International Journal of Foundations of Computer Science

AN EXTENSION OF AN AUTOMATED TERMINATION METHOD OF RECURSIVE FUNCTIONS*

FAIROUZ KAMAREDDINE
Department of Computing and Electrical Engineering, Heriot-Watt University, Edinburgh EH14 4AS, Scotland.
fairouz@cee.hw.ac.uk

and

FRANCOIS MONIN
IRISA, Campus de Beaulieu, Rennes Cedex 35042, France.
monin@irisa.fr

Received (received date)
Revised (revised date)
Communicated by Editor's name

ABSTRACT

Inductive proofs are commonly used in automated deduction systems or functional programming, such as for instance the *ProPre* system, for establishing the termination of recursively defined functions. Such proofs deal with the structural orderings of the term algebras that define the domain of the functions. However there exists other interesting functions whose termination requires different underlying orderings. To treat a class of such functions that are not taken into account by systems such as *ProPre*, we develop termination properties that can be shown automatically. In contrast with the *ProPre* system that builds formal trees based on inductive proofs, we generate measures that satisfy an extended termination property and well-founded orderings which ensure the termination of the functions.

1. Introduction

Termination of functions defined on recursive data structures is an important property in the development of correct software such as functional programming and automated deduction. The *standard approach* to show the termination of recursively defined functions is to prove that the arguments in each recursive call are smaller than the initially given input with respect to a well-founded ordering.

Usually, most termination methods use predefined orders, or lexicographic combinations of it (see e.g. [30, 24, 26]), or orderings given by the users (see e.g. [3, 6]). However, in the standard approach, finding suitable orderings is essential for the automation of the termination. In [8] a method was developed to automatically syn-

*Supported by EPSRC GR/L15685. We are grateful for the anonymous referees for their useful comments.

thesise suitable measures based on polynomial orderings and in [22] a termination procedure was proposed to generate orderings with the use of ordinal measures.

The latter method comes from the analysis of another approach, the *formal approach*, that relies on *formal termination proofs* [21] built in a natural deduction style. The formal approach is essentially used to allow the extraction of λ-terms that compute the functions as codes of programs (see e.g. [16]). Based on this paradigm of *proofs as programs*, the *ProPre* system, designed in [20], shows the automated termination of a recursive function by building a (partial) formal tree, associated with the specification of the function, which satisfies the so-called *right terminal state property*. Due to this property, *ProPre* is then able to derive, from the partial tree, a complete formal proof tree ensuring the termination of the function.

It has been shown in [22, 10] that it is possible to extract well-founded orderings, namely ordinal measures, from the above formal proofs devised in the fully automated system *ProPre* [21, 20], such that the arguments in each recursive call of the given function are smaller than the initially given input (i.e., the calls decrease wrt the well-founded ordering). The extraction of a special class of measures, called *ramified measures*, appears as a useful way to find out new orderings as these measures are in particular not limited to lexicographic orderings.

In this paper we propose to extend both the termination procedure of *ProPre* and the extraction of new orderings. In contrast to earlier work where the synthesis of suitable measures especially relied on *ProPre*'s procedure, we do not work in the context of formal proofs but in a setting released from the particular logical framework of *ProPre*. The new context allows us to construct trees associated with the specifications of functions which must satisfy a property, called the *hierarchical property*. Instead of finding a *formal* proof, we can now directly infer measures, from the trees, whose decreasing property is ensured by the *hierarchical property*.

Work already carried out on the concept of trees developed from specifications includes: *completeness of definition* (see e.g. [9]), *test sets* and *inductive reducibility* (see e.g. [23, 13, 2]) and *ProPre* itself. But such trees have mainly been used for proving inductive properties for equational or conditional theories, or in the case of *ProPre*, trees appear in a *formal context* where they takes advantage of the structure of the function definition. However, the main property that allows the system to find termination proofs of recursive functions is not simply the definition of trees but the *right terminal state property* that they must satisfy. This one gives the soundness of the method and also characterises in some sense the class of the recursively defined functions that can be proven terminating in the system.

Like the above approaches, we make use of some kind of trees as in [22] with some refinements, in particular to deal specifications of incompletely defined functions. However, the novelty of our termination method relies on the hierarchical property developed in this paper and on the introduction of new measures.

Functional programs can also be regarded as term rewriting systems. However, due to the special forms of functions, it seems more convenient to consider in the case of functional programming, well-founded orderings for which only the arguments of the function and the recursive subcalls are taken into account. Compare this with

rewriting systems where the usual orderings are considered for all the left-hand sides and right-hand sides of the terms, as it is for instance with the recursive path ordering [5]. In particular, the rewrite terms that are proven terminating by such orderings are characterised by the *simple termination* property [28], namely the *monotonicity* and the *subterm* property. Our results imply that this condition is not necessary and we are able to deal with a larger set of functions even those whose termination does not require these notions of simple termination, monotonicity and subterm property.

The above mentioned standard approach is often used by theorem provers such as the well-known Nqthm prover [3], which aim at establishing the decreasing property of measures on the recursive calls of the algorithms. But usually the measures are a lexicographic combination of a fixed ordering or are given by the user. The present method aims at providing suitable measures in an *automated way* and therefore could be used by other theorem provers by providing the measures obtained from the formal proofs in *ProPre* [22, 12].

Hence, our work has the advantage that now we can automatically establish:

- the termination of all those functions that could be shown terminating by *ProPre* (and hence whose specifications are complete and whose termination methods require structural induction),
- the termination of functions whose specifications are not necessarily complete and whose termination requires other orderings than those used in structural induction, simple termination, monotonicity and the subterm property,
- suitable measures that could be passed to other theorem provers to be used in establishing the decreasing number of recursive calls.

The paper is divided as follows:

- In Section 2, we introduce the *ProPre* system where the formal framework has been fully abstracted while keeping the termination part. This gives the *term distributing trees* for specifications of recursive functions, and the *abstracted terminal state property* satisfied by term distributing trees.
- In Section 3, we extend the notion of a term distributing tree into the *recursive term distributing tree* in order to consider recursive functions that may be incomplete in the sense of [29]. We also introduce the *split specifications* that enable us to enlarge the set of term distributing trees.
- In Section 4, we introduce a termination property called the *hierarchical property*. We show that our notion strictly includes the *ProPre* notion of *the right terminal state property* by establishing in Theorem 1 that if a distributing tree $\mathcal{A}$ has the terminal state property in the system *ProPre*, then $\mathcal{A}$ has the hierarchical property and that the opposite does not hold.
- In Section 5, in order to make sense of the concept of the *hierarchical property* satisfied by the recursive term distributing trees, we explain how it is possible

from specifications, to define ordinal measures against trees. Our main theorem of this section, Theorem 2, establishes that our new notion of hierarchical property implies that the measures decrease in the recursive call of the functions, and as a consequence enable us to establish the termination of functions. We also explain why the ramified measures that come from the analysis of the *ProPre* system are not suited in the extended context. The inadequacy of the earlier measures and the necessity to build new ordinal measures characterise in some sense the new class of functions that extend those functions whose termination can be established in *ProPre*.

2. The *ProPre* system in an abstract context

The *ProPre* system is a program synthesis system presented in [20, 21, 19] based on the paradigm of *Programming by Proofs*. In this approach programs are *coded* by λ-terms extracted from proofs of termination of functions defined by a set of equations. The extraction is obtained by syntactical termination proofs in a formal deduction style exploiting the Curry-Howard correspondence. The theory can be found for instance in [16, 18, 17, 25]. In this system, the user can specify data types and functions in an ML like syntax, but when compiling, a fully automated proof search strategy is used. The input of that search strategy is a specification of a function and the output is either a termination proof providing a λ-term that compute the function or an error message in case of failure. Notice that the termination method of a first version of *ProPre* was also implemented in a former version of Coq in [19] to deal with first order definitions of functions.

The analysis of the formal proof trees obtained in the system made it possible to relate measures [22, 10, 11] that have the decreasing property in the recursive calls of the specifications of the functions. These measures can be defined from *distributing trees*, devised in [20], which are partial trees; and their decreasing property relies on the notion of a *right terminal state property* that must satisfy the distributing trees in *ProPre* in order to be extended into a complete formal proof trees.

Though the specifications are first order equations, the logical framework of the programming language designed in *ProPre* is a second order language which is mandatory by the theory [16, 25]. That is, in order to associate λ-terms to functions, the system builds proof trees with second order formulas which are characterised, as earlier mentioned, by the right terminal state property. An analysis in [12] shows that it was possible to abstract all proposition informations from each proof tree so that one can look at the skeleton form of the proof tree, called *term distributing tree*, where now only first order is involved giving rise the notion of *abstracted terminal state property* instead of the right terminal state property. Notice that the study of [12] has shown that one can also go back to the former proof trees from the *skeleton tree* and the abstracted terminal state property. Therefore things become clearer since in some sense the termination part can be further investigated independently of the extraction part of λ-terms. The connection between formal proofs and abstracted property with measures can be illustrated by Figure 1.

Because in this paper we consider the termination part and not the extraction

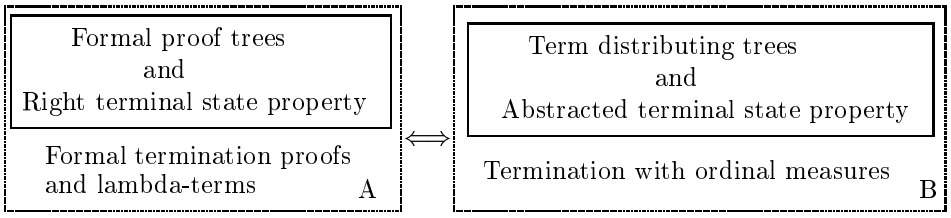


Figure 1: A connection between the two approaches.

part of λ-terms, we will describe in this section the right part of the above picture. We refer to [25, 20] for details on the extraction of formal proofs in *ProPre* and of the associated λ-terms from function specifications We refer to [12] for detail on how one can establish the correspondence of the two parts in the above picture.

Note that the definitions presented in the section below, that are basic definitions, do not only concern ProPre but our work too.

2.1. Specifications

Before presenting the term distributing trees and the abstracted terminal state property in the next section, we introduce the specifications of functions that can be defined in the system *ProPre*. Although they play an important role in the functional programming language of the system *ProPre*, we do not mention the definition of data types of *ProPre* for the sake of simplicity, but we will assume for our purpose a set of *sorts* for the definition of the types of the functions. We will also use here, the terminology of rewrite systems. Though the first definitions below apply to higher order as well, these are useful in the presentation of the specifications that are first order definitions of the functions considered in *ProPre*.

Definition 2.1. (functions) We assume a set $\mathcal{F}$ of *function symbols*, called signature, and a set S of *sorts*. To each function $f \in \mathcal{F}$ we associate a natural number n that denotes its *arity* and a *type* $s_1, \ldots, s_n \to s$ with $s, s_1, \ldots, s_n \in S$. We may write $f : s_1, \ldots, s_n \to s$ to introduce both the function f and its type. A function is called constant if its arity is 0. We assume that the set of functions $\mathcal{F}$ is divided into two disjoint sets $\mathcal{F}_c$ and $\mathcal{F}_d$. Functions in $\mathcal{F}_c$ (which includes the constants) are called *constructor symbols* or *constructors* and those in $\mathcal{F}_d$ are called *defined function symbols* or *defined functions*.

Definition 2.2. (terms) Let $\mathcal{X}$ be a countable set of *variables* disjoint from $\mathcal{F}$. We assume that only one sort is associated to each variable of $\mathcal{X}$ and that for each sort s there is a countable number of variables in $\mathcal{X}$ of sort s. If s is a sort and if F and X are respectively subsets of $\mathcal{F}_c \cup \mathcal{F}_d$ and $\mathcal{X}$, then the set $\mathcal{T}(F, X)_s$ of the terms of sort s is the smallest set such that:

1. every element x of X of sort s is a term of sort s,
2. if f is a function in $\mathcal{F}$ of type $s_1, \ldots, s_n \to s$ and if $t_1, \ldots, t_n$ are terms respectively of sorts $s_1, \ldots, s_n$, then $f(t_1, \ldots, t_n)$ is a term of sort s.

If X is empty, $\mathcal{T}(F,X)_s$ is also denoted by $\mathcal{G}(F)_s$. The set $\mathcal{T}(F,X)$ is $\bigcup_{s\in S}\mathcal{T}(F,X)_s$ for every subset F of $\mathcal{F}$ and every subset X of $\mathcal{X}$. An element of $\mathcal{G}(\mathcal{F})_s$ is called a *ground term* (of sort s); i.e., no variable occurs in a ground term. An element of $\mathcal{T}(\mathcal{F}_c,\mathcal{X})_s$ is called a *constructor term* (of sort s); i.e., every function symbol which occurs in a ground term is a constructor symbol. An element of $\mathcal{T}(\mathcal{F}_c,\mathcal{X})_s \cap \mathcal{G}(\mathcal{F})_s = \mathcal{G}(\mathcal{F}_c)_s$ is called a *ground constructor term* (of sort s). If $c : s$ denotes a constant (of sort s), i.e. its arity is 0, the *constant term* $c()$ (of sort s) is also denoted c. For each term t, $\mathcal{V}ar(t)$ denotes the set of variables that occur in t.

Definition 2.3. (substitutions) A *sorted ground substitution* σ is a substitution, i.e. a mapping from the set $\mathcal{X}$ of variables to the set of terms $\mathcal{T}(\mathcal{F},\mathcal{X})$, such that for every sort s and every variable x of sort s, $\sigma(x)$ is a ground term of sort s. A *sorted constructor substitution* σ is a substitution such that for every sort s and every variable x of sort s, $\sigma(x)$ is a constructor term of sort s.

Any substitution σ from Definition 2.3 can be extended, as usual, into a mapping from $\mathcal{T}(\mathcal{F},\mathcal{X})$ to $\mathcal{T}(\mathcal{F},\mathcal{X})$, such that $\sigma(\mathcal{T}(\mathcal{F},\mathcal{X})_s) \subseteq \mathcal{T}(\mathcal{F},\mathcal{X})_s$ for each sort s.

Definition 2.4. (rewrite system) A rewrite system $\mathcal{R}$ is a subset of $\mathcal{T}(\mathcal{F},\mathcal{X}) \times \mathcal{T}(\mathcal{F},\mathcal{X})$ with $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l)$ for each element (l,r) of $\mathcal{R}$. An element (l,r) of $\mathcal{R}$ is called a *rewrite rule* and is denoted by $l \to r$. A rewrite rule $l \to r$ is called *left-linear* iff each variable occurs only once in the left-hand side l of the rewrite rule $l \to r$. A rewrite system is *non overlapping* iff no left-hand sides unify each other.

In functional programming languages close to ML that use data types, functions are described by sets of *equations* whose terms have the same type, and the arguments of the functions are constructor terms. The specifications of the functions can be described in term of rewrite systems that correspond to first order equations:

Definition 2.5. (specification) A *specification* or *a (sorted) constructor system* $\mathcal{E}$ of a function $f : s_1,\ldots,s_n \to s$ in $\mathcal{F}_d$ is a non overlapping rewriting system of left-linear rules $\{e_1 \to e'_1,\ldots,e_p \to e'_p\}$ such that for all $1 \le i \le p$, e_i is of the form $f(t_1,\ldots,t_n)$ with $t_j \in \mathcal{T}(\mathcal{F}_c,\mathcal{X})_{s_j}$, $j = 1,\ldots,n$, and $e'_i \in \mathcal{T}(\mathcal{F}_c \cup \mathcal{F}_d,\mathcal{X})_s$.

The expression *sorted* in the above definitions may be omitted, and we will say *constructor substitution* instead of *sorted constructor substitution* for short.

Definition 2.6. (recursive calls)
Let $\mathcal{E}$ be a specification of a function $f : s_1,\ldots,s_n \to s$. A *recursive call* of f is a pair $(f(t_1,\ldots,t_n), f(u_1,\ldots,u_n))$ where $f(t_1,\ldots,t_n)$ is a left-hand side of a rewrite rule of $\mathcal{E}$ and $f(u_1,\ldots,u_n)$ is a subterm of the corresponding right hand side.

To illustrate the above definitions with examples, let $nat \in S$ be the sort of natural numbers, with the constant 0 in $\mathcal{F}_c$ of type nat and the constructor successor $s : nat \to nat$ in $\mathcal{F}_c$. Let $bool$ be the boolean sort in S, the constants $true$ and $false$ in $\mathcal{F}_c$ be of type $bool$, and the function $not : bool \to bool$ be in $\mathcal{F}_d$ with the specification given by the usual rules: $not(true) \to false$, $not(false) \to true$.

Example 2.7. Let $Ack : nat, nat \to nat$ be the Ackermann function in $\mathcal{F}_d$. A specification $\mathcal{E}_{Ack}$ of the function Ack is given by the rewrite rules: $Ack(0,y) \to s(y)$, $Ack(s(x),0) \to Ack(x,s(0))$, and $Ack(s(x),s(y)) \to Ack(x,Ack(s(x),y))$.

The three recursive calls of the above specification are $(Ack(s(x),0), Ack(x,s(0)))$; $(Ack(s(x),s(y)), Ack(s(x),y))$; and $(Ack(s(x),s(y)), Ack(x,Ack(s(x),y)))$.

Note that the function Ack is completely defined in the following sense: for each (w_1,w_2) in $\mathcal{G}(\mathcal{F}_c)_{nat} * \mathcal{G}(\mathcal{F}_c)_{nat}$ there is a left-hand side $Ack(t_1,t_2)$ of a rewrite rule of the specification $\mathcal{E}_{Ack}$ and a substitution φ with $(\varphi(t_1),\varphi(t_2)) = (w_1,w_2)$. This corresponds to the completeness of definition in [9] (see also e.g. [29]). We give now another example of a specification, coming from [1], whose definition is incomplete.

Example 2.8. Let $\mathcal{E}_{eo}$ be the specification of the function $evenodd : nat, nat \rightarrow bool$ in $\mathcal{F}_d$ with the rewrite rules: $evenodd(x,0) \rightarrow not(evenodd(x,s(0)))$, $evenodd(0,s(0)) \rightarrow false$, and $evenodd(s(x),s(0)) \rightarrow evenodd(x,0)$.

The specification $\mathcal{E}_{eo}$ is incomplete as no left-hand side of the rules of the specification matches the term $evenodd(u,s(s(v)))$ for (u,v) in $\mathcal{G}(\mathcal{F}_c)_{nat} * \mathcal{G}(\mathcal{F}_c)_{nat}$. According to the second argument, the *evenodd* function computes either the *even* function or the *odd* function. The two recursive calls of the above specification are $(evenodd(x,0), evenodd(x,s(0)))$ and $(evenodd(s(x),s(0)), evenodd(x,0))$.

2.2. Term distributing trees and the terminal state property

We present the term distributing trees of [22] which can be viewed as a skeleton form of partial trees built in *ProPre* [20] where all proposition informations have been abstracted. A lot of information is lost from the proof trees by this abstracting operation and different formal trees may reduce to the same term distributing tree [12]. But all the termination information from the formal proof trees is fortunately recovered by our new notion of terminal state property in Definition 2.18.

Definition 2.9. (term distributing trees) Let $\mathcal{E}$ be a specification of a function $f : s_1,\ldots,s_n \rightarrow s$. A *term distributing tree* $\mathcal{A}$ for $\mathcal{E}$ is defined as follows:

1. The root of $\mathcal{A}$ is a tuple of the form $(x_1,\ldots,x_n)$ where x_i is a variable of sort s_i for each $i \leq n$;

2. each node of $\mathcal{A}$ is of the form $(t_1,\ldots,t_n)$ and there is a variable x' of a sort s' in the term $f(t_1,\ldots,t_n)$ such that the set of children of the node, for $x'_1,\ldots,x'_r$ not in $t_1,\ldots t_n$, is $\{(t_1,\ldots t_n)[C(x'_1,\ldots x'_r)/x'],\ C : s'_1,\ldots,s'_r \rightarrow s' \in \mathcal{F}_c\}$.

3. each leaf $(t_1,\ldots,t_n)$ of $\mathcal{A}$ is exactly one left-hand side $f(t_1,\ldots,t_n)$ of an equation of $\mathcal{E}$ (up to the root function symbol f and the renaming of variables of the equations).

For instance if $(s(x),y)$ is a node in a term distributing tree where s is the successor function, then its two children can be either of the form $(s(0),y)$ and $(s(s(x')),y)$, or of the form $(s(x),0)$ and $(s(x),s(y'))$. An example of a distributing tree is given with Example 2.11 below using also Notation 2.10. The following notation will be particularly useful for the next Section and for the definition of ordinal measures in Section 5.

Notation 2.10. Let $\mathcal{E}$ be a specification of a function $f : s_1,\ldots,s_n \rightarrow s$, and take a term distributing tree $\mathcal{A}$ of the specification $\mathcal{E}$. If b is a branch in $\mathcal{A}$ from the root

θ, i.e. of the form $(x_1, \ldots, x_n)$, to a leaf θ', we then will use the following notation $(\theta_1, x'_1), \ldots, (\theta_{k-1}, x'_{k-1}), \theta_k$ to denote the branch b with $\theta_1 = \theta$, $\theta_k = \theta'$, where x'_i denotes the variable x' for the node θ_i in clause 2 of Definition 2.9 for every $i < k$.

Example 2.11. (example of term distributing trees) A distributing tree $\mathcal{A}$ of the Ackermann function is described in Figure 2.

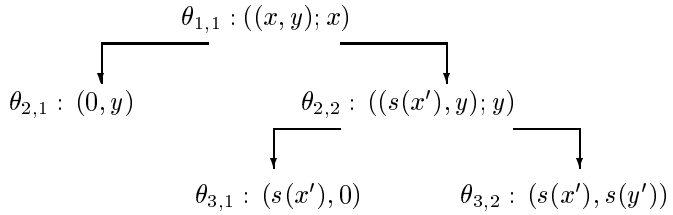


Figure 2: Term distributing tree of $\mathcal{E}_{Ack}$.

The definition of term distributing trees leads to the following:

Remark 2.12. (*ProPre* deals with completely defined functions) Let $f : s_1, \ldots, s_n \to s$ be a function with specification $\mathcal{E}$. Let $\mathcal{A}$ be a term distributing tree of $\mathcal{E}$. For each $(w_1, \ldots, w_n)$ of $\mathcal{G}(\mathcal{F}_c)_{s_1} * \ldots * \mathcal{G}(\mathcal{F}_c)_{s_n}$ there is one and only one leaf θ of $\mathcal{A}$ and a ground constructor substitution φ such that $\varphi(\theta) = (w_1, \ldots, w_n)$.

This means in particular that if one wants to associate a term distributing tree to a specification $\mathcal{E}$, then $\mathcal{E}$ must be completely defined.

Fact 2.13. The Ackermann function of Example 2.7 is completely defined and has a term distributing tree.

Fact 2.14. The *evenodd* function of Example 2.8 is not completely defined and hence there is no term distributing tree associated to its specification $\mathcal{E}_{eo}$.

In *ProPre*, the termination method deals with inductive proofs that require the completeness of the functions for the extraction of λ-terms. However this restriction can be removed in the new setting where Clause 3 of Definition 2.9 will be modified in the next section so that the termination method will become insensible to the fact that the function is completely or incompletely defined. Note that the algorithm devised in [29] was probably the first efficient algorithm to test whether a definition of a function is *ambiguous* and/or *complete*. However, the termination method here is insensible of the completeness of a definition of a function.

Remark-Notation 2.15. Let $\mathcal{A}$ be a term distributing tree for a specification and $b = (\theta_1, x_1), \ldots, (\theta_{l-1}, x_{l-1}), \theta_l$ be a branch from the root θ_1 of $\mathcal{A}$ to a leaf θ_l.

- Then for each node θ_i, θ_j with $1 \leq i \leq j \leq l$, there exists a constructor substitution $\sigma_{\theta_j, \theta_i}$ such that $\sigma_{\theta_j, \theta_i}(\theta_i) = \theta_j$. Furthermore we have the following equality $\sigma_{\theta_k, \theta_j} \circ \sigma_{\theta_j, \theta_i} = \sigma_{\theta_k, \theta_i}$ for all $i \leq j \leq k \leq l$.
- If a node θ_i in the branch b matches a term u of a recursive call (t, u), then the substitution will be denoted by $\rho_{\theta_i, u}$.

Notation 2.16. For a term distributing tree $\mathcal{A}$ of a specification and a left-hand side t of an equation of the specification, we will use the notation $b(t)$ to denote

the branch in $\mathcal{A}$ that leads to the leaf t' (which is a tuple) such that $t = f(t')$. Conversely if b is a branch of $\mathcal{A}$, L_b will denote the leaf of the branch b.

The rest of this section is devoted to the new *terminal state property* that characterises the formal proofs in *ProPre* abstracted in the present setting (part B of Figure 1). We do not recall *ProPre*'s notion of right terminal state property (part A) because it involves sophisticated second order formulas which are not necessary for our purpose. This notion however can be found in [20].

An effective measure m on the terms ranging over natural numbers, closed under substitutions (i.e. $m(u) > m(v)$ implies $m(\sigma(u)) > m(\sigma(v))$) is used in *ProPre*. This is made explicit by the following *size measure* $|\ .\ |_{\#}$ that consists, for each term t, in counting the number of its subterms including t itself:

$$|t|_{\#} = \begin{cases} 1 & \text{if } t \in \mathcal{X}, \\ 1 + |t_1|_{\#} + \ldots + |t_n|_{\#} & \text{if } t = g(t_1, \ldots, t_n),\ g \in \mathcal{F}_c. \end{cases}$$

Such a measure or variants are often used as for instance in Nqthm [3] with the function *count* or in [30] although the ordering is a fixed one. In contrast with mentioned works where the ordering is a fixed one, it turns out that it is not the case in *ProPre* as other more complex orderings, called *ramified measures* in [22], can be extracted from the formal proofs obtained in the system.

An *auxiliary* ordering $\sqsubseteq$ on terms is introduced to deal with the above measure.

Definition 2.17. (the ordering relation $\sqsubset$) Let $u, v \in \mathcal{T}(\mathcal{F}_c, \mathcal{X})_s$ for a given sort s. We say that $u \sqsubset v$ iff: (a) $|u|_{\#} < |v|_{\#}$, and (b) $\mathcal{V}ar(u) \subseteq \mathcal{V}ar(v)$, and (c) u is linear.

The relation $\sqsubset$ is only used as a part of the terminal state property but does not correspond to the general ordering ensuring the termination of the functions obtained in *ProPre* in part B. As mentioned, this one has to be obtained in relation to the full statement of the abstracted terminal state property defined below. We recall that we will use Remark-Notation 2.15 and 2.16: $\sigma_{\theta_i,\theta_j}, \rho_{\theta_i,u}, \ldots$.

Definition 2.18. (abstracted terminal state property) Let $\mathcal{A}$ be a term distributing tree for a specification. We say that $\mathcal{A}$ has the *abstracted terminal state property* if there is an application $\mu : \mathcal{A} \to \{0, 1\}$ defined on the nodes of $\mathcal{A}$ such that $\mu(L) = 0$ for each leaf L, and for all recursive calls (t, u), there is a node (θ, x) in the branch $b(t)$ with $\mu(\theta) = 1$ such that θ matches u with $\rho_{\theta,u}(x) \sqsubset \sigma_{L_{b(t)},\theta}(x)$ and for all ancestors (θ', x') of θ in $b(t)$ with $\mu(\theta') = 1$, we have $\rho_{\theta',u}(x') \sqsubseteq \sigma_{L_{b(t)},\theta'}(x')$.

Example 2.19. Take the term distributing tree of the Ackermann function of Example 2.11 given in Figure 2. Let $\mu(\theta_{1,1}) = \mu(\theta_{2,2}) = 1$. Using Definition 2.17, Definition 2.18 applies to the term distributing tree of the Ackermann function.

For simplicity, we will use the same terminology in both part of Figure 1, and drop the term *abstracted* as no confusion will be possible. In the same way that the right terminal state property in part A makes sense in the formal context, the above definition makes sense with ordinal measures. That is, it is possible to associate measures to distributing trees that satisfy the above property, decreasing in the recursive call of the functions. The connection between terminal state property and

the decreasing measures is postponed until Section 5 where it will be done in the context of the new measures and the hierarchical property defined further.

3. Recursive distributing trees

This section modifies the concepts of distributing tree while the main core of the termination method will be given in the next sections. We take advantage of the connection shown in Figure 1. As earlier mentioned, *ProPre* deals with complete definitions in the sense of [29] using formal proof trees based on inductive methods and characterised by the right terminal state property (part A of Figure 1). Instead of formal proofs we will still consider trees but in the abstract context where ordinal measures will be associated and for which the decreasing property in the recursive calls will ensure the termination of the algorithms. In the latter case the definitions of function do not need to be complete. This will allow us to introduce *recursive distributing trees*. The second refinement consists in enlarging the set of candidate trees for the same specification by introducing *split specifications* as follows:

Definition 3.1. (split specifications) Let $\mathcal{E}$ be a specification of a function $f : s_1, \ldots, s_n \to s$. A *split specification* $\mathcal{E}'$ of $\mathcal{E}$ is a specification of the same function $f : s_1, \ldots, s_n \to s$ such that for every equation (t', u') of $\mathcal{E}'$ there is an equation (t, u) of $\mathcal{E}$ and a substitution σ such that $(\sigma(t), \sigma(u)) = (t', u')$.

A term distributing can be associated indifferently to a specification or to one of its split specifications due to the following lemma:

Lemma 3.2. (relating specifications and split specifications) Let $\mathcal{E}$ be a specification of a function $f : s_1, \ldots, s_n \to s$ and let $\mathcal{E}'$ be a split specification of $\mathcal{E}$ which is complete. f is terminating for the specification $\mathcal{E}'$ iff f is terminating for the specification $\mathcal{E}$.

Proof: $\Leftarrow$) is clear. For $\Rightarrow$), we assume that f is non terminating for $\mathcal{E}$. So there is an infinite sequel $t_1 \to t_2 \to \ldots \to t_m \to \ldots$ coming from the rules with the specifications of defined functions and $\mathcal{E}$. Consider each i such that $t_i = C[\sigma(l)]$ and $t_{i+1} = C[\sigma(r)]$ where σ is a substitution, a context C and an equation $l \to r$ of $\mathcal{E}$ with l of the form $f(u_1, \ldots, u_n)$. But $\mathcal{E}'$ is complete, so there is an equation of $l' \to r'$ of $\mathcal{E}'$ and a substitution τ such that $\tau(l') = \sigma(l)$. According to the definition of the split specification, there is an equation (t'', u'') of $\mathcal{E}$ and a substitution ρ with $(\rho(l''), \rho(r'')) = (l', r')$. As $\mathcal{E}$ is non-overlapping we have $\tau \circ \rho = \sigma$ with $(l'', r'') = (l, r)$ and so $\sigma(r) = \tau(r')$. Therefore we can write $t_i = C[\tau(l')]$ and $t_{i+1} = C[\tau(r')]$, and we deduce that f is non terminating for $\mathcal{E}'$. □

Note that a specification is a split specification of itself. In the rest of the paper we may use the expression *specification* to denote both a specification or a split specification. Now, we introduce the *recursive distributing trees* that correspond to the term distributing trees but where a condition of the definition is weakened, and we give two examples after the definition.

Definition 3.3. (recursive distributing trees) Let $\mathcal{E}$ be a specification of a function $f : s_1, \ldots, s_n \to s$. A *recursive distributing tree* $\mathcal{A}$ for $\mathcal{E}$ is defined by:

1. The root of $\mathcal{A}$ is tuple of the form $(x_1, \ldots, x_n)$ where x_i is a variable of sort s_i for each $i \leq n$,

2. each node of $\mathcal{A}$ is of the form $(t_1, \ldots, t_n)$ and there is a variable x' of a sort s' in $f(t_1, \ldots, t_n)$ such that the set of children of the node, for variables $x'_1, \ldots, x'_r$ not in $t_1, \ldots t_n$, is $\{(t_1, \ldots t_n)[C(x'_1, \ldots x'_r)/x'],\ C : s'_1, \ldots, s'_r \rightarrow s' \in \mathcal{F}_c\}$,

3. There is an injective mapping $\mathcal{I}$ between the set of the left-hand sides of the equations and the set of leaves of $\mathcal{A}$, with $\mathcal{I}(f(t_1, \ldots, t_n)) = (t_1, \ldots, t_n)$, for each left-hand side $f(t_1, \ldots, t_n)$ of an equation (up to a renaming of variables).

Note that a term distributing tree is also a recursive distributing tree and that the same notations in the earlier section apply for recursive distributing trees. For a recursive distributing tree $\mathcal{A}$, we will note $\mathcal{A}'$ the associated tree for which the leaves of $\mathcal{A}$ that have no antecedent by the application $\mathcal{I}$ are removed.

Example 3.4. (a recursive distributing tree for a specification) A recursive distributing tree for the specification of the function *evenodd* is described in Figure 3 with an associated tree (where the $\otimes$ symbol denotes the removed leaves).

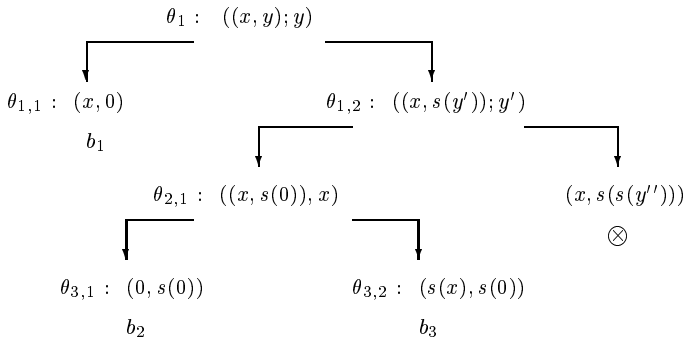


Figure 3: Recursive distributing tree of $\mathcal{E}_{eo}$ and the associated tree.

Example 3.5. (a recursive distributing tree for a (split) specification) In Figure 4, we give a recursive distributing tree for a split specification of the function *evenodd* with its associated tree.

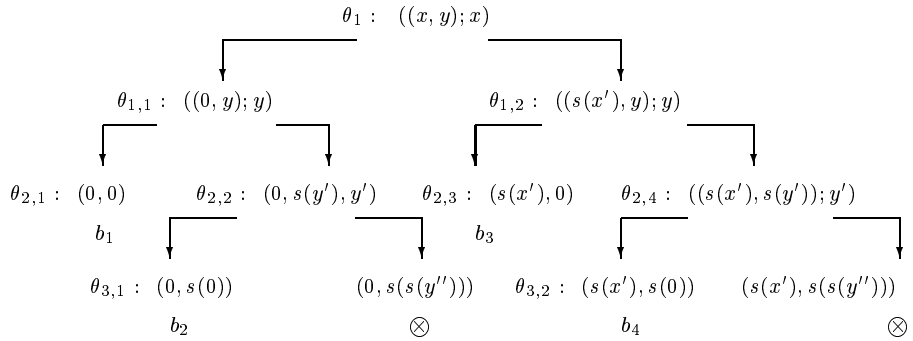


Figure 4: Recursive distributing tree of a split specification of $\mathcal{E}_{eo}$ and its associated tree.

Work has been done around the concept of *recursive distributing tree* with the concept of *completeness of definition* (e.g. [9, 29]) or *test sets* (e.g. [23, 13, 2]). However the main extension here relies on the establishment of the termination property and the associated ordinal functions that can also be defined on term distributing trees (part B of Figure 1). Therefore we will show that the termination method extends the method of the earlier section.

4. Extending the ProPre terminal state property

We want take advantage of the method of *ProPre* but given in the new setting (part B of Figure 1) instead of the former context (part A). We believe that it is easier to investigate ordinal measures as they are more *flexible*, at least in the present context, than the formal proofs which *carry* information that is not necessarily related to the termination part. We use the structure of the trees and introduce a termination property, called *hierarchical property*. This property allows us to show the decreasing property of measures in the recursive call of the functions and therefore the termination of these functions.

Let us consider an inductive data type D defined in ProPre. This gives rise to natural structural orderings according to its buidling (see [25, 20]). Let us denote $\prec$ such an ordering. Based on an usual induction schema, it allows a property $P(x)$ to be proven for every x by first showing $P(c)$ for each constant c of type D and then proving $P(cf(u_1, \ldots, u_m))$ for each constructor cf assuming $P(v)$ with $v \prec cf(u_1, \ldots, u_m)$. This can for instance be applied when P expresses the termination property of the Ackermann function of Example 2.7.

Well-founded orderings are particularly suited for termination using inductive proofs and as it is claimed in [25] any other well-founded orderings are actually difficult to find automatically. As a simple illustration, the function equal to 0 whose dummy specification $\mathcal{E}_0$ can be described as follows:

$$f(0) \to f(s(0)); \quad f(s(0)) \to f(s(s(0)); \quad f(s(s(x)) \to 0.$$

Although a well-founded ordering can of course be easily found by a human in this case, it is difficult to obtain one in an automated way. In particular it appears that automatically proving the following correctness of the specification: $f(x) = 0$ with the termination statement is not so adequate as it is usually done with a structural ordering. In fact, *simplifying* the rules in this case by merely applying the rules to each ground term provides a solution. But this cannot always be easily done as the example of the *quot* function, coming from [15, 1], shows while the above discussion on the structural ordering remains the same. A specification $\mathcal{E}_{quot}$ of the $quot : nat, nat, nat \to nat$ function is: $quot(0, s(y), s(z)) \to 0$,
$quot(s(x), s(y), z) \to quot(x, y, z)$, and $quot(x, 0, s(z)) \to s(quot(x, s(z), s(z))$.
As mentioned in [1], the value of $quot(x, y, z)$ corresponds to $1 + \lfloor \frac{x-y}{z} \rfloor$ when $z \neq 0$ and $y \leq x$, that is to say $quot(x, y, y)$ computes $\lfloor \frac{x}{y} \rfloor$.

Because of the last equation in the above specification, we would like to have a well-founded ordering on $nat \times nat \times nat$ for which at least $(x, 0, s(z)) \prec (x, s(z), s(z))$ holds. But a lexicographic combinations of the usual ordering on natural numbers is not suitable there, nor are the ramified measures that come from the inductive

method of *ProPre* [12]. The *ProPre* system also fails for the *evenodd* function for similar reasons to those of the *quot* function.

Note that a direct application of the recursive path orderings [5] or polynomial interpretations [4, 7, 27], or the Knuth-Bendix orderings [14] fails as well. These have been designed to be applied in a large context (rewrite system or algebraic systems for the Knuth-Bendix algorithm). However, it turns out that a similar argument on the inductive methods concerning the earlier examples also holds for the simplification orderings. That is, they fail because of the two first rules of the specification $\mathcal{E}_0$, the first rule of $\mathcal{E}_{eo}$ and the third rule of $\mathcal{E}_{quot}$.

The usual inductive methods do not deal with the termination of such functions and of the *evenodd* function. The specifications of these functions and of the *evenodd* functions are not complete. But even adding *dummy* equations that preserve the termination property of the functions, the new specifications cannot still be treated by usual inductive methods (such as those used by *ProPre*) because of the mentioned rules in the specification. In particular it can be easily checked that the recursive distributing trees of Figure 3 and Figure 4 which correspond to a refinement of term distributing trees do not have the terminal state property.

The approach we follow in the next section is to first introduce a notion of a recursive distributing tree which, like in inductive methods, takes advantage of the structure of the left-hand side of the equations. But in contrast to usual inductive termination methods, the construction of recursive distributing trees allows us to state a new terminal state property in the search of termination that also takes account of orderings that may now be different from usual structural orderings.

We introduce here some definitions that will allow us to state a *new terminal state property*. This new terminal state will be crucial in the study of termination of recursive functions whose proofs of termination require non structural orderings. We show in Theorem 1 that the new terminal state property strictly extends the terminal state property which corresponds to that of the *ProPre* system.

4.1. The hierarchical property

We first need to introduce fresh variables as follows. For each position q and sort s, we will assume a new variable of sort s indexed by q and distinct from those of $\mathcal{X}$. This allows us to introduce the following definition.

Definition 4.1. Let t be a term and q be a position. The term $[[t]]_q$ is defined as follows: $[[x]]_q = x$ if x is a variable, $[[C(t_1, \ldots, t_n)]]_q = C([[t_1]]_{q\cdot 1}, \ldots, [[t_n]]_{q\cdot n})$ if $C \in F_c$, and $[[g(t_1, \ldots, t_n)]]_q = x_q$ if $g \in F_d$.
For a term $u = g(u_1, \ldots, u_n)$ and a substitution φ, $g(\varphi[[u]])$ will denote the term $g(\varphi([[u]]_1), \ldots, \varphi([[u]]_n))$.

Along with the ordering $\sqsubset$ defined in Section 2, we introduce the following relations: for u, v in $\mathcal{T}(\mathcal{F}_c, \mathcal{X})_s$, we say that $u \trianglerighteq v$ if $u \not\sqsubset v$ with $\neg$(b) or $\neg$(c) or $|v|_\# < |u|_\#$ in Definition 2.17; and we say that $u \preccurlyeq v$ if $u \not\sqsubset v$ with (b) and (c) and $|u|_\# = |v|_\#$.

In the following definitions, we consider a function $f : s_1, \ldots, s_n \to s$, a (split)

specification $\mathcal{E}$, and the associated tree $\mathcal{A}'$ of a recursive distributing tree $\mathcal{A}$ of $\mathcal{E}$.

Definition 4.2. For each node θ, C_θ will denote $\{b \in \mathcal{A}', \theta \in b\}$ and $\mathcal{R}_\theta$ the set of recursive calls (t,u) such that $b(t) \in C_\theta$. If (t,u) is a recursive call, then $\mathcal{M}_{\mathcal{A}'}(u) = \{b \in \mathcal{A}', \exists \varphi, \varphi' \text{ such that } f(\varphi[[u]]) = \varphi'(f(L_b))\}$ and $\mathcal{Q}_{\mathcal{A}}(t,u) = \{\theta \in b(t), \exists \sigma, \sigma(f(\theta)) = u\}$.

Note that the set $\mathcal{Q}_{\mathcal{A}}(t,u)$ is not empty since the root node belongs to $\mathcal{Q}_{\mathcal{A}}(t,u)$. Let b be a branch and two nodes $\theta, \theta' \in b$, we say that $\theta < \theta'$ if θ is *closer* than θ' to the root (i.e. if θ is an ancestor of θ'). So we can write $\mathcal{N}_{\mathcal{A}}(t,u) = \max \mathcal{Q}_{\mathcal{A}}(t,u)$.

For each node θ of $\mathcal{A}'$ we assume an associated subset $\mathcal{G}_\theta$ of $\mathcal{R}_\theta$ which will be made explicit in Definition 4.5. The meaning of the two following definitions is to give decreasing criteria that extends those of Definitions 2.17 and 2.18 and relies in particular on the *hierarchical structure* of the trees. Notice that the definitions below should be given simultaneously (Definitions 4.3, 4.4, 4.5, 4.6), but these, defined on the height of the tree $\mathcal{A}$, are introduced separately to ease the readability.

Definition 4.3. Let (θ, x) be a node of $\mathcal{A}'$ and $\mathcal{G}_\theta$ be a subset of $\mathcal{R}_\theta$. For each recursive call (t,u) of $\mathcal{G}_\theta$ such that $\theta \in \mathcal{Q}_{\mathcal{A}}(t,u)$, we assume that one of the two following cases below holds and we define $\xi^\theta_{(t,u)}$, as follows:

1. If $\rho_{\theta,u}(x) \sqsubset \sigma_{L_{b(t)},\theta}(x)$ or $\rho_{\theta,u}(x) \trianglerighteq \sigma_{L_{b(t)},\theta}(x)$, then $\xi^\theta_{(t,u)} = 1$,
2. If $\rho_{\theta,u}(x) \preccurlyeq \sigma_{L_{b(t)},\theta}(x)$, then $\xi^\theta_{(t,u)} = 0$.

The above definition is intended to deal with orderings that may be different from the usual structural orderings. This leads to also introduce the following:

Definition 4.4. Let (θ, x) be a node of $\mathcal{A}'$ and $\mathcal{G}_\theta$ be a subset of $\mathcal{R}_\theta$. For each recursive call (t,u) of $\mathcal{G}_\theta$ such that $\theta \in \mathcal{Q}_{\mathcal{A}}(t,u)$ and for each branch $b \in \mathcal{C}_\theta$, we will define $\eta^\theta_{(t,u),b}$ in the following way:

1. We first consider all (t,u) such that $\rho_{\theta,u}(x) \trianglerighteq \sigma_{L_{b(t)},\theta}(x)$ and take for $b \in \mathcal{C}_\theta$:
$$\eta^\theta_{(t,u),b} = \begin{cases} 0 & \text{if } b \in \mathcal{M}_{\mathcal{A}'}(u), \\ 1 & \text{otherwise} \end{cases}$$
2. Next, we consider each (t,u) in $\mathcal{G}_\theta$ such that there is a (t',u') with $\eta^\theta_{(t',u'),b(t)} = 0$, and for which no $\eta^\theta_{(t,u),b'}$ is defined for any $b' \in \mathcal{C}_\theta$. We then take
$$\eta^\theta_{(t,u),b} = \begin{cases} 0 & \text{if } b \in \mathcal{M}_{\mathcal{A}'}(u), \\ 1 & \text{otherwise} \end{cases}$$
3. Finally if item 2 cannot be applied, we put $\eta^\theta_{(t,u),b} = 1$ for each $b \in \mathcal{C}_\theta$.

Note that cases 1 and 2 in Definition 4.4 are distinct because $\xi^\theta_{(t,u)}$ is algorithmically defined; namely case 1 is the initial case and case 2 is a (finite) *loop case*.

Now we can define, for each node θ of $\mathcal{A}'$ and each left-hand side t of an equation such that θ with $b(t) \in C_\theta$, the value:
$$\eta^\theta_t = \prod_{\substack{(t',u') \in \mathcal{G}_\theta \\ \theta \in \mathcal{Q}_{\mathcal{A}}(t',u')}} \eta^\theta_{(t',u'),b(t)} \text{ if } \mathcal{G}_\theta \neq \emptyset \text{ and } 0 \text{ otherwise.}$$

We now explicit the subset $\mathcal{G}_\theta$ of $\mathcal{R}_\theta$ for each node θ. The following definition states whether from each node, a recursive call can be eliminated from a set of recursive calls according to some conditions.

Definition 4.5. Let θ_1 be the root of the recursive distributing tree $\mathcal{A}$. We first put $\mathcal{G}_{\theta_1} = \mathcal{R}_{\theta_1}$. Now assume that $\mathcal{G}_\theta$ is defined for a node θ of $\mathcal{A}'$ and let θ' be a child of θ with θ' in $\mathcal{A}'$. The set $\mathcal{G}_{\theta'}$ is then defined as follows: $(t,u) \in \mathcal{G}_{\theta'}$ iff $(t,u) \in \mathcal{R}_{\theta'} \cap \mathcal{G}_\theta$ and $(\xi^\theta_{(t,u)}, \eta^\theta_t) \neq (1,1)$.
We also define an application F on each node θ distinct from a leaf. The application F can be seen as a necessary condition for the termination statement. That is, roughly, if a recursive call (t,u) has to still be considered for a node θ while $\theta > \mathcal{N}_\mathcal{A}(t,u)$, then the recursive distributing tree will not have the hierarchical property. Note that an analogous condition also holds for the terminal state property.

Definition 4.6. (F: a necessary condition for termination) Let θ be a node of associated tree $\mathcal{A}'$ of the recursive distributing tree $\mathcal{A}$ which is distinct from a leaf. We put $F(\theta) = 0$ if there is a child θ' of θ and (t,u) in $\mathcal{G}_{\theta'}$ such that $\theta > \mathcal{N}_\mathcal{A}(t,u)$; and we put $F(\theta) = 1$ otherwise.

Now the hierarchical property can be defined below.

Definition 4.7. (hierarchical property) The recursive distributing tree $\mathcal{A}$ is said to have the new terminal state property if for each node θ of $\mathcal{A}'$ distinct from a leaf we have $F(\theta) = 1$ and for each branch b there is node θ' in b such that $\mathcal{G}_{\theta'} = \emptyset$.

Let us now present again the two examples of recursive distributing tree of the specification of $\mathcal{E}_{eo}$ and a split specification of $\mathcal{E}_{eo}$ introduced respectively in Figure 3 and Figure 4. The first one does not have the new terminal state property (Example 4.8), while the second does (Example 4.9).

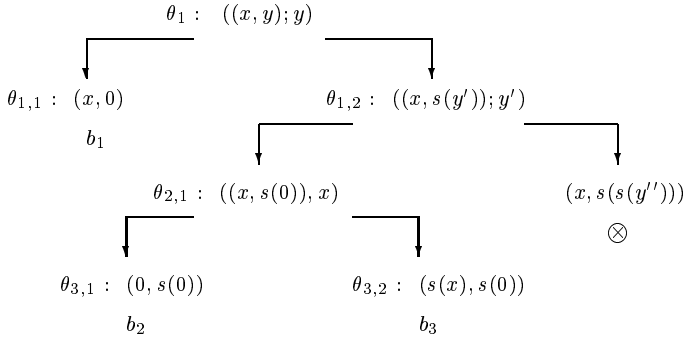


Figure 5: The associated tree of a recursive distributing tree of $\mathcal{E}_{eo}$.

Example 4.8. (recursive distributing tree without the new terminal state property) We show that the recursive distributing tree $\mathcal{A}$ given in Figure 5 of the specification of the *evenodd* function does not have the new terminal state property. Let $t_1 = (x,0)$, $t_2 = (0,s(0))$, $t_3 = (s(x),s(0))$. The set $\mathcal{G}_{\theta_1}$ is $\mathcal{R}_{\theta_1} = \{r_1, r_2\}$ with $r_1 = ((x,0),(x,s(0)))$, $r_2 = ((s(x),s(0)),(x,0))$. We have $\xi^{\theta_1}_{r_1} = 1$, $\xi^{\theta_1}_{r_2} = 1$, $\eta^{\theta_1}_{r_1,b_1} = 1$, $\eta^{\theta_1}_{r_1,b_2} = 0$ and $\eta^{\theta_1}_{r_1,b_3} = 0$. Then we have $\eta^{\theta_1}_{r_2,b_1} = 0$, $\eta^{\theta_1}_{r_2,b_2} = 0$ and $\eta^{\theta_1}_{r_2,b_3} = 1$. Therefore $\eta^{\theta_1}_{t_1} = 0$, $\eta^{\theta_1}_{t_2} = 0$ and $\eta^{\theta_1}_{t_3} = 0$. Thus $\mathcal{G}_{\theta_{1,1}} = \{r_1\}$; that is enough to conclude that $\mathcal{A}$ does not satisfy the new terminal state property.

Example 4.9. (recursive distributing tree with the new terminal state property) Let us consider the recursive distributing tree $\mathcal{A}$ given in Figure 6 of

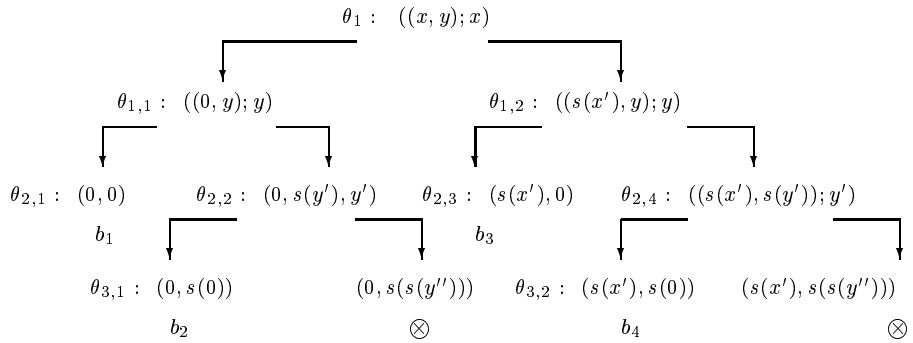


Figure 6: Associated tree of a recursive distributing tree of a split specification of $\mathcal{E}_{eo}$.

the (split) specification of the *evenodd* function. $\mathcal{A}$ satisfies the new terminal state property:
Let $t_1 = (0,0)$, $t_2 = (0,s(0))$, $t_3 = (s(x),0))$, $t_4 = (s(x),s(0))$. The set $\mathcal{G}_{\theta_1}$ is $\mathcal{R}_{\theta_1} = \{r_1, r_2, r_3\}$ with $r_1 = ((0,0),(0,s(0)))$, $r_2 = ((s(x),0),(s(x),s(0)))$, and $r_3 = ((s(x),s(0)),(x,0))$.
It is easy to see that $\xi^{\theta_1}_{r_1} = 0$, $\xi^{\theta_1}_{r_2} = 0$, $\xi^{\theta_1}_{r_3} = 1$, and $\eta^{\theta_1}_{r_1,b} = 1$, $\eta^{\theta_1}_{r_2,b} = 1$, $\eta^{\theta_1}_{r_3,b} = 1$ for each branch b. We have, therefore, $\eta^{\theta_1}_{t} = 1$ for each term t. We then obtain $\mathcal{G}_{\theta_{1,1}} = \{r_1\}$, $\mathcal{G}_{\theta_{1,2}} = \{r_2\}$ and $F(\theta_1) = 1$.
We now get $\xi^{\theta_{1,1}}_{r_1} = 1$, $\eta^{\theta_{1,1}}_{r_1,b_2} = 0$, $\eta^{\theta_1}_{r_1,b_1} = 1$ and therefore $\eta^{\theta_{1,1}}_{t_1} = 1$, $\eta^{\theta_{1,1}}_{t_2} = 0$. So $\mathcal{G}_{\theta_{2,1}} = \emptyset$, $\mathcal{G}_{\theta_{2,2}} = \mathcal{R}_{\theta_{2,2}} = \emptyset$, and $F(\theta_{1,1}) = 1$. Furthermore $\xi^{\theta_{1,2}}_{r_2} = 1$, and $\eta^{\theta_{1,2}}_{r_2,b_4} = 0$, $\eta^{\theta_{1,2}}_{r_2,b_3} = 1$. We then get $\eta^{\theta_{1,2}}_{t_3} = 1$, $\eta^{\theta_{1,2}}_{t_4} = 0$. Thus $\mathcal{G}_{\theta_{2,3}} = \emptyset$, and we also have $\mathcal{G}_{\theta_{2,4}} = \emptyset$; hence $F(\theta_{1,2}) = 1$. Now $\mathcal{G}_{\theta_{3,2}} = \mathcal{G}_{\theta_{2,4}} = \emptyset$ and $F(\theta_{2,4}) = 1$.
Therefore we can conclude that $\mathcal{A}$ satisfies the new terminal state property.

4.2. The theorem of generalisation

The new terminal state property can be viewed as a faithful extension of the terminal state property devised in the *ProPre* system in the following way:
Theorem 1 *Let $\mathcal{A}$ be a term distributing tree of a specification $\mathcal{E}$ of a function. If $\mathcal{A}$ has the terminal state property in the system ProPre, then $\mathcal{A}$ satisfies the new terminal state property. The opposite does not hold.*
Proof: See Appendix.

Notice that the extension does not only rely on the fact that a recursive distributing tree may not be a term distributing tree but on the new state property. For instance, the specification $\mathcal{E}_0$ and any split specification of $\mathcal{E}_0$ does not have any term distributing tree that has the terminal state property. This is also the case for the *evenodd* function and the *quot* function. That is to say, one can add to the specification of the function *evenodd*, an equation which is harmless for the termination of the function but which completes the domain of the definition of the function. For example, one can add the *dummy equation* $evenodd(x, s(s(y))) = true$. One can also use any split specification of the new completed specification, nevertheless

the *evenodd* function cannot be handled by the *ProPre* system which implies there is no term distributing tree for this function that has the terminal state property.

The terminal state property defined in the *ProPre* system ensures the termination of the concerned functions because a complete formal proof tree can be built from any distributing tree that enjoys the terminal state property (see [20]). In [22, 10] it was shown that ordinal functions can also be associated to distributing trees. These functions were proven to have the decreasing property in the recursive calls of the specifications if the trees satisfy the terminal state property too. As this also implies the termination of the concerned functions, this therefore can be seen, but with a different approach, as a new proof of the soundness of the mentioned property. In this paper, as we work on recursive term trees with a new terminal state property, it appears that formal proofs cannot be built, as it can be done with *ProPre*, in order to state the soundness of the new terminal state property. However we will establish the soundness by associating ordinal measures that will enjoy the decreasing property. In the next section, we explain why the measures associated to the formal proofs found by *ProPre* do not fit in our new context and why we will need to extend the definition of the measures of [22] to obtain new ordinal measures suited to the new terminal state property.

5. Soundness of the method with decreasing measures

In this section we explain how it is possible to define ordinal measures against trees of functions where if the ordinal measure decreases in the recursive call of the function, then this function terminates. We recall the ramified measures that come from the analysis of the *ProPre* system and we give new measures which will help in establishing terminations of functions where the proofs of terminations are non-inductive. Our main theorem of this section (Theorem 2) establishes that our new notion of right terminal state and our extended notion of measures, enable us to establish the termination of functions (inductive and non inductive).

We first recall the ordinal measures of [22, 10]. We need the following definition:
Definition 5.1. (height of a node in a tree) Let $\mathcal{A}$ be a tree and θ be a node of $\mathcal{A}$. The *height* of θ in $\mathcal{A}$, denoted by $\mathcal{H}(\theta, \mathcal{A})$, is the height of the subtree of $\mathcal{A}$, whose root is θ, minus one.
Let $\mathcal{A}$ be a term distributing tree of a specification. We assume that for each node θ_i distinct from a leaf, there is an associated application m_i to θ_i from the set of ground terms to the set of natural numbers.
Definition 5.2. (ramified measures) Let $\mathcal{E}$ be a specification of a function $f : s_1, \ldots, s_n \to s$ and $\mathcal{A}$ a term distributing tree of $\mathcal{E}$. We define the *ramified measure* $\Omega_{\mathcal{A}} : \mathcal{G}(\mathcal{F}_c)_{s_1} * \ldots * \mathcal{G}(\mathcal{F}_c)_{s_n} \to \omega^\omega$, where ω is the least infinite ordinal, as follows: Let v be an element of the domain $\mathcal{G}(\mathcal{F}_c)_{s_1} * \ldots * \mathcal{G}(\mathcal{F}_c)_{s_n}$ and θ be the leaf of $\mathcal{A}$ such that there is a substitution φ with $\varphi(\theta) = v$ (cf. Remark 2.12). Let b be the branch $(\theta_1, x_1), \ldots, (\theta_{k-1}, x_{k-1}), \theta$ of $\mathcal{A}$ from the root θ_1 to θ, let $\sigma_{\theta_j, \theta_i}$ be the substitutions related to b (cf. Remark-Notation 2.15) and for each θ_l the associated

application m_l, $l \leq k-1$. Then

$$\Omega_{\mathcal{A}}(v) = \sum_{i=1}^{k-1} \omega^{\mathcal{H}(\theta_i,\mathcal{A})} * m_i(\varphi(\sigma_{\theta_k,\theta_i}(x_i))) \ .$$

The ramified measures can be illustrated with Figure 7.

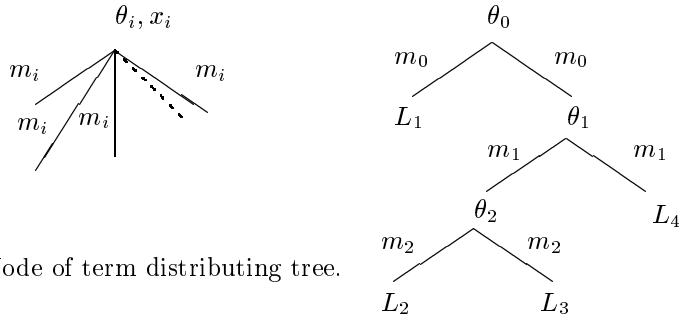


Figure 7: Term distributing tree and ramified measure.

Amongst the class of the ramified measures defined above, two subclasses of measures were related to the formal proofs obtained in *ProPre*. A first class of measures, called *R-measures*, was related to formal proofs coming from an earlier version of *ProPre* (see [22]) by substituting a mapping lg for the m_i measures in $\Omega_{\mathcal{A}}$ (see Figure 8). The function lg, sometimes called *size measure* like $|\ .\ |_{\#}$, is defined by:

$$lg(t) = \begin{cases} 1 & \text{if } t \in \mathcal{X}, \\ 1 + \sum_{s_j = s} lg(t_j) & \text{if } t = g(t_1, \ldots, t_n),\ g : s_1, \ldots, s_n \to s \in \mathcal{F}. \end{cases}$$

It was shown that any formal termination proof found by *ProPre* implies the corresponding ordinal measures to have the decreasing property in the recursive calls of the specifications (see [22]).

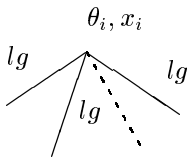


Figure 8: Node of term distributing tree and ramified measure.

Example 5.3. (ramified measure of the term distributing tree of *Ack*) The ramified measure of the term distributing tree of the Ackermann *Ack* function defined in Section 2 is: $\Omega_{\mathcal{A}}(0, y) = \omega$, $\Omega_{\mathcal{A}}(s(x), 0) = \omega * (1 + lg(x)) + 1$ and $\Omega_{\mathcal{A}}(s(x), s(y)) = \omega * (1 + lg(x)) + (1 + lg(y))$.

Another class of measures could also be defined from new formal proofs using an extended version of *ProPre* (see [10]) in which new inductive rules were introduced.

These measures, called *I-measures*, can be defined using both the skeleton form of distributing trees (i.e., term distributing trees) and the terminal state property given in Section 2.2. The m_i functions are obtained with $m_i = \mu(\theta_i) * |\ .\ |_{\#}$ in the definition of $\Omega_{\mathcal{A}}$ illustrated with Figure 9.

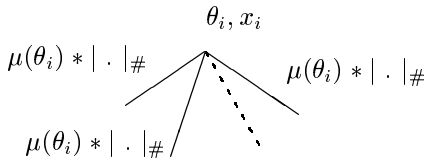


Figure 9: Node of term distributing tree and ramified measure.

[20] showed that each function proven to terminate in the first version of *ProPre* can also be proven terminating in the new version. Note that, unlike the case of R-measures, the m_i functions depend on the nodes θ_i in the definition of I-measures.

A natural question is to know whether there is also a suitable subclass of the ramified measures that can be related to the recursive distributing trees that have the new terminal state property. This is important as it would enable the termination of functions to be established in our context.

As already mentioned there is no term distributing tree for $\mathcal{E}_0$ that has the terminal state property. This does not a priori imply that an R-measure or an I-measure associated to a term distributing tree does not have the decreasing property. However, it can be easily checked that the decreasing property actually does not hold for these measures. This is also the case for instance with the *evenodd* function or the *quot* function (even with completed specifications). However the ordinal function: $\Omega_1(u,0) = \omega * |u|_{\#} + 1$ $\Omega_1(u, s(v)) = \omega * |u|_{\#}$ satisfies the decreasing property in the recursive calls of the specification $\mathcal{E}_0$. Also, the following ordinal function satisfies the decreasing property for the specification of the *quot* function: $\Omega_2(u, s(v), w) = \omega * |u|_{\#}$ $\Omega_2(u, 0, w) = \omega * |u|_{\#} + 1$.

It would be possible to find, amongst the class of the ramified measures, decreasing measures for the mentioned functions. But the choice of the m_i functions that occur in $\Omega_{\mathcal{A}}$ is difficult to obtain in an automated way dealing with the termination of such functions. In particular we would like to have m_i functions as simple as possible, as is the case for those obtained for the R-measures or the I-measures. We will therefore need to enlarge the definition of the ramified measures in order to obtain a new class which will be related to the recursive distributing tree. We will then show that the new terminal state property implies the decreasing of the associated measures. We first need to introduce the following definition:

Definition 5.4. (node measures) Let $\mathcal{A}$ be a recursive distributing tree of a specification of a function. For each node θ_i of $\mathcal{A}$, and each subranch starting from the θ_i, we will assume that there is an associated application m_i, called *node measure*, from the set of ground terms to the set of natural numbers. The application m_i will be also noted $m_{\theta_i,\theta}$ where θ is the leaf of the concerned subranch.

Example 5.5. (an illustration of node measures) The node measures can be illustrated with Figure 10.

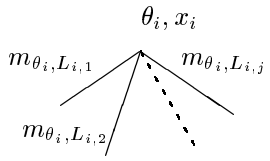


Figure 10: Nodes measure.

For a function $f : s_1, \ldots, s_n \to s$ with a specification $\mathcal{E}$, we will note $T^f_{s_1,\ldots,s_n}$ the set of the elements $(v_1, \ldots, v_n)$ in $\mathcal{G}(\mathcal{F}_c)_{s_1} \times \ldots \times \mathcal{G}(\mathcal{F}_c)_{s_n}$ such that there is substitution σ and a left-hand side $(t_1, \ldots, t_n)$ of an equation of $\mathcal{E}$ with $\sigma(t_i) = v_i$ for every $1 \leq i \leq n$.

We now define the extended ramified measures as follows:

Definition 5.6. (extended ramified measures) Let $\mathcal{E}$ be a specification of a function $f : s_1, \ldots, s_n \to s$ and $\mathcal{A}'$ be the associated tree of a recursive distributing tree $\mathcal{A}$ of $\mathcal{E}$. The *extended ramified measure* $\Omega_{\mathcal{A}} : \mathcal{T}^f_{s_1,\ldots,s_n} \to \omega^\omega$ is defined by:
Let v be an element of the domain $\mathcal{T}^f_{s_1,\ldots,s_n}$ and θ be the leaf of $\mathcal{A}'$ such that there is a substitution φ with $\varphi(\theta) = v$ (cf. Remark 2.12). Let b be the branch $(\theta_1, x_1), \ldots, (\theta_{k-1}, x_{k-1}), \theta$ of $\mathcal{A}'$ from the root θ_1 to θ, let $\sigma_{\theta_j,\theta_i}$ be the substitutions related to b (cf. Remark-Notation 2.15) and let $m_{\theta_i,\theta}$ be the associated node measure for each θ_l. Then

$$\Omega_{\mathcal{A}}(v) = \sum_{i=1}^{k-1} \omega^{\mathcal{H}(\theta_i,\mathcal{A}')} * m_{\theta_i,\theta}(\varphi(\sigma_{\theta_k,\theta_i}(x_i))) \ .$$

Example 5.7. (an illustration of an extended ramified measure) Considering the earlier term distributing tree (which is also a recursive tree) of Figure 7, an extended ramified measures can be illustrated with Figure 11.

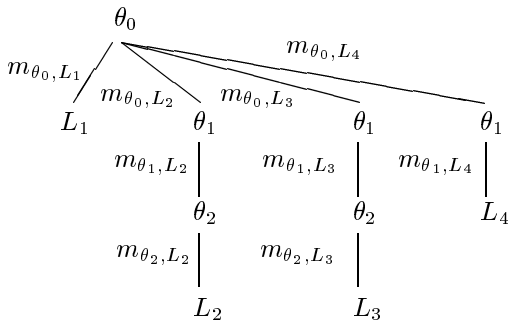


Figure 11: Extended ramified measure.

Amongst the class of ramified measures, the R- and I-measures turned out to be suitable for the termination of a class of functions as they could be associated to formal proofs obtained in *ProPre*. We show here that there also exists a subclass of the extended ramified measures that fit with the recursive term distributing trees that satisfy the new terminal state property. These are defined as follows:

Definition 5.8. (hole measures) Let $\mathcal{E}$ be a specification of a function $f : s_1, \ldots, s_n \to s$ and let $\mathcal{A}'$ be the associated tree of a recursive distributing tree $\mathcal{A}$ of $\mathcal{E}$. The *hole measure* $\Omega_{\mathcal{A}} : \mathcal{T}^f_{s_1,\ldots,s_n} \to \omega^\omega$ is defined as follows:
Let v be an element of the domain $\mathcal{T}^f_{s_1,\ldots,s_n}$ and θ be the leaf of $\mathcal{A}'$ such that there is a substitution φ with $\varphi(\theta) = v$. Let b be the branch $(\theta_1, x_1), \ldots, (\theta_{k-1}, x_{k-1}), \theta$ of $\mathcal{A}'$ from the root θ_1 to θ. Then

$$\Omega_{\mathcal{A}}(t) = \sum_{i=1}^{k-1} \omega^{\mathcal{H}(\theta_i,\mathcal{A}')} * (\eta_t^{\theta_i} * |(\rho(\sigma_{\theta_k,\theta_i}(x_i)))|_\#) \ .$$

That is to say $m_{\theta_i,\theta} = \eta_t^{\theta_i} * |\ .\ |_\#$.
Note that, due to the relation between the leaf θ and the term t, $\eta_t^{\theta_i} * |\ .\ |_\#$ depends both on θ_i and θ in the above definition. This can be illustrated with Figure 12.

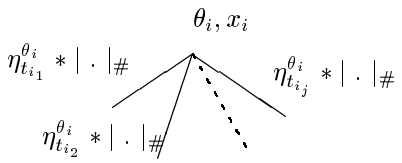


Figure 12: Node measure of hole measures.

Example 5.9. (associated ordinal measure of a recursive distributing tree) By Definition 5.8 and the values obtained in Example 4.9, the associated ordinal measure of the recursive distributing tree in Figure 3 of the *evenodd* function is:

$$\Omega_{\mathcal{A}}(x, 0) = \omega^2 * |x|_\# + \omega, \qquad \Omega_{\mathcal{A}}(x, s(0)) = \omega^2 * |x|_\#.$$

Note that the above ordinal Ω_1 can also be derived from the shape of $\Omega_{\mathcal{A}}$ on $T^{eo}_{na,nat}$.
It is clear that $\Omega_{\mathcal{A}}$ has the decreasing property in each recursive call on the domain of the function *evenodd*. This result can be generalised with Theorem 2 below.
The following theorem allows us to state the soundness of the new terminal state property. That is, if there a recursive distributing tree for a specification of a function that enjoys the new terminal state property, then the function is terminating.

Theorem 2 (soundness of the new terminal state property) *Let $\mathcal{E}$ be a specification of a function $f : s_1, \ldots, s_n \to s$ and $\mathcal{A}$ be a recursive distributing tree for $\mathcal{E}$ that has the new terminal state property. Then the extended ramified measure $\Omega_{\mathcal{A}}$ satisfies the decreasing property. That is to say, for each recursive call $(f(t_1, \ldots, t_n), f(u_1, \ldots, u_n))$ of $\mathcal{E}$ and every ground constructor substitution φ, such that $\varphi([[u_1]]_1), \ldots, \varphi([[u_n]]_n)$ is in $T^f_{s_1,\ldots,s_n}$, we have:*

$$\Omega_{\mathcal{A}}(\varphi(t_1), \ldots, \varphi(t_n)) > \Omega_{\mathcal{A}}(\varphi([[u_1]]_1), \ldots, \varphi([[u_n]]_n)).$$

Proof: Take a ground constructor substitution φ with $\varphi([[u_1]]_1), \ldots, \varphi([[u_n]]_n) \in T^f_{s_1,\ldots,s_n}$. We show that $\Omega_{\mathcal{A}}(\varphi(t_1), \ldots, \varphi(t_n)) > \Omega_{\mathcal{A}}(\varphi([[u_1]]_1), \ldots, \varphi([[u_n]]_n))$. Let t and u respectively denote the terms $f(t_1, \ldots, t_n)$ and $f(u_1, \ldots, u_n)$. By definition of the associated tree $\mathcal{A}'$ of $\mathcal{A}$, there is a branch b of $\mathcal{A}'$ of the form $(\theta_1, x_1), \ldots, (\theta_{k-1}, x_{k-1}), \theta_k$ where θ_1 is the root of $\mathcal{A}'$ and the leaf θ_k corresponds to t, i.e. $f(\theta_k) = t$ and $b(t) = b$. As $\varphi([[u_1]]_1), \ldots, \varphi([[u_n]]_n) \in T^f_{s_1,\ldots,s_n}$, there is an equation $(v, v') \in \mathcal{E}$ and a substitution τ such that $\tau(v) = f(\varphi[[u]])$ where $[[u]]$

denotes the uplet $(([[u_1]]_1), \ldots, ([[u_n]]_n))$.
Let $b(v)$ be the branch $(\theta'_1, x'_1), \ldots, (\theta'_{r-1}, x'_{r-1}), \theta'_r$ of $\mathcal{A}'$ with $f(\theta'_r) = v$.
As $\mathcal{A}$ has the new terminal state property, there is a node θ in b such that $\mathcal{G}_\theta = \emptyset$. But $\mathcal{G}_{\theta_1} = \mathcal{R}_{\theta_1}$ contains all the recursive calls of $\mathcal{E}$, and so $(t,u) \in \mathcal{G}_{\theta_1}$. Hence, there is a value J, such that $(t,u) \in \mathcal{G}_{\theta_J}$ but $(t,u) \notin \mathcal{G}_{\theta_{J+1}}$.
According to Definition 4.5, we have $(t,u) \in \mathcal{G}_{\theta_J} \subset \mathcal{G}_{\theta_{J-1}} \subset \ldots \subset \mathcal{G}_{\theta_1}$ and for all $1 \leq i < J$, $(\xi^{\theta_i}_{(t,u)}, \eta^{\theta_i}_t) \neq (1,1)$. Furthermore, because $\mathcal{A}$ has the new terminal terminal state property, $F(\theta_J) = 1$. But $(t,u) \in \mathcal{G}_{\theta_J}$, thus $\theta_J < \mathcal{N}_\mathcal{A}(t,u)$ by Definition 4.6, that is to say $\theta_J \in \mathcal{Q}_\mathcal{A}(t,u)$. So $f(\theta_J)$ matches u and therefore matches $\varphi(u)$. Since $f(\theta_J)$ is linear and φ is a ground constructor substitution, we deduce that $f(\theta_J)$ also matches $f(\varphi[[u]])$. But $f(\theta'_r)$ matches $f(\varphi[[u]])$ as well with the substitution τ. As $f(\theta_J)$ and $f(\theta'_r)$ match a common term, by construction of the recursive distributing trees, this implies that θ_J and θ'_r are in the same branch; and so $\theta_J \leq \theta'_r$ because θ'_r is a leaf. Thus we have $x'_i = x_i$ and $\theta'_i = \theta_i$ for $1 \leq i \leq J$. This gives us the relation $\varphi \circ \rho_{\theta_i,u}(x_i) = \tau \circ \sigma_{\theta'_r,\theta'_i}(x_i)$ for $1 \leq i \leq J$ according to the substitutions $\tau \circ \sigma_{\theta'_r,\theta'_i}$ and φ, and to the fact $\rho_{\theta_i,u}(x_i)$ is a constructor term by Definition 4.3. So we can write:

$$\begin{array}{l}\Omega_\mathcal{A}(\varphi([[u_1]]_1), \ldots, \varphi([[u_n]]_n)) = \sum_{i=1}^{J-1} \omega^{\mathcal{H}(\theta_i,\mathcal{A}')} * m_{\theta_i,\theta'_r}(\varphi \circ \rho_{\theta_i,u}(x_i)) \\ +\omega^{\mathcal{H}(\theta_J,\mathcal{A}')} * m_{\theta_J,\theta'_r}(\varphi \circ \rho_{\theta_J,u}(x_J)) + \sum_{i=J+1}^{r-1} \omega^{\mathcal{H}(\theta_i,\mathcal{A}')} * m_{\theta'_i,\theta'_r}(\tau \circ \sigma_{\theta'_r,\theta'_i}(x'_i))\end{array} \quad (1)$$

We are going now to show that

$$m_{\theta_i,\theta'_r}(\varphi \circ \rho_{\theta_i,u}(x_i)) \leq m_{\theta_i,\theta_k}(\varphi \circ \sigma_{\theta_k,\theta_i}(x_i)),\ i < J \text{ and} \quad (2)$$

$$m_{\theta_J,\theta'_r}(\varphi \circ \rho_{\theta_J,u}(x_J)) < m_{\theta_J,\theta_k}(\varphi \circ \sigma_{\theta_k,\theta_J}(x_J)) \quad (3)$$

- Let $1 \leq i < J$. We know that $\theta_i \in \mathcal{Q}_\mathcal{A}(t,u)$, and $(\xi^{\theta_i}_{(t,u)}, \eta^{\theta_i}_t) \neq (1,1)$.

- Let us consider the case $\eta^{\theta_i}_t = 0$. By definition of $\eta^{\theta_i}_{\_,\_}$, this implies there is a recursive call (t',u') such that $\eta^{\theta_i}_{(t',u'),b(t)} = 0$. But $\tau(v) = f(\varphi[\![u]\!])$ and $b(v)$ is in $\mathcal{M}_{\mathcal{A}'}(u)$. Hence, due to $\eta^{\theta_i}_{(t',u'),b(t)} = 0$ and $b(v) \in \mathcal{M}_{\mathcal{A}'}(u)$, we get $\eta^{\theta_i}_{(t,u),b(v)} = 0$ with Definition 4.4, and thus $\eta^{\theta_i}_v = 0$. The later equality implies $m_{\theta_i,\theta'_r} = 0$ and Inequality (2) then holds.
- We now consider the case $\eta^{\theta_i}_t = 1$ with $\xi^{\theta_i}_{(t,u)} = 0$. This gives us $m_{\theta_i,\theta_k} = |\ .\ |_\#$ and $\rho_{\theta_i,u}(x_i) \preccurlyeq \sigma_{\theta_k,\theta_i}(x_i)$. We deduce, from the definition of $\preccurlyeq$, that $|\varphi \circ \rho_{\theta_i,u}(x_i))|_\# \leq |\varphi \circ \sigma_{\theta_k,\theta_i}(x_J)|_\#$. Hence, Inequality (2) holds again whatever the value $\eta^{\theta_i}_v$ in m_{θ_i,θ'_r}.

• We now show Inequality (3). By definition of J, we have $\xi^{\theta_J}_{(t,u)} = 1$ and $\eta^{\theta_J}_t = 1$, and Inequality (3) boils down to $m_{\theta_J,\theta'_r}(\varphi \circ \rho_{\theta_J,u}(x_J)) < |\varphi \circ \sigma_{\theta_k,\theta_J}(x_J)|_\#$. As $\xi^{\theta_J}_{(t,u)} = 1$, two cases are then possible:
- *i)* $\rho_{\theta_J,u}(x_J) \sqsubset \sigma_{\theta_k,\theta_J}(x_J)$. But $\sqsubset$ is closed under substitutions, so (3) holds whatever the value $\eta^{\theta_J}_v$ in m_{θ_J,θ'_r}.
- *ii)* $\rho_{\theta_J,u}(x_J) \sqsupseteq \sigma_{\theta_k,\theta_J}(x_J)$. This implies $\eta^{\theta_J}_{(t,u),b(v)} = 0$ since $b(v) \in \mathcal{M}_{\mathcal{A}'}(u)$. Hence $\eta^{\theta_J}_v = 0$ and thus $m_{\theta_J,\theta'_r} = 0$. As $|w|_\# > 0$ for every w, we get Inequality (3).

We know that

$$\Omega_{\mathcal{A}}(\varphi(t_1),\ldots,\varphi(t_n)) = \sum_{i=1}^{J-1} \omega^{\mathcal{H}(\theta_i,\mathcal{A}')} * m_{\theta_i,\theta_k}(\varphi \circ \sigma_{\theta_k,\theta_i}(x_i))$$
$$+\omega^{\mathcal{H}(\theta_J,\mathcal{A}')} * m_{\theta_J,\theta_k}(\varphi \circ \sigma_{\theta_k,\theta_J}(x_J)) + \sum_{i=J+1}^{k-1} \omega^{\mathcal{H}(\theta_i,\mathcal{A}')} * m_{\theta_i,\theta_k}(\varphi \circ \sigma_{\theta_k,\theta_i}(x_i)) \quad (4)$$

Due to the expression of (1) and (4) and the inequalities (2) and (3), we can now conclude that $\Omega_{\mathcal{A}}(\varphi(t_1),\ldots,\varphi(t_n)) > \Omega_{\mathcal{A}}(\varphi([[u_1]]_1),\ldots,\varphi([[u_n]]_n))$. □

6. Conclusion

The system *ProPre* treats a class of term rewriting systems specifying a recursive function and deals with the automation of the proofs of termination of these recursive functions. Because the termination proofs of the *ProPre* system depend on structural orderings, e.g. by the size of terms, the system is difficult to prove specifications in which a recursive call in the right-hand side of a rule is not smaller than the left-hand side of the rule.

In this paper we proposed a method that extends the automation of the proofs of termination of recursive functions used in *ProPre*. Whereas *ProPre* could only deal with the automation of inductive proofs, our method allows the automation of a larger class of recursive functions because it can handle non structural orderings. The extension of the ProPre system proposed by this paper consists of mainly three parts.

1. If any term whose root symbol is a function defined by a specification can match the left-hand side of a rule, the specification is called complete. The original *ProPre* system can deal with complete specifications only. Indeed, the term distributing tree of a specification, which is defined for proving termination in the *ProPre* system, must satisfy the following condition: each leaf of the term distributing tree corresponds exactly one left-hand side of a rule of the specification. In this paper we defined the new distributing tree, called the *recursive distributing tree*, which has the weaker condition than the original. The recursive distributing tree can treat non-complete specifications.

2. For a specification that is hard to deal with directly, we propose the *split specification*. The split specification can be obtained by replacing a rule with rules that are instances of it. It is easier to prove the termination of a split specification than that of the source specification. We prove that a specification terminates if and only if the split one does.

3. Finally, we come to the main part of our extension: A termination proof in the *ProPre* system works as follows:

   (a) Define a term distributing tree of a given specification.

   (b) Verify that the tree has the terminal state property. If so, the specification is terminating. Roughly speaking, the terminal state property

means that each recursive call is structurally smaller than the left-hand side.

In this paper, we proposed the new terminal state property. Because we use the ordering by which a recursive call compared to the left-hand side does not depend the structure of terms, our method can apply to specifications in which simplification orderings fail, such as the recursive path ordering and so on.

Because constructing a recursive distributing tree and verifying whether the distributing tree has the terminal state property can be computed in a finite time, our paper contributes to the area of establishing *automatically* termination proofs of recursive functions. Indeed, we also prove that our method is stronger than the termination method of the original *ProPre* system. Moreover, we show that there are examples of specifications that cannot be proved terminating using *ProPre*, but can using our proposed extension. Our results may help in handling more realisitic examples that cannot be proved terminating (automatically) by structural methods.

Appendix

We give the proof of Theorem 1 stated again here below.

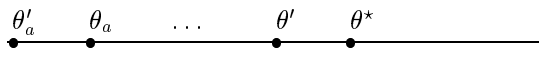
Theorem 3 *Let $\mathcal{A}$ be a term distributing tree of a specification $\mathcal{E}$ of a function. If $\mathcal{A}$ has the terminal state property in the system ProPre, then $\mathcal{A}$ satisfies the new terminal state property. The opposite does not hold.*

Proof: Let us consider a term distributing tree $\mathcal{A}$ of a specification. Recall that a term distributing tree $\mathcal{A}$ is a recursive distributing tree.

We are going to show that the application F of Definition 4.6 has the value 1 on each node distinct from a leaf. In order to get a contradiction we assume there is a node θ in $\mathcal{A}$, distinct from a leaf, such that $F(\theta_a) = 0$. So, as $F(\theta_a) = 0$, there is a child θ'_a of θ_a and a recursive call (t, u) in $\mathcal{G}_{\theta'_a} \subseteq \mathcal{R}_{\theta'_a}$ with $\theta_a > \mathcal{N}_{\mathcal{A}}(t, u)$. Because $\mathcal{A}$ has the terminal state property, there is a node $(\theta^\star, y)$ in the branch $b(t)$, with $\mu(\theta^\star) = 1$, that matches u with $\rho_{\theta^\star,u}(y) \sqsubset \sigma_{L_{b(t)},\theta^\star}(y)$ and for every ancestor $(\theta'^\star, y')$ of $\theta^\star$ in $b(t)$, such that $\mu(\theta'^\star) = 1$, we have $\rho_{\theta'^\star,u}(y') \sqsubseteq \sigma_{L_{b(t)},\theta'^\star}(y')$. So $\theta^\star$ is in $Q_{\mathcal{A}}(t, u)$ and we also have $\xi^{\theta^\star}_{(t,u)} = 1$. As $\theta^\star < \theta_a$ since $\theta^\star \leq \mathcal{N}_{\mathcal{A}}(t, u)$, let θ' be the child of $\theta^\star$ in the branch $b(t)$ where is also θ_a. We have in particular $\mathcal{G}_{\theta'_a} \subseteq \mathcal{G}_{\theta_a} \subseteq \mathcal{G}_{\theta'}$ and thus $(t, u) \in \mathcal{G}_{\theta'}$. Hence, according to Definition 4.5 and the fact that $\xi^{\theta^\star}_{(t,u)} = 1$, we deduce that $\eta^{\theta^\star}_t = 0$. This means that there is a recursive call $(t'', u'') \in \mathcal{G}_{\theta^\star}$ with $\eta^{\theta^\star}_{(t'',u''),b(t)} = 0$. Therefore, by construction of $\eta^{\theta^\star}_{-,-}$, this implies that there is also a recursive call $(t', u') \in \mathcal{G}_{\theta^\star}$, with $\rho_{\theta^\star,u'}(y) \trianglerighteq \sigma_{L_{b(t')},\theta^\star}(y)$, and a branch $b' \in \mathcal{C}_{\theta^\star}$ with $\eta^{\theta^\star}_{(t',u'),b'} = 0$.

As $\mathcal{A}$ has the terminal state property, there is a node $(\theta^\star_p, x_p)$, with $\mu(\theta^\star_p) = 1$, in the branch $b(t')$ such that $\rho_{\theta^\star_p,u'}(x_p) \sqsubset \sigma_{L_{b(t')},\theta^\star_p}(x_p)$ and for every ancestor $(\theta'^\star_p, x'_p)$ of $\theta^\star_p$ such that $\mu(\theta'^\star_p) = 1$, we have $\rho_{\theta'^\star_p,u'}(x'_p) \sqsubseteq \sigma_{L_{b(t')},\theta'^\star_p}(x'_p)$. Because (t', u') is in $\mathcal{G}_{\theta^\star} \subset \mathcal{R}_{\theta^\star}$, this implies that $\theta^\star$ is also in $b(t')$. Therefore we can compare $\theta^\star$ and $\theta^\star_p$ in the branch $b(t')$. The case $\theta^\star \leq \theta^\star_p$ is not possible otherwise, due

the above property of $\theta_p^\star$, we would have $\rho_{\theta^\star,u'}(y) \sqsubseteq \sigma_{L_{b(t')},\theta^\star}(y)$ that contradicts $\rho_{\theta^\star,u'}(y) \trianglerighteq \sigma_{L_{b(t')},\theta^\star}(y)$. So $\theta_p^\star < \theta^\star$, that in particular implies $\mathcal{G}_{\theta^\star} \subset \mathcal{G}_{\theta_p^\star}$.



$$(t,u) \in \mathcal{G}_{\theta'_a} \subset \mathcal{G}_{\theta_a} \subset \quad \ldots \quad \subset \mathcal{G}_{\theta'} \subset \mathcal{G}_{\theta^\star} \ni (t',u')$$

We can now repeat the same reasoning for $< \theta^\star, (t',u'), (\theta_p^\star, x_p) >$ as we did with $< \theta_a, (t,u), (\theta^\star, y) >$ because $(t',u') \in \mathcal{G}_{\theta_p^\star}$ and we also have $\xi^{\theta_p^\star}_{(t',u')} = 1$ since $\rho_{\theta_p^\star,u'}(x_p) \sqsubset \sigma_{L_{b(t')},\theta_p^\star}(x_p)$. This shows that we can construct an infinite sequence of nodes starting from θ_a with $\theta_a > \theta^\star > \theta_p^\star > \theta_q^\star > \theta_r^\star > \ldots$. But this is obviously impossible and we get our contradiction; hence $F(\theta_a) = 1$ and the application F has only the value 1 on each node distinct from a leaf.
Likewise, using a similar reasoning as in the first part of the proof, we cannot have $\mathcal{G}(\theta) \neq \emptyset$ for every node θ of some branch b in $\mathcal{A}$ without getting a contradiction. Hence, we deduce that $\mathcal{A}$ has the new terminal state property. Finally, the counterexample 4.9 shows that the opposite does not hold. □

References

1. T. Arts and J. Giesl. Automatically proving termination where simplification orderings fail. In *Proceedings of Theory and Practice of Software Development TAPSOFT'97*, volume 1214 of *Lecture Notes in Computer Science*, 1997. Springer-Verlag.
2. A. Bouhoula and M. Rusinowitch. Implicit induction in conditional theories. *Journal of Automated Reasoning*, 14(2):189-235, 1995.
3. R. S. Boyer and J S. Moore. *A computational logic handbook*. Academic Press, 1988.
4. A. Ben Cherifa and P. Lescanne. Termination of rewriting systems by polynomial interpretations and its implementation. *Science of Computer Programming*, 9(2):137–159, 1987.
5. Dershowitz. Orderings for term-rewriting systems. *Theoretical Computer Science*, 17(3):279–301, 1982.
6. J. Giesl. Automated termination proofs with measures functions. In *Proceedings of the 19th annual german conference on artificial intelligence*, volume 981 of *Lecture Notes in Artificial Intelligence*, pages 149–160, 1995. Springer-Verlag.
7. J. Giesl. Generating polynomial orderings for termination proofs. In *Proceedings of the 6th International Conference on Rewriting Techniques and Application*, volume 914 of *Lecture Notes in Computer Science*, 1995. Springer-Verlag.
8. J. Giesl. Termination analysis for functional programs using term orderings. In *Proceedings of the Second International Static Analysis Symposium*, volume 983 of *Lecture Notes in Computer Science*, pages 154–171, Glasgow, 1995. Springer-Verlag.
9. G. Huet and J. M. Hullot. Proofs by induction in equational theories with constructors. *Journal of Computer and System Sciences*, 25(2):239–266, 1982.
10. F. Kamareddine and F. Monin. On formalised proofs of termination of recursive functions. In G. Nadathur, editor, *Proceedings of the International Conference on Principles and Practice of Declarative Programming*, volume 1702 of *Lecture Notes*

in Computer Science, pages 29–46, 1999. Springer-Verlag.

11. F. Kamareddine and F. Monin. On Automating Inductive and Non-Inductive Termination Methods. In P.S. Thiagarajan and R. Yap, editors, *Proceedings of the Asian Computing Science Conference*, volume 1742 of *Lecture Notes in Computer Science*, pages 177-189, 1999. Springer-Verlag.
12. F. Kamareddine and F. Monin. On the simplification of formal proofs with abstracted propositions in an automated system for program synthesis. Technical report, Heriot-Watt University, 2000.
13. D. Kapur, P. Narendran, and H. Zhang. Automatic inductionless induction using test sets. *Journal of Symbolic Computation*, 11(1-20:83-111, 1991.
14. D. E. Knuth and P. B. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational problems in abstract algebra*, pages 263–297. Pergamon Press, 1970.
15. T. Kolbe. Challenge problems for automated termination proofs of term rewriting systems. Technical report, Technische Hochshule Darmstadt, Alexanderstr 10, 64283 Darmstadt, Germany, 1996.
16. J. L. Krivine. *Lambda-calculus, Types and Models*. Computers and Their Applications. Ellis Horwood, 1993.
17. J. L. Krivine and M. Parigot. Programming with proofs. *J. Inf. Process Cybern*, 26(3):149–167, 1990.
18. D. Leivant. Typing and computational properties of lambda expression. *Theoretical Computer Science*, 44:51–68, 1986.
19. P. Manoury. A user's friendly syntax to define recursive functions as typed lambda-terms. In *Proceedings of Type for Proofs and Programs TYPES'94*, volume 996 of *Lecture Note in Computer Science*, 1994. Springer-Verlag.
20. P. Manoury and M. Simonot. *Des preuves de totalité de fonctions comme synthèse de programmes*. PhD thesis, University Paris 7, 1992.
21. P. Manoury and M. Simonot. Automatizing termination proofs of recursively defined functions. *Theoretical Computer Science*, 135(2):319–343, 1994.
22. F. Monin and M. Simonot. An ordinal measure based procedure for termination of functions. *Theoretical Computer Science*, 254:63–94, 2001.
23. D. R. Musser. On proving inductive properties of abstract data types. In *Proceedings 7th ACM symposium on principles of Programming Languages*, ACM, pages 154–162, 1980.
24. F. Nielson and H. R. Nielson. operational semantics of termination types. *Nordic Journal of Computing*, 3(2):144–187, 1996.
25. M. Parigot. Recursive programming with proofs. *Theoretical Computer Science*, 94(2):335–356, 1992.
26. C. Sangler. Termination of algorithms over non-freely generated data types. In *Proceedings of International Conference on Automated Deduction*, volume 1104 of *Lecture Notes in Artificial Intelligence*, pages 121–136, 1996. Springer-Verlag.
27. J. Steinbach. Generating polynomial orderings. *Information Processing Letters*, 49, 1994.
28. J. Steinbach. Simplification orderings: history of results. *Fundamenta Informaticae*, 24:47–87, 1995.
29. J. J. Thiel. Stop losing sleep over incomplete data type specifications. In *Proceedings 11th ACM symposium on principles of programming languages*, pages 76–82, 1984.

30. C. Walther. On proving the termination of algorithms by machine. *Artificial Intelligence*, 71(1):101–157, 1994.