Invited talk for FORTE/PSTV
9-10:30 on 11 Oct 2000, Pisa, Italy

# From Refutation to Verification

John Rushby

Computer Science Laboratory
SRI International
Menlo Park, California, USA

## Formal Methods in Practice

- Formal methods have achieved a modest degree of acceptance in some areas
  - E.g., hardware, protocols
- But mainly for purposes of refutation
  - That is, looking for errors
  - E.g., debugging, testing
- Verification is much less practiced
  - That is, showing the absence of errors
- Why is this? What can be done about it?

## Refutation

**Technology:** algorithmic, mostly model checking (and testing derived from model checking), some typechecking

**Human interaction:**

- Building and tweaking models; stating properties
- Turning knobs and dials
- Interpreting failure (bug, or weak model?)
    - Counterexamples help greatly

Resembles design activity

**Payoff:** discovery of high value bugs

- Can estimate the cost of each bug found
- And can directly compare with other technologies

**Limitation:** when to stop?

- Lack of refutation is not the same as verification

## Verification

**Technology**: mostly theorem proving

**Human interaction**:

- Building models; stating properties
- Guiding the theorem prover
  - And strengthening invariants
- Interpreting failure (bug, or bad proof?)

Focuses on proof, and idiosyncrasies of prover

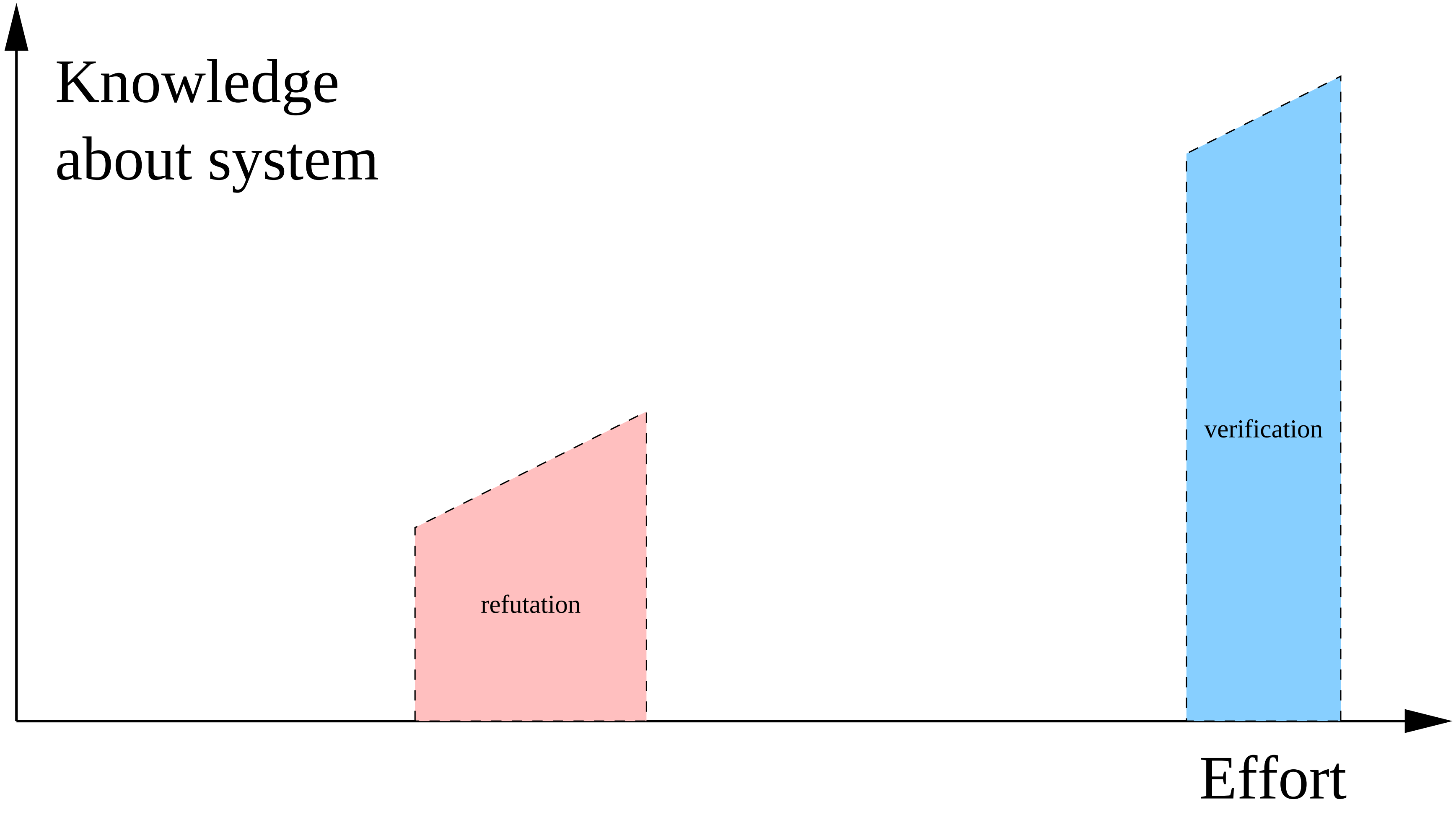
**Payoff**: definitive (product-based) assurance...with caveats

- May also find subtle bugs

**Limitation**: all or nothing

- And correctness is not even a goal in many industries

## Refutation and Verification



## Summarizing

- Refut'n can be cost-effective, but doesn't get you to verif'n
  - Interaction concerns the model, the technology is automated, it resembles familiar activities
  - It is acceptable to practitioners

  Challenge: why cannot the technology of refutation (particularly model checking) be used for verification?
- Verif'n has high potential payoff, but few interm'd benefits
  - Interaction concerns the proof and the prover, technology is not automated, intimidating
  - It is not acceptable to practitioners

  Challenge: why cannot theorem proving be made automatic?
- Overall challenges: why cannot model checking and theorem proving work together? And why cannot we move smoothly from refutation to verification?

## Our Context: Mathematics in Engineering

- Engineers build mathematical models of a design, its environment, and requirements
- Use calculation to establish that the design in the context of the environment satisfies the requirements
- Only useful when mechanized (cf. CFD)
- Used in the design loop (exploration, debugging) and also for verification
  - Model, calculate, interpret, repeat
- Need to be sure that design is implemented correctly, and that models of environment and requirements are "correct"
  - These concerns are separate from verification

## Formal Methods

- Same idea
- The applied math of Computer Science is formal logic
- So the models are formal descriptions in some logical system:
  $assumptions + design + environment \vdash requirements$
  (model checking replaces $\vdash$ by $\models$)
- And calculation is mechanized by automated deduction: theorem proving, model checking, etc.
- Formal calculations can cover all modeled behaviors, even if numerous or infinite (the power of symbolic reasoning)
  - Whereas complete testing is infeasible
  - And incomplete testing is unreliable

  This advantage of calculation over testing is a singular property of discrete systems

## Formal Methods for Assurance and Certification

- Want highly accurate models, so that formal calculations support strong claims—i.e., verification
- Then, using formal calculations, some activities that are traditionally performed by reviews
  - Processes that depend on human judgment and consensus

  can be replaced or supplemented by analyses
  - Processes that can be repeated and checked by others, and potentially so by machine
- Language from DO-178B/ED-12B (guidelines for certification of airborne software)

## However...

- Formal calculations
  - Are undecidable in general, so need human guidance
  - And even decidable problems have much greater computational complexity than mechanizing continuous mathematics
- So, must often trade off accuracy or completeness of the model for tractability and automation of calculation
  - E.g., aggressive downscaling to allow model checking

## So The Difficulty In Model Checking For Verification Is...

- The models (and properties) have to be simplified to make them tractable to fully automated analysis
  - This approach works for refutation
  - 'Cos experience indicates we learn more (find more bugs) by exploring all behaviors of a simplified model than by probing just some of the behaviors of the real thing (as with testing or simulation)
- But simplified models may not be fully accurate with respect to the property of interest
  - And that's why they cannot be used for verification
- Idea 1: construct simplified models that *are* accurate
- Idea 2: find fully automated methods that require less brutal simplification of the model

## And The Difficulty With Theorem Proving Is. . .

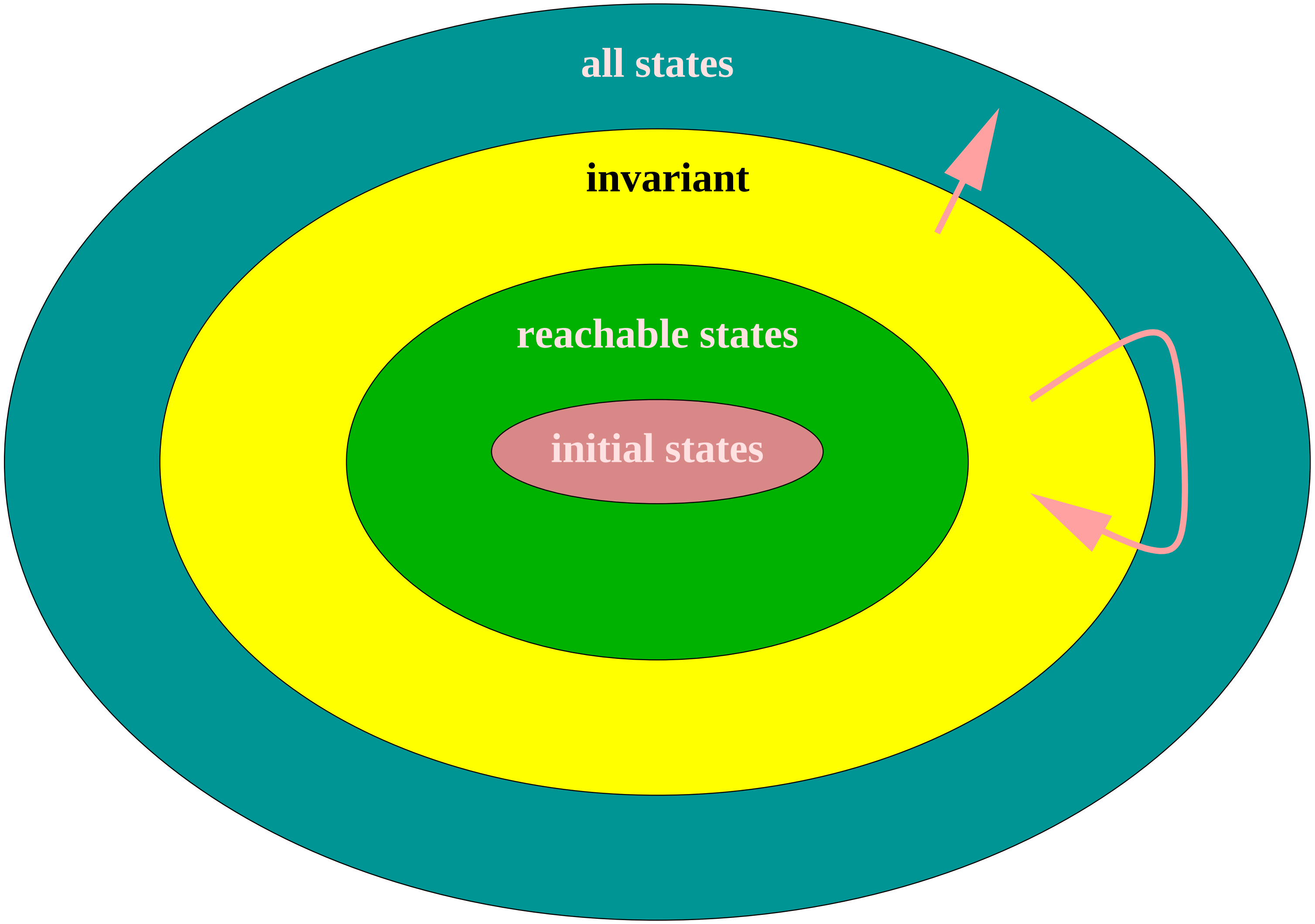
- Theorem proving can handle accurate models, but requires interactive human guidance
  - Interactive theorem proving is a waste of human talent
- The general theorem proving problem is undecidable
  - So full automation requires heuristics
  - Which will sometimes fail
- Classical verification poses correctness as a single "big theorem"
  - So failure to prove it (when true) is catastrophic
- Idea 3: "failure-tolerant" theorem proving
  - Lots of small theorems instead of one big one
- Idea 4: increase the power of non-heuristic automation

## Except The Real Difficulty With Theorem Proving Is. . .

- Not the tactical problems of guiding the theorem prover
- But the strategic problem of finding an inductive invariant
- To establish an invariant or safety property (one true of all reachable states) by theorem proving, we invent another property that implies the one of interest and that is inductive
  - Includes all the initial states
  - Is closed on the transitions

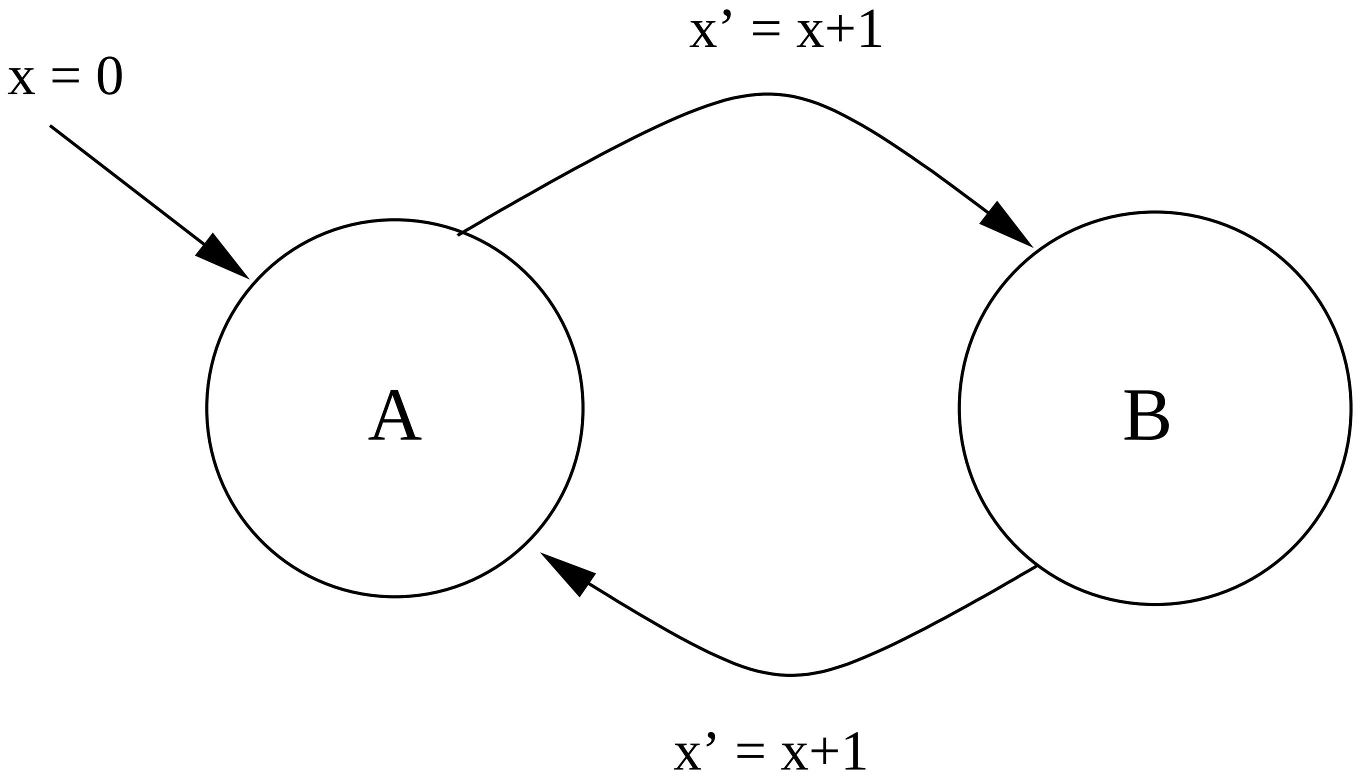
  The reachable states are the smallest set that is inductive
- Trouble is, naturally stated invariants are seldom inductive
  - The second condition is violated
- Need to make them smaller (stronger) to exclude the states that take you outside the invariant

# Noninductive Invariants In Pictures



## Example

Prove that x is even when control is at A

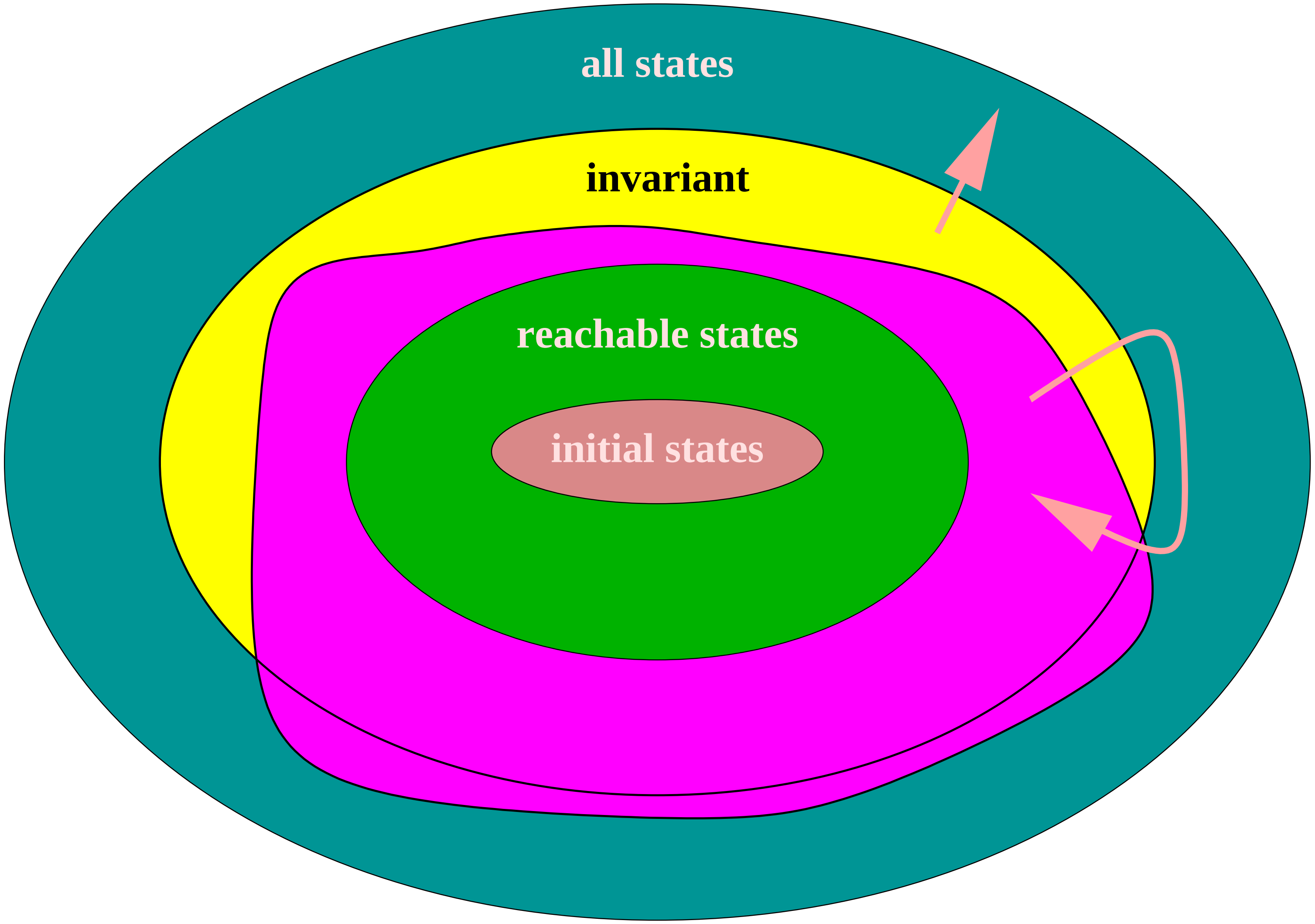



Need to know it's odd when control is at B

## Strengthening Invariants To Make Them Inductive

- Postulate a new invariant that excludes the states (so far discovered) that take you outside the desired invariant
- Show that the conjunction of the new and the desired invariant is inductive
- Iterate until success or exasperation
- Process can be made systematic, but it always tedious
- Bounded retransmission protocol required 57 such iterations
  - Took a couple of months to complete (Havelund and Shankar)

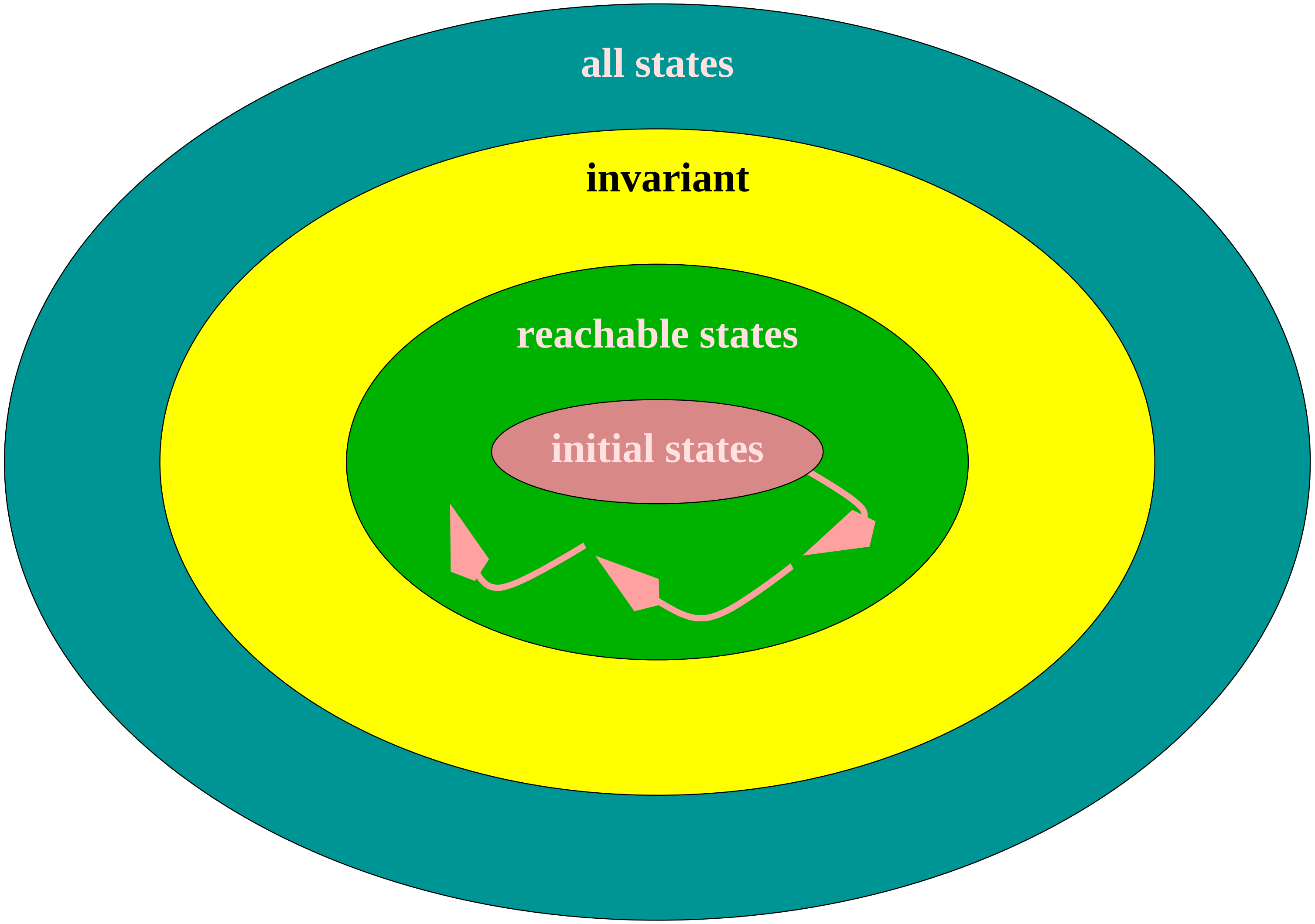
# Inductive Invariants In Pictures



## How Does A Model Checker Do It?

- Model checkers can verify an invariant by enumerating the reachable states and checking the property at each one
  - Only works when the number of reachable states is finite (and relatively small)
- Can also work backwards from bad states and check you never reach an initial state

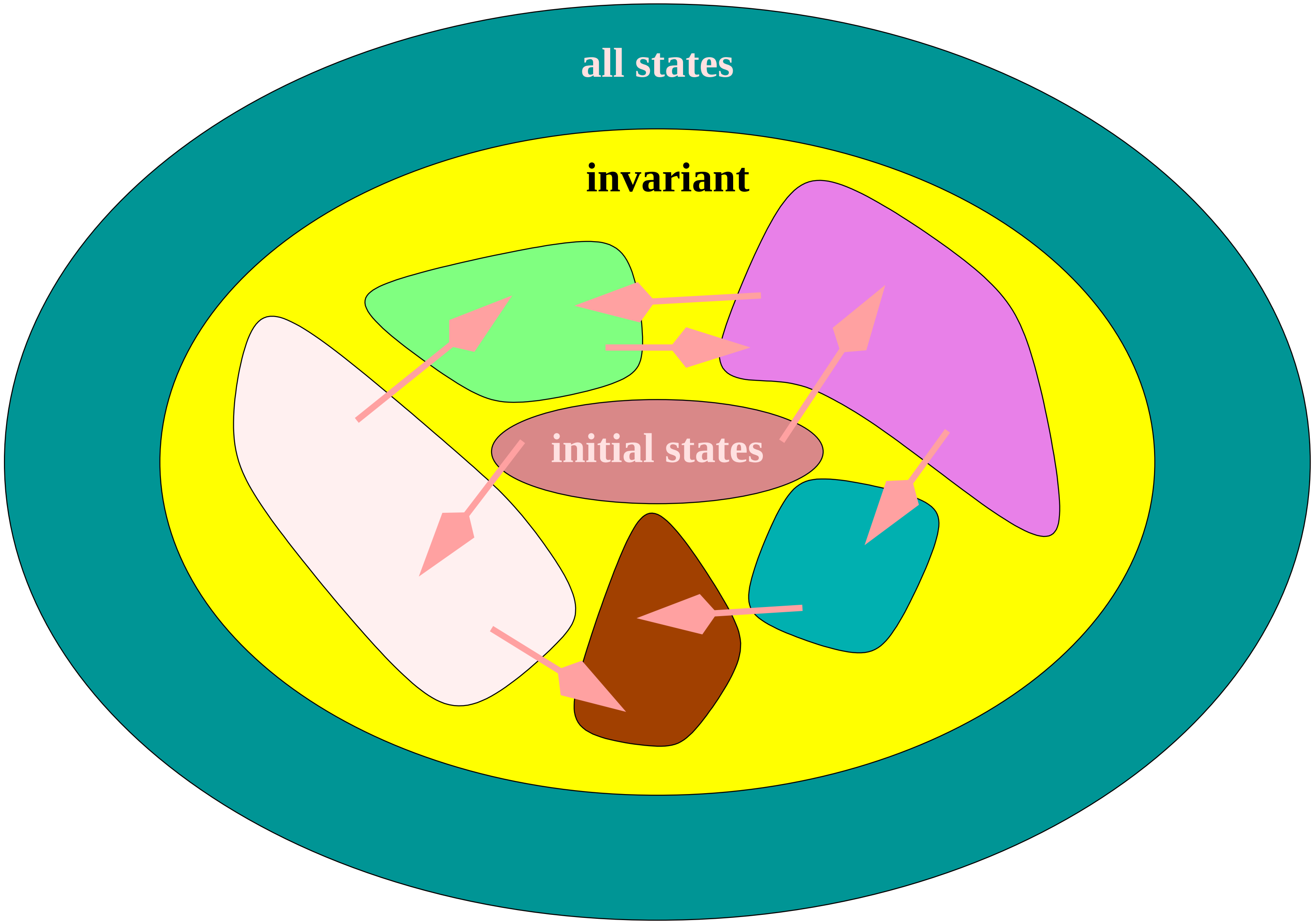
# Direct Enumeration Of Reachable States



## Can A Theorem Prover Emulate That?

- With theorem proving, we cannot enumerate the reachable states individually
  - Too many of them
- So try grouping them into sets called configurations and do a forward reachability calculation on the configurations
  - Mechanized by symbolic evaluation
- We are done if the configurations are closed under the transition relation, and each reachable configuration is a subset of the desired invariant
  - Equivalently, if each configuration is characterized by a property that implies the desired invariant
- This is the method of disjunctive invariants
- Sometimes effective (see next talk)

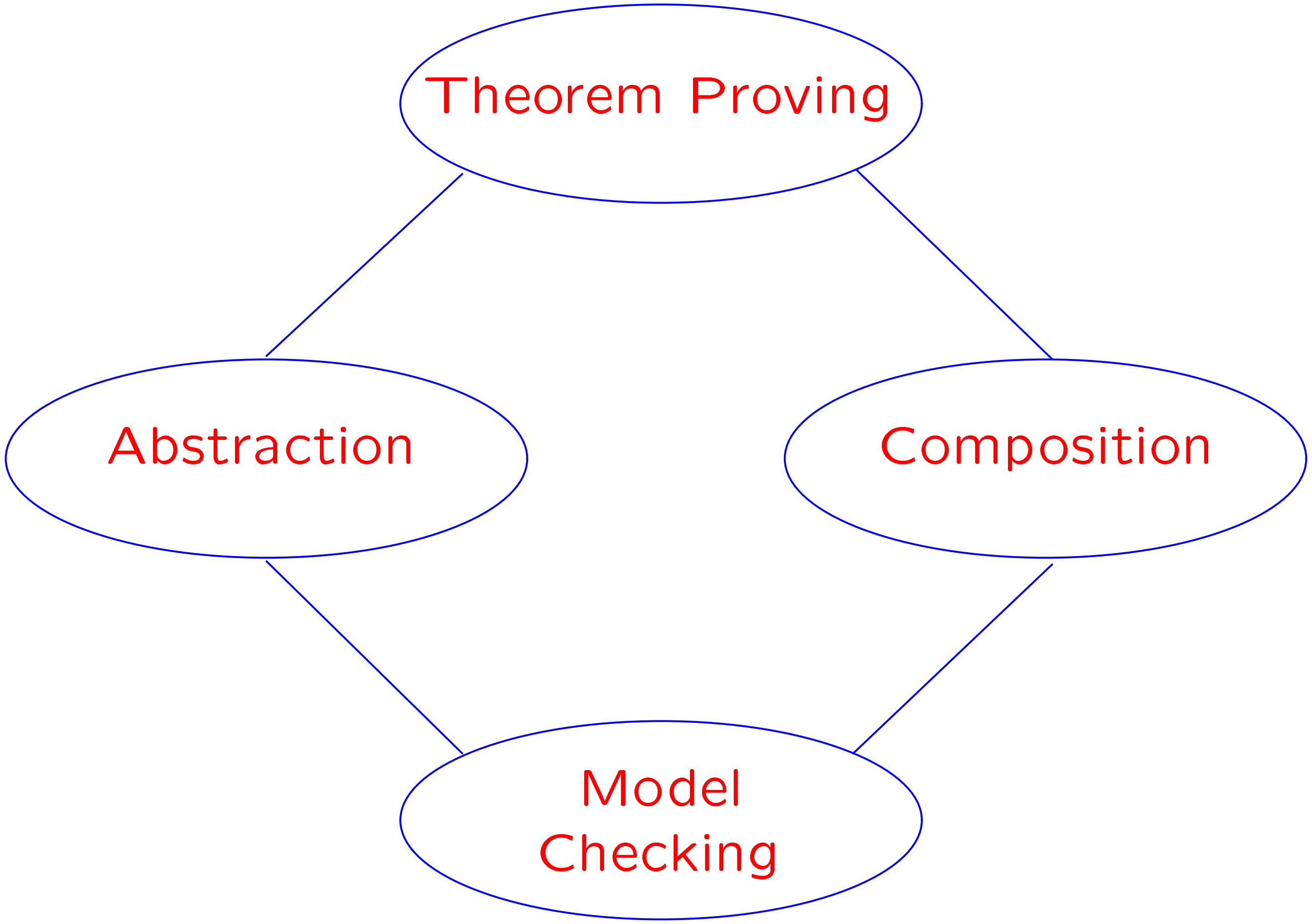
# Disjunctive Invariants In Pictures



## But Why Emulate A Model Checker?

- Why not use one directly?
- Construct a finite state abstraction of the system
- Prove that it is property preserving
- And model check it
- Problem is: it's quite hard to justify the abstraction
  - May require auxiliary invariants
    - ⋆ 45 of the 57 invariants required for BRP
  - Back where we started
- But it's the right direction
- Idea 5: calculate the abstracted system
  - Using automated abstraction

## Abstraction is a Bridge Between Deductive and Algorithmic Methods



## Abstraction

- Given a transition system $G$ on $S$ and property $P$, a property-preserving abstraction yields a transition system $\hat{G}$ on $\hat{S}$ and property $\hat{P}$ such that

$$\hat{G} \models \hat{P} \Rightarrow G \models P$$

- Strongly property preserving abstraction:

$$\hat{G} \models \hat{P} \Leftrightarrow G \models P$$

- A good abstraction typically (for universal properties) introduces nondeterminism while preserving the property

- Remaining problem: Construction of reasonably precise $\hat{G}$ and $\hat{P}$ given $G$ and $P$

## Data Abstraction [Cousot & Cousot]

- Replace concrete variable $x$ over datatype $C$ by an abstract variable $x'$ over datatype $A$ through a mapping $h : [C \rightarrow A]$.
- Examples (Bandera system [Dwyer et al.]): Parity, $mod\ N$, zero-nonzero, intervals, cardinalities, {0, 1, many}, {empty, nonempty}
- Given $f : [C \rightarrow C]$, construct $\hat{f} : [A \rightarrow set[A]]$:
(observe how data abstraction introduces nondeterminism)

$$b \in \hat{f}(a) \Leftrightarrow \exists x : a = h(x) \wedge b = h(f(x))$$

$$b \notin \hat{f}(a) \Leftrightarrow \ \vdash \forall x : a = h(x) \Rightarrow b \neq h(f(x))$$

- Theorem-proving failure affects accuracy, not soundness
- In example: abstract naturals to {0, 1}; addition becomes mod 2 (theorem proving still cannot solve this directly)

## Predicate Abstraction [Graf-Saïdi]

- Abstracts out relations between variables, e.g., $x < y$, $x + y = z$
- Variables ranging over infinite datatypes can be replaced by Boolean variables representing the predicates on those variables
- Predicates can be extracted from guards, assignments, and the property of interest
- Guessing predicates is easier than invariant strengthening (and is also more general [Rusu & Singerman'99])
- Predicate abstraction available in PVS as (abstract-and-mc)

## Construction of Predicate Abstractions

- Given $\phi : [S \rightarrow \hat{S}]$ induced by the abstracted predicates, construct $\hat{G}$ by

$$\hat{G}(\hat{s}_1, \hat{s}_2) \Leftrightarrow \exists s_1, s_2 : \hat{s}_1 = \phi(s_1) \wedge \hat{s}_2 = \phi(s_2) \wedge G(s_1, s_2)$$

$$\neg\hat{G}(\hat{s}_1, \hat{s}_2) \Leftrightarrow \vdash \forall s_1, s_2 : \hat{s}_1 \neq \phi(s_1) \vee \hat{s}_2 \neq \phi(s_2) \vee \neg G(s_1, s_2)$$

- Theorem-proving failure affects accuracy, not soundness
- There is another method (exponentially more efficient) [Saïdi & Shankar]
- In example: abstract on predicate even?$(x)$

## Automated Abstraction

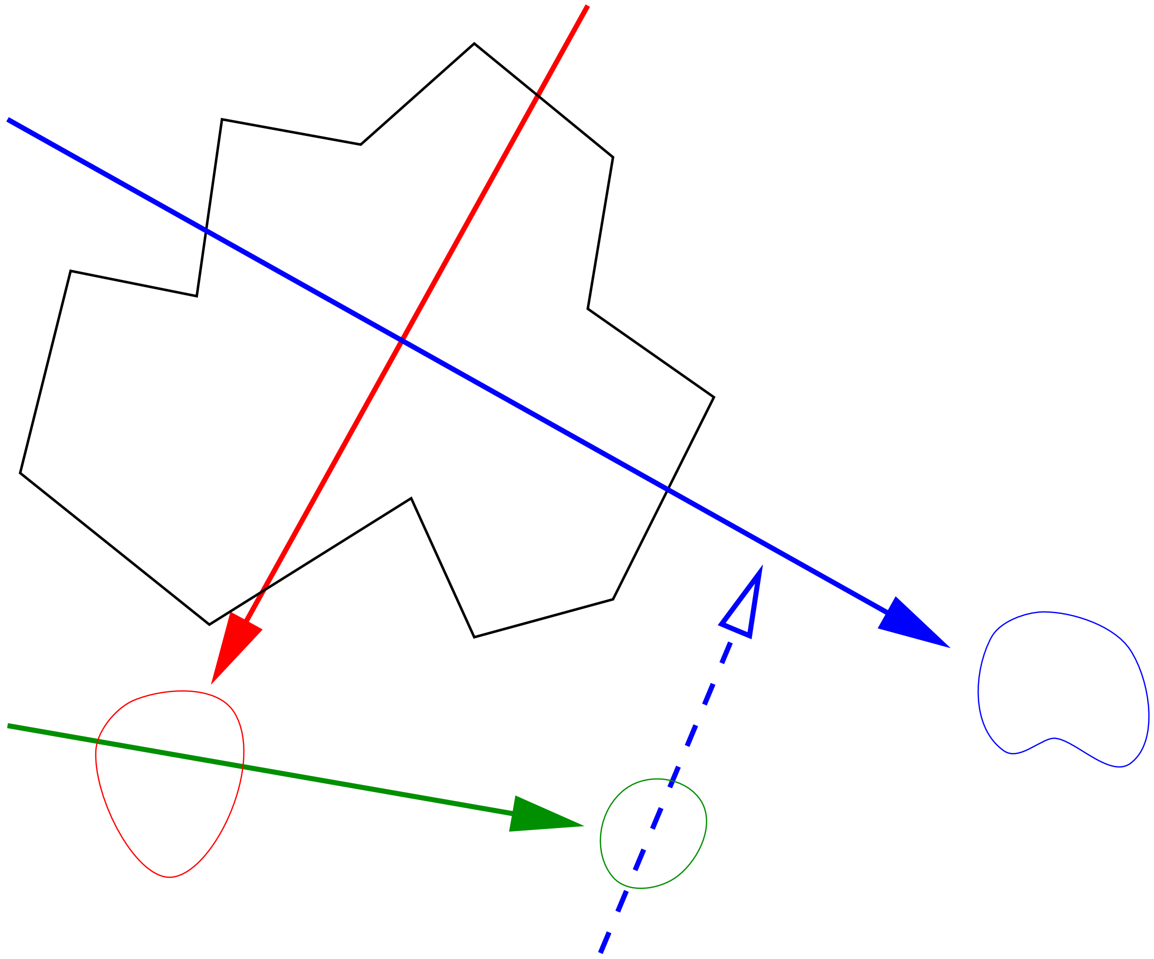
- Can often construct a simplified model that is faithful to the original (for a given property of interest)
  - The reduced model can by analyzed by model checking
  - And failure to detect bugs does certify their absence
- These reduced models can be constructed automatically by mechanized data or predicate abstraction
  - The construction is done by trying to prove lots of little theorems
    - ⋆ If a proof fails, abstracted model will be more conservative, but often still good enough
- But the construction often requires auxiliary invariants

## The Bridge Goes In Both Directions

- Model checkers often calculate the reachable stateset
  - Which is the strongest invariant

  And then throw it away
- The concretization of the reachable states of an abstraction is an invariant of the concrete system
  - And often a strong one
- So modify a model checker to return the reachable states as a formula that a theorem prover can manipulate
- Has been done (by Sergey Berezin) for CMU SMV and is used in InVeSt [Bensalem, Lakhnech & Owre]

# Integrated, Iterated Analysis



## Even More Integrated, Iterated Analysis!

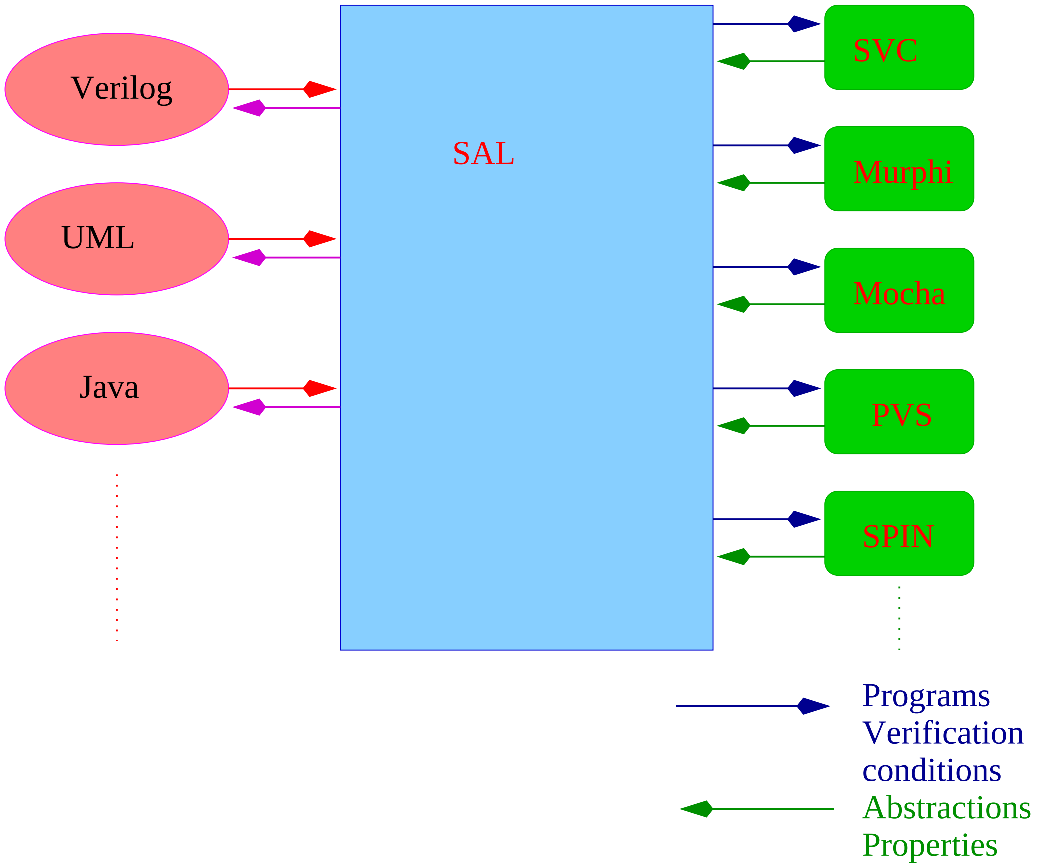
- (Approximations to) fixpoints of weakest preconditions or strongest postconditions also generate invariants and can strengthen those extracted from an abstraction
  - Mechanized by theorem proving
  - (Strongest postconditions are equivalent to symbolic simulation, which is independently useful)
- Counterexamples from failed model check help distinguish bugs from weak abstractions, and also help refine the abstraction
  - Suggest additional properties (invariants) that will help the theorem prover construct a tighter model
  - Suggest additional predicates on which to abstract

## Truly Integrated, Iterated Analysis!

- Recast the goal as one of calculating and accumulating properties about a design (symbolic analysis)
- Rather than just verifying or refuting a specific property
- Properties convey information and insight, and provide leverage to construct new abstractions
  - And hence more properties
- Requires restructuring of verification tools
  - So that many work together
  - And so that they return symbolic values and properties rather than just yes/no results of verifications
- This is what SAL is about: Symbolic Analysis Laboratory

## SAL Architecture

The SAL Language serves as a hub




Not all of these boxes are populated yet

## SAL Language

- A way to specify transition relations and their properties
- Developed in a loose collaboration with Stanford (David Dill), Berkeley (Tom Henzinger), and Verimag (Saddek Bensalem)
  - InVeSt has adopted the SAL language
- Has definitions, guarded commands, modules, synchronous and asynchronous composition
- External representations use XML

## Related Systems

- David Dill's treatment (Stanford)
  - Variant on predicate abstraction; can do garbage collector automatically in an hour or so, CAV '99
- Ken McMillan's treatment (Cadence)
  - Lightweight theorem proving whose subgoals are discharged by model checking; can do Tomosulo automatically, CAV 00
- Bandera (Matt Dwyer, John Hatcliffe, Kansas State)
  - Uses abstract interpretation, works on Java, ICSE 00
- SLAM (Tom Ball, Sriram Rajamani, Microsoft Research)
  - Boolean programs, SPIN 00
- Veritech (Shmuel Katz, Orna Grumberg, The Technion)
  - Another intermediate rep'n; eclectic front and backends

## Bringing The Ideas Together

- SAL et al realize Idea 5
  - Calculate the abstracted system
- Which creates the opportunity for Ideas 1 and 3
  - Construct simplified models that are accurate
  - "Failure-tolerant" theorem proving
- Then, by allowing unsound abstractions

$$\hat{G} \models \hat{P} \not\Rightarrow G \models P$$

  We can do refutation as well as verification

- By selecting abstractions (sound/unsound) and properties (little/big) we can fill in the space between refutation and verification

## Remaining Ideas

- Idea 4: increase the power of non-heuristic automation
  - Decision procedures
    - Combination of: propositional satisfiability, equality over uninterpreted function symbols with (linear) arithmetic, arrays, datatypes
  - Quantifier elimination for decidable fragment of the above

  We are making these available as ICS

- Others have made more powerful decision procedures available for some of these (e.g., SAT, SAT with equality)

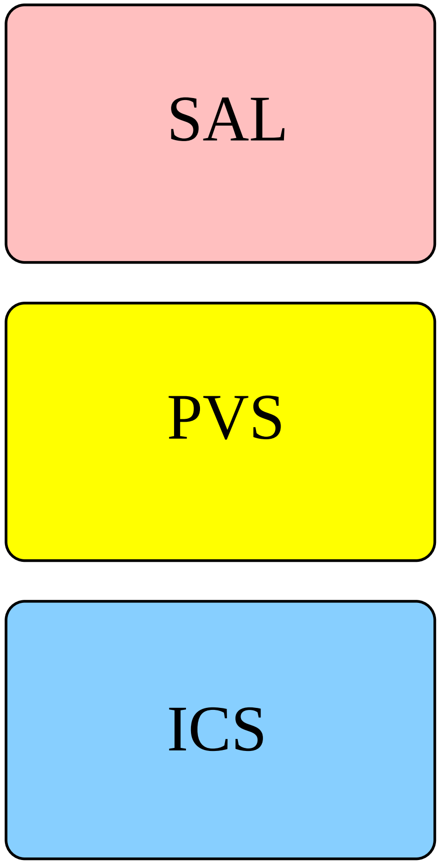
- And decision procedures for more powerful theories (e.g., Mona for WS1S)

## Remaining Ideas

Idea 2: find fully automated methods that require less brutal simplification of the model

- Finite-state model checking builds on BDDs
- Many other decision procedures can be extended to model checking
  - Lossy-Channel Systems (LCS)
  - WS1S
- So renewed interest in decision procedures contributes here

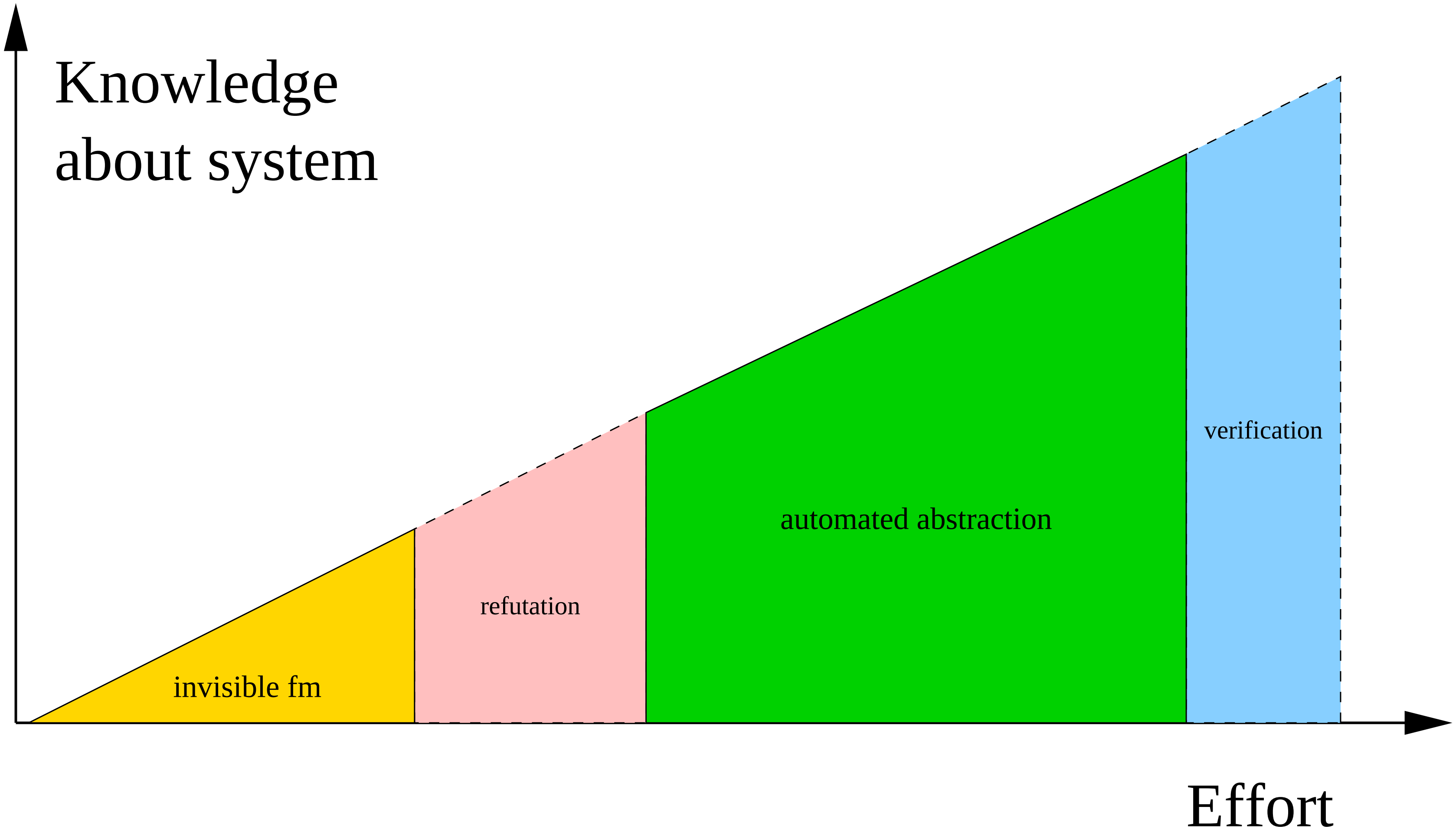
# What We Are building



## ICS: Standalone Decision Procedures

- ICS = Integrated Canonizer-Solver (= ICanSolve)
- Can be used in table-checkers, compilers (ESC) etc.
- Aim is to enable invisible formal methods
- Or Ubiquitous formal methods

## From Refutation To Verification



## Acknowledgments

- N. Shankar, Sam Owre, Harald Rueß, Hassen Saïdi
- Saddek Bensalem, Jean-Christophe Filliâtre, Klaus Havelund, Friedrich von Henke, Yassine Lakhnech, César Muñoz, Holger Pfeifer, Vlad Rusu, Eli Singerman, and many others