Subset coercions in Coq

Matthieu Sozeau
under direction of Christine Paulin-Mohring

Paris-Sud 11 University
LRI - Démons Team & INRIA Futurs - Proval Project

TYPES'06 Workshop
18-21 April 2006

The Big Picture

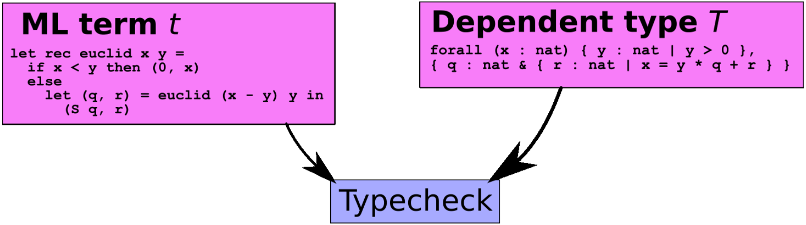
ML term *t*

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

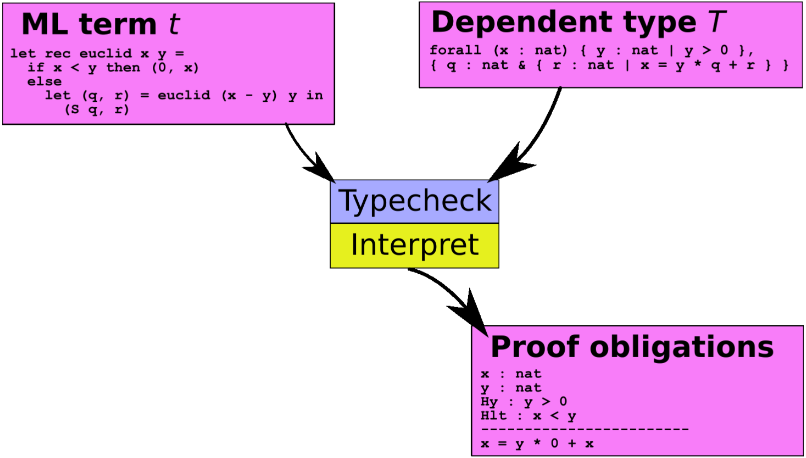
Dependent type *T*

```
forall (x : nat) { y : nat | y > 0 },
{ q : nat & { r : nat | x = y * q + r } }
```

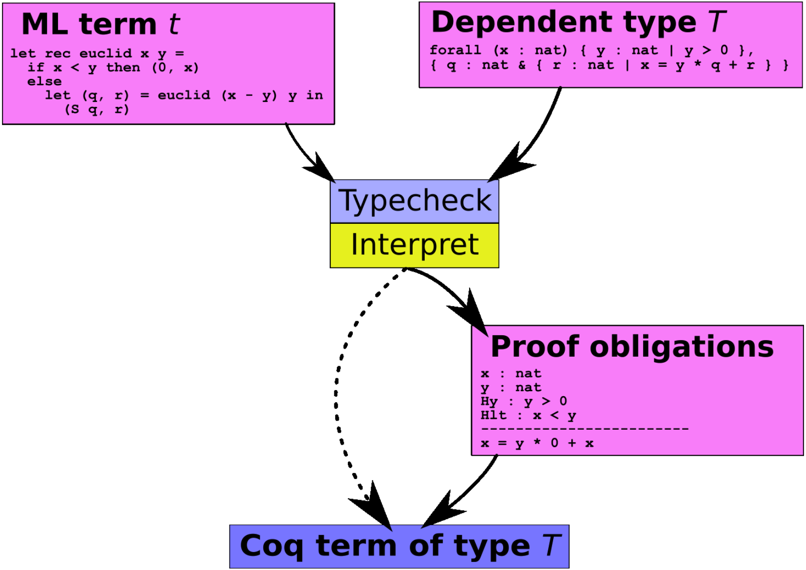
The Big Picture



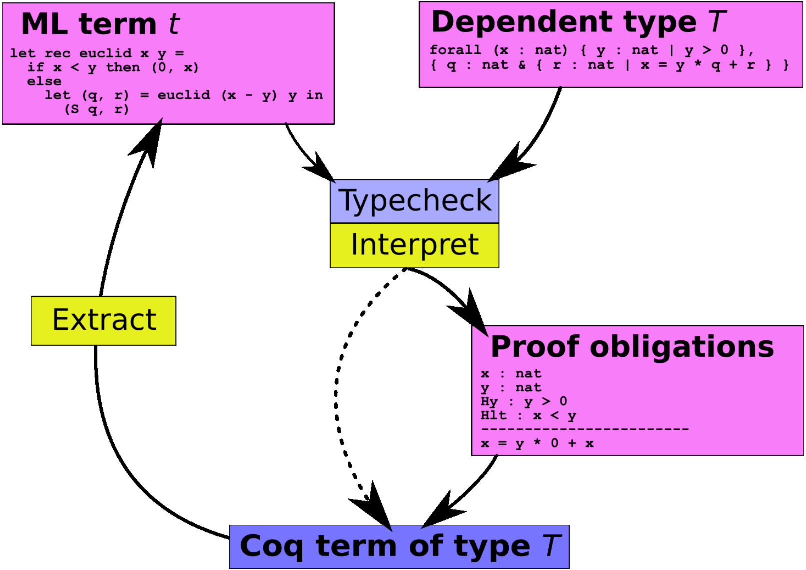
The Big Picture



The Big Picture



The Big Picture



Outline

1. The idea
2. Theoretical development
   - Russell
   - Traduction in Coq
3. The implementation

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 ;
- Separates 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 ;
- Separates object and property.

Adapting the idea

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

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 ;

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

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

. . . to Subset coercions

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

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

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

. . . to Subset coercions

1. A property-irrelevant language (Russell) with decidable typing ;
2. A total traduction to Coq terms with holes ;
3. A method to turn the holes into proof obligations.

Outline

1. The idea
2. Theoretical development
   - Russell
   - Traduction in Coq
3. The implementation

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

Calculus of Constructions - Conv +

$$\textsc{Coerce}\ \frac{\Gamma \vdash t : U \qquad \Gamma \vdash T : s \qquad U \rhd T}{\Gamma \vdash t : T}$$

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

Calculus of Constructions - CONV +

$$\textsc{Coerce}\ \frac{\Gamma \vdash t : U \qquad \Gamma \vdash T : s \qquad U \rhd T}{\Gamma \vdash t : T}$$

$$\textsc{Sub-Conv}\ \frac{T \equiv_{\beta\pi} U}{T \rhd U} \qquad \textsc{Sub-Trans}\ \frac{S \rhd T \qquad T \rhd U}{S \rhd U}$$

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

Calculus of Constructions - Conv +

$$\textsc{Coerce}\ \frac{\Gamma \vdash t : U \qquad \Gamma \vdash T : s \qquad U \rhd T}{\Gamma \vdash t : T}$$

$$\textsc{Sub-Conv}\ \frac{T \equiv_{\beta\pi} U}{T \rhd U} \qquad \textsc{Sub-Trans}\ \frac{S \rhd T \qquad T \rhd U}{S \rhd U}$$

$$\textsc{Sub-Subset}\ \frac{U \rhd V}{\{\, x : U \mid P \,\} \rhd V} \qquad \textsc{Sub-Proof}\ \frac{U \rhd V}{U \rhd \{\, x : V \mid P \,\}}$$

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

Calculus of Constructions - CONV +

$$\textsc{Coerce}\ \frac{\Gamma \vdash t : U \qquad \Gamma \vdash T : s \qquad U \rhd T}{\Gamma \vdash t : T}$$

$$\textsc{Sub-Conv}\ \frac{T \equiv_{\beta\pi} U}{T \rhd U} \qquad \textsc{Sub-Trans}\ \frac{S \rhd T \qquad T \rhd U}{S \rhd U}$$

$$\textsc{Sub-Subset}\ \frac{U \rhd V}{\{\, x : U \mid P \,\} \rhd V} \qquad \textsc{Sub-Proof}\ \frac{U \rhd V}{U \rhd \{\, x : V \mid P \,\}}$$

$$\frac{0 : \mathtt{nat} \qquad \mathtt{nat} \rhd \{\, x : \mathtt{nat} \mid x \neq 0 \,\}}{0 : \{\, x : \mathtt{nat} \mid x \neq 0 \,\}}$$

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

Calculus of Constructions - Conv +

$$\textsc{Coerce}\ \frac{\Gamma \vdash t : U \qquad \Gamma \vdash T : s \qquad U \rhd T}{\Gamma \vdash t : T}$$

$$\textsc{Sub-Conv}\ \frac{T \equiv_{\beta\pi} U}{T \rhd U} \qquad \textsc{Sub-Trans}\ \frac{S \rhd T \qquad T \rhd U}{S \rhd U}$$

$$\textsc{Sub-Subset}\ \frac{U \rhd V}{\{\, x : U \mid P \,\} \rhd V} \qquad \textsc{Sub-Proof}\ \frac{U \rhd V}{U \rhd \{\, x : V \mid P \,\}}$$

$$\cfrac{\cfrac{0 : \mathtt{nat} \qquad \mathtt{nat} \rhd \{\, x : \mathtt{nat} \mid x \neq 0 \,\}}{0 : \{\, x : \mathtt{nat} \mid x \neq 0 \,\}}}{? : 0 \neq 0}$$

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

Calculus of Constructions - CONV +

$$\text{COERCE}\ \frac{\Gamma \vdash t : U \qquad \Gamma \vdash T : s \qquad U \rhd T}{\Gamma \vdash t : T}$$

$$\text{SUB-CONV}\ \frac{T \equiv_{\beta\pi} U}{T \rhd U} \qquad \text{SUB-TRANS}\ \frac{S \rhd T \qquad T \rhd U}{S \rhd U}$$

$$\text{SUB-SUBSET}\ \frac{U \rhd V}{\{\, x : U \mid P \,\} \rhd V} \qquad \text{SUB-PROOF}\ \frac{U \rhd V}{U \rhd \{\, x : V \mid P \,\}}$$

$$\text{SUB-PROD}\ \frac{U \rhd T \qquad V \rhd W}{\Pi x : T.V \rhd \Pi x : U.W}$$

Results

Theorem (Decidability of type checking)

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

$$\textsc{App}\ \frac{\Gamma \vdash f : T \quad T \rhd \Pi x : A.B \qquad \Gamma \vdash e : E \qquad E \rhd A}{\Gamma \vdash (f\ e) : B[e/x]}$$

Results

Theorem (Decidability of type checking)

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

Lemma (Elimination of transitivity)

If $T \rhd U \wedge U \rhd V$ *then* $T \rhd V$.

$$\textsc{App}\ \frac{\Gamma \vdash f : T \quad T \rhd \Pi x : A.B \qquad \Gamma \vdash e : E \qquad E \rhd A}{\Gamma \vdash (f\ e) : B[e/x]}$$

Outline

The target system

CIC with metavariables

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

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

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

From Coq to Russell

The easy way

$$\begin{array}{lcl} (\sigma_1\ t)^\circ & = & t^\circ \\ (\text{elt}\ T\ P\ t\ p)^\circ & = & t^\circ \\ (\sigma_2\ t)^\circ & = & \bot \\ (?_P)^\circ & = & \bot \end{array}$$

If $\Gamma \vdash_? t : T$ then $\Gamma^\circ \vdash t^\circ : T^\circ$ if $()^\circ$ is defined on Γ, t and T.

From Coq to Russell and back

The easy way

$$
\begin{array}{lcl}
(\sigma_1\ t)^\circ & = & t^\circ \\
(\text{elt}\ T\ P\ t\ p)^\circ & = & t^\circ \\
(\sigma_2\ t)^\circ & = & \bot \\
(?_P)^\circ & = & \bot
\end{array}
$$

If $\Gamma \vdash_? t : T$ then $\Gamma^\circ \vdash t^\circ : T^\circ$ if $()^\circ$ is defined on Γ, t and T.

The hard way

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

Traduction : deriving explicit coercions

Traduction for coercions

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

Traduction : deriving explicit coercions

Traduction for coercions

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

Definition

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

Traduction : deriving explicit coercions

Traduction for coercions

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

Definition

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

Traduction : deriving explicit coercions

Traduction for coercions

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

Definition

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

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

Traduction : deriving explicit coercions

Traduction for coercions

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

Definition

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

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

Traduction : deriving explicit coercions

Traduction for coercions

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

Definition

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

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

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

Traduction : deriving explicit coercions

Traduction for coercions

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

Definition

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

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

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

Traduction : deriving explicit coercions

Traduction for coercions

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

Definition

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

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

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

Example

$$\frac{\Gamma \vdash_? 0 : \texttt{nat} \qquad \Gamma \vdash_? c : \texttt{nat} \rhd \{\, x : \texttt{nat} \mid x \neq 0 \,\}}{\Gamma \vdash_? (\mathit{elt} \quad \bullet \; ?_{(x \neq 0)[\bullet/x]})[0] = \mathit{elt} \quad 0 \; ?_{0 \neq 0} : \{\, x : \texttt{nat} \mid x \neq 0 \,\}}$$

Traduction : interpretation of terms $[\![\,]\!]_\Gamma$

Example (Application)

$$\frac{\Gamma \vdash f : T \quad T \rhd \Pi x : V.W : s \qquad \Gamma \vdash u : U \qquad U \rhd V}{\Gamma \vdash (f\ u) : W[u/x]}$$

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

Theorem (Soundness)

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

Theoretical matters

$\vdash_?$'s equational theory :

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

Theoretical matters

$\vdash_?$'s equational theory :

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

$\Rightarrow$ Proof Irrelevance

Outline

The PROGRAM vernacular

Architecture

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

1. Use the COQ parser : `Program` **Definition** $f\ : T := t.$;

The PROGRAM vernacular

Architecture

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

1. Use the COQ parser : `Program` **Definition** $f \; : T := t.$;
2. Typecheck $\Gamma^\circ \vdash t : T$ and generate $[\![\Gamma^\circ]\!] \vdash_? [\![t]\!]_{\Gamma^\circ} : [\![T]\!]_{\Gamma^\circ}$;

`Program` **Definition** $f \; : [\![T]\!]_{\Gamma^\circ} := [\![t]\!]_{\Gamma^\circ}$.

The PROGRAM vernacular

Architecture

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

1. Use the COQ parser : `Program` **Definition** $f\ : T := t.$;
2. Typecheck $\Gamma^\circ \vdash t : T$ and generate $[\![\Gamma^\circ]\!] \vdash_? [\![t]\!]_{\Gamma^\circ} : [\![T]\!]_{\Gamma^\circ}$;
3. Interactive proving of obligations ;

`Program` **Definition** $f\ : [\![T]\!]_{\Gamma^\circ} := [\![t]\!]_{\Gamma^\circ} +$ obligations.

The Program vernacular

Architecture

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

1. Use the Coq parser : `Program` **Definition** $f\ : T := t.$;
2. Typecheck $\Gamma^\circ \vdash t : T$ and generate $[\![\Gamma^\circ]\!] \vdash_? [\![t]\!]_{\Gamma^\circ} : [\![T]\!]_{\Gamma^\circ}$;
3. Interactive proving of obligations ;
4. Final definition.

Definition $f\ : [\![T]\!]_{\Gamma^\circ} := [\![t]\!]_{\Gamma^\circ} +$ obligations.

The Program vernacular

Architecture

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

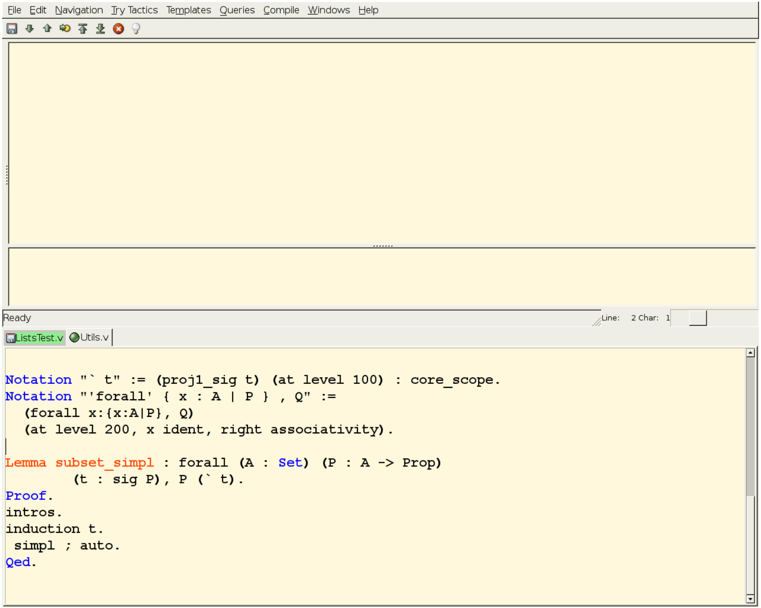
1. Use the Coq parser : `Program` **Definition** $f \ : T := t.$;
2. Typecheck $\Gamma^{\circ} \vdash t : T$ and generate $[\![\Gamma^{\circ}]\!] \vdash_{?} [\![t]\!]_{\Gamma^{\circ}} : [\![T]\!]_{\Gamma^{\circ}}$;
3. Interactive proving of obligations ;
4. Final definition.

Definition $f \ : [\![T]\!]_{\Gamma^{\circ}} := [\![t]\!]_{\Gamma^{\circ}} +$ obligations.

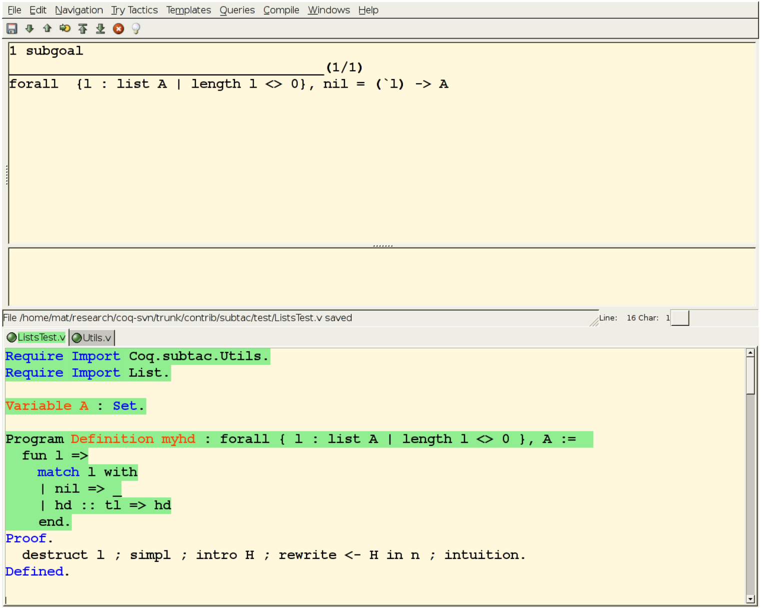
Remark (Restriction)

We assume $\Gamma^{\circ} = \Gamma$ *and* $\Gamma \vdash_{CCI} [\![T]\!]_{\Gamma} : s$.

PROGRAM : The list example



PROGRAM : The list example

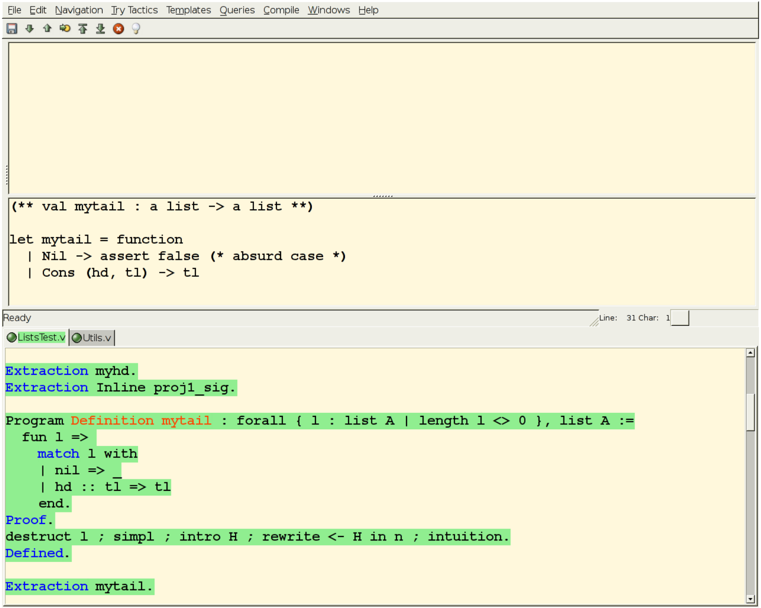


PROGRAM : The list example

```
(** val myhd : a list -> a **)

let myhd l =
  match proj1_sig l with
    | Nil -> assert false (* absurd case *)
    | Cons (hd, tl) -> hd
```

```
Variable A : Set.

Program Definition myhd : forall { l : list A | length l <> 0 }, A :=
  fun l =>
    match l with
    | nil => _
    | hd :: tl => hd
    end.
Proof.
  destruct l ; simpl ; intro H ; rewrite <- H in n ; intuition.
Defined.


Extraction myhd.
Extraction Inline proj1_sig.
```

PROGRAM : The list example



PROGRAM : The list example

File Edit Navigation Try Tactics Templates Queries Compile Windows Help

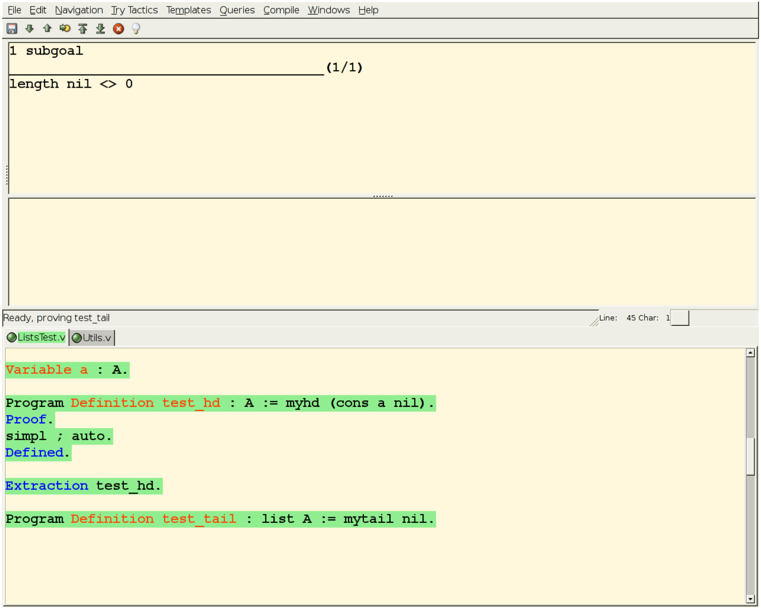
```
1 subgoal
______________________________________(1/1)
length (a :: nil) <> 0
```

Ready, proving test_hd   Line: 39 Char: 1

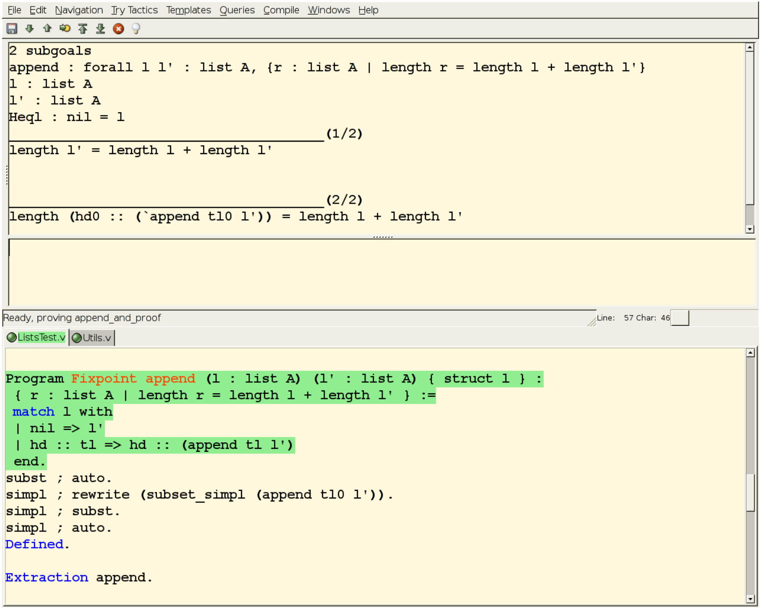
ListsTest.v   Utils.v

```
Variable a : A.

Program Definition test_hd : A := myhd (cons a nil).
Proof.
simpl ; auto.
Defined.

Extraction test_hd.

Program Definition test_tail : list A := mytail nil.
```

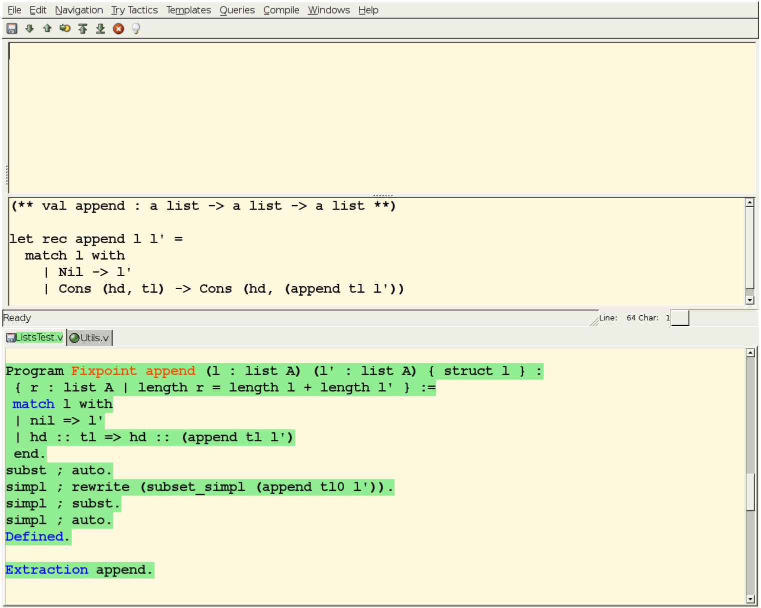
Program : The list example



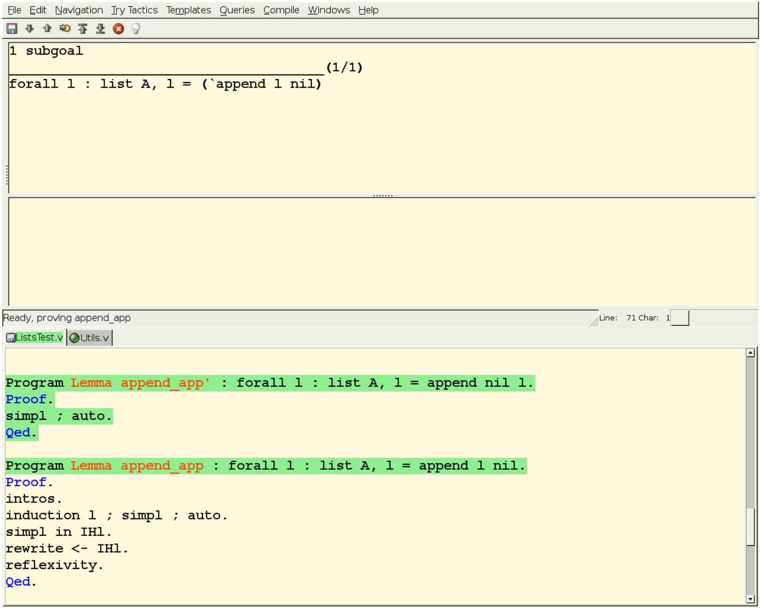
PROGRAM : The list example



PROGRAM : The list example



Program : The list example



Conclusion

Our contribution

A more flexible programming language, (almost) conservative over CIC, integrated with the existing environment and a formal justification of “*Predicate subtyping*”.

Conclusion

Our contribution

A more flexible programming language, (almost) conservative over CIC, integrated with the existing environment and a formal justification of "*Predicate subtyping*".

Future work

- Application to more constructs ((co-)inductive types) and commands.
- Improvements of Coq (existential variables, type inference, proof irrelevance).
- Complete and useful interpretation of ML languages.

Addendum : some practical enhancements

- Handling of dependent existential variables (WIP).

Addendum : some practical enhancements

- Handling of dependent existential variables (WIP).
- Pattern-matching and equalities.

$$\textbf{match } v \textbf{ return } T \textbf{ with } p_1 \Rightarrow t_1 \cdots p_n \Rightarrow t_n$$

$$\begin{array}{lcl}
(\textbf{match } \mu v \textbf{ as } t' & \textbf{return} & t' = \mu(v) \rightarrow T) \textbf{ with} \\
p_1 \Rightarrow \textbf{fun } h & \Rightarrow & t_1 \\
 & \vdots & \\
p_n \Rightarrow \textbf{fun } h & \Rightarrow & t_n) \\
(\textbf{refl_equal } \mu(v)) & &
\end{array}$$

Addendum : some practical enhancements

- Handling of dependent existential variables (WIP).
- Pattern-matching and equalities.
- Well-founded recursion.

$$\texttt{Program}\ \textbf{Fixpoint}\ f\ (a : nat)\ \{\textbf{wf}\ lt\ a\} : \texttt{nat} := t$$

Addendum : some practical enhancements

- Handling of dependent existential variables (WIP).
- Pattern-matching and equalities.
- Well-founded recursion.

$$\texttt{Program}\ \textbf{Fixpoint}\ f\ (a : nat)\ \{\textbf{wf}\ lt\ a\} : \texttt{nat} := t$$

$$\frac{\begin{array}{lcl} a & : & \texttt{nat} \\ f & : & \{x : \texttt{nat} \mid x < a\} \to \texttt{nat} \end{array}}{\begin{array}{lcl} t & : & \texttt{nat} \end{array}}$$