Asynchronous Editing for Coq

Carst Tankink

Inria Saclay – Île de France

December 8, 2014

In this talk, I will demonstrate the integration of the jEdit editor with Coq, developed as part of the Paral-ITP project. jEdit is an editor with very little off-the-shelf functionality, designed to be extended by language-specific plugins. These plugins receive user changes asynchronously, and use these changes to produce feedback. Because the computation model is asynchronous, there is no way for the user to instruct where and when the proof assistant computes on the proof script, as in Proof General and CoqIDE. Instead, the proof assistant plugin is always active in the background, and automatically reacts to changes in the script, reporting states, errors and markup information.

The main intent of the talk is to demonstrate the tool, and discuss the differences between asynchronous and synchronous interaction, as well as the future directions of the middleware used by jEdit to communicate with the prover.

There are two main benefits to interacting asynchronously with Coq. The first is more freedom for the author in editing the proof, the second is that it allows the proof assistant to schedule the processing of a proof more sophisticatedly.

Interaction benefits The main benefit in interaction is that, in jEdit, the author can edit a proof script at any place, adding lemmas and modifying proofs, without advancing the computation of the prover. In fact, errors that the author introduces in a script do not always influence the state of future proofs, and the author is free to move between to proves to get the statement of a lemma just right before finishing a proof.

Sophistication of Computation Scheduling In the ProofGeneral model, the proof structure is revealed to the proof assistant step by step, and the user always wants feedback for the last line executed. In an asynchronous model, the entire proof script is available to the proof assistant at once, and can be used to compute the structure of the proof. This structure can then be used to schedule computations, possibly in parallel, to provide the user with feedback immediately.

For example, consider a proof with two lemmas. The first takes a long time to verify, but the user wants to edit the second. In ProofGeneral, when the user focuses on the second lemma, the proof assistant would first verify the first lemma, taking a long time and only then give the control back to the user.

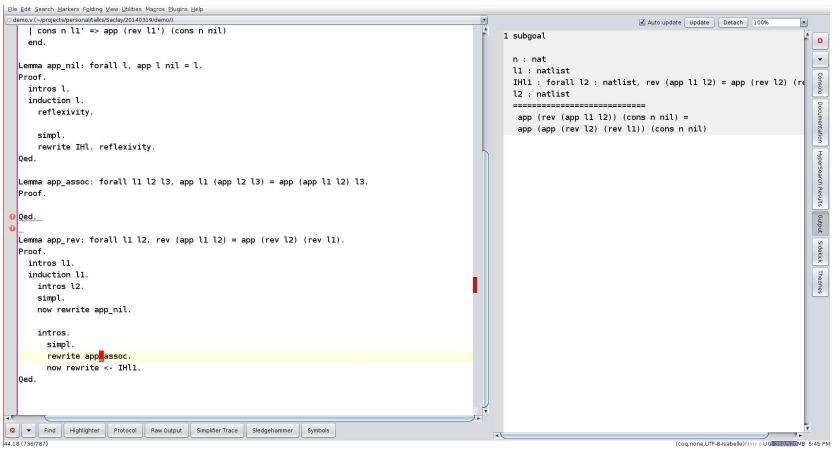

Figure 1: jEdit with Coq processing: despite the lemma app_assoc having an error, it can still be used.

In the asynchronous case, the proof assistant can postpone processing the first lemma, and first process the user's changes to the second lemma.

Approach

We obtain support for Coq in jEdit by adapting the existing editor-prover middleware (PIDE) for Isabelle [3], to work with Coq's parallel execution framework, as developed by Tassi [2].

The initial version of the Coq plugin still relies heavily on Isabelle-specific functions, but we use this implementation to factor out the generic parts, and provide Coq-specific functionality.

Thoughts

Developing the PIDE middleware to support Coq does not only bring support for jEdit, but allows developing other tools, not necessarily editors, that use the prover for some tasks. Potential avenues of exploration are web-based collaborative environment (already being explored by Ring and Lüth using the current, Isabelle-centered, implementation of PIDE [1]), or adding prover verification to question-and-answer sites like StackOverflow.[1]

References

[1] Christoph Lüth and Martin Ring. A web interface for isabelle: The next generation. In *CICM 2013*, 2013.

[2] Enrico Tassi. Designing a state transaction machine for Coq. In *Proceedings of the 2012 Coq Workshop*, 2012.

[3] Makarius Wenzel. Asynchronous proof processing with Isabelle/Scala and Isabelle/jEdit. *ENTCS*, 2012.

[1] `http://stackoverflow.com/questions/tagged/coq`