Connecting Coq with first-order logic by scope

Chantal Keller
Joint work with Valentin Blot, Louise Dubois de Prisque, Pierre Vial

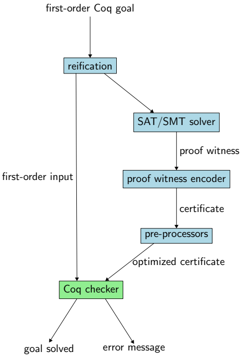
LMF, Université Paris-Saclay

June, 1st 2022

SMTCoq automatic tactics

On a goal automatable by SMT solvers:

- run one or multiple external SMT solver on the negation of the goal
- use their answers to prove the goal



SMTCoq automatic tactics

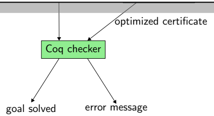
On a goal automatable by SMT solvers:

- run one o egation of the goal
- use their

```
Section Group.
Variable G : Type.
Variable e : G.
Variable op : G → G → G.
Variable inv : G → G.

Hypothesis associative :
  forall a b c : G, op a (op b c) = op (op a b) c.
Hypothesis identity :
  forall a : G, (op e a = a) ∧ (op a e = a).
Hypothesis inverse :
  forall a : G, (op a (inv a) = e) ∧ (op (inv a) a = e).

Lemma unique_identity e':
  (forall z, op e' z = z) → e' = e.
Proof.
  smt (associative, identity, inverse).
Qed.
End Group.
```



Difficulties for automation in Coq

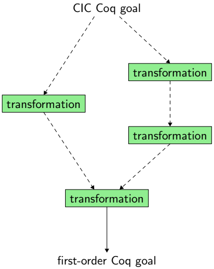
The dream:

- the mathematician/programmer concentrates on the difficult parts of proofs
- Coq automatically fills trivial gaps

But:

- Coq's logic makes even simple goals far from first-order logic
- useful to provide good feedback when it fails
- multiple representations of the same objects (e.g: Peano integers are easier to reason about but one may care about efficiency)

Ongoing project: Sniper



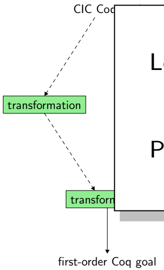
Small-grained transformations:

- transform a goal `G` into a new goal `G'` (may leave subgoals to users)
- produce a Coq proof that `G'` $\rightarrow$ `G`

Sniper:

- provides a bunch of useful transformations
- provides strategies to combine them (called scope)
- then calls SMT solvers

Ongoing project: Sniper



Small-grained transformations:

- transform a goal G into a new goal G' (may leave subgoals to users)

```
Lemma search_app :
  forall {A: Type} (x: A) (l1 l2: list A),
   search x (l1 ++ l2) = (search x l1) || (search x l2).
Proof. induction l1; snipe. Qed.
```

- provides a bunch of useful transformations
- provides strategies to combine them (called scope)
- then calls SMT solvers

Outline

Example of a transformation: datatypes

For each inductive types appearing in the goal, state:

Example of lists:

Example of a transformation: datatypes

For each inductive types appearing in the goal, state:

- constructors are pairwise disjoint

Example of lists:

- `∀A (x:A) (l:list A), [] ≠ x::l`

Example of a transformation: datatypes

For each inductive types appearing in the goal, state:

- constructors are pairwise disjoint
- constructors are injective

Example of lists:

- $\forall$A (x:A) (l:list A), [] $\neq$ x::l
- $\forall$A (x y:A) (l m:list A), x::l $=$ y::m $\rightarrow$ x $=$ y $\wedge$ l $=$ m

Example of a transformation: datatypes

For each inductive types appearing in the goal, state:

- constructors are pairwise disjoint
- constructors are injective
- every term of this type is generated by one of the constructors

Example of lists:

- $\forall$`A (x:A) (l:list A), [] `$\neq$` x::l`
- $\forall$`A (x y:A) (l m:list A), x::l = y::m `$\rightarrow$` x = y `$\wedge$` l = m`
- $\forall$`A (l:list A), l = [] `$\vee$` l = (hd l defA)::(tl m deflA)`

Implemented transformations

Atomic and pairwise-independent transformations:

- make explicit the semantics of uninterpreted symbols (algebraic datatypes, constant and function definitions)
- eliminate unknown constructions (higher-order equalities, polymorphism, pattern matching, fixpoints)
- reflect bijective types and bool/Prop (trakt, Enzo Crance, Assia Mahboubi and Denis Cousineau)

+ a strategy: scope

Example

`A:Type` `l:list A` `n:nat`

`@length A l` $=$ `n+1` $\rightarrow$ `l` $\neq$ `[]`

Example

1. inductive types

$$\texttt{A:Type} \qquad \texttt{l:list A} \qquad \texttt{n:nat}$$

$$\forall\texttt{(x:nat), 0} \neq \texttt{S x} \qquad \forall\texttt{(x y:nat), S x} = \texttt{S y} \rightarrow \texttt{x} = \texttt{y}$$
$$\forall\texttt{B (x:B) (l:list B), []} \neq \texttt{x::l}$$
$$\forall\texttt{B (x y:B) (l m:list B), x::l} = \texttt{y::l} \rightarrow \texttt{x} = \texttt{y} \wedge \texttt{l} = \texttt{m}$$

$$\texttt{@length A l} = \texttt{n+1} \rightarrow \texttt{l} \neq \texttt{[]}$$

Example

1. inductive types
2. definitions

$$\texttt{A:Type} \qquad \texttt{l:list A} \qquad \texttt{n:nat}$$

$$\forall\texttt{(x:nat), 0} \neq \texttt{S x} \qquad \forall\texttt{(x y:nat), S x} = \texttt{S y} \rightarrow \texttt{x} = \texttt{y}$$
$$\forall\texttt{B (x:B) (l:list B), []} \neq \texttt{x::l}$$
$$\forall\texttt{B (x y:B) (l m:list B), x::l} = \texttt{y::l} \rightarrow \texttt{x} = \texttt{y} \wedge \texttt{l} = \texttt{m}$$

$$\texttt{length} = \texttt{fun (B:Type)} \Rightarrow \texttt{fix length (l:list B) := match l...}$$

$$\texttt{@length A l} = \texttt{n+1} \rightarrow \texttt{l} \neq \texttt{[]}$$

Example

1. inductive types
2. definitions
3. expansion

$$\texttt{A:Type} \quad \texttt{l:list A} \quad \texttt{n:nat}$$

$$\forall\texttt{(x:nat), 0} \neq \texttt{S x} \quad \forall\texttt{(x y:nat), S x} = \texttt{S y} \rightarrow \texttt{x} = \texttt{y}$$
$$\forall\texttt{B (x:B) (l:list B), []} \neq \texttt{x::l}$$
$$\forall\texttt{B (x y:B) (l m:list B), x::l} = \texttt{y::l} \rightarrow \texttt{x} = \texttt{y} \wedge \texttt{l} = \texttt{m}$$

$$\texttt{length} = \texttt{fun (B:Type)} \Rightarrow \texttt{fix length (l:list B) := match l...}$$

$$\forall\texttt{B (l:list B), length B l} =$$
$$\texttt{(fun (B:Type)} \Rightarrow \texttt{fix length (l:list B) := match l...) B l}$$

$$\texttt{@length A l} = \texttt{n+1} \rightarrow \texttt{l} \neq \texttt{[]}$$

Example

1. inductive types
2. definitions
3. expansion
4. fixpoints

$$\texttt{A:Type} \quad \texttt{l:list A} \quad \texttt{n:nat}$$

$$\forall\texttt{(x:nat), 0} \neq \texttt{S x} \quad \forall\texttt{(x y:nat), S x} = \texttt{S y} \to \texttt{x} = \texttt{y}$$
$$\forall\texttt{B (x:B) (l:list B), []} \neq \texttt{x::l}$$
$$\forall\texttt{B (x y:B) (l m:list B), x::l} = \texttt{y::l} \to \texttt{x} = \texttt{y} \wedge \texttt{l} = \texttt{m}$$

$$\texttt{length} = \texttt{fun (B:Type)} \Rightarrow \texttt{fix length (l:list B) := match l...}$$

$$\forall\texttt{B (l:list B), length B l} =$$
$$\texttt{(fun (B:Type)} \Rightarrow \texttt{fix length (l:list B) := match l...) B l}$$

$$\forall\texttt{B (l:list B), @length B l} = \texttt{match l...}$$

$$\texttt{@length A l} = \texttt{n+1} \to \texttt{l} \neq \texttt{[]}$$

Example

1. inductive types
2. definitions
3. expansion
4. fixpoints
5. p. matching

$$\texttt{A:Type} \quad \texttt{l:list A} \quad \texttt{n:nat}$$

$$\forall\texttt{(x:nat), 0} \neq \texttt{S x} \quad \forall\texttt{(x y:nat), S x} = \texttt{S y} \rightarrow \texttt{x} = \texttt{y}$$
$$\forall\texttt{B (x:B) (l:list B), []} \neq \texttt{x::l}$$
$$\forall\texttt{B (x y:B) (l m:list B), x::l} = \texttt{y::l} \rightarrow \texttt{x} = \texttt{y} \wedge \texttt{l} = \texttt{m}$$

$$\texttt{length} = \texttt{fun (B:Type)} \Rightarrow \texttt{fix length (l:list B) := match l...}$$

$$\forall\texttt{B (l:list B), length B l} =$$
$$\texttt{(fun (B:Type)} \Rightarrow \texttt{fix length (l:list B) := match l...) B l}$$

$$\forall\texttt{B (l:list B), @length B l} = \texttt{match l...}$$

$$\frac{\forall\texttt{B, @length B []} = \texttt{0} \qquad \forall\texttt{B (x:B) (l:list B), @length B (x::l)} = \texttt{S (@length B l)}}{\texttt{@length A l} = \texttt{n+1} \rightarrow \texttt{l} \neq \texttt{[]}}$$

Example

1. inductive types
2. definitions
3. expansion
4. fixpoints
5. p. matching
6. polymorphism

$$\texttt{A:Type} \qquad \texttt{l:list A} \qquad \texttt{n:nat}$$

$$\forall\texttt{(x:nat), 0} \neq \texttt{S x} \qquad \forall\texttt{(x y:nat), S x} = \texttt{S y} \rightarrow \texttt{x} = \texttt{y}$$
$$\forall\texttt{B (x:B) (l:list B), []} \neq \texttt{x::l}$$
$$\forall\texttt{B (x y:B) (l m:list B), x::l} = \texttt{y::l} \rightarrow \texttt{x} = \texttt{y} \wedge \texttt{l} = \texttt{m}$$

$$\texttt{length} = \texttt{fun (B:Type)} \Rightarrow \texttt{fix length (l:list B) := match l...}$$

$$\forall\texttt{B (l:list B), length B l} =$$
$$\texttt{(fun (B:Type)} \Rightarrow \texttt{fix length (l:list B) := match l...) B l}$$

$$\forall\texttt{B (l:list B), @length B l} = \texttt{match l...}$$

$$\frac{\forall\texttt{B, @length B []} = \texttt{0} \qquad \forall\texttt{B (x:B) (l:list B), @length B (x::l)} = \texttt{S (@length B l)}}{\texttt{@length A l} = \texttt{n+1} \rightarrow \texttt{l} \neq \texttt{[]}}$$

Example

1. inductive types
2. definitions
3. expansion
4. fixpoints
5. p. matching
6. polymorphism

$$\texttt{A:Type} \quad \texttt{l:list A} \quad \texttt{n:nat}$$

$$\forall\texttt{(x:nat), 0} \neq \texttt{S x} \quad \forall\texttt{(x y:nat), S x} = \texttt{S y} \rightarrow \texttt{x} = \texttt{y}$$
$$\forall\texttt{(x:A) (l:list A), []} \neq \texttt{x::l}$$
$$\forall\texttt{B (x y:B) (l m:list B), x::l} = \texttt{y::l} \rightarrow \texttt{x} = \texttt{y} \wedge \texttt{l} = \texttt{m}$$

$$\texttt{length} = \texttt{fun (B:Type)} \Rightarrow \texttt{fix length (l:list B) := match l...}$$

$$\forall\texttt{B (l:list B), length B l} =$$
$$\texttt{(fun (B:Type)} \Rightarrow \texttt{fix length (l:list B) := match l...) B l}$$

$$\forall\texttt{B (l:list B), @length B l} = \texttt{match l...}$$

$$\frac{\forall\texttt{B, @length B []} = \texttt{0} \qquad \forall\texttt{B (x:B) (l:list B), @length B (x::l)} = \texttt{S (@length B l)}}{\texttt{@length A l} = \texttt{n+1} \rightarrow \texttt{l} \neq \texttt{[]}}$$

Example

1. inductive types
2. definitions
3. expansion
4. fixpoints
5. p. matching
6. polymorphism

$$\texttt{A:Type} \quad \texttt{l:list A} \quad \texttt{n:nat}$$

$$\forall\texttt{(x:nat), 0} \neq \texttt{S x} \quad \forall\texttt{(x y:nat), S x} = \texttt{S y} \rightarrow \texttt{x} = \texttt{y}$$
$$\forall\texttt{(x:A) (l:list A), []} \neq \texttt{x::l}$$
$$\forall\texttt{(x y:A) (l m:list A), x::l} = \texttt{y::l} \rightarrow \texttt{x} = \texttt{y} \wedge \texttt{l} = \texttt{m}$$

$$\texttt{length} = \texttt{fun (B:Type)} \Rightarrow \texttt{fix length (l:list B) := match l...}$$

$$\forall\texttt{B (l:list B), length B l} =$$
$$\texttt{(fun (B:Type)} \Rightarrow \texttt{fix length (l:list B) := match l...) B l}$$

$$\forall\texttt{B (l:list B), @length B l} = \texttt{match l...}$$

$$\frac{\forall\texttt{B, @length B []} = \texttt{0} \qquad \forall\texttt{B (x:B) (l:list B), @length B (x::l)} = \texttt{S (@length B l)}}{\texttt{@length A l} = \texttt{n+1} \rightarrow \texttt{l} \neq \texttt{[]}}$$

Example

1. inductive types
2. definitions
3. expansion
4. fixpoints
5. p. matching
6. polymorphism

$$\texttt{A:Type} \quad \texttt{l:list A} \quad \texttt{n:nat}$$

$$\forall\texttt{(x:nat), 0} \neq \texttt{S x} \quad \forall\texttt{(x y:nat), S x} = \texttt{S y} \rightarrow \texttt{x} = \texttt{y}$$
$$\forall\texttt{(x:A) (l:list A), []} \neq \texttt{x::l}$$
$$\forall\texttt{(x y:A) (l m:list A), x::l} = \texttt{y::l} \rightarrow \texttt{x} = \texttt{y} \wedge \texttt{l} = \texttt{m}$$

$$\texttt{length} = \texttt{fun (B:Type)} \Rightarrow \texttt{fix length (l:list B) := match l...}$$

$$\forall\texttt{B (l:list B), length B l} =$$
$$\texttt{(fun (B:Type)} \Rightarrow \texttt{fix length (l:list B) := match l...) B l}$$

$$\forall\texttt{B (l:list B), @length B l} = \texttt{match l...}$$

$$\frac{\texttt{@length A []} = \texttt{0} \qquad \forall\texttt{(x:A) (l:list A), @length A (x::l)} = \texttt{S (@length A l)}}{\texttt{@length A l} = \texttt{n+1} \rightarrow \texttt{l} \neq \texttt{[]}}$$

Example

1. inductive types
2. definitions
3. expansion
4. fixpoints
5. p. matching
6. polymorphism

$$\texttt{A:Type} \quad \texttt{l:list A} \quad \texttt{n:nat}$$

$$\forall\texttt{(x:nat), 0} \neq \texttt{S x} \quad \forall\texttt{(x y:nat), S x} = \texttt{S y} \rightarrow \texttt{x} = \texttt{y}$$
$$\forall\texttt{(x:A) (l:list A), []} \neq \texttt{x::l}$$
$$\forall\texttt{(x y:A) (l m:list A), x::l} = \texttt{y::l} \rightarrow \texttt{x} = \texttt{y} \wedge \texttt{l} = \texttt{m}$$

$$\texttt{length} = \texttt{fun (B:Type)} \Rightarrow \texttt{fix length (l:list B) := match l...}$$

$$\forall\texttt{(l:list A), length} =$$
$$\texttt{(fun (B:Type)} \Rightarrow \texttt{fix length (l:list B) := match l...) A l}$$

$$\forall\texttt{(l:list A), @length A l} = \texttt{match l...}$$

$$\frac{\texttt{@length A []} = \texttt{0} \qquad \forall\texttt{(x:A) (l:list A), @length A (x::l)} = \texttt{S (@length A l)}}{\texttt{@length A l} = \texttt{n+1} \rightarrow \texttt{l} \neq \texttt{[]}}$$

Example

1. inductive types
2. definitions
3. expansion
4. fixpoints
5. p. matching
6. polymorphism

$$\texttt{A:Type} \quad \texttt{l:list A} \quad \texttt{n:nat}$$

$$\forall\texttt{(x:nat), O} \neq \texttt{S x} \quad \forall\texttt{(x y:nat), S x} = \texttt{S y} \rightarrow \texttt{x} = \texttt{y}$$
$$\forall\texttt{(x:A) (l:list A), []} \neq \texttt{x::l}$$
$$\forall\texttt{(x y:A) (l m:list A), x::l} = \texttt{y::l} \rightarrow \texttt{x} = \texttt{y} \wedge \texttt{l} = \texttt{m}$$

$$\texttt{length} = \texttt{fun (B:Type)} \Rightarrow \texttt{fix length (l:list B) := match l}\ldots$$

$$\forall\texttt{(l:list A), length} =$$
$$\texttt{(fun (B:Type)} \Rightarrow \texttt{fix length (l:list B) := match l}\ldots\texttt{) A l}$$

$$\forall\texttt{(l:list A), @length A l} = \texttt{match l}\ldots$$

$$\frac{\texttt{@length A []} = \texttt{O} \qquad \forall\texttt{(x:A) (l:list A), @length A (x::l)} = \texttt{S (@length A l)}}{\texttt{@length A l} = \texttt{n+1} \rightarrow \texttt{l} \neq \texttt{[]}}$$

Relies on multiple meta-programming tools for Coq

Currently:

- facts are generated with MetaCoq or coq-elpi
- facts are proved in Ltac
- everything is combined in Ltac

Outline

Ongoing: applications

- FreeSpec: certifying impure computations in Coq

  complementary to FreeSpec tactics to reason about programs

- Coq Tezos of OCaml: translation and verification of the Tezos protocol

  automate verification as much as possible

Ongoing: new transformations

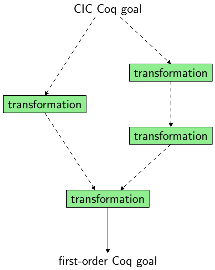
Ex from FreeSpec: automatically decide some inductive predicates

```
Inductive doors_o_caller : Ω → forall (a : Type),
                                        DOORS a → Prop :=
  | req_is_open (d : door) (ω : Ω) :
                          doors_o_caller ω bool (IsOpen d)
  | req_toggle (d : door) (ω : Ω) :
          (sel d ω = false →sel (co d) ω = false) →
                         doors_o_caller ω unit (Toggle d).
```

↓

```
Definition doors_o_caller : Ω → forall (a : Type),
                                        DOORS a → Prop :=
  fun ω a D ⇒ match D with
                 | IsOpen _ ⇒ true
                 | Toggle d ⇒ implb (negb (sel d ω))
                                    (negb (sel (co d) ω))
              end.
```

Future: adaptative strategies and user-defined transformations



Outline

Thanks

opam install coq−sniper

https://github.com/smtcoq/sniper

Many thanks to Nomadic Labs and Inria