

DTU



On Exams with the Isabelle Proof Assistant

Frederik Krogsdal Jacobsen Jørgen Villadsen

Technical University of Denmark

Introduction

- Our graduate course in automated reasoning needs an evaluation of student outcomes
- We use Isabelle throughout the course
- The exam is also in Isabelle
- How can we test that students know what is going on when Isabelle has so much automation?
- How can we make the grading as easy as possible?

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/Pure

- Students are asked to prove simple logical formulas
- Higher-order logic
- No automation
- Proof rules must be applied manually
- Isabelle can infer which rule to use in many, but not all, cases

Example

subsection <Question 1>

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

proposition < $p \longleftrightarrow \neg \neg p$ >
by blast

Example

subsection <Question 1>

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

proposition < $p \longleftrightarrow \neg \neg p$ >

proof

assume p

show < $\neg \neg p$ >

proof

assume < $\neg p$ >

from this and <p> show $\perp$..

qed

next

assume < $\neg \neg p$ >

show p

proof (rule ccontr)

assume < $\neg p$ >

with < $\neg \neg p$ > show $\perp$..

qed

qed

Programming and proving

- Simple functional programs
- Simple proofs by induction
- Automation is allowed
- Finding the right level of complexity is hard

Example

subsection <Question 1>

text < Using only the constructors 0 and Suc and no arithmetical operators, define a recursive function `triple` :: `<nat  $\Rightarrow$  nat>` and prove `<triple n = 3 * n>`. >

Example

subsection <Question 1>

text < Using only the constructors 0 and Suc and no arithmetical operators, define a recursive function `triple :: nat  $\Rightarrow$  nat` and prove `<triple n = 3 * n>`. >

```
fun triple :: nat  $\Rightarrow$  nat where
  <triple 0 = 0> |
  <triple (Suc n) = Suc (Suc (Suc (triple n)))>
```

```
lemma <triple n = 3 * n>
  by (induct n) simp_all
```

Natural Deduction Assistant

- Graphical web application for natural deduction proofs
- Classical first-order logic
- Impossible to apply proof rules wrong, but students still have to choose which ones to use
- Students export proofs to Isabelle

Example

subsection <Question 1>

text < Use NaDeA to prove the following formula and include the load lines as a (*...*) comment. >

proposition < $((A \rightarrow B) \rightarrow A) \rightarrow A$ > **by** metis

Example

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
```

Example

```

proposition " $((A \rightarrow B) \rightarrow A) \rightarrow A$ "
by blast

theorem "semantics e f g (Imp (Imp (Imp (Pre 0 []) (Pre 1 [])) (Pre 0 [])) (Pre 0 []))"
proof (rule soundness)
  show "OK (Imp (Imp (Imp (Pre 0 []) (Pre 1 [])) (Pre 0 [])) (Pre 0 [])) []"
  proof (rule Imp_I)
    show "OK (Pre 0 []) [Imp (Imp (Pre 0 []) (Pre 1 [])) (Pre 0 [])]"
    proof (rule Boole)
      show "OK Falsity [Imp (Pre 0 []) Falsity, Imp (Imp (Pre 0 []) (Pre 1 [])) (Pre 0 [])]"
      proof (rule Imp_E)
        show "OK (Imp (Pre 0 []) Falsity) [Imp (Pre 0 []) Falsity, Imp (Imp (Pre 0 []) (Pre 1 [])) (Pre 0 [])]"
        by (rule Assume) simp
      next
        show "OK (Pre 0 []) [Imp (Pre 0 []) Falsity, Imp (Imp (Pre 0 []) (Pre 1 [])) (Pre 0 [])]"
        proof (rule Imp_E)
          show "OK (Imp (Imp (Pre 0 []) (Pre 1 [])) (Pre 0 [])) [Imp (Pre 0 []) Falsity, Imp (Imp (Pre 0 []) (Pre 1 [])) (Pre 0 [])]"
          by (rule Assume) simp
        next
          show "OK (Imp (Pre 0 []) (Pre 1 [])) [Imp (Pre 0 []) Falsity, Imp (Imp (Pre 0 []) (Pre 1 [])) (Pre 0 [])]"
          proof (rule Imp_I)
            show "OK (Pre 1 []) [Pre 0 [], Imp (Pre 0 []) Falsity, Imp (Imp (Pre 0 []) (Pre 1 [])) (Pre 0 [])]"
            proof (rule Boole)
              show "OK Falsity [Imp (Pre 1 []) Falsity, Pre 0 [], Imp (Pre 0 []) Falsity, Imp (Imp (Pre 0 []) (Pre 1 [])) (Pre 0 [])]"
              proof (rule Imp_E)
                show "OK (Imp (Pre 0 []) Falsity) [Imp (Pre 1 []) Falsity, Pre 0 [], Imp (Pre 0 []) Falsity, Imp (Imp (Pre 0 []) (Pre 1 [])) (Pre 0 [])]"
                by (rule Assume) simp
              next
                show "OK (Pre 0 []) [Imp (Pre 1 []) Falsity, Pre 0 [], Imp (Pre 0 []) Falsity, Imp (Imp (Pre 0 []) (Pre 1 [])) (Pre 0 [])]"
                by (rule Assume) simp
              qed
            qed
          qed
        qed
      qed
    qed
  qed

```

Sequent Calculus Verifier

- Text-based web application for sequent calculus proofs
- Classical first-order logic
- Users must manually specify the result of applying each proof rule
- Students export proofs to Isabelle

Example

subsection <Question 2>

text <Construct a SeCaV proof for the following formula and add (* ... *) with the shortened proof>

proposition < $(\forall x. p\ x) \wedge q \longrightarrow r \vee (\exists x. p\ x)$ > **by** metis

Example

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

```

1  Imp (Con (Uni p[0]) q) (Dis r (Exi p[0]))
2
3  AlphaImp
4    Neg (Con (Uni p[0]) q)
5    Dis r (Exi p[0])
6  AlphaCon
7    Neg (Uni p[0])
8    Neg q
9    Dis r (Exi p[0])
10 GammaUni
11   Neg p[a]
12   Neg q
13   Dis r (Exi p[0])
14 Ext
15   Dis r (Exi p[0])
16   Neg p[a]
17 AlphaDis
18   r
19   Exi p[0]
20   Neg p[a]
21 Ext
22   Exi p[0]
23   Neg p[a]
24 GammaExi
25   p[a]
26   Neg p[a]
27 Basic

```

```

proposition <(( $\forall x. (p\ x) \wedge q \rightarrow (r \vee (\exists x. (p\ x)))$ )> 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]))
  ]
  >

```

More problems with complexity

- The full power of Isabelle is too much for an exam situation
- Extremely difficult to find theorems which are too difficult for automation but possible to solve within 15–30 minutes
- Solution: make students “fix” proofs

Example

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 < $\neg (\forall x. p\ x)$ >
  shows < $\exists x. \neg p\ x$ >
  \<proof>

  (*

proof (rule ccontr)
  assume < $\exists x. \neg p\ x$ >
  have < $\forall x. p\ x$ >
  proof
    fix a
    show < $p\ a$ >
    proof (rule ccontr)
      show < $\neg p\ a$ >
      then have < $\exists x. \neg p\ x$ > ..
      with < $\neg (\exists x. \neg p\ x)$ > assume  $\bot$  ..
    qed
  qed
  with < $\neg (\forall x. p\ x)$ > show  $\top$  ..
qed

*)
```

Example

```

lemma Foobar:
  assumes <math>\neg (\forall x. p\ x)>
  shows <math>\exists x. \neg p\ x>
proof (rule ccontr)
  assume <math>\exists x. \neg p\ x>
  have <math>\forall x. p\ x>
  proof
    fix a
    show <math>p\ a>
    proof (rule ccontr)
      show <math>\neg p\ a>
      then have <math>\exists x. \neg p\ x> ..
      with <math>\neg (\exists x. \neg p\ x)> assume <math>\bot> ..
    qed
  qed
  with <math>\neg (\forall x. p\ x)> show <math>\top> ..
qed

```

Example

```

lemma Foobar:
  assumes  $\neg (\forall x. p\ x)$ 
  shows  $\langle \exists x. \neg p\ x \rangle$ 
proof (rule ccontr)
  assume  $\langle \exists x. \neg p\ x \rangle$ 
  have  $\langle \forall x. p\ x \rangle$ 
  proof
    fix a
    show  $\langle p\ a \rangle$ 
    proof (rule ccontr)
      show  $\langle \neg p\ a \rangle$ 
      then have  $\langle \exists x. \neg p\ x \rangle$  ..
      with  $\langle \neg (\exists x. \neg p\ x) \rangle$  assume  $\perp$  ..
    qed
  qed
  with  $\langle \neg (\forall x. p\ x) \rangle$  show  $\top$  ..
qed

```

Example

```

lemma Foobar:
  assumes  $\neg (\forall x. p\ x)$ 
  shows  $\exists x. \neg p\ x$ 
proof (rule ccontr)
  assume  $\exists x. \neg p\ x$ 
  have  $\forall x. p\ x$ 
  proof
    fix a
    show  $p\ a$ 
    proof (rule ccontr)
      assume  $\neg p\ a$ 
      then have  $\exists x. \neg p\ x$  ..
      with  $\neg (\exists x. \neg p\ x)$  show  $\bot$  ..
    qed
  qed
  with  $\neg (\forall x. p\ x)$  show  $\top$  ..
qed

```

Example

```

lemma Foobar:
  assumes <math>\neg (\forall x. p\ x)>
  shows <math>\exists x. \neg p\ x>
proof (rule ccontr)
  assume <math>\neg (\exists x. \neg p\ x)>
  have <math>\forall x. p\ x>
  proof
    fix a
    show <math>p\ a>
    proof (rule ccontr)
      assume <math>\neg p\ a>
      then have <math>\exists x. \neg p\ x> ..
      with <math>\neg (\exists x. \neg p\ x)> show \bot ..
    qed
  qed
  with <math>\neg (\forall x. p\ x)> show \top ..
qed

```

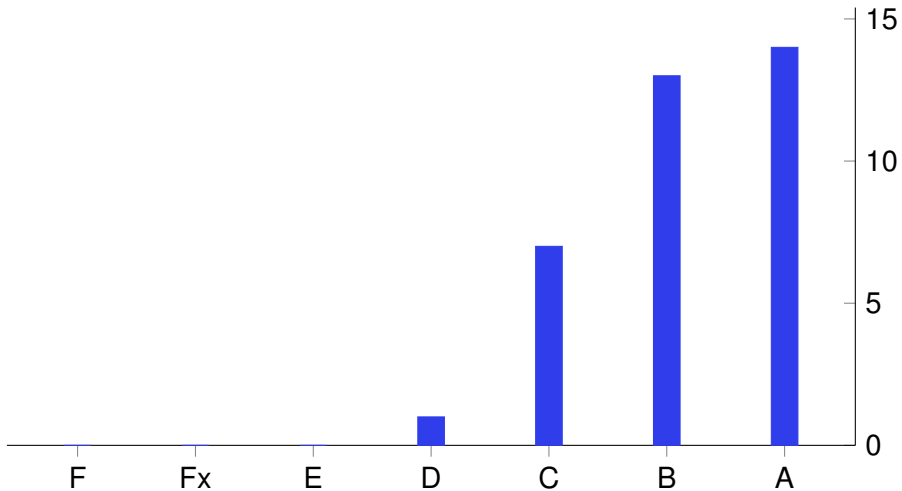
Example

```
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

Conclusion Grades



Automated grading?

- Difficult because students hand in with many syntax errors
- Some work on this has been done for Coq
- In an exam situation it seems unfeasible

Future work

- 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
- Can we make grading more automated?
- Are there other things we should test?