On automating the extraction of programs from proofs using product types

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

François Monin
IRISA
Campus de Beaulieu
Rennes, France

Mauricio Ayala-Rincón
Departamento de Matemática
Universidade de Brasília
Brasília D.F., Brasil

Introduction

- We are interested in programming language with the point of view: *Proofs as Programs* (Curry-Howard correspondence).

Introduction

- We are interested in programming language with the point of view: *Proofs as Programs* (Curry-Howard correspondence).
- The specifications are the types and the lambda-terms are the extracted programs (the code).

Introduction

- We are interested in programming language with the point of view: *Proofs as Programs* (Curry-Howard correspondence).
- The specifications are the types and the lambda-terms are the extracted programs (the code).
- The verification of the types (compilation) is a proof of program.

Introduction

- We are interested in programming language with the point of view: *Proofs as Programs* (Curry-Howard correspondence).
- The specifications are the types and the lambda-terms are the extracted programs (the code).
- The verification of the types (compilation) is a proof of program.
- The ProPre system was designed as a prototype to show the feasibility of the theory.

Motivation

- The difficulty is to find formal proofs automatically.

Motivation

- The difficulty is to find formal proofs automatically.
- Example:

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

The term $quot(x, y, y)$ computes $\lfloor \frac{x}{y} \rfloor$.

Motivation

- The difficulty is to find formal proofs automatically.
- Example:

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

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

The term $quot(x, y, y)$ computes $\lfloor \frac{x}{y} \rfloor$.

- The proofs are expressed in natural deduction style.

Motivation

- The difficulty is to find formal proofs automatically.
- Example:

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

The term $quot(x,y,y)$ computes $\lfloor \frac{x}{y} \rfloor$.

- The proofs are expressed in natural deduction style.
- The automated termination proofs $\neq$ techniques of rewriting systems.

Motivation

- Is it possible to go further than the ProPre system but using the same theory ?

Motivation

- Is it possible to go further than the ProPre system but using the same theory ?
- We analyse the proofs made in the system.

Motivation

- Is it possible to go further than the ProPre system but using the same theory ?
- We analyse the proofs made in the system.
- We then develop some particular formal proofs using product types.

Motivation

- Is it possible to go further than the ProPre system but using the same theory ?
- We analyse the proofs made in the system.
- We then develop some particular formal proofs using product types.
- The formal proofs are released from the termination part.

Motivation

- Is it possible to go further than the ProPre system but using the same theory ?
- We analyse the proofs made in the system.
- We then develop some particular formal proofs using product types.
- The formal proofs are released from the termination part.
- This allows automated termination proofs to be incorporated while lambda-terms are still extracted from the proofs.

Motivation

- Is it possible to go further than the ProPre system but using the same theory ?
- We analyse the proofs made in the system.
- We then develop some particular formal proofs using product types.
- The formal proofs are released from the termination part.
- This allows automated termination proofs to be incorporated while lambda-terms are still extracted from the proofs.
- The class of automated extracted programs are thus enlarged.

Overview

- The ProPre system
- Logical framework: AF2, TTR
- The rules and proofs in ProPre
- Analysis of the I-proofs
- The skeleton proofs
- The connection between skeleton proofs and I-proofs
- The product type
- The canonical proofs
- Conclusion

The ProPre system

- ProPre is a program synthesis system.

The ProPre system

- ProPre is a program synthesis system.
- Example:

The ProPre system

- ProPre is a program synthesis system.
- Example:
  - The type of the list of natural number in ProPre:

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

The ProPre system

- ProPre is a program synthesis system.
- Example:
  - The type of the list of natural number in ProPre:

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

  - The append function in ProPre:

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

The ProPre system

- ProPre is a program synthesis system.
- Example:
  - The type of the list of natural number in ProPre:

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

  - The append function in ProPre:

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

- The systems leads from a specification of a function to a program.

The ProPre system

- Functional programming language based on the paradigm:

  Programming by Proofs
  ("Proofs as Programs")

The ProPre system

- Functional programming language based on the paradigm:
  Programming by Proofs
  ("Proofs as Programs")
- Type System:
  program extraction $\Longrightarrow$ lambda-term

The ProPre system

- Functional programming language based on the paradigm:
  Programming by Proofs
  ("Proofs as Programs")
- Type System:
  program extraction $\Longrightarrow$ lambda-term
- Automated strategies for proving termination of recursive functions.

Logical Framework

- The type system is a Second Order Type with Lambda-Calculus: *Second Order Functional Arithmetic, AF2* (D. Leivant, J.L. Krivine).

Logical Framework

- The type system is a Second Order Type with Lambda-Calculus: *Second Order Functional Arithmetic, AF2* (D. Leivant, J.L. Krivine).
- Data types are multisorted terms algebras defined by formulas with one free variable.

Logical Framework

- The type system is a Second Order Type with Lambda-Calculus: *Second Order Functional Arithmetic, AF2* (D. Leivant, J.L. Krivine).
- Data types are multisorted terms algebras defined by formulas with one free variable.
  - The integers sort nat

    $0 :\longrightarrow nat,\ s : nat \longrightarrow nat$

Logical Framework

- The type system is a Second Order Type with Lambda-Calculus: *Second Order Functional Arithmetic, AF2* (D. Leivant, J.L. Krivine).
- Data types are multisorted terms algebras defined by formulas with one free variable.
  - The integers sort nat
    $0 :\to nat,\ s : nat \to nat$
  - The data type $N(x)$ of natural numbers:
    $\forall X(X(0) \to (\forall y(X(y) \to X(s(y))) \to X(x)))$

Logical Framework

- The type system is a Second Order Type with Lambda-Calculus: *Second Order Functional Arithmetic, AF2* (D. Leivant, J.L. Krivine).
- Data types are multisorted terms algebras defined by formulas with one free variable.
  - The integers sort nat
    $0 :\rightarrow nat,\ s : nat \rightarrow nat$
  - The data type $N(x)$ of natural numbers:
    $\forall X (X(0) \rightarrow (\forall y (X(y) \rightarrow X(s(y))) \rightarrow X(x)))$
- Logical Interpretation coincides with the Algorithmic Interpretation of the formula.

The Logical framework

- Lambda-terms correspond to the algorithmic content of the formulas.

The Logical framework

- Lambda-terms correspond to the algorithmic content of the formulas.
-

Data-Type : Formula of Second Order

$$\Downarrow$$

Programs for constructors (sucessor for integers, cons for lists, etc ...)

Intuitionistic rules

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

$$\frac{\Gamma,\ A \vdash\ B}{\Gamma \vdash\ A{\rightarrow}B} \quad ({\longrightarrow}_i) \qquad \frac{\Gamma \vdash\ A \qquad \Gamma \vdash\ A{\rightarrow}B}{\Gamma \vdash\ B} \quad ({\longrightarrow}_e)$$

$$\frac{\Gamma \vdash\ A}{\Gamma \vdash\ \forall y A} \quad (\forall^1_i) \qquad \frac{\Gamma \vdash\ A}{\Gamma \vdash\ \forall Y A} \quad (\forall^2_i)$$

$$\frac{\Gamma \vdash\ \forall y A}{\Gamma \vdash\ A[\tau/y]} \quad (\forall^1_e) \qquad \frac{\Gamma \vdash\ \forall Y A}{\Gamma \vdash\ A[T/Y]} \quad (\forall^2_e)$$

Second Order Functional Arithmetic

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

$$\frac{\Gamma,\ x{:}A \vdash t{:}B}{\Gamma \vdash \lambda x.t{:}A{\rightarrow}B}\ (\rightarrow_i) \qquad \frac{\Gamma \vdash u{:}A \qquad \Gamma \vdash t{:}A{\rightarrow}B}{\Gamma \vdash (t\ u){:}B}\ (\rightarrow_e)$$

$$\frac{\Gamma \vdash t{:}A}{\Gamma \vdash t{:}\forall y A}\ (\forall_i^1) \qquad \frac{\Gamma \vdash t{:}A}{\Gamma \vdash t{:}\forall Y A}\ (\forall_i^2)$$

$$\frac{\Gamma \vdash t{:}\forall y A}{\Gamma \vdash t{:}A[\tau/y]}\ (\forall_e^1) \qquad \frac{\Gamma \vdash t{:}\forall Y A}{\Gamma \vdash t{:}A[T/Y]}\ (\forall_e^2)$$

A main result in AF2

- A statement of a theorem:

A main result in AF2

- A statement of a theorem:
  - Assume $D_1, \ldots, D_n, D$ data types, f a function symbol, $\mathcal{E}_f$ a set of equations, t a lambda-term.

A main result in AF2

- A statement of a theorem:
  - Assume $D_1, \ldots, D_n, D$ data types, f a function symbol, $\mathcal{E}_f$ a set of equations, t a lambda-term.
  - If

$$\vdash_{\mathcal{E}_f} t : \forall x_1, \ldots, \forall x_n \{D_1[x_1], \ldots, D_n[x_n] \rightarrow D[f(x_1, \ldots, x_n)]\}$$

A main result in AF2

- A statement of a theorem:
  - Assume $D_1, \ldots, D_n, D$ data types, f a function symbol, $\mathcal{E}_f$ a set of equations, t a lambda-term.
  - If

$$\vdash_{\mathcal{E}_f} t : \forall x_1, \ldots, \forall x_n \{D_1[x_1], \ldots, D_n[x_n] \to D[f(x_1, \ldots, x_n)]\}$$

  - Then "t computes f"

A main result in AF2

- A statement of a theorem:
  - Assume $D_1, \ldots, D_n, D$ data types, f a function symbol, $\mathcal{E}_f$ a set of equations, t a lambda-term.
  - If

$$\vdash_{\mathcal{E}_f} t : \forall x_1, \ldots, \forall x_n \{D_1[x_1], \ldots, D_n[x_n] \to D[f(x_1, \ldots, x_n)]\}$$

  - Then "t computes f"
- Let $f : nat \to nat$. If $\vdash_{\mathcal{E}_f} t : \forall x(N(x) \to N[f(x)])$ then
  $\vdash_{\mathcal{E}_f} f(s^n(0)) = s^m(0)$ iff $(t\ \underline{n}) \to_\beta \underline{m}$

Recursive Type Theory

- TTR is an extension of AF2 (M. Parigot)

Recursive Type Theory

- TTR is an extension of AF2 (M. Parigot)
- Its aims is to allow more efficiency extracted programs.

Recursive Type Theory

- TTR is an extension of AF2 (M. Parigot)
- Its aims is to allow more efficiency extracted programs.
- It uses a logical operator of least fixed point allowing recursive definitions of data types.

Recursive Type Theory

- TTR is an extension of AF2 (M. Parigot)
- Its aims is to allow more efficiency extracted programs.
- It uses a logical operator of least fixed point allowing recursive definitions of data types.
- A logical hiding connective for hiding the algorithmic content of some part of the proofs.

Somes rules in TTR

- Rules of the hiding operator $\restriction$

  If A is a formula, u, v terms then $A \restriction (u \prec v)$ is a formula.

$$\frac{\Gamma \vdash_{\mathcal{E}} t{:}A \qquad \Gamma \vdash_{\mathcal{E}} e}{\Gamma \vdash_{\mathcal{E}} t{:}A{\restriction}e}\ (\restriction_1) \qquad \frac{\Gamma \vdash_{\mathcal{E}} t{:}A{\restriction}e}{\Gamma \vdash_{\mathcal{E}} t{:}A}\ (\restriction_2) \qquad \frac{\Gamma \vdash_{\mathcal{E}} t{:}A{\restriction}e}{\Gamma \vdash_{\mathcal{E}} e}\ (\restriction_3)$$

Somes rules in TTR

- Rules of the hiding operator $\upharpoonright$

  If A is a formula, u, v terms then $A \upharpoonright (u \prec v)$ is a formula.

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

- External induction rule

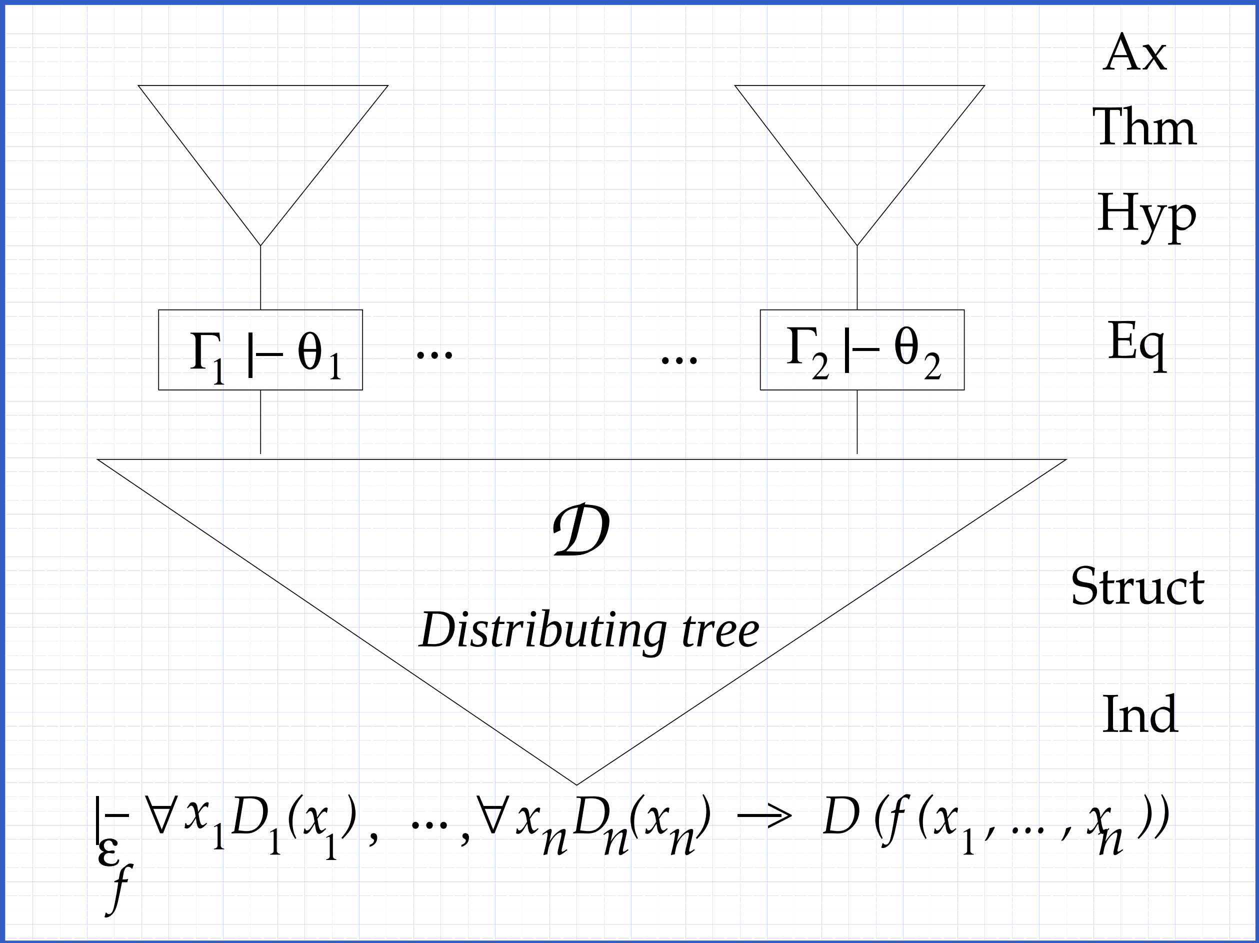
$$\frac{\Gamma \vdash_{\mathcal{E}} t : \forall x[\forall z[Dz_{\prec x} \rightarrow B[z/x]] \rightarrow [D(x) \rightarrow B]]}{\Gamma \vdash_{\mathcal{E}} (T\ t) : \forall x[D(x) \rightarrow B]}\ (Ext)$$

  T is a turing fixed-point operator,
  The relation $\prec$ is a well founded partial ordering on the terms of the algebra.

Macro Rules (tactics, derived rules)

- Thm : Application of an already proven termination statement (auxiliary functions)
- Hyp : Application of induction hypotheses
- Ax : Application of Axiom
- Eq : Application of an equational rule
- Struct : Use of structural rules + manipulations of formulas (Reasoning by cases)
- Ind : Use of induction rules + manipulations of formulas

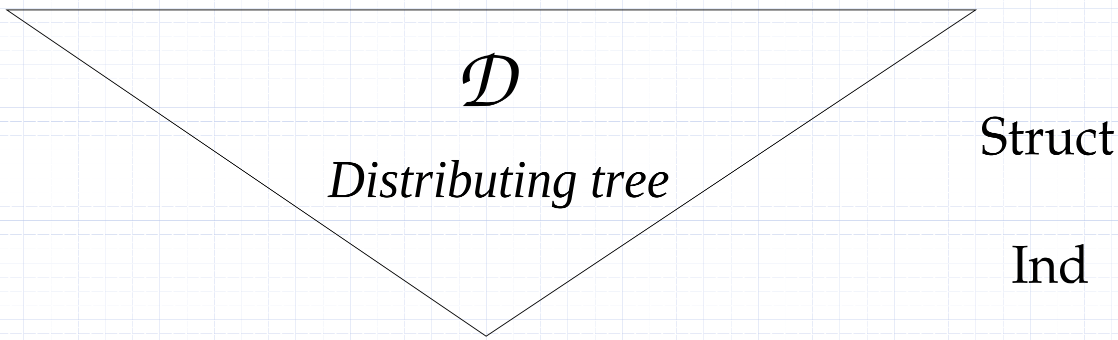
Shape of I-Proofs



Shape of I-Proofs

The Distributing tree must follow a property:

The formal terminal state property



$$\vdash_{\varepsilon_f} \forall x_1 D_1(x_1), \ \ldots, \forall x_n D_n(x_n) \Rightarrow D(f(x_1, \ldots, x_n))$$

Enlarging the class of extracted programs

- We revisit the ProPre system and analyse the formal proofs obtained in Propre.

Enlarging the class of extracted programs

- We revisit the ProPre system and analyse the formal proofs obtained in Propre.
  - In order to alleviate and simplify the notion of formal terminal state property (kernel of the I-proofs)

Enlarging the class of extracted programs

- We revisit the ProPre system and analyse the formal proofs obtained in Propre.
  - In order to alleviate and simplify the notion of formal terminal state property (kernel of the I-proofs)
  - In order to enlarge the class of extracted programs

Enlarging the class of extracted programs

- We revisit the ProPre system and analyse the formal proofs obtained in Propre.
  - In order to alleviate and simplify the notion of formal terminal state property (kernel of the I-proofs)
  - In order to enlarge the class of extracted programs
- We make simplification of Distributing Trees and Formulas.

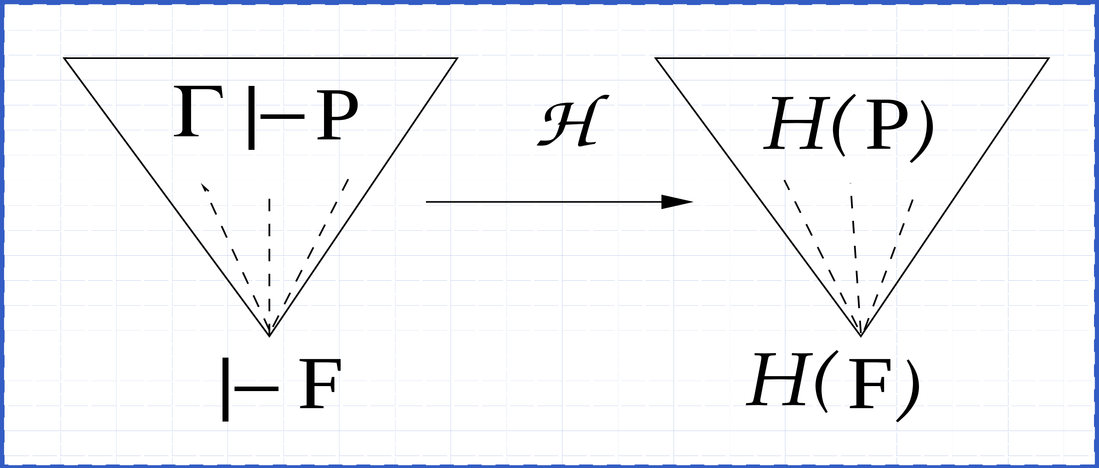
The skeleton proofs

- The heart of a formula F: it gives rise to a term t.

The skeleton proofs

- The heart of a formula F: it gives rise to a term t.
- The *skeleton operation* on the distributing tree gives rise to a term distributing tree.

The skeleton proofs

- The heart of a formula F: it gives rise to a term t.
- The *skeleton operation* on the distributing tree gives rise to a term distributing tree.
- 

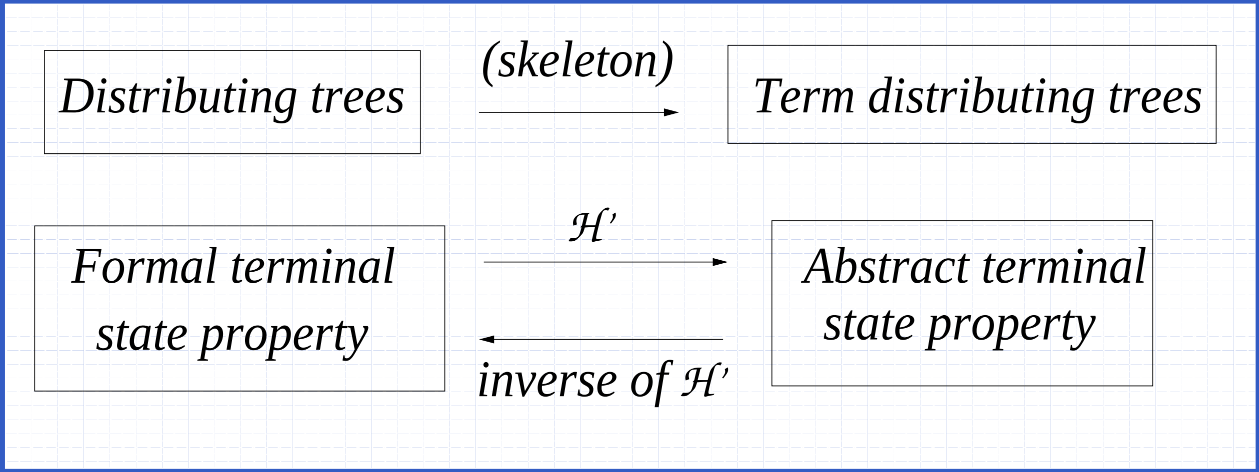
Formal proofs from skeleton forms

- The skeleton operation is not injective

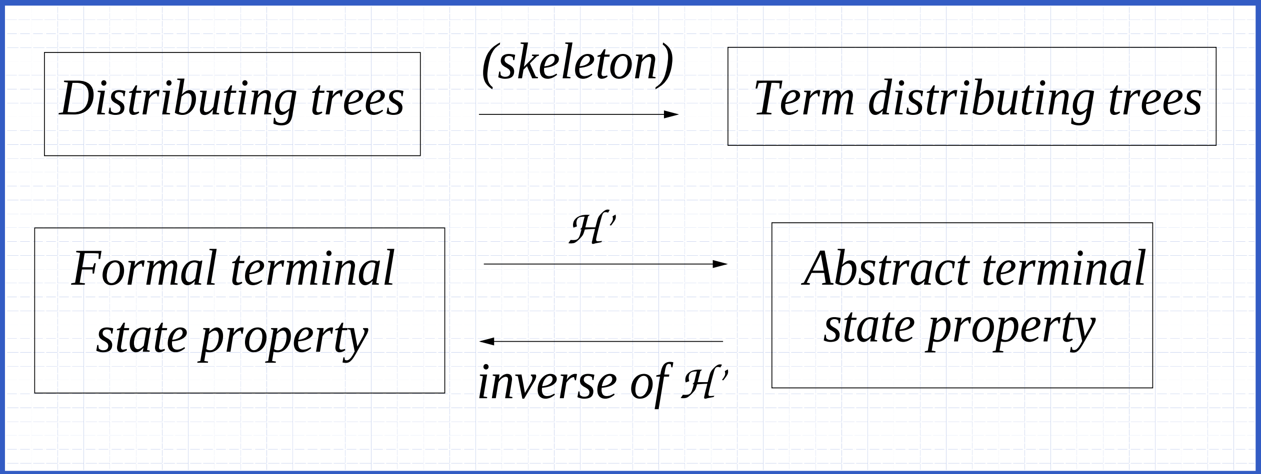
Formal proofs from skeleton forms

- The skeleton operation is not injective
- The design of abstract terminal state property

Formal proofs from skeleton forms

- The skeleton operation is not injective
- The design of abstract terminal state property
- 

Formal proofs from skeleton forms

- The skeleton operation is not injective
- The design of abstract terminal state property
- 

- We can rebuild proofs from *Atsp*

Termination proofs

- It is easier to work on term distributing trees for termination proofs.

Termination proofs

- It is easier to work on term distributing trees for termination proofs.
- We can extend the termination property independently from formal proofs.

Termination proofs

- It is easier to work on term distributing trees for termination proofs.
- We can extend the termination property independently from formal proofs.
- Can we extend the class of extracted programs in the same way as in ProPre ?

Termination proofs

- It is easier to work on term distributing trees for termination proofs.
- We can extend the termination property independently from formal proofs.
- Can we extend the class of extracted programs in the same way as in ProPre ?
- Example:

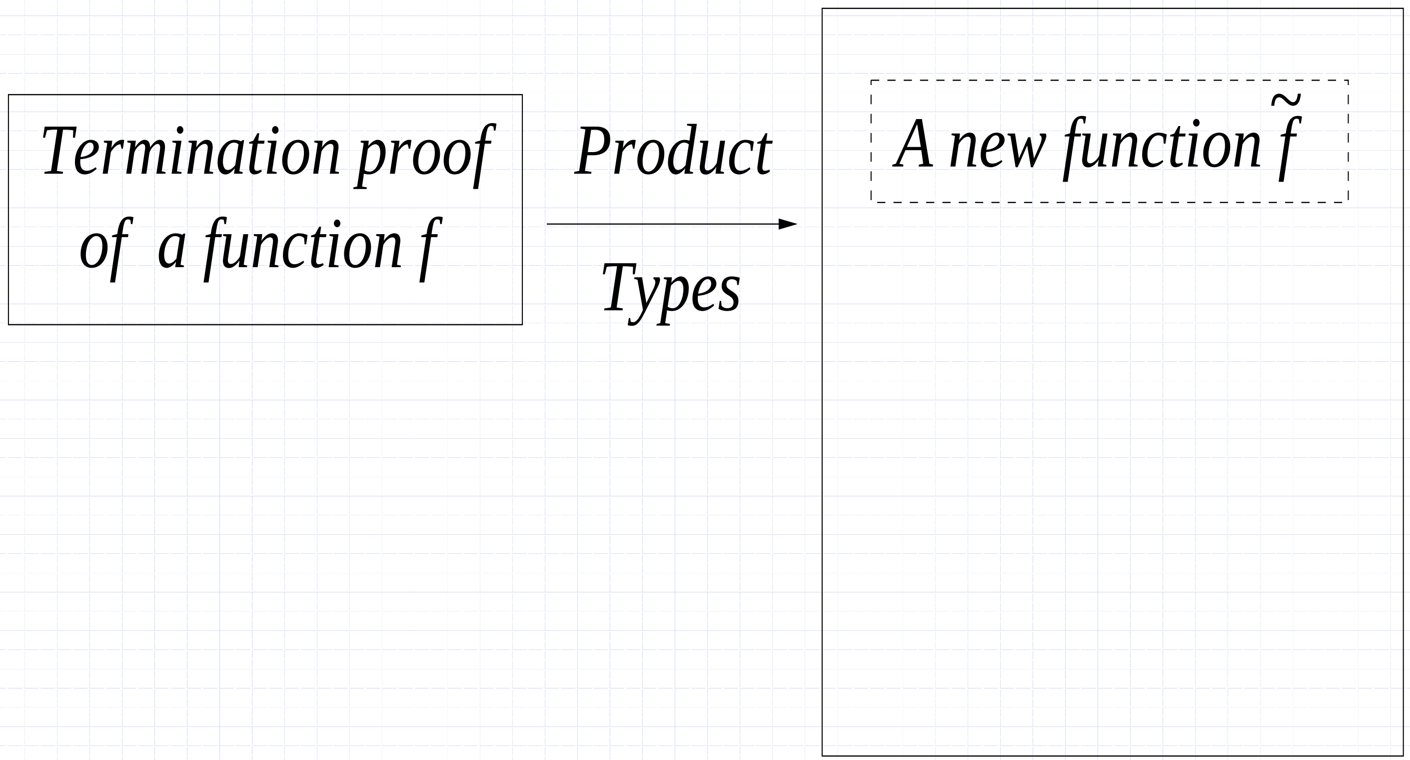
$$quot(x, 0, 0) = 0, \qquad quot(s(x), s(y), z) = quot(x, y, z),$$
$$quot(0, s(y), z) = 0, \quad quot(x, 0, s(z)) = s(quot(x, s(z), s(z)))$$

The term $quot(x, y, y)$ computes $\lfloor \frac{x}{y} \rfloor$.

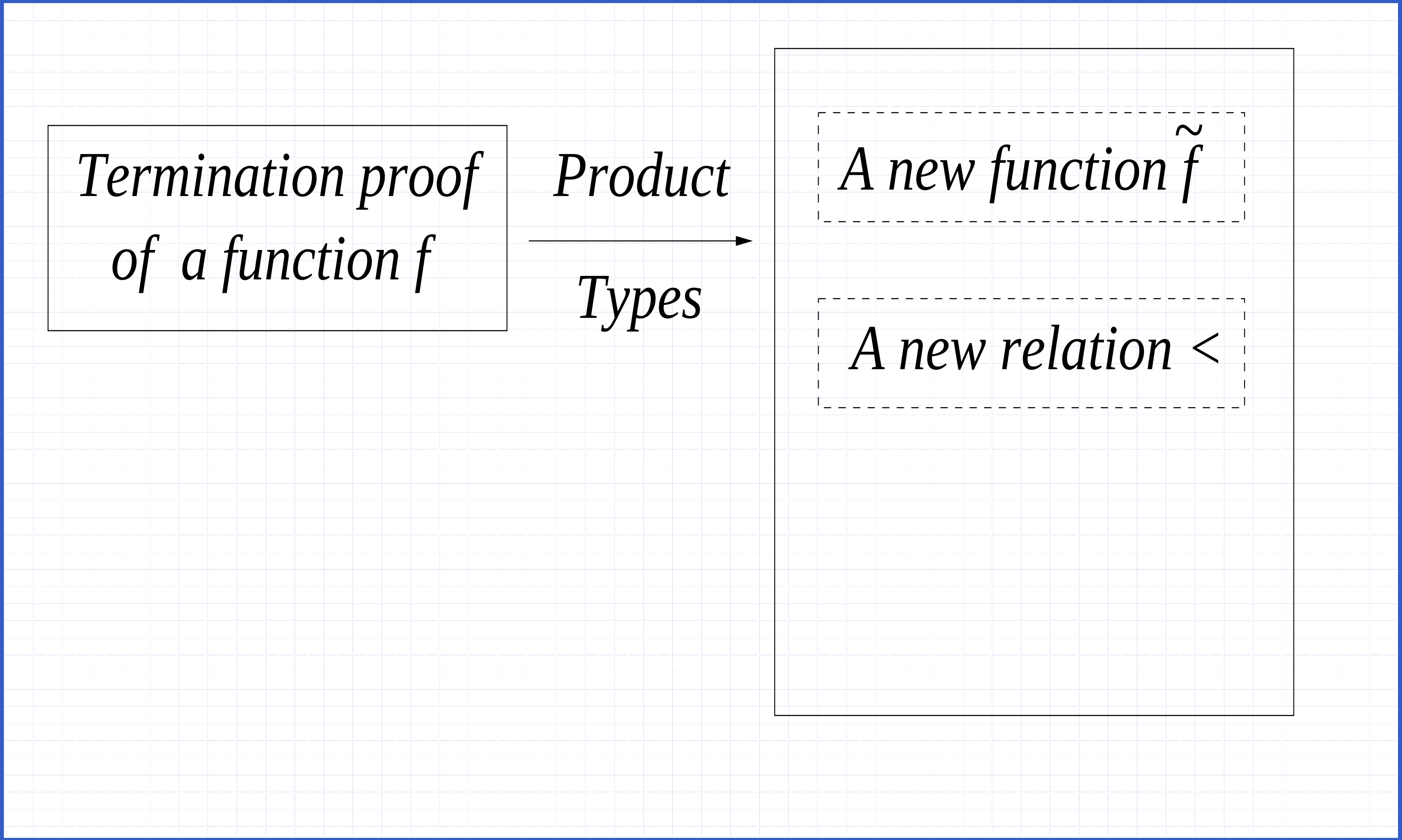
The main scheme

Termination proof of a function f

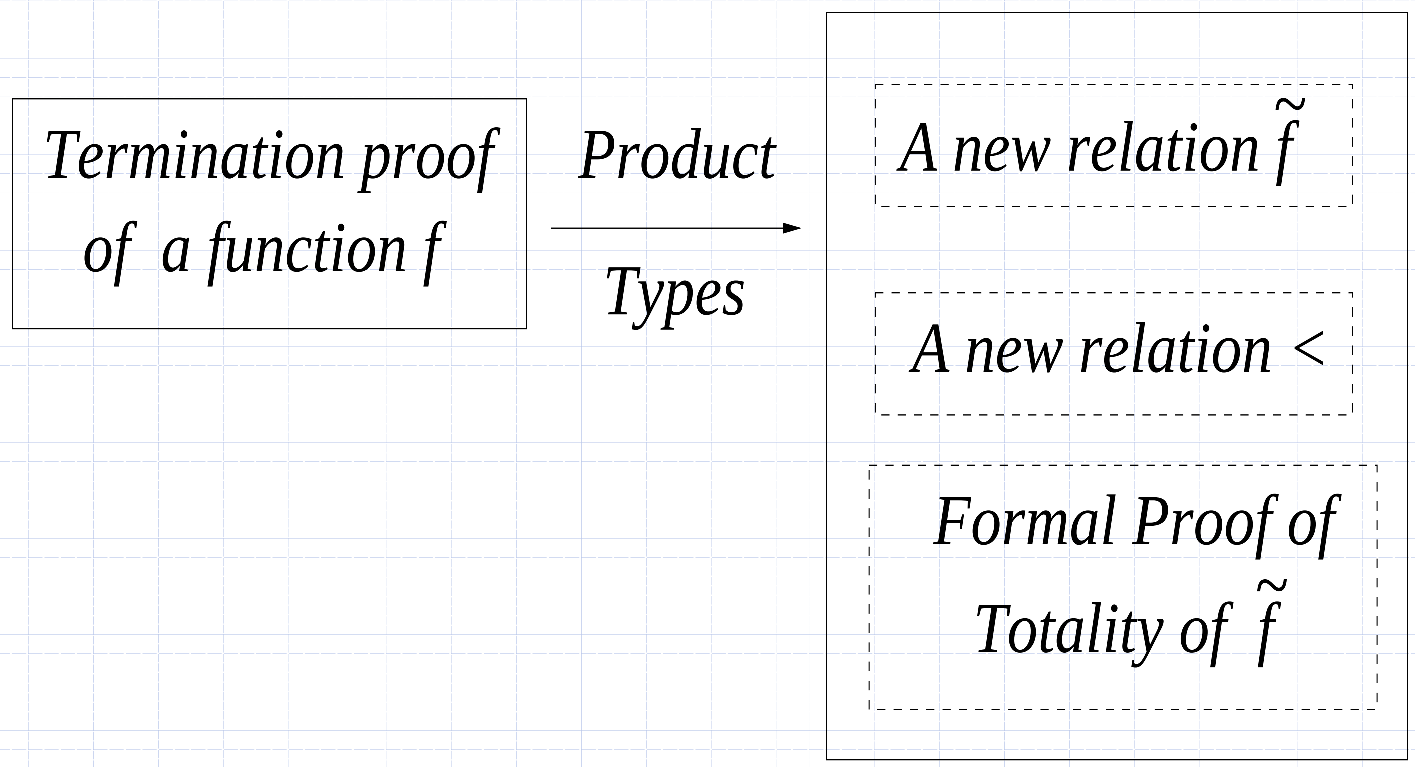
The main scheme



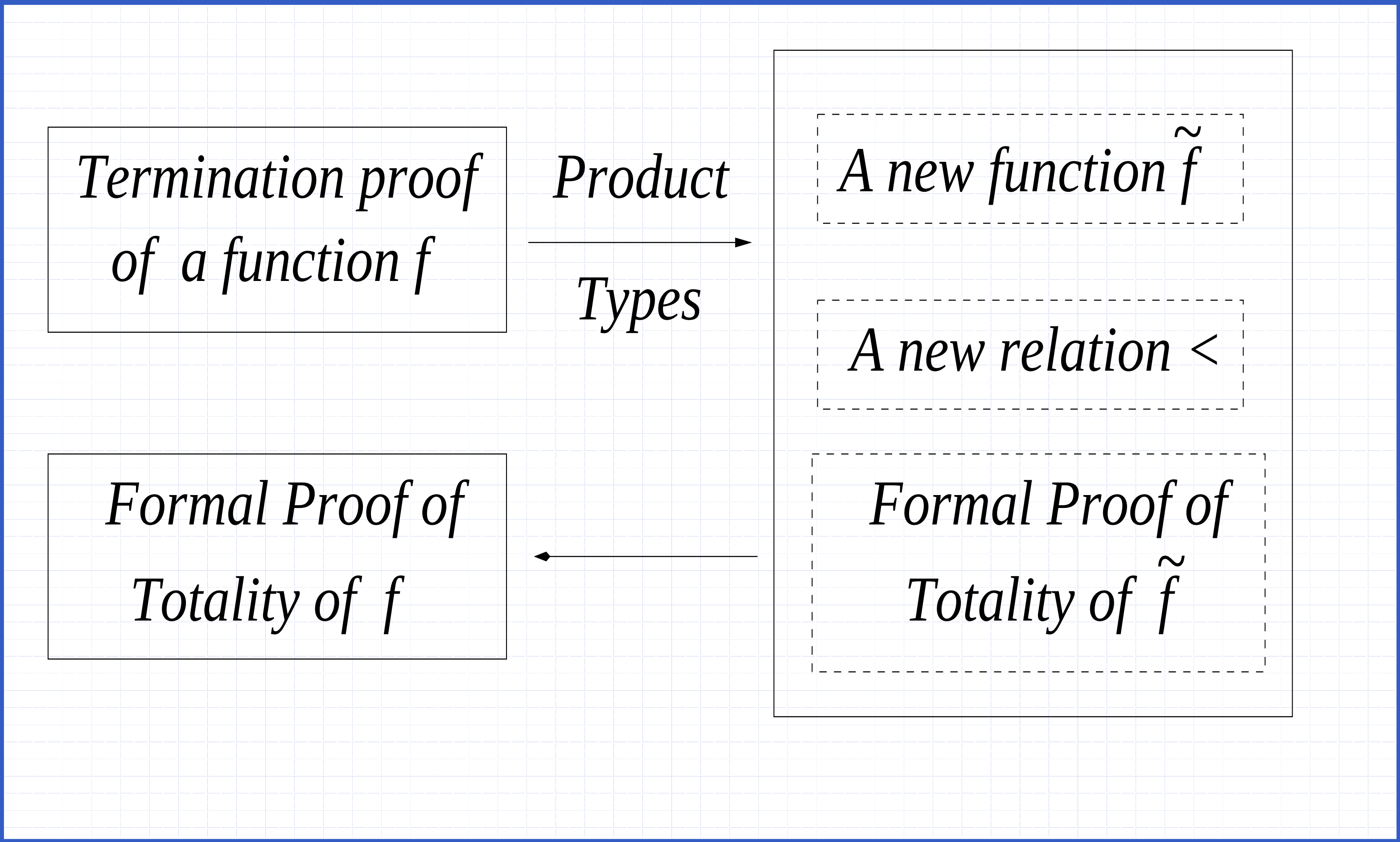
The main scheme



The main scheme



The main scheme



Product Type

- Let $f : D_1, \ldots , D_n \rightarrow D$ be a function with $\mathcal{E}_f$

Product Type

- Let $f : D_1, \ldots, D_n \longrightarrow D$ be a function with $\mathcal{E}_f$
- The product type of $D_1, \ldots, D_n$ is

$$\forall X \forall y_1, \ldots, y_n D_1(y_1), \ldots, D_n(y_n) \longrightarrow (X(cp(y_1, \ldots, y_n)) \longrightarrow X(x))$$

Product Type

- Let $f : D_1, \ldots, D_n \rightarrow D$ be a function with $\mathcal{E}_f$
- The product type of $D_1, \ldots, D_n$ is

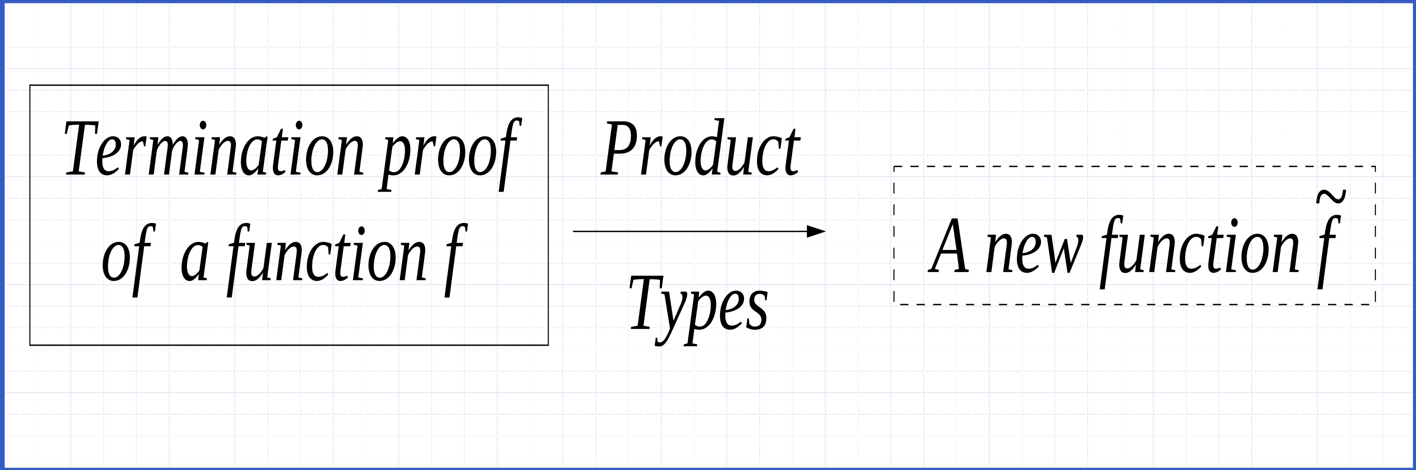
$$\forall X \forall y_1, \ldots, y_n D_1(y_1), \ldots, D_n(y_n) \rightarrow (X(cp(y_1, \ldots, y_n)) \rightarrow X(x))$$

- We can define a new function $\tilde{f}$ with $\mathcal{E}_{\tilde{f}}$ from $\mathcal{E}_f$

Product Type

- Let $f : D_1, \ldots, D_n \to D$ be a function with $\mathcal{E}_f$
- The product type of $D_1, \ldots, D_n$ is

$$\forall X \forall y_1, \ldots, y_n D_1(y_1), \ldots, D_n(y_n) \to (X(cp(y_1, \ldots, y_n)) \to X(x))$$

- We can define a new function $\tilde{f}$ with $\mathcal{E}_{\tilde{f}}$ from $\mathcal{E}_f$
- 

Product Type

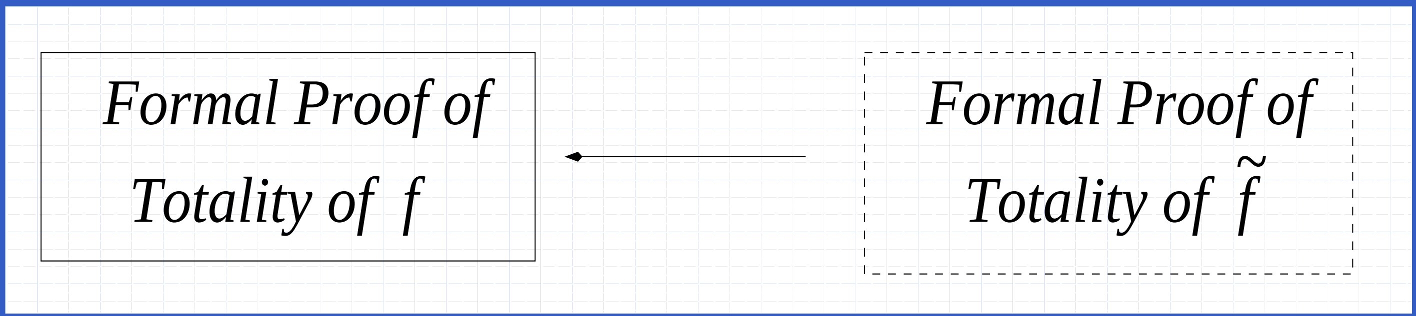
- The termination statement of $\tilde{f}$ is $T_{\tilde{f}} = \forall x((D_1 \times \ldots \times D_n)(x) \rightarrow D(\tilde{f}(x)))$.

Product Type

- The termination statement of $\tilde{f}$ is $T_{\tilde{f}} = \forall x((D_1 \times \ldots \times D_n)(x) \rightarrow D(\tilde{f}(x)))$.
- Fact: If there is a λ-term $\tilde{F}$ such that $\vdash_{\mathcal{E}_{\tilde{f}}} \tilde{F} : T_{\tilde{f}}$, then there is a λ-term F such that $\vdash_{\mathcal{E}'_f} F : T_f$ with

$$\mathcal{E}'_f = \mathcal{E}_f \cup \{f(x_1, \ldots, x_n) = \tilde{f}(cp(x_1, \ldots, x_n))\} \cup \mathcal{E}_{\tilde{f}}.$$

Product Type

- The termination statement of $\tilde{f}$ is $T_{\tilde{f}} = \forall x((D_1 \times \ldots \times D_n)(x) \rightarrow D(\tilde{f}(x)))$.
- Fact: If there is a λ-term $\tilde{F}$ such that $\vdash_{\mathcal{E}_{\tilde{f}}} \tilde{F} : T_{\tilde{f}}$, then there is a λ-term F such that $\vdash_{\mathcal{E}'_f} F : T_f$ with $\mathcal{E}'_f = \mathcal{E}_f \cup \{f(x_1, \ldots, x_n) = \tilde{f}(cp(x_1, \ldots, x_n))\} \cup \mathcal{E}_{\tilde{f}}$.
- 

Canonical I-proofs

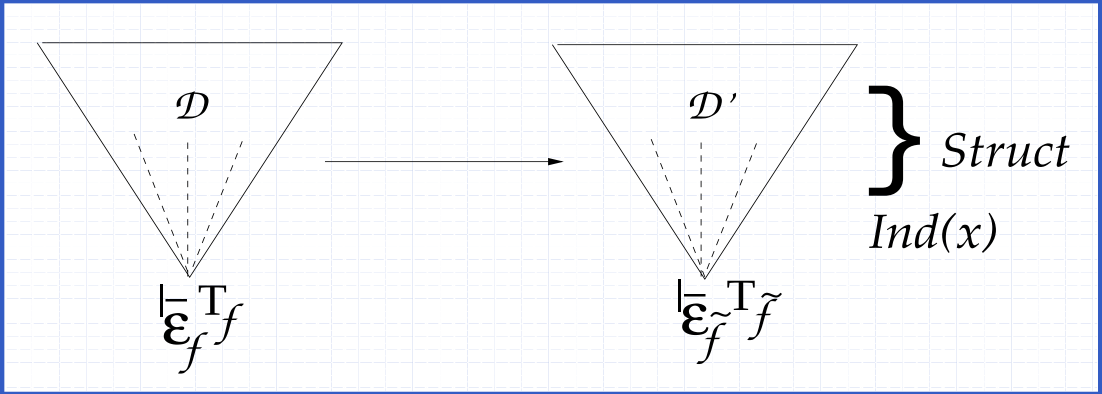
- We change the relation $\prec$ about $\tilde{f}$.:

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

Canonical I-proofs

- We change the relation $\prec$ about $\tilde{f}$.:

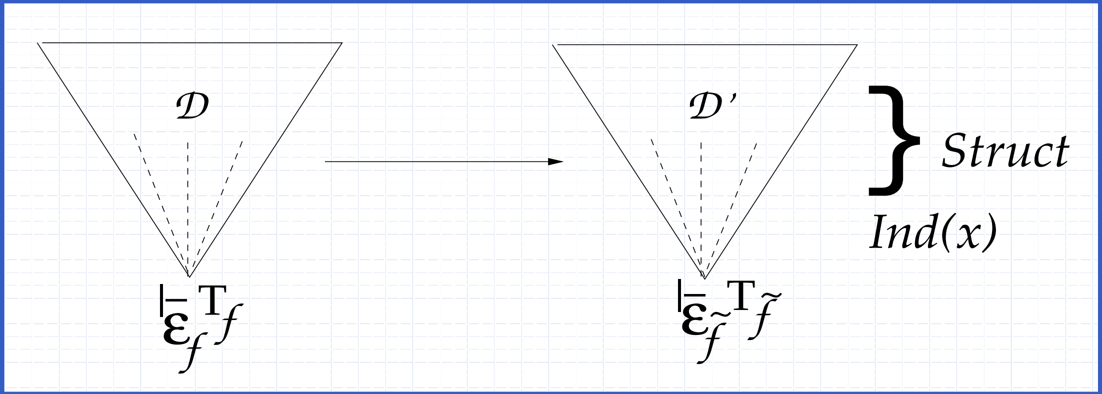
$$\frac{\Gamma \vdash_{\mathcal{E}} \; t : \forall x[\forall z[Dz_{\prec x} \to B[z/x]] \to [D(x) \to B]]}{\Gamma \; \vdash_{\mathcal{E}} \; (T\; t) : \forall x[D(x) \to B]} \quad (Ext)$$

- 

Canonical I-proofs

- We change the relation $\prec$ about $\tilde{f}$.:

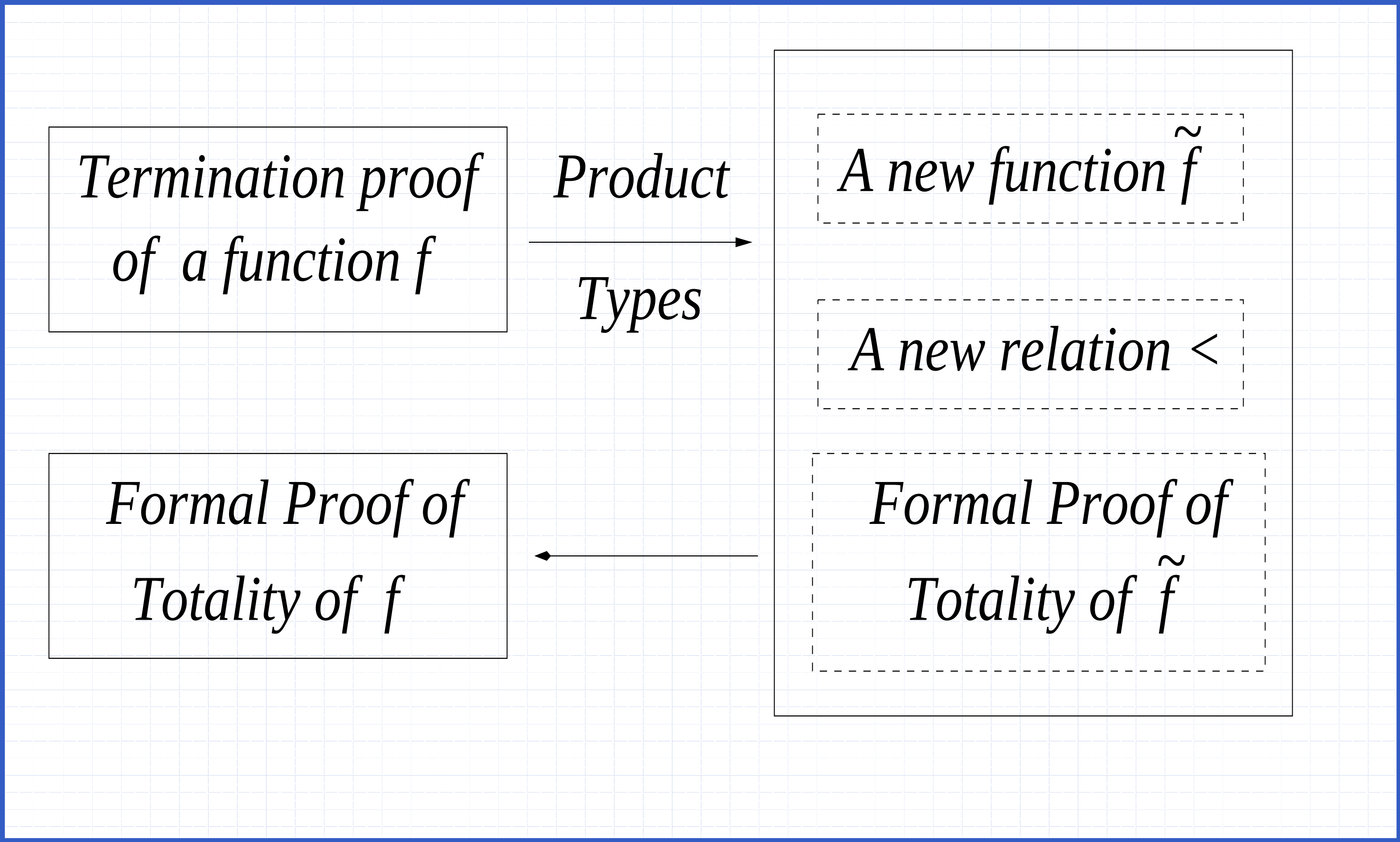
$$\frac{\Gamma \vdash_{\mathcal{E}} \ t : \forall x[\forall z[Dz_{\prec x} \rightarrow B[z/x]] \rightarrow [D(x) \rightarrow B]]}{\Gamma \ \vdash_{\mathcal{E}} \ (T\ t) : \forall x[D(x) \rightarrow B]} \quad (Ext)$$

- 

- The hiding rules allow the formal proofs to be released from the termination part.

$$\frac{\Gamma \vdash_{\mathcal{E}} \ t{:}A \qquad \Gamma \vdash_{\mathcal{E}} \ e}{\Gamma \vdash_{\mathcal{E}} \ t{:}A{\restriction}e} \qquad \frac{\Gamma \vdash_{\mathcal{E}} \ t{:}A{\restriction}e}{\Gamma \vdash_{\mathcal{E}} \ t{:}A}$$

Canonical I-proofs



Conclusion

- The ProPre system showed the feasibility of the theory based on "Proofs as Programs".

Conclusion

- The ProPre system showed the feasibility of the theory based on "Proofs as Programs".
- A main issue is the automation of formal proofs.

Conclusion

- The ProPre system showed the feasibility of the theory based on "Proofs as Programs".
- A main issue is the automation of formal proofs.
- We have shown we can go further for the automation of extracted programs.

Conclusion

- The ProPre system showed the feasibility of the theory based on "Proofs as Programs".
- A main issue is the automation of formal proofs.
- We have shown we can go further for the automation of extracted programs.
- It remains the implementation.

The End