TFPIE 2021

Teaching Automated Reasoning and Formally Verified Functional Programming in Agda and Isabelle/HOL

Asta Halkjær From, Jørgen Villadsen

DTU Compute, Denmark

Paper (20 pages) available on workshop page

We formalize micro provers for propositional logic in the proof assistants Isabelle/HOL and Agda

We use the provers in an advanced automated reasoning course at the Technical University of Denmark (DTU) where they concretize discussions of termination, soundness and completeness

The students are familiar with functional programming beforehand but formalizing the provers, and other programs, introduces the students to formally verified functional programming in a proof assistant

Automated Reasoning:

Math theorems, logic puzzles, distributed systems & functional programming

2020 27 students 2021 52 students

https://kurser.dtu.dk/course/02256

The formalizations/programs, 64 lines (47 sloc) in file Micro_Prover.thy and 416 lines (333 sloc) in file microprover.agda are available here:

https://github.com/logic-tools/micro

Agda, Coq, Lean — **Dependent Types**

HOL4, HOL Light, Isabelle/HOL — **Simple Types**

Isabelle/HOL screenshot with automated reasoning challenge problem (1970)

https://en.wikipedia.org/wiki/McCarthy_91_function

```
theory McCarthy imports Main begin

— ‹McCarthy 91 function›

function M :: ‹int ⇒ int› where ‹M i = (if 100 < i then i - 10 else M (M (i + 11)))›

  sorry

termination

  sorry

theorem ‹M i = (if 100 < i then i - 10 else 91)›

  sorry

end
```

Prover

Input: A formula

Output: YES / NO

$A \longrightarrow A$

$A \longrightarrow B$

$A \longrightarrow B \longrightarrow A$

$\neg\neg A \longrightarrow A$

$(A \longrightarrow B \longrightarrow C) \longrightarrow (A \longrightarrow B) \longrightarrow A \longrightarrow C$

$A \wedge B \longrightarrow A$

$(A \longrightarrow C) \longrightarrow (B \longrightarrow C) \longrightarrow A \vee B \longrightarrow C$

$A \wedge B \longrightarrow B$

$A \longrightarrow A \vee B$

$A \longrightarrow B \longrightarrow A \wedge B$

$B \longrightarrow A \vee B$

Sequent Calculus

Formulas p, q, $\ldots$ in classical propositional logic are built from propositional symbols, falsity ($\bot$) and implications ($p \to q$).

Abbreviations:

$$\neg p \equiv p \to \bot \qquad p \wedge q \equiv \neg(p \to \neg q) \qquad p \vee q \equiv \neg p \to q$$

Let Γ and Δ be finite sets of formulas.

The axioms of the sequent calculus are of the form:

$$\Gamma \cup \{p\} \vdash \Delta \cup \{p\} \qquad \Gamma \cup \{\bot\} \vdash \Delta$$

The rules of the sequent calculus are left and right introduction rules:

$$\frac{\Gamma \vdash \Delta \cup \{p\} \qquad \Gamma \cup \{q\} \vdash \Delta}{\Gamma \cup \{p \to q\} \vdash \Delta} \qquad \frac{\Gamma \cup \{p\} \vdash \Delta \cup \{q\}}{\Gamma \vdash \Delta \cup \{p \to q\}}$$

$$\dfrac{\dfrac{\dfrac{\mathrm{A} \vdash \mathrm{B}, \mathrm{A}}{\vdash \mathrm{A} \to \mathrm{B}, \mathrm{A}}\,(\to\mathrm{r}) \qquad \mathrm{A} \vdash \mathrm{A}}{(\mathrm{A} \to \mathrm{B}) \to \mathrm{A} \vdash \mathrm{A}}\,(\to\mathrm{l})}{\vdash ((\mathrm{A} \to \mathrm{B}) \to \mathrm{A}) \to \mathrm{A}}\,(\to\mathrm{r})$$

Proof in sequent calculus from logitext.mit.edu

Interactive Theorem Prover (ITP)

Examples	$p \to p$	1 proof step
	$p \to (p \to q) \to q$	3 proof steps
	$p \to q \to q \to p$	4 proof steps
	$p \to \neg\neg p$	4 proof steps using abbreviation for $\neg$
Exercises	$p \to q \to p$	3 proof steps
	$(p \to q \to r) \to (p \to q) \to p \to r$	9 proof steps
	$\neg p \to \neg\neg\neg p$	3 proof steps using abbreviation for $\neg$
	$p \vee \neg p$	1 proof step using abbreviation for $\neg$ and $\vee$
Assignment	$(p \to q) \to p \to q$	1 proof step
	$\neg\neg p \to p$	5 proof steps using abbreviation for $\neg$
	$p \wedge (p \to q) \to q$	7 proof steps using abbreviation for $\wedge$
	$p \wedge q \to r \to p \wedge r$	10 proof steps using abbreviation for $\wedge$

Automatic Theorem Prover (ATP)

Integration of Standard ML in Isabelle

```
structure Micro_Prover : sig
  datatype 'a form = Pro of 'a | Falsity | Imp of 'a form * 'a form
  val prover : 'a HOL.equal -> 'a form -> bool
end = struct

datatype 'a form = Pro of 'a | Falsity | Imp of 'a form * 'a form;

fun member A_ uu [] = false
  | member A_ m (n :: a) = (if HOL.eq A_ m n then true else member A_ m a);

fun common A_ uu [] = false
  | common A_ a (m :: b) = (if member A_ m a then true else common A_ a b);

fun mp A_ a b (Pro n :: c) [] = mp A_ (n :: a) b c []
  | mp A_ a b c (Pro n :: d) = mp A_ a (n :: b) c d
  | mp A_ uu uv (Falsity :: uw) [] = true
  | mp A_ a b c (Falsity :: d) = mp A_ a b c d
  | mp A_ a b (Imp (p, q) :: c) [] =
    (if mp A_ a b c [p] then mp A_ a b (q :: c) [] else false)
  | mp A_ a b c (Imp (p, q) :: d) = mp A_ a b (p :: c) (q :: d)
  | mp A_ a b [] [] = common A_ a b;

fun prover A_ p = mp A_ [] [] [] [p];
```

Isabelle/HOL theory with proofs for list functions

primrec *member* **where**
⟨ *member* - [] = *False* ⟩ |
⟨ *member m* (*n* # *A*) = (*if m* = *n then True else member m A*) ⟩

lemma *member-iff* [*iff*]: ⟨ *member m A* ⟷ $m \in$ *set A* ⟩
by (*induct A*) *simp-all*

primrec *common* **where**
⟨ *common* - [] = *False* ⟩ |
⟨ *common A* (*m* # *B*) = (*if member m A then True else common A B*) ⟩

lemma *common-iff* [*iff*]: ⟨ *common A B* ⟷ *set A* ∩ *set B* ≠ {} ⟩
by (*induct B*) *simp-all*

Datatypes for formulas

Function for semantics

Abbreviation for sequent calculus

datatype *'a form* = *Pro 'a* | *Falsity* (‹ ⊥ ›) | *Imp* ‹ *'a form* › ‹ *'a form* › (**infix** ‹ → › *0*)

primrec *semantics* **where**
‹ *semantics i* (*Pro n*) = *i n* › |
‹ *semantics* - ⊥ = *False* › |
‹ *semantics i* (*p* → *q*) = (*semantics i p* ⟶ *semantics i q*) ›

abbreviation ‹ *sc X Y i* ≡ (∀*p* ∈ *set X*. *semantics i p*) ⟶ (∃*q* ∈ *set Y*. *semantics i q*) ›

Micro Prover

function *mp* **where**

⟨ *mp A B* (*Pro n* # *C*) [] = *mp* (*n* # *A*) *B C* [] ⟩ |

⟨ *mp A B C* (*Pro n* # *D*) = *mp A* (*n* # *B*) *C D* ⟩ |

⟨ *mp* - - (*Falsity* # -) [] = *True* ⟩ |

⟨ *mp A B C* (*Falsity* # *D*) = *mp A B C D* ⟩ |

⟨ *mp A B* (*Imp p q* # *C*) [] = (*if mp A B C* [*p*] *then mp A B* (*q* # *C*) [] *else False*) ⟩ |

⟨ *mp A B C* (*Imp p q* # *D*) = *mp A B* (*p* # *C*) (*q* # *D*) ⟩ |

⟨ *mp A B* [] [] = *common A B* ⟩

by *pat-completeness simp-all*

termination by (*relation* ⟨ *measure* (λ(-,-,*C*,*D*). $\sum p \leftarrow C$ @ *D*. *size p*) ⟩) *simp-all*

lemma *mp-iff* [*iff*]: ⟨ *mp A B C D* $\longleftrightarrow$ μ *A B C D* = {} ⟩

by (*induct rule*: μ.*induct*) *simp-all*

Main theorem

```
function μ where
  ‹μ A B (Pro n # C) [] = μ (n # A) B C []› |
  ‹μ A B C (Pro n # D) = μ A (n # B) C D› |
  ‹μ - - (⊥ # -) [] = {}› |
  ‹μ A B C (⊥ # D) = μ A B C D› |
  ‹μ A B ((p → q) # C) [] = μ A B C [p] ∪ μ A B (q # C) []› |
  ‹μ A B C ((p → q) # D) = μ A B (p # C) (q # D)› |
  ‹μ A B [] [] = (if set A ∩ set B = {} then {A} else {})›
  by pat-completeness simp-all

termination by (relation ‹measure (λ(-,-,C,D). ∑p ← C @ D. size p)›) simp-all

lemma sat: ‹sc (map Pro A @ C) (map Pro B @ D) (λn. n ∈ set L) ⟹ L ∉ μ A B C D›
  by (induct rule: μ.induct) auto

theorem main: ‹(∀i. sc (map Pro A @ C) (map Pro B @ D) i) ⟷ μ A B C D = {}›
  by (induct rule: μ.induct) (auto simp: sat)
```

```
definition ‹prover p ≡ mp [] [] [] [p]›

corollary ‹prover p ⟷ (∀i. semantics i p)›
  unfolding prover-def by (simp flip: main)
```

Entire Isabelle theory shown

Conclusions

- **Proofs that have been informal in previous courses, for instance of termination, can now be verified by the machine, and the provers provide practical examples**
- **Similarly, the formal meta-languages provided by the formalizations clarify boundaries that can be muddled with pen and paper, for instance between syntactic and semantic arguments**
- **We find that the automation available in Isabelle/HOL provides succinctness while the verification in Agda closer resembles functional programming**

References

Asta Halkjær From, Jørgen Villadsen & Patrick Blackburn.

Isabelle/HOL as a Meta-Language for Teaching Logic.

In Pedro Quaresma, Walther Neuper, and João Marcos, editors

9th International Workshop on Theorem Proving Components for Educational Software

ThEdu @ International Joint Conference on Automated Reasoning (IJCAR 2020)

Paris, France, 29th June 2020. Proceedings, volume 328 of EPTCS, pages 18–34, 2020. doi:10.4204/EPTCS.328.2.

Asta Halkjær From, Alexander Birch Jensen, Anders Schlichtkrull & Jørgen Villadsen (2020): *Teaching a Formalized Logical Calculus*. In: *Proceedings of the 8th International Workshop on Theorem proving components for Educational software (ThEdu'19)*, pp. 73–92.

Jørgen Villadsen (2020): *A Micro Prover for Teaching Automated Reasoning*. In: *Seventh Workshop on Practical Aspects of Automated Reasoning (PAAR 2020) — Presentation Only / Online Papers*, pp. 1–12. Available at `http://paar2020.gforge.inria.fr/`.

Jørgen Villadsen (2020): *Tautology Checkers in Isabelle and Haskell*. In Francesco Calimeri, Simona Perri & Ester Zumpano, editors: *Proceedings of the 35th Edition of the Italian Conference on Computational Logic (CILC 2020), Rende, Italy, 13-15 October 2020*, *CEUR Workshop Proceedings* 2710, CEUR-WS.org, pp. 327–341. Available at `http://ceur-ws.org/Vol-2710/paper-21.pdf`.