter behaves as a standard Prolog interpreter for classical Horn clauses, and, in addition, efficiently handles statements about equality.

I would like to thank Laurent Kott who first suggested the computational use of equational clauses, and Herve Gallaire for helpful discussions and support.

APPENDIX: A COMPUTATIONAL EXPERIMENT WITH SEC

The following Prolog program P1, borrowed from [3], computes the sum of two integers, using the successor relation:

C1: $+(1, X, Y) \leftarrow \text{succ}(X, Y)$

C2: $+(U, Y, W) \leftarrow \text{succ}(X, U), +(X, Y, Z), \text{succ}(Z, W)$

C3: $\text{succ}(1,2) \leftarrow$

C4: $\text{succ}(2,3) \leftarrow$

...

C10: $\text{succ}(8,9) \leftarrow$

Three successive goals are computed:

G1: $\leftarrow +(4,3,X)$ answer: $\{X \leftarrow 7\}$

G2: $\leftarrow +(4,X,7)$ answer: $\{X \leftarrow 3\}$

G3: $\leftarrow +(X,Y,5)$ answers: $\{X \leftarrow 1, Y \leftarrow 4\}$, $\{X \leftarrow 2, Y \leftarrow 3\}$, $\{X \leftarrow 3, Y \leftarrow 2\}$, $\{X \leftarrow 4, Y \leftarrow 1\}$.

Computation with such a relational program is clearly highly expensive. Computation of G3 leads for instance to 8! calls of C2, the body literal $+(X, Y, Z)$ being involved before Z has been evaluated to 4 through the computation of $\text{succ}(Z, 5)$. In return, the translation of the classical clauses C1,...,C10 into equational clauses gives a very natural program P2:

EC1: $+(1, X) = \text{succ}(X) \leftarrow$

EC2: $+(\text{succ}(X), Y) = \text{succ}(+(X, Y)) \leftarrow$

EC3: $2 = \text{succ}(1) \leftarrow$

EC4: $3 = \text{succ}(2) \leftarrow$

...

EC10: $9 = \text{succ}(8) \leftarrow$

The goal G1 has now the following form:

$\leftarrow +(4,3) = X,$

which is evaluated to,

$\leftarrow \text{succ}(\text{succ}(\text{succ}(\text{succ}(\text{succ}(\text{succ}(1)))))) = X,$

and gives by reflecting the answer:

$\{X \leftarrow \text{succ}(\text{succ}(\text{succ}(\text{succ}(\text{succ}(\text{succ}(1))))))\}$

The goal G2 has now the form:

$\leftarrow +(4, X) = 7,$

which is evaluated to:

$\leftarrow \text{succ}(\text{succ}(\text{succ}(\text{succ}(X)))) = \text{succ}(\text{succ}(\text{succ}(\text{succ}(\text{succ}(\text{succ}(1))))))$,

and gives by reflecting the answer:

$\{X \leftarrow \text{succ}(\text{succ}(1))\}$

The goal G3 has now the form:

$\leftarrow +(X, Y) = 5,$

which is evaluated to:

$\leftarrow +(X, Y) = \text{succ}(\text{succ}(\text{succ}(\text{succ}(1))))$.

By superposition with EC1 and reflecting, we get the first answer:

$\{X \leftarrow 1,\ Y \leftarrow \text{succ}(\text{succ}(\text{succ}(1)))\}$.

On the other hand, by superposition of G3 with EC2, we have:

$\leftarrow \text{succ}(+(X', Y)) = \text{succ}(\text{succ}(\text{succ}(\text{succ}(1))))$, with X bound to $\text{succ}(X')$.

Then, through superposition with EC1 and reflecting, we get the second answer:

$\{X' \leftarrow 1,\ Y \leftarrow \text{succ}(\text{succ}(1))\}$, so $\{X \leftarrow \text{succ}(1),\ Y \leftarrow \text{succ}(\text{succ}(1))\}$.

The process going on, we find the two last answers:

$\{X \leftarrow \text{succ}(\text{succ}(1)),\ Y \leftarrow \text{succ}(1)\}$ and $\{X \leftarrow \text{succ}(\text{succ}(\text{succ}(1))),\ Y \leftarrow 1\}$

The computation of the successful answers is thus straightforward. Unfortunately, the computation does not end, for, by superposition with EC2, we generate the infinite following sequence of clauses:

$\leftarrow \text{succ}(\text{succ}(\text{succ}(\ldots(\text{succ}(+(X, Y)))\ldots))) = \text{succ}(\text{succ}(\text{succ}(\text{succ}(1))))$

which clearly has no answer.

For the above inputs, the most suitable program (combining the straightforward computation mode of P2 and the termination of P1) is the program P2′ obtained from P2 by substituting EC1, EC2 by:

EC1′: $+(1, X, \text{succ}(X)) = \text{true} \leftarrow$

EC2′: $+(\text{succ}(X), Y, \text{succ}(Z)) = \text{true} \leftarrow +(X, Y, Z) = \text{true}$

However, the program P2′ is generally less powerful than the pure functional program P2, because the ability of evaluating the function $+(X, Y)$ is lost. The resolution with P2′ of the inequation $+(X, Y) \leq 5$ for instance would be much more clumsy than with P2. Nevertheless, the program P2′ illustrates the flexibility of the language of equational clauses as a mixed functional-relational language.

At last, let us notice that, if the clauses EC3,..., EC10 had been oriented the other way, similar computations would have been performed, but with an *extended*

program. For instance, from a clause EC3′: succ(1) = 2←, one would have derived (by normalized superposition with EC2) the definite clause:

$$+(2, Y) = \text{succ}(\text{succ}(Y)) \leftarrow .$$

REFERENCES

1. Bellia, M., Degano, P., and Levi, G., A Functional Plus Predicate Logic Programming Language, *Proc. Logic Programming Workshop* 334–347 (July 1980).
2. Colmerauer, A., Van Caneghem, M., and Kanoui, H., PROLOG II, manuels de reference, d'utilisation et d'exemples, GIA: Groupe d'Intelligence Artificielle, Marseille, 1982.
3. Colmerauer, A., Prolog in 10 figures, *Proc. IJCAI*, Karlsruhe, 487–499, 1983.
4. Dershowitz, N., Computing with rewrite systems, Report No ATR-83 (8478)-1, The Aerospace Corporation, El Segundo, CA, 1983.
5. van Emden, M. H. and Kowalski, R. A., The Semantics of Predicate Logic as a Programming Language, *J. ACM* 23:733–742 (Oct 1976).
6. Fages, F., Associative-Commutative Unification, *Proc. CADE-7*, Napa, 1984.
7. Fribourg, L., A Superposition Oriented Theorem Prover, Technical Report 83/11, L.I.T.P (to appear in TCS, short version in Proc. IJCAI-83, 923–925).
8. Fribourg, L., Oriented equational clauses as a programming language, Technical Report DIN 84002, Laboratoires de Marcoussis, France, 1984.
9. Hoffman, C. M. and O'Donnell, M. J., Programming with Equations, *ACM Trans. Programming Languages and Systems* 4:83–112 (Jan 1982).
10. Hsiang, J. and Dershowitz, N., Rewrite Methods for Clausal and Non-clausal Theorem Proving, *Proc. 10th ICALP*, Barcelona, 1983.
11. Huet G. and Oppen, D. C., Equations and Rewrite Rules: A Survey, Formal Language Theory: Perspectives and Open Problems, Ronald V. Book, (ed.), Academic Press, New York, 1980, pp. 349–405.
12. Kornfeld, W., Equality for Prolog, *Proc. IJCAI*, Karlsruhe, 1983, pp. 514–519.
13. Plotkin, G., Building in Equational Theories, In: *Machine Intelligence 7*, B. Meltzer and D. Michie (eds.), Halsted Press, New York, 1973, pp. 73–90.
14. Robinson, J. A., A Machine-Oriented Logic Based on the Resolution Principle, *J. ACM* 12:23–41 (Jan 1965).
15. Robinson, G. A. and Wos, L., Paramodulation and Theorem Proving in First Order Theories with Equality, In: *Machine Intelligence 4*, Meltzer & Michie (eds.), American Elsevier, New York, 1969, pp. 135–150.
16. Slagle, J. R., Automated Theorem-Proving for Theories with Simplifiers, Commutativity and Associativity, *J. ACM* 21:622–642 (Oct 1974).