formal verification of very large proofs in coq

luís cruz-filipe[1]

(joint work with peter schneider-kamp[1]
and also kim skak larsen[1], joão marques-silva[2])

[1]department of mathematics and computer science
university of southern denmark

[2]department of informatics
faculty of science, university of lisbon

part workshop, dtu compute, copenhagen
september 7th, 2017

outline

outline

how it all started

a problem in sorting networks

how many compare-and-swap gates are needed to construct a circuit that sorts any n inputs?

how it all started

a problem in sorting networks

how many compare-and-swap gates are needed to construct a circuit that sorts any n inputs?

deceptively simple

- known solutions for $n \leq 8$ since the 1960s (knuth & floyd + van voorhis)
- solved for $n = 9$ (corollary: $n = 10$) in 2014 (codish, lcf, frank & psk @ ictai'14)

how it all started

a problem in sorting networks

how many compare-and-swap gates are needed to construct a circuit that sorts any n inputs?

deceptively simple

- known solutions for $n \leq 8$ since the 1960s (knuth & floyd + van voorhis)
- solved for $n = 9$ (corollary: $n = 10$) in 2014 (codish, lcf, frank & psk @ ictai'14)

$\rightsquigarrow$ computed by an *ad-hoc* prolog program, made some people unhappy

making more people happy

a formalized coq proof (lcf & psk @ itp'15)

- directly reimplement prolog code in coq, prove its soundness
- use an oracle to skip expensive search steps
- directly verified all cases up to $n = 8$

massive optimizations (lcf & psk @ cicm'15)

- use extra information about properties of the oracle
- modify the oracle to optimize performance
- verified also $n = 9$

a methodology?

the interesting question

can we generalize this?

a methodology?

the interesting question

can we generalize this?

yes?

abstract main ideas, propose a general methodology
(lcf, larsen & psk @ jar, accepted)

- identify relevant characteristics of the problem
- describe our workflow abstractly
- propose other possible applications

a methodology?

the interesting question

can we generalize this?

yes?

abstract main ideas, propose a general methodology
(lcf, larsen & psk @ jar, accepted)

folklore

generalization not in the accepted version

- "trivial ideas", "folklore", "common knowledge"
- missing a formal evaluation

a new application

unsat verification (lcf, marques-silva & psk @ tacas'16)

goal: blindly follow the methodology

$\rightsquigarrow$ in direct conflict with previous approaches

a new application

unsat verification (lcf, marques-silva & psk @ tacas'16)

goal: blindly follow the methodology

$\rightsquigarrow$ in direct conflict with previous approaches

a huge surprise

- prototype after two days (up to 30$\times$ slower than drat-trim)
- significant increase to state-of-the-art
- one week after preprint: two independent replications
- at cade'17: three extensions to more expressive format
  coq and acl2 (lcf, heule, hunt, kaufmann & psk)
  isabelle (lammich)

outline

optimal sorting networks (i/ii)

sorting networks

- a *comparator network* on n inputs is a sequence of compare-and-swap gates, represented as pairs $\langle i, j \rangle$ with $1 \leq i < j \leq n$
- a gate $\langle i, j \rangle$ acts on an input $\bar{x} \in \{0, 1\}^n$ by exchanging x_i and x_j if $x_i > x_j$
- a *sorting network* on n inputs is a comparator network that sorts all inputs in $\{0, 1\}^n$
- a sorting network is *(size-)optimal* if there is no shorter sorting network on the same number of inputs

optimal sorting networks (ii/ii)

finding optimal sorting networks

the generate-and-prune method

- start from the empty sequence
- add one comparator in all possible ways
- remove all networks N such that $\exists N' \exists \pi . N' \leq_\pi N$

$\rightsquigarrow$ details of the subsumption relation $N' \leq N$ are immaterial (but note existential quantifiers)

propositional unsatisfiability

the problem

given a propositional formula in cnf, show that it is unsatisfiable

usual techniques

- add clauses that are logical consequences of the cnf
- add clauses that are redundant (preserve satisfiability)
- derive the empty clause

outline

existential subproblems

requirement

proof includes problems of the form $\exists X.\varphi$, with X hard to find

existential subproblems

requirement

proof includes problems of the form $\exists X.\varphi$, with X hard to find

sorting networks

- networks to be removed
- justifications for removal

unsatisfiability proofs

- clauses to be added
- justifications for additions

$\leadsto$ suggests using an oracle

data dependency

requirement

proof's structure depends on the answers computed along the way

data dependency

requirement

proof's structure depends on the answers computed along the way

sorting networks

- there are several possibilities, e.g.
  $N \leq_\pi N'$ and $N' \leq_{\pi'} N$ or $N_1 \leq_{\pi_1} N_2 \leq_{\pi_2} N_3$
- order affects efficiency and future removals

unsatisfiability proofs

- adding clauses yields new derivable clauses
- removing clauses makes cnf smaller, prevents some derivations

$\rightsquigarrow$ suggests the performance of the oracle is relevant

known subproblems

requirement

we can compute all answers before executing the proof

known subproblems

requirement

we can compute all answers before executing the proof

in both cases

the oracle's information determines future execution

$\rightsquigarrow$ allows global optimizations of the oracle

outline

formalize

goal

formalize the theory underlying the problem, without focusing on the actual target proof

formalize

goal

formalize the theory underlying the problem, without focusing on the actual target proof

sorting networks

theory of sorting networks

- closely followed knuth's reference work + one article about subsumption
- around 2 months' (low-priority) work

formalize

goal

formalize the theory underlying the problem, without focusing on the actual target proof

unsatisfiability proofs

propositional logic

- deep encoding in coq
- standard definitions (satisfaction, entailment)
- around 1/2 day's work

implement

goal

implement a straightforward checker using an oracle

implement

goal

implement a straightforward checker using an oracle

sorting networks

follow original prolog code

- oracle produces subsumption triples $\langle N, \pi, N' \rangle$ generated by the original proof

$\rightsquigarrow$ can verify optimality for up to 8 inputs

implement

goal

implement a straightforward checker using an oracle

unsatisfiability proof

based on reverse unit propagation

- oracle also indicates which clauses to use in reverse unit propagation

$\rightsquigarrow$ can check large unsatisfiability proofs in reasonable time

optimize and reprove

goal

target bottlenecks in execution, change oracle and checker in lockstep to improve efficiency

optimize and reprove

goal

target bottlenecks in execution, change oracle and checker in lockstep to improve efficiency

sorting networks

- better data structures for searching
- rearrange order of subsumptions for list-based search
- delay checking of side conditions for better performance

⇝ requires previous knowledge of all subsumptions, as some justifications have to be changed
⇝ around 1–2 days per change
⇝ able to verify optimality for 9 inputs

optimize and reprove

goal

target bottlenecks in execution, change oracle and checker in lockstep to improve efficiency

unsatisfiability proofs

- better data structures for storing/analysing cnf
- delete clauses as soon as possible

$\rightsquigarrow$ requires previous knowledge of whole proof to make sure clauses can be deleted

$\rightsquigarrow$ changes in data structures harder due to problems with the coq libraries

$\rightsquigarrow$ competitive execution times

outline

conclusions

- verification of very large proofs using a coq-certified checker
- systematic use of the same methodology
- starting to "inspire" other researchers :)

thank you!