First-Order Logic According to Harrison

Alexander Birch Jensen, Anders Schlichtkrull &
Jørgen Villadsen, DTU Compute, Denmark

14 December 2021

Abstract

We present a certified declarative first-order prover with equality based on John Harrison's Handbook of Practical Logic and Automated Reasoning, Cambridge University Press, 2009. ML code reflection is used such that the entire prover can be executed within Isabelle as a very simple interactive proof assistant. As examples we consider Pelletier's problems 1-46.

Contents

Preample

Preliminary formalizations are described here:

Alexander Birch Jensen: *Development and Verification of a Proof Assistant.* Master's Thesis, Technical University of Denmark, 2016. http://findit.dtu.dk/en/catalog/2345011633

Alexander Birch Jensen, Anders Schlichtkrull & Jørgen Villadsen: *Verification of an LCF-Style First-Order Prover with Equality.* Isabelle Workshop 2016. https://github.com/logic-tools/sml-handbook

FOL-Harrison

```
theory FOL_Harrison
imports
  Main
begin
```

Module Proven

Syntax of first-order logic

```
type_synonym id = String.literal

datatype tm = Var id | Fn id "tm list"

datatype 'a fm = Truth | Falsity | Atom 'a | Imp "'a fm" "'a fm" | Iff "'a fm" "'a fm" |
    And "'a fm" "'a fm" | Or "'a fm" "'a fm" | Not "'a fm" | Exists id "'a fm" | Forall id "'a fm"

datatype fol = Rl id "tm list"

datatype "thm" = Thm (concl: "fol fm")
```

Definition of rules and axioms

```
abbreviation (input) "fail_thm ≡ Thm Truth"

definition fol_equal :: "fol fm ⇒ fol fm ⇒ bool"
where
  "fol_equal p q ≡ p = q"

definition zip_eq :: "tm list ⇒ tm list ⇒ fol fm list"
where
  "zip_eq l l' ≡ map (λ(t, t'). Atom (Rl (STR ''='') [t, t'])) (zip l l')"

primrec occurs_in :: "id ⇒ tm ⇒ bool" and occurs_in_list :: "id ⇒ tm list ⇒ bool"
where
  "occurs_in i (Var x) = (i = x)" |
  "occurs_in i (Fn _ l) = occurs_in_list i l" |
  "occurs_in_list _ [] = False" |
  "occurs_in_list i (h # t) = (occurs_in i h ∨ occurs_in_list i t)"

primrec free_in :: "id ⇒ fol fm ⇒ bool"
where
  "free_in _ Truth = False" |
  "free_in _ Falsity = False" |
  "free_in i (Atom a) = (case a of Rl _ l ⇒ occurs_in_list i l)" |
  "free_in i (Imp p q) = (free_in i p ∨ free_in i q)" |
  "free_in i (Iff p q) = (free_in i p ∨ free_in i q)" |
  "free_in i (And p q) = (free_in i p ∨ free_in i q)" |
  "free_in i (Or p q) = (free_in i p ∨ free_in i q)" |
  "free_in i (Not p) = free_in i p" |
  "free_in i (Exists x p) = (i ≠ x ∧ free_in i p)" |
  "free_in i (Forall x p) = (i ≠ x ∧ free_in i p)"

primrec equal_length :: "tm list ⇒ tm list ⇒ bool"
where
  "equal_length l [] = (case l of [] ⇒ True | _ # _ ⇒ False)" |
  "equal_length l (_ # r') = (case l of [] ⇒ False | _ # l' ⇒ equal_length l' r')"

definition modusponens :: "thm ⇒ thm ⇒ thm"
```

```
where
  "modusponens s s' ≡ case concl s of Imp p q ⇒
      let p' = concl s' in if fol_equal p p' then Thm q else fail_thm | _ ⇒ fail_thm"

definition gen :: "id ⇒ thm ⇒ thm"
where
  "gen x s ≡ Thm (Forall x (concl s))"

definition axiom_addimp :: "fol fm ⇒ fol fm ⇒ thm"
where
  "axiom_addimp p q ≡ Thm (Imp p (Imp q p))"

definition axiom_distribimp :: "fol fm ⇒ fol fm ⇒ fol fm ⇒ thm"
where
  "axiom_distribimp p q r ≡ Thm (Imp (Imp p (Imp q r)) (Imp (Imp p q) (Imp p r )))"

definition axiom_doubleneg :: "fol fm ⇒ thm"
where
  "axiom_doubleneg p ≡ Thm (Imp (Imp (Imp p Falsity) Falsity) p)"

definition axiom_allimp :: "id ⇒ fol fm ⇒ fol fm ⇒ thm"
where
  "axiom_allimp x p q ≡ Thm (Imp (Forall x (Imp p q)) (Imp (Forall x p) (Forall x q)))"

definition axiom_impall :: "id ⇒ fol fm ⇒ thm"
where
  "axiom_impall x p ≡ if ¬ free_in x p then Thm (Imp p (Forall x p)) else fail_thm"

definition axiom_existseq :: "id ⇒ tm ⇒ thm"
where
  "axiom_existseq x t ≡ if ¬ occurs_in x t
      then Thm (Exists x (Atom (Rl (STR ''='') [Var x, t]))) else fail_thm"

definition axiom_eqrefl :: "tm ⇒ thm"
where
  "axiom_eqrefl t ≡ Thm (Atom (Rl (STR ''='') [t, t]))"

definition axiom_funcong :: "id ⇒ tm list ⇒ tm list ⇒ thm"
where
  "axiom_funcong i l l' ≡ if equal_length l l'
      then Thm (foldr Imp (zip_eq l l') (Atom (Rl (STR ''='') [Fn i l, Fn i l']))) else fail_thm"

definition axiom_predcong :: "id ⇒ tm list ⇒ tm list ⇒ thm"
where
  "axiom_predcong i l l' ≡ if equal_length l l'
      then Thm (foldr Imp (zip_eq l l') (Imp (Atom (Rl i l)) (Atom (Rl i l')))) else fail_thm"

definition axiom_iffimp1 :: "fol fm ⇒ fol fm ⇒ thm"
where
  "axiom_iffimp1 p q ≡ Thm (Imp (Iff p q) (Imp p q))"

definition axiom_iffimp2 :: "fol fm ⇒ fol fm ⇒ thm"
where
  "axiom_iffimp2 p q ≡ Thm (Imp (Iff p q) (Imp q p))"

definition axiom_impiff :: "fol fm ⇒ fol fm ⇒ thm"
where
  "axiom_impiff p q ≡ Thm (Imp (Imp p q) (Imp (Imp q p) (Iff p q)))"

definition axiom_true :: "thm"
where
  "axiom_true ≡ Thm (Iff Truth (Imp Falsity Falsity))"

definition axiom_not :: "fol fm ⇒ thm"
```

```
where
  "axiom_not p ≡ Thm (Iff (Not p) (Imp p Falsity))"

definition axiom_and :: "fol fm ⇒ fol fm ⇒ thm"
where
  "axiom_and p q ≡ Thm (Iff (And p q) (Imp (Imp p (Imp q Falsity)) Falsity))"

definition axiom_or :: "fol fm ⇒ fol fm ⇒ thm"
where
  "axiom_or p q ≡ Thm (Iff (Or p q) (Not (And (Not p) (Not q))))"

definition axiom_exists :: "id ⇒ fol fm ⇒ thm"
where
  "axiom_exists x p ≡ Thm (Iff (Exists x p) (Not (Forall x (Not p))))"
```

Code generation for rules and axioms

```
export_code
  modusponens gen axiom_addimp axiom_distribimp axiom_doubleneg axiom_allimp axiom_impall
  axiom_existseq axiom_eqrefl axiom_funcong axiom_predcong axiom_iffimp1 axiom_iffimp2
  axiom_impiff axiom_true axiom_not axiom_and axiom_or axiom_exists concl
in SML module_name Proven

code_printing constant fol_equal ⇀ (SML) "_ = _" — More efficient

export_code
  modusponens gen axiom_addimp axiom_distribimp axiom_doubleneg axiom_allimp axiom_impall
  axiom_existseq axiom_eqrefl axiom_funcong axiom_predcong axiom_iffimp1 axiom_iffimp2
  axiom_impiff axiom_true axiom_not axiom_and axiom_or axiom_exists concl
in SML module_name Proven

code_printing constant fol_equal ⇀ (SML) — Delete

export_code
  modusponens gen axiom_addimp axiom_distribimp axiom_doubleneg axiom_allimp axiom_impall
  axiom_existseq axiom_eqrefl axiom_funcong axiom_predcong axiom_iffimp1 axiom_iffimp2
  axiom_impiff axiom_true axiom_not axiom_and axiom_or axiom_exists concl
in SML module_name Proven
```

Semantics of first-order logic

```
definition length2 :: "tm list ⇒ bool"
where
  "length2 l ≡ case l of [_,_] ⇒ True | _ ⇒ False"

primrec — Semantics of terms
  semantics_term :: "(id ⇒ 'a) ⇒ (id ⇒ 'a list ⇒ 'a) ⇒ tm ⇒ 'a" and
  semantics_list :: "(id ⇒ 'a) ⇒ (id ⇒ 'a list ⇒ 'a) ⇒ tm list ⇒ 'a list"
where
  "semantics_term e _ (Var x) = e x" |
  "semantics_term e f (Fn i l) = f i (semantics_list e f l)" |
  "semantics_list _ _ [] = []" |
  "semantics_list e f (t # l) = semantics_term e f t # semantics_list e f l"

primrec — Semantics of formulas
  semantics :: "(id ⇒ 'a) ⇒ (id ⇒ 'a list ⇒ 'a) ⇒ (id ⇒ 'a list ⇒ bool) ⇒ fol fm ⇒ bool"
where
  "semantics _ _ _ Truth = True" |
  "semantics _ _ _ Falsity = False" |
  "semantics e f g (Atom a) = (case a of Rl i l ⇒ if i = STR ''='' ∧ length2 l
      then (semantics_term e f (hd l) = semantics_term e f (hd (tl l)))
      else g i (semantics_list e f l))" |
  "semantics e f g (Imp p q) = (semantics e f g p ⟶ semantics e f g q)" |
```

```
  "semantics e f g (Iff p q) = (semantics e f g p ⟷ semantics e f g q)" |
  "semantics e f g (And p q) = (semantics e f g p ∧ semantics e f g q)" |
  "semantics e f g (Or p q) = (semantics e f g p ∨ semantics e f g q)" |
  "semantics e f g (Not p) = (¬ semantics e f g p)" |
  "semantics e f g (Exists x p) = (∃v. semantics (e(x := v)) f g p)" |
  "semantics e f g (Forall x p) = (∀v. semantics (e(x := v)) f g p)"
```

Definition of proof system

```
inductive OK :: "fol fm ⇒ bool" ("⊢ _" 0)
where
  modusponens:
                    "⊢ concl s ⟹ ⊢ concl s' ⟹ ⊢ concl (modusponens s s')" |
  gen:
                    "⊢ concl s ⟹ ⊢ concl (gen _ s)" |
  axiom_addimp:
                    "⊢ concl (axiom_addimp _ _)" |
  axiom_distribimp:
                    "⊢ concl (axiom_distribimp _ _ _)" |
  axiom_doubleneg:
                    "⊢ concl (axiom_doubleneg _)" |
  axiom_allimp:
                    "⊢ concl (axiom_allimp _ _ _)" |
  axiom_impall:
                    "⊢ concl (axiom_impall _ _)" |
  axiom_existseq:
                    "⊢ concl (axiom_existseq _ _)" |
  axiom_eqrefl:
                    "⊢ concl (axiom_eqrefl _)" |
  axiom_funcong:
                    "⊢ concl (axiom_funcong _ _ _)" |
  axiom_predcong:
                    "⊢ concl (axiom_predcong _ _ _)" |
  axiom_iffimp1:
                    "⊢ concl (axiom_iffimp1 _ _)" |
  axiom_iffimp2:
                    "⊢ concl (axiom_iffimp2 _ _)" |
  axiom_impiff:
                    "⊢ concl (axiom_impiff _ _)" |
  axiom_true:
                    "⊢ concl axiom_true" |
  axiom_not:
                    "⊢ concl (axiom_not _)" |
  axiom_and:
                    "⊢ concl (axiom_and _ _)" |
  axiom_or:
                    "⊢ concl (axiom_or _ _)" |
  axiom_exists:
                    "⊢ concl (axiom_exists _ _)"
```

proposition "⊢ Imp p p"
⟨*proof*⟩

Soundness of proof system

lemma map':

```
  "¬ occurs_in x t ⟹ semantics_term e f t = semantics_term (e(x := v)) f t"
  "¬ occurs_in_list x l ⟹ semantics_list e f l = semantics_list (e(x := v)) f l"
```

⟨*proof*⟩

lemma map:

```
  "¬ free_in x p ⟹ semantics e f g p ⟷ semantics (e(x := v)) f g p"
```
⟨*proof*⟩

lemma length2_equiv:
```
  "length2 l ⟷ [hd l, hd (tl l)] = l"
```
⟨*proof*⟩

lemma equal_length_sym:
```
  "equal_length l l' ⟹ equal_length l' l"
```
⟨*proof*⟩

lemma equal_length2:
```
  "equal_length l l' ⟹ length2 l ⟷ length2 l'"
```
⟨*proof*⟩

lemma imp_chain_equiv:
```
  "semantics e f g (foldr Imp l p) ⟷ (∀q ∈ set l. semantics e f g q) ⟶ semantics e f g p"
```
⟨*proof*⟩

lemma imp_chain_zip_eq:
```
  "equal_length l l' ⟹
      semantics e f g (foldr Imp (zip_eq l l') p) ⟷
      semantics_list e f l = semantics_list e f l' ⟶ semantics e f g p"
```
⟨*proof*⟩

lemma funcong:
```
  "equal_length l l' ⟹
      semantics e f g (foldr Imp (zip_eq l l') (Atom (Rl (STR ''='') [Fn i l, Fn i l'])))"
```
⟨*proof*⟩

lemma predcong:
```
  "equal_length l l' ⟹
      semantics e f g (foldr Imp (zip_eq l l') (Imp (Atom (Rl i l)) (Atom (Rl i l'))))"
```
⟨*proof*⟩

theorem soundness:
```
  "⊢ p ⟹ semantics e f g p"
```
⟨*proof*⟩

corollary "¬ (⊢ Falsity)"
⟨*proof*⟩

ML Code Reflection

code_reflect
```
  Proven
```
datatypes
```
  fm = Falsity | Truth | Atom | Imp | Iff | And | Or | Not | Exists | Forall
```
and
```
  tm = Var | Fn
```
and
```
  fol = Rl
```
functions
```
  modusponens gen axiom_addimp axiom_distribimp axiom_doubleneg axiom_allimp axiom_impall
  axiom_existseq axiom_eqrefl axiom_funcong axiom_predcong axiom_iffimp1 axiom_iffimp2
  axiom_impiff axiom_true axiom_not axiom_and axiom_or axiom_exists concl
```

⟨*ML*⟩

Main Examples

⟨*ML*⟩

Other Examples

$\langle ML \rangle$

end

Acknowledgements

The SML code is based on the OCaml code accompanying John Harrison's Handbook of Practical Logic and Automated Reasoning, Cambridge University Press, 2009

Thanks to Jasmin Blanchette, Asta Halkjær From, John Bruntse Larsen, Andrei Popescu and Tom Ridge for discussions.