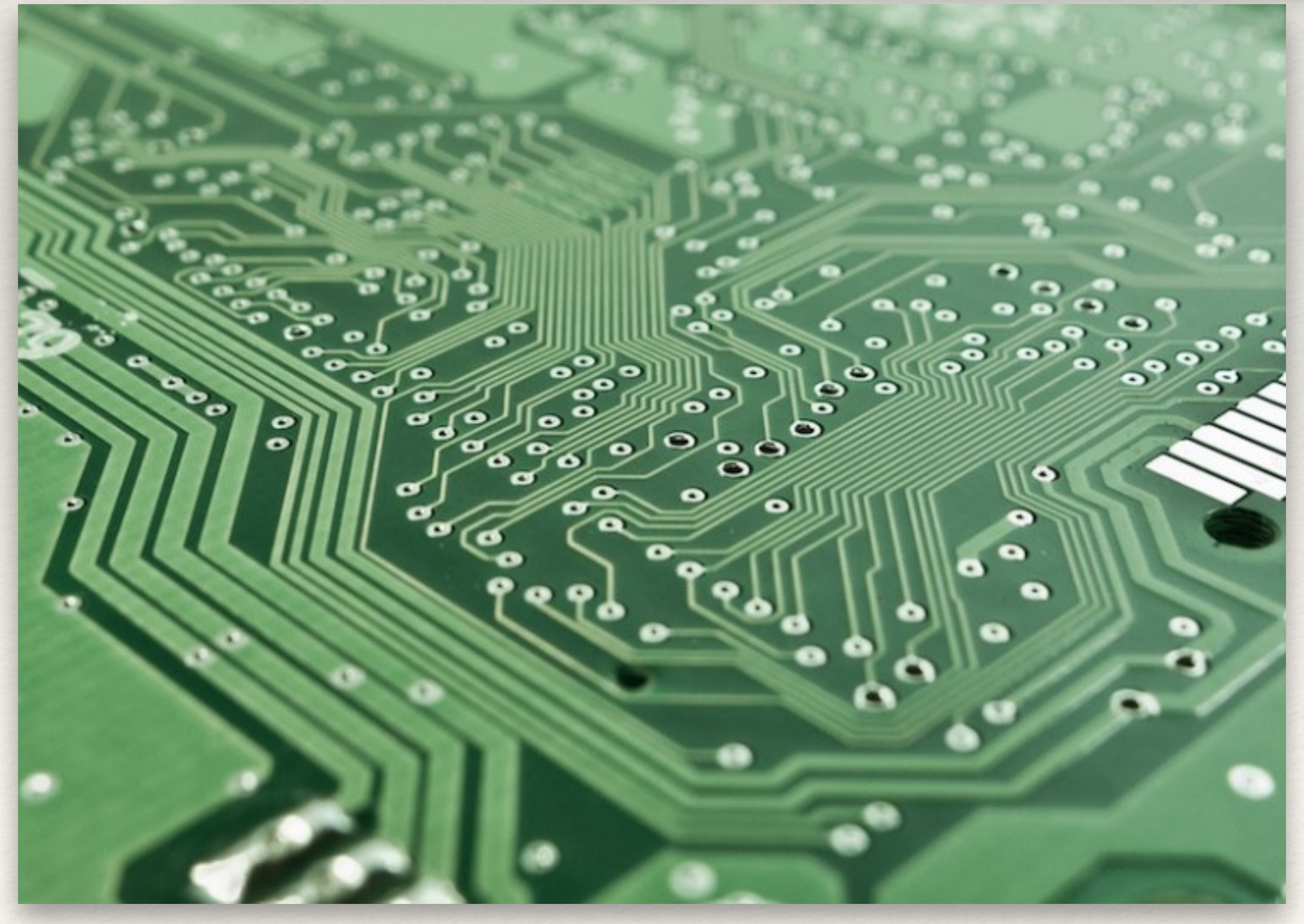
Formal Checkers and Solvers for Hardware Design and Verification
Part I

Pono:
Performant, Adaptable, and Extensible SMT-based Model Checking

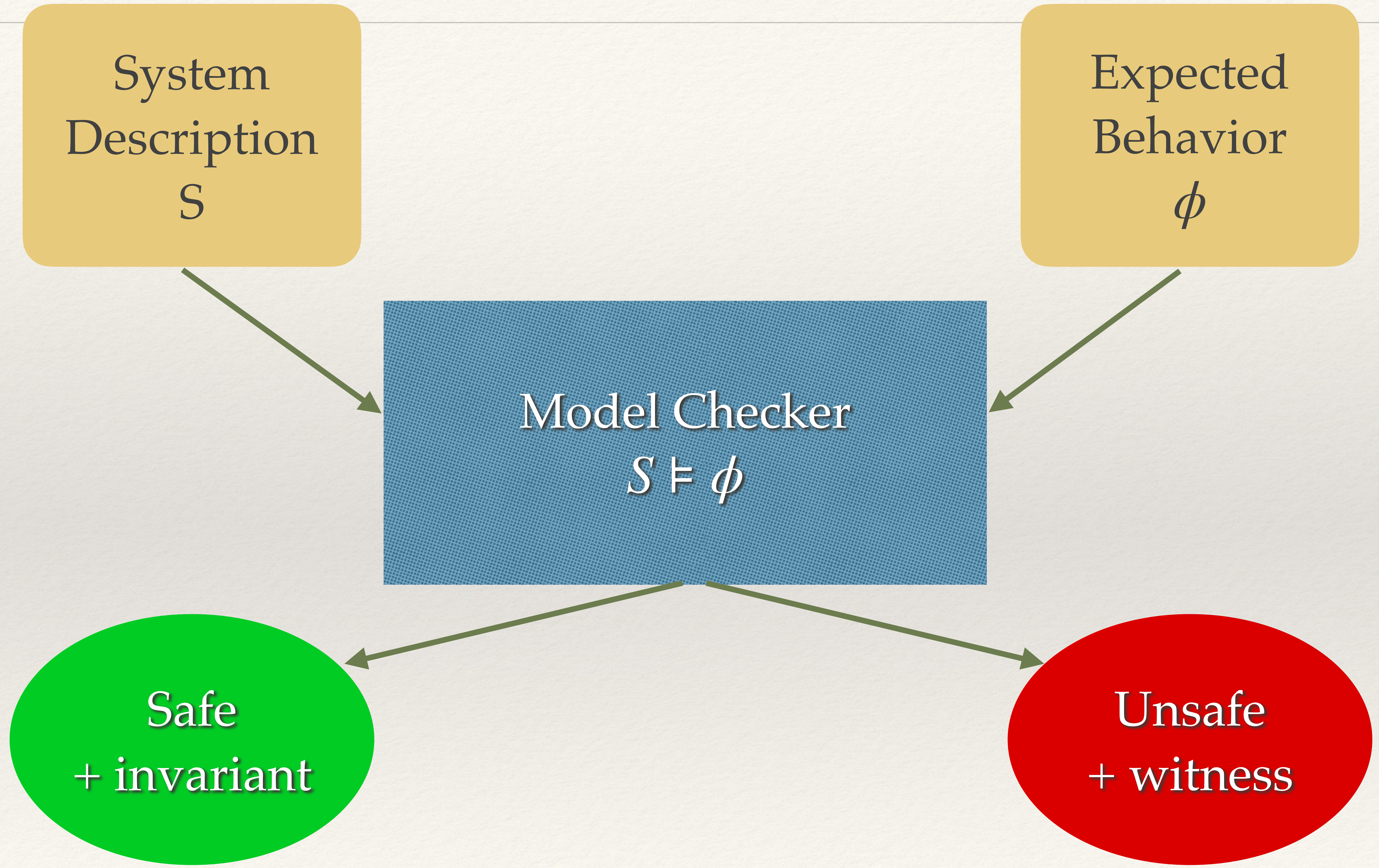
Makai Mann
Ahmed Irfan
Florian Lonsing
Yahan Yang
Clark Barrett

Formal Verification

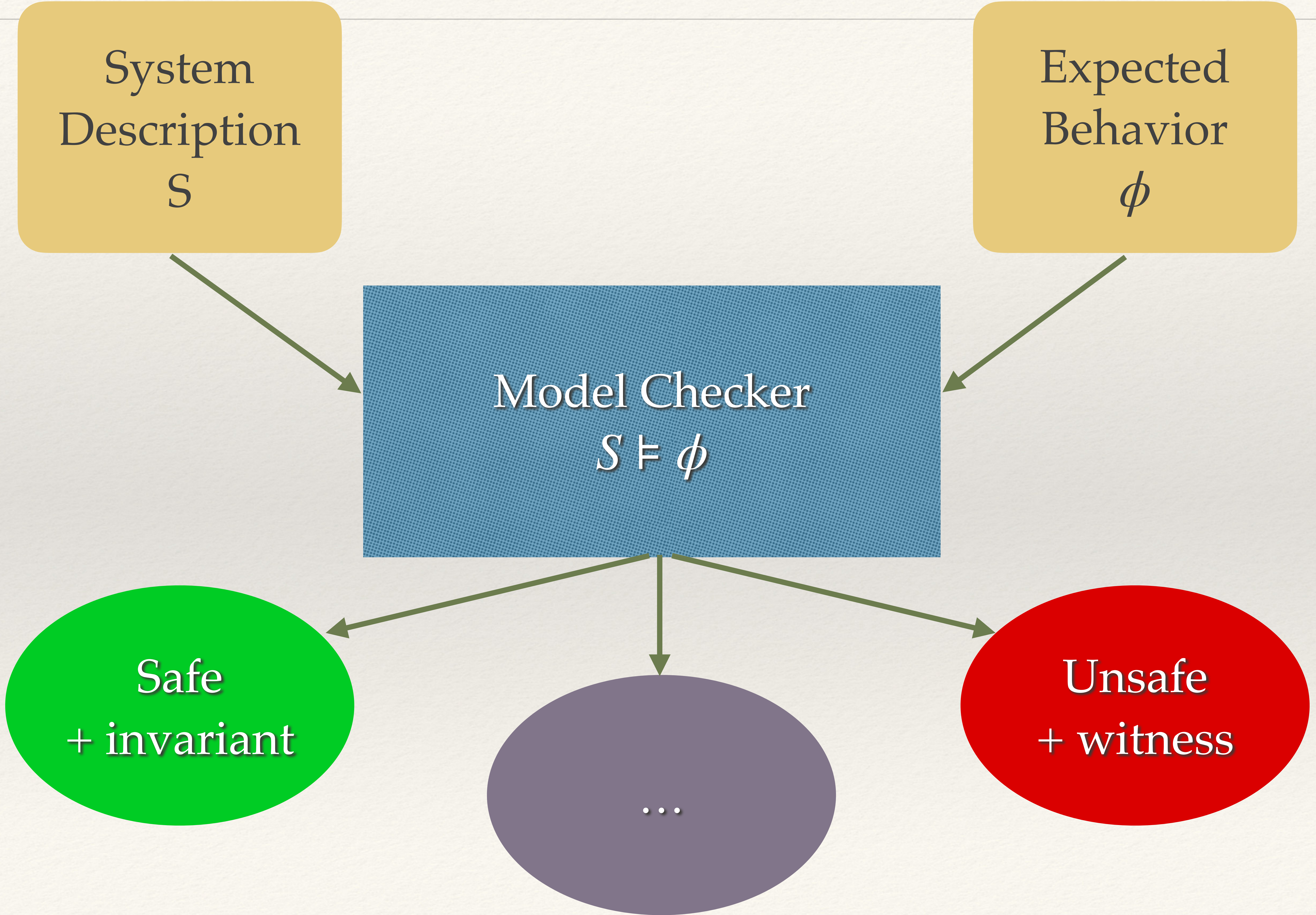
- Expensive or safety-critical failures
- Principled, exhaustive coverage



Model Checking



Model Checking



Some Directions for Model Checking

- Lift to Satisfiability Modulo Theories (SMT)
- New model checking algorithms
- Automatic / manual abstraction refinement approaches

Why SMT?

- Maintains structure of problem
- Abstractions with dedicated reasoning (e.g. arrays for memories)
- Quantified reasoning

- Two directions:
  - Faster SMT solvers
  - Better use of SMT solvers

Pono

- Solver-agnostic SMT-based Model Checker

- "Pono": right, correct, moral

Solver-agnostic Model Checker

- Solvers have different strengths
- New developments every year (showcased at SMT-COMP)
- Supporting multiple solvers good for:
  - Portfolio approaches
  - Utilizing union of supported features/strengths

Pono in AHA

- Property Checking + integration with Fault
- Lake mapping (up next!)
- AQED
- Future Uses:
  - Counter abstraction
  - SyGuS for rewriting
  - …and more!

High Level Goals of Pono

- Performant
- Adaptable
- Extensible

Performant

- Competitive implementations of standard model checking algorithms
  - BMC
  - BMC + simple path
  - K-Induction
  - Interpolant-based
  - IC3 (in progress)

Performant

- Favorable performance compared to CoSA
- Competed as "Cosa2" in Hardware Model Checking Competition 2019

sponsors

FWF Der Wissenschaftsfonds.

Results

In the SINGLE bit-vector track the top three places are:

1. **AVR**
   Aman Goel, Karem Sakallah (University of Michigan)
2. **CoSA2**
   Makai Mann, Ahmed Irfan, Florian Lonsing, Clark Barrett (Stanford University)
3. **CoNPS-btormc-THP**
   Norbert Manthey (hobbyist, former postdoc @ TU Dresden)

In the SINGLE bit-vector+array track the top three places are:

1. **CoSA2**
   Makai Mann, Ahmed Irfan, Florian Lonsing, Clark Barrett (Stanford University)
2. **AVR**
   Aman Goel, Karem Sakallah (University of Michigan)
3. **CoNPS-btormc-THP**
   Norbert Manthey (hobbyist, former postdoc @ TU Dresden)

Oski Award

CoSA2 for solving the largest number of benchmarks overall.

Adaptable

- Limitations of a black box
  - Translation step very important
  - Not always easily reducible to invariant checking
- Integrated verification — not a new idea, but hard to do right in practice
- Flexible API for solving diverse problems

Extensible

- Adaptability for users
- Extensibility for developers

- Infrastructure
  - Open-source and **simple**
  - Serve as a research platform for experts

```cpp
namespace pono {

Bmc::Bmc(const Property & p, SmtSolver & solver) : super(p, solver)
{
  initialize();
}

Bmc::Bmc(const PonoOptions & opt, const Property & p, smt::SmtSolver & solver)
    : super(opt, p, solver)
{
  initialize();
}

Bmc::~Bmc() {}

void Bmc::initialize()
{
  super::initialize();
  solver_->assert_formula(unroller_.at_time(ts_.init(), 0));
}

ProverResult Bmc::check_until(int k)
{
  for (int i = 0; i <= k; ++i) {
    if (!step(i)) {
      return ProverResult::FALSE;
    }
  }
  return ProverResult::UNKNOWN;
}

bool Bmc::step(int i)
{
  if (i <= reached_k_) {
    return true;
  }

  bool res = true;
  if (i > 0) {
    solver_->assert_formula(unroller_.at_time(ts_.trans(), i - 1));
  }

  solver_->push();
  logger.log(1, "Checking bmc at bound: {}", i);
  solver_->assert_formula(unroller_.at_time(bad_, i));
  Result r = solver_->check_sat();
  if (r.is_sat()) {
    res = false;
  } else {
    solver_->pop();
  }

  ++reached_k_;

  return res;
}
```

Demo

- Checking invariant of memory with a predicate over all stored data

- SMT abstractions
  - Represent memory with an array
  - Quantifiers
  - Uninterpreted function to represent an arbitrary predicate
  - (Could also abstract index using unbounded integers)

Next Up: SMT Improvements