

Asynchronous processing of formal documents in Coq

Enrico Tassi

Inria

In recent years Interactive Theorem Provers (ITPs) have been successfully adopted to give an ultimate degree of reliability both to complex software systems, like the L4 microkernel and the CompCert C compiler, and to mathematical theorems, like the Odd Order Theorem. These large formalization projects have pushed interactive provers beyond their limits highlighting their deficiencies and opening for new challenges. The most compelling new requirement is *reactivity*: how long does it take the system to react to a user change and give him feedback. For example if one takes the full proof of the Odd Order Theorem [1] and makes a change, even trivial, in the first file and asks the prover any kind of feedback on the last file he has to wait approximatively 2 hours. Better reactivity can surely be achieved by faster processing of the formal documents, even more by doing it in a smarter way: giving precedence to the tasks the user is really interested in and postponing the others. In the light of that it is clear that the design of interactive provers needs to adapt to these new demands, and that this has to happen *in conjunction with their user interfaces*.

In the context of Paral-ITP project (ANR-11-INSE-001) the way Coq processes a formal document has been completely redesigned from the ground up, with the aim of allowing better reactivity, faster re-compilation and finally pave the way to modern user interfaces [3]. The seminal work of Wenzel [5, 6] in parallelizing the Isabelle system has been of great inspiration for this work.

This talk describes the new design of Coq and reports on the implementation that was undertaken during the last one and a half year. The aim is to give system developers an overview of the design choices that are crucial to obtain a reactive prover. In particular we discuss the following topics: 1) asynchronous interaction model, 2) formal document representation, 3) asynchronous proofs, 4) parallel execution in OCaml, 5) meta data collection.

Asynchronous interaction model. The first requirement for having a more reactive prover is to not impose an eager execution of commands. The Read Eval Print Loop on top of which most user interfaces are built imposes such constraint. For example the prover must tell the user interface if a command fails or succeeds immediately: the prover is not given a second chance to report any feedback. If all commands have to be executed eagerly it is then impossible to postpone uninteresting tasks. The natural solution to this problem is to adopt an asynchronous interaction loop, as originally proposed by Wenzel. Commands sent to the prover and responses set to the user interface are equipped with unique identifiers. Responses are not expected to come in any precise order: the identifiers enable the user interface to make sense of the responses and to report feedback on the text they refer to.

Document representation. Once the prover is not forced to produce an answer immediately a vast panel of possible optimizations is available. In our opinion the most interesting one is the static analysis of the document. In particular the system can represent it as a Direct Acyclic Graph (DAG) where nodes are system states and edges carry the tasks one has to perform in order to compute a state. On that representation it is easy to take scheduling decisions: for instance independent paths from one node to another can be performed in parallel. A requirement for that is that the language in which the formal

document is written has a sufficiently clear semantics. For example the subdivision of a document into separate lemmas can help to identify independent proofs. But for this to work the beginning/ending of proofs must be clearly delimited. Branches in the DAG also identify failures/recovery points. A broken proof of a statement does not necessarily make the following part of the document invalid: if one fixes such proof (without changing the statement) then there is no need to recheck what follows.

Mathematical model of asynchronously computed proofs. The logical environment of a prover like Coq contains the statement and the proof evidence of theorems. A theorem enters the logical environment only if its proof evidence has been checked. When the checking of proofs is executed out of order this condition has to be relaxed. Wenzel introduced the concept of proof promise [4] in the core logic of Isabelle for this precise purpose. Another viable option is to rely on the semantics of the underlying programming language: the data type for the proof evidence is replaced by a lazy (and pure) computation that, when executed, produces the proof evidence and checks its correctness. Finally a logical environment is said to be sound only if all the lazy computations it contains have been evaluated.

Parallel execution in OCaml. To take advantage of multi core microprocessors one can process independent tasks in parallel. Coq is written in the OCaml programming language that leaves no option but splitting the work among separate processes. While this imposes a clean message passing discipline, one has to design a mechanism to efficiently send to a worker process the relevant bits of the system state. Trimming down the system state can be particularly difficult if the state of the prover is unstructured and non first class, as in the case of Coq. The ephemerons [2] finalization mechanism was successfully employed as a solution for that problem. The data one wants to trim down is not stored directly in the system state but represented by its ephemeron access key. When the system state is marshalled and sent over the wire, the unwanted data is not sent along with its corresponding ephemeron key since it lives outside the marshalled object. Finally, when an ephemeron key becomes unreferenced its corresponding data is eventually collected by the OCaml runtime, as if such data was equally unreferenced.

Metadata collection. When multiple workers are checking independent parts of the same document the feedback each worker produces needs to reach the user interface. In our design the master process acts as a proxy and provides a mono directional bus on which workers write.

References

- [1] Georges Gonthier, Andrea Asperti, Jeremy Avigad, Yves Bertot, Cyril Cohen, François Garillot, Stéphane Le Roux, Assia Mahboubi, Russell O'Connor, Sidi Ould Biha, Ioana Pasca, Laurence Rideau, Alexey Solovyev, Enrico Tassi & Laurent Théry (2013): *A Machine-Checked Proof of the Odd Order Theorem*. In: *Proceedings of ITP, LNCS 7998*, pp. 163–179. Available at <http://hal.inria.fr/hal-00816699>.
- [2] Barry Hayes (1997): *Ephemerons: A New Finalization Mechanism*. In: *Proceedings of OOPSLA*, pp. 176–183. Available at <http://doi.acm.org/10.1145/263698.263733>.
- [3] Carst Tankink (2014): *PIDE for asynchronous interaction with Coq*. In: *Proceedings of UITP*.
- [4] Makarius Wenzel (2009): *Parallel proof checking in Isabelle/Isar*. In: *Proceedings of PLMMS*, pp. 13–21.
- [5] Makarius Wenzel (2011): *Isabelle as Document-Oriented Proof Assistant*. In: *Proceedings of ICM/MKM, LNCS 6824*, pp. 244–259. Available at http://dx.doi.org/10.1007/978-3-642-22673-1_17.
- [6] Makarius Wenzel (2013): *READ-EVAL-PRINT in Parallel and Asynchronous Proof-checking*. In: *Proceedings of UITP*, pp. 57–71. Available at <http://dx.doi.org/10.4204/EPTCS.118.4>.