Students' Proof Assistant (SPA)

Anders Schlichtkrull, Jørgen Villadsen, and Andreas Halkjær From

DTU Compute, AlgoLoG, Technical University of Denmark, 2800 Kongens Lyngby, Denmark
andschl@dtu.dk jovi@dtu.dk s144442@student.dtu.dk

Abstract

The Students' Proof Assistant (SPA) aims to both teach how to use a proof assistant like Isabelle but also to teach how reliable proof assistants are built. Technically it is a miniature proof assistant inside the Isabelle proof assistant. In addition we conjecture that a good way to teach structured proving is with a concrete prover where the connection between semantics, proof system, and prover is clear. In fact, the proofs in Lamport's TLAPS proof assistant have a very similar structure to those in the declarative prover SPA. To illustrate this we compare a proof of Pelletier's problem 43 in TLAPS, Isabelle/Isar and SPA.

1 Introduction

The Students' Proof Assistant (SPA) aims to both teach how to use a proof assistant like Isabelle [5] and to teach how reliable proof assistants are built. SPA is a miniature proof assistant running inside Isabelle (https://github.com/logic-tools/spa). It is based on work by Harrison [1] and the entire development runs in Isabelle's ML environment as an interactive application. We have just finished a prototype of SPA based on our publication [2]. In that publication we formalized the kernel of an axiomatic system in Isabelle and exported it to SML-code. This code served as the kernel of a translation from OCaml to SML of a proof assistant from Harrison's chapter on the topic [1]. We chose to let the definitions in the formalization and translation follow Harrison's book very strictly. Harrison's code works well in a book on the broad topic of practical logic and automated reasoning but is in places perhaps overly general for teaching material only on the topic of proof assistants. For instance Harrison's datatype for formulas is parameterized with the type for atoms: for first-order logic the parameter is instantiated to a type for first-order predicates, and for propositional logic the parameter is he instantiated with the strings and ignores the constructors for universal and existential quantification. Since we will only consider a proof assistant for first-order logic we do not need to parameterize on a type of atoms – we can put first-order predicates directly into the definition. We see more opportunities for improving the development for our purpose.

Our previous publication discussed three ways to represent the type of theorems:

1. A datatype wrapping the type of formulas in a constructor.
2. A type exactly characterizing the provable formulas (theorems).
3. A type exactly characterizing the valid formulas.

In the first case the theorems were then further characterized by a predicate. The latter two cases can be made using Isabelle's **typedef** command as well as functionality for lifting functions that work on formulas to work on the new type. We previously argued for 1 because of its simplicity, but for SPA we have had a change of hearts and go with solution 2. The reason is that while lifting and **typedef** are arguably advanced concepts they make it easier to inspect the verification as we argued [2] and furthermore the idea behind **typedef** is after all quite simple – it can be seen as elevating a set to a type.

We conjecture that a good way to teach structured proving is with a concrete prover where the connection between semantics, proof system, and prover is clear. Even for paper proofs Lamport recommends writing in a structured style [3, 4].

2 Pelletier's Problem 43

In fact, the proofs in Lamport's TLAPS proof assistant have a very similar structure to those in the declarative prover SPA. To illustrate this we compare a proof of Pelletier's problem 43 [2] in TLAPS (Figure 1), Isabelle/Isar (Figure 2) and SPA (Figure 3).

Pelletier's problem 43 is:

$$(\forall x\ y.\ Q(x,y) \longleftrightarrow (\forall z.\ P(z,x) \longleftrightarrow P(z,y))) \longrightarrow (\forall x\ y.\ Q(x,y) \longleftrightarrow Q(y,x))$$

The idea is that based on a binary relation P the relation Q is defined to consist of any pair (x, y) that has the property

$$\forall z.\ P(z,x) \longleftrightarrow P(z,y)$$

In other words, any x and y are related by Q if they relate equivalently to any z with regards to P. The problem states that Q is symmetric.

We explain the SPA proof informally, using regular notation:

Proof:
We are trying to prove an implication, so we start off by assuming the antecedent, calling it "A" so we can refer to it later in the proof. Since the statement is universally quantified, we arbitrarily fix an x and a y to use in the proof. We then show the biimplication by showing the conjunction of both directions and using the command *at once* which does pure first-order reasoning and can easily handle small steps such as this one.

Consider first the direction $Q(x,y) \longrightarrow Q(y,x)$. This is an implication so we start again by assuming the antecedent. We do not have to name it this time, as we only use it in the proof of the next statement where it can be referenced using *so*. This assumption matches the left-hand side of "A" which we appeal to using *by* and are thus allowed to conclude the right-hand side: That x and y are equivalent with regards to P. The next line swaps the order of the biimplication *at once*, which allows us to then appeal to "A" again, this time in the opposite direction, and conclude the goal $Q(x,y)$. Note that in the final step, the quantified x in "A" is instantiated with our fixed y and y with x.

The proof of the direction $Q(y,x) \longrightarrow Q(x,y)$ is exactly symmetric to the one above.

□

We mark the end of a (sub)proof by *qed* in TLAPS, Isabelle/Isar and SPA.

3 Discussion

The similarities between the proofs in TLAPS, Isabelle/Isar and SPA are evident. This corroborates that the skills the students learn in our miniature proof assistant are transferable to full-fledged proof assistants.

For Pelletier's problem 43 the automation in a proof assistant like Isabelle can actually prove the whole formula without the user supplying a proof. This raises the question: Why bother teaching students how to prove such lemmas in more detail than a single call to the automation? Our answer is that when working in a proof assistant for higher-order logic one often works by breaking down the logical structure of the problem. This often amounts to first-order reasoning.

Hereafter one takes a look at the involved mathematical objects and tries to come up with a needed lemma involving these objects. Doing this of course requires knowledge of breaking the logical structure down to begin with.

We have not found related work where a miniature proof assistant is developed inside another proof assistant. Overall our approach is related to the IsaFoL (https://bitbucket.org/isafol) project which unites researchers in formalizing logic in Isabelle. Among the formalizations in the project are SAT-solving, first-order resolution, a paraconsistent logic, natural deduction, sequent calculi and more.

4 Conclusion

SPA is an advanced e-learning tool for teaching proof assistants for students in computer science as well as in mathematics and complements our other e-learning tool, NaDeA (A Natural Deduction Assistant with a Formalization in Isabelle), which is available online and has been used by hundreds of BSc and MSc students at DTU in regular courses [6].

We have a number of ideas for improving SPA such as tighter integration between the miniature proof assistant and Isabelle, implementing a resolution prover and formalizing completeness of SPA's kernel. Better integration with Isabelle could help with inputting first-order formulas and giving better error-reporting when students make mistakes in formal proofs in SPA.

Amongst Pelletier's problems another interesting one is problem 34 which is also known as Andrews's Challenge [1]. The truth of the formula is not obvious at first glance since it relies on the fact that biimplication is both commutative and, perhaps surprisingly, associative. Proving this formula and similar ones in SPA is future work too.

Acknowledgements

We thank Alexander Birch Jensen for collaboration on the first formalization and we thank Martin Elsman, Lars Hupel, John Bruntse Larsen and Makarius Wenzel for fruitful discussions. We are also grateful to John Harrison for encouragement – John once remarked to us that "it is rather reassuring that I don't have to worry any longer about bugs in that part of the code" given our formalization and code generation in Isabelle.

References

[1] J. Harrison. *Handbook of Practical Logic and Automated Reasoning*. Cambridge University Press, 2009.

[2] A. B. Jensen, J. B. Larsen, A. Schlichtkrull, and J. Villadsen. Programming and verifying a declarative first-order prover in Isabelle/HOL. *AI Communications*, 31(3):281–299, 2018.

[3] L. Lamport. How to write a proof. *Global Analysis in Modern Mathematics*, pages 311–321, 1993. Also published in *American Mathematical Monthly*, 102(7):600-608, August-September 1995.

[4] L. Lamport. How to write a 21st century proof. *Journal of fixed point theory and applications*, 11(1):43–63, 2012.

[5] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[6] J. Villadsen, A. H. From, and A. Schlichtkrull. Natural deduction and the Isabelle proof assistant. In P. Quaresma and W. Neuper, editors, *Proceedings 6th International Workshop on Theorem proving components for Educational Software (ThEdu)*, volume 267 of *EPTCS*, pages 140–155, 2017.

```
MODULE p43

THEOREM p43 ≜
  ∀ S :
    ∀ P ∈ [S → [S → BOOLEAN ]] :
      ∀ Q ∈ [S → [S → BOOLEAN ]] :
        (∀ x ∈ S : ∀ y ∈ S : (Q[x][y] ≡ (∀ z ∈ S : P[z][x] ≡ P[z][y]))) ⇒
         (∀ x ∈ S : ∀ y ∈ S : (Q[x][y] ≡ Q[y][x]))
  ⟨1⟩1. SUFFICES ASSUME NEW S,
                         NEW P ∈ [S → [S → BOOLEAN ]],
                         NEW Q ∈ [S → [S → BOOLEAN ]],
                         ∀ x ∈ S : ∀ y ∈ S : (Q[x][y] ≡ (∀ z ∈ S : P[z][x] ≡ P[z][y]))
                 PROVE  ∀ x ∈ S : ∀ y ∈ S : (Q[x][y] ≡ Q[x][y])
    OBVIOUS
  ⟨1⟩2. SUFFICES ASSUME NEW x ∈ S,
                         NEW y ∈ S
                 PROVE Q[x][y] ≡ Q[y][x]
    OBVIOUS
  ⟨1⟩3. Q[x][y] ⇒ Q[y][x]
    ⟨2⟩1. SUFFICES ASSUME Q[x][y]
                   PROVE Q[y][x]
      OBVIOUS
    ⟨2⟩2. ∀ z ∈ S : P[z][x] ≡ P[z][y]
      BY ⟨2⟩1, ⟨1⟩1
    ⟨2⟩3. ∀ z ∈ S : P[z][y] ≡ P[z][x]
      BY ⟨2⟩2
    ⟨2⟩4. Q[y][x]
      BY ⟨1⟩1, ⟨2⟩3
    ⟨2⟩5. QED
      BY ⟨2⟩4
  ⟨1⟩4. Q[y][x] ⇒ Q[x][y]
    ⟨2⟩1. SUFFICES ASSUME Q[y][x]
                   PROVE Q[x][y]
      OBVIOUS
    ⟨2⟩2. ∀ z ∈ S : P[z][y] ≡ P[z][x]
      BY ⟨1⟩1, ⟨2⟩1
    ⟨2⟩3. ∀ z ∈ S : P[z][x] ≡ P[z][y]
      BY ⟨2⟩2
    ⟨2⟩4. Q[x][y]
      BY ⟨1⟩1, ⟨2⟩3
    ⟨2⟩5. QED
      BY ⟨2⟩4
  ⟨1⟩6. QED
    BY ⟨1⟩3, ⟨1⟩4
```

Figure 1: Proof of Pelletier's Problem 43 in the TLA+ Proof System (TLAPS)

```
theory Pelletier_43 imports Main begin

theorem ‹(∀x y. Q(x,y) ⟷ (∀z. P(z,x) ⟷ P(z,y))) ⟶ (∀x y. Q(x,y) ⟷ Q(y,x))›
proof
  assume A: ‹∀x y. Q(x,y) ⟷ (∀z. P(z,x) ⟷ P(z,y))›
  show ‹∀x y. Q(x,y) ⟷ Q(y,x)›
  proof (rule, rule)
    fix x y
    show ‹Q(x,y) ⟷ Q(y,x)›
    proof -
      have ‹(Q(x,y) ⟶ Q(y,x)) ∧ (Q(y,x) ⟶ Q(x,y))›
      proof
        show ‹Q(x,y) ⟶ Q(y,x)›
        proof
          assume ‹Q(x,y)›
          then have ‹∀z. P(z,x) ⟷ P(z,y)› using A by iprover
          then have ‹∀z. P(z,y) ⟷ P(z,x)› by iprover
          then show ‹Q(y,x)› using A by iprover
        qed
      next
        show ‹Q(y,x) ⟶ Q(x,y)›
        proof
          assume ‹Q(y,x)›
          then have ‹∀z. P(z,y) ⟷ P(z,x)› using A by iprover
          then have ‹∀z. P(z,x) ⟷ P(z,y)› by iprover
          then show ‹Q(x,y)› using A by iprover
        qed
      qed
      then show ‹Q(x,y) ⟷ Q(y,x)› by iprover
    qed
  qed
qed

end
```

Figure 2: Proof of Pelletier's Problem 43 in Isabelle/Isar

```
prove
  (<!"(forall x y. Q(x,y) <=> forall z. P(z,x) <=> P(z,y)) ==> forall x y. Q(x,y) <=> Q(y,x)"!>)
  [
    assume [("A", <!"forall x y. Q(x,y) <=> forall z. P(z,x) <=> P(z,y)"!>)],
    conclude (<!"forall x y. Q(x,y) <=> Q(y,x)"!>) proof
    [
      fix "x", fix "y",
      conclude (<!"Q(x,y) <=> Q(y,x)"!>) proof
      [
        have (<!"(Q(x,y) ==> Q(y,x)) /\\ (Q(y,x) ==> Q(x,y))"!>) proof
        [
          conclude (<!"Q(x,y) ==> Q(y,x)"!>) proof
          [
            assume [("", <!"Q(x,y)"!>)],
            so have (<!"forall z. P(z,x) <=> P(z,y)"!>) by ["A"],
            so have (<!"forall z. P(z,y) <=> P(z,x)"!>) at once,
            so conclude (<!"Q(y,x)"!>) by ["A"],
            qed
          ],
          conclude (<!"Q(y,x) ==> Q(x,y)"!>) proof
          [
            assume [("", <!"Q(y,x)"!>)],
            so have (<!"forall z. P(z,y) <=> P(z,x)"!>) by ["A"],
            so have (<!"forall z. P(z,x) <=> P(z,y)"!>) at once,
            so conclude (<!"Q(x,y)"!>) by ["A"],
            qed
          ],
          qed
        ],
        so our thesis at once,
        qed
      ],
      qed
    ],
    qed
  ]

*}
```

Figure 3: Proof of Pelletier's Problem 43 in Students' Proof Assistant (SPA)