DTU

DTU

Mobility Reading Group Seminar @ University of Oxford

Learning Proof Competence with Computer Assistance

Frederik Krogsdal Jacobsen

Technical University of Denmark

Overview

- Computer-assisted learning of proof competence:
  - professional
  - representational
  - communicational
  - methodological
- Covers a number of papers with my collaborators: Jørgen Villadsen, Asta Halkjær From, Nadine Karsten, Uwe Nestmann, Kim Jana Eiken
- . . . and some ongoing research and student projects

Topics that we’ve worked with

- Learning proofs in pure logic:
  - Sequent calculus
  - Natural deduction
  - Higher-order logic
  - Metatheory
  - *Ongoing: resolution*
- Learning proofs in computer science:
  - Proof assistants
  - Program verification
  - *Ongoing: lambda calculus*
  - *Ongoing: graph theory*

Is computer-assisted learning good or bad?

- Claimed benefits of computer-assisted learning:
  - Trains abstract thinking
  - Makes rules and structure clear
  - Instant feedback
  - Experiments with executable definitions

Is computer-assisted learning good or bad?

- Claimed benefits of computer-assisted learning:
  - Trains abstract thinking
  - Makes rules and structure clear
  - Instant feedback
  - Experiments with executable definitions
- Claimed drawbacks of computer-assisted learning:
  - Hard to learn syntax
  - Hard to understand errors
  - Difficult to transfer competences to pen-and-paper

Is computer-assisted learning good or bad?

- Claimed benefits of computer-assisted learning:
  - Trains abstract thinking
  - Makes rules and structure clear
  - Instant feedback
  - Experiments with executable definitions
- Claimed drawbacks of computer-assisted learning:
  - Hard to learn syntax
  - Hard to understand errors
  - Difficult to transfer competences to pen-and-paper
- Issues for instructors:
  - Overhead in introducing tools
  - Hard to design good exercises
  - Worrying about cheating
  - Need to develop tools for each subject

Our curriculum

Year

1	2	3	4	. . .
BSc			MSc	
Discrete Mathematics (mandatory)	Functional Programming (mandatory)	Logical Systems and Logic Programming	Automated Reasoning	
Introductory Programming (mandatory)	Computer Science Modelling (mandatory)		Program Verification	
Algorithms and Data Structures 1 (mandatory)	Algorithms and Data Structures 2		Formal Aspects of Software Engineering	
	Introduction to Artificial Intelligence		Artificial Intelligence and Multi-Agent Systems	
	Introduction to Machine Learning and Data Mining		Logical Theories for Uncertainty and Learning	

Trying to flatten the learning curve

- NaDeA
- SeCaV
- PureProof
- ResolutionOnline
- ProofBuddy

Natural Deduction Assistant

- Graphical interface for natural deduction proofs
- Classical first-order logic with functions
- Metatheory formalized in Isabelle
- Impossible to make syntax mistakes, and only applicable proof rules can be chosen
- Easy to use, but annoyingly slow after a while

Web interface

Natural Deduction Assistant

Load Code Help ProofJudge 36/36 Stop Undo ☺

1	Imp_I	[] ((A→B)→A)→A
2	Boole	[(A→B)→A] A
3	Imp_E	[A→ ⊥, (A→B)→A] ⊥
4	Assume	[A→ ⊥, (A→B)→A] A→ ⊥
5	Imp_E	[A→ ⊥, (A→B)→A] A
6	Assume	[A→ ⊥, (A→B)→A] (A→B)→A
7	Imp_I	[A→ ⊥, (A→B)→A] A→B
8	Boole	[A, A→ ⊥, (A→B)→A] B
9	Imp_E	[B→ ⊥, A, A→ ⊥, (A→B)→A] ⊥
10	Assume	[B→ ⊥, A, A→ ⊥, (A→B)→A] A→ ⊥
11	Assume	[B→ ⊥, A, A→ ⊥, (A→B)→A] A

Sequent Calculus Verifier

- A sequent calculus for the same logic
- Text-based — syntax mistakes are possible

Example

```
Dis p[a, b] (Neg p[a, b])

AlphaDis
  p[a, b]
  Neg p[a, b]
Basic
```

SeCaV

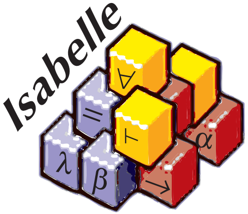
Web interface

Sequent Calculus Verifier | Help and Input Examples | 27:6 | Copy Output to Clipboard | SeCaV Unshortener 1.A

```
Imp (Con (Uni p[0]) q) (Dis r (Exi p[0]))

AlphaImp
  Neg (Con (Uni p[0]) q)
  Dis r (Exi p[0])
AlphaCon
  Neg (Uni p[0])
  Neg q
  Dis r (Exi p[0])
GammaUni
  Neg p[a]
  Neg q
  Dis r (Exi p[0])
Ext
  Dis r (Exi p[0])
  Neg p[a]
AlphaDis
  r
  Exi p[0]
  Neg p[a]
Ext
  Exi p[0]
  Neg p[a]
GammaExi
  p[a]
  Neg p[a]
Basic
```

```
proposition ‹((∀x. (p x)) ∧ q) ⟶ (r ∨ (∃x. (p x)))› by metis

text ‹
  Predicate numbers
    0 = p
    1 = q
    2 = r
  Function numbers
    0 = a
  ›

lemma ‹⊩
  [
    Imp (Con (Uni (Pre 0 [Var 0])) (Pre 1 [])) (Dis (Pre 2 []) (Exi (Pre 0 [Var 0])))
  ]
  ›
proof -
  from AlphaImp have ?thesis if ‹⊩
    [
      Neg (Con (Uni (Pre 0 [Var 0])) (Pre 1 [])),
      Dis (Pre 2 []) (Exi (Pre 0 [Var 0]))
    ]
    ›
    using that by simp
  with AlphaCon have ?thesis if ‹⊩
    [
      Neg (Uni (Pre 0 [Var 0])),
      Neg (Pre 1 []),
      Dis (Pre 2 []) (Exi (Pre 0 [Var 0]))
    ]
    ›
    using that by simp
  with GammaUni[where t=‹Fun 0 []›] have ?thesis if ‹⊩
    [
      Neg (Pre 0 [Fun 0 []]),
      Neg (Pre 1 []),
      Dis (Pre 2 []) (Exi (Pre 0 [Var 0]))
    ]
    ›
    using that by simp
  with Ext have ?thesis if ‹⊩
```

Isabelle



- Generic proof assistant
- Isabelle/HOL is the main logic today
- But also: Isabelle/ZF, Isabelle/Cube, . . .

Editors

- Isabelle/jEdit is the main interface
- Recently, Isabelle/VSCode has become usable

Intuitionistic propositional logic

- Formalization in Isabelle/Pure
- Why? No clutter, just the rules
- No automation
- Students are forced to write structured proofs and think about which rules to use

Intuitionistic higher-order logic

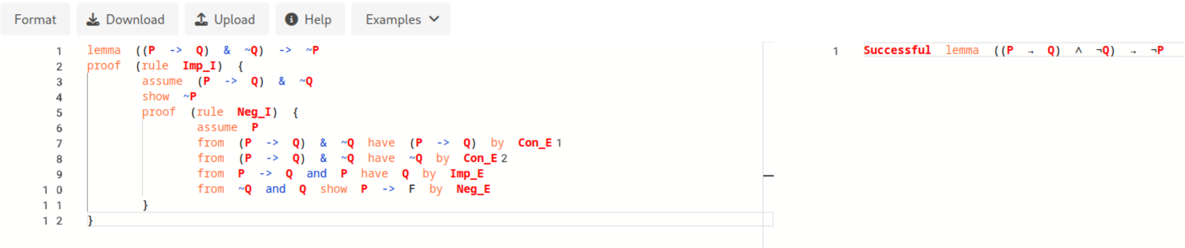
- Introduce higher-order logic
- More involved examples
- Learning how to work with quantifiers

Classical higher-order logic

- Essentially just Isabelle/HOL, but with no automation
- Learning how to approach proofs by contradiction through various possible rules
- Quite involved examples
- Builds a good understanding of what automation does under the hood

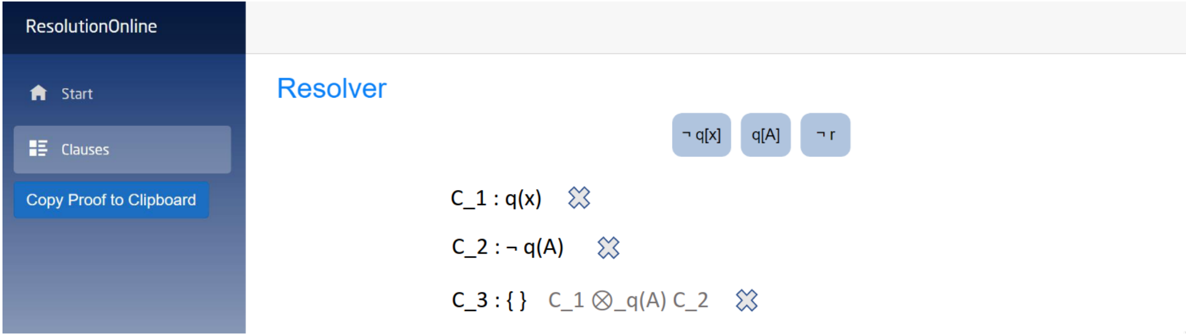
WIP: Web interface

- Makes clear what rules are available
- Allows simpler syntax
- Allows better error messages



WIP: ResolutionOnline

- Graphical proof assistant for the resolution rule and unification

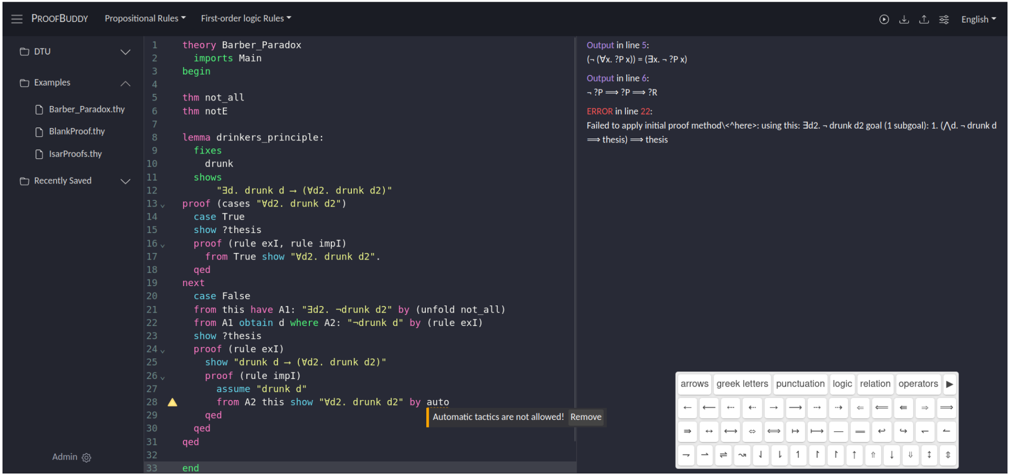


A web interface for Isabelle

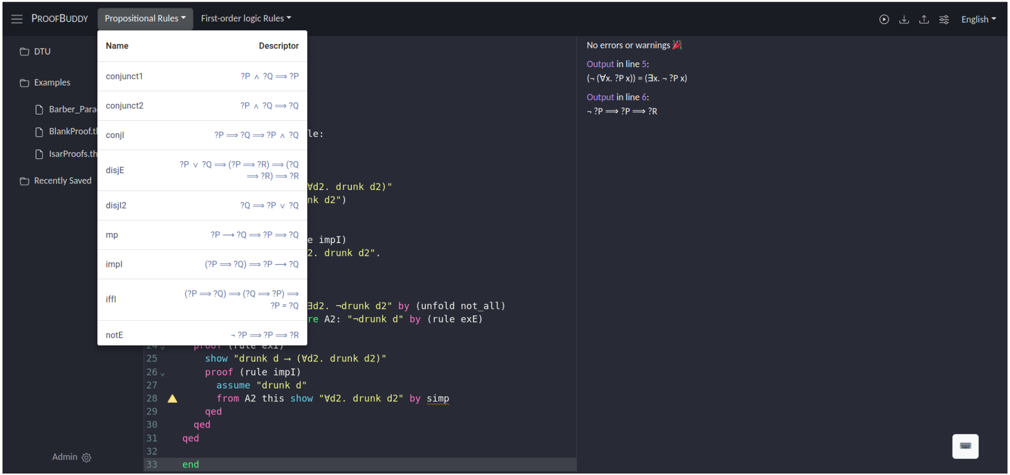
- Unifying interface for specialized proof assistants
- Restrict features for specific learning goals and exercises
- Introduce concepts one by one
- Immediate individual feedback for students
- Collect data about the student behavior
- Exercises tailored to students' learning needs

ProofBuddy

Example



Restricting features enables discovery



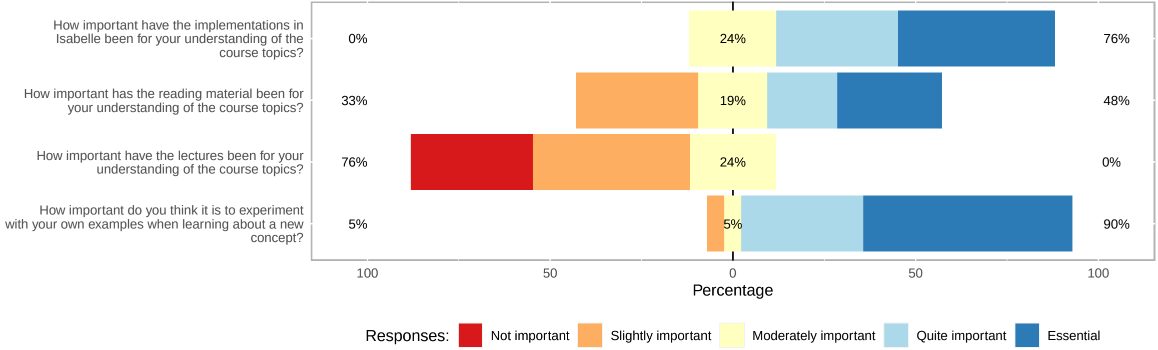
Enabling classroom use

- ProofBuddy for data collection
- Initial didactic research
- Approaches to exams
- Open problem: comparing to pen and paper
- Open problem: automated grading
- Open problem: guidelines for exercise design

Hypotheses we tried to test

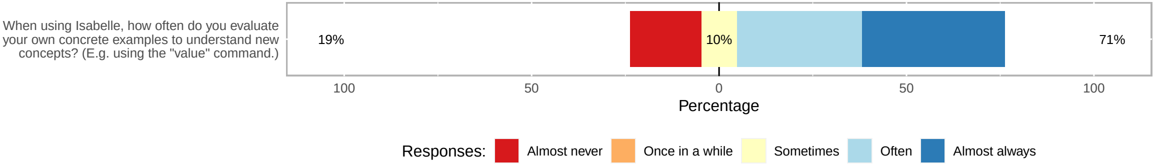
1. Concrete implementations in a programming language aid understanding of concepts in logic
2. Students experiment with definitions to gain understanding
3. Our formalizations make it clear to students how to implement the concepts in practice
4. Our course makes students able to design and implement 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

Results



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

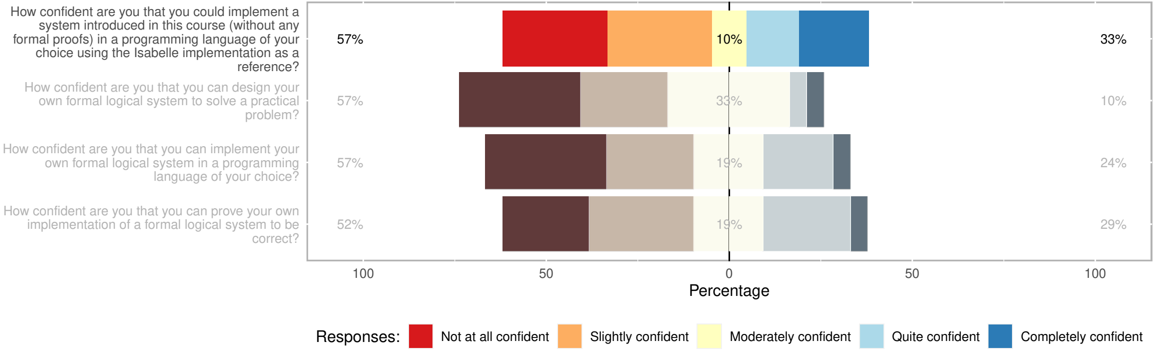
Results



Confirmed: Students experiment with definitions to gain understanding

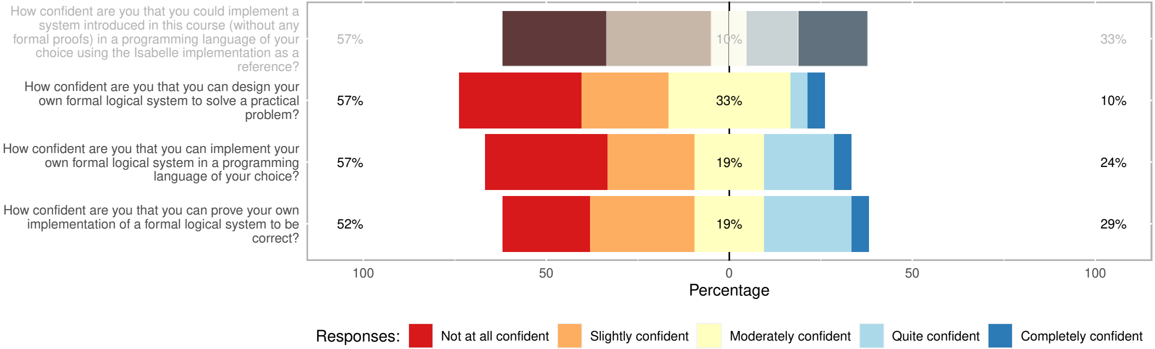
Initial didactic research

Results



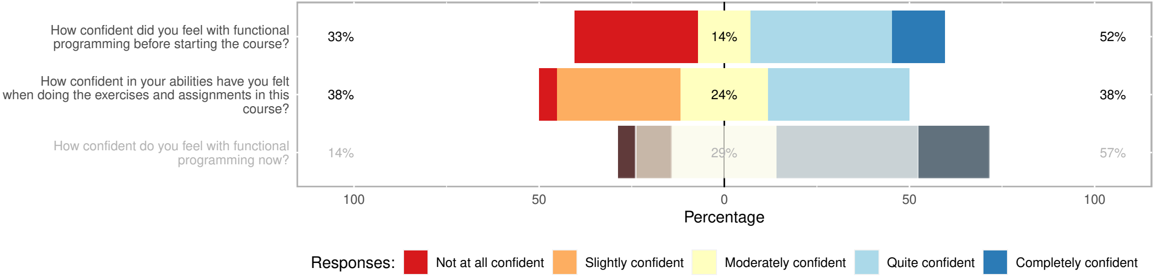
Rejected: Our formalizations make it clear to students how to implement the concepts in practice

Results



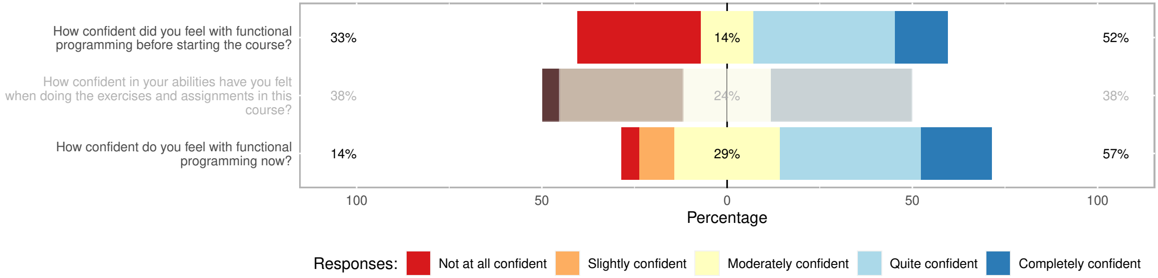
Rejected: Our course makes students able to design and implement their own logical systems

Results



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

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

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

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

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?

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?

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?

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?

Overview of our exam questions

1. Isabelle proofs without automation
2. Verification of functional programs in Isabelle/HOL
3. Natural deduction proofs
4. Sequent calculus proofs
5. General proofs in Isabelle/HOL with Isar

Isabelle proofs without automation

```
subsection ‹Question 1›

text ‹ Replace "by blast" with a proof in Pure_True (if possible omit names of Pure_True rules). ›

proposition ‹p ⟷ ¬ ¬ p›
  by blast
```

Isabelle proofs without automation

```
subsection ‹Question 1›

text ‹ Replace "by blast" with a proof in Pure_True (if possible omit names of Pure_True rules). ›

proposition ‹p ⟷ ¬ ¬ p›
proof
  assume p
  show ‹¬ ¬ p›
  proof
    assume ‹¬ p›
    from this and ‹p› show ⊥ ..
  qed
next
  assume ‹¬ ¬ p›
  show p
  proof (rule ccontr)
    assume ‹¬ p›
    with ‹¬ ¬ p› show ⊥ ..
  qed
qed
```

Verification of functional programs

```
subsection ‹Question 1›

text ‹ Using only the constructors 0 and Suc and no arithmetical operators, define a recursive function
triple :: ‹nat ⇒ nat› and prove ‹triple n = 3 * n›. ›
```

Verification of functional programs

```
subsection ‹Question 1›

text ‹ Using only the constructors 0 and Suc and no arithmetical operators, define a recursive function
triple :: ‹nat ⇒ nat› and prove ‹triple n = 3 * n›. ›

fun triple :: ‹nat ⇒ nat› where
  ‹triple 0 = 0› |
  ‹triple (Suc n) = Suc (Suc (Suc (triple n)))›

lemma ‹triple n = 3 * n›
  by (induct n) simp_all
```

General proofs in Isabelle

```
section ‹Problem 5 - 20%›

subsection ‹Question 1›

text ‹ Replace \<proof> with the "proof ... qed" lines in the following comment and correct the errors
such that the structured proof is a proper proof in Isabelle/HOL (do not alter the lemma text). ›

lemma Foobar:
  assumes ‹¬ (∀x. p x)›
  shows ‹∃x. ¬ p x›
  \<proof>

(*

proof (rule ccontr)
  assume ‹∃x. ¬ p x›
  have ‹∀x. p x›
  proof
    fix a
    show ‹p x›
    proof (rule ccontr)
      show ‹¬ p x›
      then have ‹∃x. ¬ p x› ..
      with ‹¬ (∃x. ¬ p x)› assume ⊥ ..
    qed
  qed
  with ‹¬ (∀x. p x)› show ⊤ ..
qed

*)
```

General proofs in Isabelle

```
lemma Foobar:
  assumes ‹¬ (∀x. p x)›
  shows ‹∃x. ¬ p x›
proof (rule ccontr)
  assume ‹¬ (∃x. ¬ p x)›
  have ‹∀x. p x›
  proof
    fix a
    show ‹p a›
    proof (rule ccontr)
      assume ‹¬ p a›
      then have ‹∃x. ¬ p x› ..
      with ‹¬ (∃x. ¬ p x)› show ⊥ ..
    qed
  qed
  with ‹¬ (∀x. p x)› show ⊥ ..
qed
```

Our experiences with the approach

- Difficult to come up with problems of the right complexity
- Relatively easy to grade submissions
- Students seem to have no problem understanding how to fill in answers and hand in
- How do we design problems with a good level of complexity?
  - Auxiliary tools can help mitigate complexity issues, but require a lot of work to create
  - Project-based exams may be easier to design, but take a long time to create and are difficult to scale up to many students

Future work

- Much more didactic research is needed to support efficacy hypotheses
- Unify approaches across subfields
- Establish best practices for classroom use
- Develop material for the middle of the learning curve
- Lots of opportunities for research!