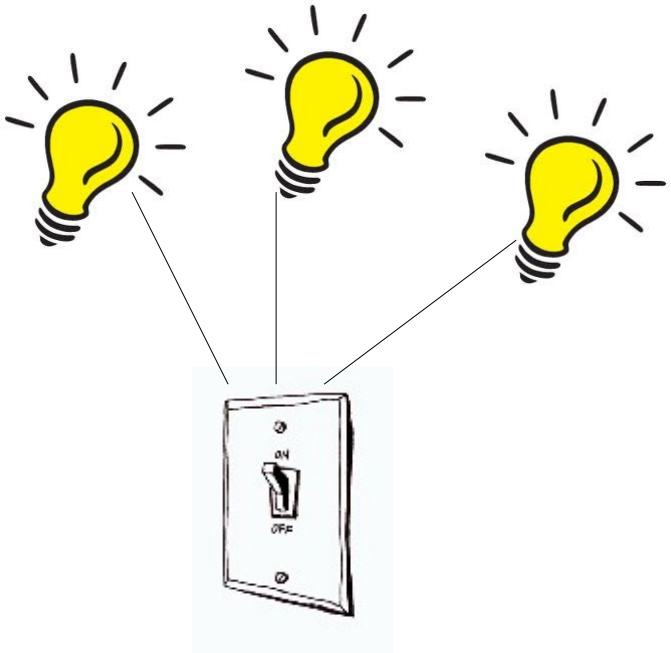
Formal Methods

Formalism vs. Verification

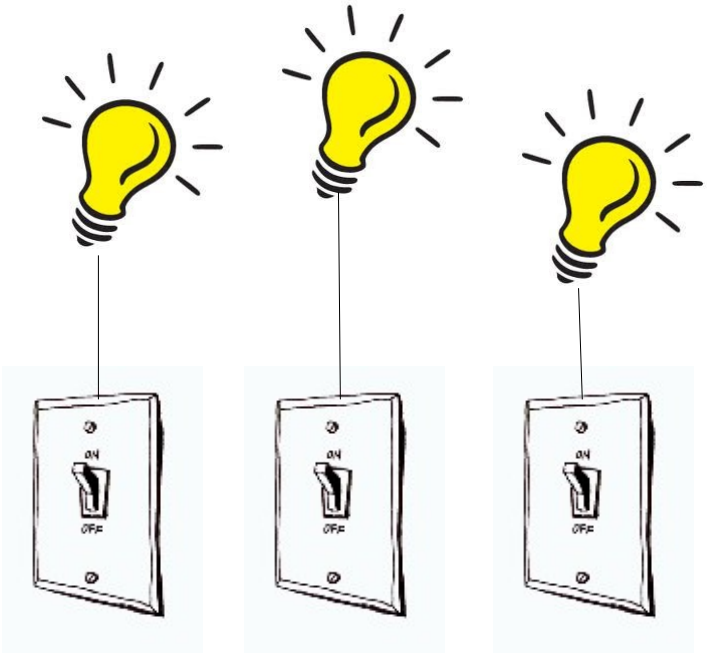
- Verification means proving the program correct
- Depends on formal and correct specifications
  - How do you verify the specs?
  - Applies mainly/only to algorithmic parts
- Verification implies formalism
- Formalism can be less than verification

example

“all lights in the room are controlled by a switch”



There exists a single switch that controls all lights



For each light, there exists a unique switch that controls it

Lesser Formal Methods

- Be precise
  - Say exactly what you mean
- Reduce ambiguity
  - Rely on agreed semantics
- Be comprehensive
  - Use a checklist of attributes
- Be methodical
  - e.g. the multiple charts of UML

David Parnas, Really rethinking “formal methods”, *Computer* **41(3)**, pp. 28-34, Jan 2010

Formal methods are not really being used by industry

- If they were, we wouldn't see papers about success stories
- Claims don't stand up to scrutiny
  - Heroic efforts needed
  - Overselling of method or results
- Many successes are simple byproduct of smart people scrutinizing the code
- Industry would use anything that gives benefits; they don't use Z and other formalisms

Three alarming gaps:

- Research vs. practice
  - Academics do mathematics unrelated to real programs and large systems
  - Programmers don't get math
- Software vs. other engineering disciplines
  - We teach technology, not applicable science
  - Speak different language from other engineers
- Computer science vs. mathematics
  - We invent new mathematics and don't use enough classical approaches

Rethinking state:

- In programs variables define the state
- In math they are placeholders
- This is not the same thing
  - Are a[i] and a[2] the same or not?
- Need to find a good way to represent state

Rethinking termination:

- Normally we require programs to terminate to be considered correct
- Extension: partial correctness, where if the program terminates then the answer is correct
- But many programs are designed to run indefinitely
  - Specifically reactive systems
- Need to find a good way to represent normal non-termination

- Similarly, nondeterminism is normal
  - But most formalisms don't handle it
- Side effects are also normal
  - But again most formalisms don't handle them

Rethinking time:

- Normally we don't consider time as part of correctness
- In real-time systems this is crucial
  - Can't be too slow or too quick
- Need to find a good way to represent time without special handling

The role of mathematics

- In software, it is to prove correctness
- In engineering, it is to calculate quantities
  - Engineering is typically about choosing among alternative “correct” designs
  - Use calculations to make comparisons
- Mathematical abstractions must still be true
  - Simplification leading to untrue predictions are harmful