

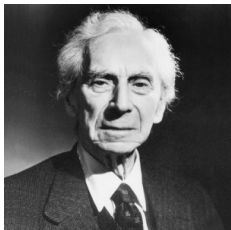
# Coq: An Environment for Proving and Programming

MATTHIEU SOZEAU

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

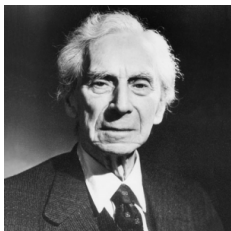
Ph.D student's seminar  
January 26th 2009  
LIFO - Orléans





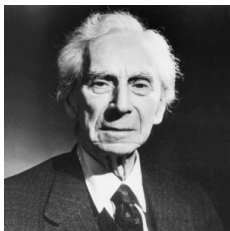
- ▶ **Russell's paradox (1901)** Let  $R \triangleq \{X \mid X \notin X\}$ . Then  $R \in R \leftrightarrow R \notin R$ , a **contradiction!**

# From Bertrand Russell to Coq (1901-today)



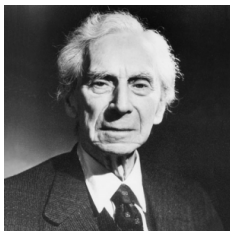
- ▶ **Russell's paradox (1901)** Let  $R \triangleq \{X \mid X \notin X\}$ . Then  $R \in R \leftrightarrow R \notin R$ , a **contradiction!**
- ▶ **Russell's theory of types (1908)** rules out  $R$ : there are sets, sets of sets and so on.

# From Bertrand Russell to Coq (1901-today)



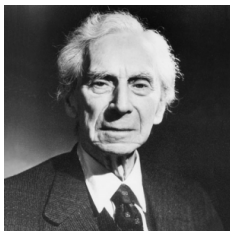
- ▶ **Russell's paradox (1901)** Let  $R \triangleq \{X \mid X \notin X\}$ . Then  $R \in R \leftrightarrow R \notin R$ , a **contradiction!**
- ▶ **Russell's theory of types (1908)** rules out  $R$ : there are sets, sets of sets and so on.
- ▶ **Church's simple theory of types (1940)** for the  $\lambda$ -calculus.

# From Bertrand Russell to CoQ (1901-today)



- ▶ **Russell's paradox (1901)** Let  $R \triangleq \{X \mid X \notin X\}$ . Then  $R \in R \leftrightarrow R \notin R$ , a **contradiction!**
- ▶ **Russell's theory of types (1908)** rules out  $R$ : there are sets, sets of sets and so on.
- ▶ **Church's simple theory of types (1940)** for the  $\lambda$ -calculus.
- ▶ **The Curry-Howard isomorphism (1958,1969)** Verifying a mathematical **proof** is **equivalent** to type-checking a **program**.

# From Bertrand Russell to CoQ (1901-today)



- ▶ **Russell's paradox (1901)** Let  $R \triangleq \{X \mid X \notin X\}$ . Then  $R \in R \leftrightarrow R \notin R$ , a **contradiction!**
- ▶ **Russell's theory of types (1908)** rules out  $R$ : there are sets, sets of sets and so on.
- ▶ **Church's simple theory of types (1940)** for the  $\lambda$ -calculus.
- ▶ **The Curry-Howard isomorphism (1958,1969)** Verifying a mathematical **proof** is **equivalent** to type-checking a **program**.
- ▶ **Type Theory (1970)** **Mechanized** formal reasoning.

# Simply-typed $\lambda$ -calculus

$x, y, z \in \mathcal{V}$

$n, m \in \mathbb{N}$

$s, t, u, v ::= x$   
          |  $\lambda x : \tau, t$   
          |  $s t$   
          | **n**

# Simply-typed $\lambda$ -calculus

$x, y, z \in \mathcal{V}$

$n, m \in \mathbb{N}$

$s, t, u, v ::= x$   
          |  $\lambda x : \tau, t$   
          |  $s t$   
          | **n**

$\tau, \sigma, \rho ::= \text{nat}$   
          |  $\sigma \rightarrow \rho$

# Simply-typed $\lambda$ -calculus

$x, y, z \in \mathcal{V}$

$n, m \in \mathbb{N}$

$s, t, u, v ::= x$   
          |  $\lambda x : \tau, t$   
          |  $s t$   
          | **n**

$\tau, \sigma, \rho ::= \mathbf{nat}$   
          |  $\sigma \rightarrow \rho$

$\Gamma, \Delta ::= \square$   
          |  $\Gamma, x : \tau$

# Simply-typed $\lambda$ -calculus

$x, y, z \in \mathcal{V}$

$n, m \in \mathbb{N}$

$s, t, u, v ::= x$   
|  $\lambda x : \tau, t$   
|  $s t$   
| **n**

$\tau, \sigma, \rho ::= \mathbf{nat}$   
|  $\sigma \rightarrow \rho$

$\Gamma, \Delta ::= []$   
|  $\Gamma, x : \tau$

**$\beta$ -reduction:**  $(\lambda x : \tau, t) u \rightarrow_{\beta} t[x := u]$

**VAR**  $\frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau}$

**ABS**  $\frac{\Gamma, x : \sigma \vdash t : \tau}{\Gamma \vdash \lambda x : \sigma, t : \sigma \rightarrow \tau}$

**APP**  $\frac{\Gamma \vdash f : \sigma \rightarrow \rho \quad \Gamma \vdash e : \sigma}{\Gamma \vdash f e : \rho}$

**NAT**  $\frac{}{\Gamma \vdash \mathbf{n} : \mathbf{nat}}$

# Example

$\Gamma \triangleq \text{plus} : \text{nat} \rightarrow \text{nat} \rightarrow \text{nat}$

---

---

---

$$\overline{\Gamma \vdash \lambda x : \text{nat}, \text{plus } x \ 1 : \text{nat} \rightarrow \text{nat}}$$
$$(\lambda x : \text{nat}, \text{plus } x \ 1) \ 0 \rightarrow_{\beta} \text{plus } 0 \ 1 \rightarrow 1$$

# Example

$\Gamma \triangleq \text{plus} : \text{nat} \rightarrow \text{nat} \rightarrow \text{nat}$

---

---

$$\frac{\Gamma, x : \text{nat} \vdash \text{plus } x \ 1 : \text{nat}}{\Gamma \vdash \lambda x : \text{nat}, \text{plus } x \ 1 : \text{nat} \rightarrow \text{nat}}$$

$(\lambda x : \text{nat}, \text{plus } x \ 1) \ 0 \rightarrow_{\beta} \text{plus } 0 \ 1 \rightarrow 1$

# Example

$\Gamma \triangleq \text{plus} : \text{nat} \rightarrow \text{nat} \rightarrow \text{nat}$

---

---

$$\frac{\Gamma, x : \text{nat} \vdash \text{plus } x : \text{nat} \rightarrow \text{nat} \quad \Gamma, x : \text{nat} \vdash \mathbf{1} : \text{nat}}{\Gamma, x : \text{nat} \vdash \text{plus } x \mathbf{1} : \text{nat}}$$

---

$$\Gamma \vdash \lambda x : \text{nat}, \text{plus } x \mathbf{1} : \text{nat} \rightarrow \text{nat}$$

$(\lambda x : \text{nat}, \text{plus } x \mathbf{1}) \mathbf{0} \rightarrow_{\beta} \text{plus } \mathbf{0} \mathbf{1} \rightarrow \mathbf{1}$

# Example

$\Gamma \triangleq \text{plus} : \text{nat} \rightarrow \text{nat} \rightarrow \text{nat}$

$$\frac{\frac{\frac{\Gamma, x : \text{nat} \vdash \text{plus} : \text{nat} \rightarrow \text{nat} \rightarrow \text{nat}}{\Gamma, x : \text{nat} \vdash \text{plus } x : \text{nat} \rightarrow \text{nat}} \quad \frac{\Gamma, x : \text{nat} \vdash x : \text{nat}}{\Gamma, x : \text{nat} \vdash \mathbf{1} : \text{nat}}}{\Gamma, x : \text{nat} \vdash \text{plus } x \mathbf{1} : \text{nat}}}{\Gamma \vdash \lambda x : \text{nat}, \text{plus } x \mathbf{1} : \text{nat} \rightarrow \text{nat}}$$

$(\lambda x : \text{nat}, \text{plus } x \mathbf{1}) \mathbf{0} \rightarrow_{\beta} \text{plus } \mathbf{0} \mathbf{1} \rightarrow \mathbf{1}$

# More datatypes and computations

- ▶ **Datatypes:** booleans, naturals, trees...

$$\frac{\Gamma \vdash b : \mathbf{bool} \quad \Gamma \vdash t : \sigma \quad \Gamma \vdash u : \sigma}{\Gamma \vdash \mathbf{if } b \mathbf{ then } t \mathbf{ else } u : \sigma}$$

$\mathbf{if } \mathbf{true} \mathbf{ then } t \mathbf{ else } u \rightarrow_{\mathbf{if}} t$

$\mathbf{if } \mathbf{false} \mathbf{ then } t \mathbf{ else } u \rightarrow_{\mathbf{if}} u$

# More datatypes and computations

- ▶ **Datatypes:** booleans, naturals, trees...

$$\frac{\Gamma \vdash b : \text{bool} \quad \Gamma \vdash t : \sigma \quad \Gamma \vdash u : \sigma}{\Gamma \vdash \text{if } b \text{ then } t \text{ else } u : \sigma}$$

`if true then t else u`  $\rightarrow_{\text{if}}$  `t`  
`if false then t else u`  $\rightarrow_{\text{if}}$  `u`

- ▶ **Computations:** recursion.

$$\frac{\Gamma, f : \sigma \rightarrow \rho \vdash t : \sigma \rightarrow \rho}{\Gamma \vdash \text{fix } f := t : \sigma \rightarrow \rho} \quad \text{fix } f := t \rightarrow_{\text{fix}} t[f := \text{fix } f := t]$$

```
fix plus := λn m : nat,  
  match n with  
  | 0 ⇒ m  
  | S n' ⇒ S (plus n' m)  
end : nat → nat → nat
```

# Moving to F: parametric polymorphism

$$\begin{array}{l} s, t, u, v ::= \dots \\ \quad | \Lambda \alpha. t \\ \quad | s \tau \end{array} \qquad \beta\text{-reduction: } (\Lambda \alpha, t) \tau \rightarrow_{\beta} t[\alpha := \tau]$$
$$\text{ABSTY} \frac{\Gamma, \alpha \vdash t : \tau}{\Gamma \vdash \Lambda \alpha. t : \forall \alpha, \tau}$$
$$\begin{array}{l} \tau, \sigma, \rho ::= \dots \\ \quad | \alpha \\ \quad | \forall \alpha, \tau \end{array} \qquad \text{APPTY} \frac{\Gamma \vdash f : \forall \alpha, \tau}{\Gamma \vdash f \sigma : \tau[\alpha := \sigma]}$$

$$\Gamma \vdash \text{id} \triangleq (\Lambda \alpha, \lambda x : \alpha, x) : \forall \alpha, \alpha \rightarrow \alpha$$

$$\Gamma \vdash \text{id nat} : \text{nat} \rightarrow \text{nat}$$

$$\Lambda \alpha \beta \gamma, \lambda f : \alpha \rightarrow \beta \rightarrow \gamma, \lambda y : \beta, \lambda x : \alpha, (f \ x \ y)$$

# Polymorphic datatypes

- ▶ Generic containers:

**Inductive** `option`  $\alpha :=$

| **None** : `option`  $\alpha$   
| **Some** :  $\alpha \rightarrow$  `option`  $\alpha$

**Inductive** `list`  $\alpha :=$

| **nil** : `list`  $\alpha$   
| **cons** :  $\alpha \rightarrow$  `list`  $\alpha \rightarrow$  `list`  $\alpha$

- ▶ Reusable functionality:

**default** :  $\forall \alpha, \alpha \rightarrow$  `option`  $\alpha \rightarrow \alpha$

**map** :  $\forall \alpha \beta, (\alpha \rightarrow \beta) \rightarrow$  `list`  $\alpha \rightarrow$  `list`  $\beta$

# The Curry-Howard Isomorphism

The two following interpretations are equivalent:

- ▶ `id` is a term of type  $\forall\alpha, \alpha \rightarrow \alpha$ .
- ▶ `id` is a proof of the theorem: for all propositions  $\alpha$ ,  $\alpha$  implies  $\alpha$ .

- ▶ **Terms** represent **programs** as well as **proof witnesses**.
- ▶ **Types** represent **specifications** as well as **logical statements**.

**Terms**

1, 2...

**Types**

`nat`

**Formulas**

`inhabited nat`

# The Curry-Howard Isomorphism

The two following interpretations are equivalent:

- ▶ `id` is a term of type  $\forall\alpha, \alpha \rightarrow \alpha$ .
- ▶ `id` is a proof of the theorem: for all propositions  $\alpha$ ,  $\alpha$  implies  $\alpha$ .

- ▶ **Terms** represent **programs** as well as **proof witnesses**.
- ▶ **Types** represent **specifications** as well as **logical statements**.

**Terms**

1, 2...

|

**Types**

nat

unit

**Formulas**

inhabited nat

True,  $\top$

# The Curry-Howard Isomorphism

The two following interpretations are equivalent:

- ▶ `id` is a term of type  $\forall\alpha, \alpha \rightarrow \alpha$ .
- ▶ `id` is a proof of the theorem: for all propositions  $\alpha$ ,  $\alpha$  implies  $\alpha$ .

- ▶ **Terms** represent **programs** as well as **proof witnesses**.
- ▶ **Types** represent **specifications** as well as **logical statements**.

Terms	Types	Formulas
<code>1, 2...</code>	<code>nat</code>	<code>inhabited nat</code>
<code>!</code>	<code>unit</code>	<code>True, <math>\top</math></code>
	<code>void</code>	<code>False, <math>\perp</math></code>

# The Curry-Howard Isomorphism

The two following interpretations are equivalent:

- ▶ `id` is a term of type  $\forall \alpha, \alpha \rightarrow \alpha$ .
- ▶ `id` is a proof of the theorem: for all propositions  $\alpha$ ,  $\alpha$  implies  $\alpha$ .

- ▶ **Terms** represent **programs** as well as **proof witnesses**.
- ▶ **Types** represent **specifications** as well as **logical statements**.

Terms	Types	Formulas
$1, 2 \dots$	<code>nat</code>	inhabited <code>nat</code>
<code>!</code>	<code>unit</code>	<code>True</code> , $\top$
	<code>void</code>	<code>False</code> , $\perp$
$(x, y)$	$\alpha \times \beta$	$\alpha \wedge \beta$

# The Curry-Howard Isomorphism

The two following interpretations are equivalent:

- ▶ `id` is a term of type  $\forall\alpha, \alpha \rightarrow \alpha$ .
- ▶ `id` is a proof of the theorem: for all propositions  $\alpha$ ,  $\alpha$  implies  $\alpha$ .

- ▶ **Terms** represent **programs** as well as **proof witnesses**.
- ▶ **Types** represent **specifications** as well as **logical statements**.

Terms	Types	Formulas
<code>1, 2...</code>	<code>nat</code>	<code>inhabited nat</code>
<code>!</code>	<code>unit</code>	<code>True, <math>\top</math></code>
	<code>void</code>	<code>False, <math>\perp</math></code>
<code>(x, y)</code>	<code><math>\alpha \times \beta</math></code>	<code><math>\alpha \wedge \beta</math></code>
<code>left x, right y</code>	<code><math>\alpha + \beta</math></code>	<code><math>\alpha \vee \beta</math></code>

# The Curry-Howard Isomorphism

The two following interpretations are equivalent:

- ▶ **id** is a term of type  $\forall\alpha, \alpha \rightarrow \alpha$ .
- ▶ **id** is a proof of the theorem: for all propositions  $\alpha$ ,  $\alpha$  implies  $\alpha$ .

- ▶ **Terms** represent **programs** as well as **proof witnesses**.
- ▶ **Types** represent **specifications** as well as **logical statements**.

Terms	Types	Formulas
$1, 2 \dots$	<b>nat</b>	<b>inhabited nat</b>
<b>I</b>	<b>unit</b>	<b>True, <math>\top</math></b>
	<b>void</b>	<b>False, <math>\perp</math></b>
$(x, y)$	$\alpha \times \beta$	$\alpha \wedge \beta$
<b>left</b> $x$ , <b>right</b> $y$	$\alpha + \beta$	$\alpha \vee \beta$
$\lambda x : \alpha, t$	$\alpha \rightarrow \beta$	$\alpha \Rightarrow \beta$

# Dependent types

How to talk about **objects** in types, e.g:  $\forall x : \text{nat}, x = 0 \vee x > 0$

$$\begin{array}{l} \tau, \sigma, \rho ::= \dots \\ | \quad t \\ | \quad \Pi x : \alpha, \tau \end{array}$$

$$\frac{\Gamma, x : \sigma \vdash t : \tau}{\Gamma \vdash \lambda x : \sigma, t : \Pi x : \sigma, \tau}$$

$$\frac{\Gamma \vdash f : \Pi x : \sigma, \tau \quad \Gamma \vdash e : \sigma}{\Gamma \vdash fe : \tau[x := e]}$$

$$\frac{\Gamma \vdash t : \tau \quad \tau \equiv_{\beta} \sigma}{\Gamma \vdash t : \sigma}$$

## DEMO

- ▶ A higher-order, polymorphic logic:

$$\forall A (P Q : A \rightarrow \mathbf{Prop}), (\forall x, P x \leftrightarrow Q x) \rightarrow \forall x, P x \rightarrow Q x$$

$$\forall x y : \mathbb{N}, x + y = y + x$$

- ▶ A higher-order, polymorphic logic:

$$\forall A (P Q : A \rightarrow \mathbf{Prop}), (\forall x, P x \leftrightarrow Q x) \rightarrow \forall x, P x \rightarrow Q x$$

$$\forall x y : \mathbb{N}, x + y = y + x$$

- ▶ A purely functional programming language with dependent types:

$$\text{div} : \forall (a : \text{nat}) (b : \text{nat} \mid b \neq 0), \\ \{ (q, r) : \text{nat} \times \text{nat} \mid a = b \times q + r \wedge r < b \}$$

- ▶ A higher-order, polymorphic logic:

$$\forall A (P Q : A \rightarrow \mathbf{Prop}), (\forall x, P x \leftrightarrow Q x) \rightarrow \forall x, P x \rightarrow Q x$$

$$\forall x y : \mathbb{N}, x + y = y + x$$

- ▶ A purely functional programming language with dependent types:

$$\text{div} : \forall (a : \text{nat}) (b : \text{nat} \mid b \neq 0), \\ \{ (q, r) : \text{nat} \times \text{nat} \mid a = b \times q + r \wedge r < b \}$$

**Problem:** writing such programs is difficult.

# Programming with tactics

**Lemma** `eucl_dev` :  $\forall n, n > 0 \rightarrow \forall m:\text{nat},$   
 $\{ (q, r) : \text{nat} \times \text{nat} \mid n > r \wedge m = q \times n + r \}.$

**Proof.**

```
intros b H a; pattern a; apply gt_wf_rec; intros n H0.
elim (le_gt_dec b n).
intro lebn.
case (H0 (n - b)); auto with arith.
intros [q r] [g e].
 $\exists$  (S q, r); simpl; auto with arith.
elim plus_assoc.
elim e; auto with arith.
intros gtbn.
 $\exists$  (0, n); simpl; auto with arith.
```

**Qed.**

# Programming directly

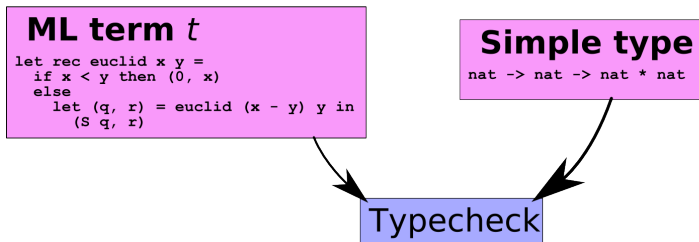
```
λ b (H : b > 0) a,  
gt_wf_rec a (λ n, { (q, r) : nat × nat | b > r ∧ n = q × b + r })  
  (λ n  
    (H0 : Π m : nat, n > m → { (q, r) : nat × nat | b > r ∧ m = q × b + r })),  
  sumbool_rec  
    (λ _ : {b ≤ n} + {b > n},  
      { (q, r) : nat × nat | b > r ∧ n = q × b + r })  
    (λ lebn : b ≤ n,  
      let (x, x0) := H0 (n - b) (lt_minus n b lebn H) in  
      (let (q, r) as p  
        return  
          ((let (q, r) := p in b > r ∧ n - b = q × b + r) →  
            { (q, r) : nat × nat | b > r ∧ n = q × b + r }) := x in  
      λ y : b > r ∧ n - b = q × b + r,  
      match y with  
      | conj g e ⇒  
        elt  
          (eq_ind (b + (q × b + r)) (λ n0, b > r ∧ n = n0)  
            (eq_ind (n - b) (λ n0, b > r ∧ n = b + n0)  
              (conj g (le_plus_minus b n lebn))  
                (q × b + r) e) (b + q × b + r) (plus_assoc b (q × b) r))  
        end) x0) (λ gtbn : b > n, elt (conj gtbn refl))  
    (le_gt_dec b n))
```

We can write programs as usual and still give them rich types.

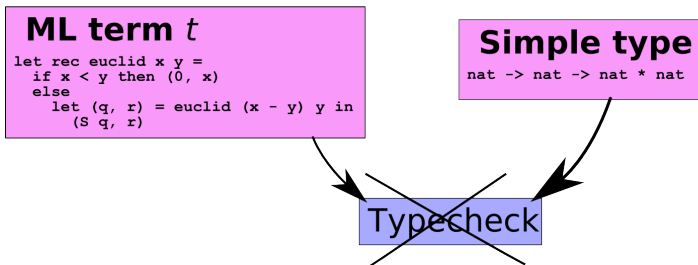
## ML term $t$

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

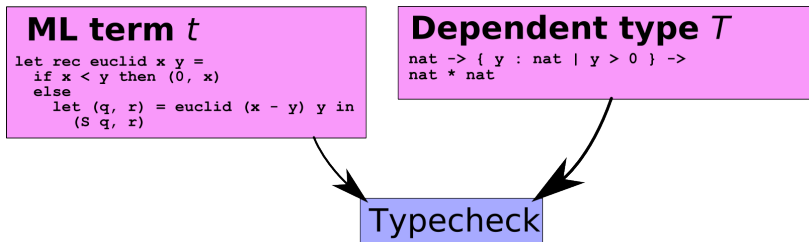
# The Big Picture



# The Big Picture



# The Big Picture



# The Big Picture

## 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)
```

## Dependent type $T$

```
nat -> { y : nat | y > 0 } ->  
nat * nat
```

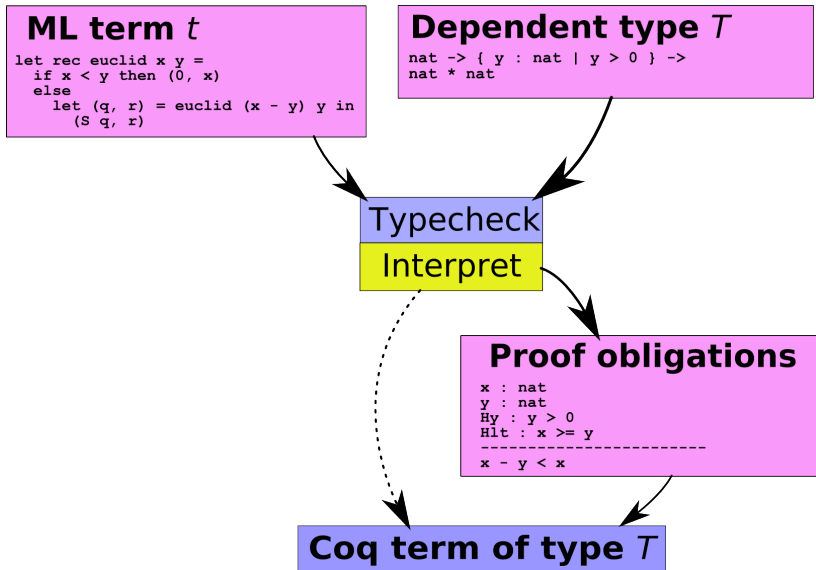
Typecheck

Interpret

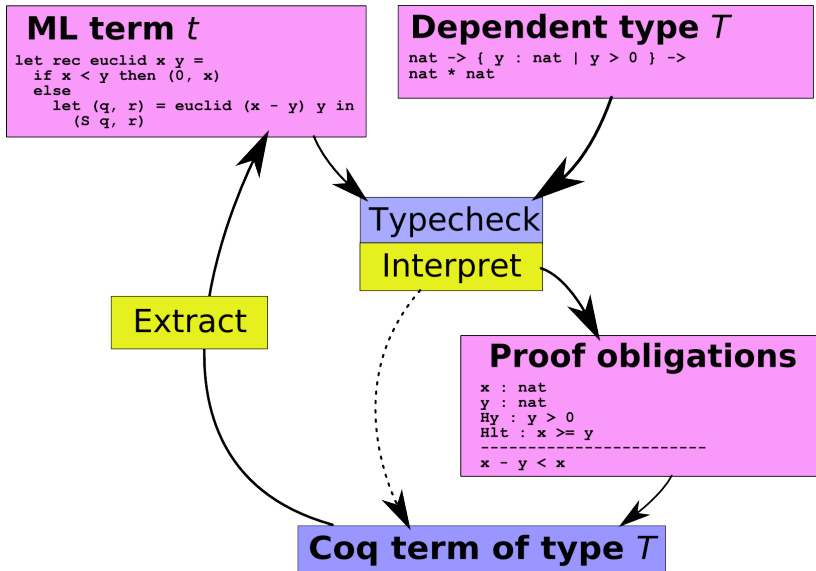
## Proof obligations

```
x : nat  
y : nat  
Hy : y > 0  
Hlt : x >= y  
-----  
x - y < x
```

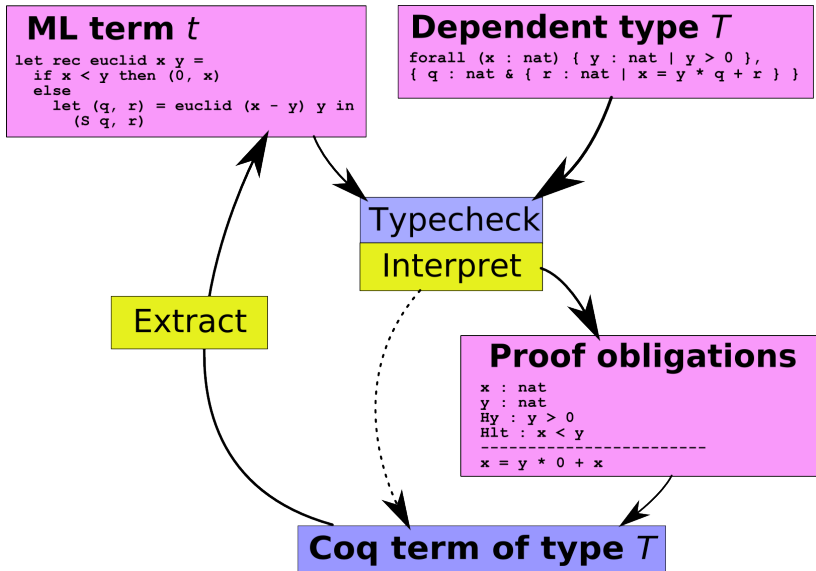
# The Big Picture



# The Big Picture



# The Big Picture





## Definition

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

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

## PVS

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

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

## PVS

- ▶ Specialized typing algorithm for subset types, generating *Type-checking conditions*.
- + Practical success ;

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

## PVS

- ▶ Specialized typing algorithm for subset types, generating *Type-checking conditions*.
- + Practical success ;
- No strong safety guarantee in PVS.

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

## PVS

- ▶ Specialized typing algorithm for subset types, generating *Type-checking conditions*.
- + Practical success ;
- No strong safety guarantee in PVS.

$$\frac{t : T \quad p : P[t/x]}{\text{elt } t \text{ } p : \{ x : T \mid P \}} \quad \frac{t : \{ x : T \mid P \}}{\sigma_1 t : T}$$

- 1 A property-irrelevant language (RUSSELL) with **decidable** typing

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

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

## ... to Subset Coercions

- 1 A property-irrelevant language (RUSSELL) with **decidable** typing
- 2 A **total** interpretation to COQ terms with **holes**

$$\frac{\Gamma \vdash t : \{ x : T \mid P \}}{\Gamma \vdash \sigma_1 t : T}$$

$$\frac{\Gamma \vdash t : T \quad \Gamma, x : T \vdash P : \mathbf{Prop}}{\Gamma \vdash \mathbf{elt} t ?_{P[t/x]} : \{ x : T \mid P \}} \quad \Gamma \vdash ?_{P[t/x]} : P[t/x]$$

## ... 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.

$$\frac{\Gamma \vdash t : \{ x : T \mid P \}}{\Gamma \vdash \sigma_1 t : T}$$

$$\frac{\Gamma \vdash t : T \quad \Gamma, x : T \vdash P : \mathbf{Prop} \quad \Gamma \vdash p : P[t/x]}{\Gamma \vdash \mathbf{elt} \ t \ p : \{ x : T \mid P \}}$$

Calculus of Constructions with

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

Calculus of Constructions with

$$\frac{\Gamma \vdash t : U \quad \Gamma \vdash U \triangleright T : s}{\Gamma \vdash t : T}$$

$$\frac{\Gamma \vdash T \equiv_{\beta} U : s}{\Gamma \vdash T \triangleright U : s}$$

Calculus of Constructions with

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

$$\frac{\Gamma \vdash U \triangleright V : \mathbf{Set} \quad \Gamma, x : U \vdash P : \mathbf{Prop}}{\Gamma \vdash \{ x : U \mid P \} \triangleright V : \mathbf{Set}}$$

$$\frac{\Gamma \vdash U \triangleright V : \mathbf{Set} \quad \Gamma, x : V \vdash P : \mathbf{Prop}}{\Gamma \vdash U \triangleright \{ x : V \mid P \} : \mathbf{Set}}$$

Calculus of Constructions with

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

$$\frac{\Gamma \vdash U \triangleright V : \mathbf{Set} \quad \Gamma, x : U \vdash P : \mathbf{Prop}}{\Gamma \vdash \{ x : U \mid P \} \triangleright V : \mathbf{Set}}$$

$$\frac{\Gamma \vdash U \triangleright V : \mathbf{Set} \quad \Gamma, x : V \vdash P : \mathbf{Prop}}{\Gamma \vdash U \triangleright \{ x : V \mid P \} : \mathbf{Set}}$$

**Example**  $\frac{\Gamma \vdash 0 : \mathbb{N} \quad \Gamma \vdash \mathbb{N} \triangleright \{ x : \mathbb{N} \mid x \neq 0 \} : \mathbf{Set}}{\Gamma \vdash 0 : \{ x : \mathbb{N} \mid x \neq 0 \}}$

Calculus of Constructions with

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

$$\frac{\Gamma \vdash U \triangleright V : \mathbf{Set} \quad \Gamma, x : U \vdash P : \mathbf{Prop}}{\Gamma \vdash \{ x : U \mid P \} \triangleright V : \mathbf{Set}}$$

$$\frac{\Gamma \vdash U \triangleright V : \mathbf{Set} \quad \Gamma, x : V \vdash P : \mathbf{Prop}}{\Gamma \vdash U \triangleright \{ x : V \mid P \} : \mathbf{Set}}$$

**Example** 
$$\frac{\Gamma \vdash 0 : \mathbb{N} \quad \Gamma \vdash \mathbb{N} \triangleright \{ x : \mathbb{N} \mid x \neq 0 \} : \mathbf{Set}}{\Gamma \vdash 0 : \{ x : \mathbb{N} \mid x \neq 0 \}}$$

$$\Gamma \vdash ? : 0 \neq 0$$

Calculus of Constructions with

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

$$\frac{\Gamma \vdash U \triangleright V : \mathbf{Set} \quad \Gamma, x : U \vdash P : \mathbf{Prop}}{\Gamma \vdash \{ x : U \mid P \} \triangleright V : \mathbf{Set}}$$

$$\frac{\Gamma \vdash U \triangleright V : \mathbf{Set} \quad \Gamma, x : V \vdash P : \mathbf{Prop}}{\Gamma \vdash U \triangleright \{ x : V \mid P \} : \mathbf{Set}}$$

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

Calculus of Constructions with

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

$$\frac{\Gamma \vdash U \triangleright V : \mathbf{Set} \quad \Gamma, x : U \vdash P : \mathbf{Prop}}{\Gamma \vdash \{ x : U \mid P \} \triangleright V : \mathbf{Set}}$$

$$\frac{\Gamma \vdash U \triangleright V : \mathbf{Set} \quad \Gamma, x : V \vdash P : \mathbf{Prop}}{\Gamma \vdash U \triangleright \{ x : V \mid P \} : \mathbf{Set}}$$

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

$\triangleright$  is symmetric!

## Theorem (Subject Reduction)

*If  $\Gamma \vdash t : T$  and  $t \rightarrow_{\beta} u$  then  $\Gamma \vdash u : T$*



## Theorem (Decidability of type checking and type inference)

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

The target system : CIC with metavariables

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

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

$$\frac{\Gamma \vdash_{?} P : \mathbf{Prop}}{\Gamma \vdash_{?} ?_{\Gamma} P : P}$$

We build an interpretation  $\llbracket - \rrbracket_{\Gamma}$  from RUSSELL to  $\text{CIC}_{?}$  terms.

The target system : CIC with metavariables

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

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

$$\frac{\Gamma \vdash_{?} P : \mathbf{Prop}}{\Gamma \vdash_{?} ?_{\Gamma} \vdash P : P}$$

We build an interpretation  $\llbracket - \rrbracket_{\Gamma}$  from RUSSELL to  $\text{CIC}_{?}$  terms.

Our goal: proving soundness

$$\text{If } \Gamma \vdash t : T \text{ then } \llbracket \Gamma \rrbracket \vdash_{?} \llbracket t \rrbracket_{\Gamma} : \llbracket T \rrbracket_{\Gamma}.$$

## Interpretation of coercions

If  $\Gamma \vdash T \triangleright U : s$  then there exists  $c$  so that  $\Gamma \vdash_{\gamma} c : T \triangleright U$ .

## Interpretation of coercions

If  $\Gamma \vdash T \triangleright U : s$  then there exists  $c$  so that  $\Gamma \vdash_{\gamma} c : T \triangleright U$ .

$$\frac{\Gamma \vdash_{\gamma} T \equiv_{\beta} U : s}{\Gamma \vdash_{\gamma} \quad : T \triangleright U}$$

## Interpretation of coercions

If  $\Gamma \vdash T \triangleright U : s$  then there exists  $c$  so that  $\Gamma \vdash_{\gamma} c : T \triangleright U$ .

$$\frac{\Gamma \vdash_{\gamma} T \equiv_{\beta} U : s}{\Gamma \vdash_{\gamma} \bullet : T \triangleright U}$$

## Interpretation of coercions

If  $\Gamma \vdash T \triangleright U : s$  then there exists  $c$  so that  $\Gamma \vdash_{\gamma} c : T \triangleright U$ .

$$\frac{\Gamma \vdash_{\gamma} T \equiv_{\beta} U : s}{\Gamma \vdash_{\gamma} \bullet : T \triangleright U}$$

$$\Gamma \vdash_{\gamma} \quad : \{ x : T \mid P \} \triangleright T$$

## Interpretation of coercions

If  $\Gamma \vdash T \triangleright U : s$  then there exists  $c$  so that  $\Gamma \vdash_{\gamma} c : T \triangleright U$ .

$$\frac{\Gamma \vdash_{\gamma} T \equiv_{\beta} U : s}{\Gamma \vdash_{\gamma} \bullet : T \triangleright U}$$
$$\Gamma \vdash_{\gamma} \sigma_1 \bullet : \{ x : T \mid P \} \triangleright T$$

## Interpretation of coercions

If  $\Gamma \vdash T \triangleright U : s$  then there exists  $c$  so that  $\Gamma \vdash_{\gamma} c : T \triangleright U$ .

$$\frac{\Gamma \vdash_{\gamma} T \equiv_{\beta} U : s}{\Gamma \vdash_{\gamma} \bullet : T \triangleright U}$$

$$\Gamma \vdash_{\gamma} \sigma_1 \bullet : \{ x : T \mid P \} \triangleright T$$

$$\Gamma \vdash_{\gamma} \phantom{\sigma_1 \bullet} : T \triangleright \{ x : T \mid P \}$$

## Interpretation of coercions

If  $\Gamma \vdash T \triangleright U : s$  then there exists  $c$  so that  $\Gamma \vdash_{?} c : T \triangleright U$ .

$$\frac{\Gamma \vdash_{?} T \equiv_{\beta} U : s}{\Gamma \vdash_{?} \bullet : T \triangleright U}$$

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

$$\Gamma \vdash_{?} \text{elt } \bullet \text{ ? } \llbracket P \rrbracket_{\Gamma, x:T} [\bullet/x] : T \triangleright \{ x : T \mid P \}$$

## Interpretation of coercions

If  $\Gamma \vdash T \triangleright U : s$  then there exists  $c$  so that  $\Gamma \vdash_{\gamma} c : T \triangleright U$ .

$$\frac{\Gamma \vdash_{\gamma} T \equiv_{\beta} U : s}{\Gamma \vdash_{\gamma} \bullet : T \triangleright U}$$

$$\Gamma \vdash_{\gamma} \sigma_1 \bullet : \{ x : T \mid P \} \triangleright T$$

$$\Gamma \vdash_{\gamma} \text{elt } \bullet \text{ ?}_{\llbracket P \rrbracket_{\Gamma, x:T}[\bullet/x]} : T \triangleright \{ x : T \mid P \}$$

## Example

$$\frac{\Gamma \vdash_{\gamma} 0 : \mathbb{N} \quad \Gamma \vdash_{\gamma} \text{elt } \bullet \text{ ?}_{\bullet \neq 0} : \mathbb{N} \triangleright \{ x : \mathbb{N} \mid x \neq 0 \}}{\Gamma \vdash_{\gamma} \text{elt } 0 \text{ ?}_{0 \neq 0} : \{ x : \mathbb{N} \mid x \neq 0 \}}$$

## Example (Application)

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

$$\llbracket f u \rrbracket_{\Gamma} = \text{let } \pi = \text{coerce}_{\Gamma} T (\Pi x : V.W) \text{ in} \\ \text{let } c = \text{coerce}_{\Gamma} U V \text{ in} \\ (\pi \llbracket f \rrbracket_{\Gamma}) (c \llbracket u \rrbracket_{\Gamma})$$

## Theorem (Soundness)

If  $\Gamma \vdash t : T$  then  $\llbracket \Gamma \rrbracket \vdash? \llbracket t \rrbracket_{\Gamma} : \llbracket T \rrbracket_{\Gamma}$ .

$\vdash_?$ 's equational theory:

$$\beta\text{-rules} \left\{ \begin{array}{l} (\beta) \quad (\lambda x : X.e) v \quad \equiv \quad e[v/x] \\ (\sigma_i) \quad \sigma_i (\text{elt}_{E,P} e_1 e_2) \quad \equiv \quad e_i \end{array} \right\} \text{CoQ}$$

$\vdash_?$ 's equational theory:

$$\beta\text{-rules} \left\{ \begin{array}{l} (\beta) \quad (\lambda x : X.e) v \quad \equiv \quad e[v/x] \\ (\sigma_i) \quad \sigma_i (\text{elt}_{E,P} e_1 e_2) \quad \equiv \quad e_i \end{array} \right\} \text{CoQ}$$

$$\eta\text{-rules} \left\{ \begin{array}{l} (\eta) \quad (\lambda x : X.e x) \quad \equiv \quad e \quad \text{if } x \notin \mathcal{FV}(e) \\ (\text{SP}) \quad \text{elt}_{E,P} (\sigma_1 e) (\sigma_2 e) \quad \equiv \quad e \end{array} \right.$$

$\vdash_?$ 's equational theory:

$$\beta\text{-rules} \left\{ \begin{array}{l} (\beta) \quad (\lambda x : X.e) v \quad \equiv \quad e[v/x] \\ (\sigma_i) \quad \text{elt}_{E,P} e_1 e_2 \quad \equiv \quad e_i \end{array} \right\} \text{CoQ}$$

$$\eta\text{-rules} \left\{ \begin{array}{l} (\eta) \quad (\lambda x : X.e x) \quad \equiv \quad e \quad \text{if } x \notin \mathcal{FV}(e) \\ (\text{SP}) \quad \text{elt}_{E,P} (\sigma_1 e) (\sigma_2 e) \quad \equiv \quad e \end{array} \right.$$

$$\text{Proof Irrelevance} \quad \text{elt}_{E,P} t p \quad \equiv \quad \text{elt}_{E,P} t' p' \quad \text{if } t \equiv t'$$

$\vdash_?$ 's equational theory:

$$\beta\text{-rules} \left\{ \begin{array}{l} (\beta) \quad (\lambda x : X.e) v \quad \equiv \quad e[v/x] \\ (\sigma_i) \quad \sigma_i (\text{elt}_{E,P} e_1 e_2) \quad \equiv \quad e_i \end{array} \right\} \text{CoQ}$$

$$\eta\text{-rules} \left\{ \begin{array}{l} (\eta) \quad (\lambda x : X.e x) \quad \equiv \quad e \quad \text{if } x \notin \mathcal{FV}(e) \\ (\text{SP}) \quad \text{elt}_{E,P} (\sigma_1 e) (\sigma_2 e) \quad \equiv \quad e \end{array} \right.$$

$$\text{Proof Irrelevance} \quad \text{elt}_{E,P} t p \quad \equiv \quad \text{elt}_{E,P} t' p' \quad \text{if } t \equiv t'$$

... have practical effects

Difficulty to reason on code:  $t \equiv u \not\rightarrow \llbracket t \rrbracket_\Gamma \equiv_{\text{CoQ}} \llbracket u \rrbracket_\Gamma$ .



# The PROGRAM vernacular

Replaces the Program tactic by Catherine Parent.

# The PROGRAM vernacular

Replaces the Program tactic by Catherine Parent.

## Architecture

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

# The PROGRAM vernacular

Replaces the Program tactic by Catherine Parent.

## Architecture

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

- 1 Use the COQ parser ;

Program Definition  $f : T := t$  .

# The PROGRAM vernacular

Replaces the Program tactic by Catherine Parent.

## Architecture

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

- 1 Use the COQ parser ;
- 2 Typecheck  $\Gamma \vdash t : T$  and generate  $\llbracket \Gamma \rrbracket \vdash? \llbracket t \rrbracket_{\Gamma} : \llbracket T \rrbracket_{\Gamma}$  ;

Program Definition  $f : \llbracket T \rrbracket_{\Gamma} := \llbracket t \rrbracket_{\Gamma}$  .

# The PROGRAM vernacular

Replaces the Program tactic by Catherine Parent.

## Architecture

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

- 1 Use the COQ parser ;
- 2 Typecheck  $\Gamma \vdash t : T$  and generate  $\llbracket \Gamma \rrbracket \vdash? \llbracket t \rrbracket_{\Gamma} : \llbracket T \rrbracket_{\Gamma}$  ;
- 3 Interactive proof of the obligations ;

Program Definition  $f : \llbracket T \rrbracket_{\Gamma} := \llbracket t \rrbracket_{\Gamma} + \text{obligations}$ .

# The PROGRAM vernacular

Replaces the Program tactic by Catherine Parent.

## Architecture

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

- 1 Use the COQ parser ;
- 2 Typecheck  $\Gamma \vdash t : T$  and generate  $\llbracket \Gamma \rrbracket \vdash? \llbracket t \rrbracket_{\Gamma} : \llbracket T \rrbracket_{\Gamma}$  ;
- 3 Interactive proof of the obligations ;
- 4 Final definition.

Definition  $f : \llbracket T \rrbracket_{\Gamma} := \llbracket t \rrbracket_{\Gamma} + \text{obligations}$ .

DEMO

A **methodology** to build certified code in Coq, distributed since 2005:

- ▶ Used to develop a non-trivial example: Dependent Finger Trees (ICFP'07).

A **methodology** to build certified code in  $\text{Coq}$ , distributed since 2005:

- ▶ Used to develop a non-trivial example: Dependent Finger Trees (ICFP'07).
- ▶ Already experimented by others, e.g. the Ynot project (Nanevsky, Morrisett & Birkedal).

A **methodology** to build certified code in Coq, distributed since 2005:

- ▶ Used to develop a non-trivial example: Dependent Finger Trees (ICFP'07).
- ▶ Already experimented by others, e.g. the Ynot project (Nanevsky, Morrisett & Birkedal).
- ▶ Independent of the theory, can be ported to other systems: Matita (by Enrico Tassi), Agda, Epigram.

- ▶ We hinted at the important **foundational issues** with  $\eta$ -rules and proof-irrelevance that need to be solved (with Benjamin Werner).
- ▶ **Automation** for discharging proof-obligations (Sean Wilson at Edinburgh).

## COQ: An Environment for Proving and Programming

MATTHIEU SOZEAU

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

Ph.D student's seminar  
January 26th 2009  
LIFO - Orléans

Try



8.2!

