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

WOLLIC’02, 30th July-2nd August 2002, Rio de Janeiro, Brazil
**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:

\[ \text{quot}(x, 0, 0) = 0, \quad \text{quot}(s(x), s(y), z) = \text{quot}(x, y, z), \]
\[ \text{quot}(0, s(y), z) = 0, \quad \text{quot}(x, 0, s(z)) = s(\text{quot}(x, s(z), s(z))) \]

The term \( \text{quot}(x, y, y) \) computes \( \left\lfloor \frac{x}{y} \right\rfloor \).
Motivation

- The difficulty is to find formal proofs automatically.

Example:

\[ \text{quot}(x, 0, 0) = 0, \quad \text{quot}(s(x), s(y), z) = \text{quot}(x, y, z), \]
\[ \text{quot}(0, s(y), z) = 0, \quad \text{quot}(x, 0, s(z)) = s(\text{quot}(x, s(z), s(z))) \]

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

- The proofs are expressed in natural deduction style.
Motivation

- The difficulty is to find formal proofs automatically.
- Example:
  \[ \text{quot}(x, 0, 0) = 0, \quad \text{quot}(s(x), s(y), z) = \text{quot}(x, y, z), \]
  \[ \text{quot}(0, s(y), z) = 0, \quad \text{quot}(x, 0, s(z)) = s(\text{quot}(x, s(z), s(z))) \]

The term \( \text{quot}(x, y, y) \) 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:
    \[
    \text{TypeLn} : \text{Nil} \mid \text{consNLn};
    \]
  - The append function in ProPre:
    \[
    \text{Let append} : \text{Ln} \mid \text{Ln} \rightarrow \text{Ln}
    \]
    \[
    \text{Nil y} \rightarrow y
    \]
    \[
    (\text{Cons n x}) y \rightarrow (\text{Cons n (append x y)});
    \]
The ProPre system

- ProPre is a program synthesis system.
- Example:
  - The type of the list of natural number in ProPre:
    \[
    \text{Type } \text{Ln} : \text{Nil} \mid \text{cons N Ln};
    \]
  - The append function in ProPre:
    \[
    \text{Let } \text{append} : \text{Ln} \mid \text{Ln} \rightarrow \text{Ln}
    \]
    \[
    \text{Nil y => y}
    \]
    \[
    \text{(Cons n x) y => (Cons n (append x y))};
    \]
- 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 $\Rightarrow$ lambda-term
The ProPre system

- Functional programming language based on the paradigm: Programming by Proofs ("Proofs as Programs")
- Type System: program extraction $\Rightarrow$ 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 $\text{nat}$
    
    $0 : \to \text{nat}, \ s : \text{nat} \to \text{nat}$
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 $nat$
    
    $0 : \rightarrow nat, s : nat \rightarrow nat$

  • 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 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 \( \text{nat} \)
    \[
    0 : \rightarrow \text{nat}, \ s : \text{nat} \rightarrow \text{nat}
    \]
  - 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.

\[ \text{Data-Type: Formula of Second Order} \]

\[ \downarrow \]

Programs for constructors (successor for integers, cons for lists, etc...)
Intuitionistic rules

\( \Gamma, A \vdash A \) (ax)

\( \frac{\Gamma \vdash A[u]}{\Gamma \vdash A[v]} \) (eq)

\( \frac{\Gamma \vdash A \quad \Gamma \vdash A \to B}{\Gamma \vdash B} \) (→e)

\( \frac{\Gamma \vdash A}{\Gamma \vdash \forall y A} \) (∀₁)
Second Order Functional Arithmetic

\( \Gamma, \ x : A \vdash x : A \) \quad (ax)

\( \Gamma \vdash t : A[u] \quad \Gamma \vdash \varepsilon \ u = v \) \quad (eq)

\( \Gamma, \ x : A \vdash t : B \) \quad (\to_i)

\( \Gamma \vdash u : A \quad \Gamma \vdash t : A \to B \) \quad (\to_e)

\( \Gamma \vdash t : A \) \quad (\forall_1^i)

\( \Gamma \vdash t : \forall y A \) \quad (\forall_1^e)

\( \Gamma \vdash t : \forall y A \) \quad (\forall_2^i)

\( \Gamma \vdash t : \forall y A \) \quad (\forall_2^e)

\( \Gamma \vdash t : \forall y A \) \quad (\forall_1^i)

\( \Gamma \vdash t : \forall Y A \) \quad (\forall_2^e)
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, $E_f$ a set of equations, $t$ a lambda-term.
  - If
    $$\vdash_{E_f} t : \forall x_1, \ldots, \forall x_n \{D_1[x_1], \ldots, D_n[x_n] \rightarrow D[f(x_1, \ldots, x_n)]\}$$
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
    \[
    \vdash_{\mathcal{E}_f} t : \forall x_1, \ldots, \forall x_n \{ D_1[x_1], \ldots, D_n[x_n] \rightarrow D[f(x_1, \ldots, x_n)] \}
    \]
  - 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
    $$\vdash_{\mathcal{E}_f} t : \forall x_1, \ldots, \forall x_n \{ D_1[x_1], \ldots, D_n[x_n] \rightarrow D[f(x_1, \ldots, x_n)] \}$$
  - Then "$t$ computes $f$"
  - Let $f : \text{nat} \rightarrow \text{nat}$. If $\vdash_{\mathcal{E}_f} t : \forall x (N(x) \rightarrow N[f(x)])$ then
    $$\vdash_{\mathcal{E}_f} f(s^n(0)) = s^m(0) \text{ iff } (t \ n) \rightarrow_{\beta} m$$

WOLLIC’02, 30th July-2nd August 2002; Rio de Janeiro, Brazil
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.
Some rules in TTR

- Rules of the hiding operator $\uparrow$

If $A$ is a formula, $u$, $v$ terms then $A \uparrow (u \simeq v)$ is a formula.

$$
\frac{\Gamma \vdash_{\xi} t : A \quad \Gamma \vdash_{\xi} e}{\Gamma \vdash_{\xi} t : A \uparrow e} \quad (\uparrow 1) \quad \frac{\Gamma \vdash_{\xi} t : A \uparrow e}{\Gamma \vdash_{\xi} t : A} \quad (\uparrow 2) \quad \frac{\Gamma \vdash_{\xi} t : A \uparrow e}{\Gamma \vdash_{\xi} e} \quad (\uparrow 3)
$$
Somes rules in TTR

• Rules of the hiding operator ⌫

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

\[
\frac{\Gamma \vdash \varepsilon \ t : A}{\Gamma \vdash \varepsilon \ t : A \upharpoonright e} \quad \left(\uparrow 1\right) \quad \frac{\Gamma \vdash \varepsilon \ t : A \upharpoonright e}{\Gamma \vdash \varepsilon \ e} \quad \left(\uparrow 2\right) \quad \frac{\Gamma \vdash \varepsilon \ t : A \upharpoonright e}{\Gamma \vdash \varepsilon \ e} \quad \left(\uparrow 3\right)
\]

• External induction rule

\[
\frac{\Gamma \vdash \varepsilon \ t : \forall x [\forall z [Dz \prec x \rightarrow B [z/x]] \rightarrow [D(x) \rightarrow B]]}{\Gamma \vdash \varepsilon \ (T \ t) : \forall x [D(x) \rightarrow B]} \quad \left(\text{Ext}\right)
\]

$T$ 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

\[ \Gamma_1 \vdash \theta_1 \quad \ldots \quad \ldots \quad \Gamma_2 \vdash \theta_2 \]

\[ \mathcal{D} \quad \text{Distributing tree} \]

\[ \vdash_{\xi_f} \forall x_1 D_1(x_1), \ldots, \forall x_n D_n(x_n) \rightarrow D(f(x_1, \ldots, x_n)) \]
The Distributing tree must follow a property:

**The formal terminal state property**

\[ \forall x_1 \forall x_2 \ldots \forall x_n (D_1(x_1) \land \ldots \land D_n(x_n) \rightarrow D(f(x_1, \ldots, x_n))) \]
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
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.

\[\Gamma \vdash P \quad \mathcal{H} \quad H(P)\]

\[\vdash F \quad H(F)\]
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

\[ \begin{array}{ccc}
\text{Distributing trees} & \xrightarrow{\text{skeleton}} & \text{Term distributing trees} \\
\text{Formal terminal state property} & \xrightarrow{\mathcal{H}'} & \text{Abstract terminal state property}
\end{array} \]

- We can rebuild proofs from \( \text{Atsp} \)

WOLLIC’02, 30th July-2nd August 2002; Rio de Janeiro, Brazil
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:

\[
\text{quot}(x, 0, 0) = 0, \quad \text{quot}(s(x), s(y), z) = \text{quot}(x, y, z), \\
\text{quot}(0, s(y), z) = 0, \quad \text{quot}(x, 0, s(z)) = s(\text{quot}(x, s(z), s(z))
\]

The term \(\text{quot}(x, y, y)\) computes \(\left\lfloor \frac{x}{y} \right\rfloor\).
The main scheme

Termination proof of a function $f$
The main scheme

Termination proof of a function $f$ → Product Types → A new function $\tilde{f}$
The main scheme

Termination proof of a function $f$ \quad Product Types \quad A new function $\tilde{f}$

A new relation $<$
The main scheme

Termination proof of a function $f$ \hspace{2cm} Product \hspace{2cm} A new relation $\sim$

Types \hspace{2cm} A new relation $<$

Formal Proof of Totality of $f$
The main scheme

<table>
<thead>
<tr>
<th>Termination proof of a function $f$</th>
<th>Product Types</th>
<th>A new function $\tilde{f}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Formal Proof of Totality of $f$</td>
<td></td>
<td>A new relation $&lt;$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Formal Proof of Totality of $\tilde{f}$</td>
</tr>
</tbody>
</table>

**Formal Proof of Totality of $f$**

**Product Types**

**A new function $\tilde{f}$**

**A new relation $<$**

**Formal Proof of Totality of $\tilde{f}$**
Product Type

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

- Let $f : D_1, \ldots , D_n \to D$ be a function with $E_f$
- The product type of $D_1, \ldots , D_n$ is

$$\forall X \forall y_1, \ldots , y_n D_1(y_1), \ldots , D_n(y_n) \to (X(cp(y_1, \ldots , y_n)) \to X(x))$$
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 $f : D_1, \ldots, D_n \rightarrow D$ 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 $\mathcal{E}_f$

<table>
<thead>
<tr>
<th>Termination proof of a function $f$</th>
<th>Product Types</th>
<th>A new function $\tilde{f}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

WOLLIC’02, 30th July-2nd August 2002; Rio de Janeiro, Brazil
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 $\lambda$-term $\tilde{F}$ such that
  $$\vdash_{\mathcal{E}_{\tilde{f}}} \tilde{F} : T_{\tilde{f}},$$
  then there is a $\lambda$-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 $\lambda$-term $\tilde{F}$ such that
  \[ \vdash \varepsilon_{\tilde{f}} \tilde{F} : T_{\tilde{f}}, \]
  then there is a $\lambda$-term $F$ such that
  \[ \vdash \varepsilon'_f F : T_f \]
  with
  \[ \varepsilon'_f = \varepsilon_f \cup \{ f(x_1, \ldots, x_n) = \tilde{f}(cp(x_1, \ldots, x_n)) \} \cup \varepsilon_{\tilde{f}}. \]
Canonical I-proofs

- We change the relation \( \prec \) about \( \tilde{f} \):

\[
\Gamma \vdash \varepsilon\ t : \forall x[\forall z(Dz \prec x \rightarrow B[z/x] \rightarrow [D(x) \rightarrow B])] \\
\Gamma \vdash \varepsilon\ (T\ t) : \forall x[D(x) \rightarrow B] \quad (Ext)
\]
We change the relation $\prec$ about $\tilde{f}$:

\[
\Gamma \vdash \varepsilon \quad t : \forall x[\forall z[Dz \prec x \rightarrow B[z/x]] \rightarrow [D(x) \rightarrow B]] \\
\Gamma \vdash \varepsilon \quad (T \ t) : \forall x[D(x) \rightarrow B]
\]

(Canonical I-proofs)
We change the relation $\prec$ about $\tilde{f}$:

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

$$\Gamma \vdash_{\mathcal{E}} (T \ t) : \forall x[D(x) \rightarrow B]$$

(Ext)

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

$$\Gamma \vdash_{\mathcal{E}} t : A \quad \Gamma \vdash_{\mathcal{E}} e$$

$$\Gamma \vdash_{\mathcal{E}} t : A \vdash e$$

$$\Gamma \vdash_{\mathcal{E}} t : A \vdash e$$

$$\Gamma \vdash_{\mathcal{E}} t : A$$

WOLLIC’02, 30th July-2nd August 2002; Rio de Janeiro, Brazil
Canonical I-proofs

Termination proof of a function $f$ \quad Product Types \quad A new function $\tilde{f}$

Formal Proof of Totality of $f$ \quad Formal Proof of Totality of $\tilde{f}$

A new relation $<$
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