[**TODO:** Make sure to exclude no files (not rewriting)]

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

February 2021

© Massachusetts Institute of Technology 2021. All rights reserved.

Author . . . . . .
Department of Electrical Engineering and Computer Science
(draft)

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 (draft), in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computer Science and Engineering

Abstract

Formal verification is increasingly valuable as our world comes to rely more on software for critical infrastructure. A significant and under-studied cost of developing mechanized proofs, especially at scale, is the computer performance of proof generation. This thesis aims to be a partial guide to identifying and resolving performance bottlenecks in dependently typed tactic-driven proof assistants such as Coq.

We present a survey of the landscape of performance issues in Coq, with a number of micro- and macro-benchmarks. We describe various metrics that allow prediction of performance, 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. To our knowledge such a roadmap to performance bottlenecks is a new contribution of this thesis.

We identify three main categories of workarounds and partial solutions to performance problems: 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 technical contribution presented by this thesis is a reflective framework for partial evaluation and rewriting, already used to compile a code generator for field-arithmetic cryptographic primitives which generates code currently used in Google Chrome. We believe this prototype is the first scalably performant realization of an approach for code specialization which does not require adding to the trusted code base. Our extensible engine, which combines the traditional concepts of tailored term reduction and automatic rewriting from hint databases with on-the-fly generation of inductive codes for constants, is also of interest to replace these ingredients in proof assistants' proof checkers and tactic engines. Additionally, we use the development of this framework itself as a case study for the various performance issues that can arise when designing large proof libraries.

Finally, we present a novel method of simple and fast reification, developed and published during the course of doctoral study.

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