DTU

lambda DAλS

28-29 JULY 2022

KRAKÓW | POLAND

# Teaching Functional Programmers Logic and Metatheory

Frederik Krogsdal Jacobsen    Jørgen Villadsen

Technical University of Denmark

DTU

lambda DAYS
28-29 JULY 2022
KRAKÓW | POLAND

# Introduction

- We want to teach our students logic
- Pen and paper proofs are good, but the connections to the real world are not always clear
- Textbooks sometimes sweep complexity under the rug
- Functional programming (in Isabelle/HOL) to the rescue!

# Our approach

- Define every concept as a functional program
- Use a proof assistant (a functional language with a *very* strong type system) to prove any theorems
- Students can still read the informal definitions and proofs...
- ...but they can consult the formalization if they need more details
- Implementations allow experiments and easy modifications
- The proof assistant makes sure we didn't forget anything in our proofs
- Isabelle/HOL allows export to languages such as Haskell and Scala

# Our contributions

- Functional implementations of several logical systems
- Formally verified proofs of common properties
- Evaluation of the usefulness of our approach

# Basics

## Syntax

Proposition $P, Q, R, S, \ldots$

Falsity $\bot$

Implication $P \to Q$

## Semantics

We have an interpretation $i$, which assigns truth values to propositions

Proposition: Look up $P$ in $i$

Falsity: Always false

Implication: If $P$ is true, then $Q$ must also be

# What do we want to find out?

Satisfiability: is it possible to choose an *i* that makes the formula true?

Validity: is the formula true no matter what *i* we choose?

We can decide this by trying all possible options, but the number grows exponentially

# Sequent calculus

- Assumptions $\vdash$ Goals

$$\frac{}{\Gamma \cup \{p\} \vdash \Delta \cup \{p\}}\ \textsc{Basic} \qquad \frac{}{\Gamma \cup \{\bot\} \vdash \Delta}\ \textsc{Falsity-Left}$$

$$\frac{\Gamma \vdash \Delta \cup \{p\} \qquad \Gamma \cup \{q\} \vdash \Delta}{\Gamma \cup \{p \to q\} \vdash \Delta}\ \textsc{Implication-Left}$$

$$\frac{\Gamma \cup \{p\} \vdash \Delta \cup \{q\}}{\Gamma \vdash \Delta \cup \{p \to q\}}\ \textsc{Implication-Right}$$

# Properties of the calculus

Soundness: if we can construct a proof using the rules, the formula is valid

Completeness: if the formula is valid, we can construct a proof using the rules

# Properties of the calculus

Soundness: if we can construct a proof using the rules, the formula is valid

Completeness: if the formula is valid, we can construct a proof using the rules

**Proof**

By induction.

# It's never that easy

$$\frac{}{\Gamma \cup \{p\} \vdash \Delta \cup \{p\}}\ \textsc{Basic} \qquad \frac{}{\Gamma \cup \{\bot\} \vdash \Delta}\ \textsc{Falsity-Left}$$

$$\frac{\Gamma \vdash \Delta \cup \{p\} \qquad \Gamma \cup \{q\} \vdash \Delta}{\Gamma \cup \{p \to q\} \vdash \Delta}\ \textsc{Implication-Left}$$

$$\frac{\Gamma \cup \{p\} \vdash \Delta \cup \{q\}}{\Gamma \vdash \Delta \cup \{p \to q\}}\ \textsc{Implication-Right}$$

Why does the process of constructing a proof terminate exactly when the formula is valid?

# Let's make a computer check our work

**datatype** *'a form*
$= Pro\ 'a\ (\langle \cdot \rangle) \mid Falsity\ (\langle \bot \rangle) \mid Imp\ \langle 'a\ form \rangle\ \langle 'a\ form \rangle$ (**infixr** $\langle \rightarrow \rangle$ *0*)

**primrec** *semantics* **where**
$\langle\ semantics\ i\ (\cdot n) = i\ n\ \rangle \mid$
$\langle\ semantics\ \text{-}\ \bot = False\ \rangle \mid$
$\langle\ semantics\ i\ (p \rightarrow q) = (semantics\ i\ p \longrightarrow semantics\ i\ q)\ \rangle$

**abbreviation** $\langle\ sc\ X\ Y\ i \equiv (\forall p \in set\ X.\ semantics\ i\ p) \longrightarrow (\exists q \in set\ Y.\ semantics\ i\ q)\ \rangle$

# Let's make a computer check our work

**primrec** *member* **where**
$\langle$ *member* - [] = *False* $\rangle$ |
$\langle$ *member m* (*n* # *A*) = (*m* = *n* $\vee$ *member m A*) $\rangle$

**lemma** *member-iff* [*iff*]: $\langle$ *member m A* $\longleftrightarrow$ *m* $\in$ *set A* $\rangle$
**by** (*induct A*) *simp-all*

**primrec** *common* **where**
$\langle$ *common* - [] = *False* $\rangle$ |
$\langle$ *common A* (*m* # *B*) = (*member m A* $\vee$ *common A B*) $\rangle$

**lemma** *common-iff* [*iff*]: $\langle$ *common A B* $\longleftrightarrow$ *set A* $\cap$ *set B* $\neq$ {} $\rangle$
**by** (*induct B*) *simp-all*

# Let's make a computer check our work

**function** *mp* **where**

$\langle$ $mp\ A\ B\ (\cdot n \# C)\ [] = mp\ (n \# A)\ B\ C\ []$ $\rangle$ |
$\langle$ $mp\ A\ B\ C\ (\cdot n \# D) = mp\ A\ (n \# B)\ C\ D$ $\rangle$ |
$\langle$ $mp\ \text{-}\ \text{-}\ (\bot \# \text{-})\ [] = True$ $\rangle$ |
$\langle$ $mp\ A\ B\ C\ (\bot \# D) = mp\ A\ B\ C\ D$ $\rangle$ |
$\langle$ $mp\ A\ B\ ((p \rightarrow q) \# C)\ [] = (mp\ A\ B\ C\ [p] \land mp\ A\ B\ (q \# C)\ [])$ $\rangle$ |
$\langle$ $mp\ A\ B\ C\ ((p \rightarrow q) \# D) = mp\ A\ B\ (p \# C)\ (q \# D)$ $\rangle$ |
$\langle$ $mp\ A\ B\ []\ [] = common\ A\ B$ $\rangle$

**by** *pat-completeness simp-all*

**termination**

**by** ($relation$ $\langle$ $measure\ (\lambda(\text{-}, \text{-}, C, D).\ \sum p \leftarrow C\ @\ D.\ size\ p)$ $\rangle$) *simp-all*

**theorem** *main*: $\langle$ $(\forall i.\ sc\ (map \cdot A\ @\ C)\ (map \cdot B\ @\ D)\ i) \longleftrightarrow mp\ A\ B\ C\ D$ $\rangle$

**by** (*induct rule*: *mp.induct*) (*simp-all*, *blast*, *meson*, *fast*)

# Let's make a computer check our work

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

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

# The course

- MSc level course with functional programming prerequisites
- Many students have never seen formal logic before
- Lectures, exercise sessions (with TAs), and assignments
- Exam is all programming and proving in Isabelle/HOL
- 43 students after the first few weeks

# The curriculum

| Weeks | Topics |
|---|---|
| 1 – 2 | Basic set theory, propositional logic, sequent calculus, automatic theorem proving |
| 3 – 4 | Syntax and semantics of first-order logic, natural deduction, the LCF approach |
| 5 – 6 | Isar, intuitionistic logic, foundational systems |
| 7 – 8 | Proof by contradiction, classical logic, higher-order logic, type theory |
| 9 – 10 | Proofs in sequent calculus |
| 11 – 13 | Metatheory, prover algorithms, program verification |

# Survey design



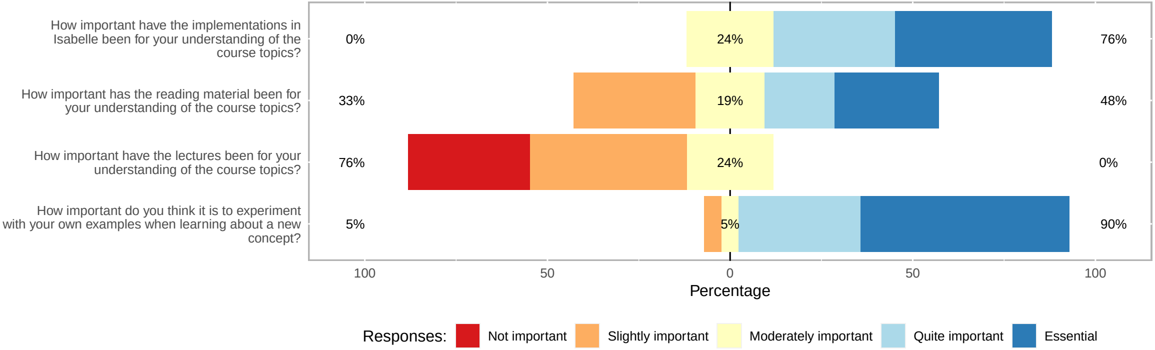
- Anonymous survey of our 43 students
- 21 answers (48.8% response rate, 15.5% margin of error)
- 6 hypotheses
- 12 questions

# Hypotheses

1. Concrete implementations in a programming language aid understanding of concepts in logic.
2. Students experiment with definitions to gain understanding.
3. Our formalizations make it clear to students how to implement the concepts in practice.
4. Our course makes students able to design and implement their own logical systems.
5. Prior experience with functional programming is useful for our course.
6. Our course helps students gain proficiency in functional programming.

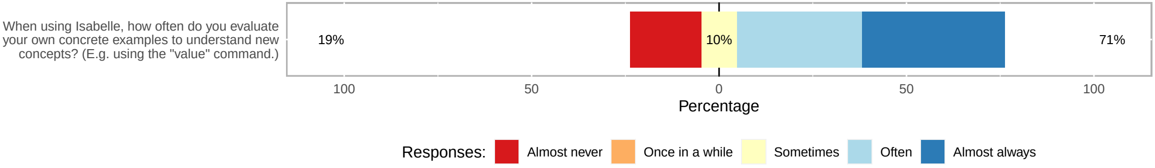
## What do our students think?

# Results




Plausible: Concrete implementations in a programming language aid understanding of concepts in logic

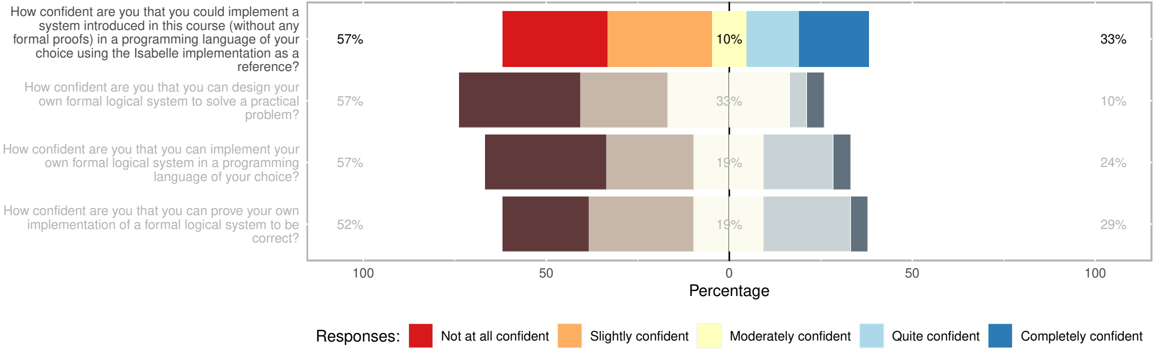
# Results




Confirmed: Students experiment with definitions to gain understanding

What do our students think?

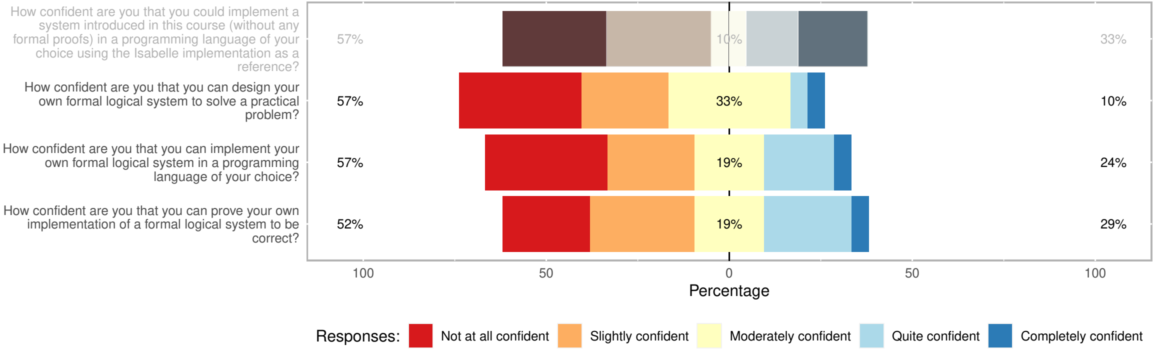
# Results




**Rejected:** Our formalizations make it clear to students how to implement the concepts in practice

## What do our students think?

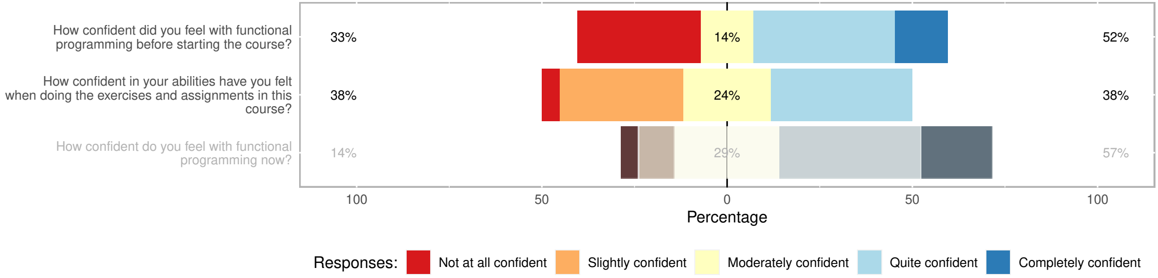
# Results




Rejected: Our course makes students able to design and implement their own logical systems

What do our students think?

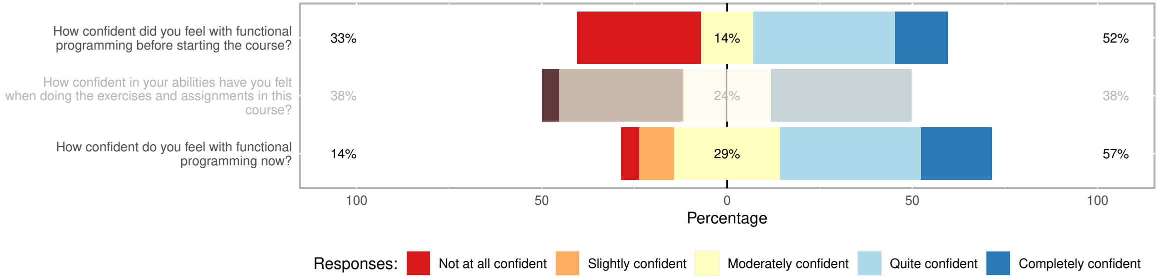
# Results




Confirmed: Prior experience with functional programming is useful for our course (small to moderate association)

What do our students think?

# Results




Confirmed: Our course helps students gain proficiency in functional programming (large positive effect)

# Interesting trends

**Warning: Post-hoc analysis!**

- It seems that students who think experimentation is more important do it less in Isabelle
- Students who were not confident functional programmers at the end were less confident that they could implement systems
- Students do not seem to get elevated past a basic understanding of functional programming
- Advanced concepts in functional programming do not seem to be needed

# Open questions

- Why do students who think experimentation is important seem to do it less? Do they do it on paper instead?
- Does functional programming experience play a significant role in understanding of how to implement concepts in practice?
- Does functional programming experience play a significant role in understanding of how to design and implement one's own logical systems?
- Does our course have a positive effect on functional programming skill for students who are already confident functional programmers?

# Ideas for improving our course

- Reduce time spent on lectures
- Add a project-based assignment
- Consider how to teach students how to design their own systems

# Future surveys

- Larger sample sizes
- Pursue open questions
- Add information on grades and demographics
- Reduce self-selection bias
- Comparisons to pen-and-paper courses

# Conclusion



- Our approach seems promising, but it needs more work
- Students find it hard to implement their own designs
- There are lots of opportunities for further research

More information:

Isabelle `https://isabelle.systems/`

Our paper `https://dx.doi.org/10.4204/EPTCS.363.5`

Related papers `https://people.compute.dtu.dk/fkjac`