Formally verified algorithms and data structures in Coq: concepts and techniques

(Formalnie zweryfikowane algorytmy i struktury danych w Coqu: koncepty i techniki)

Wojciech Kołowski

Praca magisterska

Promotor: narazienikt

Uniwersytet Wrocławski
Wydział Matematyki i Informatyki
Instytut Informatyki

Czerwiec '20 chyba że koronawirus

Abstract

We discuss how to design, implement, specify and verify functional algorithms and data structures, concentrating on formal proofs rather than asymptotic complexity or actual performance. We present concepts and techniques, both of which often rely on one key principle – the reification and representation, using Coq's powerful type system, of something which in the classical-imperative approach is intangible, like the flow of information in a proof or the shape of a function's recursion. We illustrate our approach using rich examples and case studies.

Omawiamy sposoby projektowania, implementowania, specyfikowania i weryfikowania funkcyjnych algorytmów i struktur danych, skupiając się bardziej na dowodach formalnych niż na asymptotycznej złożoności czy faktycznym czasie działania. Prezentujemy koncepty i techniki, obie często opierające na jednej kluczowej zasadzie – reifikacji i reprezentacji, za pomocą potężnego systemu typów Coqa, czegoś co w klasycznym, imperatywnym podejściu jest nieuchwytne, jak przepływ informacji w dowodzie czy kształt rekursji funkcji. Nasze podejście bogato ilustrujemy przykładami i studiami przypadku.

Contents

Chapter 1

Introduction

1.1 Imperative and functional algorithms

Differences between imperative and functional algorithms.

1.2 Complexity, performance and correctness

Differences between performance-oriented design and formal-correctness-oriented design.

1.3 An overview of available literature

Literature review, Okasaki is old and bad for Coq, SF3 is shallow.

Chapter 2

Binary search trees – an extended case study

Binary search trees: a case study to show the basic workflow and that it's not that obvious how to get basic stuff right.

Chapter 3

A man, a plan, a canal – MSc thesis

3.1 Things to write about

- Design: we shouldn't require proofs in order to run programs. Ways of doing general recursion and, connected with it, functional induction as the way-to-go proof technique. Maybe something about the equations plugin. A word about classes, records and modules.
- Quicksort: in functional languages we have so powerful abstractions that we can actually implement algorithms and not just programs.
- Braun mergesort: in order not to waste resources, we sometimes have to reify abstract patterns, like the splitting in mergesort.
- Cool data structures: ternary search trees, finger trees.

Bibliography