CPP 2015 notification for paper 17

1 message

CPP 2015 <cpp2015@easychair.org> Thu, Nov 27, 2014 at 9:24 AM
To: Ilya Sergey <ilya.sergey@imdea.org>

Dear Ilya,

We are sorry to inform you that your paper "Introducing Functional Programmers to Interactive Theorem Proving and Program Verification (Teaching Experience Report)" was not selected by the program committee to appear at the 2015 conference on Certified Proofs and Programs.

Comments from the reviewers are attached to this message. We hope that these comments will be helpful to you in continuing your research.

Thank you again for your interest in CPP 2015.

Sincerely,

- Xavier Leroy and Alwen Tiu
CPP 2015 program chairs

---------- REVIEW 1 ----------
PAPER: 17
TITLE: Introducing Functional Programmers to Interactive Theorem Proving and Program Verification (Teaching Experience Report)
AUTHORS: Ilya Sergey and Aleksandar Nanevski

OVERALL EVALUATION: 0 (borderline paper)

---------- REVIEW ----------
The article presents a short course that appears to teach formalised reasoning in Separation Logic and Hoare Type Theory (HTT) in Coq. The course makes use of the Ssreflect library and its core concept, boolean reflection, and exploits its minimal tactic language. Further discussed topics are Coq's canonical structures and dependent records together with mixin-composition. The course in general appears to beeline for HTT and cover little on the sides.

The presented course was held once at a five-day summer school with a group of five students.

Observations:

The discussion of the Ssreflect tactic language and its benefits misses a crucial point. The situation is actually quite similar to purely functional programming. In both scenarios we are given a very small collection of powerful primitives with clearly defined and easy to grasp semantics. In both cases this allows the user to write potentially very concise and elegant proof scripts/programs. The (unmentioned) issue, however, is that going from understanding the primitives to understanding the meaning of a sufficiently complex compound term presents an amazingly steep learning curve, even for good students. From a didactic point of view one has to very carefully control the increase in complexity and allow for sufficient time and practice before the concepts really sink in. Otherwise the results are usually as reported in Section 4.2 which appear to somewhat surprise the authors.

When Canonical Structures are discussed, one of the main motivations for the mixin-composition is elided, namely that, as far as I know, type-checking the coercion of a record type into its carrier is exponential in the depth of the record nesting. Hence, using mixins to keep the carrier at the top mainly serves to improve efficiency. For details see reference no. [7].

The discussion of automation in the context of proof assistants could be improved. To me, it is not clear from the text why "ad hoc" automation is supposedly bad while the use of the [heval] automation tactic from the author's library is perfectly fine. My best guess is that it relates to generic vs. purpose-specific, or it might be about the failure-behaviour after changing definitions (i.e. if, where and how such a tactic fails after modifications).

There appears to be a misunderstanding of Henz and Hobor's objectives in their teaching report. Their focus is to introduce students to logic (the term formalisation is used in a pre-mechanisation sense), mainly at an undergraduate level, with the help of a proof assistant, as opposed to the present article which appears to primarily focus on mechanisation as the object of study. They also do not rely on a prior knowledge in modal logic, in contrast to what is claimed
here. Moreover, prior knowledge of abstract algebra is tacitly assumed here. As a consequence the whole comparison of the two approaches appears somewhat beside the point.

I would have liked to see the following points addressed more
clearly:
1. Who exactly is the target audience?
  a) Undergraduate/Bachelor level
  b) Graduate/Master level
  c) Beginning Research/PhD students
[My best guess is c)]

2. What exactly are the initial requirements?
  a) Functional programming and software engineering are explicitly mentioned.
  b) Mathematics/abstract algebra appears in Section 3.5 as an implicit prerequisite.
  c) While never mentioned, a basic understanding of (paper-)formal logical reasoning and
    mathematics appear to be quite essential.

3. Over what time-span should the course be held; or respectively, how many lectures & exercise sessions does it encompass?
[My best guess is that at least a trimester is required to communicate all the essentials to group of students that is more diverse than a small summer school class.]

4. What exactly is the main teaching objective of the course?
Claim: Interactive Theorem Proving and Program Verification
My Impression: basic mechanisation, mechanised HTT with the author's library, and mechanisation techniques for large scale proof developments.

Evaluation:

Personally I have to admit that I find it difficult to assess if and in what way the research community would benefit from the publication of the article as it stands. I believe, that there are two main possible objectives that a teaching article may shoot for. Firstly, it may simply serve to inform the wider community about the existence of the course concept and point to freely available online material for those interested in using the design for their own teaching requirements. This appears to be suitable for brand new concepts. Alternatively it may report on a course that has been tested over a number of iterations, possibly with variations, and thoroughly evaluated, e.g. by using a University's student feedback mechanism (free form text and/or numerical).

The referenced article from Henz and Hobor clearly is of the second kind, while the one considered here somehow appears to attempt both and achieve neither to full satisfaction. With respect to the former objective it appears slightly too verbose when outlining the syllabus of the course and a bit too unspecific with respect to key facts (audience, teaching goal, requirements). With respect to the second there is simply too little evidence (5 students, 5 days, pre-selection bias) to attribute any statistical significance to the viability of the course design.

I do believe that the course itself is well-designed to enable students to confidently work with HTT and Separation Logic in Coq and Ssreflect, but the presentation thereof may require a slight
adjustment.

Stylistic Remarks and Suggestions:

- The authors should be careful to avoid using Ssreflect-specific syntax without explaining it,

e.g. [Const of A & B : t] as shorthand for [Const : A -> B -> t].

- There are some language issues that a spellchecker won't catch, i.e. some missing/superfluous articles, singular vs plural, inconsistent capitalisation, prepositions and typos that result in wrong, but correctly spelled, words (e.g. p7r: "coercions a transitive" should be "coercions are transitive"). Also the usage of"specs" (p8r) might be a bit too colloquial.

- on p4l : "... are proofs of P->Q, Q->R, and P, respectively, ..."; the last P should be capital, since lower case p is used as the proof of P.

- p5l : "case analysing on a proof of ltngtP" should probably be either "case analysing the proof ltngtP"; or "case analysing on a proof of nat_rels ...", due to the fact that [ltngtP m n] _is_ the proof of [nat_rels ...] and does not itself posses a proof.

- ps5,6 : the discussion of coercions, views and view hints is a bit confusing. View hints are mentioned, but not explained at all and it is not apparent from the text, that views are simply lemmas that adhere to a certain structure and are called such when used in a particular way. It might be helpful to clarify this section.

- p10l : "... which wraps the allocation and deallocation of the around fact_acc."; there is some part of the sentence missing here.

- While certainly not intentional, the initial wording in the 2nd and 4th paragraph of section 4.1, may appear to some readers to carry a slightly condescending connotation or subtext. It may be worth rewording these.

---------- REVIEW 2 ----------
PAPER: 17
TITLE: Introducing Functional Programmers to Interactive Theorem Proving and Program Verification (Teaching Experience Report)
AUTHORS: Ilya Sergey and Aleksandar Nanevski

OVERALL EVALUATION: -1 (weak reject)

---------- REVIEW ----------
The paper describes a new course on interactive theorem proving and program verification. The students are expected to be experienced in functional programming. The outline of the course is as follows:
- Use Ssreflect extension of Coq
- Functional programming in Coq
- Propositional reasoning
- Equality and rewriting
- Decidability and boolean reflection
- Inductive proofs
- Representation of algebraic structures using dependent records and canonical structures
- HTT (Hoare type theory) in Ssreflect

The most original part of the course is the final treatment of HTT. The parts of the course leading to HTT are non-surprising given the literature on Ssreflect. Certainly the topics covered before HTT are essential and of basic interest. From the paper I could not see whether the part on algebraic structures is essential for the presentation of HTT.

The paper is well-written and brings across its point clearly.

The course has been given once to five students majoring in mathematics and software engineering in a five day summer format. The paper reports about this experience and the student's feedback. Clearly, more iterations of the course and more students are needed for a reasonable evaluation.

The course may work as a five day full-time course for beginning PhD students in that it throws important ideas and techniques at students who such inspired and exited digest and understand the ideas afterwards given enough time and additional material.

I don't think the course will work as a university course. Here we need more of a divide and conquer strategy. A basic course may introduce logic principles based on constructive type theory and their realization in a proof assistant (e.g., Coq/SSreflect). A second course may address deductive program verification with Hoare and separation logic and how it can supported in a proof assistant. There should be substantial case studies for the use of separation logic discussed both informally (i.e., mathematically) and formally (i.e., realization in proof assistant). Merging the two courses into one and doing it in 5 days seems crazy. This may only work for students who understand program verification and separation logic well and who want to learn how this can be realized in a proof assistant. Certainly it does not suffice that the students are experienced in writing imperative and functional programs. Programm verification requires many additional skills you don't acqu!
ire by writing programs.

---------- REVIEW 3 ----------
PAPER: 17
TITLE: Introducing Functional Programmers to Interactive Theorem Proving and Program Verification (Teaching Experience Report)
AUTHORS: Ilya Sergey and Aleksandar Nanevski

OVERALL EVALUATION: -1 (weak reject)

---------- REVIEW ----------
This paper describes the syllabus of a course teaching Coq. It differs from related Coq courses because it uses ssreflect and because it explains small scale reflection and dependent records. Most of the paper describes the curriculum, including a final case study devoted to "Hoare Type Theory" (HTT). There is a brief description of the one course where this material was used.

I have a number of issues with this paper:

1. The paper is completely Coq-centric. Although the title does not mention Coq and promises some generality "Interactive Theorem Proving and Program Verification", the contents is completely focused on Coq. Of course you have to use some specific proof assistant to teach these topics, but the paper never looks further than the specific proof assistant it uses. In the list of points that set this approach apart (in Section 2) and in the description of the curriculum, it is all about very specific features of Coq.

The paper is a perfect fit for a Coq workshop but not
for a general theorem proving conference. It simply lacks the broader perspective and focuses on Coq-specific details. Just as an example, when the authors discuss the choice between ssreflect and more traditional Coq proof scripts, they never ask the global question: what kind of proof style should we teach our students to teach them something that is reusable elsewhere? I recommend "How to write a proof" by Lamport.

Note that as a comparison I looked at my other CPP papers. They use various proof assistants but the points they make are transferable. I don't see that here.

2. The course has been taught exactly once as a 5-day summer course given by the authors to 5 self-selected students of the Saint Petersburg State University. I am afraid that is a rather meager empirical basis. Nor has there been any meaningful evaluation of the course (only some anecdotal evidence), which would of course be hard for a course with such small numbers and without an exam. Teaching is like Software Engineering: to prove the effectiveness of some approach you need empirical evidence. This is sorely missing.

3. The review of related material is again very blinkered, although google will quickly reveal quite a bit of other related work. I recommend the paper

as a starting point. Its references contain a number of worthwhile published papers you should look at and discuss in your own work. They are not all about Coq, but that is precisely my point (see also 1 above).

4. The paper is unclear about what the intended audience is, except for "functional programmers". There should be a clear statement what prerequesites the students are expected to have, ideally at the beginning. Instead we get conflicting statements along the way. Two examples:

- At the beginning of 3.1.3 it says "By this moment ... they understand that values of inductive datatypes are constructed by applying constructors to the arguments and destructed by pattern matching." - I thought they are functional programmers and knew this from the beginning?

- At the beginning of 3.6.1 you write "We expected that the students of the course would all have been exposed to Hoare logic in their university education." Later you write "We didn't assume any knowledge of intuitionistic logic or Hoare logic".

5. There is a very steep jump in complexity when HTT is introduced. I frankly doubt that this will work for many students unless they are both experienced functional programmers and have had a proper introduction to Hoare logic in some other course. More importantly, I have an issue with your HTT. You introduce the complexity of separation logic and a heap merely to reason about what would normally be stack allocated variables. I would call this "Hoare logic made even more difficult". Your example factorial program, if written in standard imperative style, would be blown away by state of the art provers of imperative programs. It gives students an entirely wrong idea of the subject.

These are my major issues. I recommend a major revision and to run the course with a larger audience to obtain more than anecdotal evidence.

Minor comments:

"Thus, most undergraduates in computer science receive a strong background in programming, imperativ as well as functional." I recommend a reality check.

Lemma nat2_ind: it is somewhat sad to see that you need to prove this by hand. A decent function definition facility should automatically give you this lemma as a consequence of the termination of evenb. In which case this argument against evenb disappears.

---------- REVIEW 4 ----------
PAPER: 17
TITLE: Introducing Functional Programmers to Interactive Theorem Proving and Program Verification (Teaching Experience Report)
AUTHORS: Ilya Sergey and Aleksandar Nanevski

OVERALL EVALUATION: 0 (borderline paper)

---------- REVIEW ----------
This paper presents a newly developed course that teaches theorem proving and programming using the Coq proof assistant. The course covers an impressive breadth of topics in a very short space of time. The paper outlines the content of the course and explains some of the key design decision (use of ssreflect and minimal primitives

for proofs). It also compares the course with the available literature
for teaching Coq.

The course seems interesting and the paper is well written, although
the middle part is somewhat list-like (but I guess it's a good public
record of what was included and where). It's a shame that there were
only five students, so the evaluation part of the course turns out to
be less valuable than one would like.

I find the "Teaching Experience Report" subtitle a little misleading.
There is not that much about "Teaching Experience" in this paper;
instead there is a lot that fits under "Course Preparation".

I'm not a Coq expert, so I cannot comment on whether the specific
design decisions were sensible.

Typos:
"ocasional" --> "occasional"
"The Coq's command" --> "Coq's command"
"about half of the students" (seems a bit odd for a group of 5)
"a situation that" --> "a situation where"
"has immediately suggested" --> "immediately suggested"

---------- REVIEW 5 ----------
PAPER: 17
TITLE: Introducing Functional Programmers to Interactive Theorem Proving and Program Verification (Teaching Experience Report)
AUTHORS: Ilya Sergey and Aleksandar Nanevski

OVERALL EVALUATION: 2 (accept)

---------- REVIEW ----------
This paper describes a short introductory course about the Coq proof
assistant, aimed at functional programmers who have had no prior experience
with an interactive theorem prover. The topic of the course is program
verification for both functional and imperative code. In contrast to Pierce's
Software Foundations course, this version is based on the ssreflect extensions
and library.

Overall this paper should be accepted. It is well written, and includes enough
background for educators who are familiar with Coq but not so familiar with
ssreflect. (In fact, I found it to be a good overview of that topic.) In
addition to the experiences described in this paper the authors have made their
course materials publicly available. The lecture notes seem very
well-structured and well-prepared.

I really liked the structure of the course and the material they chose to
cover. In particular, the idea of doing propositional logic before equational
reasoning makes sense and it is good to see there experience with this
schedule.

A few high-level comments:

- The paper says that the course was taught in five days (full days of
lectures, I presume). It would have been nice to have a more detailed time
frame with the topics they cover, when each thing was exposed, how long it
took, etc.

- A criticism that I have is that the paper focuses much more on
what they chose to present then on _how_ they did it, or how _well_

it worked with the students. This is especially true for the part on HTT. Another nit is that they seem to spend a fair amount of time on the specifics of SSR (e.g. the fact that they have design patterns for doing inheritance and organizing algebraic hierarchies).

- The course itself didn't include any examinations, so it is difficult to evaluate how effective it was at teaching students the desired material. Likewise, it is not clear from section 4.2 if the students were asked questions that would reveal this information. In particular, it would be good to know how effective this course is in teaching students unfamiliar topics from program verification (such as Hoare Logic). Did the students report that their understanding of Hoare Logic changed as a result of this course material?

p3.

- This is probably a minor point, but they say "the students [...] are forced to rely on primitive recursion" -- as far as I know, even though Coq does not allow for general recursion, it can define much more than PR functions.

- They say "it is natural to explain the usual inference rules in terms of functional programming": does that mean that they assume that students are already familiar with inference rules?

p4.

- "indexed type families correspond to custom..." I couldn't really understand what they meant with this sentence. I understood a bit better how their method for presenting this worked by seeing the tables they gave afterwards, but even those were not entirely easy to grasp (e.g., the table for eq has x as a constructor, while the table for nat_rels doesn't mention n or m as arguments).

- "In fact, the case-analysis on Coq's ...": does that mean that that proof would fail with the usual definition of equality?

p6.

- I can see you mean by "orbit", but I really don't know what exactly is meant there.

p8.

- "If mutation is required, it should be done via heap references. A consequence is that the effectful commands of our programs has to be able to return non-unit results, unlike in separation logic, where no results are returned" -- Not sure I understand, please clarify...