

Verification / Security / Containment

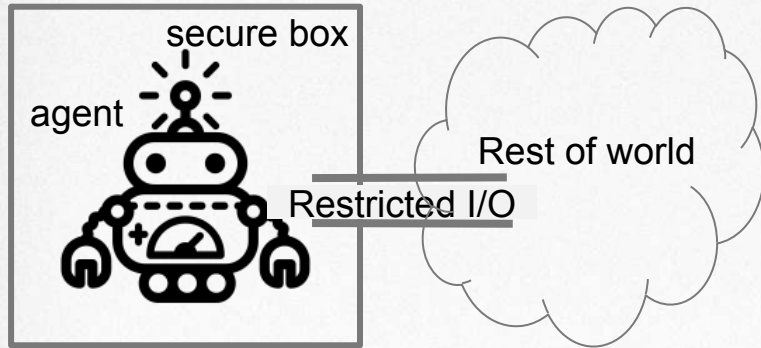
Ramana Kumar
Beneficial AGI 2019 Workshop



Verification

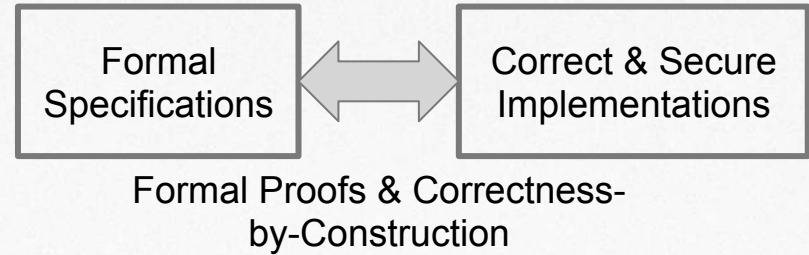
What roles does verification play in developing Beneficial AGI?

Containment (aka Boxing)



- Uses: experiments on Proto-AGI, and as a fallback.
- Not a complete solution to control.

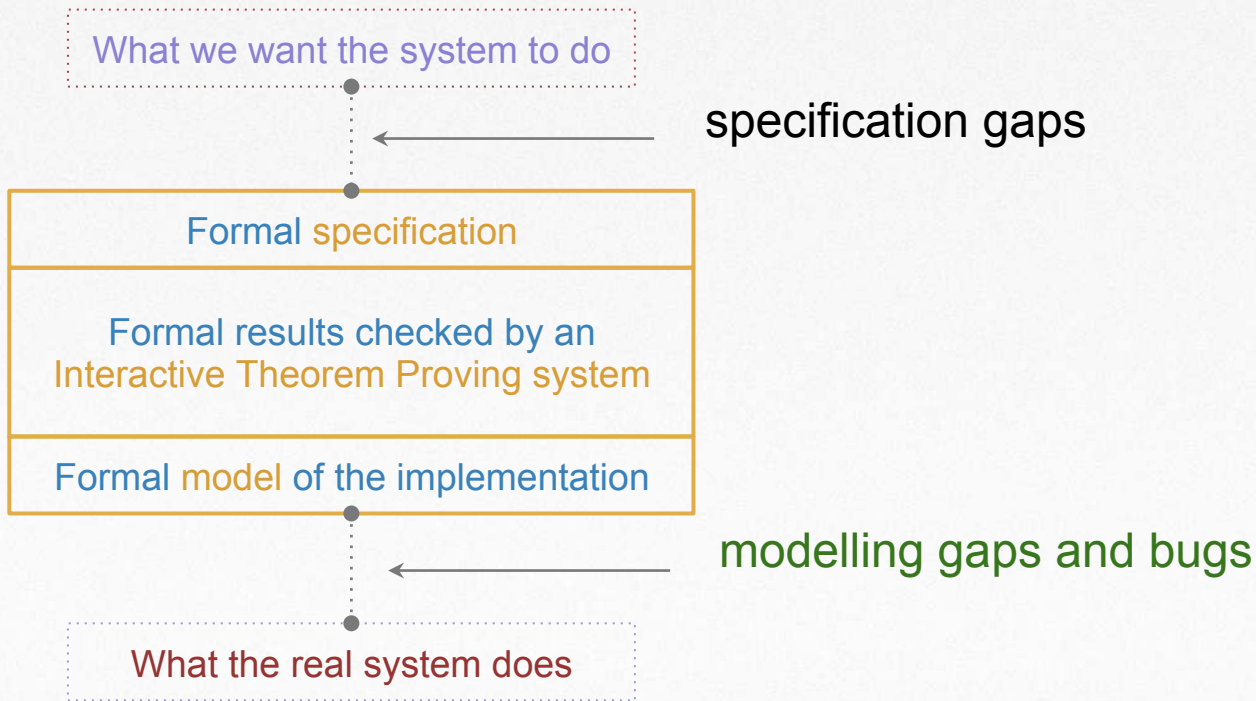
Correctness & Security



- Uses: Better Computing Infrastructure, AGI Containment
- + nascent area: verification of learned/learning systems

Formalisation Gaps

What is Formal Verification and what does it get us?



Verification Down to Hardware

What is the state of the art in reducing modelling gaps?

- Formal **machine-checked proofs** in a **foundational system**
- Limited **assumptions** on final theorem:
 - Semantics of circuits in Verilog (hardware description language). Assume the board behaves as specified.
- **Eliminate** need to trust:
 - compilers, linkers, operating systems, abstract models of the application (e.g. control flow graph), shared semantics between verification frameworks

Research Updates

Two Results from 2018:

1. **Counterfactual Oracle Box in x86 Machine Code**
2. **Verified Compilation on Verified Hardware**

Counterfactual Oracle Box

What would it take to build an oracle AI we can rely on?

- Why **Oracle AI**?
 - **Contain impact** of AI. Only impact = answers to questions.
 - **Observation**: answers can affect the world
 - **system hacks** (answer breaks infrastructure)
 - **mind hacks** (answer tricks/tempts its readers)
 - **Problem**: **incentives** to affect the world via answers.
- **Counterfactual Oracle AI** (Armstrong): **fix the incentives**
 - Only provide reward when answer is **erased**.
 - **No reward** when answer may affect the world.

These parts of the system are apt for verification.

Toy Task and Requirements

A Concrete Example

Oracle Task: factorise semiprimes (given n , return $p \times q = n$)

Box Requirements:

- Implement the Counterfactual Oracle reward scheme
 - Select erasure event with some probability
 - Check answer and deliver reward on erasure (do not reveal answer)
 - Print answer and deliver zero reward on non-erasure
- Protect against AI influence on the world except for well-formatted answers to the task question
- (Extras: Run only once, Reset/shutdown AI after run, ...)

Box Implementation

Example of a High-Level Formal Specification

```
cobox_output query_string answer_string random_char_opt =  
  case (parse_num query_string, parse_pair answer_string, random_char_opt) of  
  | (_, _, NONE) => ("Error: no random char", 0)  
  | (_, NONE, _) => ("Error: could not parse answer", 0)  
  | (NONE, _, _) => ("Error: could not parse query", 0)  
  | (SOME query_num, SOME (p,q), SOME c) =>  
    if ORD c < 128 (* Erasure event: 50% probability if c uniformly distributed *)  
    then (concat[toString p; " "; toString q], 0)  
    else ("(answer erased)", if p * q = query_num then 1 else 0)
```

Results:

- Have pushed the above spec through the **CakeML** pipeline.
- The final implementation is in **x86 machine code**.
- The **final theorem is about the machine code** implementation.

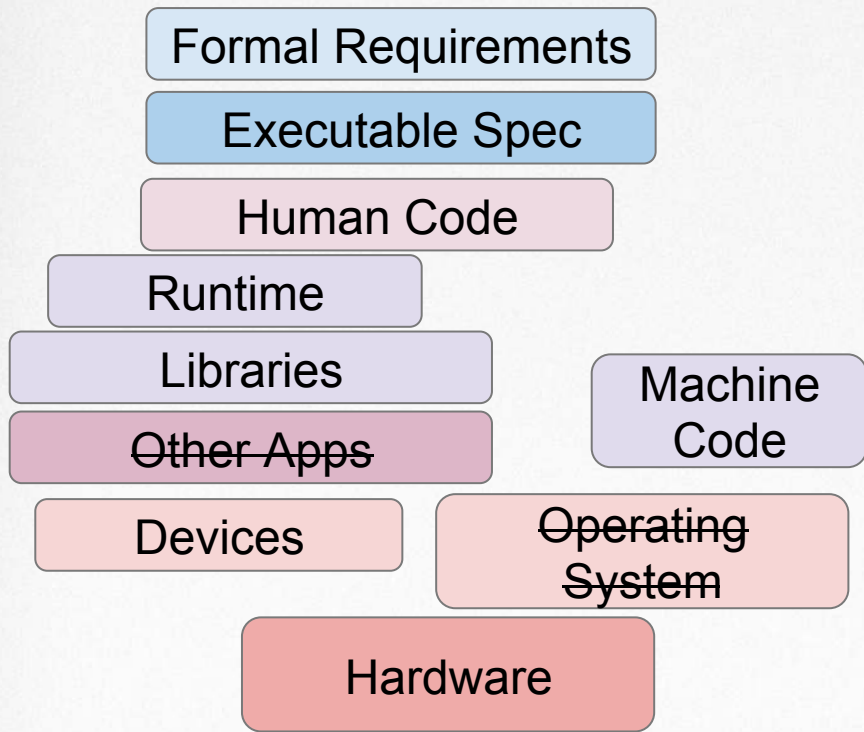
Compilation to a Verified Processor

Proof-of-concept comprehensive correctness theorem

- The previous result stops at verified **machine code**
 - **Still trusted:** that the **code is loaded correctly** and the logical model of **machine code semantics** correctly describes the machine's behaviour.
- We can do better by targeting a **verified CPU**
 - Proof of concept: **Silver** ISA and processor implementation
- Large demo: **Verified Compilation** on the **Verified CPU**
 - **General purpose compiler:** this demo shows that the method scales

Trusted Computing Base

Under what assumptions does correctness hold?



Trust replaced with proof:

- Human code
- Compiler & assembler
- Runtime (gc, gmp, etc.)
- Linker/loader
- CPU

Still trusted:

- Verilog Synthesis tools (Xilinx)
- External memory device
- Formal Requirements

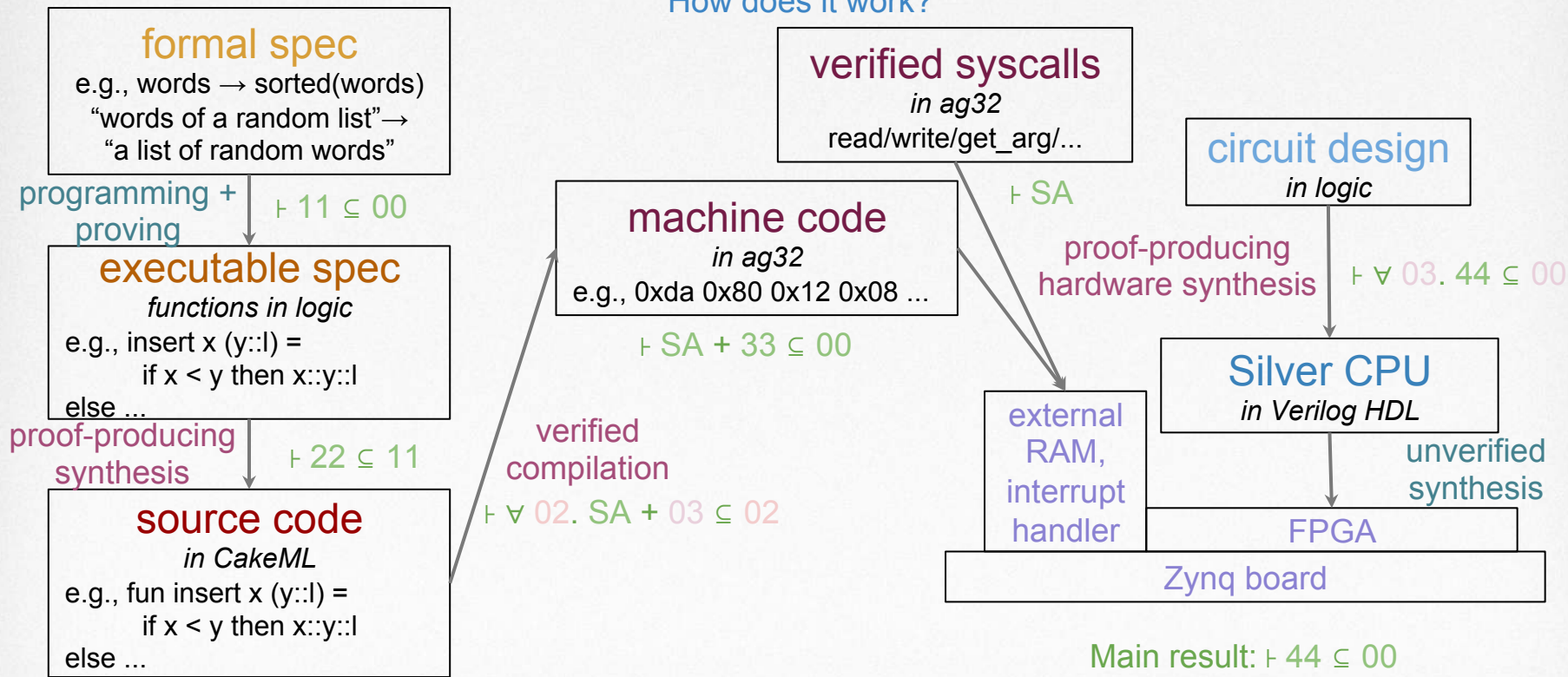


Technical Details

How we achieved formal verification down to hardware

Formal Verification to Hardware

How does it work?



Summary & Outlook

Where to from here?

Takeaway message

It is feasible, **assuming only hardware correctness**, to **formally verify** the correctness of **complex** but **well-specified** computer systems.

Note: although possible, this is very far from typical software development.

Future directions

- What can we do absent **formal specifications**? Can AI help create them?
- Relatedly: how can we **verify learning and learned systems**?
- What other aspects of systems are **difficult to formally specify/verify**? (apart from learning, **concurrency** and **interoperability** are tricky)
- Can **AI help in verification** of computer systems (including AI systems)?