End-to-End Verification for Subgraph Solving [Demo]

Yong Kiam Tan

Institute for Infocomm Research, A*STAR and Nanyang Technological University

WHOOPS '25

Talk Logistics

- This talk is mainly a **demo** of how a proof checker is verified using infrastructure from the CakeML project.
- Complementary to most other talks at WHOOPS, but especially "Proof logging for subgraph solving" by Ciaran McCreesh
- Feel free to interrupt on Zoom if you need more clarification.
- Agenda:
  - Brief introduction to interactive theorem proving.
  - A quick tour of tools used to verify the CakePB "backend".
  - A quick tour of a CakePB "frontend" for clique solving.

Interactive Theorem Proving (ITP)

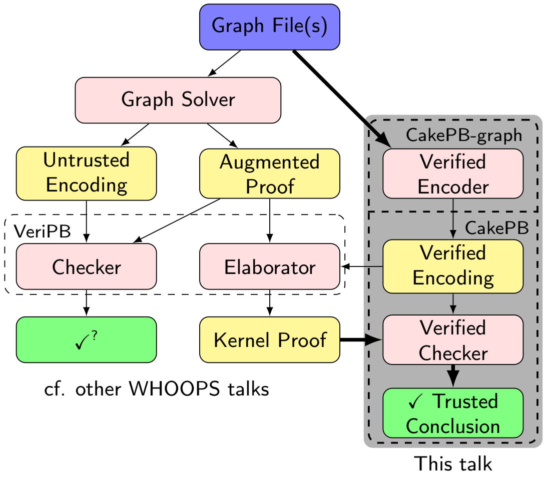
"By interactive theorem proving, we mean some arrangement where the machine and a human user ***work together interactively*** *to produce a formal proof." — quoted from Harrison et al. [1]*

- Lots of recent interest in ITP, especially with Lean.
- History of ITPs goes way back (around 60s/70s).
- Integration of automated tools in ITPs has been hugely successful, e.g., Sledgehammer in Isabelle/HOL.
- **Proof certificates/logs** can be independently checked in an ITP as a way of obtaining automation **without** trusting ATPs.

This talk: Proof checking, where the checker's implementation itself is verified inside an ITP—note the subtle difference!

Demo: HOL4 interaction.

Verified Proof Checking for Subgraph Solving (I)

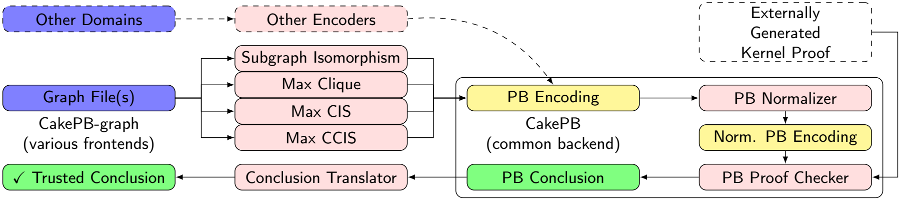


General Workflow:

1. Solver generates result & proof.
2. VeriPB elaborates proof into (simpler) kernel format.
3. CakePB checks elaborated proof (with verified encoding).

Demo: CakePB-graph (verified PB proof checker for max clique size)

Verified Proof Checking for Subgraph Solving (II)



- It would be very expensive to build a whole new proof checking framework for each new graph/combinatorial problem.
- 0-1 ILP (PB) reasoning has served well as a common, expressive language for such proofs.

The same benefit applies to the **verified checker**, with a common PB checking "backend" supporting several proof checker "frontends".

+ Diverse frontend usage informs key optimizations for CakePB backend.
+ CakePB backend optimizations automatically benefit every frontend.

CakePB (backend)

Our proof checkers are built and verified within the CakeML project.

- Programming language (CakeML) with formal semantics and a **verified compiler** to machine code (x86, ARM8, etc.).
- Various tools for generating verified CakeML code.

CAKEML
A Verified Implementation of ML

Demo: A taste of verification tools used in CakePB.

- Defining a syntax and semantics for PB constraints.
- Proving (abstractly) some cutting planes rules.
- Refinement towards formally verified CakeML source code (translation, separation logic).

CakePB-graph (frontend)

Our proof checkers are built and verified within the CakeML project.

- Programming language (CakeML) with formal semantics and a **verified compiler** to machine code (x86, ARM8, etc.).
- Various tools for generating verified CakeML code.

Demo: Putting things together with a clique frontend and an end-to-end compilation theorem.

- Defining a syntax and semantics for clique.
- Proving the encoding into PB.
- Plugging together the encoder and backend.

Summary

Our proof checkers are built and verified within the CakeML project.

- Programming language (CakeML) with formal semantics and a **verified compiler** to machine code (x86, ARM8, etc.).
- Various tools for generating verified CakeML code.

Get in touch with us if you:

- Need a new verified CakePB frontend.
- Need improvements to CakePB's backend performance.
- Want either of the above, AND you are keen to dive into the verification yourself. We're happy to help you get started!

References

[1] Harrison, J., Urban, J., and Wiedijk, F. (2014). History of interactive theorem proving. In Siekmann, J. H., editor, Computational Logic, volume 9 of Handbook of the History of Logic, pages 135–214. Elsevier.