PROGRAM-ing in COQ

MATTHIEU SOZEAU
under the direction of CHRISTINE PAULIN-MOHRING

LRI, Univ. Paris-Sud - DÉMONS Team & INRIA SACLAY - PROVAL Project

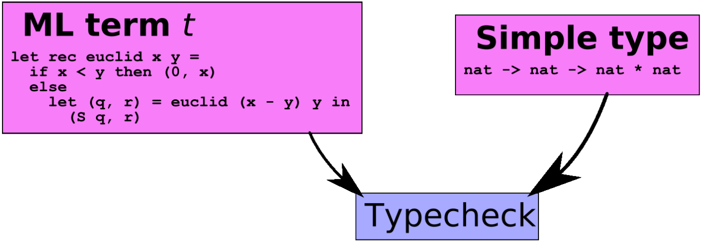
Foundations of Programming seminar
February 15th 2008
University of Nottingham

The Big Picture

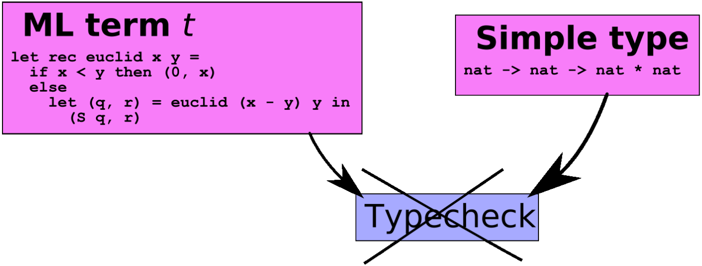
ML term *t*

```
let rec euclid x y =
  if x < y then (0, x)
  else
    let (q, r) = euclid (x - y) y in
      (S q, r)
```

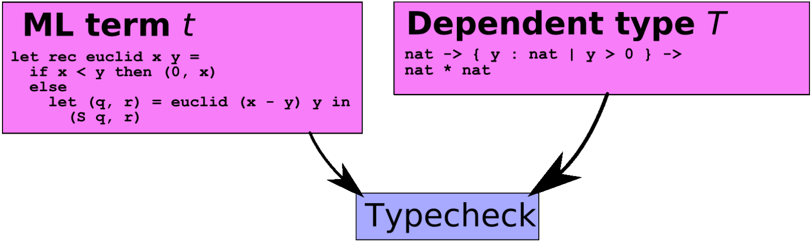
The Big Picture



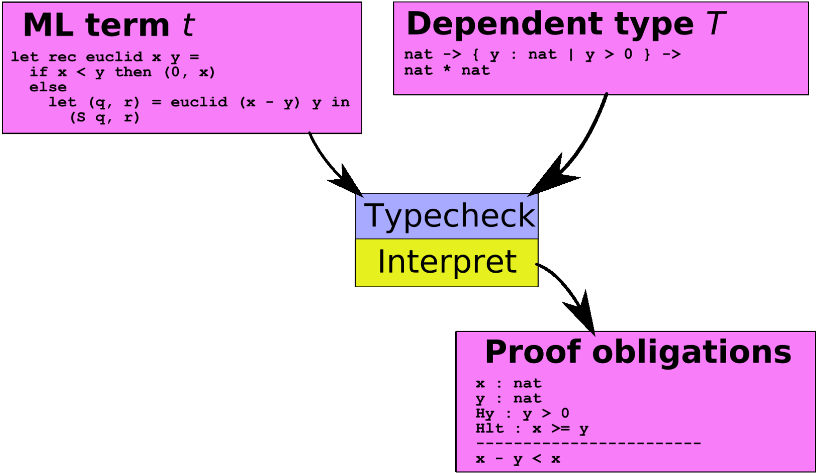
The Big Picture



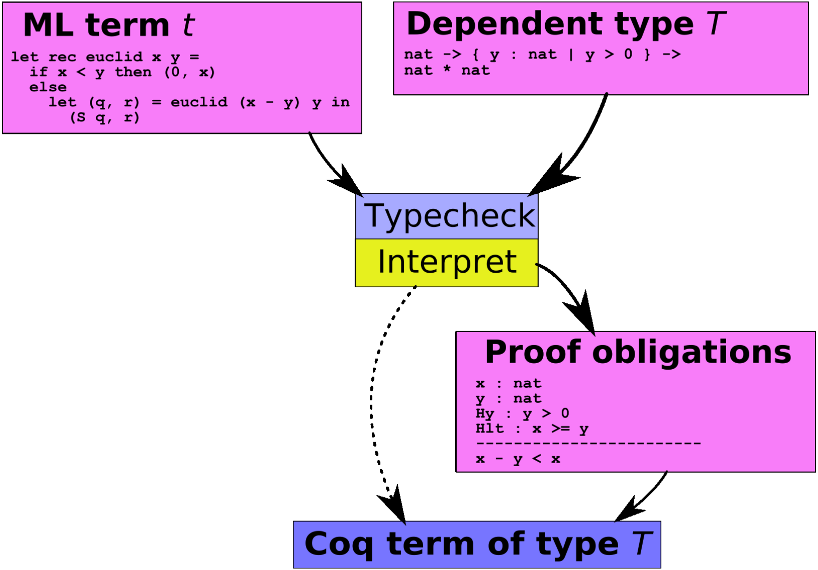
The Big Picture



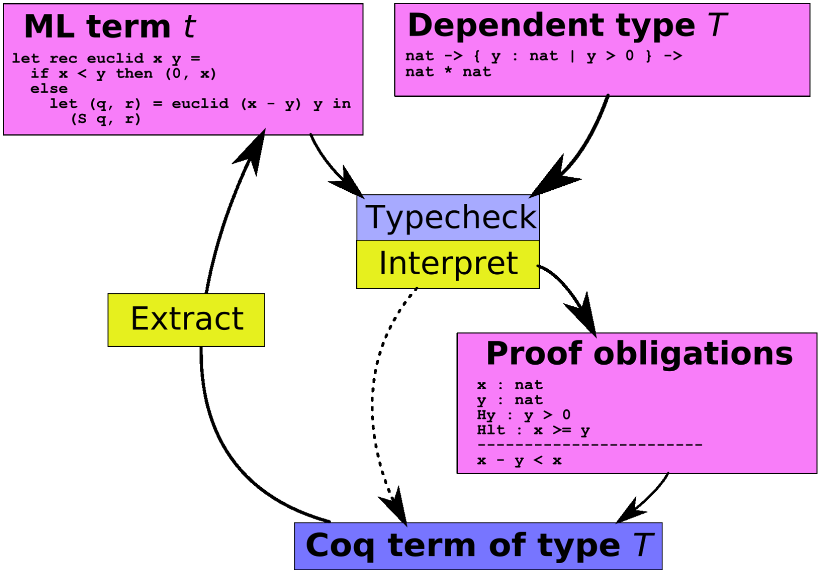
The Big Picture



The Big Picture



The Big Picture

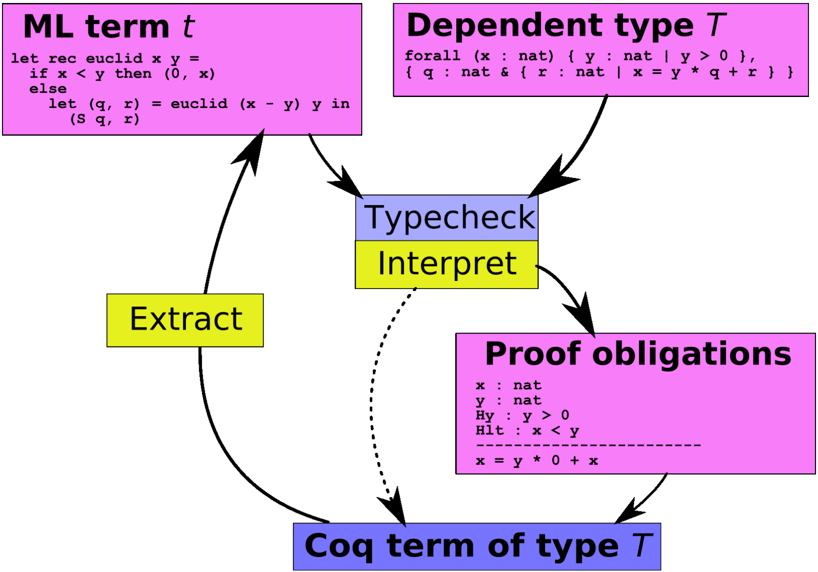


The Big Picture

Inductive $\mathit{diveucl}\ a\ b$: Set :=
  $\mathit{divex} : \forall\ q\ r,\ b > r \rightarrow a = q \times b + r \rightarrow \mathit{diveucl}\ a\ b.$

Lemma $\mathit{eucl\_dev} : \forall\ n,\ n > 0 \rightarrow \forall\ m{:}\mathit{nat},\ \mathit{diveucl}\ m\ n.$
Proof.
  $\mathit{intros}\ b\ H\ a$; $\mathit{pattern}\ a$ in $\vdash \times$; $\mathit{apply}\ \mathit{gt\_wf\_rec}$; $\mathit{intros}\ n\ H0.$
  $\mathit{elim}\ (\mathit{le\_gt\_dec}\ b\ n).$
  $\mathit{intro}\ \mathit{lebn}.$
  $\mathit{elim}\ (H0\ (n - b))$; auto with $\mathit{arith}.$
  $\mathit{intros}\ q\ r\ g\ e.$
  $\mathit{apply}\ \mathit{divex}$ with $(S\ q)\ r$; simpl in $\vdash \times$; auto with $\mathit{arith}.$
  $\mathit{elim}\ \mathit{plus\_assoc}.$
  $\mathit{elim}\ e$; auto with $\mathit{arith}.$
  $\mathit{intros}\ \mathit{gtbn}.$
  $\mathit{apply}\ \mathit{divex}$ with $0\ n$; simpl in $\vdash \times$; auto with $\mathit{arith}.$
Qed.

The Big Picture



The CURRY-HOWARD isomorphism

Programming language = Proof system

The Curry-Howard isomorphism

Programming language = Proof system

Program extends the Coq proof-assistant into a dependently-typed programming environment.

The Curry-Howard isomorphism

Programming language = Proof system

Program extends the Coq proof-assistant into a dependently-typed programming environment.

Epigram **PVS** **DML** **Ωmega**

The Curry-Howard isomorphism

Programming language = Proof system

Program extends the Coq proof-assistant into a dependently-typed programming environment.

- **Logical Framework** Type Theory.

Epigram **PVS** ~~**DML**~~ ~~**Ωmega**~~

The Curry-Howard isomorphism

Programming language = Proof system

Program extends the Coq proof-assistant into a dependently-typed programming environment.

- **Logical Framework** Type Theory.
  Separates proofs and programs using sorts ⇒ Extraction

~~**Epigram**~~ ~~**PVS**~~ ~~**DML**~~ ~~**Ωmega**~~

The Curry-Howard isomorphism

Programming language = Proof system

Program extends the Coq proof-assistant into a dependently-typed programming environment.

- **Logical Framework** Type Theory.
  Separates proofs and programs using sorts ⇒ Extraction
- **Paradigm** Purely functional.

Epigram **PVS** ~~**DML**~~ **Ωmega**

The Curry-Howard isomorphism

Programming language = Proof system

Program extends the Coq proof-assistant into a dependently-typed programming environment.

- **Logical Framework** Type Theory.
  Separates proofs and programs using sorts ⇒ Extraction
- **Paradigm** Purely functional.
  Total, no separation of terms and types.

Epigram **PVS** ~~**DML**~~ ~~**Ωmega**~~

The Curry-Howard isomorphism

Programming language = Proof system

Program extends the Coq proof-assistant into a dependently-typed programming environment.

- **Logical Framework** Type Theory.
  Separates proofs and programs using sorts ⇒ Extraction
- **Paradigm** Purely functional.
  Total, no separation of terms and types.
- **Development style and proof automation** Interactive, semi-automatic proof using tactics.

~~Epigram~~ PVS ~~DML~~ ~~Ωmega~~

The Curry-Howard isomorphism

Programming language = Proof system

Program extends the Coq proof-assistant into a dependently-typed programming environment.

- **Logical Framework** Type Theory.
  Separates proofs and programs using sorts ⇒ Extraction
- **Paradigm** Purely functional.
  Total, no separation of terms and types.
- **Development style and proof automation** Interactive, semi-automatic proof using tactics.
- **Phase distinction** none

Epigram ~~**PVS**~~ **DML** **Ωmega**

The Curry-Howard isomorphism

Programming language = Proof system

Program extends the Coq proof-assistant into a dependently-typed programming environment.

- **Logical Framework** Type Theory.
  Separates proofs and programs using sorts ⇒ Extraction
- **Paradigm** Purely functional.
  Total, no separation of terms and types.
- **Development style and proof automation** Interactive, semi-automatic proof using tactics.
- **Phase distinction** ⇒ in Program

~~**Epigram**~~ **PVS** ~~**DML**~~ ~~**Ωmega**~~

A simple idea

Definition

$\{x : T \mid P\}$ is the set of objects of set T verifying property P.

- Useful for specifying, widely used in mathematics ;
- Links object and property.

A simple idea

Definition

$\{x : T \mid P\}$ is the set of objects of set T verifying property P.

- Useful for specifying, widely used in mathematics ;
- Links object and property.

Adapting the idea

$$\frac{t : T \qquad P[t/x]}{t : \{\ x : T \mid P\ \}} \qquad \frac{t : \{\ x : T \mid P\ \}}{t : T}$$

A simple idea

Definition

$\{x : T \mid P\}$ is the set of objects of set T verifying property P.

- Useful for specifying, widely used in mathematics ;
- Links object and property.

Adapting the idea

$$\frac{t : T \qquad p : P[t/x]}{(t, p) : \{\ x : T \mid P\ \}} \qquad \frac{t : \{\ x : T \mid P\ \}}{\texttt{proj}\ t : T}$$

From "*Predicate subtyping*" ...

PVS

- Specialized typing algorithm for subset types, generating *Type-checking conditions*.

$t : \{ x : T \mid P \}$	used as	$t : T$	ok
$t : T$	used as	$t : \{ x : T \mid P \}$	if $P[t/x]$

From "*Predicate subtyping*" ...

PVS

- Specialized typing algorithm for subset types, generating *Type-checking conditions*.

$t : \{ x : T \mid P \}$	used as	$t : T$	ok
$t : T$	used as	$t : \{ x : T \mid P \}$	if $P[t/x]$

\+ Practical success ;

From "*Predicate subtyping*" ...

PVS

- Specialized typing algorithm for subset types, generating *Type-checking conditions*.

$t : \{ x : T \mid P \}$	used as	$t : T$	ok
$t : T$	used as	$t : \{ x : T \mid P \}$	if $P[t/x]$

\+ Practical success ;

− No strong safety guarantee in PVS.

. . . to Subset coercions

1. A property-irrelevant language (Russell) with decidable typing ;

$$\frac{\Gamma \vdash t : \{\, x : T \mid P \,\}}{\Gamma \vdash \quad t : T}$$

$$\frac{\Gamma \vdash t : T \qquad \Gamma, x : T \vdash P : \texttt{Prop}}{\Gamma \vdash \; t \quad : \{\, x : T \mid P \,\}}$$

... to Subset coercions

1. A property-irrelevant language (Russell) with decidable typing ;
2. A total interpretation to Coq terms with holes ;

$$\frac{\Gamma \vdash t : \{\ x : T \mid P\ \}}{\Gamma \vdash \texttt{proj}\ t : T}$$

$$\frac{\Gamma \vdash t : T \qquad \Gamma, x : T \vdash P : \texttt{Prop}}{\Gamma \vdash (t, ?) : \{\ x : T \mid P\ \}}\ \Gamma \vdash ? : P[t/x]$$

. . . to Subset coercions

1. A property-irrelevant language (Russell) with decidable typing ;
2. A total interpretation to Coq terms with holes ;
3. A mechanism to turn the holes into proof obligations and manage them.

$$\frac{\Gamma \vdash t : \{\, x : T \mid P \,\}}{\Gamma \vdash \texttt{proj}\ t : T}$$

$$\frac{\Gamma \vdash t : T \qquad \Gamma, x : T \vdash P : \texttt{Prop} \qquad \Gamma \vdash p : P[t/x]}{\Gamma \vdash (t, p) : \{\, x : T \mid P \,\}}$$

Russell syntax

$$
\begin{array}{lcl}
x & \in & \mathcal{V} \\
 & & \\
s, t, u, v & ::= & x \\
 & | & \texttt{Set} \\
 & | & \texttt{Prop} \\
 & | & \texttt{Type}
\end{array}
$$

Russell syntax

$$
\begin{array}{rcl}
x & \in & \mathcal{V} \\
\\
s, t, u, v & ::= & x \\
 & | & \texttt{Set} \\
 & | & \texttt{Prop} \\
 & | & \texttt{Type} \\
 & | & \lambda x : s.t \\
 & | & s\ t \\
 & | & \Pi x : s.t
\end{array}
$$

RUSSELL syntax

$$
\begin{array}{lcl}
x & \in & \mathcal{V} \\
\\
s, t, u, v & ::= & x \\
 & | & \texttt{Set} \\
 & | & \texttt{Prop} \\
 & | & \texttt{Type} \\
 & | & \lambda x : s.t \\
 & | & s\ t \\
 & | & \Pi x : s.t \\
 & | & (u, v)_{\Sigma x:s.t} \\
 & | & \pi_1\ s \mid \pi_2\ s \\
 & | & \Sigma x : s.t
\end{array}
$$

RUSSELL syntax

$$
\begin{array}{lcl}
x & \in & \mathcal{V} \\
\\
s, t, u, v & ::= & x \\
& | & \texttt{Set} \\
& | & \texttt{Prop} \\
& | & \texttt{Type} \\
& | & \lambda x : s.t \\
& | & s\ t \\
& | & \Pi x : s.t \\
& | & (u, v)_{\Sigma x : s.t} \\
& | & \pi_1\ s \mid \pi_2\ s \\
& | & \Sigma x : s.t \\
& | & \{\ x : s \mid t\ \}
\end{array}
$$

Russell typing $\vdash$ and coercion $\rhd$

Calculus of Constructions with

$$\frac{\Gamma \vdash t : U \qquad \Gamma \vdash U \equiv_{\beta\pi} T : s}{\Gamma \vdash t : T}$$

RUSSELL typing $\vdash$ and coercion $\rhd$

Calculus of Constructions with

$$\frac{\Gamma \vdash t : U \qquad \Gamma \vdash U \rhd T : s}{\Gamma \vdash t : T} \qquad \frac{\Gamma \vdash T \equiv_{\beta\pi} U : s}{\Gamma \vdash T \rhd U : s}$$

Russell typing $\vdash$ and coercion $\rhd$

Calculus of Constructions with

$$\frac{\Gamma \vdash t : U \qquad \Gamma \vdash U \rhd T : s}{\Gamma \vdash t : T} \qquad \frac{\Gamma \vdash T \equiv_{\beta\pi} U : s}{\Gamma \vdash T \rhd U : s}$$

$$\frac{\Gamma \vdash U \rhd V : \texttt{Set} \qquad \Gamma, x : U \vdash P : \texttt{Prop}}{\Gamma \vdash \{\, x : U \mid P \,\} \rhd V : \texttt{Set}}$$

$$\frac{\Gamma \vdash U \rhd V : \texttt{Set} \qquad \Gamma, x : V \vdash P : \texttt{Prop}}{\Gamma \vdash U \rhd \{\, x : V \mid P \,\} : \texttt{Set}}$$

Calculus of Constructions with

$$\frac{\Gamma \vdash t : U \qquad \Gamma \vdash U \rhd T : s}{\Gamma \vdash t : T} \qquad \frac{\Gamma \vdash T \equiv_{\beta\pi} U : s}{\Gamma \vdash T \rhd U : s}$$

$$\frac{\Gamma \vdash U \rhd V : \texttt{Set} \qquad \Gamma, x : U \vdash P : \texttt{Prop}}{\Gamma \vdash \{\ x : U \mid P\ \} \rhd V : \texttt{Set}}$$

$$\frac{\Gamma \vdash U \rhd V : \texttt{Set} \qquad \color{red}{\Gamma, x : V \vdash P : \texttt{Prop}}}{\Gamma \vdash U \rhd \{\ x : V \mid P\ \} : \texttt{Set}}$$

Example $$\frac{\Gamma \vdash 0 : \mathbb{N} \qquad \Gamma \vdash \mathbb{N} \rhd \{\ x : \mathbb{N} \mid x \neq 0\ \} : \texttt{Set}}{\Gamma \vdash 0 : \{\ x : \mathbb{N} \mid x \neq 0\ \}}$$

Calculus of Constructions with

$$\frac{\Gamma \vdash t : U \qquad \Gamma \vdash U \rhd T : s}{\Gamma \vdash t : T} \qquad \frac{\Gamma \vdash T \equiv_{\beta\pi} U : s}{\Gamma \vdash T \rhd U : s}$$

$$\frac{\Gamma \vdash U \rhd V : \texttt{Set} \qquad \Gamma, x : U \vdash P : \texttt{Prop}}{\Gamma \vdash \{\ x : U \mid P\ \} \rhd V : \texttt{Set}}$$

$$\frac{\Gamma \vdash U \rhd V : \texttt{Set} \qquad \Gamma, x : V \vdash P : \texttt{Prop}}{\Gamma \vdash U \rhd \{\ x : V \mid P\ \} : \texttt{Set}}$$

Example $$\cfrac{\cfrac{\Gamma \vdash 0 : \mathbb{N} \qquad \Gamma \vdash \mathbb{N} \rhd \{\ x : \mathbb{N} \mid x \neq 0\ \} : \texttt{Set}}{\Gamma \vdash 0 : \{\ x : \mathbb{N} \mid x \neq 0\ \}}}{\Gamma \vdash ? : 0 \neq 0}$$

Calculus of Constructions with

$$\frac{\Gamma \vdash t : U \qquad \Gamma \vdash U \rhd T : s}{\Gamma \vdash t : T} \qquad \frac{\Gamma \vdash T \equiv_{\beta\pi} U : s}{\Gamma \vdash T \rhd U : s}$$

$$\frac{\Gamma \vdash U \rhd V : \texttt{Set} \qquad \Gamma, x : U \vdash P : \texttt{Prop}}{\Gamma \vdash \{\, x : U \mid P \,\} \rhd V : \texttt{Set}}$$

$$\frac{\Gamma \vdash U \rhd V : \texttt{Set} \qquad \Gamma, x : V \vdash P : \texttt{Prop}}{\Gamma \vdash U \rhd \{\, x : V \mid P \,\} : \texttt{Set}}$$

$$\frac{\Gamma \vdash U \rhd T : s_1 \qquad \Gamma, x : U \vdash V \rhd W : s_2}{\Gamma \vdash \Pi x : T.V \rhd \Pi x : U.W : s_2}$$

$$\frac{\Gamma \vdash T \rhd U : s \qquad \Gamma, x : T \vdash V \rhd W : s}{\Gamma \vdash \Sigma x : T.V \rhd \Sigma y : U.W : s} \; s \in \{\texttt{Set}, \texttt{Prop}\}$$

Calculus of Constructions with

$$\frac{\Gamma \vdash t : U \qquad \Gamma \vdash U \rhd T : s}{\Gamma \vdash t : T} \qquad \frac{\Gamma \vdash T \equiv_{\beta\pi} U : s}{\Gamma \vdash T \rhd U : s}$$

$$\frac{\Gamma \vdash U \rhd V : \texttt{Set} \qquad \Gamma, x : U \vdash P : \texttt{Prop}}{\Gamma \vdash \{\, x : U \mid P \,\} \rhd V : \texttt{Set}}$$

$$\frac{\Gamma \vdash U \rhd V : \texttt{Set} \qquad \Gamma, x : V \vdash P : \texttt{Prop}}{\Gamma \vdash U \rhd \{\, x : V \mid P \,\} : \texttt{Set}}$$

$$\frac{\Gamma \vdash U \rhd T : s_1 \qquad \Gamma, x : U \vdash V \rhd W : s_2}{\Gamma \vdash \Pi x : T.V \rhd \Pi x : U.W : s_2}$$

$$\frac{\Gamma \vdash T \rhd U : s \qquad \Gamma, x : T \vdash V \rhd W : s}{\Gamma \vdash \Sigma x : T.V \rhd \Sigma y : U.W : s} \; s \in \{\texttt{Set}, \texttt{Prop}\}$$

$\rhd$ is symmetric!

Results

Theorem (Decidability of type checking and type inference)

$\Gamma \vdash t : T$ *is decidable.*

$$\frac{\Gamma \vdash f : T \quad \Gamma \vdash T \rhd \Pi x : A.B : s \qquad \Gamma \vdash e : E \qquad \Gamma \vdash E \rhd A : s'}{\Gamma \vdash (f\ e) : B[e/x]}$$

Results

Theorem (Decidability of type checking and type inference)

$\Gamma \vdash t : T$ *is decidable.*

$$\frac{\Gamma \vdash f : T \quad \Gamma \vdash T \rhd \Pi x : A.B : s \qquad \Gamma \vdash e : E \qquad \Gamma \vdash E \rhd A : s'}{\Gamma \vdash (f\ e) : B[e/x]}$$

Coq corner

Mechanised proofs of Subject Reduction and equivalence between declarative and algorithmic presentations of the system.

The target system : CIC with metavariables

$$\frac{\Gamma \vdash_? t : T \qquad \Gamma \vdash_? p : P[t/x]}{\Gamma \vdash_? \mathbf{elt}\ T\ P\ t\ p : \{\ x : T \mid P\ \}}$$

$$\frac{\Gamma \vdash_? t : \{\ x : T \mid P\ \}}{\Gamma \vdash_? \sigma_1\ t : T} \qquad \frac{\Gamma \vdash_? t : \{\ x : T \mid P\ \}}{\Gamma \vdash_? \sigma_2\ t : P[\sigma_1\ t/x]}$$

$$\frac{\Gamma \vdash_? P : \texttt{Prop}}{\Gamma \vdash_? ?_P : P}$$

We build an interpretation $[\![\_]\!]_\Gamma$ from RUSSELL to CIC$_?$ terms.

The target system : CIC with metavariables

$$\frac{\Gamma \vdash_? t : T \qquad \Gamma \vdash_? p : P[t/x]}{\Gamma \vdash_? \textbf{elt}\ T\ P\ t\ p : \{\ x : T \mid P\ \}}$$

$$\frac{\Gamma \vdash_? t : \{\ x : T \mid P\ \}}{\Gamma \vdash_? \sigma_1\ t : T} \qquad \frac{\Gamma \vdash_? t : \{\ x : T \mid P\ \}}{\Gamma \vdash_? \sigma_2\ t : P[\sigma_1\ t/x]}$$

$$\frac{\Gamma \vdash_? P : \texttt{Prop}}{\Gamma \vdash_? ?_P : P}$$

We build an interpretation $[\![\_]\!]_\Gamma$ from RUSSELL to $\text{CIC}_?$ terms.

Our goal

If $\Gamma \vdash t : T$ then $[\![\Gamma]\!] \vdash_? [\![t]\!]_\Gamma : [\![T]\!]_\Gamma$.

Deriving explicit coercions

Interpretation of coercions

If $\Gamma \vdash T \rhd U : s$ then $\Gamma \vdash_? c[\bullet] : T \rhd U$ which implies $[\![\Gamma]\!], x : [\![T]\!]_\Gamma \vdash_? c[x] : [\![U]\!]_\Gamma$.

Deriving explicit coercions

Interpretation of coercions

If $\Gamma \vdash T \rhd U : s$ then $\Gamma \vdash_? c[\bullet] : T \rhd U$ which implies $[\![\Gamma]\!], x : [\![T]\!]_\Gamma \vdash_? c[x] : [\![U]\!]_\Gamma$.

Definition

$$\frac{T \equiv_{\beta\pi} U}{\Gamma \vdash_? \quad : T \rhd U}$$

Deriving explicit coercions

Interpretation of coercions

If $\Gamma \vdash T \rhd U : s$ then $\Gamma \vdash_? c[\bullet] : T \rhd U$ which implies $[\![\Gamma]\!], x : [\![T]\!]_\Gamma \vdash_? c[x] : [\![U]\!]_\Gamma$.

Definition

$$\frac{T \equiv_{\beta\pi} U}{\Gamma \vdash_? \bullet : T \rhd U}$$

Deriving explicit coercions

Interpretation of coercions

If $\Gamma \vdash T \rhd U : s$ then $\Gamma \vdash_? c[\bullet] : T \rhd U$ which implies $[\![\Gamma]\!], x : [\![T]\!]_\Gamma \vdash_? c[x] : [\![U]\!]_\Gamma$.

Definition

$$\frac{T \equiv_{\beta\pi} U}{\Gamma \vdash_? \bullet : T \rhd U}$$

$$\Gamma \vdash_? \quad : \{ x : T \mid P \} \rhd T$$

Deriving explicit coercions

Interpretation of coercions

If $\Gamma \vdash T \rhd U : s$ then $\Gamma \vdash_? c[\bullet] : T \rhd U$ which implies $[\![\Gamma]\!], x : [\![T]\!]_\Gamma \vdash_? c[x] : [\![U]\!]_\Gamma$.

Definition

$$\frac{T \equiv_{\beta\pi} U}{\Gamma \vdash_? \bullet : T \rhd U}$$

$$\Gamma \vdash_? \sigma_1 \bullet : \{ x : T \mid P \} \rhd T$$

Deriving explicit coercions

Interpretation of coercions

If $\Gamma \vdash T \rhd U : s$ then $\Gamma \vdash_? c[\bullet] : T \rhd U$ which implies $[\![\Gamma]\!], x : [\![T]\!]_\Gamma \vdash_? c[x] : [\![U]\!]_\Gamma$.

Definition

$$\frac{T \equiv_{\beta\pi} U}{\Gamma \vdash_? \bullet : T \rhd U}$$

$$\Gamma \vdash_? \sigma_1\ \bullet : \{\ x : T \mid P\ \} \rhd T$$

$$\Gamma \vdash_? \qquad\qquad\qquad : T \rhd \{\ x : T \mid P\ \}$$

Deriving explicit coercions

Interpretation of coercions

If $\Gamma \vdash T \rhd U : s$ then $\Gamma \vdash_? c[\bullet] : T \rhd U$ which implies $[\![\Gamma]\!], x : [\![T]\!]_\Gamma \vdash_? c[x] : [\![U]\!]_\Gamma$.

Definition

$$\frac{T \equiv_{\beta\pi} U}{\Gamma \vdash_? \bullet : T \rhd U}$$

$$\Gamma \vdash_? \sigma_1 \bullet : \{ x : T \mid P \} \rhd T$$

$$\Gamma \vdash_? \mathbf{elt}\ \_\ \_\ \bullet\ ?_{[\![P]\!]_{\Gamma, x:T}[\bullet/x]} : T \rhd \{ x : T \mid P \}$$

Deriving explicit coercions

Interpretation of coercions

If $\Gamma \vdash T \rhd U : s$ then $\Gamma \vdash_? c[\bullet] : T \rhd U$ which implies $[\![\Gamma]\!], x : [\![T]\!]_\Gamma \vdash_? c[x] : [\![U]\!]_\Gamma$.

Definition

$$\frac{T \equiv_{\beta\pi} U}{\Gamma \vdash_? \bullet : T \rhd U}$$

$$\Gamma \vdash_? \sigma_1 \bullet : \{ x : T \mid P \} \rhd T$$

$$\Gamma \vdash_? \textbf{elt} \_ \_ \bullet\ ?_{[\![P]\!]_{\Gamma, x:T}[\bullet/x]} : T \rhd \{ x : T \mid P \}$$

Example

$$\frac{\Gamma \vdash_? 0 : \mathbb{N} \qquad \Gamma \vdash_? \textbf{elt} \_ \_ \bullet\ ?_{(x \neq 0)[\bullet/x]} : \mathbb{N} \rhd \{ x : \mathbb{N} \mid x \neq 0 \}}{\Gamma \vdash_? \textbf{elt} \_ \_ 0\ ?_{0 \neq 0} : \{ x : \mathbb{N} \mid x \neq 0 \}}$$

Interpretation of terms

Example (Application)

$$\frac{\Gamma \vdash f : T \quad \Gamma \vdash T \rhd \Pi x : V.W : s \qquad \Gamma \vdash u : U \qquad \Gamma \vdash U \rhd V : s'}{\Gamma \vdash (f\ u) : W[u/x]}$$

$$\begin{array}{rcl} [\![f\ u]\!]_\Gamma & \triangleq & \textbf{let}\ \pi = \textbf{coerce}_\Gamma\ T\ (\Pi x : V.W)\ \textbf{in} \\ & & \textbf{let}\ c = \textbf{coerce}_\Gamma\ U\ V\ \textbf{in} \\ & & (\pi[[\![f]\!]_\Gamma])\ (c[[\![u]\!]_\Gamma]) \end{array}$$

Theorem (Soundness)

If $\Gamma \vdash t : T$ *then* $[\![\Gamma]\!] \vdash_? [\![t]\!]_\Gamma : [\![T]\!]_\Gamma$.

Theoretical matters . . .

$\vdash_?$'s equational theory:

$$
\begin{array}{llcll}
(\beta) & (\lambda x : X.e)\ v & \equiv & e[v/x] & \\
(\pi_i) & \pi_i\ (e_1, e_2)_T & \equiv & e_i & \\
(\sigma_i) & \sigma_i\ (\mathbf{elt}\ E\ P\ e_1\ e_2) & \equiv & e_i & \\
(\eta) & (\lambda x : X.e\ x) & \equiv & e & \text{if } x \notin FV(e) \\
(\mathrm{SP}) & \mathbf{elt}\ E\ P\ (\sigma_1\ e)\ (\sigma_2\ e) & \equiv & e &
\end{array}
$$

$\vdash_?$'s equational theory:

$$
\begin{array}{llcll}
(\beta) & (\lambda x : X.e)\ v & \equiv & e[v/x] & \\
(\pi_i) & \pi_i\ (e_1, e_2)_T & \equiv & e_i & \\
(\sigma_i) & \sigma_i\ (\mathbf{elt}\ E\ P\ e_1\ e_2) & \equiv & e_i & \\
(\eta) & (\lambda x : X.e\ x) & \equiv & e & \text{if } x \notin FV(e) \\
(\mathrm{SP}) & \mathbf{elt}\ E\ P\ (\sigma_1\ e)\ (\sigma_2\ e) & \equiv & e & \\
(\mathrm{PI}) & \mathbf{elt}\ E\ P\ t\ p & \equiv & \mathbf{elt}\ E\ P\ t'\ p' & \text{if } t \equiv t'
\end{array}
$$

$\Rightarrow$ Proof Irrelevance

Theoretical matters ...

$\vdash_?$'s equational theory:

$$
\begin{array}{llcll}
(\beta) & (\lambda x : X.e)\ v & \equiv & e[v/x] & \\
(\pi_i) & \pi_i\ (e_1, e_2)_T & \equiv & e_i & \\
(\sigma_i) & \sigma_i\ (\textbf{elt}\ E\ P\ e_1\ e_2) & \equiv & e_i & \\
(\eta) & (\lambda x : X.e\ x) & \equiv & e & \text{if } x \notin FV(e) \\
(\mathrm{SP}) & \textbf{elt}\ E\ P\ (\sigma_1\ e)\ (\sigma_2\ e) & \equiv & e & \\
(\mathrm{PI}) & \textbf{elt}\ E\ P\ t\ p & \equiv & \textbf{elt}\ E\ P\ t'\ p' & \text{if } t \equiv t'
\end{array}
$$

$\Rightarrow$ Proof Irrelevance

... have practical effects

Difficulty to reason on code: $\textbf{elt}\ T\ P\ x\ p_1 \not\equiv \textbf{elt}\ T\ P\ x\ p_2$ where $p_1, p_2 : P\ x$.

Inductive types

Different representations

vector $n \triangleq \{ \ x$: list $A \mid$ *length* $x = n \ \}$ or
vector $n \triangleq$ vnil : vector $0 \mid$ vcons : $A \rightarrow \forall n,$ vector $n \rightarrow$ vector $(\mathsf{S}\ n)$?

Inductive types

Different representations

vector $n \triangleq \{ x : \text{list } A \mid \textit{length } x = n \}$ or

vector $n \triangleq$ vnil : vector 0 | vcons : $A \to \forall n,$ vector $n \to$ vector $(\mathsf{S}\ n)$?

$$\frac{\Gamma \vdash v : \text{vector } x \qquad \dfrac{\Gamma \vdash x = y : \texttt{Prop}}{\Gamma \vdash \text{vector } x \rhd \text{vector } y : \texttt{Set}}}{\Gamma \vdash v : \text{vector } y}$$

Architecture

Wrap around COQ's vernacular commands (`Definition`, `Fixpoint`, `Lemma`, ...).

Architecture

Wrap around COQ's vernacular commands (`Definition`, `Fixpoint`, `Lemma`, ...).

1. Use the COQ parser.

$$\texttt{Program Definition}\ f\ :\ T\ \ :=\ t\ \qquad\qquad\qquad .$$

Architecture

Wrap around COQ's vernacular commands (`Definition`, `Fixpoint`, `Lemma`, ...).

1. Use the COQ parser.
2. Typecheck $\Gamma \vdash t : T$ and generate $[\![\Gamma]\!] \vdash_? [\![t]\!]_\Gamma : [\![T]\!]_\Gamma$;

`Program Definition` $f\ : [\![T]\!]_\Gamma := [\![t]\!]_\Gamma$.

Architecture

Wrap around COQ's vernacular commands (`Definition`, `Fixpoint`, `Lemma`, ...).

1. Use the COQ parser.
2. Typecheck $\Gamma \vdash t : T$ and generate $[\![\Gamma]\!] \vdash_? [\![t]\!]_\Gamma : [\![T]\!]_\Gamma$;
3. Interactive proving of obligations ;

`Program Definition` $f \; : [\![T]\!]_\Gamma := [\![t]\!]_\Gamma +$ obligations.

Architecture

Wrap around COQ's vernacular commands (`Definition`, `Fixpoint`, `Lemma`, ...).

1. Use the COQ parser.
2. Typecheck $\Gamma \vdash t : T$ and generate $[\![\Gamma]\!] \vdash_? [\![t]\!]_\Gamma : [\![T]\!]_\Gamma$;
3. Interactive proving of obligations ;
4. Final definition.

`Definition` $f \; : [\![T]\!]_\Gamma := [\![t]\!]_\Gamma +$ obligations.

Architecture

Wrap around COQ's vernacular commands (`Definition`, `Fixpoint`, `Lemma`, ...).

1. Use the COQ parser.
2. Typecheck $\Gamma \vdash t : T$ and generate $[\![\Gamma]\!] \vdash_? [\![t]\!]_\Gamma : [\![T]\!]_\Gamma$;
3. Interactive proving of obligations ;
4. Final definition.

   Restriction We assume $\Gamma \vdash_{CCI} [\![T]\!]_\Gamma : s$.

`Definition` $f \;:\; [\![T]\!]_\Gamma := [\![t]\!]_\Gamma +$ obligations.

Hello world: Euclidean division

DEMO

Pattern-matching revisited

Put logic into the terms.

Let $e : \mathbb{N}$:

$$\begin{array}{lll} \textbf{match } e & \textbf{return} & T \textbf{ with} \\ \mid \mathsf{S}\ n \Rightarrow & & t_1 \\ \mid 0 \Rightarrow & & t_2 \\ \textbf{end} & & \end{array}$$

Pattern-matching revisited

Put logic into the terms.

Let $e : \mathbb{N}$:

$$\begin{array}{ll}
\textbf{match } e \textbf{ as } t & \textbf{return } t = e \rightarrow T \textbf{ with} \\
\mid \mathsf{S}\ n \Rightarrow & \textbf{fun } (H : \mathsf{S}\ n = e) \Rightarrow t_1 \\
\mid 0 \Rightarrow & \textbf{fun } (H : 0 = e) \Rightarrow t_2 \\
\textbf{end} & (\textbf{refl_equal } e)
\end{array}$$

Pattern-matching revisited

Put logic into the terms.

Further refinements

- Each branch typed only once ;

Let $e : \mathbb{N}$:

$$
\begin{array}{ll}
\textbf{match } e \textbf{ as } t & \textbf{return } t = e \rightarrow T \textbf{ with} \\
\mid \mathsf{S}\ (\mathsf{S}\ n) \Rightarrow & \textbf{fun } (H : \mathsf{S}\ (\mathsf{S}\ n) = e) \Rightarrow t_1 \\
\mid n \Rightarrow & \textbf{fun } (H : n = e) \Rightarrow t_2 \\
\textbf{end} & (\textbf{refl\_equal } e)
\end{array}
$$

Pattern-matching revisited

Put logic into the terms.

Further refinements

- Each branch typed only once ;

Let $e : \mathbb{N}$:

$$
\begin{array}{ll}
\textbf{match } e \textbf{ as } t & \textbf{return } t = e \rightarrow T \textbf{ with} \\
\mid \mathsf{S}\ (\mathsf{S}\ n) \Rightarrow & \textbf{fun } (H : \mathsf{S}\ (\mathsf{S}\ n) = e) \Rightarrow t_1 \\
\mid \mathsf{S}\ 0 \Rightarrow & \textbf{fun } (H : \mathsf{S}\ 0 = e) \Rightarrow t_2 \\
\mid 0 \Rightarrow & \textbf{fun } (H : 0 = e) \Rightarrow t_2 \\
\textbf{end} & (\textbf{refl\_equal } e)
\end{array}
$$

Pattern-matching revisited

Put logic into the terms.

Further refinements

- Each branch typed only once ;
- Add inequalities for intersecting patterns ;

Let $e : \mathbb{N}$:

$$\begin{array}{ll}
\textbf{match } e \textbf{ as } t & \textbf{return } t = e \rightarrow T \textbf{ with} \\
\mid \mathsf{S}\ (\mathsf{S}\ n) \Rightarrow & \textbf{fun } (H : \mathsf{S}\ (\mathsf{S}\ n) = e) \Rightarrow t_1 \\
\mid n \Rightarrow & \textbf{fun } (H : n = e) \Rightarrow \textbf{let } H' : \forall n', n \neq \mathsf{S}\ (\mathsf{S}\ n') \textbf{ in } t_2 \\
\textbf{end} & (\textbf{refl_equal } e)
\end{array}$$

Pattern-matching revisited

Put logic into the terms.

Further refinements

- Each branch typed only once ;
- Add inequalities for intersecting patterns ;
- Generalized to dependent inductive types.

Let e : vector n:

match e **return** T **with**
| vnil $\Rightarrow$ t_1
| vcons $x\ n'\ v'$ $\Rightarrow$ t_2
end

Pattern-matching revisited

Put logic into the terms.

Further refinements

- Each branch typed only once ;
- Add inequalities for intersecting patterns ;
- Generalized to dependent inductive types.

Let e : vector n:

$$\textbf{match } e \textbf{ as } t \textbf{ in } \text{vector } n' \textbf{ return } n' = n \to t \simeq e \to T \textbf{ with}$$
$$\mid \text{vnil} \Rightarrow \textbf{fun } (H : 0 = n)(Hv : \text{vnil} \simeq e) \Rightarrow t_1$$
$$\mid \text{vcons } x\ n'\ v' \Rightarrow \textbf{fun } (H : \text{S } n' = n)(Hv : \text{vcons } x\ n'\ v' \simeq e) \Rightarrow t_2$$
$$\textbf{end}(\text{refl\_equal } n)(\text{JMeq\_refl } e)$$

Sugar

Obligations

Unresolved implicits (_) are turned into obligations, à la `refine`.

Bang

$! \triangleq$ (`False_rect _ _`) where `False_rect` : $\forall A$: Type, False $\rightarrow A$. It corresponds to ML's `assert(false)`.

$$\textbf{match } 0 \textbf{ with } 0 \Rightarrow 0 \mid n \Rightarrow \,! \textbf{ end}$$

Sugar

Obligations

Unresolved implicits (_) are turned into obligations, à la `refine`.

Bang

$! \triangleq (\texttt{False\_rect}\ \_\ \_)$ where $\texttt{False\_rect} : \forall A : \texttt{Type}, \texttt{False} \rightarrow A$. It corresponds to ML's `assert(false)`.

$$\textbf{match}\ 0\ \textbf{with}\ 0 \Rightarrow 0 \mid n \Rightarrow !\ \textbf{end}$$

Destruction

Let $\textbf{let} \mid p\ := \ t\textbf{in}\ e \triangleq \textbf{match}\ t\ \textbf{with}\ p \Rightarrow e\ \textbf{end}$. p can be an arbitrary pattern.

Recursion

Support for well-founded recursion and measures.

$$\texttt{Program Fixpoint}\ f\ (a : \mathbb{N})\ \{\textsf{\textbf{wf}} < a\} : \mathbb{N} := b.$$

Recursion

Support for well-founded recursion and measures.

$$\texttt{Program Fixpoint}\ f\ (a : \mathbb{N})\ \{\textbf{wf} < a\} : \mathbb{N} := b.$$

$$\frac{\begin{array}{lcl} a & : & \mathbb{N} \\ f & : & \{x : \mathbb{N} \mid x < a\} \to \mathbb{N} \end{array}}{\begin{array}{lcl} b & : & \mathbb{N} \end{array}}$$

DEMO

Conclusion

Our contributions

- A more flexible programming language, (almost) conservative over CIC, integrated with the existing environment and a formal justification of "*Predicate subtyping*".
- A tool to make programming in COQ using the full language possible, which can effectively be used for non-trivial developments.

Ongoing and future work

- Reasoning support through tactics
- Implementation of proof-irrelevance in COQ's kernel
- Overloading support through a typeclass mechanism.

The End

http://www.lri.fr/~sozeau/research/russell.en.html