Automatizing Termination Proofs of Recursively Defined Functions

Pascal MANOURY Marianne SIMONOT

Equipe de logique
CNRS URA 753
UFR de mathématiques, 45-55, 5^{et}
Université Paris VII
2 place Jussieu, 75251 Paris cedex 05
email : eleph@logique.jussieu.fr
email : simonot@logique.jussieu.fr

Abstract

This paper describes a fully automated strategy for proving termination of functions defined by recursive equations. This strategy has been implemented in the ProPre program synthesis system as one of the mechanism that leads from specifications of functions to programs. It has been successfully tested on the Ackerman function, a functional specification of the quick-sort and various other standard functional algorithms.

Introduction

The context of this work

Our work was originaly motivated by the paradigm of *Programming by Proofs.* Using this approach, programs are extracted from an (intuitionistic) proof that a given logical specification is satisfied. To be more precise, in the framework that we studied, programs are λ-terms extracted from the proof that a given function, defined by a set of equations, always terminates. That is to say : a proof that the function is totaly defined on its domain. The theoretical foundations of this way to obtain *correct programs* can be found in [1], [2] and [5].

Over this foundations, our aim was to build a (functional) programming language. To realize it, we turn our attention to *automatizing termination proofs.* One important theoretical domain concerned with termination of equationally defined functions is the investigation of the termination of *rewriting systems.*
But, because of the use we want to make of our proofs (namely : extract a λ-term), techniques used in rewriting theory cannot be directly used. There is two reasons for this :
- the first one is that we don't want to have only a termination proof, but also a *proof of totality.* That is to say that we don't want only to prove that for any input, the program realizing the function will give an output, but we want also prove that, for instance, for any integer, the program will return an integer as the result of the computation.
- the second one is that the proof we are looking for, in order to extract from it a λ-term (via the Curry-Howard correspondence), must be expressed in a *formal logical framework* (namely, the natural deduction).

Termination arguments used for rewriting systems don't satisfy those two requisits. That's why we had to design an original method appropriate to our precise purpose. In particular, we specially investigated the use of primitive recursive induction as expressed in, say, Peano's arithmetic which can be generalized to any data types defined in terms of multi-sorted algebræ.

The content of this work

We will now focus our attention on the prover itself regardless of the question of how the program synthesis is made. The reader interested in computer applications is referred to [3]. In order to have a general purpose, we will express proofs in a first order predicate calculus theory where data types and functions over them are described by axioms.
Of course, since the termination problem is undecidable and since our proof search strategy is fully automated and always terminates, we have to restrict our ambition. We express two kinds of restrictions. The first one concerns the equational data : in 1.3 we define the notion of *equational data for (a function) f.* The second one concerns the shape of the proof : in 3 we define the notion of *R-proof.*
As our strategy builds step by step proofs we can say that it proceeds by backward reasoning in three successive steps :

1. it uses induction as often as necessary ;

2. then it uses equations ;

3. finally it uses induction hypothesis and termination theorems of auxiliary functions to achieve the proof.

In fact, because of the very structure of R-proofs, the problem of finding such a proof can be restricted to the problem of finding the right inductions. This restriction is formally expressed by the definition of what we call the *right terminal state* (see 4.1). Furthermore, the analysis of R-proofs provides us with two properties from which we can design an algorithm satisfying our expectation.
The first one that we call the *decreasing property* concerns the equational data and is defined in 4.2. The second one is a criterium for choosing the *suitable variable* on which to induct and is defined in 4.3. The algorithm is described in 4.4.
We prove in 5 the equivalence between the right terminal state property and the conjunction of the decreasing property and the suitable variable. This equivalence implies that the algorithm based on these two properties is sound and complete. Section 6 describes some experiments with the prover.

1 The theory

1.1 The language

We present in this section the first order language in which will be expressed axioms defining data types and equations defining functions.

1.1.1 The individual terms

We take for the set of individual variables the set ω^ω of the finite sequences of integers. The point mark (".") will denote the concatenation and ε the empty sequence. We will adopt the following typographic convention to represent individual variables : the expression x_{121} will represent the variable 1.2.1, x_{iq} will denote the variable $\mathsf{i.q}$ and so on.

To build terms upon this set of variables we consider two sets of functional symbols : $\mathcal{C}$ and $\mathcal{F}$ which contain respectively the names of constructors of data types (see 1.2.1) and the names of users defined functions (see 1.3). We want our terms to be sorted. So let $\mathcal{D}$ be the set of the names of those sorts. To each element f of $\mathcal{C}$ or $\mathcal{F}$ must be attached a tuple of elements of $\mathcal{D}$ which will be written $D_1, ..., D_n- > D$. This tuple attached to each functional symbol is an inprovement of the notion of arity and must be regarded as contraints for building well formed terms. As the expression $D_1, ..., D_n- > D$ naively denotes the *"type"* of f we will speak of $D_1, ..., D_n- > D$ as the type of f. But this naive notion of type must not be confused with the one used in

type theory or in typed λ-calculus systems.
So let $\mathcal{U}$ be the set of usual first order terms build upon ω^ω, $\mathcal{C}$ and $\mathcal{F}$. Now let α be a function from ω^ω to $\mathcal{D}$. Because elements of $\mathcal{C} \cup \mathcal{F}$ are sorted, α can obviously be extend in a partial function from $\mathcal{U}$ to $\mathcal{D}$. So, our set of individual terms will be the subset $\mathcal{T}$ of $\mathcal{U}$ such that for all $t \in \mathcal{U}$ there exists α such that $\alpha(t)$ is defined.

We use $\mathcal{V}ar(t)$ to denote the set of all variables occurring in a term t.
We say $t \in \mathcal{T}$ *is of type* D iff $\alpha(t) = D$. If u is a sub-term of t, then u *is of type* D' *in* t iff $\alpha(u) = D'$.
We define the set $\mathcal{A}c(D)$ as the set of terms of type D such that all functional symbols occurring in them are elements of $\mathcal{C}$.

Let us now introduce the notion of the *address* of a term u in t.

Definition 1.1 *Let* $Adr : \mathcal{T} \times \mathcal{T} \longrightarrow \omega^\omega$ *be a relation such that :*

- *if* $t = u$ *then* $Adr(u, t) = \varepsilon$ *else*
- *if* $t = f(t_1, \dots, t_n)$ *and there exists* i $(1 \le i \le n)$ *such that* $Adr(u, t_i)$ *is defined, then* $Adr(u, t) = i.Adr(u, t_i)$.

We say that k *is* the address of u in t *or that* u *is* of address k *in* t *iff* $Adr(u, t) = \mathsf{k}$. *We will denote* u *by* t_{k}.

Note that Adr will define a function if it is restricted to the domain $\mathcal{T}' \times \mathcal{T}$ where $\mathcal{T}'$ denotes the subset of $\mathcal{T}$ of linear terms (*eg* : terms in which each variable has at most one occurence).
We now consider a particular subset of terms that we call *well named terms*. Remember that ω^ω is our set of variables.

Definition 1.2 *Let* $t, u \in \mathcal{T}$, *then* t *is* a well named term *iff for each variable* $\mathsf{x}_{\mathsf{k}} \in \mathcal{V}ar(t)$, $Adr(\mathsf{x}_{\mathsf{k}}, t) = \mathsf{k}$. *The pair* (t, u) *is a* well named pair of terms *iff* t *is a well named term and* $\mathcal{V}ar(u) \subseteq \mathcal{V}ar(t)$.

Note that well named terms are always linear.

Remark 1.1 *If* (t, u) *is a well named pair of terms, and if* $\mathsf{x}_{\mathsf{k}} \in \mathcal{V}ar(u)$ *then* $\mathsf{x}_{\mathsf{k}} \in \mathcal{V}ar(t)$, *and, as* t *is a well named term,* $t_{\mathsf{k}} = \mathsf{x}_{\mathsf{k}}$.

Matching and well named terms Recall that a term t *matches* a term u iff there exists a substitution σ of the set of variables such that $\sigma(t) = u$. We say, in this case, that t matches u *according to* σ.

Remark 1.2 *Let* t *be a term which matches* u *according to* σ. *Then for all* $\mathsf{k} \in \omega^\omega$ *such that* t_{k} *exists,* $\sigma(t_{\mathsf{k}}) = u_{\mathsf{k}}$.

In particular, if t is a well named term and $\mathsf{x}_{\mathsf{k}} \in \mathcal{V}ar(t)$ then $\sigma(\mathsf{x}_{\mathsf{k}}) = u_{\mathsf{k}}$.

1.1.2 The formulæ

Our logical language is the one of first order predicate calculus with equality. At this logical level, we consider $\mathcal{D}$ as a set of unary predicate constants. Formulæ are inductively defined as usual on $\mathcal{T}$, $\mathcal{D} \cup \{=\}$ and the logical symbols $\rightarrow$ and $\forall$. We use the usual notions of free occurrence of a variable in a formula and the notion of substitution : $\varphi[t/x]$ denotes the formula obtained by the substitution of t for all the free occurrences of x in φ.

1.2 The axioms of the theory

1.2.1 Axioms for data types

Data types are given by means of a predicate constant (the name of the data type), a set of functional symbols choosen in $\mathcal{C}$ (the names of constructors), a set of first order formulæ (the types of the constructors) and an axiom schema (the induction principle).

For instance, suppose $N \in \mathcal{D}$ is the name for the data type of the natural numbers, then the data type of lists of natural numbers is defined by :

- the predicate constant LN ;
- the functional symbols Nil and $Cons$ with respective types LN and $N, LN \to LN$.
- the first order axioms $LN(Nil)$ and $\forall n(Nn \to \forall x(LNx \to LN(Cons(n, x))))$;
- the axiom schema

$$\forall x(LNx \to (\theta[Nil/x] \to \forall n(Nn \to \forall y(LNy \to (\theta[y/x] \to \theta[Cons(n, y)/x])) \to \theta[x])))$$

  where $\theta[x]$ is a formula in which the variable x occurs free.

We assume that all the types of constructors are of the form $D_1, \ldots, D_m, D, \ldots, D \to D$, where for all i $(1 \leq i \leq m), D_i \neq D$.

Let $\mathcal{TH}$ be a set of formulæ, containing all the first order axioms which describe the constructors of data types. This set can be completed by including all the formulæ expressing the totality of functions whose proofs has been made. We shall suppose that elements of $\mathcal{TH}$ are all of the form $\forall \mathsf{x}_1(D_1\mathsf{x}_1 \to \ldots \forall \mathsf{x}_\mathsf{n}(D_n\mathsf{x}_\mathsf{n} \to Df(\mathsf{x}_1, \ldots, \mathsf{x}_\mathsf{n}))\ldots)$ where $f \in \mathcal{C} \cup \mathcal{F}$. If f is of type $D_1, \ldots, D_n \to D$ then we will write Th_f for the formula $\forall \mathsf{x}_1(D_1\mathsf{x}_1 \to \ldots \forall \mathsf{x}_\mathsf{n}(D_n\mathsf{x}_\mathsf{n} \to Df(\mathsf{x}_1, \ldots, \mathsf{x}_\mathsf{n}))\ldots)$.

1.2.2 Equational axioms for functions

Functions are defined in our system by sets of equations but our prover does not deal, of course, with all equational data's. We now define what we call an *equational data for f* where f is the symbol of the function aimed by the equational data.

Definition 1.3 *Let f be of type $D_1, \ldots, D_n \to D$. A set of equations*

$$\begin{cases} t_1 = u_1 \\ \quad\vdots \\ t_m = u_m \end{cases}$$

is an equational data for f *iff*

- *for each i $(1 \leq i \leq m)$, (t_i, u_i) is a well named pair of terms,*
- *for each i $(1 \leq i \leq m)$, t_i is of the form $f(t_i^1, \ldots, t_i^n)$ and for each j $(1 \leq j \leq n)$, $t_i^j \in \mathcal{A}c(D_j)$,*
- *for each i $(1 \leq i \leq m)$, if g is a functional symbol occurring in u_i then either $g = f$ either $Th_g \in \mathcal{TH}$.*

Note that the last point prevents mutually dependent recursive definitions.
Because we use equations as oriented objects, equational data's are seen as sets of pairs :
$\{(t_1, u_1), \ldots, (t_m, u_m)\}$.

Definition 1.4 *Let $\mathcal{E}$ be an equational data for f and (t, u) an element of $\mathcal{E}$, then each pair (t, v) where v is a sub-term of u of the form $f(v_1, \ldots, v_n)$ is called an* inductive call *of (t, u) or an* inductive call *of $\mathcal{E}$.*

Note that inductive calls are well named pairs of terms.

2 The deduction system

In the two following subsections, we characterize formulæ and judgements processed in the first part of R-proofs (the induction part). In the third subsection we give the definition of the rules used in the R-proofs. Examples of those objects in the R-proof may be found in section 6.

2.1 The R-formulæ

Definition 2.1 *The formula* $\forall x_1(D_1x_1 \to \dots \forall x_m(D_mx_m \to Df(t_1,\dots,t_n))\dots)$ *is an* R-formula *iff :*

- $D_1,\dots,D_m$ *and* D *are elements of* $\mathcal{D}$ *;*
- $f(t_1,\dots,t_n)$ *is of type* D *;*
- *if* f *is of type* $E_1,\dots,E_n \to D$ *then for all* i $(1 \le i \le n)$, $t_i \in \mathcal{A}c(E_i)$ *;*
- *for all* i $(1 \le i \le m)$, x_i *is of type* D_i *in* $f(t_1,\dots,t_n)$.

The term $f(t_1,\dots,t_n)$ is called *the core* of the R-formula. We write $C(\theta)$ for the core of any R-formula θ. The formulæ $D_1x_1,\dots,D_mx_m$ are the *premises* of the R-formula.

The R-formula : $\forall x_1(D_1x_1 \to \dots \forall x_m(D_mx_m \to Df(t_1,\dots,t_n))\dots)$ will be written :
$\forall x_1 D_1x_1,\dots,\forall x_m D_mx_m \to Df(t_1,\dots,t_n)$, and more generally, a formula of the form $\forall x_1(D_1x_1 \to \dots \forall x_m(D_mx_m \to \varphi)\dots)$ will be written $\forall x_1 D_1x_1,\dots,\forall x_m D_mx_m \to \varphi$.
For instance, elements of $\mathcal{TH}$ are R-formulæ.

2.2 The R-judgements

A judgement is a pair (Γ,θ), usually written $\Gamma \vdash \theta$, where Γ is a set of formulæ and θ is a formula. The expression $\Gamma,\varphi_1,\dots,\varphi_k$ is an abbreviation for $\Gamma \cup \{\varphi_1,\dots,\varphi_k\}$. The expression $\Gamma[H]$ will mean that $H \in \Gamma$.

Before defining what R-judgements are, we need to introduce some terminology which will distinguish between the arguments of a constructor.

Definition 2.2 *Let* $t = cf(t_1,\dots,t_n)$ *be an element of* $\mathcal{A}c(D)$ *for some data type* D*, then* t_i $(1 \le i \le n)$ *is an* inductive argument *of* t *iff* t_i *is of type* D *in* t *; it is a* parameter argument *otherwise.*
A strict sub-term u *in* t *is said to be in* inductive position *in* t *iff* u *is an inductive argument of a sub-term of* t *; it is in* parameter position *otherwise.*

This definition is extended to the terms of the form $f(c_1,\dots,c_n)$ where $f \in \mathcal{F}$ is of type $D_1,\dots,D_n \to D$ and for each i $(1 \le i \le n), c_i \in \mathcal{A}c(D_i)$. A strict sub-term u in $f(c_1,\dots,c_n)$ is in inductive position in $f(c_1,\dots,c_n)$ iff there exists i $(1 \le i \le n)$ such that u is in inductive position in c_i ; it is in parameter position otherwise.
Note that the terms $c_1,\dots,c_n$ are in parameter position in $f(c_1,\dots,c_n)$. If a sub-term u of a term t is a variable in inductive (resp. parameter) position in t, we call u *an inductive (resp. parameter) variable* of t. The set of inductive (resp. parameter) variables of t will be denoted by $\mathcal{V}i(t)$ (resp. $\mathcal{V}p(t)$).
Note that the notions of inductive variables and parameter variables are defined for R-formulæ's cores.

Example In $Cons(s(n),Cons(m,x))$, the sub-terms n, $Cons(m,x)$ and x are in inductive position, while $s(n)$ and m are in parameter position. The variables n and x are inductive variables. The only parameter variable is m.

Definition 2.3 *Let* θ *be an R-formula. Then* θ *is* paraclosed *iff all parameter variables occurring in* $C(\theta)$ *are bound and all inductive variables are free.*

For instance, the elements of $\mathcal{TH}$ are paraclosed formulæ.
The formula $\forall \mathsf{x}_{11} N\mathsf{x}_{11} \to Nf(Cons(\mathsf{x}_{11},\mathsf{x}_{12}))$ is a paraclosed formula,
but $\forall \mathsf{x}_{12}, LN\mathsf{x}_{12} \to Nf(Cons(\mathsf{x}_{11},\mathsf{x}_{12}))$ is not.

Definition 2.4 *A judgement* $\Gamma \vdash \theta$ *is an* R-judgement *iff*

- θ *is a paraclosed R-formula and* $C(\theta)$ *is a well named term ;*
- *each formula occurring in* Γ *is either an R-formula or an atomic formula of the form* Dx *where* $D \in \mathcal{D}$ *and* $x \in \omega^\omega$*;*
- *for all* $x \in \omega^\omega$ *and for all* $D \in \mathcal{D}$*, we have* $x \in \mathcal{V}i(C(\theta))$ *and* x *is of type* D *in* $C(\theta)$.

2.3 The deduction rules

Our prover will make proofs with the help of some macro-rules (or *tactics*, or *derived rules*) which are built up from introduction and elimination rules of the connector $\rightarrow$ and the universal quantifier $\forall$. It will also make use of the equational and of the axiom rules. For a complete presentation of this underlying deduction system see, for instance, [1].
A deduction rule is written

$$\frac{\Gamma_1 \vdash \varphi_1 \ ... \ \Gamma_k \vdash \varphi_k}{\Gamma \vdash \varphi} \, {\scriptstyle Nr}$$

and means that *if* $\Gamma_1 \vdash \varphi_1, \ldots, \Gamma_k \vdash \varphi_k$ *then* $\Gamma \vdash \varphi$. The judgements $\Gamma_1 \vdash \varphi_1, \ldots, \Gamma_k \vdash \varphi_k$ are the premises of the rule, the judgement $\Gamma \vdash \varphi$ is its conclusion and Nr its name. We can also read the rules as bottom-up rules. In this case, we call $\Gamma \vdash \varphi$ the *goal* of the rule and $\Gamma_1 \vdash \varphi_1, \ldots, \Gamma_k \vdash \varphi_k$ the *subgoals generated by the rule.*

The first two macro-rules that we are going to define correspond respectively to applying a theorem and to applying an induction hypothesis.

Macro-rule Thm Let Th_g be an element of $\mathcal{TH}$ (that is to say an R-formula of the form $\forall \mathsf{x}_1 D_1 \mathsf{x}_1, \ldots, \forall \mathsf{x}_m D_m \mathsf{x}_m \rightarrow Dg(\mathsf{x}_1, \ldots, \mathsf{x}_m)$). The macro-rule Thm is :

$$\frac{\Gamma \vdash D_1 t_1 \ \ldots \ \Gamma \vdash D_m t_m}{\Gamma \vdash Dg(t_1, \ldots, t_m)} \, {\scriptstyle Thm}$$

Note that g can be a constant (eg : a functional symbol of arity 0), in this case, the rule Thm doesn't generate any more subgoal.

Macro-rule Hyp Let H be the R-formula $\forall x_1 D_1 x_1, \ldots, \forall x_m D_m x_m \rightarrow Df(u_1, \ldots, u_n)$, and let $f(t_1, \ldots, t_n)$ be a term. If $f(u_1, \ldots, u_n)$ matches $f(t_1, \ldots, t_n)$ according to σ and if σ is such that for all variable x occurring free in H, $\sigma(x) = x$ then we will say that H *can be applied to* $f(t_1, \ldots, t_n)$. The macro-rule Hyp is :

$$\frac{\Gamma[H] \vdash D_1 \sigma x_1 \ \ldots \ \Gamma[H] \vdash D_m \sigma x_m}{\Gamma[H] \vdash Df(t_1, \ldots, t_n)} \, {\scriptstyle Hyp}$$

As noticed above, the formula H can have no premise. In this case, the rule Hyp reduces to the usual axiom rule.

Macro-rule Ax This macro-rule denotes the usual axiom rule which is used for some atomic formula. The Ax macro-rule is :

$$\Gamma[Dx] \vdash Dx$$

where $D \in \mathcal{D}$ and $x \in \omega^\omega$.

Macro-rule Eq Let $\forall x_1 D_1 x_1, \ldots, \forall x_m D_m x_m \rightarrow Dt$ be an R-formula and $t = u$ some equation, then macro-rule Eq is :

$$\frac{\Gamma, D_1 x_1, \ldots, D_m x_m \vdash Du}{\Gamma \vdash \forall x_1 D_1 x_1, \ldots, \forall x_m D_m x_m \rightarrow Dt} \, {\scriptstyle Eq}$$

Macro-rule Rec_D This macro-rule mainly corresponds to the use of the induction principle for a data type D, but it will make some manipulations on formulæ before and after using the induction principle. The aim of those manipulations is that : if the conclusion of the goal of the macro-rule Rec_D is an R-judgement then the conclusions of all subgoals generated by the macro-rule will be also an R-judgement. To simplify the presentation, we will not give the general pattern of the Rec_D macro-rule but we will illustrate its use with two examples.

The first one uses recursion over natural numbers. Let recall first one usual formulation of this recursion rule.

$$\frac{\Gamma \vdash \theta[0/x] \qquad \Gamma \vdash \forall y(Ny \to (\theta[y/x] \to \theta[s(y)/x]))}{\Gamma \vdash \forall x(Nx \to \theta[x])}$$

where $\theta[x]$ denotes a formula where the variable x occurs free.
Suppose now that we want to prove $\forall \mathsf{x}_1 N\mathsf{x}_1, \forall \mathsf{x}_2 N\mathsf{x}_2 \to Nf(\mathsf{x}_1, \mathsf{x}_2)$ in a context Γ where x_1 and x_2 have no free occurrence and using recursion on x_2. Our strategy will do the following :

$$\frac{\Gamma \vdash \forall \mathsf{x}_1(N\mathsf{x}_1 \to Nf(\mathsf{x}_1, 0)) \qquad \Gamma, N\mathsf{x}_{21}, \forall \mathsf{x}_1(N\mathsf{x}_1 \to Nf(\mathsf{x}_1, \mathsf{x}_{21})) \vdash \forall \mathsf{x}_1(N\mathsf{x}_1 \to Nf(\mathsf{x}_1, s(\mathsf{x}_{21}))}{\Gamma \vdash \forall \mathsf{x}_1 N\mathsf{x}_1, \forall \mathsf{x}_2 N\mathsf{x}_2 \to Nf(\mathsf{x}_1, \mathsf{x}_2)}$$

This piece of proof is trivialy derived from the above usual recursion rule and rules for eliminate and introduce the $\to$ connective and the $\forall$ quantifier. If we read this macro-rule *bottom-up*, we see that it first put the premisse $N\mathsf{x}_2$ in the context, then reintroduce the premisse $N\mathsf{x}_1$ and apply the usual recursion rule. Finaly, in the successor case, it puts the hypothesis $N\mathsf{x}_{21}$ and the recurrence hypothesis in the context.

Our second example illustrates the use of recursion on lists of natural numbers. Let recall first one usual expression of the recursion rule over lists.

$$\frac{\Gamma \vdash \theta[Nil/x] \qquad \Gamma \vdash \forall n Nn, \forall y LNy, \theta[y/x] \to \theta[Cons(n, y)/x]}{\Gamma \vdash \forall x(LNx \to \theta[x])}$$

Then suppose we want to prove $\forall \mathsf{x}_1 LN\mathsf{x}_1 \to Nf(\mathsf{x}_1)$ using recursion on x_1. Our strategy will build :

$$\frac{\Gamma \vdash Nf(Nil) \qquad \Gamma, N\mathsf{x}_{12}, Nf(\mathsf{x}_{12}) \vdash \forall \mathsf{x}_{11}(N\mathsf{x}_{11} \to Nf(Cons(\mathsf{x}_{11}, \mathsf{x}_{12})))}{\Gamma \vdash \forall \mathsf{x}_1(LN\mathsf{x}_1 \to Nf(\mathsf{x}_1))}$$

Remark here that, in the *Cons* case, the *parameter* argument x_{11} of *Cons* is bound.

Note also that in our both examples goals produced by the macro-rule are R-judgements. In particular, conclusions are *paraclosed* formulæ and their core are well named terms.
We will often omit to mention the data type and simply write *Rec* to refer to this rule in general. On the other hand, we will mention the variable involved in the rule and write *Rec*(x). The variable x will be called a *recursion variable*.

3 The R-proofs

A basic idea of our algorithm is to restrict the search space to a certain class of proofs. This is the class we want to define precisely now. We give an example of an R-proof in section 6.

Definition 3.1 *Let $\mathcal{E}$ be an equational data for some $f \in \mathcal{F}$. Let $\mathcal{A}$ be a proof tree whose root is an R-judgement $(\Gamma_0 \vdash \theta_0)$ and which is built on the Ax, Thm, Hyp, Eq and Rec rules.*
Then $\mathcal{A}$ is an R-prooftree *according to $\mathcal{E}$ iff :*

- *for each branch $\Gamma_0 \vdash \theta_0, \ldots, \Gamma_n \vdash \theta_n$ of $\mathcal{A}$, there exists $e (0 \leq e < n)$ such that :*
  - *for all i $(0 \leq i < e)$, θ_i is the conclusion of a Rec rule ;*
  - *θ_e is the conclusion of an Eq rule ;*
  - *for all i $(e < i \leq n)$, θ_i is the conclusion of an Ax, a Thm or a Hyp rule.*
- *if $E = \{\Gamma_{e_1} \vdash \theta_{e_1}, \ldots, \Gamma_{e_k} \vdash \theta_{e_k}\}$ is the set of the judgements of $\mathcal{A}$ which are conclusions of the Eq rule then there exists a one to one function $b : \mathcal{E} \longrightarrow E$ such that $b(t,u) = \Gamma_{e_i} \vdash \theta_{e_i}$ iff $C(\theta_{e_i}) = t$.* [1]

The subtree of $\mathcal{A}$ whose root is $\Gamma_0 \vdash \theta_0$ and whose set of leaves is E is called the recursion tree *of $\mathcal{A}$. Moreover, because of the bijection $b : \mathcal{E} \longrightarrow E$, this recursion tree will be called a* distributing tree for $\mathcal{E}$. *Note that this bijection is well-defined because the t's and the $C(\theta_{e_i})$'s are well named terms.*

Those notions of recursion tree and distributing tree will be of great importance in the following. So let illustrate them by the following example : let f be a function defined by the equations

$$\begin{array}{lcl} f(0,0,z) & = & z \\ f(0,s(y),z) & = & f(0,y,s(z)) \\ f(s(x),y,0) & = & f(x,s(y),s(0)) \\ f(s(x),y,x(z)) & = & f(x,s(y),f(s(x),s(y),z)) \end{array}$$

Our strategy will produce from this set of equations a distributing tree which can be represented as follows, where underlined variables represent bound variables.
for instance $Nf(\underline{\mathsf{x}_1},\underline{\mathsf{x}_2},\underline{\mathsf{x}_3})$ represents the R-formula $\forall \mathsf{x}_1 N\mathsf{x}_1, \forall \mathsf{x}_2 N\mathsf{x}_2, \forall \mathsf{x}_3 N\mathsf{x}_3 \rightarrow Nf(\mathsf{x}_1,\mathsf{x}_2,\mathsf{x}_3)$ and $Nf(s(\mathsf{x}_{11}),\underline{\mathsf{x}_2},\mathsf{x}_{31})$ represents $\forall \mathsf{x}_2 N\mathsf{x}_2 \rightarrow Nf(\mathsf{x}_{11},\mathsf{x}_2,\mathsf{x}_{31})$.

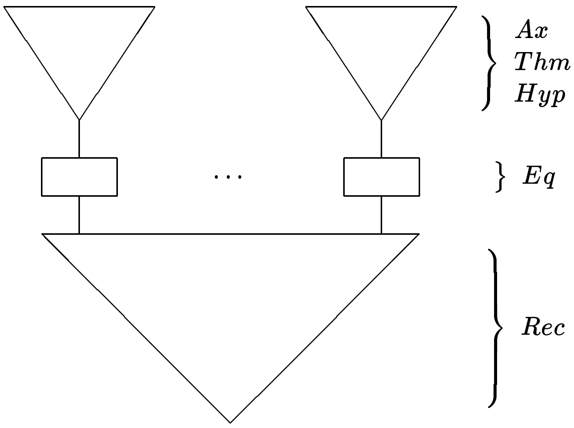
$$\dfrac{\dfrac{\vdash Nf(0,0,\underline{\mathsf{x}_3}) \quad \Gamma_2 \vdash Nf(0,s(\mathsf{x}_{21}),\underline{\mathsf{x}_3})}{\vdash Nf(0,\underline{\mathsf{x}_2},\underline{\mathsf{x}_3})}\,{\scriptstyle Rec(\mathsf{x}_2)} \qquad \dfrac{\Gamma_1 \vdash Nf(s(\mathsf{x}_{11}),\underline{\mathsf{x}_2},0) \quad \Gamma_3 \vdash Nf(s(\mathsf{x}_{11}),\underline{\mathsf{x}_2},s(\mathsf{x}_{31}))}{\Gamma_1 \vdash Nf(s(\mathsf{x}_{11}),\underline{\mathsf{x}_2},\underline{\mathsf{x}_3})}\,{\scriptstyle Rec(\mathsf{x}_3)}}{\vdash Nf(\underline{\mathsf{x}_1},\underline{\mathsf{x}_2},\underline{\mathsf{x}_3})}\,{\scriptstyle Rec(\mathsf{x}_1)}$$

with
$\Gamma_1 = N\mathsf{x}_{11}, Nf(\mathsf{x}_{11},\underline{\mathsf{x}_2},\underline{\mathsf{x}_3})$
$\Gamma_2 = N\mathsf{x}_{21}, Nf(0,\mathsf{x}_{21},\underline{\mathsf{x}_3})$
$\Gamma_3 = \Gamma_1, N\mathsf{x}_{31}, Nf(s(\underline{\mathsf{x}_{11}}),\underline{\mathsf{x}_2},\mathsf{x}_{31})$

Remark 3.1 *Note that if $\Gamma_0 \vdash \theta_0, \ldots, \Gamma_e \vdash \theta_e$ is a branch of a recursion tree then for all i $(0 \leq i < e)$, for all j $(i \leq j \leq e)$, $C(\theta_i)$ matches $C(\theta_j)$. As a consequence, if $C(\theta_j)$ matches some term t then $C(\theta_i)$ does also.*

An R-proof tree can be figured like this :

[1] We should have weaken our conditions on this b function. We should have requested only b to be a surjective function. In this case we would have to make an arbitrary choice between differents equations having same left part. Also, we should have requested only that $C(\theta_{e_i})$ matches t instead of equality. This would mean that we would allowed our self to make induction that are not explicitly given in the user's specification and, in this case, arise the problem of termination of the algorithm.



As we are concerned with proving the totality of functions, we will only consider R-prooftrees with root of the form $\emptyset \vdash Th_f$.

Definition 3.2 *Let $f \in \mathcal{F}$, let $\mathcal{E}$ be an equational data for f, and let $\mathcal{A}$ be an R-proof tree according to $\mathcal{E}$. Then that $\mathcal{A}$* is an R-proof for f (according to $\mathcal{E}$) *iff $\mathcal{A}$ is a proof of $\emptyset \vdash Th_f$.*

Note that, as $\emptyset \vdash Th_f$ is an R-judgement, if we name variables as described in paragraph **??** then all judgements of the recursion tree of $\mathcal{A}$ are R-judgements.

Before going further, we want to give some remarks about the induction hypothesis generated in the R-proof trees. Let $\Gamma \vdash \theta$ be the conclusion of a *Rec* rule and $\Gamma' \vdash \theta'$ be one of its premises.

Remark 3.2 *Assume that H is an induction hypothesis of Γ', that is to say that $H \in \Gamma'$ and for all $D \in \mathcal{D}$, for all $x \in \omega^\omega$, $H \neq Dx$. If $H \notin \Gamma$ then there exists $a \in \omega$ and $\mathsf{q} \in \omega^\omega$ such that $C(H) = C(\theta[\mathsf{x_{qa}}/\mathsf{x_q}])$. Note that $\mathsf{x_{qa}}$ is free in H.*

Remark 3.3 *If $\Gamma \vdash \theta$ is an R-judgement then $C(\theta)$ is a well named term. It is easy to check that if $\mathsf{x_k} \neq \mathsf{x_q}$ occurs in $C(H)$ then $Adr(\mathsf{x_k}, C(H)) = \mathsf{k}$. In this case, we say $C(H)$ is a well named term* excepted for $\mathsf{x_k}$.

Remark 3.4 *If $C(H) = C(\theta[\mathsf{x_{qa}}/\mathsf{x_q}])$ then $Adr(\mathsf{x_{qa}}, C(H)) = \mathsf{q}$.*

Remark 3.5 *If $\mathsf{x_k} \in \mathcal{V}i(C(H))$ and $\mathsf{x_k} \neq \mathsf{x_q}$ then $\mathsf{x_k} \in \mathcal{V}i(C(\theta))$. As θ is a paraclosed formula, we say that H is paraclosed* excepted for $\mathsf{x_q}$.

4 The R-proof searching algorithm

4.1 Restriction of the problem

The main problem to build an R-proof for the totality of a recursively defined function is to get the right induction hypothesis. That is to say, the problem is to build distributing trees which satisfy the following property :

Definition 4.1 *Let $\mathcal{E}$ be an equational data for a given f and let $\mathcal{A}$ be a distributing tree for $\mathcal{E}$. We say $\mathcal{A}$ satisfies the* right terminal state *property for $\mathcal{E}$ iff for all leaf $\Gamma \vdash \theta$ of $\mathcal{A}$, if $(t, u) \in \mathcal{E}$ is the equation such that $t = C(\theta)$ then for all inductive calls (t, v) of (t, u) there exists $H \in \Gamma$ such that H can be applied to v.*

The following proposition establishes the equivalence between searching for an R-proof and searching for a distributing tree which satisfies the right terminal state property.

Proposition 4.1 *Let $\mathcal{E}$ be an equational data for a given f and let $\mathcal{A}$ be an R-proof tree according to $\mathcal{E}$ whose root is $\emptyset \vdash Th_f$. Then $\mathcal{A}$ is an R-proof for f iff the recursion tree of $\mathcal{A}$ satisfies the right terminal state property.*

Proof
The proof is made by induction on the core of the leaves of the recursion tree (see [4]). ♠

In the two following subsections, we give two properties. We will see in section 5 that they are necessary and sufficient for a given distributing tree to satisfy the right terminal state.

The first one (*the decreasing property*) is a necessary condition to get induction hypothesis. It is checked on the terms of the equational data. From the termination problem point of view, it ensures that there is no infinite loop in a same inductive call.

The second one (*the suitable variable*) is a dynamic property which is checked at each step of the construction of the recursion tree. Intuitively, it ensures that inductive calls are compatible, that is to say that there can't be infinite loop between several inductive calls. It provides us with a criterium to decide on which variable induct.

4.2 The decreasing property

This property affects the equational data. We know that if this property is not satisfied then there is no use to search for an R-proof of the function (see proposition 5.2).

The definition of this so-called decreasing property needs some preliminaries. Let us define the function $Struct : \mathcal{T} \longrightarrow \mathcal{T}$ by :

if t is a variable then $Struct(t) = t$;

if $t = f(t_1, \ldots, t_n)$ where $f \in \mathcal{F}$ then $Struct(t) = t$;

if $t = cf(t_1, \ldots, t_{m+p})$ where $cf \in \mathcal{C}$ and is of type $D_1, \ldots, D_m, D, \ldots, D \rightarrow D$ then $Struct(t) = cf(\mathsf{x}_0, \ldots, \mathsf{x}_0, Struct(t_{m+1}), \ldots, Struct(t_{m+p}))$.

Definition 4.2 *Let $D \in \mathcal{D}$ and $t, u \in \mathcal{A}c(D)$. Then $t \equiv u$ iff $Struct(t) = Struct(u)$.*

For instance we have $Cons(n, x) \equiv Cons(s(n), x)$ but not $Cons(n, x) \equiv Cons(n, y)$. We have two useful lemmas concerning this relation.

Lemma 4.1 *Let $D \in \mathcal{D}$, let $cf(t_1, \ldots, t_n)$ and $cf(u_1, \ldots, u_n) \in \mathcal{A}c(D)$. Then $cf(t_1, \ldots, t_n) \equiv cf(u_1, \ldots, u_n)$ iff for each i $(1 \leq i \leq n)$, if t_i is an inductive argument of $cf(t_1, \ldots, t_n)$ then $t_i \equiv u_i$.*

Proof
By induction on the t_i's. ♠

Lemma 4.2 *Let $D \in \mathcal{D}$ and $t, u \in \mathcal{A}c(D)$. If t is not a variable and matches u according to σ and if for all $x \in \mathcal{V}i(t), \sigma(x) = x$ then $t \equiv u$.*

Proof
We put $t = cf(t_1, \ldots, t_n)$ and, using lemma 4.1, we reason by induction on the t_i's. ♠

Let t and u be two terms, let q be an address in t. Then we call *the graft of u on t at* q the term obtained by replacing the sub-term t_q of t (eg : the sub-term of t whose address is q in t) by u. The expression $t[\mathsf{q} \leftarrow u]$ denotes this graft. For instance $Cons(s(n), x)[11 \leftarrow s(m)] = Cons(s(s(m)), x)$.

We define on the addresses a substraction operation by : $\mathsf{p} - \mathsf{q} = \mathsf{r}$ iff $\mathsf{p} = \mathsf{q.r}$. And we define an ordering by : $\mathsf{q} \leq_a \mathsf{p}$ iff there exists r such that $\mathsf{p} = \mathsf{q.r}$. For instance $12 \leq_a 121$.
This ordering will be used to tell about the sub-terms t_k of t which contain a given t_q. Actually the set of the t_k's containing t_q is the set of the t_k's such that $\mathsf{k} \leq_a \mathsf{q}$. Using substaction, we get that $Adr(t_\mathsf{q}, t_\mathsf{k}) = \mathsf{q} - \mathsf{k}$. For instance, the address of $Cons(0, x)$ in $Cons(n, Cons(0, x))$ is 2 and the address of 0 in $Cons(n, Cons(0, x))$ is 21 so the address of 0 in $Cons(0, x)$ is 21 - 2 = 1.

Definition 4.3 *Let $t \in \mathcal{A}c(D)$ and u a term of type D, let k be an address in t. Then $t \gg_\mathsf{k} u$ iff $t_\mathsf{k} = cf(\ldots, \mathsf{x}_\mathsf{q}, \ldots)$, the variable x_q is an inductive argument of t_k and for all $\mathsf{k}' \leq_a \mathsf{k}$, $t_{\mathsf{k}'}[\mathsf{k} - \mathsf{k}' \leftarrow \mathsf{x}_\mathsf{q}] \equiv u_{\mathsf{k}'}$.*
We call t a successor *of u (writen $t \gg u$) iff there exists k such that $t \gg_\mathsf{k} u$.*

Notice that we have in particular $cf(\ldots, \mathsf{x_q}, \ldots) \gg_{\boldsymbol{\varepsilon}} \mathsf{x_q}$. To give some more examples, we have $s(m) \gg_{\boldsymbol{\varepsilon}} m$, $Cons(n, Cons(m, x)) \gg_2 Cons(a, x)$ (whatever a may be) and
$Cons(n, Cons(s(m), x)) \gg_{21} Cons(n, Cons(m, x))$.

We have for the successor relation the following lemmas :

Lemma 4.3 *If $t \gg_{\mathsf{q}} u$ then for all $\mathsf{p} \leq_a \mathsf{q}$, we have $t_{\mathsf{p}} \gg_{\mathsf{q-p}} u_{\mathsf{p}}$.*

Proof
Immediate from definitions. ♠

Lemma 4.4 *We have $t \gg_{\mathsf{i.k}} u$ iff $t = cf(t_1, \ldots, t_n)$, $u = cf(u_1, \ldots, u_n)$, $t_i \gg_{\mathsf{k}} u_i$ and for all j $(1 \leq j \leq n)$, if t_j is an inductive argument of t and if $j \neq i$ then $t_j \equiv u_j$.*

The decreasing property it self is defined by :

Definition 4.4 *Let $\mathcal{E}$ be an equational data for f. Then $\mathcal{E}$ satisfies the* decreasing property *iff for each $(f(t_1, \ldots, t_n), u) \in \mathcal{E}$, if $(f(t_1, \ldots, t_n), f(v_1, \ldots, v_n))$ is an inductive call of $(f(t_1, \ldots, t_n), u)$ then there exists i $(1 \leq i \leq n)$ such that $t_i \gg v_i$.*

In other words, *if $(t, u) \in \mathcal{E}$ and (t, v) is an inductive call of (t, u) then there exists $i \in \omega$ such that $t_{\mathsf{i}} \gg v_{\mathsf{i}}$.*

We said above (4.1) that this decreasing property is "a necessary condition to get induction hypothesis". This is in fact the main motivation for defining this property and the successor relation we introduced is a reflect of this. Let see, with the example of the Ad_l function (see section 6), how this relation between terms reflects the possibility of getting induction hypothesis.
We have in the Ad_l equationnal definition two inductive calls :

$$(Ad_l(Cons(0, \mathsf{x_{12}})), Ad_l(\mathsf{x_{12}})) \text{ and } (Ad_l(Cons(s(\mathsf{x_{111}}), \mathsf{x_{12}})), Ad_l(Cons(\mathsf{x_{111}}, \mathsf{x_{12}})))$$

In the first case we use the first induction hypothesis which is produced by the induction on $\mathsf{x_1}$ (namely : $NAd_l(\mathsf{x_{12}})$). This is the base case of our definition, *eg* the case of $\gg_{\boldsymbol{\varepsilon}}$. For the second inductive call, we will use the induction hypothesis obtained after the nested induction on $\mathsf{x_{11}}$: $NAd_l(Cons(\mathsf{x_{111}}, \mathsf{x_{12}}))$. This hypothesis actually corresponds to the fact that $Cons(s(\mathsf{x_{111}}), \mathsf{x_{12}}) \gg_1 Cons(\mathsf{x_{111}}, \mathsf{x_{12}})$. In the hypothesis, both variables $\mathsf{x_{111}}$ and $\mathsf{x_{12}}$ are free and then can't be instanciated. Moreover, by construction of the *Rec* macro-rule, they are free because they are *inductive* variables. So, because they are free, they must be the same at the same adress in the right part of the inductive call. That is what reflects the relation $Cons(s(\mathsf{x_{111}}), \mathsf{x_{12}}) \gg_1 Cons(\mathsf{x_{111}}, \mathsf{x_{12}})$: the subterm $s(\mathsf{x_{111}})$ of $Cons(s(\mathsf{x_{111}}), \mathsf{x_{12}})$ is replaced by $\mathsf{x_{111}}$ and $\mathsf{x_{12}}$ remains unchanged.

An important remark to make about the $\gg$ relation we defined is that it is not an ordering suitable for induction and even its transitive closure is not. Actually, it is possible to build infinite sequence of terms. For instance we have : $Cons(s(n), Cons(m, x)) \gg Cons(n, Cons(s(m)), x) \gg Cons(s(n), Cons(m, x)) \gg \ldots$

4.3 The suitable variable

Definition 4.5 *Let $\mathcal{E}$ be an equational data for some f, let $\mathcal{A}$ be a distributing tree for $\mathcal{E}$, let $\Gamma \vdash \theta$ be a judgement of $\mathcal{A}$, let (t, v) be an inductive call of $\mathcal{E}$ such that $C(\theta)$ matches t and v according (respectively) to σ_t and σ_v, let x be a variable of θ. We call x* suitable for θ and (t, v) *if :*

1. *$\sigma_t(x) \gg \sigma_v(x)$ or $\sigma_t(x) \equiv \sigma_v(x)$;*
2. *if $\sigma_t(x) \gg_{\boldsymbol{\varepsilon}} \sigma_v(x)$ then for all variable $x' \neq x$, if x' is free in θ then $\sigma_t(x') = x'$.*

We will also say *x is suitable in $\mathcal{A}$* iff there exists a judgement $\Gamma \vdash \theta$ of $\mathcal{A}$ for which x is suitable for $\Gamma \vdash \theta$ and all the inductive calls (t, v) of $\mathcal{E}$ such that $C(\theta)$ matches t and v.

We said above (4.1) that choosing suitable variables as recursion variables "ensure that inductive calls are compatibles". Actually, we saw how, if each inductive call (t, u) satisfies the decreasing property, we are able, for each of them *separately* to build a recursion tree such that one of its leave corresponds to t (by mean of the b bijection (see 3.1)) and contains an hypothesis which can be applied to u. To build an R-proof taking in account all the equationnal data, and then, all the inductive calls, we have to merge *to gather*, all the recursion trees available for each inductive call. At each step of the construction of the recursion tree, choosing suitable variable as recursion variable ensure that this merging is possible.
May be the best way to illustrate it is to give a counter- example. That is to say, an example of specification for which there is no possible choice. So, let f be the dummy function defined by $\mathcal{E}$:

$$\begin{array}{lcl} f(0,0) & = & 0 \\ f(0,s(\mathsf{x}_{21})) & = & f(s(0),\mathsf{x}_{21}) \\ f(s(\mathsf{x}_{11}),0) & = & f(\mathsf{x}_{11},s(0)) \\ f(s(\mathsf{x}_{11}),s(\mathsf{x}_{21})) & = & f(\mathsf{x}_{11},\mathsf{x}_{21}) \end{array}$$

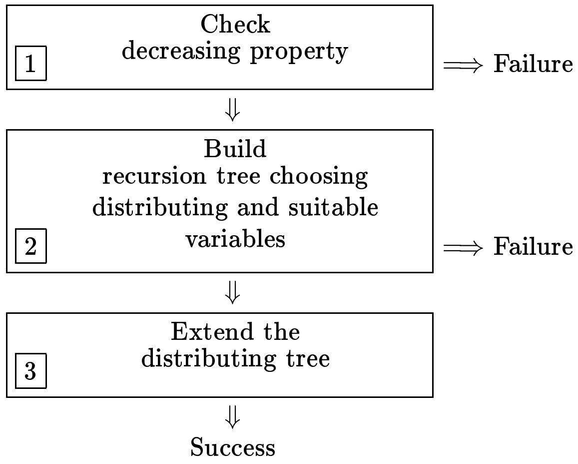
Each inductive call satisfies the decreasing property because $s(\mathsf{x}_{11}) \gg_{\mathcal{E}} \mathsf{x}_{11}$ and $s(\mathsf{x}_{21}) \gg_{\mathcal{E}} \mathsf{x}_{21}$. But the second and the third equation make impossible to choose either the first or the second argument to use induction.
Actually, the problem comes from $(f(0,s(\mathsf{x}_{21})), f(s(0),\mathsf{x}_{21}))$ and $(f(s(\mathsf{x}_{11}),0), f(\mathsf{x}_{11},s(0)))$: to get an induction hypothesis which can be applied to $f(s(0),\mathsf{x}_{21})$, the first argument of f must be bound so induction must be used on the second argument; but, *at the same time*, to get an induction hypothesis which can be applied to $f(\mathsf{x}_{11},s(0))$, the second argument of f must be bound and so induction must be used on the first argument !
Any distributing tree $\mathcal{A}$ for $\mathcal{E}$ begins with the judgement $\theta = \vdash \forall \mathsf{x}_1 N\mathsf{x}_1, \forall \mathsf{x}_2 N\mathsf{x}_2 \to Nf(\mathsf{x}_1,\mathsf{x}_2)$. But x_1 is not suitable for θ and $(f(0,s(\mathsf{x}_{21})), f(s(0),\mathsf{x}_{21}))$ and x_2 is not suitable for θ and $(f(s(\mathsf{x}_{11}),0), f(\mathsf{x}_{11},s(0)))$. Then it is impossible to choose a recursion variable which respects the suitable criterium.

4.4 The algorithm

Given an equational data $\mathcal{E}$ and a distributing tree according to $\mathcal{E}$, we know that $\mathcal{A}$ satisfies the right terminal state property iff $\mathcal{E}$ satisfies the decreasing property and each recursion variable of $\mathcal{A}$ is suitable (see 5).
There is a simple property that recursion variables of a distributing tree must satisfy. Actually, it is easy to check that $\mathcal{A}$ is a distributing tree iff for all judgements $\Gamma \vdash \theta$ of $\mathcal{A}$ and for all $(t, u) \in \mathcal{E}$, if $C(\theta)$ matches t (according to σ_t^θ) and if x is the recursion variable of $\Gamma \vdash \theta$ then $\sigma_t^\theta(x)$ is not a variable. Let us call *distributing variables* this kind of recursion variables.
The conjunction of the suitable and distributing properties provides us with a criterium to choose dynamically the variables on which induct when we want to build a distributing tree satisfying the right terminal state property.

Remains now to design an algorithm to extend $\mathcal{A}$ in an R-proof. That is to say : to build a proof of all the judgements $\Gamma_e \vdash \theta_e$ which are leaves of $\mathcal{A}$, using (once) the Eq rule and then, the Ax, Hyp, and Thm rules. The inductive proof of proposition 4.1 provides us with a recursive function to implement such an algorithm. Notice that this function can't fail and use no back-tracking when it is applied to the leaves of a distributing tree satisfying the right terminal state.

Let us now present the whole algorithm :



Lemmas 4.1 and 4.4 provide us with recursive functions to implement $\boxed{1}$.

To implement $\boxed{2}$ we define a function named `build-tree` which returns either a recursion tree which satisfies the right terminal state, either the symbol `Failure`. We represent R-proof trees by the algebra
`P := Failure | Ax | Thm(j,[P;..;P]) | Hyp(j,[P;..;P]) | Eq(j,P) | Rec(x,j,[P;..;P])`
where `j` is the judgement concluding the proof and `x` the recursion variable. Let `(Ga,th)` represents an R-judgement, `E` represents a set of equations. Assume that `(core th)` returns the core of `th` ; assume that `(subE th E)` returns the list of the elements `(t,u)` of `E` such that `(core th)` matches `t` ; assume that the function `(elig (Ga,th) E)` returns the list of the suitable variables for `(Ga,th)` and `E` ; finally, assume that `(fail A)` returns `true` if the recursion tree `A` contains `Failure`, returns `false` otherwise.
So, if `Ef` represents the equational data $\mathcal{E}$, we define (in "ML style") the function `build-tree` as follow :

```
let build-tree Thf = (build-tree-rec ([],Thf) Ef (elig ([],Thf) Ef))
whererec build-tree-rec (Ga,th) E = function
 [] -> if E=[(t,u)] and (core th)=t then
         (extend (Ga,th) (t,u))
       else
         Failure
|x::xs
 -> let A = Rec(x,
                (Ga,th),
                [(build-tree-rec (Ga1,th1) E1 (elig Ga1 th1 E1);
                 ...;
                 (build-tree-rec (Gak,thk) Ek (elig Gak thk Ek)]
    in
     if (fail A) then
       (build-tree-rec (Ga,th) E xs)
     else
       A
;;
```

where `(Ga1,th1)`, ..., `(Gak,thk)` represent the subgoals generated by the application of the *Rec* rule on the judgement `(Ga,th)` for the variable `x` and `E1`,...,`Ek` are the subsets of `E` corresponding respectively to `(subE th1 E)`,...,`(subE thk E)`.

Assume that `(core th)` is the atomic formula `D(t)`, terms are defined by the algebra `T := Var(x) | Fun(f,[T,..,T])`. Let `f0` be the function whose termination proof we are searching for. Assume that `(find-hyp Ga Fun(f,[t1,..,tm]))` returns the formula of `Ga` which can be applied to `t` ; assume that `(app-hyp h t)` returns the lists of the instances of the premises of `h` when applied to `t`. We define the function `extend` which implements $\boxed{3}$ as :

```
let extend (Ga,th) (t,u) = Eq((Ga,th),(extend-more Ga' D(u)))

whererec extend-more Ga' = function
 Var(x) -> Ax
|Fun(f,[t1,..,tm])
 -> if f=f0 then
      let h = (find-hyp Ga' Fun(f,[t1,..,tm]) in
      let [D1(s1);..;Dk(sk)] = (app-hyp h Fun(f,[t1,..,tm])) in
      Hyp((Ga',Fun(f,[t1,..,tn])),
          [(extend-more Ga' D1(s1));
           ..;
           (extend-more Ga' Dk(sk)])
    else
      Thm((Ga',Fun(f,[t1,..,tn])),
          [(extend-more Ga' D1(t1));
           ..;
           (extend-more Ga' Dn(tn)])
;;
```

where `(Ga',D(u))` is the subgoal generated by the Eq rule and
where `(Ga',D1(t1))`, ..., `(Ga',Dn(tn))` are the subgoals generated by the Thm rule.

5 Soundness and completeness of the algorithm

By the three propositions of this section, we prove the equivalence between distributing trees according to an equational data which satisfies the right terminal state property and the conjunction of decreasing property of the equational data and the fact that all the recursion variables of the distributing tree are suitable. By proposition 4.1, this equivalence gives us the soundness and completeness of our algorithm.

When, for some R-formula θ and some term t, $C(\theta)$ matches t, we will write σ_t^θ the substitution such that $\sigma_t^\theta(C(\theta)) = t$.

Proposition 5.1 *Let $\mathcal{E}$ be an equational data for some f, let $\mathcal{A}$ be a distributing tree for $\mathcal{E}$. If $\mathcal{E}$ satisfies the decreasing property and if each recursion variable of $\mathcal{A}$ is suitable then $\mathcal{A}$ satisfies the right terminal state property.*

Proof
Let (t, v) be an inductive call in $\mathcal{E}$. Because $\mathcal{A}$ is a distributing tree we have $\Gamma_e \vdash \theta_e$ which is a leaf of $\mathcal{A}$ such that $C(\theta_e) = t$. Let $\Gamma_0 \vdash \theta_0, \ldots, \Gamma_e \vdash \theta_e$ be the branches of $\mathcal{A}$ such that $\Gamma_0 = \emptyset$ and $\theta_0 = Th_f$.
Let us first prove the following fact : there exists m $(0 < m \leq e)$ such that for all j $(0 \leq j < m)$, $C(\theta_j)$ matches v and for all j' $(m \leq j' \leq e)$, $C(\theta_{j'})$ does not.

By hypothesis, there exists $i \in \omega$ such that $t_\mathsf{i} \gg v_\mathsf{i}$ (e.g. there exists $\mathsf{q}' \in \omega^\omega$ such that $t_\mathsf{i} \gg_{\mathsf{q}'} v_\mathsf{i}$). In this case, it is easy to deduce that t does not match v (induction on q'), that is to say $C(\theta_e)$ does not match v. So, there exists m $(0 < m \leq e)$ which the least one such that $C(\theta_m)$ doesn't match v. We know that for all m' such that

$(m \leq m' \leq e)$, $C(\theta_m)$ matches $C(\theta_{m'})$. Then $C(\theta_{m'})$ can't match v. The situation can be figured this way :

$$\dfrac{\begin{array}{c}\Gamma_e \vdash \theta_e \\ \vdots \\ \dots \quad \Gamma_m \vdash \theta_m \quad \dots\end{array}}{\begin{array}{c}\Gamma_{m-1} \vdash \theta_{m-1} \\ \vdots \\ \emptyset \vdash Th_f\end{array}} \; Rec(\mathsf{x_q})$$

where θ_{m-1} matches v and θ_m doesn't. Note that because $C(\theta_e) = t$, we have $C(\theta_{m-1})$ and $C(\theta_m)$ match t (see remark 3.1). We will prove that there exists $H \in \Gamma_m$ which can be applied to v. Let first prove that $\sigma_t^{\theta_{m-1}}(\mathsf{x_q}) \gg_{\varepsilon} \sigma_v^{\theta_{m-1}}(\mathsf{x_q})$.

Notice that,because $C(\theta_m) = C(\theta_{m-1})[cf(\mathsf{x_{q1}}, \dots, \mathsf{x_{qp}})/\mathsf{x_q}]$, for some $cf \in \mathcal{C}$ and some $p \in \omega$, and because $C(\theta_{m-1})$ and $C(\theta_m)$ match t, we have $\sigma_t^{\theta_{m-1}}(\mathsf{x_q})$ of the form $cf(\dots)$.
Now, because $\mathsf{x_q}$ is suitable we know that $\sigma_t^{\theta_{m-1}}(\mathsf{x_q}) \equiv \sigma_v^{\theta_{m-1}}(\mathsf{x_q})$ or $\sigma_t^{\theta_{m-1}}(\mathsf{x_q}) \gg \sigma_v^{\theta_{m-1}}(\mathsf{x_q})$.

If we suppose $\sigma_t^{\theta_{m-1}}(\mathsf{x_q}) \equiv \sigma_v^{\theta_{m-1}}(\mathsf{x_q})$ then $\sigma_v^{\theta_{m-1}}(\mathsf{x_q})$ is of the form $cf(\dots)$ and $C(\theta_m)$ matches v. This fact contradicts our hypothesis and therefore $\sigma_t^{\theta_{m-1}}(\mathsf{x_q}) \not\equiv \sigma_v^{\theta_{m-1}}(\mathsf{x_q})$.

Assuming $\sigma_t^{\theta_{m-1}}(\mathsf{x_q}) \gg_{\mathsf{p}} \sigma_v^{\theta_{m-1}}(\mathsf{x_q})$ with $\mathsf{p} \neq \varepsilon$ leads to same contradiction. Therefore $\sigma_t^{\theta_{m-1}}(\mathsf{x_q}) \gg_{\varepsilon} \sigma_v^{\theta_{m-1}}(\mathsf{x_q})$.

This means that there exists $a \in \omega$ such that $\sigma_t^{\theta_{m-1}}(\mathsf{x_q}) = cf(\dots, \mathsf{x_{qa}}, \dots)$ and $\sigma_v^{\theta_{m-1}}(\mathsf{x_q}) = \mathsf{x_{qa}}$. And because $\mathsf{x_{qa}}$ is an inductive variable of $\sigma_t^{\theta_{m-1}}(\mathsf{x_q})$ we have $H \in \Gamma_m$ such that $C(H) = C(\theta_{m-1})[\mathsf{x_{qa}}/\mathsf{x_q}]$. Obvioulsy, as $C(\theta_{m-1})$ matches v, $C(H)$ does too. So, to prove that H can be applied to v, we will prove *(a)* $\sigma_v^H(\mathsf{x_{qa}}) = \mathsf{x_{qa}}$ and *(b)* for all variable $\mathsf{x_k} \neq \mathsf{x_{qa}}$, if $\mathsf{x_k}$ is free in H then $\sigma_v^H(\mathsf{x_k}) = \mathsf{x_k}$.

(a) By remark 3.4 $Adr(\mathsf{x_{qa}}, C(H)) = Adr(\mathsf{x_q}, C(\theta_{m-1})) = \mathsf{q}$ and therefore, by remark 1.2, $\sigma_v^H(\mathsf{x_{qa}}) = \sigma_v^{\theta_{m-1}}(\mathsf{x_q}) = \mathsf{x_{qa}}$.

(b) If $\mathsf{x_k}$ is a free variable of H which is not equal to $\mathsf{x_{qa}}$ then $\mathsf{x_k}$ is a free variable of θ_{m-1} which is not equal to $\mathsf{x_q}$ and then, because, in this case, $Adr(\mathsf{x_k}, C(H)) = Adr(\mathsf{x_k}, C(\theta_{m-1})) = \mathsf{k}$, we have $\sigma_v^H(\mathsf{x_k}) = \sigma_v^{\theta_{m-1}}(\mathsf{x_k})$. So to prove $\sigma_v^H(\mathsf{x_k}) = \mathsf{x_k}$ we only have to prove $\sigma_v^{\theta_{m-1}}(\mathsf{x_k}) = \mathsf{x_k}$. Remember that from the clause 2 of suitability for $\mathsf{x_q}$ we have that $\sigma_t^{\theta_{m-1}}(\mathsf{x_k}) = \mathsf{x_k}$. So proving $\sigma_t^{\theta_{m-1}}(\mathsf{x_k}) \equiv \sigma_v^{\theta_{m-1}}(\mathsf{x_k})$ will give us, by definition of $\equiv$, the expected result.

Because $\mathsf{x_k}$ is a free variable of θ_{m-1} and Th_f is a closed formula, the variable $\mathsf{x_k}$ has been introduced by a *Rec* rule. So there exists l $(0 < l < m)$ and a variable $\mathsf{x_p}$ such that

$$\dfrac{\begin{array}{c}\Gamma_{m-1} \vdash \theta_{m-1} \\ \vdots \\ \Gamma_l \vdash \theta_l\end{array}}{\begin{array}{c}\Gamma_{l-1} \vdash \theta_{l-1} \\ \vdots \\ \emptyset \vdash Th_f\end{array}} \; Rec(\mathsf{x_p})$$

where $C(\theta_l) = C(\theta_{l-1})[cf(\dots, \mathsf{x_{pb}}, \dots)/\mathsf{x_p}]$ and $\mathsf{x_{pb}} = \mathsf{x_k}$.
By hypothesis, $C(\theta_{m-1})$ matches t and v, therefore $C(\theta_{l-1})$ and $C(\theta_l)$ does too. Note that because $C(\theta_{m-1})$ and $C(\theta_l)$ are well named terms, $\sigma_t^{\theta_{m-1}}(\mathsf{x_{pb}}) = \sigma_t^{\theta_l}(\mathsf{x_{pb}})$ and $\sigma_v^{\theta_{m-1}}(\mathsf{x_{pb}}) = \sigma_v^{\theta_l}(\mathsf{x_{pb}})$. So to prove $\sigma_t^{\theta_{m-1}}(\mathsf{x_{pb}}) \equiv \sigma_v^{\theta_{m-1}}(\mathsf{x_{pb}})$ we shall prove $\sigma_t^{\theta_l}(\mathsf{x_{pb}}) \equiv \sigma_v^{\theta_l}(\mathsf{x_{pb}})$.
Note also that because $Adr(\mathsf{x_p}, C(\theta_{l-1})) = Adr(cf(\dots, \mathsf{x_{pb}}, \dots), C(\theta_l))$ we have $\sigma_t^{\theta_{l-1}}(\mathsf{x_p}) = cf(\dots, \sigma_t^{\theta_l}(\mathsf{x_{pb}}), \dots)$ and $\sigma_v^{\theta_{l-1}}(\mathsf{x_p}) = cf(\dots, \sigma_v^{\theta_l}(\mathsf{x_{pb}}), \dots)$. Finally, note that because $\mathsf{x_{pb}}$ is a free variable of θ_l, it is an inductive variable of $C(\theta_l)$. Therefore $\sigma_t^{\theta_l}(\mathsf{x_{pb}})$ is an inductive argument of $\sigma_t^{\theta_{l-1}}(\mathsf{x_p})$ and, $\sigma_v^{\theta_l}(\mathsf{x_{pb}})$ is an inductive argument of $\sigma_v^{\theta_{l-1}}(\mathsf{x_p})$.

Now, because $\mathsf{x_p}$ is also supposed to be suitable either $\sigma_t^{\theta_{l-1}}(\mathsf{x_p}) \equiv \sigma_v^{\theta_{l-1}}(\mathsf{x_p})$ or $\sigma_t^{\theta_{l-1}}(\mathsf{x_p}) \gg \sigma_v^{\theta_{l-1}}(\mathsf{x_p})$.

If we suppose $\sigma_t^{\theta_{l-1}}(\mathsf{x_p}) \equiv \sigma_v^{\theta_{l-1}}(\mathsf{x_p})$ then, using lemma 4.1 and the fact that $\sigma_t^{\theta_l}(\mathsf{x_{pb}})$ and $\sigma_v^{\theta_l}(\mathsf{x_{pb}})$ are inductive arguments of, respectively, $\sigma_t^{\theta_{l-1}}(\mathsf{x_p})$ and $\sigma_v^{\theta_{l-1}}(\mathsf{x_p})$, we get $\sigma_t^{\theta_l}(\mathsf{x_{pb}}) \equiv \sigma_v^{\theta_l}(\mathsf{x_{pb}})$.

If we suppose now $\sigma_t^{\theta_{l-1}}(\mathsf{x_p}) \gg \sigma_v^{\theta_{l-1}}(\mathsf{x_p})$, first note that we can't have $\sigma_t^{\theta_{l-1}}(\mathsf{x_p}) \gg_{\boldsymbol{\varepsilon}} \sigma_v^{\theta_{l-1}}(\mathsf{x_p})$ otherwise, by definition of $\gg_{\boldsymbol{\varepsilon}}$, the sub-term $\sigma_v^{\theta_{l-1}}(\mathsf{x_p})$ would be a variable and this would contradict the fact that $C(\theta_l)$ matches v. So there exists $c \in \omega$ and $\mathsf{p}' \in \omega^\omega$ such that $\sigma_t^{\theta_{l-1}}(\mathsf{x_p}) \gg_{\mathsf{c.p'}} \sigma_v^{\theta_{l-1}}(\mathsf{x_p})$.
Using lemma 4.4, we will get the result if we prove $Adr(\sigma_t^{\theta_l}(\mathsf{x_{pb}}), \sigma_t^{\theta_{l-1}y}(\mathsf{x_p})) \neq c$.
Suppose this is not the case : by lemma 4.3, we should have $\sigma_t^{\theta_l}(\mathsf{x_{pb}}) \gg_{\mathsf{p'}} \sigma_v^{\theta_l}(\mathsf{x_{pb}})$. Therefore $\sigma_t^{\theta_l}(\mathsf{x_{pb}})$ should be of the form $cf(\ldots)$. But this is impossible because we know that $\sigma_t^{\theta_l}(\mathsf{x_{pb}}) = \sigma_t^{\theta_{m-1}}(\mathsf{x_{pb}}) = \mathsf{x_{pb}}$. ♠

Proposition 5.2 *Let $\mathcal{E}$ be an equational data for some f and $\mathcal{A}$ a distributing tree for $\mathcal{E}$ with root $\emptyset \vdash Th_f$. If $\mathcal{A}$ satisfies the right terminal state property then for all inductive call $(f(t_1,\ldots,t_n), f(v_1,\ldots,v_n))$ of $\mathcal{E}$, there exists i $(1 \leq i \leq n)$ such that $t_i \gg v_i$.*

In other words, if we put $t = f(t_1,\ldots,t_n)$ and $v = f(v_1,\ldots,v_n)$ then there must exist i $(1 \leq i \leq n)$ and $\mathsf{q}' \in \omega^\omega$ such that $t_{\mathsf{i}} \gg_{\mathsf{q'}} v_{\mathsf{i}}$.

Proof
Because $\mathcal{A}$ satisfies the right terminal state property, we know that there exists $\Gamma_e \vdash \theta_e$ which is a leaf of $\mathcal{A}$ such that Γ_e contains an hypothesis H which can be applied to v. Consider the branch of $\mathcal{A}$, $\Gamma \vdash \theta_0, \ldots, \Gamma_e \vdash \theta_e$ where $\Gamma_0 = \emptyset$ and $\theta_0 = Th_f$.
Note that, by definition of the *Rec* rule, for all m $(0 \leq m < e)$, $\Gamma_m \subseteq \Gamma_{m+1}$. Because $\Gamma_0 = \emptyset$, there must exists m $(0 \leq m < e)$ such that $H \notin \Gamma_m$ and $H \in \Gamma_{m+1}$. We are, then, in the following situation :

$$\begin{array}{c}
\Gamma_e \vdash \theta_e \\
\vdots \\
\dfrac{\ldots \quad \Gamma_{m+1}[H] \vdash \theta_{m+1} \quad \ldots}{\Gamma_m \vdash \theta_m}\ Rec(\mathsf{x_q}) \\
\vdots \\
\emptyset \vdash Th_f
\end{array}$$

where $C(H) = C(\theta_m)[\mathsf{x_{qa}}/\mathsf{x_q}]$ for some $a \in \omega$ and $\mathsf{x_{qa}}$ is free in H (see remark 3.2).
Because $Th_f = \forall \mathsf{x_1} D_1 \mathsf{x_1}, \ldots, \forall \mathsf{x_n} D_n \mathsf{x_n} \to Df(\mathsf{x_1},\ldots,\mathsf{x_n})$, there exists i $(1 \leq i \leq n)$ such that $\mathsf{q} = \mathsf{i.q'}$. So, we shall prove that $t_{\mathsf{i}} \gg_{\mathsf{q'}} v_{\mathsf{i}}$. By definition of $\gg_{\mathsf{q'}}$ we have to prove that for all $\mathsf{k} \in \omega^\omega$, if $\mathsf{k} \leq_a \mathsf{q}'$ then $t_{\mathsf{ik}}[\mathsf{q}' - \mathsf{k} \leftarrow \mathsf{x_{qa}}] \equiv v_{\mathsf{ik}}$.

If $\mathsf{q}' = \mathsf{k}$ then $\mathsf{q}' - \mathsf{k} = \boldsymbol{\varepsilon}$ and, in this case, $t_{\mathsf{ik}}[\boldsymbol{\varepsilon} \leftarrow \mathsf{x_{qa}}] = \mathsf{x_{qa}}$. We have then to prove that $v_{\mathsf{ik}} = \mathsf{x_{qa}}$ or, equivalently that $v_{\mathsf{q}} = \mathsf{x_{qa}}$.
Because we supposed that H can be applied to v and because $\mathsf{x_{qa}}$ is free in H, we have a substitution σ_v^H such that $\sigma_v^H(\mathsf{x_{qa}}) = \mathsf{x_{qa}}$. Moreover, as the address of $\mathsf{x_{qa}}$ in $C(H)$ is q, we have that $\sigma_v^H(\mathsf{x_{qa}}) = v_{\mathsf{q}}$ and finally we get that $v_{\mathsf{q}} = \mathsf{x_{qa}}$.

If $\mathsf{k} <_a \mathsf{q}'$, let $h = C(H)$. To establish $t_{\mathsf{ik}}[\mathsf{q}' - \mathsf{k} \leftarrow \mathsf{x_{qa}}] \equiv v_{\mathsf{ik}}$ we will prove the two following equivalences :

(a) $h_{\mathsf{ik}} \equiv v_{\mathsf{ik}}$

(b) $t_{\mathsf{ik}}[\mathsf{q}' - \mathsf{k} \leftarrow \mathsf{x_{qa}}] \equiv h_{\mathsf{ik}}$

Note that because H contains the variable $\mathsf{x_{qa}}$ at the address q and because we supposed $\mathsf{k} <_a \mathsf{q}'$, the sub-term h_{ik} exists.

Let us prove *(a)* : on one hand, because we supposed that H can be applied to v, we get a substitution σ_v^H such that for all free variable x of H, $\sigma_v^H(x) = x$; on the other hand, we know by lemma 4.2 that, to get $h_{\mathsf{ik}} \equiv v_{\mathsf{ik}}$, it is enough to prove *(a1)* h_{ik} is not a variable and *(a2)* for all variable $x \in \mathcal{V}i(h)$, we have that $\sigma_v^H(x) = x$.

(a1) Recalling $\mathsf{k} <_a \mathsf{q}'$ and $Adr(\mathsf{x_{qa}}, h) = \mathsf{q} = \mathsf{iq}'$, we have that the variable $\mathsf{x_{qa}}$ is a strict sub-term of h_{ik} and then h_{ik} can't be a variable.

To get *(a2)*, we have to prove that if $x \in \mathcal{V}i(h)$ then x is free in H. According to remark 3.5, H is a paraclosed formula excepted for $\mathsf{x_{qa}}$. So if $x \in \mathcal{V}i(h)$ and if $x \neq \mathsf{x_{qa}}$ we get that x is free in H. And even if $\mathsf{x_{qa}} \in \mathcal{V}i(h)$ we know that it is free in H.

Let us now prove *(b)* : by hypothesis, we have $C(\theta_m)$ matching t. Therefore, using remark 3.2, we have σ_t^H such that $\sigma_t^H(h) = t$. Clearly, the substitution σ defined by $\sigma(\mathsf{x_{qa}}) = \mathsf{x_{qa}}$ *and* $\sigma(x) = \sigma_t^H(x)$ *if* $x \neq \mathsf{x_{qa}}$ is such that $\sigma(h) = t[\mathsf{q} \leftarrow \mathsf{x_{qa}}]$ and, because $Adr(t_\mathsf{q}, t) = Adr(t_{\mathsf{q'-k}}, t_{\mathsf{ik}})$, we also have that $\sigma(h_{\mathsf{ik}}) = t_{\mathsf{ik}}[\mathsf{q'} - \mathsf{k} \leftarrow \mathsf{x_{qa}}]$. To prove *(b)*, using lemma 4.2, it suffices to prove that for all $\mathsf{x_p} \in \mathcal{V}i(h_{\mathsf{ik}})$, $\sigma(\mathsf{x_p}) = \mathsf{x_p}$.

If $\mathsf{x_{qa}} \in \mathcal{V}i(h_{\mathsf{ik}})$ we get directly our result by definition of σ.

If $\mathsf{x_p} \neq \mathsf{x_{qa}}$ and $\mathsf{x_p} \in \mathcal{V}i(h_{\mathsf{ik}})$, as we've seen when we proved *(a)*, we have $\sigma_v^H(\mathsf{x_p}) = \mathsf{x_p}$. By remark 3.3, $Adr(\mathsf{x_p}, h) = \mathsf{p}$. Therefore, by remark 1.2, $\sigma_t^H(\mathsf{x_p}) = t_\mathsf{p} = \sigma(\mathsf{x_p})$. Because $\sigma_v^H(\mathsf{x_p}) = \mathsf{x_p}$, we have $\mathsf{x_p} \in \mathcal{V}ar(v)$; because (t, v) is a well named pair of terms, $\mathsf{x_p} \in \mathcal{V}ar(t)$ and $t_\mathsf{p} = \mathsf{x_p}$. Therefore $\sigma(\mathsf{x_p}) = \sigma_t^H(\mathsf{x_p}) = \mathsf{x_p}$. ♠

Proposition 5.3 *Let $\mathcal{E}$ be an equationnal data for some f and $\mathcal{A}$ be a distributing tree for $\mathcal{E}$. If $\mathcal{A}$ satisfies the right terminal state property then every recursion variable of $\mathcal{A}$ is suitable.*

Proof

Suitability of variables is defined relatively to a judgement and a set of inductive calls. So, under the hypothesis that $\mathcal{A}$ satisfies the right terminal state, we want to prove that for all inductive call (t, v) of $\mathcal{E}$ and for all judgement $\Gamma \vdash \theta$ of $\mathcal{A}$ (which is not e leaf of $\mathcal{A}$), if $C(\theta)$ matches t and if $\mathsf{x_q}$ is the recursion variable of θ then $\mathsf{x_q}$ is suitable for (t, v) and $\Gamma \vdash \theta$.

Because $\mathcal{A}$ is a distributing tree, there exists a leaf of $\mathcal{A}$, say $\Gamma_e \vdash \theta_e$, such that $C(\theta_e) = t$. Let $\Gamma_0 \vdash \theta_0, \ldots \Gamma_e \vdash \theta_e$ be the branch of $\mathcal{A}$ such that $\Gamma_0 = \emptyset$ and $\theta_0 = Th_f$, let (t, v) be an inductive call such that $C(\theta_e)$ matches t. We are going to prove that for all i $(0 \leq i < e)$, if $\mathsf{x_q}$ is the recursion variable of θ_i then $\mathsf{x_q}$ is suitable for $\Gamma_i \vdash \theta_i$ and (t, v).

Reasoning as we did in proposition 5.1, there exists a judgement $\Gamma_m \vdash \theta_m$ and an hypothesis $H \in \Gamma_{m+1}$ which can be applied to v. Let figure the situation this way :

$$\begin{array}{c} \Gamma_e \vdash \theta_e \\ \vdots \\ \dfrac{\Gamma_{m+1}[H] \vdash \theta_{m+1}}{\Gamma_m \vdash \theta_m} \; Rec(\mathsf{x_q}) \\ \vdots \\ \emptyset \vdash Th_f \end{array}$$

where $C(H) = C(\theta_m)[\mathsf{x_{qa}}/\mathsf{x_q}]$ for some $a \in \omega$ and Γ_m doesn't contain H.

We will now distinguish three cases : *(a)* the case of the $\Gamma_j \vdash \theta_j$'s where $m < j < e$; *(b)* the case of $\Gamma_m \vdash \theta_m$ and *(c)* the case of the $\Gamma_j \vdash \theta_j$ where $0 \leq j < m$.

(a) Because H can be applied to v, $C(\theta_m)$ matches v. Therefore, using the remark 3.1, for all i $(0 \leq i \leq m)$, $C(\theta_i)$ matches v. Because H can be applied to v, because the variable $\mathsf{x_{qa}}$ is free in H and its address in $C(H)$ is q, we have $v_\mathsf{q} = \mathsf{x_{qa}}$. Therefore $C(\theta_{m+1}) = C(\theta_m)[cf(\ldots)/\mathsf{x_q}]$, for some constructor cf, and $C(\theta_{m+1})$ doesn't match v. It is easy to check that for all j $(m < j < e)$, $C(\theta_j)$ doesn't match v. So the suitability property is trivially satisfied by the recursion variables of the θ_j's.

(b) Let us examine now the case of θ_m : is $\mathsf{x_q}$ suitable for θ_m and (t, v) ?

As it is done in proposition 5.2 we get $\sigma_t^\theta(\mathsf{x_q}) \gg_{\mathcal{E}} \sigma_v^\theta(\mathsf{x_q})$. So the first clause of the suitability property is satisfied.

It remains now to satisfy the clause 2. So let $\mathsf{x_k} \neq \mathsf{x_q}$ be a free variable of θ_m. Using remark 3.5, $\mathsf{x_k}$ is also a free variable of H. Because we supposed that H can be applied to v, we get $\sigma_v^H(\mathsf{x_k}) = \mathsf{x_k}$. Using remark 3.3, we have $Adr(\mathsf{x_k}, C(H)) = \mathsf{k} = Adr(\mathsf{x_k}, C(\theta_m))$. So $\sigma_v^\theta(\mathsf{x_k}) = v_\mathsf{k} = \sigma_v^H(\mathsf{x_k}) = \mathsf{x_k}$.

(c) We have finally to examine the case of the recursion variables of the θ_j's where $0 \leq j < m$. Before doing this we prove the following :
let $\tau = C(\theta_m)$, for all $\mathsf{p} \in \omega^\omega$, if τ_p is a strict sub-term of $C(\theta_m)$ which is not a variable then

(c1) if $\varepsilon <_a \mathsf{p} <_a \mathsf{q}$ then $\sigma_t^{\theta_m}(\tau_\mathsf{p}) \gg_{\mathsf{q}-\mathsf{p}} \sigma_v^{\theta_m}(\tau_\mathsf{p})$

(c2) else $\sigma_t^{\theta_m}(\tau_\mathsf{p}) \equiv \sigma_v^{\theta_m}(\tau_\mathsf{p})$

(c1) As in proposition 5.2, there exists $i \in \omega$ and $\mathsf{q}' \in \omega^\omega$ such that $\mathsf{q} = \mathsf{i}\mathsf{q}'$ and $t_\mathsf{i} \gg_{\mathsf{q}'} v_\mathsf{i}$. If we suppose $\mathsf{p} <_a \mathsf{q}$ then, using lemma 4.3, $\sigma_t^{\theta_m}(\tau_\mathsf{p}) \gg_{\mathsf{q}-\mathsf{p}} \sigma_v^{\theta_m}(\tau_\mathsf{p})$.
(c2) Otherwise, because the sub-term of address q in $C(\theta_m)$ is the variable x_q, there is no sub-term τ_p of $C(\theta_m)$ such that $\mathsf{q} <_a \mathsf{p}$. Then p and q are not comparable and τ_p doesn't contain x_q.
Using remark 3.2, it is easy to check that τ_p is also a sub-term of $C(H)$ and $\sigma_v^H(\tau_\mathsf{p}) = \sigma_v^{\theta_m}(\tau_\mathsf{p})$ and $\sigma_t^H(\tau_\mathsf{p}) = \sigma_t^{\theta_m}(\tau_\mathsf{p})$. To prove $\sigma_t^{\theta_m}(\tau_\mathsf{p}) \equiv \sigma_v^{\theta_m}(\tau_\mathsf{p})$ we will prove *(c21)* $\tau_\mathsf{p} \equiv \sigma_v^H(\tau_\mathsf{p})$ and *(c22)* $\tau_\mathsf{p} \equiv \sigma_t^H(\tau_\mathsf{p})$ using lemma 4.2.

(c21) Let $\mathsf{x}_\mathsf{r} \in \mathcal{V}i(\tau_\mathsf{p})$. Because $\mathcal{V}i(\tau_\mathsf{p}) \subseteq \mathcal{V}i(C(\theta_m))$ and because H is paraclosed excepted for x_{qa}, the variable x_r is free in H. We supposed that H can be applied to v, so $\sigma_v^H(\mathsf{x}_\mathsf{r}) = \mathsf{x}_\mathsf{r}$. Therefore, $\sigma_v^H(\tau_\mathsf{p}) \equiv \tau_\mathsf{p}$.

(c22) Because $C(H)$ is a well named term excepted for x_{qa}, we have that $Adr(\mathsf{x}_\mathsf{r}, C(H)) = \mathsf{r}$ and then, $\sigma_v^H(\mathsf{x}_\mathsf{r}) = t_\mathsf{r}$. Because $\sigma_v^H(\mathsf{x}_\mathsf{r}) = \mathsf{x}_\mathsf{r}$ we have $\mathsf{x}_\mathsf{r} \in \mathcal{V}ar(v)$ and, because (t, v) is a well named pair of terms, $t_\mathsf{r} = \mathsf{x}_\mathsf{r}$. So $\sigma_t^H(\mathsf{x}_\mathsf{r}) = \mathsf{x}_\mathsf{r}$ and $\sigma_t^H(\tau_\mathsf{p}) \equiv \tau_\mathsf{p}$.

Let us return now to the recursion variables the θ_j's for $0 \leq j < m$. Let x_p be the recursion variable of θ_j. We want to prove that x_p is suitable for θ_j and (t, v).
If $\tau = C(\theta_m)$, we know that $C(\theta_j)$ matches τ (remark 3.1). So we can put $\sigma_\tau^{\theta_j}(\mathsf{x}_\mathsf{p}) = \tau_\mathsf{p}$. We also have $\sigma_t^{\theta_j}(\mathsf{x}_\mathsf{p}) = \sigma_t^{\theta_m}(\tau_\mathsf{p})$ and $\sigma_v^{\theta_j}(\mathsf{x}_\mathsf{p}) = \sigma_v^{\theta_m}(\tau_\mathsf{p})$. Because x_p is the recursion variable of θ_j, it doesn't appear in θ_m and then, τ_p is not a variable.

From *(c1)* and *(c2)* we get that x_p satisfies the clause 1 of suitability.

Let us prove now that the clause 2 is trivially satisfied (*e.g.* we can't have $\sigma_t^{\theta_j}(\mathsf{x}_\mathsf{p}) \gg_\varepsilon \sigma_v^{\theta_j}(\mathsf{x}_\mathsf{p})$).
If $\mathsf{p} <_a \mathsf{q}$ then $\mathsf{q} - \mathsf{p} \neq \varepsilon$ and, in this case, we have $\sigma_t^{\theta_j}(\mathsf{x}_\mathsf{p}) \gg_{\mathsf{q}-\mathsf{p}} \sigma_v^{\theta_j}(\mathsf{x}_\mathsf{p})$. By definition of $\gg$, we know that $\sigma_v^{\theta_j}(\mathsf{x}_\mathsf{p})$ contains a variable (namely, x_{qa}) at the address $\mathsf{q} - \mathsf{p}$. As $\mathsf{q} - \mathsf{p} \neq \varepsilon$, the variable x_{qa} is a strict sub-term of $\sigma_v^{\theta_j}(\mathsf{x}_\mathsf{p})$ can't therefore be a variable. So we can't have $\sigma_t^{\theta_j}(\mathsf{x}_\mathsf{p}) \gg_\varepsilon \sigma_v^{\theta_j}(\mathsf{x}_\mathsf{p})$.
Otherwise, by *(c2)*, we have $\sigma_t^{\theta_j}(\mathsf{x}_\mathsf{p}) \equiv \sigma_v^{\theta_j}(\mathsf{x}_\mathsf{p})$ and it is easy to check that, in this case, we can't have $\sigma_t^{\theta_j}(\mathsf{x}_\mathsf{p}) \gg_\varepsilon \sigma_v^{\theta_j}(\mathsf{x}_\mathsf{p})$. ♠

6 Practising the prover

6.1 One example

The equations

It is possible (but not usual) to define the function which computes the sum of the elements of a list by the following set of equations.

$$\begin{array}{l} Ad_l(Nil) = 0 \\ Ad_l(Cons(0, ns)) = Ad_l(ns) \\ Ad_l(Cons(s(n), ns)) = s(Ad_l(Cons(n, ns))) \end{array}$$

We will now give the R-proof which is generated by our prover. We will distinguish the recursion part of the proof from the two other parts.

The recursion tree

Let N and LN be respectively the integer and list of integer data types. We have to prove : $\forall x LNx \rightarrow NAd_l(x)$. The corresponding recursion tree is :

$$\dfrac{\emptyset \vdash NAd_l(Nil) \qquad \dfrac{\Gamma_1 \vdash NAd_l(Cons(0,x')) \qquad \Gamma_2 \vdash NAd_l(Cons(s(n'),x'))}{\Gamma_1 \vdash \forall n Nn \rightarrow NAd_l(Cons(n,x'))}{\scriptstyle Rec(n)}}{\emptyset \vdash \forall x LNx \rightarrow NAd_l(x)}{\scriptstyle Rec(x)}$$

where $\Gamma_1 = \{Lnx', NAd_l(x')\}$ and $\Gamma_2 = \Gamma_1 \cup \{Nn', NAd_l(Cons(n',x'))\}$. This recursion tree is a distributing tree which satisfies the right terminal state.

The achievement of the proof

In order to get an R-proof it remains to get a proof of each leaf of the recursion tree using once the *Eq* rule and then using the *Hyp*, *Thm* and *Ax* rules. Those three proofs are :

$$\dfrac{\dfrac{}{\emptyset \vdash N(0)}{\scriptstyle Thm(0)}}{\emptyset \vdash NAd_l(Nil)}{\scriptstyle Eq} \qquad \dfrac{\dfrac{}{\Gamma_1 \vdash NAd_l(x')}{\scriptstyle Hyp}}{\Gamma_1 \vdash NAd_l(Cons(0,x'))}{\scriptstyle Eq} \qquad \dfrac{\dfrac{\dfrac{}{\Gamma_2 \vdash N(Ad_l(Cons(n',x')))}{\scriptstyle Hyp}}{\Gamma_2 \vdash Ns(Ad_l(Cons(n',x')))}{\scriptstyle Thm(s)}}{\Gamma_2 \vdash NAd_l(Cons(s(n'),x'))}{\scriptstyle Eq}$$

This proof illustrates the fact that the relation $\gg$ between lists does not reduce to the standard successor relation. Actually the usual successor of $Cons(s(n), ns)$ is ns. In our case we have $Cons(s(n), ns) \gg_1 Cons(n, ns)$.

6.2 R-proofs and orderings

We want now to point out some features of our prover concerning orderings underlying inductive proofs.

As our prover is able to prove (without backtrack) the termination of Ackermann's function defined as follows,

$$\begin{array}{l} Ack(0,m) = s(m) \\ Ack(s(n),0) = Ack(n,s(0)) \\ Ack(s(n),s(m)) = Ack(n, Ack(s(n),m)) \end{array}$$

we can say that it is able to make inductive proofs using lexicographic ordering on pairs of natural numbers. The first remark we want to give about it is that our prover is not limited to primitive recursive functions. The second remark is that even if one reverses, in the definition of Ackermann's function, the order of the arguments like this :

$$\begin{array}{l} Ack'(m,0) = s(m) \\ Ack'(0,s(n)) = Ack'(s(0),n) \\ Ack'(s(m),s(n)) = Ack'(Ack'(m,s(n)),n) \end{array}$$

the prover will still be able to make the proof. That is to say that it analyses the function definitions not only at a syntactical level but also at a logical one.
Moreover, the prover is able to detect more subtle orderings as it is shown in the above example of Ad_l function. Actually, the proof our algorithm produces for this function is equivalent to an inductive proof using lexicographic ordering on the pair made of the length of the list and the first element of the list.
Let us give, to finish with those informal remarks, the following definition :

$$\begin{array}{l} A(0,y,0) = s(y) \\ A(0,y,s(z)) = A(0,s(y),z) \\ A(s(x),0,z) = A(x,s(0),s(z)) \\ A(s(x),s(y),z) = A(x, A(s(x),y,S(z)), s(z)) \end{array}$$

This is an *"Ackermann-like"* function defined on three arguments. Our prover is quite able to give a termination proof for it. The equivalent underlying ordering on tuples one can use to prove termination of A is not immediately expressible as a lexicographic ordering. We say that it is a *treelike lexicographic ordering.* Let $<$ be the usual ordering on natural numbers, we can define the $<_{TLO}$ ordering on tuples, used for the function A, by :

$(x, y, z) <_{TLO} (x', y', z')$ iff

$x < x'$ or

if $x = 0$ then $z < z'$ else $y < y'$.

This ability of our prover to synthetize proofs involving, from the mathematical view point, non trivial orderings is due to the fact that it doesn't use a library of predefined orderings but makes an analysis of the given definition of the function whose termination he has to prove.

References

[1] J.L Krivine, M. Parigot *Programming with proofs* ; in SCT 87 (1987), Wendish-Rietz ; in EIK 26 (1990) pp 146-167.

[2] D. Leivant *Reasonning about functionnal programs and complexity classes associated with type discipline* ; in FOCS (1983) pp 460-469.

[3] P. Manoury, M. Parigot, M. Simonot ProPre *: a programming language with proofs* ; in proceedings LPAR 92, LNAI 624.

[4] P. Manoury and M. Simonot *Des preuves de totalité de fonctions comme synthèse de programmes* ; Phd Thesis.

[5] M. Parigot *Recursive programming with proofs*; in TCS 94 (1992) pp 335-356.