Electronic Notes in Theoretical Computer Science 67 (2002)
URL: http://www.elsevier.nl/locate/entcs/volume67.html 20 pages

On automating the extraction of programs from proofs using product types

Fairouz Kamareddine [1]

School of Mathematical and Computer Sciences, Heriot-Watt University, Edinburgh, Scotland

François Monin [2]

IRISA, Campus de Beaulieu, 35 042 Rennes Cedex, France

Mauricio Ayala-Rincón [3]

Departamento de Matemática, Universidade de Brasília, Brasília D.F., Brasil

Abstract

We investigate an automated program synthesis system based on the paradigm of programming by proofs. To automatically extract a λ-term that computes a recursive function given by a set of equations the system must find a formal proof of the totality of the given function. Because of the particular logical framework, usually such approaches make it difficult to use techniques such as those in rewriting theory. We overcome this difficulty for the automated system that we consider by exploiting product types. As a consequence, this would enable the incorporation of termination techniques used in other areas while still extracting programs.

Keywords: Program extraction, automated termination, product types.

1 Introduction

The Curry-Howard isomorphism [3] which gives a correspondence between programs and proofs of specifications plays a major role in type theory. Programming methods using the *proof as program* paradigm ensure some correctness of programs extracted from a proof of function totality and provides a logical framework for which the behaviour of programs can be analysed. Systems which exploit the proof as program paradigm include *Second Order Functional Arithmetic AF2* [6,8] and *Recursive Type Theory TTR* [15]. Both

[1] Email: fairouz@cee.hw.ac.uk
[2] Email: monin@irisa.fr
[3] Email: ayala@mat.unb.br

AF2 and *TTR* use equations as algorithmic specifications where the compilation phase corresponds to formal termination proofs of the specifications of functions from which λ-terms that compute the functions are extracted.

Using the logical framework of *TTR*, an automated system called *ProPre*, has been developed by P. Manoury and M. Simonot [11,10]. The automated termination problem turns out to be a major issue in the development of the system. Alongside the system, where data types and specifications of functions are introduced by the user in an ML-style, an algorithm has been designed using strategies to search for formal termination proofs for each specification. When the system succeeds in developing a formal termination proof for a specification, a λ-term that computes the function is given.

As mentioned in [11], the automated termination proofs in this system differ from the usual techniques of rewriting systems because they have to follow several requirements. They must be *proofs of totality* in order to enable the extraction of λ-terms. In *ProPre*, one has to make sure not only that the programs will give an output for any input, but also that for any well-typed input the result will also be well-typed. Finally, the proofs must also be expressed in a formal logical framework, namely, the natural deduction style. The λ-terms are obtained from the proof trees that are built in a natural deduction style according to the recursive type theory *TTR*.

Therefore enhancing automated proofs strategy is a central issue in programming languages like *AF2* or *TTR*. While termination methods for functional programing based on ordinal measures have been developed in [13,5] relating to the formal proofs devised in [11,12], the purpose of this paper is to analyse in some sense the reverse of the question. That is, we analyse the possibility to incorporate new termination techniques for the extraction of programs in the *ProPre* or *TTR* context.

In order to simplify the analysis of the formal proofs obtained in the logical framework of *ProPre*, we show that the kernel of these formal proofs, called *formal terminal state property (ftsp)*, can be abstracted using a simple data structure. This gives rise to a simple termination property, which we call *abstract terminal state property (atsp)*. The interest of *atsp* is that on one hand the termination condition is sufficient to show the termination using the ordinal measures of [5] independently of the particular logical framework of *ProPre*, and on the other hand we also prove that we can automatically reconstruct a formal proof directly from an *atsp* so that a lambda term can be extracted. That is to say the first result of this paper is to establish a correspondence between *atsp* with a class of ordinal measures in a simple context for the termination and the formal proofs built in *ProPre*.

This correspondence implies that the termination proofs of recursive functions obtained in [4] do not admit in general a formal proof in *ProPre*. Indeed the class of these functions is larger than those proven with the class of ordinal measures of [13,5]. To overcome the fact that there is in general no formal proof in *ProPre* for these functions, the second result presented in this paper allows the synthesis of these functions still making use of the whole framework of *ProPre* but in a different way in *TTR*. Actually the result turns out to be stronger since it can be applied to recursive functions whose termination is proven by other automated methods such as techniques coming from rewriting theory (see e.g. [1]). The principle consists in simulating a semantic method. That is, from a well-founded ordering for which each recursive call is decreasing, one must be able to build a formal proof by considering general induction on tuples of arguments of the function. Though the principle is natural, this approach becomes difficult when we want to extract programs because we have to take into account the logical framework and the structures of the proofs.

2 Logical framework

ProPre [9,11,12] relies on the *proofs as programs* paradigm that exploits the Curry-Howard isomorphism and deals with the recursive type theory *TTR* [15]. In *ProPre*, the user needs to only define data types and functions. λ-terms are automatically extracted from the formal proofs of the termination statements of functions which can be viewed as the compilation part. *ProPre* deals with recursive functions. The data types and functions are defined in an ML like syntax. For instance, if `N` denotes the type of natural numbers, then the list of natural numbers is defined by:

```
Type Ln :  Nil | Cons N Ln;
```

and the *append* function is specified by:

```
Let append :  Ln, Ln -> Ln
Nil y => y |    (Cons n x) y => (Cons n (append x y));
```

Once a data type is introduced by the user, a second order formula is automatically generated. E.g., the following second order formula is automatically generated and associated to the list of natural numbers:

$Ln(x) := \forall X(X(nil) \to (\forall n(N(n) \to \forall y(X(y) \to X(cons(n,y))))) \to X(x))$.

This formula stands for the least set that contains the *nil* element and is closed under the constructor *cons*. Each data type will be abbreviated by a unary data symbol, as it is for instance with the symbol N that represents the data type of natural numbers. Furthermore, once a function is specified in the system, a *termination statement* is automatically produced [10]. As an example, the termination statement of the *append* function is the formula:

$\forall x(Ln(x) \to \forall y(Ln(y) \to Ln(append(x,y))))$.

The system then attempts to prove the termination statement of the function using the set of equations that define the function. In a successful case, a λ-term that computes the function is synthesized from the building of a formal proof in a natural deduction style [11]. Informally, if T is a λ-term obtained for the function *append* and t_1, t_2 are λ-terms that respectively model terms u_1 of type `Ln` and u_2 of type `Ln`, then the λ-term $((T\ t_1)\ t_2)$ reduces to a normal form V that represents the value of $append(u_1,u_2)$ of type `Ln`.

See [6,7,14,15] for details of the theory that allows to derive λ-terms from termination proofs of the specification in natural deduction style.

2.1 The typing rules of AF2 (which are also part of TTR)

We assume a set $\mathcal{F}$ of function symbols and a countable set $\mathcal{X}$ of individual variables. The logical terms are inductively defined as follows:

- individual variables are logical terms;
- if f is an n-ary function symbol in $\mathcal{F}$ and $t_1, \dots, t_n$ are logical terms, then $f(t_1, \dots, t_n)$ is a logical term.

We assume a countable set of predicate variables and define Formulas by:

- if X is an n-ary predicate variable and $t_1, \dots, t_n$ are logical terms, then $X(t_1, \dots, t_n)$ is a formula;
- if A and B are formulas then $A \to B$ is a formula;
- if A is a formula and η is a 1st or 2nd order variable, then $\forall \eta A$ is a formula.

We use $\forall x A \to B$ to denote $\forall x(A \to B)$. A formula of the form $F_1 \to (F_2 \to \dots (F_{n-1} \to F_n) \dots)$ will also be denoted by $F_1, \dots, F_n \to F$. For instance $\forall x D_1(x), \forall y D_2(y) \to F$ stands for the formula $\forall x(D_1(x) \to \forall y(D_2(y) \to F))$.

A typing judgment is an expression of the form: "$x_1 : F_1, \dots, x_n : F_n \vdash_{\mathcal{E}} t : F$", where $x_1, \dots, x_n$ are distinct λ-variables, t is a λ-term, $F, F_1, \dots, F_n$ are formulas and $\mathcal{E}$ is a set of equations on logical terms. The left-hand side of the judgment is called the context. Note that we can freely use the same notation for both the λ-terms and the logical terms which occur in the formulas, as the context will clarify whether a term is a λ-term or a logical term. In particular, the word "variable" may also refer to a "λ-variable". The typing rules of *AF2* are given in Table 1 where $\mathcal{E}$ is a set of equations on logical terms, Γ is a context of the form $x_1 : A_1, \dots, x_n : A_n$ and may be empty; y (resp. Y) is a first (resp. second) order variable not occurring free in $A_1, \dots, A_n$; τ, u, v are first order terms and T is a formula.

$$\frac{}{\Gamma, x : A \vdash_{\mathcal{E}} x : A}\ (ax) \qquad \frac{\Gamma \vdash_{\mathcal{E}} t : A[u/y] \quad \mathcal{E} \vdash u = v}{\Gamma \vdash_{\mathcal{E}} t : A[v/y]}\ (eq)$$

$$\frac{\Gamma, x : A \vdash_{\mathcal{E}} t : B}{\Gamma \vdash_{\mathcal{E}} \lambda x.t : A \to B}\ (\to_i) \qquad \frac{\Gamma \vdash_{\mathcal{E}} u : A \quad \Gamma \vdash_{\mathcal{E}} t : A \to B}{\Gamma \vdash_{\mathcal{E}} (t\ u) : B}\ (\to_e)$$

$$\frac{\Gamma \vdash_{\mathcal{E}} t : A}{\Gamma \vdash_{\mathcal{E}} t : \forall y A}\ (\forall_i^1) \qquad \frac{\Gamma \vdash_{\mathcal{E}} t : \forall y A}{\Gamma \vdash_{\mathcal{E}} t : A[\tau/y]}\ (\forall_e^1)$$

$$\frac{\Gamma \vdash_{\mathcal{E}} t : A}{\Gamma \vdash_{\mathcal{E}} t : \forall Y A}\ (\forall_i^2) \qquad \frac{\Gamma \vdash_{\mathcal{E}} t : \forall Y A}{\Gamma \vdash_{\mathcal{E}} t : A[T/Y]}\ (\forall_e^2)$$

Table 1
Rules of the Second Order Functional Arithmetic (AF2)

Types and formal data types play an important role in *AF2* and *TTR* in relation to a notion of *realizability* [7] that ensures the extracted λ-terms compute the defined functions (cf. [6,7]). If for an n-ary function f we have: $\vdash_{\mathcal{E}} t : \forall x_1 \dots \forall x_n (D_1(x_1) \to (\dots \to (D_n(x_n) \to D(f(x_1, \dots, x_n)) \dots)$

for some λ-term t where $D_1, \ldots, D_n, D$ denote formal data types, then the λ-term t computes the function f according to the set $\mathcal{E}$.

2.2 Some rules of TTR

As for $AF2$, we do not state the data types and the realizability notion of *TTR*. In particular, we do not give the second order least fixed point operator μ (see [14]) which allows defining the data types which are represented here by unary data symbols $D, D', \ldots, D_1, \ldots, D_n$. For the sake of clarity we do not state all the rules (which also include those of *AF2*).

In *TTR*, a binary symbol $\upharpoonright$, called *hiding operator* in [14], is added. Its meaning is a conjunction which only keeps the algorithmic contents of the left part in order to prevent unnecessary algorithmic content of the termination proof to be carried out in the λ-terms (see [15,11]). It is used with a relation $\prec$, made precise below. The definition of formulas given in section 2.1 is now completed as follows: If A is a formula, and u, v are terms then $A \upharpoonright (u \prec v)$ is a formula. The rules related to the hiding operator are given in Table 2.

$$\frac{\Gamma \vdash t : A \quad \Gamma \vdash_{\mathcal{E}} e}{\Gamma \vdash_{\mathcal{E}} t : A \upharpoonright e}\ (\upharpoonright_1) \qquad \frac{\Gamma \vdash_{\mathcal{E}} t : A \upharpoonright e}{\Gamma \vdash t : A}\ (\upharpoonright_2) \qquad \frac{\Gamma \vdash_{\mathcal{E}} t : A \upharpoonright e}{\Gamma \vdash_{\mathcal{E}} e}\ (\upharpoonright_3)$$

Table 2

Rules of the hiding operator $\upharpoonright$.

If A is a formula where a distinguished variable x occurs, we abbreviate the formula $A[u/x] \upharpoonright (u \prec v)$ with the notation $Au_{\prec v}$.

Among the rules of *TTR*, several rules are used to reproduce, from the programming point of view, the reasoning by induction. The rule below stands in *TTR* for an external induction rule where the relation $\prec$ denotes a well-founded partial ordering on the terms of the algebra:

$$\frac{\Gamma \vdash_{\mathcal{E}} \ t : \forall x[\forall z[Dz_{\prec x} \to B[z/x]] \to [D(x) \to B]]}{\Gamma \vdash_{\mathcal{E}} \ (T\ t) : \forall x[D(x) \to B]} \quad (Ext)$$

In the rule (Ext), the lambda term T is the Turing fixed-point operator, D is a data type and x is a variable not occurring in the formula B.

From the (Ext) rule, it is possible to derive the $\widetilde{Ind}$ formula:

$\widetilde{Ind} := \forall x(D^r(x) \to \forall X(\forall y(D^r(y) \to \forall z(D^r z_{\prec y} \to X(z)) \to X(y)) \to X(x)))$.

That is, there is a λ-term that witnesses the proof of $\widetilde{Ind}$. This is stated in Lemma 2.1 below, which is given for the type of natural numbers in [14].

Lemma 2.1. *For each recursive data type, there exists a λ-term ind such that: $\vdash ind : \widetilde{Ind}$.*

The proof of Lemma 2.1 in [14], given only with the type of natural numbers, can be applied to any data type: $ind = (T\ \lambda x \lambda y \lambda z((z\ y)\ \lambda m((x\ m)\ z)))$, where T is the Turing fixed-point operator, is valid for any data type. Lemma 2.1 is useful for the definition of a macro-rule, called the Ind-rule, in *ProPre*.

2.3 The ProPre system

We assume that the set of functions $\mathcal{F}$ is divided into two disjoint sets, the set $\mathcal{F}_c$ of *constructor symbols* and the set $\mathcal{F}_d$ of *defined function symbols* also called defined functions. Each function f is supposed to have a type denoted by $D_1, \ldots, D_n \to D$ where $D_1, \ldots, D_n, D$ denote data symbols and n denotes the arity of the function f. We may write $f : D_1, \ldots, D_n \to D$ to both introduce a function f and its type $D_1, \ldots, D_n \to D$.

Definition 2.2. [Specification; termination statement; recursive call]
• A *specification* $\mathcal{E}_f$ of a defined function $f : D_1, \ldots, D_n \to D$ in $\mathcal{F}_d$ is a non overlapping set of left-linear equations $\{(e_1, e'_1), \ldots, (e_p, e'_p)\}$ such that for all $1 \leq i \leq p$, e_i is of the form $f(t_1, \ldots, t_n)$ where t_j is a constructor term (i.e. without occurrences of defined function symbols) of type D_j, $j = 1, \ldots, n$, and e'_i is a term of type D.
• The *termination statement* of a function $f : D_1, \ldots, D_n \to D$ is the formula $\forall x_1(D_1(x_1) \to \ldots \to \forall x_n(D_n(x_n) \to D(f(x_1, \ldots, x_n))))$.
• Let $\mathcal{E}_f$ a specification of a function f. A *recursive call* of f is a pair (t, v) where t is the left-hand side of an equation (t, u) of $\mathcal{E}_f$ and v a subterm of u of the form $f(v_1, \ldots, v_n)$.

An equation (l, r) of a specification may be written $l = r$ (as an equational axiom in *TTR*). We may also drop the brackets to ease the readability.

The formal proofs of *ProPre*, called I-proofs, are built upon distributing trees, based on two main rules derived from the *TTR Struct* rule and the *Ind* rule in [11]. The distributing trees built in *ProPre* are characterized by a property called *formal terminal state property*. This section presents these two main rules, the distributing trees and the formal terminal state property.

Notation 2.3. If P is the formula $F_1, \ldots, F_k, \forall x D'(x), F_{k+1}, \ldots, F_m \to D(t)$, then $P_{-D(x)}$, will denote the formula $F_1, \ldots, F_k, F_{k+1}, \ldots, F_m \to D(t)$.

The above notation is correct as it will be used at the same time when the quantified variable x will be substituted by a term in the formula $P_{-D(x)}$ with respect to the context (cf. next two lemmas with Notation 2.4) or when the variable x will be introduced in the context.

Notation 2.4. Let C be a constructor symbol of a type $D_1, \ldots, D_k \to D$. Let $x_1, \ldots, x_k, z$ be distinct variables. Let $F(x)$ be a formula in which the variable x is free and the variables $z, x_1, \ldots, x_k$ do not occur and let $t = C(x_1, \ldots, x_k)$. Then $\Phi_C(F(x))$ and $\Psi_C(F(x))$ will be respectively the following formulas:

- $\Phi_C(F(x))$ is: $\forall x_1 D_1(x_1), \ldots, \forall x_k D_k(x_k) \to F[t/x]$;
- $\Psi_C(F(x))$ is: $\forall x_1 D_1(x_1), \ldots, \forall x_k D_k(x_k), \forall z(Dz_{\prec t} \to F[z/x]) \to F[t/x]$.

The notation may suggest some kind of formulas that are actually useful in the construction of I-proofs which are defined as follows:

Definition 2.5. [***I-formulas and restrictive hypothesis***]
• A formula F is called an *I-formula* iff F is of the form $H_1, \ldots, H_m \to D(f(t_1, \ldots, t_n))$ for some:
– data type D, defined function f,
– formulas H_i for $i = 1, \ldots, m$ such that H_i is of the form $\forall x D'(x)$ or of the

form $\forall z(D' z_{\prec u} \rightarrow F')$ for some data type D', I-formula F' and term u.
- An *I-restrictive hypothesis* of an I-formula F of the form $H_1, \ldots, H_m \rightarrow D(f(t_1, \ldots, t_n))$ is a formula H_i of the form $\forall z(D' z_{\prec u} \rightarrow F')$. We say that H' is a *restrictive hypothesis* to an I-restrictive hypothesis $H = \forall z(D' z_{\prec u} \rightarrow F')$ if H' is an I-restrictive hypothesis of the I-formula F'.

As by definition, an I-formula is recursive, it may involve sub-I-formulas. An I-restrictive hypothesis is not an I-formula. We use the term *restrictive hypothesis* to also denote I-restrictive hypothesis. The termination statement of a defined function is an I-formula which has no restrictive hypothesis.

The lemmas below state that one can use two additional rules, called *Struct* rule and *Ind* rule, in *TTR* as they can be derived from the other rules of *TTR*. These rules correspond to macro-rules, the former one can be seen as a reasoning by cases, while the last one stands for an induction rule.

Lemma-Definition 2.6. [**The** *Ind* **rule**]
Let D be a data type and consider all the constructor functions C_i of type $D_{i_1}, \ldots, D_{i_k} \rightarrow D$, $0 \leq i_k$, $i = 1, \ldots, q$. Let P be a formula of the form $F_1, \ldots, F_k, \forall x D(x), F_{k+1}, \ldots, F_m \rightarrow D'(t)$, and Γ a context. For $\Psi_{C_i}(P_{-D(x)})$ given as in Notation 2.4, the induction *Ind* rule on type D is:

$$\frac{\Gamma \vdash \Psi_{C_1}(P_{-D(x)}) \quad \ldots \quad \Gamma \vdash \Psi_{C_q}(P_{-D(x)})}{\Gamma \vdash P} \qquad Ind(x)$$

Along with the *Ind* rule, the *Struct* rule defined below, which is also a macro-rule derived from *TTR*, can be considered as a reasoning by cases.

Lemma-Definition 2.7. [**The** *Struct* **rule**]
Let D be a data type and consider all the constructor functions C_i of type $D_{i_1}, \ldots, D_{i_k} \rightarrow D$, $0 \leq i_k$, $i = 1, \ldots, q$. Let P be a formula of the form $F_1, \ldots, F_k, \forall x D(x), F_{k+1}, \ldots, F_m \rightarrow D'(t)$, and Γ a context. For $\Phi_{C_i}(P_{-D(x)})$ given as in Notation 2.4, the *Struct* rule on type D is:

$$\frac{\Gamma \vdash \Phi_{C_1}(P_{-D(x)}) \quad \ldots \quad \Gamma \vdash \Phi_{C_q}(P_{-D(x)})}{\Gamma \vdash P} \qquad Struct(x)$$

Due to these lemmas, two macro-rules can be added in *TTR*: the *Struct*-rule (Lemma 2.7) and the *Ind*-rule (Lemma 2.6). From these rules, *distributing trees* can be built in *ProPre* (see Definition 2.10).

Remark 2.8. I-formulas are preserved by the *Struct*-rule and the *Ind*-rule. That is, if P is an I-formula, then so are: $\Phi_C(P_{-D(x)})$ and $\Psi_C(P_{-D(x)})$.

Definition 2.9. [**Heart of formula**] The *heart* of a formula of the form $F = H_1, \ldots, H_m \rightarrow D(t)$, where D is a recursive data type, will be the term t, denoted by $H(F)$. The distributing trees are defined as follows:

Definition 2.10. [**Distributing tree**] Let $\mathcal{E}_f$ be a specification of a function $f : D_1, \ldots, D_n \rightarrow D$. $\mathcal{A}$ is a *distributing tree for* $\mathcal{E}_f$ iff $\mathcal{A}$ is a proof tree built only with the *Struct* rule and *Ind* rule where:

(i) the root of A is the termination statement of f with the empty context, i.e.: $\vdash \forall x_1 D_1(x_1), ..., \forall x_n D_n(x_n) \to D(f(x_1, ..., x_n))$.

(ii) if $\mathcal{L} = \{\Gamma_1 \vdash F_1, ..., \Gamma_q \vdash F_q\}$ is the set of $\mathcal{A}$'s leaves, then there exists a one to one application $\mathcal{B}$: $\mathcal{L} \hookrightarrow \mathcal{E}_f$ such that $\mathcal{B}(L) = (t, u)$ with $L = (\Gamma \vdash F)$ in $\mathcal{L}$ and the heart of F is $H(F) = t$.

Note that it can be inductively checked, from the root, using remark 2.8, that any formula in a distributing tree is an I-formula.

The I-proofs found by *ProPre* are formal termination proofs of termination statements of defined functions. They are divided into three phases:

(i) the development of a distributing tree for the specification of a defined function, characterized by the so-called formal terminal state property;

(ii) each leaf of the distributing tree is extended into a new leaf by an application of an (eq) rule;

(iii) each leaf, coming from the second step, is extended with a new sub-tree, with the use of rules defined in [11], whose leaves end with axiom rules.

Due to the following fact proven in [11], it is not necessary to consider in this paper the middle and upper parts of proof trees built in *ProPre*:

Fact 2.11. A distributing tree T can be (automatically) extended into a complete proof tree iff T enjoys the so-called *formal terminal state property.*

Hence, in order to complete the proof tree and state the termination of the function, it suffices to look at distributing trees that have the formal terminal state property. Therefore it remains for us to state the mentioned property.

Definition 2.12. An I-formula or a restrictive hypothesis P can be applied to a term t if the heart $H(P)$ of P matches t according to a substitution σ where for each variable x that occurs free in P we have $\sigma(x) = x$.

The relation $\prec$ of Definition 2.5 deals with the measure $|\ .\ |_{\#}$ on the terms, ranging over natural numbers, which counts the number of subterms of a given term t (including t), and is interpreted as follows:

Definition 2.13. Let $\mathcal{V}ar(t)$ be the set of variables occurring in t. Let u, v be terms. We say $u \prec v$ iff: $|u|_{\#} < |v|_{\#}$, $\mathcal{V}ar(u) \subseteq \mathcal{V}ar(v)$, and u is linear.

This clearly defines a well-founded ordering $\prec$ on terms. We can now state the main property that a distributing tree must enjoy in the I-proofs of *ProPre*.

Definition 2.14. [**Formal Terminal State Property**] Let $\mathcal{E}_f$ be a specification of a function f and $\mathcal{A}$ be a distributing tree for $\mathcal{E}_f$. We say that $\mathcal{A}$ satisfies the *formal terminal state property* (*ftsp*) iff for all leaves $L = (\Gamma \vdash F)$ of $\mathcal{A}$ with the equation $e \in \mathcal{E}_f$ such that $\mathcal{B}(L) = e$, where $\mathcal{B}$ is the application given in Definition 2.10, and for all recursive calls (t, v) of e, there exists a restrictive hypothesis $P = \forall z D z_{\prec s}, H_1, \ldots, H_k \to D(w)$ of F and a substitution σ such that P can be applied to v according to σ with:

(i) $\sigma(z) \prec s$ and

(ii) for all restrictive hypothesis H of P of the form $\forall y D' y_{\prec s'} \to K$ there is a restrictive hypothesis H_0 of F of the form $\forall y D' y_{\prec s_0} \to K$ with $\sigma(s') \preceq s_0$.

So, *ProPre* establishes the termination of a function f by showing that the distributing tree of the specification of f (a partial tree whose root is the termination statement of f) has the formal terminal state property (hence can be extended to a complete proof tree of the termination statement of f).

3 The abstract terminal state property

Proof structures can often be heavy and difficult to work with. However, in the constructive framework of the Curry-Howard isomorphism, compiling a recursive algorithm corresponds to establishing a formal proof of its totality. In *ProPre*, termination proofs play an important role as they make it possible to obtain λ-terms that compute programs. We set out to simplify the termination techniques developed in *ProPre* by showing that its automated formal proofs can be abstracted giving rise to a simpler property which respects termination. Instead of dealing with formulas, we will use the simpler concept of functions. Also, instead of data symbols, we will use sorts and assume that there is a correspondence between the data types of *ProPre* and our sorts. Instead of the complex concept of distributing trees of *ProPre* (Definition 2.10), we will use the simpler *term distributing trees* of [13]. By living in the easier framework, we will introduce the new *abstract terminal state property* which will play for term distributing trees a similar role to that played by the formal terminal state property for distributing trees. In this section we present a data structure for which we will be able to introduce a new termination property.
We take a countable set $\mathcal{X}$ of individual variables, assume that each variable of $\mathcal{X}$ has a unique sort and that for each sort s there is a countable number of variables in $\mathcal{X}$ of sort s. For sort s, $F \subseteq \mathcal{F}$, and $X \subseteq \mathcal{X}$, $\mathcal{T}(F, X)_s$ denotes the set of terms of sort s built from F and X. If X is empty we write $\mathcal{T}(F)_s$.

We recall the definition of *term distributing trees* of [13]. A term distributing tree is much simpler than the distributing tree of *ProPre* given in Definition 2.10. The novelty of this section will be a term distributing tree equipped with abstract terminal state property (Definition 3.5 below).

Definition 3.1. [**Term distributing tree**] Let $\mathcal{E}_f$ be a specification of a function $f : s_1, \ldots, s_n \to s$. A tree T is a *term distributing tree* for $\mathcal{E}_f$ iff:

(i) its root is of the form $f(x_1, \ldots, x_n)$ where x_i is a variable of sort s_i, $i \leq n$;

(ii) each leaf is a left-hand side of an equation of $\mathcal{E}_f$ (up to var. renaming);

(iii) each node $f(t_1, \ldots, t_n)$ of T admits one variable x' of a sort s' such that the set of children of the node is $\{f(t_1, \ldots t_n)[C(x'_1, \ldots x'_r)/x'],$ where $x'_1, \ldots, x'_r$ are not in $t_1, \ldots t_n$ and $C : s'_1, \ldots, s'_r \to s' \in \mathcal{F}_c\}$.

A term distributing tree can bee seen as a skeleton form of a distributing tree T by taking the heart of the formulas in the nodes of T, which gives rise to an operator $\mathcal{H}$ illustrated by Figure 1.

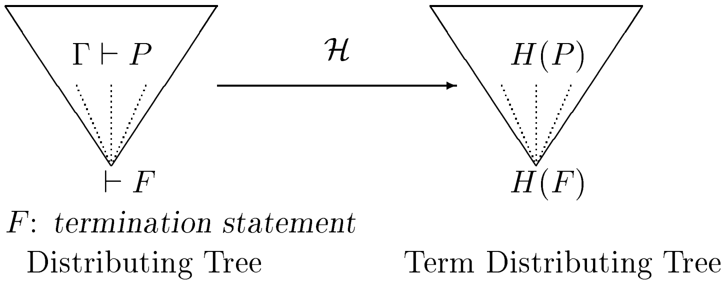


Fig. 1. The operator $\mathcal{H}$

Proposition 3.2. If there is a distributing tree for a specification $\mathcal{E}_f$ of a function f then there is also a term distributing tree for the specification $\mathcal{E}_f$.

A term distributing tree is easier to handle than a distributing tree. But, in both parts of Figure 1, term distributing trees and distributing trees may have no termination property. However, by Fact 2.11, a function terminates if we have a distributing tree that satisfies a right terminal state property. What we want is to define a notion on the term distributing trees that also ensures the termination of functions. We first give some notations and remarks.

Notation 3.3. Let T be a term distributing tree with root θ_1.
• A branch b from θ_1 to a leaf θ_k is denoted by $(\theta_1, y_1), \ldots, (\theta_{k-1}, y_{k-1}), \theta_k$ where for each $i \leq k-1$, y_i corresponds to the variable x' for node θ_i in the third clause of Definition 3.1. We use L_b to denote the leaf of the branch b.
• If a node θ matches a term u of a recursive call (t, u) then the substitution will be denoted by $\rho_{\theta,u}$ (in particular in Definition 3.5).
• For a term t of a left-hand side of an equation, $b(t)$ will denote the branch in the term distributing tree that leads to t (second clause of Definition 3.1).

Remark 3.4. • Let $f : s_1, \ldots, s_n \to s$ be a function and $\mathcal{E}_f$ be a specification of f. Let T be a term distributing tree of $\mathcal{E}_f$. Then for each $(w_1, \ldots, w_n)$ of $\mathcal{T}(\mathcal{F}_c)_{s_1} * \ldots * \mathcal{T}(\mathcal{F}_c)_{s_n}$ there is one and only one leaf θ of T and a ground constructor substitution φ such that $\varphi(\theta) = f(w_1, \ldots, w_n)$.
• Let T be a term distributing tree for a specification and let b be a branch from the root θ_1 of T to a leaf θ_k with $b = (\theta_1, x_1), \ldots, (\theta_{k-1}, x_{k-1}), \theta_k$. Then for each node θ_i, θ_j with $1 \leq i \leq j \leq k$, there exists a constructor substitution, denoted $\sigma_{\theta_j,\theta_i}$, such that $\sigma_{\theta_j,\theta_i}(\theta_i) = \theta_j$.

Now, we give the *abstract terminal state property* for term distributing trees:

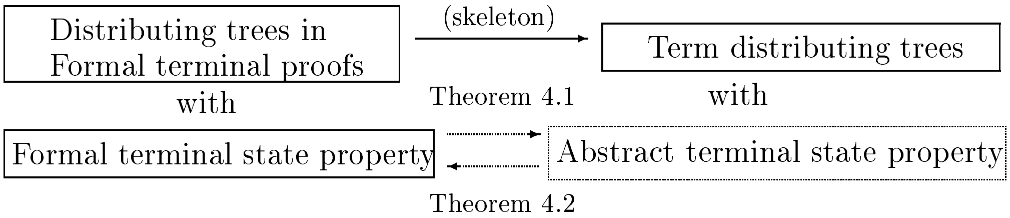
Definition 3.5. [**Abstract terminal state property**] Let T be a term distributing tree for a specification. We say that T has the *abstract terminal state property* (*atsp*) if there is an application $\mu : T \to \{0, 1\}$ on the nodes of T such that if L is a leaf, $\mu(L) = 0$, and for every recursive call (t, u), there is a node (θ, x) in the branch $b(t)$ with $\mu(\theta) = 1$ such that θ matches u with $\rho_{\theta,u}(x) \prec \sigma_{L_{b(t)},\theta}(x)$ (cf. Notations 3.3 and Remark 3.4) and for all ancestors (θ', x') of θ in $b(t)$ with $\mu(\theta') = 1$, we have $\rho_{\theta',u}(x') \preceq \sigma_{L_{b(t)},\theta'}(x')$.

Note that similarly to term distributing trees, no formula is mentioned in the definition of *atsp* and hence *atsp* is easier to handle than *ftsp* (Definition 2.14) because *atsp* only uses relations of substitutions where all proposition informations have been abstracted. However, it is not obvious that a term distributing tree that satisfies *atsp* implies the termination of the given function. A way to prove this fact would be to infer some measures from such distributing trees and to show that these measures have the decreasing property through the recursive calls of the given function so that the function terminates.

We will not follow this way and will instead prove in the next section the stronger result that from a term distributing tree that has the *atsp* we can reconstruct an I-proof, which implies that the given function terminates and also enables a λ-term that computes the function to be extracted.

4 Building formal proofs from skeleton forms

We show that • the *atsp* is an abstract form of the *ftsp* (Theorem 4.1) and that • the *atsp* is a sufficient condition to construct a distributing tree with the *ftsp* from a term distributing tree –skeleton form– (Theorem 4.2). This can be illustrated with the picture below.



We start by extending the application $\mathcal{H}$ (Figure 1) into a new operator $\mathcal{H}'$ from a distributing tree $\mathcal{A}$ to the term distributing tree $\mathcal{H}(\mathcal{A})$ which is now equipped with an application $\mu : \mathcal{H}(\mathcal{A}) \to \{0,1\}$ defined on the node of $\mathcal{H}(\mathcal{A})$, so that $\mathcal{H}'(\mathcal{A})$ will be $(\mathcal{H}(\mathcal{A}), \mu)$. A term distributing tree equipped with an application μ will also be called a μ-term distributing tree.

To define the operator $\mathcal{H}'$, the application μ is given as follows: Let $\mathcal{A}$ be a distributing tree and $(\Gamma \vdash P)$ be a node of $\mathcal{A}$. If $(\Gamma \vdash P)$ is a leaf, we take $\mu(H(P)) = 0$. If not, we consider $\mu(H(P)) = 1$ if the rule applied on $(\Gamma \vdash P)$ in $\mathcal{A}$ is the Ind rule and $\mu(H(P)) = 0$ otherwise.

Note that $\mathcal{H}$ is not injective: there is at least two distinct distributing trees $\mathcal{A}$ and $\mathcal{A}'$ such that $\mathcal{H}(\mathcal{A}) = \mathcal{H}(\mathcal{A}')$. However, $\mathcal{H}'$ is injective. Actually if we consider term distributing trees equipped with a μ-application, then $\mathcal{H}'$ becomes bijective and the inverse operator of $\mathcal{H}'$ can be stated by:

Lemma-Definition 4.1. [**$\mathcal{D}$, the inverse of $\mathcal{H}'$**] Let $\mathcal{E}_f$ be a specification of a function $f : s_1, \ldots, s_n \to s$, and let (T, μ) be a term distributing tree for $\mathcal{E}_f$ (equipped with a μ application). There is one and only one distributing tree $\mathcal{A}$ for $\mathcal{E}_f$ such that $\mathcal{H}'(\mathcal{A}) = (T, \mu)$. This one can be automatically obtained from (T, μ) and we define the application $\mathcal{D}$ with $\mathcal{D}(T, \mu) = \mathcal{A}$.

Proof. Let $F = \forall x_1 D_1(x_1), \ldots, \forall x_n D_n(x_n) \to D(f(x_1, \ldots, x_n))$ be the termination statement of f. We can inductively *build* a distributing tree $\mathcal{A}$ of the same size as T by taking the root of $\mathcal{A}$ to be $\vdash F$ and assuming the existence of a node $(\Gamma \vdash P)$ of $\mathcal{A}$, for P is an I-formula, such that:

i) P is of the form: $F_1, \ldots, F_r, \forall x D'(x), F_{r+1}, \ldots, F_p \rightarrow D(f(t_1, \ldots, t_n))$ where D and D' are data symbols, and variables in the heart of P are bound,

ii) T admits a level, the same as those $(\Gamma \vdash P)$ in $\mathcal{A}$, such that the node θ at this level is distinct from a leaf, with $\theta = f(t_1, \ldots, t_n)$ whose variable according to Definition 3.1.iii is the variable x of sort s' associated to D'.

From above, we build the children nodes of $(\Gamma \vdash P)$ in $\mathcal{A}$ as follows:

- If $\mu(\theta) = 0$, the node $(\Gamma \vdash P)$ is extended by the *Struct* rule on x in P.
- If $\mu(\theta) = 1$, the node $(\Gamma \vdash P)$ is extended using the *Ind* rule on x in P.

In both cases, as P is an I-formula, if P'_j denotes either $\Psi_{C_j}(P_{-D(x)})$ or $\Phi_{C_j}(P_{-D(x)})$ of Definitions 2.6 and 2.7 as a children node of P, then P'_j is an I-formula. As the variables that occur in P are bound, by construction of its children, the variables occurring in the heart of P'_j are bound too. Now, due to the definitions of the term distributing trees and the *Ind* and *Struct* rules, it is easy to see that there is a child node θ_j of θ such that $C(P'_j) = \theta'_j$. Therefore, the above process allows the property *ii*) to be held by each child of $(\Gamma \vdash P)$ except if the corresponding node in T is a leaf. By definition of $\mathcal{A}$, $\mathcal{C}'(\mathcal{A}) = (T, \mu)$ and its uniqueness results from injectivity of $\mathcal{C}'$. This gives the associated tree $\mathcal{A} = \mathcal{D}(T)$ of T with $\mathcal{C}'(\mathcal{D}(T, \mu)) = (T, \mu)$. Hence we deduce, because $\mathcal{C}'$ is injective, that $\mathcal{D}(\mathcal{C}'(\mathcal{A})) = \mathcal{A}$ for each distributing tree. □

This means $\mathcal{D}(\mathcal{H}'(\mathcal{A})) = \mathcal{A}$ and $\mathcal{H}'(\mathcal{D}(T, \mu)) = (T, \mu)$ for any distributing tree $\mathcal{A}$ and term distributing tree (T, μ). $\mathcal{D}$ can be illustrated with Figure 2.

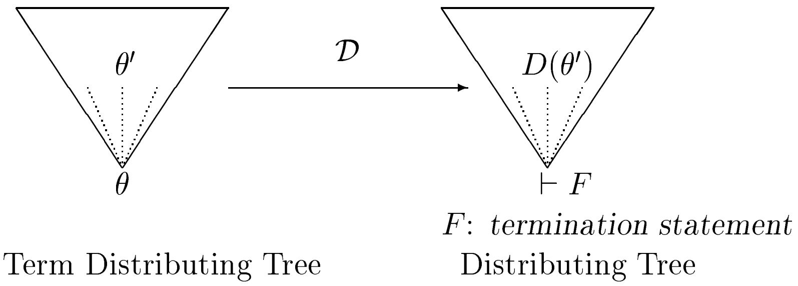


Fig. 2. The reverse operator of $\mathcal{H}'$

There is still no warranty on the termination of functions using μ-term distributing trees. But the *atsp* of μ-term distributing trees stands for the *ftsp* from which all proposition informations are abstracted in a simpler context:

Theorem 4.1 *Let $\mathcal{E}_f$ be a specification of a function f and $\mathcal{A}$ be a distributing tree for $\mathcal{E}_f$. If $\mathcal{A}$ has the formal terminal state property then the term distributing tree $\mathcal{H}'(\mathcal{A})$ has the abstract terminal state property.*

Proof. Similar to the proof of Theorem 4.2 below. □

Definition 4.2. $[N_r(Q, P)]$ Let P be an I-formula and Q a restrictive hypothesis of P. $N_r(Q, P)$ is the number of restrictive hypotheses of P that appear between the outermost restrictive hypothesis of P. E.g., if Q is the outermost restrictive hypothesis of P, then $N_r(Q, P) = 1$. $N_i(P)$ is the number of restrictive hypothesis of P.

Definition 4.3. $[Tr_b^{j,k}(Q)]$ Let $\mathcal{A}$ be a distributing tree for a specification $\mathcal{E}_f$. Let b be a branch and P a node in b at a level i from the root. We define $Tr_b^{i+1,i}(Q)$, where Q is a restrictive hypothesis of P, as the restrictive hypothesis Q' in the child P' of P in b as follows depending on whether the rule applied on P is:

- *Struct*: Q' is the restrictive hypothesis where $N_r(Q', P') = N_r(Q, P)$.
- *Ind*: Q' is such that $N_r(Q', P') = N_r(Q, P) + 1$.

We also define $Tr_b^{j,k}(Q)$ with $j > k$ as the restrictive hypothesis of the node P'' at level j in b defined by: $Tr_b^{j,k}(Q) = Tr_b^{j,j-1} \circ \ldots \circ Tr_b^{k+2,k+1} \circ Tr_b^{k+1,k}(Q)$. Finally $Tr_b^{i,i}$ will denote the identity on P.

The next theorem is the opposite of Theorem 4.1 and shows that we can automatically rebuild a distributing tree that has the *ftsp* from a *skeleton form* that has the *atsp*. As a consequence, according to Section 2.3, we can also build an I-proof and thus extract a λ-term that computes the given function.

Theorem 4.2 *Let $\mathcal{E}_f$ be a specification of a function f and (T, μ) be a μ-term distributing tree for $\mathcal{E}_f$. If (T, μ) has the abstract terminal state property then the distributing tree $\mathcal{D}(T, \mu)$ has the formal terminal state property.*

Proof. Let (T, μ) be a term distributing tree for $\mathcal{E}_f$ which has the *Atsp*. We want to show that $\mathcal{D}(T, \mu)$ has the *ftsp*. Take a recursive call (t, v) of an equation of $\mathcal{E}_f$. We have to find a restrictive hypothesis $R = \forall z D z_{\prec s}, F_1, \ldots, F_k \to D(w)$ in L of $\mathcal{D}(T, \mu)$ with $\mathcal{B}(L) = (t, v)$, where $\mathcal{B}$ is the application of Definition 2.10, such that clauses 1. and 2. of Definition 2.14 hold. Let B be the corresponding branch in $\mathcal{D}(T, \mu)$ of $b(t)$ in T, and let (θ, x) be the node in $b(t)$ given in Definition 3.5. Consider $(\Gamma \vdash P)$ in $\mathcal{D}(T, \mu)$ that is at the same level of (θ, x) in T. As $\mu(\theta) = 1$, by construction of $\mathcal{D}(T, \mu)$, a new restrictive hypothesis of the form $Q = \forall z (D z_{\prec s} \to P_{-D(x)}[z/x])$ is created in the child P' of P in B. Consider $R = Tr_B^{j,i}(Q)$ the restrictive hypothesis in B where i and j are respectively the level of P' and the leaf of B. We can write $R = \forall z (D z_{\prec s'} \to P_{-D(x)}[z/x])$ for some term s' because:

1) The free variables in Q are those of the term s, and the applied $Ind/Struct$ rule is done on a variable in P' which is out of the scope of Q.
2) As 1) first holds for $Q' = Tr_B^{i+1,i}(Q)$, next holds for $Tr_B^{i+2,i}(Q) = Tr_B^{i+2,i+1}(Q')$, $\ldots$, we have that: $R = Tr_B^{j,i}(Q) = \forall z (D z_{\prec s'} \to P_{-D(x)}[z/x])$ where the variables of $C(R)$ are closed in R.

Clause 1 We know that θ matches v with a substitution $\rho_{\theta,v}$, but $C(P) = \theta$, so R can be applied to v according to a substitution σ defined with $\sigma(z) = \rho_{\theta,v}(x)$ and $\sigma(y) = \rho_{\theta,v}(y)$ for $y \neq z$. We have to show that $\sigma(z) \prec s'$. This can be easily proven, by induction on $k \geq i$, that if $Tr_B^{k,i}(Q) = \forall z (D z_{\prec s_k} \to P_{-D(x)}[z/x])$ for some term s_k, then $s_k = \sigma_{k,i-1}(x)$ where the node θ matches the node at level k in T with the substitution $\sigma_{k,i-1}$. By definition of j, $\sigma_{j,i-1} = \sigma_{L_B,\theta}$, so $\rho_{\theta,v}(x) \prec \sigma_{j,i-1}(x)$ by Definition 3.5, and we can now deduce that $\sigma(z) \prec s'$ since $s' = s_j$. Therefore clause 1. of Definition 2.14 holds.

Clause 2 Consider a restrictive hypothesis $H = \forall z D' z_{\prec r} \to K$ in R; we have to find a restrictive hypothesis H_0 in P of the form $\forall z D' z_{\prec r_0} \to K$ such that $\sigma(r) \preceq r_0$. As H is a restrictive hypothesis of $Tr_B^{j,i}(Q)$, H is also a restrictive hypothesis of Q. Hence, one associate to H a restrictive hypothesis H' in $P' = \forall x_{i_1} D_{i_1}(x_{i_1}), \ldots, \forall x_{i_k} D_{i_k}(x_{i_k}), \underbrace{\forall z (D z_{\prec s_i} \to P_{-D(x)}[z/x])}_{Q} \to P_{-D(x)}[s_i/x]$, where H and H' respectively appear in $P_{-D(x)}[z/x]$ and $P_{-D(x)}[s_i/x]$. As H is of the form $\forall z D' z_{\prec r} \to K$ then H' is of the form $\forall z D' z_{\prec r_0} \to K$ since only the variables in the term r are free in H. Now consider the node $(\Gamma \vdash N)$ in B at a level l such that 1) a new restrictive hypothesis M is created in the child N' of N in B, namely, $N_i(N') = N_i(N) + 1$ and $N_r(M, N') = 1$, and 2) $Tr_B^{i,l}(M) = H'$. Let (θ', x') be the corresponding node in T of $(\Gamma \vdash N)$ in A. It is clear that θ' is an ancestor of θ in T since $l < j$ in $\mathcal{D}(T, \mu)$. Furthermore as $N_i(N') = N_i(N)+1$, we have $\mu(\theta') = 1$. By Definition 3.5 we have the relation $\rho_{\theta',v}(x') \preceq \sigma_{L_{b(t)},\theta'}(x')$. Let us now choose $H_0 = Tr_B^{j,l+1}(M)$ as the restrictive hypothesis in P'. Using the same property of clause 1 as we did with $Tr_B^{j,i}(Q)$, we know that r_0 is $\sigma_{j,l}(x') = \sigma_{L_{b(t)},\theta'}(x')$. Let us show that $\sigma(r) = \sigma_{\theta',v}(x')$. We note that $i-1 \geq l+1$ since $i-1$ and l are respectively the level of P and N that are distinct. We have $Tr_B^{i-1,l+1}(M) = \forall z (D' z_{\prec \sigma_{i-1,l}(x')} \to K)$ in P, where $\sigma_{i-1,l}$ is by definition the substitution $\sigma_{\theta,\theta'}$. So, according to the restrictive hypothesis Q in P', the term r in H is $\sigma_{\theta,\theta'}(x')[z/x]$. Now, by definition of σ in clause 1 of Definition 2.14, we have $\sigma(r) = \rho_{\theta,v\ \{z \to x\}}(\sigma_{\theta,\theta'}(x')[z/x]) = \rho_{\theta,v}(\sigma_{\theta,\theta'}(x'))$. But the relation of substitutions gives us $\rho_{\theta',v} = \rho_{\theta,v} \circ \sigma_{\theta,\theta'}$. So we finally obtain $\sigma(r) = \rho_{\theta',v}(x')$, and we can deduce from the above and Definition 3.5 that $\sigma(r) \preceq r_0$. Hence, clause 2. of Definition 2.14 holds. $\square$

In [5], measures were related to given functions whose decreasing property through the recursive calls were dependent on the *ftsp* enjoyed by distributing trees. We claim that it is possible to infer measures directly from term distributing trees whose decreasing property through the recursive calls of the considered functions now rely only on *atsp*. This is a straightforward consequence of the results of this section with the previous one and [5].

Following distributing tree with *atsp* makes the analysis of the I-proofs easier. In particular there are no measures from [5] associated to the *quot* function (cf. Section 5) that have the decreasing property (see [4]). By the results of this section, there are no I-proofs for such function. The next section shows that the framework of *ProPre* can be applied to new functions (e.g. *quot* function) provided an automated termination procedure (e.g. [4,1,2]) is used.

5 Synthesizing programs from termination techniques

As noted in Section 2, if we can prove, in *TTR*, a *formula* that states the totality of a function then it is possible, in term of programs, to obtain a λ-term as the code of the function. As earlier mentioned, this formula is called *termination statement* in *ProPre* (Definition 2.2). More precisely, assume that $\mathcal{E}_{f_1}, \ldots, \mathcal{E}_{f_m}$ are specifications of defined functions already proven in the

ProPre system. Let f be a new defined function with a specification $\mathcal{E}_f$. We put $\mathcal{E} = \sqcup_{j=1}^{j=n} \mathcal{E}_{f_j}$, and $\mathcal{E}_f^1 = \mathcal{E}_f \sqcup \mathcal{E}$. In order to obtain a lambda term F that computes the new function f, *ProPre* needs to establish $\vdash_{\mathcal{E}_f^1} F : T_f$ in *TTR*.

Example 5.1. Let $quot : nat, nat, nat \to nat$ be a defined function with specification $\mathcal{E}_{quot}$ given by the equations:

$$quot(x,0,0) = 0 \qquad quot(s(x), s(y), z) = quot(x,y,z)$$
$$quot(0, s(y), z) = 0 \qquad quot(x, 0, s(z)) = s(quot(x, s(z), s(z))$$

The value $quot(x,y,z)$ 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 $\lfloor \frac{x}{y} \rfloor$. Its specification does not admit an I-proof and therefore no λ-term can be associated by the *ProPre* system.

To avoid this drawback, we show, considering the framework of *ProPre* and *TTR*, that it is possible to add other automated termination procedures than that of *ProPre* regarding the automation of the extraction of λ-terms.

When *ProPre* builds a formal proof of a specification, it needs to check at different steps that some subterm in one argument of the equations *decreases* in the recursive calls according to the relation of Definition 2.13. These informations are given by a termination algorithm in *ProPre*. I.e., to convey the termination informations in the formal proof in *ProPre*, it is used with the relation $\prec$ included in formulas of the form $A[u/x] \upharpoonright (u \prec v)$ due to Table 2.

Now assume, for a given function that terminates, the equations admit only one argument. This provides a natural (partial) relation on the data type on which the function is specified so that each recursive call decreases. Also assume that an automated procedure ensures the termination of this function. Then this one can be used as the termination algorithm of *ProPre*, but we now consider the new relation instead of the earlier relation $\prec$ of *ProPre*. Due to the hiding rules of the operator $\upharpoonright$ we can develop a particular formal proof, as an I-Proof, for the considered function but where in particular the sequent $\Gamma \vdash_{\mathcal{E}} (u \prec v)$ in the rule $(\upharpoonright_1)$ with $e = (u \prec v)$ can be obtained with the new termination procedure that provides the new relation $\prec$.

If the function admits many arguments, we would like to cluster the arguments of the equations of the specification into one argument. To do so, we show that the use of uncurryfication forms of functions is harmless in *TTR* (and in $AF2$) in the sense of Lemma 5.4 by considering the product types. This enables us to follow the principle of Figure 3 where $\tilde{f}$ stands for an uncurryfication form of f. The left part of Figure 3 is obtained with Theorem 5.1.

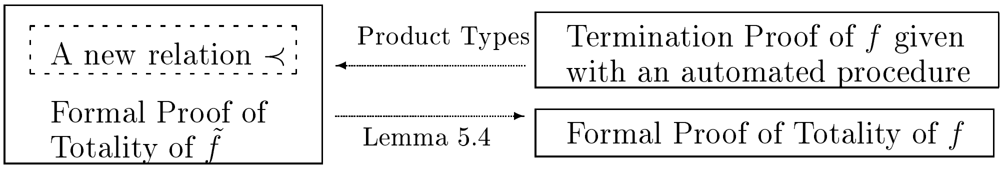


Fig. 3. A formal proof of totality of the function f.

5.1 Product types

We introduce particular specifications that correspond in some sense to uncurryfication forms of previous specifications. To do so, we will consider a product type associated to a function. As we have not stated the data types of *TTR* with the operator μ (cf. beginning of Section 2.2), for the sake of presentation, we present below the product types in the context of *AF2*. This presentation in Definition 5.2 is harmless because Lemma 5.4 below and its proof hold both in *AF2* and *TTR*.

Definition 5.2. [**Product type of a function**] Let $f : D_1, \ldots, D_n \to D$ be a defined function, $cp \in \mathcal{F}_c$ be a new constructor of arity n and take $T_f = \forall x_1 \ldots \forall x_n (D_1(x_1), \ldots, D_n(x_n) \to D(f(x_1, \ldots, x_n)))$ to be the termination statement of f. The data type $K(x)$ defined by the formula: $\forall X\ \forall y_1 \ldots \forall y_n\ D_1(y_1), \ldots, D_n(y_n) \to X(cp(y_1, \ldots, y_n)) \to X(x)$ is called the product type of $D_1, \ldots, D_n$, and is denoted by $(D_1 \times \ldots \times D_n)(x)$.

From the specification of a defined function f, we can get another defined function $\tilde{f}$ whose specification $\mathcal{E}_{\tilde{f}}$ takes into account the product type of f.

Definition 5.3. Let $f : D_1, \ldots, D_n \to D$ be a defined function with specification $\mathcal{E}_f$. Let $\tilde{f}$, the *twin function* of f, be a new defined symbol in $\mathcal{F}_d$. To define the specification $\mathcal{E}_{\tilde{f}}$ of $\tilde{f}$, we define each equation $\tilde{f}(cp(t_1, \ldots, t_n)) = \overline{v}$ of $\mathcal{E}_{\tilde{f}}$ from each equation $f(t_1, \ldots, t_n) = v$ of $\mathcal{E}_f$ where cp is the constructor symbol of the product type of f. The term $\overline{v}$ is recursively defined by:

- (i) if v is a variable or a constant then $\overline{v} = v$,
- (ii) if $v = g(u_1, \ldots, u_m)$ with g a constructor or a symbol function distinct from f, then $\overline{v} = g(\overline{u_1}, \ldots, \overline{u_m})$,
- (iii) if $v = f(u_1, \ldots, u_n)$ then $\overline{v} = \tilde{f}(cp(\overline{u_1}, \ldots, \overline{u_n}))$.

This defines the specification $\mathcal{E}_{\tilde{f}}$ of the defined function $\tilde{f}$ associated to f. The termination statement of $\tilde{f}$ is: $T_{\tilde{f}} = \forall x((D_1 \times \ldots \times D_n)(x) \to D(\tilde{f}(x)))$.

Let us consider the specification $\mathcal{E}_f$ of a function and the set of equations $\mathcal{E}'_f = \mathcal{E}_f \cup \{f(x_1, \ldots, x_n) = \tilde{f}(cp(x_1, \ldots, x_n))\}$. The set $\mathcal{E}'_f$ is not a specification according to Definition 2.2 in *ProPre*, but we can still reason in *TTR*. Assume the termination statement of $\tilde{f}$ proven in *TTR* with $\mathcal{E}_{\tilde{f}}$ and the set $\mathcal{E}$ of the specifications already proven. Now we can add the equations of $\mathcal{E}_{\tilde{f}}$ in the set $\mathcal{E}$ before proving the termination statement T_f. Due to the form of the specifications $\mathcal{E}_{\tilde{f}}$ and $\mathcal{E}_f$, the equation $f(x_1, \ldots, x_n) = \tilde{f}(cp(x_1, \ldots, x_n))$ does not add any contradiction in the set of the equational axioms $\mathcal{E}_f \sqcup \mathcal{E}$. Therefore we can now use the new set $\mathcal{E}'_f \sqcup \mathcal{E}$ to prove the termination statement T_f in *TTR*. So, the equation $f(x_1, \ldots, x_n) = \tilde{f}(cp(x_1, \ldots, x_n))$ provides the connection between $\mathcal{E}_f$ and $\mathcal{E}_{\tilde{f}}$ from the logical point of view and the proof of $T_{\tilde{f}}$ provides the computational aspect of the function f. More precisely:

Lemma 5.4. Let $f : D_1, \ldots, D_n \to D$ be a defined function with a specification $\mathcal{E}_f$, and $\mathcal{E}_{\tilde{f}}$ the specification of the twin function $\tilde{f}$. Let $\mathcal{E}_1, \ldots, \mathcal{E}_n$ be the specifications of the defined functions already proven (in *AF2* or *TTR*), $\mathcal{E} = \sqcup_{i=1}^{i=n} \mathcal{E}_i$. Let us note $\mathcal{E}^1_{\tilde{f}} = \mathcal{E}_{\tilde{f}} \sqcup \mathcal{E}$ and $\mathcal{E}^2_{\tilde{f}} = \mathcal{E}'_{\tilde{f}} \sqcup \mathcal{E}^1_{\tilde{f}}$ with $\mathcal{E}'_f = \mathcal{E}_f \cup \{f(x_1, \ldots, x_n) = \tilde{f}(cp(x_1, \ldots, x_n))\}$. If there is a λ-term $\widetilde{F}$ such that $\vdash_{\mathcal{E}^1_{\tilde{f}}} \widetilde{F} : T_{\tilde{f}}$, then there is a λ-term F such that $\vdash_{\mathcal{E}^2_{\tilde{f}}} F : T_f$.

Proof. This lemma holds both in *AF2* and *TTR*, (using the rules in Table 1). We assume familiarity with *AF2* and only give steps without naming the rules. Let $K = (D_1 \times \ldots \times D_n)$ be the product type of f with cp the associated constructor symbol. By definition of the data type K, we get in *TTR*:
$a_1 : D_1(x_1), \ldots, a_n : D_n(x_n) \vdash \lambda k(\ldots((k\ a_1)\ a_2) \ldots\ a_n) : K(cp(x_1, \ldots, x_n))$. Hence:
$a_1 : D_1(x_1), \ldots, a_n : D_n(x_n) \vdash_{\mathcal{E}^1_{\tilde{f}}} (\widetilde{F}\ \lambda k(\ldots((k\ a_1)\ a_2) \ldots\ a_n)) : D(\tilde{f}(cp(x_1, \ldots, x_n)))$.
Because $\mathcal{E}^1_{\tilde{f}} \subset \mathcal{E}^2_{\tilde{f}}$ we have:
$a_1 : D_1(x_1), \ldots, a_n : D_n(x_n) \vdash_{\mathcal{E}^2_{\tilde{f}}} (\widetilde{F}\ \lambda k(\ldots((k\ a_1)\ a_2) \ldots\ a_n)) : D(\tilde{f}(cp(x_1, \ldots, x_n)))$.
Now, we have the equation $f(t_1, \ldots, t_m) = \tilde{f}(cp(t_1, \ldots, t_m))$ in $\mathcal{E}^2_{\tilde{f}}$. Hence:
$a_1 : D_1(x_1), \ldots, a_n : D_n(x_n) \vdash_{\mathcal{E}^2_{\tilde{f}}} (\widetilde{F}\ \lambda k(\ldots((k\ a_1)\ a_2) \ldots\ a_n)) : D(f(x_1, \ldots, x_n))$.
Finally: $\vdash_{\mathcal{E}^2_{\tilde{f}}} F : T_f$, with $F = \lambda a_1 \ldots \lambda a_n(\widetilde{F}\ \lambda k(\ldots((k\ a_1)\ a_2) \ldots\ a_n))$. □

We show next that the specification of the twin of a function admits a particular I-proof assuming its termination is proven with an automated procedure.

5.2 *Canonical I-proofs*

Let $f : D_1, \ldots, D_n \to D$ be a defined function, with a specification $\mathcal{E}_f$, which is terminating with an automated procedure. As mentioned earlier, instead of using the ordering of the terms given in Definition 2.13, we define a new ordering for the symbol relation $\prec$ by considering the ordering given with the recursive calls of the equations of the specification $\mathcal{E}_{\tilde{f}}$. As in the *ProPre* system, we will assume that we have a subset $\mathcal{F}^\star_d$ of $\mathcal{F}_d$ of defined functions whose specification admits a proof of totality in *TTR* (the functions already introduced by the user) so that the defined functions occurring in the specification of f for which we want to prove the termination statement, are in $\mathcal{F}^\star_d \cup \{f\}$.

Now, let t be a term in $\mathcal{T}(\mathcal{F}, \mathcal{X})_{s'}$, for some sort s' (see Section 3), such that all the defined functions occurring in t admit a specification and are terminating. Then, for each ground sorted substitution σ, we can define the ground term $\ulcorner\!\ulcorner \sigma(t) \urcorner\!\urcorner$ as the term in $\mathcal{T}(\mathcal{F}_c)_s$ that corresponds to the normal form of $\sigma(t)$. The definition of $\ulcorner\!\ulcorner \sigma(t) \urcorner\!\urcorner$ makes sense as the functions occurring in the specification f are terminating which gives the existence of the normal form while the definition of the specifications (Definition 2.2) gives the uniqueness of the normal form. Therefore, we can state formally the relation $\prec_{\tilde{f}}$ below.

Definition 5.5. Let $\mathcal{E}_{\tilde{f}}$ be a specification of the twin function of a defined and terminating function f such that the functions occurring in the specification $\mathcal{E}_{\tilde{f}}$ admit a specification and are terminating. Let K be the product type $(D_1 \times \ldots \times D_n)$ associated to f and cp the constructor associated to K. We define a relation $\prec_{\tilde{f}}$ on K such that for each recursive call of $\mathcal{E}_{\tilde{f}}$, $(f(cp(t_1, ..., t_n)), f(cp(v_1, \ldots, v_n)))$, we have $cp(\ulcorner\!\ulcorner \sigma(v_1) \urcorner\!\urcorner, \ldots, \ulcorner\!\ulcorner \sigma(v_n) \urcorner\!\urcorner) \prec_{\tilde{f}} \sigma(cp(t_1, \ldots, t_n))$ for any ground sorted substitution σ.

Hence, we get the straightforward but useful following fact.

Fact 5.6. The above relation $\prec_{\tilde{f}}$ is a well-founded ordering on K.

Theorem 5.1 says that if a function f is terminating and if we have a distributing tree for the specification $\mathcal{E}_{\tilde{f}}$ of its twin function, having or lacking the formal terminal state property, it is then possible to get a new one having the *ftsp*. The idea consists of changing, in the initial distributing tree, the $Struct$ and Ind rules in such way that we now have a new tree with *ftsp* which can be called a canonical distributing tree. Hence, the formal proofs we are going to build will depend on the abilities of • building a distributing tree whatever its properties, and • showing the termination of the function.

Theorem 5.1 *Let $\mathcal{E}_f$ be a specification of a defined function $f : D_1, \ldots, D_n \to D$ such that the defined symbols that occur on the right-hand side of the equations of $\mathcal{E}_f$ are in $\mathcal{F}_d^\star \cup \{f\}$. Let $\mathcal{A}$ be a distributing tree for the specification $\mathcal{E}_{\tilde{f}}$ of the twin function $\tilde{f}$. Assume the function f is proven terminating by a termination procedure. Then there is a distributing tree $\mathcal{A}'$ for $\mathcal{E}_{\tilde{f}}$, which can be automatically obtained from $\mathcal{A}$, that satisfies the formal terminal state property with the relation $\prec_{\tilde{f}}$.*

Proof. Let $\mathcal{E}_f$ be a specification of a defined function $f : D_1, \ldots, D_n \to s$ such that the defined symbols that occur in the right-hand side of the equations of $\mathcal{E}_f$ are in $\mathcal{F}_d^\star \cup \{f\}$. Let $\mathcal{A}$ be a distributing tree for the specification $\mathcal{E}_{\tilde{f}}$ of the twin function $\tilde{f}$. We assume f is proven terminating by a termination procedure. Since we know that the function is terminating given by an automated procedure we can introduce the ordering $\prec_{\tilde{f}}$. From the term distributing tree $\mathcal{A}$ we can associate a new distributing tree $\mathcal{A}'$ with the ordering $\prec_{\tilde{f}}$, illustrated with Figure 4, which can be called the canonical distributing tree of $\mathcal{A}$.

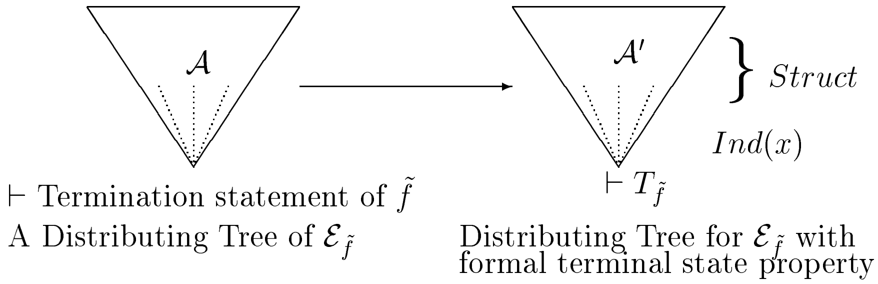


Fig. 4. The canonical distributing tree $\mathcal{A}'$ of $\mathcal{A}$

Note that $\mathcal{A}'$ can be built automatically from $\mathcal{A}$. We show that $\mathcal{A}'$ satisfies the formal terminal state property. The root of $\mathcal{A}'$ is $\vdash T_{\tilde{f}}$, with $T_{\tilde{f}} = \forall x(K(x) \to D(\tilde{f}(x)))$ the termination statement of $\tilde{f}$ where K denotes the product type $(D_1 \times \ldots \times D_n)$ and cp its associated constructor.

Let $L = (\Gamma \vdash P)$ be a leaf of $\mathcal{A}'$ and $e = (t, u)$ be the equation of $\mathcal{E}_{\tilde{f}}$ with $H(P) = t$. Let (t, v) be a recursive call of e. According to the definition of a specification and a recursive call, the terms t and v are respectively of the form $f(cp(t_1, \ldots, t_n))$ and $f(cp(v_1, \ldots, v_n))$. Because of the construction of the canonical distributing tree $\mathcal{A}'$ that uses a particular order of the application

rules *Struct* and *Ind* (also illustrated with Figure 4), P is of the form:
$\forall x'_{i_1} D'_{i_1}(x'_{i_1}), \ldots, \forall x'_{i_m} D'_{i_m}(x'_{i_m}), \forall z(Kz_{\prec cp(h_1,\ldots,h_n)} \to K(f(z))) \to K(f(cp(h_1, \ldots, h_n)))$.

As the heart of P is $H(P) = t$, we have $h_j = t_j$ for any $1 \leq j \leq n$. Now, let Q be the restrictive hypothesis $\forall z(Kz_{\prec cp(t_1,\ldots,t_n)} \to K(f(z)))$ of P. Let us show that Q can be applied to the term v according to a substitution. By the definition of Q, we have $H(Q) = f(z)$, so we can take a substitution σ with $\sigma(z) = cp(v_1, \ldots, v_n)$. We also take the value $\sigma(y) = y$ for any free variable y in Q, that is any variable y in $cp(t_1, \ldots, t_n)$. Hence Q can be applied to v according to the above substitution σ. We now have to show the two items of Definition 2.14. As we are in the conditions of Definition 5.5, we know that $cp(\ulcorner\!\ulcorner \rho(v_1) \urcorner\!\urcorner, \ldots, \ulcorner\!\ulcorner \rho(v_n) \urcorner\!\urcorner) \prec_{\tilde{f}} \rho(cp(t_1, \ldots, t_n))$ for any ground sorted substitution ρ. But $\sigma(z) = cp(v_1, \ldots, v_n)$, thus we get the first item. The second item becomes straightforward: because of the form of Q, the set of restrictive hypotheses of Q is empty. Hence, we conclude that the canonical distributing tree $\mathcal{A}'$ satisfies the formal terminal state property. □

The next theorem (and its proof) expresses Figure 3. It tells that if we know that a function f is terminating, and if we have already a proof of totality of each defined function that occurs in the specification of f (apart from f), and if we have a term distributing tree associated to the specification of f, then we are able to get a λ-term that computes f in the sense of *TTR*.

Theorem 5.2 *Let $\mathcal{E}_f$ be a specification of a defined function $f : D_1, \ldots, D_n \to D$ and $\mathcal{D}$ be a given distributing tree for the specification $\mathcal{E}_f$ such that the defined symbols that occur on the right-hand side of the equations of $\mathcal{E}_f$ are in $\mathcal{F}_d^\star \cup \{f\}$. Assume the termination of f is given by an automated procedure. Then there is a proof of totality of f in TTR that can be found automatically.*

Proof. Let $\tilde{f}$ be the twin function of f and $\mathcal{E}_{\tilde{f}}$ its specification given in Definition 5.3. By Definition 5.3, a distributing tree $\mathcal{A}$ associated to $\mathcal{E}_{\tilde{f}}$ can be automatically obtained from $\mathcal{D}$. Hence, with Theorem 5.1, we now have a (canonical) distributing tree $\mathcal{A}'$ associated to $\mathcal{E}_{\tilde{f}}$ which has the *ftsp* with $\prec_{\tilde{f}}$ as the ordering relation. As Fact 2.11 still holds with the new ordering relation, we get an I-proof of $\mathcal{E}_{\tilde{f}}$ that can be called *canonical proof*. Thus we obtain a formal proof of the termination statement $T_{\tilde{f}}$ in *TTR*. Hence, by Lemma 5.4 we finally obtain a proof of totality of f in *TTR*. □

6 Conclusion

The programming paradigms using logics built in *ProPre* uses the Curry-Howard isomorphism where a λ-term is extracted from the proof. However because of the logical framework, it is often difficult to make use of termination techniques from different areas. This paper showed that for the automated system *ProPre*, the extraction part of λ-terms can be released from the termination analysis, so that other automated termination techniques (like those of [1,2,4]) can now be included in this framework modulo distributing trees.

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 *LNCS*, pages 261–272, 1997.

[2] J. Giesl. Termination of nested and mutually recursive algorithms. *J. of Automated Reasoning*, 19:1–29, 1997.

[3] W. A. Howard. The formulæ-as types notion of construction. In J. Hindley and J. Seldin, editors, *To H.B. Curry: Essays on combinatory logic, lambda-calculus and formalism*, pages 479–490. Academic Press, 1980.

[4] F. Kamareddine and F. Monin. On automating inductive and non-inductive termination methods. In *Proceedings of the 5th Asian Computing Science Conference*, volume 1742 of *LNCS*, pages 177–189, 1999.

[5] F. Kamareddine and F. Monin. On formalised proofs of termination of recursive functions. In *Proceedings of the Int. Conf. on Principles and Practice of Declarative Programming*, volume 1702 of *LNCS*, pages 29–46, 1999.

[6] J. L. Krivine. *Lambda-calculus, Types and Models.* Computers and Their Applications. Ellis Horwood, 1993.

[7] J. L. Krivine and M. Parigot. Programming with proofs. *J. Inf. Process Cybern*, 26(3):149–167, 1990.

[8] D. Leivant. Typing and computational properties of lambda expression. *Theoretical Computer Science*, 44:51–68, 1986.

[9] 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 *LNCS*, pages 83–100, 1994.

[10] P. Manoury, M. Parigot, and M. Simonot. ProPre, a programming language with proofs. In *Proceedings of Logic Programming and Automated Reasoning*, volume 624 of *LNCS*, pages 484–486, 1992.

[11] P. Manoury and M. Simonot. *Des preuves de totalité de fonctions comme synthèse de programmes.* PhD thesis, University Paris 7, 1992.

[12] P. Manoury and M. Simonot. Automatizing termination proofs of recursively defined functions. *Theoretical Computer Science*, 135(2):319–343, 1994.

[13] F. Monin and M. Simonot. An ordinal measure based procedure for termination of functions. *Theoretical Computer Science*, 254(1-2):63–94, 2001.

[14] M. Parigot. Recursive programming with proofs: a second type theory. In *Proceedings of the European Symposium on Programming ESOP'88*, volume 300 of *LNCS*, pages 145–159, 1988.

[15] M. Parigot. Recursive programming with proofs. *Theoretical Computer Science*, 94(2):335–356, 1992.