How to characterize the language of ground normal forms

Jean-Luc Rémy, Hubert Comon

INRIA

UNITÉ DE RECHERCHE
INRIA-LORRAINE

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P. 105
78153 Le Chesnay Cedex
France
Tél.:(1) 39 63 55 11

Rapports de Recherche

N° 676

HOW TO CHARACTERIZE THE LANGUAGE OF GROUND NORMAL FORMS

Jean-Luc REMY
Hubert COMON

JUIN 1987

HOW TO CHARACTERIZE THE LANGUAGE OF GROUND NORMAL FORMS

COMMENT CARACTERISER LE LANGAGE DES FORMES NORMALES SANS VARIABLE

PAR

JEAN-LUC REMY

ET

HUBERT COMON

Résumé : Les systèmes de réécriture de termes fournissent aux informaticiens un langage pour exprimer des définitions fonctionnelles sur des domaines complexes. Dans un tel contexte, nous sommes confrontés au problème de caractériser les formes normales closes, ou sans variable, puisqu'elles représentent les résultats des calculs. Faire cela permet de tester la complétude d'une définition fonctionnelle (dans ce cas, les formes normales closes sont constituées uniquement par des constructeurs) ou de tester l'équivalence de deux définitions fonctionnelles différentes (il est alors possibble d'établir une correspondance biunivoque entre les formes normales closes).
Nous présentons ici une caractérisation des formes normales closes en termes de grammaire, et un algorithme pour dériver celle-ci à partir des membres gauches des règles de réécriture. Quand ces derniers sont linéaires, la forme de la grammaire est particulièrement simple puisqu'elle est indépendante du contexte (à condition de considérer les termes comme des mots, écrits par exemple en forme préfixe). Donc nous obtenons une procédure de décision pour la finitude de l'ensemble des formes normales closes et nous sommes également capables de déterminer les formes normales qui sont des instances, d'un terme linéaire donné. Les idées qui se trouvent derrière les algorithmes sont très simples et nous les illustrons par plusieurs exemples.

Abstract : Term rewriting systems provide the computer scientists with a language to express functional definitions on complex domains. In such an area, we face the problem to characterize the ground normal forms (whithout any variables) as they are the results of computations. Doing that allows one to check the completeness of a functional defination (then the ground normal forms are built only with constructors) or to check the equivalence of two different functional definitions (it is possible to establish a ont-to-one correspondence between the ground normal forms).
We present here a characterization of the ground normal forms in terms of grammar, and an algorithm to derive it from the form of the left-hand sides of the rewrite rule. When the left-hand sides are linear, the form of the grammar is especially simple as it is context-free when we consider the terms as words, written for instance in prefix form. Therefore we get a decision procedure for the finiteness of the set of ground normal forms and we are also able to compute the normal forms which are the instances of a given linear term. The ideas behind the algorithms we present are very simple and we illustrate them by several examples.

How to characterize the language of ground normal forms

by

H.Comon
LIFIA, BP 68,
38402 Saint Martin d'Hères cedex
France
Tel (33) 76514600 ext 5165
UUCP ...!mcvax!inria!lifia!comon
and

JL. Rémy
CRIN, BP 239,
54506 Vandoeuvre-lès-Nancy cedex
France
Tel (33) 83912128
UUCP ...!mcvax!inria!crin!remy
TELEX : NANCYUN 960646F

ABSTRACT

Term rewriting systems provide the computer scientists with a language to express functional definitions on complex domains. In such an area, we face the problem to characterize the ground normal forms (without any variables) as they are the results of computations. Doing that allows one to check the completeness of a functional definition (then the ground normal forms are built only with constructors) or to check the equivalence of two different functional definitions (it is possible to establish a one-to-one correspondence between the ground normal forms).
We present here a characterization of the ground normal forms in terms of grammar, and an algorithm to derive it from the form of the left-hand sides of the rewrite rules. When the left-hand sides are linear, the form of the grammar is especially simple as it is context-free when we consider the terms as words, written for instance in prefix form. Therefore we get a decision procedure for the finiteness of the set of ground normal forms and we are also able to compute the normal forms which are the instances of a given linear term.The ideas behind the algorithms we present are very simple and we illustrate them by several examples.

1. Introduction

In this paper, we are speaking in terms of algebraic specifications, since it corresponds to our culture, but the reader can easily translate that in the functional programming vocabulary.
In the initial algebra semantics [GTW78,Ber79] , the algebraic specification (S,Σ,E) of an Abstract Data Type denotes the quotient algebra $T_{\Sigma,E}$ of the set of "ground terms"T_Σ by the congruence relation $=_E$ generated by E. Working on or with this quotient algebra makes necessary to have a "usable" definition of it. First, it must be possible to decide of the equality of two terms. Usually, this is performed by orientating the equations of E. If the resulting rules may be completed into a canonical Term Rewriting System, then the equality of two terms is equivalent to the syntactical equality of their normal forms. Unfortunately the canonicity of the TRS is not sufficient for working on $T_{\Sigma,E}$ since the particular fact that $T_{\Sigma,E}$ is an *initial* model is not exploited. For example, the decision of the *protecting property* [MG86] and of the *full definition property* [Com86b] as well as the automatisation of inductive proofs [HH82,HKi84,Pau84,LLT86,JK86,for example] and the proof of the equivalence of two specifications [RV82] use the particular fact that we are in an initial model. More simply, it is not obvious to answer to the question: *is* $T_{\Sigma,E}$ *finite* ?
In this paper, we will assume that the TRS associated with the specification is canonical (actually only finite termination will be needed). The set of ground normal forms will be denoted by NF. In this way, the map which associates each element of NF to its class is a bijective map from NF into $T_{\Sigma,E}$. The problem is thus to describe NF. We show first that NF is the solution of a fixed point equation and thus may be viewed as a formal language. Actually, there are two ways to consider it as a language : tree language (since terms are trees) and word language (if we look , for instance, at the prefix forms of the terms). Using then "inequations"[Com86a] , we transform the fixed point equation and get a "usable" definition of NF.
This definition may be viewed as a "compilation" of the definition of NF. Indeed, it is then easier to check whether a term is in the language as well as to decide whether a set of ground terms, described by a term with variables, contains (or not) an element of the language.This last check is the " inductive-reducibility" property in [JK86]. Then, though the decision of "inductive reducibility" spends exponential time in the general case ([KNZ86]), the expensive computation may probably be done only once (computation of the grammar) . Then every new check of inductive reducibility uses the grammar and may probably be done more rapidly.
Moreover, when the lhs are linear, the description of NF produced by our

algorithm is a context-free grammar. This is the main result in this paper (theorem 4.1). It is then easy to decide whether the language generated by a non-terminal is empty as well as its finiteness.
Other results about the language of ground normal forms can be found, for instance, in [G&B85] ; however, at our knowledge, no explicit way of computing the grammar of the language was given until now.
In section 2 we give the relation satisfied by NF and which is the starting point of our work. In section 3 we recall some results on inequation solving. In section 4 we use this result for transforming the equation satisfied by NF and deriving the main result of the paper. Finally, in section 5 we give an algorithm for the reduction of the grammar, an algorithm for the decision of the finiteness of NF and an algorithm for the computation of the irreducible ground instances of a term t, when t as well as every lhs of a rule is linear.

2. Characterizing the set of ground terms in normal form by a fixed point equation

Suppose Σ is a given signature. Let T(X) denote the set of terms constructed on this signature and the set of variables X. We write T instead of T(∅). If $t \in T(X)$, then root(t) denotes the operator at the root of t and Sub(t) denotes

- the empty set if t is a constant or a variable
- $\{t_1,\ldots,t_n\}$ if $t = f(t_1, \ldots, t_n)$.

We assume in the following that "$\rightarrow$" is a reduction relation associated with a finitely terminating Term Rewriting System. Then NF will denote the set of ground normal forms i.e. the terms of T which cannot be reduced using the reduction relation "$\rightarrow$".

Then a ground term t belongs to NF iff

(i) t cannot be matched to any lhs of a rule
(ii) every term in Sub(t) is in NF

When $f \in \Sigma$, let f(NF) denote the set $\{t \in T,\ root(t) = f, \forall u \in Sub(t), u \in NF\}$. Then the above characterisation may be written:

$$\mathbf{NF = Complement(LHS) \cap (\cup_{f \in \Sigma} f(NF))}$$

Complement(LHS) denotes the set of ground terms which cannot be matched to any lhs of a rule. Obviously, this set must be defined by another way. This is what is done in section 3.
If t is a term, NF_t will denote the set of all irreducible ground instances of t. Often, when t is linear, the variables will be omitted when writing t. We will write for instance NF_f instead of $NF_{f(x1, \ldots, xn)}$.

Definition 2.1 (JK86)
A term t is ***inductively reducible*** *iff, for every substitution σ from T(X) into T, σ(t) is not in normal form.*

Note that $NF_t = \varnothing$ iff t is inductively-reducible.
The above equation may be rewritten, using these notations:

$$\mathbf{NF} = \cup_{f \in \Sigma} \mathbf{NF_f} \quad \text{and} \quad \mathbf{NF_f = Complement(LHS) \cap f(NF)}$$

We show now how to express Complement(LHS) in such a way that the computation with it will be easy.

3. Normal forms and inequations

In this section we show how to express the "complement" of the left hand sides of the rules in terms of "inequations". The concept of such inequations was introduced in [Com86a]. We recall the basic definitions.

Definition 3.1
Let D be a subset of X, F a subset of T(X), a substitution σ is a **D-solution in F of the inequation t # t'** *iff*

- $Dom(\sigma) \subseteq D$
- $\sigma(D) \subseteq F$
- *t and σ(t') are not unifiable.*

Such a solution is **D-linear** *iff, for every linear term t in T(D), σ(t) is linear.*

Note that this definition of inequations is quite different, for example, from the definition of inequations in PROLOG II [Col82] since t and σ(t') must be *not unifiable* (and not only different).

Definition 3.2
A substitution σ is **a D-solution in F of the system** $\{ t_i \# u_i, 1 \le i \le n \}$ *iff*
- $Dom(\sigma) \subseteq D$ - $\sigma(D) \subseteq F$ - *for every index i,* t_i *and* $\sigma(u_i)$ *are not unifiable.*

When F is omitted, it must be understood that F=T(X).

Proposition 3.3

Let LHS = $\{l_1,...,l_m\}$ denote the (finite) set of the left hand sides of the rules. Let t be any linear term of depth 1 (i.e. t is a linear term of the form $f(x_1,...,x_n)$).

Then $u \in NF_t$ iff it exists a substitution σ such that

(i) $u = \sigma(t)$

(ii) σ is a solution in NF of the system $\{l_i \# t, 1 \le i \le m\}$.

Indeed, - if $u \in NF_t$, then $u = \sigma(t)$ for some σ and, u being in normal form, it is unifiable with no lhs of a rule.

- if $u = \sigma(t)$ and σ is a solution in NF of the system $\{l_i \# t, 1 \le i \le m\}$, then u is not unifiable with any lhs of a rule, thus no rule may be applied at the root of u.

Moreover, every subterm of u different from the root is in $Im(\sigma)$ since depth(t) = 1, therefore every subterm of u different from the root is in normal form.

This proposition gives exactly the translation of the fixed point equation:

$$NF = \cup_{f \in \Sigma} (f(NF) \cap \text{Complement(LHS)})$$

or, more precisely, translates the equation:

$$NF_f = f(NF) \cap \text{Complement(LHS)}$$

Proposition 3.4

Let $t = f(t_1,...,t_n)$ be a linear term. Then, $u \in NF_t$ iff it exists a substitution θ s.t.

(i) $u = \theta(t)$ (ii) θ is a solution in T of the system $\{l_i \# t, 1 \le i \le m\}$

(iii) $\forall i$, $\theta(t_i)$ is in normal form.

This extends the proposition 3.3 to the case of linear terms.

The resolution of such inequation systems is performed by an "anti-unification" algorithm [Com86a].

A **normalized** set of constraints is an ordered set of constraints $\{(x_i \neq u_i), 1 \le i \le m\}$ such that: $\forall i \in \{1,...,m\}$, $\forall\ l \ge i$, $x_i \in X$ and $x_i \notin V(u_l)$.

We recall the main result:

Theorem 3.5

Let $C = \{t_i \# t'_i, 1 \le i \le k\}$ be an inequation system such that:

- $(\cup_{1 \le i \le k} V(t_i)) \cap (\cup_{1 \le i \le k} V(t'_i)) = \varnothing$
- *$\forall i$, t'_i is linear*

Let D be the set of variables occuring in a right hand side of an inequation.Then, there exists an algorithm which computes a set A of pairs (σ,c) where σ is a D-linear substitution and c is a normalized set of constraints with, for every (x,u) in c,

$V(u) \subseteq Im(\sigma)$ and such that:

(i) Completeness: if θ is a solution in $T(X)$ of the system and θ is linear, then $\exists (\sigma,c) \in A, \exists \omega \in \Omega, \theta = \omega \circ \sigma$ and, $\forall i$, $\omega(x_i) \neq \omega(u_i)$,

(ii) Soundness: $\forall \theta \in \Omega$, $(\forall i, \theta(x_i) \neq \theta(u_i)) \Rightarrow \theta \circ \sigma$ is a solution in $T(X)$ of the system.

Such constraints arise only when the inequation system has non-linear lhs.

For a single inequation, the algorithm may be sketched as follows:

We solve inequations on sequences of terms; given two terms t and t'the solutions of t#t' are the solutions of the multi-inequation <t> # <t'>:

1- Replace U and V in U # V by the sequences U' and V' where the common parts are removed. (SIMPLIFICATION). If a clash occurs, then return the identity.

2- If x is a variable which occurs only once in U remove it from U and remove the corresponding term in V.

3- If some term in U is not a variable, then replace the corresponding variable in V by all the possible constructions (SPLIT) and apply recursively the algorithm to each new multi-inequation.

4- Every term in U is a variable which occurs at least twice. Assume that the first terms in U are equal variables: u1 = u2. Then apply recursively the algorithm to U-{u1}#V-{v1} and to U-{u2}#V-{v2}. Add moreover the solutions of v1##v2.

We say that σ is a solution of v1##v2 iff σ(v1) and σ(v2) are not unifiable. (The two definitions of inequations solutions could be put together into only one definition by precising the set of variables D to include or not the variables of v1).

v1##v2 is solved using the following algorithm which supposes that v1 and v2 are linear terms which do not share any variable (this is the case here):

5- If both sides are variables then return the constraint $v1 \neq v2$.

6- If one side is a variable (v1 for example) and not the other side, then replace v1 by all the possible constructions (SPLIT) and apply recursively the algorithm.

7- If v1 and v2 are both non-variable terms and if they have not the same root, then return the identity (CLASH).

8- If v1 = v2 then return an empty set.

9- If v1 = f(u1,...,um) and v2 = f(w1,...,wm) then solve each ui##wi.

Now, we want to use this new formulation, as well as the anti-unification algorithm in order to get another expression of NF which could be used for the decision of the inductive-reducibility property.

4. Transformation of the fixed point equation

Let us show how the algorithm works on a very simple example. Let $\Sigma = \{0, p, s\}$ and the rules be:

$s(p(x)) \rightarrow x$; $p(s(x)) \rightarrow x$.

This is a specification of the integers. The TRS is canonical.

We start from the equation:

NF = ({0} ∩ Complement (LHS)) ∪ (p(NF) ∩ Complement (LHS)) ∪ (s(NF) ∩ Complement (LHS)).

Each intersection is computed by solving an inequation system:

- { p(s(x)) # 0; s(p(x)) # 0} for the first one
- {p(s(x)) # p(y); s(p(x)) # p(y) } for the second one
- {p(s(x)) # s(y) ; s(p(x)) # s(y) } for the last one.

Each inequation must be solved in NF. The first inequation system has the identity as a solution ($NF_0 = \{0\}$).

The second system is equivalent to the inequation p(s(x)) # p(y) which has 2 solutions: $y \leftarrow 0$ and $y \leftarrow p(z)$, with the condition : y is replaced by a term of NF.

This may be written: $NF_p = \{p(0)\} \cup p(NF_p)$. In the same way we find that $NF_s = \{s(0)\} \cup s(NF_s)$.

Finally NF is described by the three relations:

$$NF = \{0\} \cup NF_s \cup NF_p ,$$
$$NF_p = \{p(0)\} \cup p(NF_p) ,$$
$$NF_s = \{s(0)\} \cup s(NF_s).$$

To be strict, we consider this grammar as a grammar on words. Parentheses should be omitted and are only present for sake of readiness.

4.1 The case of linear lhs: computation of a context-free grammar.

In the general case, we need a more complicated form for the description of the language. Indeed, in the above example we have a regular grammar, if viewed as a grammar on words, but, in general, the grammar may be only context-free (when every lhs of a rule is linear) or even context-sensitive in the general case.

However let us first show how the algorithm works when every LHS of a rule is linear.

We compute a set P of productions $t \to u_1 | \ldots | u_m$ where $t,u_1,\ldots,u_n \in T(X)$.
Such a production will sometimes be written as an equality between the generated languages: if $t = f(t_1,\ldots,t_n)$ and, for every index $1 \le j \le m$, $u_j = f(u_{1,j},\ldots,u_{n,j})$, then we may write $NF_t = \cup_{1 \le j \le m} f(NF_{u1,j},\ldots,NF_{un,j})$ instead of $t \to u_1 | \ldots | u_m$. In other words, we will consider the term t as a non-terminal of the grammar, making the confusion with NF_t.

Q is a queue such that, for each term $t \in Q$, the productions having t as their left hand side are to be defined. Initially, $Q = \{ f(x^*),\ f \in \Sigma\}$, where x^* is a sequence of k fresh variables, where k is the arity of f.

Initially, $P = \{ X \to u , u \in Q\}$. This means that every ground normal form begins with a symbol of Σ. $V_N = \{X\}$. V_N will correspond to the non-terminal vocabular.

While Q is not empty repeat:

1- Take the first element in the queue: t ; remove t from Q.

2- Solve the inequation system $\{l_i \# t,\ 1 \le i \le n\ \}$ (where $LHS = \{l_i , 1 \le i \le n\}$), let S be the set given by the anti-unification algorithm. S is a set of substitutions since every LHS of a rule is linear.

3- Let f be the root of t: $t = f(t_1,\ldots,t_n)$.
Let $S = \{\sigma_1,\ldots,\sigma_m\}$ and, $\forall i \in \{1,\ldots,n\}$, $\forall j \in \{1,\ldots,m\}$, $u_{i,j} = \sigma_j(t_i)$.
Add $t \to f(u_{1,1},\ldots,u_{n,1}) | \ldots | f(u_{1,m},\ldots,u_{n,m})$ to P and add t to V_N.

4- For each $u_{i,j}$, $1 \le i \le n$, $1 \le j \le m$,
if it exists a renaming of variable θ such that $\theta(u_{i,j}) \in (V_N \cup Q)$
then replace $u_{i,j}$ by $\theta(u_{i,j})$
else add $u_{i,j}$ to Q.

Let now $V_T = \Sigma$ (or , more precisely, the functional symbols of Σ).

Theorem 4.1 (main theorem of the section)
Assume that every lhs of a rule is linear. Then, the above algorithm computes a set P of productions , a set V_N of non-terminals , a set V_T of terminal symbols and an axiom X such that NF is the language generated by the grammar (V_N,V_T,P,X). Moreover, viewed as a grammar on words, this grammar is context-free.

Proof:

This is a particular case of theorem 4.2.

Let $G = (V_N,V_T,P,X)$. G is a grammar by construction (this corresponds to the points (i), (ii) and (iii) of theorem 4.2.). Moreover, the language L(G) generated by G satisfies:

- $L(G) \subseteq NF$. This is indeed the soundness property of theorem 4.2 and is a consequence of the proposition 3.4 and the soundness property of theorem 3.5 (see the proof of theorem 4.2 for more details)

- $NF \subseteq L(G)$. This is the completeness property of theorem 4.2. This is a consequence of the proposition 3.4 and the completeness property of the theorem 3.5.

Finally, the algorithm stops (see the proof of theorem 4.2)

Example:

We shall use in this example the notations of languages instead of the notations of grammars in order to avoid the confusion between the rules of the TRS and the rules of the productions in the grammar.

We take as an example the lists (of elements of sort s) with three operators: nil:→list(s) ; unit : s→list(s) ; append : list(s) × list(s) → list(s).

These operators will be also denoted by nil,u,a respectively.

The rules are:

$a(nil,x_1) \rightarrow x_1$

$a(u(y_1),nil) \rightarrow u(y_1)$

$a(a(u(y_1),x_1),x_2) \rightarrow a(u(y_1),a(x_1,x_2))$

We write first $NF = NF_{nil} \cup NF_u \cup NF_a$

Initially, $Q = \{ nil; u(y); a(x_1,x_2)\}$

Since the systems {1 # nil, 1 lhs of a rule} and {1 # u(y), 1 lhs of a rule} have no solution, the two terms nil and u(y) are removed from Q and A becomes:

$\{ NF_{nil} = \{nil\}; NF_u = u(NF1) \}$ (NF1 is the set of ground terms of sort s)

The variables in $a(x_1,x_2)$ are renamed into x_3 and x_4 respectively. Then, the resolution of the system $\{ a(nil,x_1) \# a(x_3,x_4); a(u(y_1),nil) \# a(x_3,x_4) ; a(a(u(y_1),x_1),x_2) \# a(x_3,x_4) \}$ gives the solutions:

$(x_3 \leftarrow u(y_2); x_4 \leftarrow u(y_3))$; $(x_3 \leftarrow u(y_2); x_4 \leftarrow a(x_5,x_6))$; $(x_3 \leftarrow a(nil,x_5))$;

$(x_3 \leftarrow a(a(x_5,x_6),x_7))$;

After this step, $Q = \{ a(nil,x_5) ; a(a(x_5,x_6),x_7) \}$. Indeed the four first terms are already left hand sides of pairs of A (up to renaming). And the following equation is added to A:

$NF_a = a(NF_u,NF_u) \cup a(NF_u,NF_a) \cup a(NF_{a(nil,x5)},NF) \cup a(NF_{a(a(x5,x6),x7)},NF)$.

Now, since there is no solution to the inequation system $\{ 1 \# a(nil,x_5)$, 1 lhs of a rule$\}$, the term $a(nil,x_5)$ is removed from Q. The equation $NF_{a(nil,x5)}=\varnothing$ is added to A.

At the next step, the system {1 # $a(a(x_5,x_6),x_7)$, 1 lhs of a rule} is solved. This gives the solutions: $(x_5 \leftarrow nil)$ and $(x_5 \leftarrow a(x_8,x_9))$. This leads to add the new equation to A:

$NF_{a(a(x5,x6),x7)} = a(NF_{a(nil,x6)},NF) \cup a(NF_{a(a(x8,x9),x6)},NF)$.

Then no term is added to Q since $a(nil,x_6)$ and $a(a(x_8,x_9),x_6)$ are both already a left hand side of A (up to renaming). The result is thus the grammar:

$NF ::= NF_{nil} \mid NF_u \mid NF_a$

$NF_{nil} ::= nil$

$NF_u ::= u\ NF1$

$NF_a ::= a\ NF_u NF_u \mid a\ NF_u NF_a \mid aNF_{a(nil,x5)}NF \mid aNF_{a(a(x5,x6),x7)}NF.$

$NF_{a(nil,x5)} ::= \varnothing$

$NF_{a(a(x5,x6),x7)} ::= aNF_{a(nil,x6)}NF \mid aNF_{a(a(x8,x9),x6)}NF.$

It is then reduced (see the section 5.1); only the productions from which can be derived a terminal word are kept . We obtain thus the result:

$NF ::= NF_{nil} \mid NF_u \mid NF_a$

$NF_{nil} ::= nil$

$NF_u ::= u\ NF1$

$NF_a ::= a\ NF_u NF_u \mid a\ NF_u NF_a$

Obviously, this result is parameterized by the set NF1 of ground normal forms of sort s.

4.2. The general case

Let us now return to the general case.

The idea is thus to describe NF by a set of pairs (t,NFR_t). The set of such terms t may be viewed as the non-terminals of the grammar. NFR_t is then composed by substitutions on t (possibly together with constraints). These substitutions give all the "profiles" of instances of t which may be in normal form. Again the subterms of $\sigma(t)$ must be in normal form and are thus lhs in the description of NF (point (i) below). The terminal symbols are the symbols of the signature (the variables are not terminal symbols) and the axiom may be viewed as a free variable. However, it is still unclear how to use this kind of descriptions in the general case.

Theorem 4.2

There exists an algorithm which computes a description of the set NF of ground normal forms. This description consists of a finite set R of pairs $\{(t, NFR_t)\}$ *where:*

- *t is a linear term*
- NFR_t *is a set of pairs* $\{(\sigma, c)\}$ *where* σ *is a substitution such that* $Dom(\sigma) \subseteq V(t)$ *and c is a normalized set of constraints laying on* $Im(\sigma)$.

A simpler version of this theorem is given by the proposition 4.1., when every lhs of a rule is linear.

The algorithm may be sketched as follows:

Q is a queue such that, for each term $t \in Q$, the set NF_t is to be defined. Initially,

$Q = \{ f(x^*),\ f \in \Sigma\}$, where x^* is a sequence of k fresh variables, where k is the arity of f.

A is the set to be computed. Initially, R = $\emptyset$.

While Q is not empty repeat:

1- Take the first element in the queue: t ; remove t from Q.

2- Solve the inequation system $\{l_i \# t,\ 1 \le i \le n\}$ (where LHS = $\{l_i, 1 \le i \le n\}$), let S be the set given by the anti-unification algorithm.

3- Add (t,S) to R.

4- For each $(\sigma,c) \in S$, and for each $u \in Sub(\sigma(t))$, if

(i) $u \notin X$

(ii) $\forall \theta$ a renaming of variables, $\theta(u)$ is not yet a lhs of an element of R

(iii) $\forall \theta$ a renaming of variables, $\theta(u) \notin Q$,

then add u to Q

Note that u is always linear and we may thus use the anti-unification algorithm.

Proof of the Theorem :

Proof of correctness :

We prove that the description satisfies the following properties :

1) Syntactical part :

(i) The subterms of a term in normal form are again in normal form:

$\forall (t,NFR_t) \in R,\ \forall (\sigma,c) \in NFR_t,\ \forall u \in Sub(t),\ (\sigma(u), NFR_{\sigma(u)}) \in R$

(ii) The description contains all the basic constructions:

$\forall f \in \Sigma,\ (f(x_1, \ldots, x_n), NFR_f) \in R$

(iii) The description contains only terms proceeding from the two cases above:

$\forall (t,NFR_t) \in R,$

$(\forall u \in Sub(t), u \text{ is a variable})$ *or* $(\exists (t',NFR_{t'}) \in R, \exists (\sigma,c) \in NFR_{t'}, \exists u \in Sub(t'),\ \sigma(u) = t)$

2)Soundness part :

(iv) $\forall \theta \in \Omega_g,\ \forall (t,NFR_t) \in R,\ \forall (\sigma,c) \in NFR_t,$

$\left(\begin{array}{l} \forall (x,w) \in c,\ \theta(x) \neq \theta(w) \\ \\ \forall u \in Sub(t),\ \theta(\sigma(u)) \in NF \end{array} \right) \Rightarrow \theta(\sigma(t)) \in NF$

3)Completeness part :

(v) if $u \in NF_f$, *then* $\exists (\sigma,c) \in NFR_f,\ \exists \theta \in \Omega_g$, *such that:*

- $u = \theta(\sigma(f(x_1, \ldots, x_n)))$,
- $\forall (x,w) \in c,\ \theta(x) \neq \theta(w)$
- $\forall i \in \{1, \ldots, n\},\ \theta(\sigma(x_i)) \in NF.$

The proof of the points (i) to (iii) are left to the reader.

(iv) The soundness property is an invariant of the algorithm: when (t,S) is added to A,

$\forall \theta \in \Omega_g,\ \forall (\sigma,c) \in S,\quad (\forall (x,u) \in c,\ \theta(x) \neq \theta(u)) \Rightarrow \theta \circ \sigma$ is a solution in T(X) of the inequation system $\{l_i \# t\}$.

This follows from the soundness property in theorem 3.5.

Thus, $\theta \circ \sigma$ is not unifiable with any l_i, and no rule may be applied at the root of $\theta(\sigma(t))$.

If, moreover, $\forall u \in Sub(t)$, $\theta(\sigma(u)) \in NF$, then no rule may be applied to $\theta(\sigma(t))$. And, finally, $\theta(\sigma(t)) \in NF$.

(v) In the same way, the completeness is a consequence of the proposition 3.4 and the completeness property in theorem 3.5. The details are left to the reader

Proof of termination:

Let $d = \Sigma_{1 \le i \le m}(depth(l_i))$. It is sufficient to prove that every term added to Q during the computation has a depth smaller than d. Then the termination is insured since there is only a finite set of terms having a depth smaller than d, up to renaming.

It is easy to prove by induction that every term added to Q has a depth smaller than d:

- this is true initially
- when solving an inequation $t \# t'$, for every solution σ, $\sigma(t')$ has a depth smaller than $1+\max(depth(t),depth(t'))$. When solving an inequation system $t_i \# t'$, for every solution σ, $\sigma(t')$ has a depth smaller than $1 + \max(depth(t'), \Sigma_{1 \le i \le m}(depth(t_i)))$. (see [Com87] for more details)

Thus,

$depth(t) \le d \Rightarrow$ for every solution σ of the system $\{ l_i \# t, 1 \le i \le m \}$, $depth(\sigma(t)) \le 1+d$.

And, $\forall u \in Sub(t)$, $depth(\sigma(u)) \le d$.

Practically, the number of elements in A is much smaller than the theoretical maximum. We show in the appendix how works the algorithm on a more complicated example, including non-linear and deep lhs.

The difficulty is now to *reduce* the description of NF. When every lhs of a rule is linear, this means exactly to reduce the corresponding grammar. Moreover, it is interesting to erase productions which can never be applied. In our case, to erase from A the pairs (t,NFR_t) such that NF_t is empty. This is the aim of the next section.

5. Applications

5.1 The case where every lhs of a rule is linear.

Let G be the grammar computed by the algorithm of section 4.1.

G being context-free, we may reduce the grammar, i.e. remove every production $A \rightarrow \alpha$ such that the language generated by A is empty. For example, it is possible to apply the following procedure :

Initially, $P_0 = \{ A \rightarrow \alpha \in P, \alpha \in V_T^* \}$, $V_0 = \{A \in V_N, \exists A \rightarrow \alpha \in P_0\}$ and $i = 0$.

Repeat

$P_{i+1} := \{A \to \alpha \in P, \alpha \in (V_T \cup V_i)^*\}$;

$V_{i+1} := \{A \in V_N, \exists\ A \to \alpha \in P_{i+1}\}$;

$i := i+1$

Until $P_{i-1} = P_i$

The grammar (V_i, V_T, P_i, X) is then equivalent to G and is reduced. Such a result is well known in language theory.
In other words: every term t which belongs to V_N and do not belong to V_i is inductively reducible. Every term of V_i is not inductively reducible.

In the same way, we may decide whether NF is finite or not . Let $G=(V_N, V_T, P, X)$ be a reduced context-free grammar such that L(G) = NF. For example, we may use the procedure:

Initially, $B_0 = \varnothing$ and $i = 0$.

Repeat

$B_{i+1} := \{ t \in V_N, \forall\ t \to \alpha \in P, \alpha \in (V_T \cup B_i)^*\}$;

$i := i+1$

Until $B_i = B_{i-1}$.

Then NF is finite iff $B_i = V_N$.

Proof:

The proposition " for every term t in B_i, the language generated by t is finite" is an invariant of the algorithm. Moreover, this proposition is true initially. Thus, if $B_i = V_N$ when the algorithm stops, then NF is finite.

Conversely, assume that the algorithm stops and that $V_N \neq B_i$. Now, we define a sequence of non-terminals t_n by induction: Let $t_0 \in V_N - B_i$.

Then, assuming that t_n is defined for some $n \geq 0$ and that $t_n \in V_N - B_i$, $\exists\ t_n \to \alpha_n \in P$, $\alpha_n \notin (V_T \cup B_i)^*$, let then t_{n+1} be a non-terminal of α_n which do not belong to B_i.

Since V_N is finite, there exist at least two integers p and q such that $t_p = t_q$. This may be written: $t_p \Rightarrow^+ \beta t_p \gamma$ where β or γ is not the empty string. The grammar being reduced, it exists some words δ and η in V_T^* such that $\beta \Rightarrow^* \delta$ and $\gamma \Rightarrow^* \eta$. Moreover, the language $L(t_p)$ is not empty. Let ζ be in $L(t_p)$. Then, the regular set $\delta^* \zeta \eta^*$ is included in $L(t_p)$. This implies in particular that $L(t_p)$ is infinite and thus NF is infinite.

We may summarize these results:

Proposition 5.1

When every lhs of a rule is linear, we may decide the finiteness and the emptyness of NF.

5.2. Computing NF_t when t is linear.

t being a term of T(X), we are interested in this section in the problem of finding a description of NF_t. The problem of finding a description of NF is a particular case of the latter since it is sufficient to take a variable for t. However, we do not want only to generalize the algorithm of section 4.2 since the generalisation would have probably a large complexity. At the opposite, we shall use the result of the section 4 and get a quick procedure for computing NF_t. In this sense, the computation of the section 4 is a "compilation" of the TRS for ground terms. Let R be the set of the theorem 4.2. The following algorithm computes NF_t by "unification" with the elements of R.

Q(t) is a queue like in the algorithm of the section 4. Initially, Q(t)={t}. R(t) is the set to be computed. Initially, R(t) is empty.

While Q(t) is not empty repeat:

1- Take the first element in the queue : u; remove u from Q(t).

2- Let f be the root of u and $(f(x1,\ldots,xn),NFR_f) \in R$.

Let $B_f=\{(\sigma(f(x1,\ldots,xn)),c), (\sigma,c)\in NFR_f\}$.Let B_u the set obtained by unifying u with the elements of B_f and normalizing the constraints:

$B_u = \cup_{\theta \text{ m.g.u of u and an element of } Bf}$ Normalize(θ,θ(c)).

(see [Com86a,b,c] for the normalisation procedure)

3- Add (u, B_u) to R(t).

The next steps are the same as in the section 4.

Note that we have only changed the step 2 in the algorithm of section 4. This was done in a simplification (of the complexity) sake: we wanted to use the set already computed.

Moreover, it is clear in this algorithm that the generalization to the case of non-linear terms t lays only on the generalization of the normalization procedure.

Finally, it could be simplified when every lhs of a rule is linear. Indeed, the normalisations are, in this case, unnecessary. In this case, the algorithm will lead once more to a context-free grammar (for which, in particular, the emptyness is decidable).

6. Conclusion

We have presented a method for the computation of a description of the set of ground terms in normal form. Such a computation seems to have a lot of interesting

applications. For right now, the interesting applications are shown in the case where every lhs of a rule is linear(i.e. with at most one occurrence of each variable). Even so, it is a very interesting result. The generalisation of the grammar reductions to the case of our general grammars is the subject of an ongoing research.
Moreover, we would like to show that the complexity of the decision of inductive reducibility (for example) is polynomial, when the grammar is already computed.
Finally,we conjecture that NF is an LL-language (if viewed as a grammar on words). Then, the equality of two ground sets of normal forms would be decided like the equivalence of two grammars. This has some interesting applications. For example, assume that the TRS is canonical and that, adding the rules $l_1 \rightarrow r_1, \ldots, l_n \rightarrow r_n$, the TRS remains canonical. Then $l_1 = r_1, \ldots, l_n = r_n$ are inductive theorems iff the two sets of ground normal forms are equal.

References

[AU72] A.V.Aho & J.D. Ullmann *The theory of parsing, translation and compiling*. Vol1: Parsing,Prentice hall series in automatic computation, 1972.

[Ber79] D. Bert. *La programmation générique*. Thèse, 1979

[Col82] A. Colmerauer. *PROLOG II, Manuel de référence et modèle théorique*. Research Report, GIA, Luminy, 1982.

[Com86a] H. Comon. *Sufficient Completeness, Term Rewriting System and Anti-unification*. Proc. 8th. Conference on Automated Deduction, Oxford, 1986.

[Com86b] H. Comon. *Anti-unification and proofs by induction*. RR LIFIA 51, IMAG 619. (July 1986).

[Com87] H. Comon. *About inequations simplification*. To appear as a LIFIA research report.

[G&B85] J.H. Gallier & R.V. Book. *Reductions in tree replacement systems* Theoretical Computer Science 37 (1985) pp 123-150.

[GTW78] J.A. Goguen, J.W. Thatcher & E.G. Wagner. *An Initial Algebra Approach to the Specification, Correctness and Implementation of Abstract Data Types*. Current Trends in Programming Methodology, vol. 4., pp 80-149, Prentice Hall (1978).

[HH82] G. Huet & J.M. Hullot. *Proofs by Induction in Equational Theories with Constructors*. Journal of Computer and System Science 25-2 (1982).

[HKi84] H. Kirchner *A general inductive completion algorithm and application to abstract data types*. Proc. 7th Conference on Automated Deduction, Napa, 1984 (LNCS 170)

[JK86] JP. Jouannaud & E. Kounalis. *Proofs by Induction in Equational Theories without Constructors*. Proc. IEEE Symposium on Logic in Computer Science, Cambridge, Mass. (1986).

[KM86] D. Kapur & D. Musser *Inductive reasoning with incomplete specifications*. Proc.IEEE Symposium on Logic in Computer Science, Cambridge, Mass. (1986)

[KNZ85] D. Kapur, P. Narendran & H. Zhang. *On sufficient completeness and related properties of Term Rewriting Systems*. General Electric Company, Preprint October 1985.

[KNZ86] D. Kapur, P. Narendran & H. Zhang. *Complexity of Sufficient Completeness*. To appear in TCS.

[LLT86] A. Lazrek , P. Lescanne & JJ. Thiel. *Proving inductive equalities. Algorithms and implementation*. RR CRIN, Nancy, 1986.

[MG86] J. Meseguer & J.A. Goguen. *Initiality, Induction and Computability*. in "Application of Algebra to Language Definition and Compilation", M. Nivat & J. Reynolds eds. Cambridge U.P. (1986)

[Pau84] E. Paul *Proof by induction in equational theories with relations between constructors*. Proc. 9th. Colloquium on trees in Algebra and Programming, Bordeaux, 1984.

[Pla85] D. Plaisted *Semantic Confluence Tests and Completion Methods*. in Information and Control 65 (1985) pp 182-215.

[RV82] J.L. Rémy & P.A.S. Veloso *Comparing Data Type Specifications via Their Normal Forms* International Journal of Computer and Information Sciences 11, Nº 3 (1982) pp 141-153.

[Thi84] JJ. Thiel. *Stop loosing sleep over incomplete data type specifications*. Proc. ACM Conference on Principles of Programming Languages, Salt Lake City, 1984.

Appendix: a complete step by step development on a non-trivial example

In this example, which includes non-linear lhs, we shall use a presentation as friendly as possible, in particular, we shall use the notations of languages when there is no constraints.

$\Sigma = \{ 0 : \rightarrow s \ ; \ f: s \times s \rightarrow s \ ; \ g : s \rightarrow s \}$

Rules:

$f(x,0) \rightarrow 0$

$f(x,f(g(x),y)) \rightarrow x$

$f(x,g(x)) \rightarrow x$

$f(0,f(0,x)) \rightarrow 0$

The TRS is canonical : the finite termination is obvious and there is no critical pair . (The proof is left to the reader ; actually, in what follows, we are not concerned with the right-hand sides of the system). We use our algorithm (section 4.2) for computing a "description" of NF:

Initially: $Q_0 = \{0, g(x1), f(x1,x2)\}, \ R_0 = \{\}$;

$\{l_i \# 0, 1 \leq i \leq m\}$ has the identity as a solution, thus, at the first step,

$Q_1 = \{ g(x1), f(x1,x2)\}, \ R_1 = \{ NF_0 = \{0\}\}$

$\{l_i \# g(x_1), 1 \leq i \leq m\}$ has the identity as a solution, thus, at the second step,

$Q_2 = \{ f(x1,x2)\}, \ R_2 = \{ NF_0 = \{0\}; \ NF_g = g(NF) \}.$

$\{l_i \# f(x1,x2),\ 1 \le i \le m\}$ has 5 solutions given by the anti-unification algorithm: s1 = (x2←g(x3); x3≠x1), s2 = (x1← g(x4); x2 ← f(0,x3)), s3 = (x2← f(f(x3,x4),x5)), s4 = (x2 ← f(g(x3),x4); x3≠x1), s5 = (x1 ← f(x3,x4); x2← f(0,x5)).Thus, at the third step,

$Q_3 = \{ f(0,x3),\ f(f(x3,x4),x5),\ f(g(x3),x4)\}$,

$R_3 = \{ NF_0 = \{0\};\ NF_g = g(NF)$;

$NF_f = \{f(t1,g(t3)),\ g(t3) \in NF_g,\ t1 \in NF,\ t3 \neq t1\}$

$\cup\ \{f(g(t4),f(0,t3)),\ f(0,t3) \in NF_{f(0,x3)},\ g(t4) \in NF_g\}$

$\cup\ \{f(t1,f(f(t3,t4),t5)),\ f(f(t3,t4),t5) \in NF_{f(f(x3,x4),x5)},\ t1 \in NF\}$

$\cup\ \{f(t1,f(g(t3),t4)),\ f(g(t3),t4) \in NF_{f(g(x3),x4)},\ t1 \in NF,\ t1 \neq t4\}$

$\cup\ \{f(f(t3,t4),f(0,t5)),\ f(t3,t4) \in NF_f,\ f(0,t5) \in NF_{f(0,x3)}\}$

$\}$

$\{l_i \# f(0,x3),\ 1 \le i \le m\}$ leads to 3 solutions : s6 = (x3←g(x4); x4 ≈ 0), s7 = (x3← f(f(x4,x5),x6)), s8 = (x3← f(g(x4),x5); x5 ≠ 0).Thus, at the fourth step,

$Q_4 = \{f(f(x3,x4),x5),\ f(g(x3),x4)\}$

$R_4 = R_3 \cup$

$\{NF_{f(0,x3)} = \{\ f(0,\ g(t4)),\ g(t4) \in NF_g,\ t4 \neq 0\ \}$

$\cup\ \{f(0,f(f(t4,t5),t6)),\ f(f(t4,t5),t6) \in NF_{f(f(x3,x4),x5)}\}$

$\cup\ \{f(0,f(g(t4),t5)),\ f(g(t4),t5) \in NF_{f(g(x3),x4)},\ t5 \neq 0\}\ \}$

$\{l_i \# f(f(x3,x4),x5),\ 1 \le i \le m\}$ leads to 4 solutions: s9 = (x5 ← g(x6); x6 ≠ f(x3,x4)), s10 = (x5 ← f(0,x6)), s11 = (x5 ← f(f(x6,x7),x8)), s12 = (x5 ← f(g(x6),x7); x6 ≠ f(x3,x4)). Thus at the fifth step,

$Q_5 = \{f(g(x3),x4)\}$

$R_5 = R_4 \cup$

$\{NF_{f(f(x3,x4),x5)} =$

$\{f(f(t3,t4),g(t6)),\ f(t3,t4) \in NF_f,\ g(t6) \in NF_g,\ t6 \neq f(t3,t4)\}$

$\cup\ \{f(f(t3,t4),f(0,t6)),\ f(t3,t4) \in NF_f,\ f(0,t6) \in NF_{f(0,x3)}\}$

$\cup\ \{f(f(t3,t4),f(f(t6,t7),t8)),\ f(t3,t4) \in NF_f,\ f(f(t6,t7),t8) \in NF_{f(f(x3,x4),x5)}\}$

$\cup\ \{f(f(t3,t4),f(g(t6),t7)),\ f(t3,t4) \in NF_f,\ f(g(t6),t7) \in NF_{f(g(x3),x4)},\ t6 \approx f(t3,t4)\}$

$\{l_i \# f(g(x3),x4),\ 1 \le i \le 4\}$ leads to 4 solutions: s13 = (x4 ← g(x5); x5 ≠ g(x3)), s14 = (x4 ← f(0,x5)), s15 = (x4 ← f(f(x5,x6),x7)), s16 = (x5 ← f(g(x5),x6); x5 ≠ g(x3)). Thus, at the last step,

$Q_6 = \{\}$

$R_6 = R_5 \cup$

$\{NF_{f(g(x3),x4)} =$

$\{f(g(t3),g(t5)),\ g(t3) \in NF_g,\ g(t5) \in NF_g,\ t5 \neq g(t3)\}$

$\cup\ \{f(g(t3),f(0,t5)),\ g(t3) \in NF_g,\ f(0,t5) \in NF_{f(0,x3)}\}$

$\cup\ \{f(g(t3),f(f(t5,t6),t7)),\ g(t3) \in NF_g,\ f(f(t5,t6),t7) \in NF_{f(f(x3,x4),x5)}\}$

$\cup\ \{f(g(t3),f(g(t4),t5)),\ g(t3) \in NF_g,\ f(g(t4),t5) \in NF_{f(g(x3),x4)},\ t5 \neq g(t3)\}$

$\}$

The algorithm stops since Q_6 is empty.
The reader may note that, even in this complicated example, only 6 steps are needed. Thus, it seems that the complexity is not too large.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique