

# Practical proof reconstruction for first-order logic and set-theoretical constructions

Clément Hurlin



January 25, 2007

# Outline

- 1 Introduction
  - Two different worlds
  - Proof reconstruction
  - General procedure
  - Targeted languages
- 2 Quantifier free first-order logic
- 3 First-order logic without existential quantifiers
- 4 Full first-order logic
- 5 Set-theoretical constructions
  - Tactic-style
  - Reflection
  - Performances
- 6 Conclusion and further work



## Automatic tools

- First-order logic (FOL).
- Very efficient.
- Large piece of code:  
difficult to prove  
soundness.

## Interactive tools

- Higher order logic.
- A few automatic tactics.
- Yet, does not handle big  
formulas.
- High degree of confidence  
(*e.g.* Isabelle or Coq).

- Proof search in a tool and verification in another one.
- Combination of an automatic and an interactive tool.
- Advantages:
  - ▶ *automation* of the first.
  - ▶ *soundness* and *expressiveness* of the latter.
- Very generic approach.
- In our work: integration of haRVey within Isabelle.

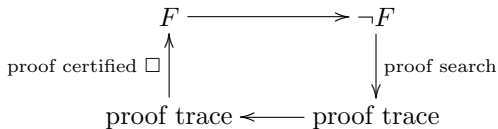
# Introduction

## General procedure



Isabelle

haRVey



Procedure



# Introduction

## General procedure



Isabelle

$F$  (validity)

haRVey

$\neg F$  (satisfiability)

$\neg F$  unsatisfiable

$\neg F$  satisfiable

Isabelle

reconstruction

failure

- The proof is entirely certified by the Isabelle kernel.
- The proof trace must be expressive enough for an automatic certification.
- Thus, no user interaction at all.



- First-order logic.
- Set-theoretical constructions:

$$(X \cap Y = \emptyset) \wedge (X \setminus Z = X) \wedge (Y \cap Z \neq \emptyset) \longrightarrow X \cap (Y \cup Z) = \emptyset,$$

- With a few restrictions, set-theoretical constructions can be reduced to first-order logic.
- They are encountered when using B or TLA+.

# Introduction

## Targeted languages



$$A \subset (\{a, b\} \cap D) \quad (\text{SET})$$

$$\forall x. [A(x) \longrightarrow (x = a \vee x = b) \wedge D(x)] \wedge \\ \exists x. [\neg A(x) \wedge (x = a \vee x = b) \wedge D(x)] \quad (\text{FOL})$$

$$\forall x. [A(x) \longrightarrow (x = a \vee x = b) \wedge D(x)] \wedge \\ (\neg A(c) \wedge (c = a \vee c = b) \wedge D(c)) \quad (\forall\text{FOL})$$

$$(A(a) \longrightarrow (a = a \vee a = b) \wedge D(a)) \wedge \\ (A(b) \longrightarrow (b = a \vee b = b) \wedge D(b)) \wedge \\ (A(c) \longrightarrow (c = a \vee c = b) \wedge D(c)) \wedge \\ (\neg A(c) \wedge (c = a \vee c = b) \wedge D(c)) \quad (\text{QF-FOL})$$



- Previous work by Pascal Fontaine, Stephan Merz *et al.*
- Uses a congruence closure algorithm.
- This algorithm decide satisfiability of formulas containing uninterpreted symbols and equalities.
- It builds equivalence classes between equal terms.

# Quantifier free first-order logic

- Previous work by Pascal Fontaine, Stephan Merz *et al.*
- Uses a congruence closure algorithm.
- This algorithm decide satisfiability of formulas containing uninterpreted symbols and equalities.
- It builds equivalence classes between equal terms.

Given the following hypotheses :  $a = b, f(b) = f(c)$ .

$a$                        $b$                        $c$

$f(a)$                        $f(b)$                        $f(c)$

# Quantifier free first-order logic

- Previous work by Pascal Fontaine, Stephan Merz *et al.*
- Uses a congruence closure algorithm.
- This algorithm decide satisfiability of formulas containing uninterpreted symbols and equalities.
- It builds equivalence classes between equal terms.

Given the following hypotheses :  $a = b, f(b) = f(c)$ .

$$a \text{ ————— } b \qquad c$$

$$f(a) \qquad f(b) \text{ ————— } f(c)$$

# Quantifier free first-order logic

- Previous work by Pascal Fontaine, Stephan Merz *et al.*
- Uses a congruence closure algorithm.
- This algorithm decide satisfiability of formulas containing uninterpreted symbols and equalities.
- It builds equivalence classes between equal terms.

Given the following hypotheses :  $a = b, f(b) = f(c)$ .

$$a \text{ ————— } b \qquad c$$

$$f(a) \text{ — }^{\mathbf{C}} \text{ — } f(b) \text{ ————— } f(c)$$



## Properties

- Reasoning is reconstructed by following the path between terms.
- This allows to reconstruct the proof in a straight-forward way.
- Reasoning can be decomposed into 4 simple rules:

## Properties

- Reasoning is reconstructed by following the path between terms.
- This allows to reconstruct the proof in a straight-forward way.
- Reasoning can be decomposed into 4 simple rules:

1 substitution and contradiction  $\llbracket s = t; P(s); \neg P(t) \rrbracket \implies False$ .

## Properties

- Reasoning is reconstructed by following the path between terms.
- This allows to reconstruct the proof in a straight-forward way.
- Reasoning can be decomposed into 4 simple rules:

- substitution and contradiction  $\llbracket s = t; P(s); \neg P(t) \rrbracket \implies False$ .
- contradiction  $\llbracket s = t; s \neq t \rrbracket \implies False$ .

# Quantifier free first-order logic

## Properties

- Reasoning is reconstructed by following the path between terms.
- This allows to reconstruct the proof in a straight-forward way.
- Reasoning can be decomposed into 4 simple rules:

- substitution and contradiction  $\llbracket s = t; P(s); \neg P(t) \rrbracket \implies False$ .
- contradiction  $\llbracket s = t; s \neq t \rrbracket \implies False$ .
- congruence  $\llbracket f = g; x = y \rrbracket \implies f(x) = g(y)$ .



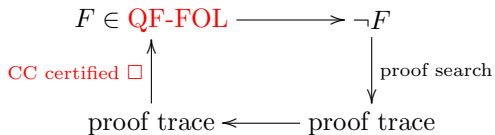
## Properties

- Reasoning is reconstructed by following the path between terms.
- This allows to reconstruct the proof in a straight-forward way.
- Reasoning can be decomposed into 4 simple rules:

- substitution and contradiction  $\llbracket s = t; P(s); \neg P(t) \rrbracket \implies False$ .
- contradiction  $\llbracket s = t; s \neq t \rrbracket \implies False$ .
- congruence  $\llbracket f = g; x = y \rrbracket \implies f(x) = g(y)$ .
- reflexivity, symmetry and transitivity of equality  
 $\llbracket a = b; c = b \rrbracket \implies a = c$ .

Isabelle

haRVey



Procedure for QF-FOL

- Brute force instantiation of universal quantifiers:

$$(\forall x.P(x)) \longrightarrow P(a_1) \wedge \cdots \wedge P(a_n)$$

where  $\{a_1, \dots, a_n\}$  is the Herbrand universe.

- Terminates if:

- (i) Instantiations are selected in a fair way.
- (ii) Formula is unsatisfiable.

# First-order logic without existential quantifiers

- Brute force instantiation of universal quantifiers:

$$(\forall x.P(x)) \longrightarrow P(a_1) \wedge \cdots \wedge P(a_n)$$

where  $\{a_1, \dots, a_n\}$  is the Herbrand universe.

- Terminates if:

(i) Instantiations are selected in a fair way.

(ii) Formula is unsatisfiable.

- **Efficiency:** certifying instantiations does not deal with the logical structure of the formula.

$$a \neq b \wedge ((P \wedge \neg Q) \vee \forall x.x = a)$$

# First-order logic without existential quantifiers

- Brute force instantiation of universal quantifiers:

$$(\forall x.P(x)) \longrightarrow P(a_1) \wedge \dots \wedge P(a_n)$$

where  $\{a_1, \dots, a_n\}$  is the Herbrand universe.

- Terminates if:

(i) Instantiations are selected in a fair way.

(ii) Formula is unsatisfiable.

- **Efficiency:** certifying instantiations does not deal with the logical structure of the formula.

$$a \neq b \wedge ((P \wedge \neg Q) \vee \forall x.x = a)$$

becomes

$$a \neq b \wedge ((P \wedge \neg Q) \vee \forall x.x = a) \wedge$$

$$\wedge (\forall x.[x = a] \longrightarrow a = a) \wedge (\forall x.[x = a] \longrightarrow b = a)$$

- A boolean abstraction mechanism is used to handle the boolean structure of the formula
- This stage is reconstructed by calling a SAT solver (previous work by Tjark Weber):

$$a \neq b \wedge ((P \wedge \neg Q) \vee \forall x. x = a)$$

becomes

$$a \neq b \wedge ((P \wedge \neg Q) \vee \forall x. x = a) \wedge$$
$$(\forall x. [x = a] \longrightarrow a = a) \wedge (\forall x. [x = a] \longrightarrow b = a)$$

- A boolean abstraction mechanism is used to handle the boolean structure of the formula
- This stage is reconstructed by calling a SAT solver (previous work by Tjark Weber):

$$a \neq b \wedge ((P \wedge \neg Q) \vee \forall x. x = a)$$

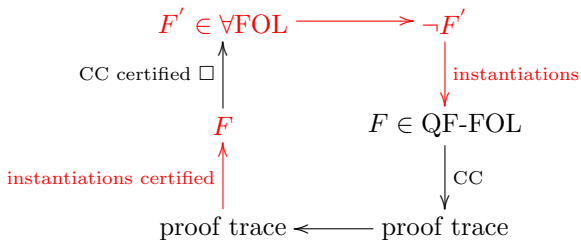
becomes

$$\underbrace{a \neq b}_{\neg A} \wedge ((P \wedge \neg Q) \vee \underbrace{\forall x. x = a}_B) \wedge$$

$$\underbrace{(\forall x. [x = a] \longrightarrow a = a)}_B \wedge \underbrace{(\forall x. [x = a] \longrightarrow b = a)}_A$$

Isabelle

haRVey



Procedure for  $\forall\text{FOL}$



- With a proof trace consisting solely of strings, most of the time was spent parsing the trace!
- The proof trace is now written in ML. [▶ example](#)
- However formulas are still strings (thus frequent calls to the Isabelle parser)
- Currently, arguing between Nancy and Munich to switch to a proof trace completely in ML or in XML format.

- Universal and existential quantifiers.
- Pre-processing: existential quantifiers removed by skolemization.
- haRVey's implements **inner** skolemization.

# Full first-order logic

- Universal and existential quantifiers.
- Pre-processing: existential quantifiers removed by skolemization.
- haRVey's implements **inner** skolemization.

Given a formula  $\phi$  each occurrence of a sub-formula of the form  $\exists x.\psi(x)$  is replaced by  $\psi\{x/f(y_1, \dots, y_n)\}$  where :

- (i)  $f$  is a new symbol function.
- (ii)  $\{y_1, \dots, y_n\}$  is the set of free variables of the sub-formula  $\exists x.\psi(x)$ .

# Full first-order logic

- Universal and existential quantifiers.
- Pre-processing: existential quantifiers removed by skolemization.
- haRVey's implements **inner** skolemization.

Given a formula  $\phi$  each occurrence of a sub-formula of the form  $\exists x.\psi(x)$  is replaced by  $\psi\{x/f(y_1, \dots, y_n)\}$  where :

- (i)  $f$  is a new symbol function.
- (ii)  $\{y_1, \dots, y_n\}$  is the set of free variables of the sub-formula  $\exists x.\psi(x)$ .

This transformation involves:

- a sub-formula.
- a set of free variables.

# Full first-order logic

In Isabelle, skolemization is done with the rule *choice* :

$$\frac{\Gamma, \exists f. \forall x. P(x, f(x)) \vdash \Delta}{\Gamma, \forall x. \exists y. P(x, y) \vdash \Delta} \text{choice}$$

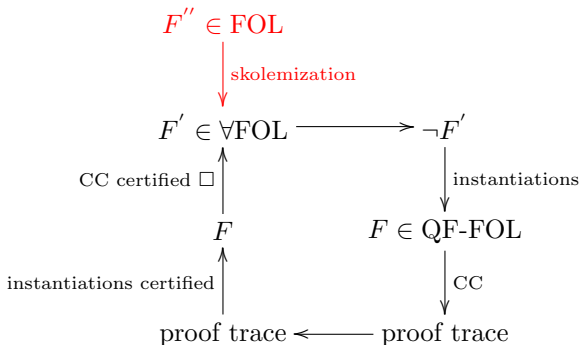
This rule has a purely **local** behavior:

$$\frac{\frac{\frac{\Gamma, \forall x. P \wedge Q(x, f(x)) \vdash \Delta}{\Gamma, \exists f. \forall x. P \wedge Q(x, f(x)) \vdash \Delta} \exists \text{ elimination}}{\Gamma, \forall x. \exists y. P \wedge Q(x, y) \vdash \Delta} \text{choice}}{\Gamma, \forall x. P \wedge \exists y. Q(x, y) \vdash \Delta} \text{extra-scope } \exists$$

- Inner skolemization cannot be simulated with *choice*.
- Thus, inner skolemization not re-playable as it in Isabelle.
- In the implementation, skolemization done directly in Isabelle (by successive applications of *choice*).
- Preceded by a *mini-scoping* step, this technique gives good results.

Isabelle

haRVey



Procedure for FOL

## Set-theoretical constructions

Transformation from SET to FOL by considering sets by their characteristic predicates:

- $a \in A \wedge A \subseteq B \rightsquigarrow \check{A}(a) \wedge \forall x. \check{A}(x) \longrightarrow \check{B}(x)$
- $a \in (A \setminus B) \wedge \emptyset \subseteq \{a\} \rightsquigarrow (\check{A}(a) \wedge \neg \check{B}(a)) \wedge (\forall x. \perp \longrightarrow x = a)$

## Set-theoretical constructions

Transformation from SET to FOL by considering sets by their characteristic predicates:

- $a \in A \wedge A \subseteq B \rightsquigarrow \check{A}(a) \wedge \forall x. \check{A}(x) \longrightarrow \check{B}(x)$
- $a \in (A \setminus B) \wedge \emptyset \subseteq \{a\} \rightsquigarrow (\check{A}(a) \wedge \neg \check{B}(a)) \wedge (\forall x. \perp \longrightarrow x = a)$

In haRVey, this transformation is done by rewriting set symbols:

- $\lambda$ -terms and **quantifiers** appear.
- $\beta$ -reduction and application of extensionality terminates on a FOL formula.

## Set-theoretical constructions

Transformation from SET to FOL by considering sets by their characteristic predicates:

- $a \in A \wedge A \subseteq B \rightsquigarrow \check{A}(a) \wedge \forall x. \check{A}(x) \longrightarrow \check{B}(x)$
- $a \in (A \setminus B) \wedge \emptyset \subseteq \{a\} \rightsquigarrow (\check{A}(a) \wedge \neg \check{B}(a)) \wedge (\forall x. \perp \longrightarrow x = a)$

In haRVey, this transformation is done by rewriting set symbols:

- $\lambda$ -terms and **quantifiers** appear.
- $\beta$ -reduction and application of extensionality terminates on a FOL formula.

It has been written in two manners:

- A classical tactic-based method.
- A method relying partially on reflection (up to 10 times faster).



For the logical structure (symbols  $=, \wedge, \vee, \longrightarrow$ ), the following theorem is used:

$$\frac{P = P', Q = Q' \vdash P' \alpha Q'}{\vdash P \alpha Q}$$

For formulas involving sets (symbols  $\in, \subseteq, \subset, =$ ) the following theorems are used:

$$\frac{\vdash \check{A}(a)}{\vdash a \in A}$$

$$\frac{\vdash \forall x. \check{A}(x) \longrightarrow \check{B}(x)}{\vdash A \subseteq B}$$

$$\frac{\vdash \forall x. [\check{A}(x) \longrightarrow \check{B}(x)] \wedge \exists x. [\check{B}(x) \wedge \neg \check{A}(x)]}{\vdash A \subset B}$$

$$\frac{\vdash \forall x. \check{A}(x) = \check{B}(x)}{\vdash A = B}$$



Handling the structure of sets involved in the goal is done with the following theorems:

$$\frac{\vdash \forall x. \check{A}(x) \wedge \check{B}(x)}{\vdash \forall x. x \in (A \cap B)} \qquad \frac{\vdash \forall x. \check{A}(x) \vee \check{B}(x)}{\vdash \forall x. x \in (A \cup B)}$$

$$\frac{\vdash \forall x. \check{A}(x) \wedge \neg \check{B}(x)}{\vdash \forall x. x \in (A \setminus B)}$$



- **Partial:** Only terms representing sets are reflected (not formulas).
- The transformation to predicates is done using Isabelle's simplifier
  - ▶ Better performance could be obtained by code extraction.

# Set-theoretical constructions

## Performances

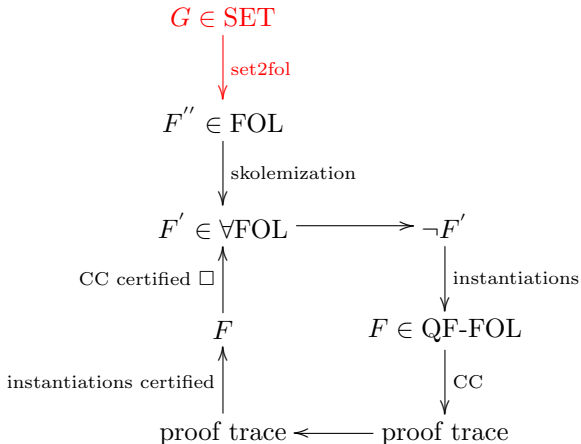


| test # | size  | induction (sec) | reflection (sec) |
|--------|-------|-----------------|------------------|
| 1      | 3401  | 2,13            | 0,72             |
| 2      | 3908  | 15,69           | 1,56             |
| 3      | 7071  | 4,11            | 2,08             |
| 4      | 4603  | 4,01            | 2,40             |
| 5      | 5906  | 59,12           | 2,85             |
| 6      | 7388  | failure         | 1,02             |
| 7      | 12666 | 65,30           | 6,99             |
| 8      | 11722 | 7,44            | 2,85             |

Induction vs reflection

Isabelle

haRVey



Complete procedure

# Conclusion and further work

## Contributions

- This method brings *expressiveness*, *automation*, and *soundness* together.
- Working implementation proof reconstruction for an expressive language.
- It can be adapted to other tools.

# Conclusion and further work

## Contributions

- This method brings *expressiveness*, *automation*, and *soundness* together.
- Working implementation proof reconstruction for an expressive language.
- It can be adapted to other tools.

## Going further:

- Enhancement of the proof trace format.
- Heuristics to select instantiations efficiently.
- Adaptation to other theories such as arithmetic.
- Definition of a standard trace format ?

Reflection consists in replacing proof search by computation :

- SET formulas are reflected by formulas  $\mathcal{SET}$  thanks to  $reify : \text{SET} \longrightarrow \mathcal{SET}$  and  $set2fol : \mathcal{SET} \longrightarrow \text{FOL}$ .
- We prove :  $\forall S. (\llbracket set2fol(S) \rrbracket) = \llbracket S \rrbracket$  (1)

Beginning with a SET formula  $S'$ , we obtain a FOL formula  $F$  :

- 1  $reify$  is applied to  $S'$  and yields  $S$  belonging to  $\mathcal{SET}$ . We prove  $(\llbracket S' \rrbracket) = \llbracket S \rrbracket$ .
- 2  $set2fol$  is applied to  $S$  and yields a formula  $F$  belonging to FOL. By instantiating (1), we have  $\llbracket S \rrbracket = (\llbracket F \rrbracket)$ .

By transitivity  $(\llbracket S' \rrbracket) = (\llbracket F \rrbracket)$ .

## Annex

ML types :

- `rv_proof_inst`: string list
- `rv_proof_congr`: ((string \* string) list \* string) list

▶ back

```
(rv_proof_inst :=
[
  "( $\forall (x::'a). ((b::'a) = ((g::'a \Rightarrow 'a)(x::'a))) \rightarrow ((b::'a) = ((g::'a \Rightarrow 'a)((g::'a \Rightarrow 'a)(b::'a))))$ )",
  "( $\forall (x::'a). ((b::'a) = ((g::'a \Rightarrow 'a)(x::'a))) \rightarrow ((b::'a) = ((g::'a \Rightarrow 'a)((f::'a \Rightarrow 'a)(a::'a))))$ )",
  "( $\forall (x::'a). ((b::'a) = ((g::'a \Rightarrow 'a)(x::'a))) \rightarrow ((b::'a) = ((g::'a \Rightarrow 'a)(a::'a)))$ )",
  "( $\forall (x::'a). ((b::'a) = ((g::'a \Rightarrow 'a)(x::'a))) \rightarrow ((b::'a) = ((g::'a \Rightarrow 'a)(b::'a)))$ )"
]);

rv_proof_congr :=
[
  (
    [
      ("INEQ" , "[ | ((b::'a) = ((f::'a \Rightarrow 'a)(a::'a))); (~ ((b::'a) = ((f::'a \Rightarrow 'a)(a::'a))) ] ==> False)",
      ("TRANS" , "[ | ((b::'a) = ((g::'a \Rightarrow 'a)(a::'a))); (((g::'a \Rightarrow 'a)(b::'a)) = ((g::'a \Rightarrow 'a)(a::'a))); ((b::'a) = ((f::'a \Rightarrow 'a)(a::'a)))" ,
      ("CONGR" , "[ | ((a::'a) = (b::'a)) ] ==> (((g::'a \Rightarrow 'a)(b::'a)) = ((g::'a \Rightarrow 'a)(a::'a)))"
    ],
    "( (b::'a) = ((f::'a \Rightarrow 'a)(a::'a)) | (~ ((b::'a) = ((g::'a \Rightarrow 'a)(a::'a))) | (~ (((f::'a \Rightarrow 'a)(a::'a)) = ((g::'a \Rightarrow 'a)(b::'a))) | (~ ((a::'a) = (b::'a)))"
  )
]
```