Performance Engineering of Proof-Based Software Systems at Scale

by

Jason S. Gross

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2021

© Jason S. Gross, MMXXI. All rights reserved.

The author hereby grants to MIT permission to reproduce and to distribute publicly paper and electronic copies of this thesis document in whole or in part in any medium now known or hereafter created.

Author . . . . . .
Department of Electrical Engineering and Computer Science
August 19, 2015

Certified by . . . . . .
Adam Chlipala
Associate Professor of Computer Science
Thesis Supervisor

Accepted by . . . . . .
Leslie A. Kolodziejski
Chair, Department Committee on Graduate Students

Performance Engineering of Proof-Based Software Systems at Scale

by
Jason S. Gross

Submitted to the Department of Electrical Engineering and Computer Science
on August 19, 2015, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computer Science and Engineering

Abstract

Formally verified proofs are important. Unfortunately, large-scale proofs, especially automated ones, in Coq, can be quite slow.

This thesis aims to be a partial guide to resolving the issue of slowness. We present a survey of the landscape of slowness in Coq, with a number of micro- and macro-benchmarks. We describe various metrics that allow prediction of slowness, such as term size, goal size, and number of binders, and note the occasional surprise-lack-of-bottleneck for some factors, such as total proof term size.

We identify three main categories of workarounds and partial solutions to slowness: design of APIs of Gallina libraries; changes to Coq's type theory, implementation, or tooling; and automation design patterns, including proof by reflection. We present lessons drawn from the case-studies of a category-theory library, a proof-producing parser-generator, and a verified compiler and code generator for low-level cryptographic primitives.

The central new contribution presented by this thesis, beyond hopefully providing a roadmap to avoid slowness in large Coq developments, is a reflective framework for partial evaluation and rewriting which, in addition to being used to compile a code-generator for field arithmetic cryptographic primitives which generates the code currently used in Google Chrome, can serve as a template for a possibly replacement for tactics such as `rewrite`, `rewrite_strat`, `autorewrite`, `simpl`, and `cbn` which achieves much better performance by running in Coq's VM while still allowing the flexibility of equational reasoning.

Thesis Supervisor: Adam Chlipala
Title: Associate Professor of Computer Science