Formal verification of security protocols

Ansuman Banerjee

Indian Statistical Institute

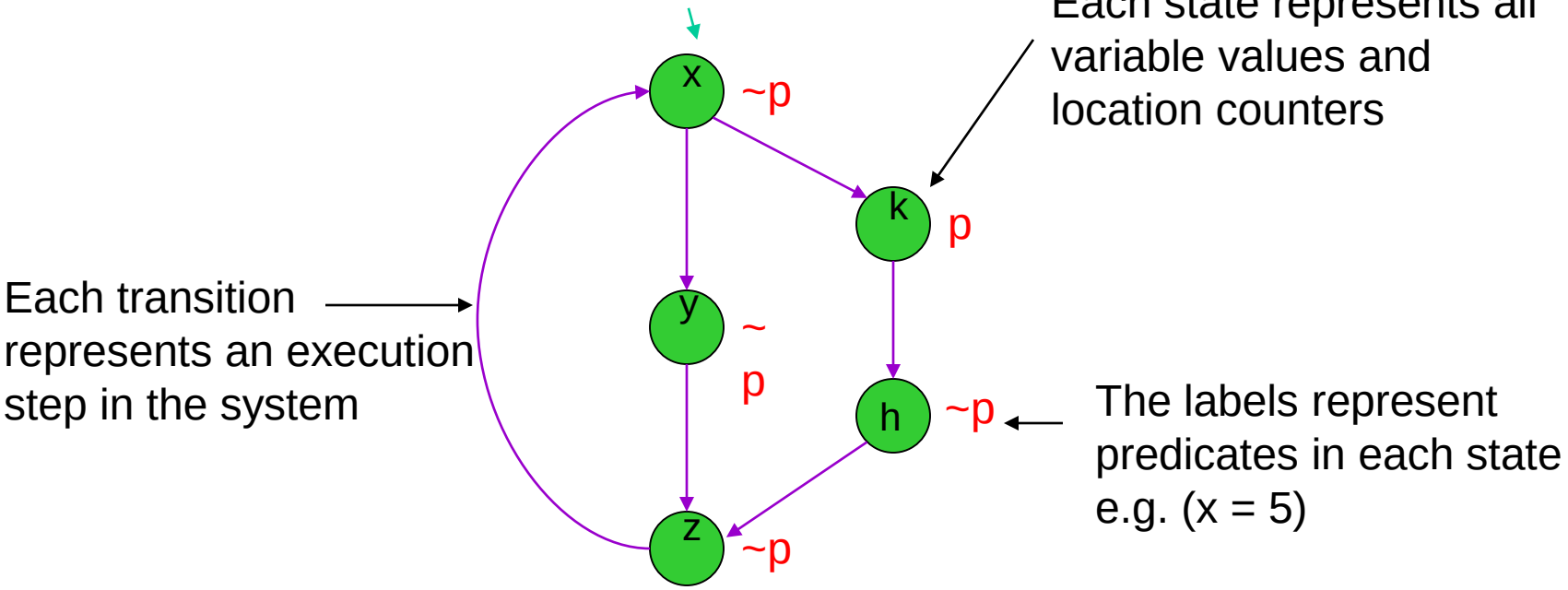
Outline

- Formal Verification: The basics
  - Model checking
- Formal verification: In the security context
  - Approaches to security protocol verification

Model Checking

- Check if a system satisfies a certain behavioral property:
  - Is the system deadlock free?
  - Whenever a packet is sent will it eventually be received?
- So it is like testing? No, major difference:
  - Looks at *all* possible behaviors of a system
- Automatic
- Questions:
  - How do we describe the system?
  - How do we express the properties?

Labeled State Graph: *Kripke Structure*

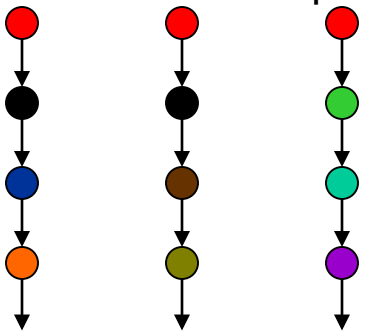


K = ({p,~p},{x,y,z,k,h},R,{x},L)

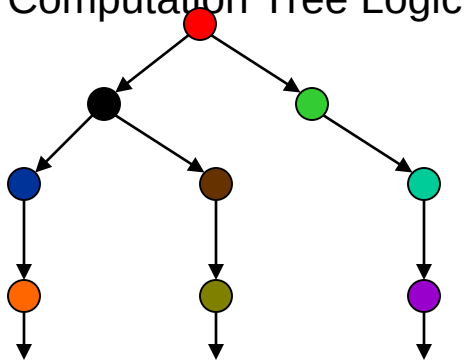
Property Specifications

- Temporal Logic
  - Express properties of event orderings in time
  - e.g. “Always” when a packet is sent it will “Eventually” be received

- Linear Time
  - Every moment has a unique successor
  - Infinite sequences (words)
  - Linear Time Temporal Logic (LTL)



- Branching Time
  - Every moment has several successors
  - Infinite tree
  - Computation Tree Logic (CTL)



Safety and Liveness

- Safety properties
  - Invariants, deadlocks, reachability, etc.
  - “something bad never happens”
- Liveness Properties
  - Fairness, response, etc.
  - “something good will eventually happen”

Model Checking

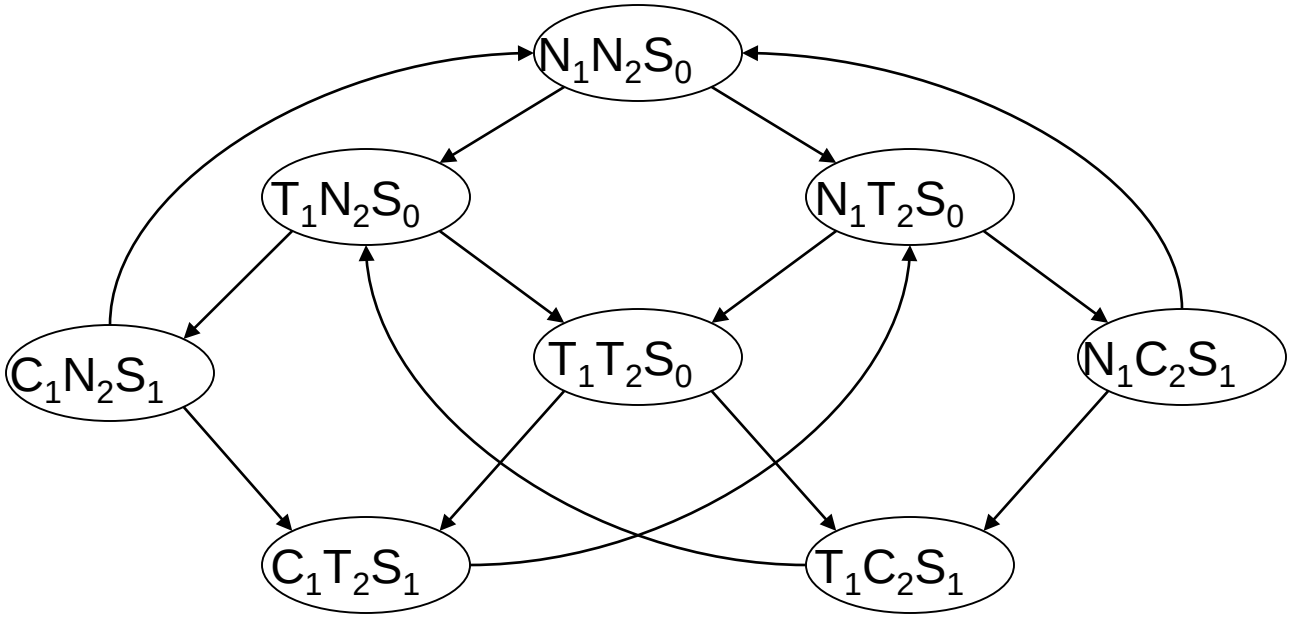
- Developed independently by **Clarke and Emerson** and by **Queille and Sifakis** in early 1980's.
- **Properties** are written in **propositional temporal logic.**
- Systems are modeled by **finite state machines**.
- Verification procedure is an **exhaustive search of the state space** of the design.
- Model checking **complements** testing/simulation.

Mutual Exclusion Example

- Two process mutual exclusion with shared semaphore
- Each process has three states
  - Non-critical (N)
  - Trying (T)
  - Critical (C)
- Semaphore can be available (S_0) or taken (S_1)
- Initially both processes are in the Non-critical state and the semaphore is available --- $N_1\, N_2\, S_0$

$$
\begin{array}{lll}
N_1 & \rightarrow & T_1 \\
T_1 \wedge S_0 & \rightarrow & C_1 \wedge S_1 \\
C_1 & \rightarrow & N_1 \wedge S_0
\end{array}
\quad \| \quad
\begin{array}{lll}
N_2 & \rightarrow & T_2 \\
T_2 \wedge S_0 & \rightarrow & C_2 \wedge S_1 \\
C_2 & \rightarrow & N_2 \wedge S_0
\end{array}
$$

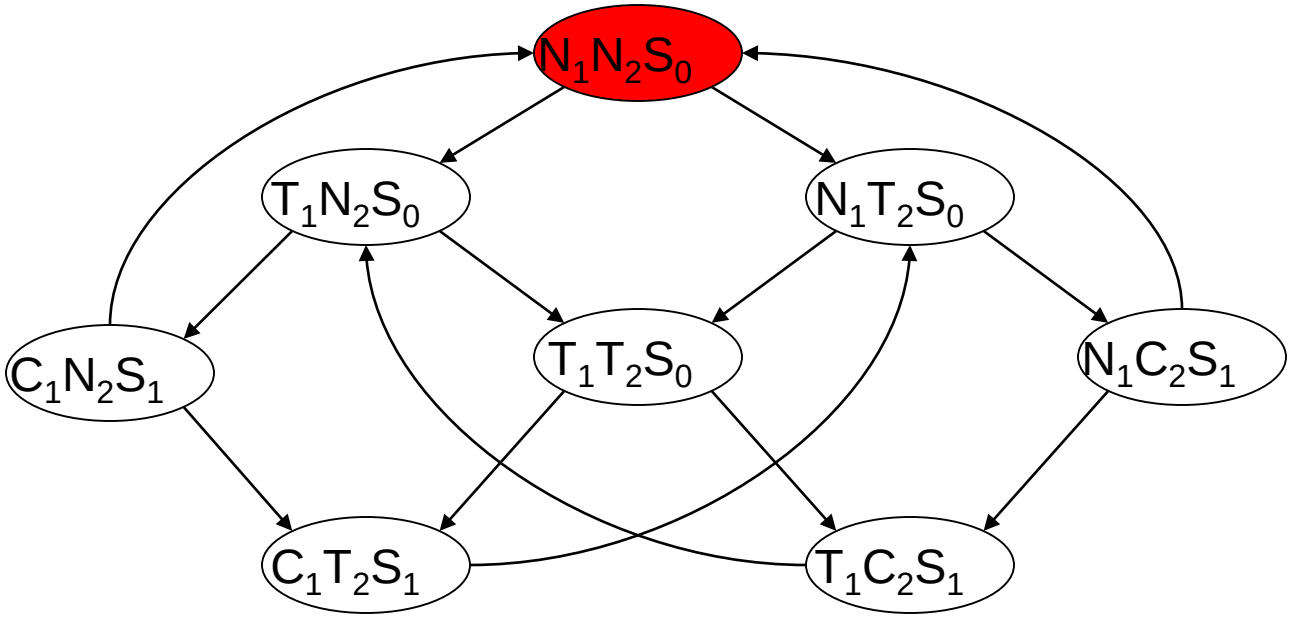
Mutual Exclusion Example



$K \models AG\ EF\ (N_1 \wedge N_2 \wedge S_0)$

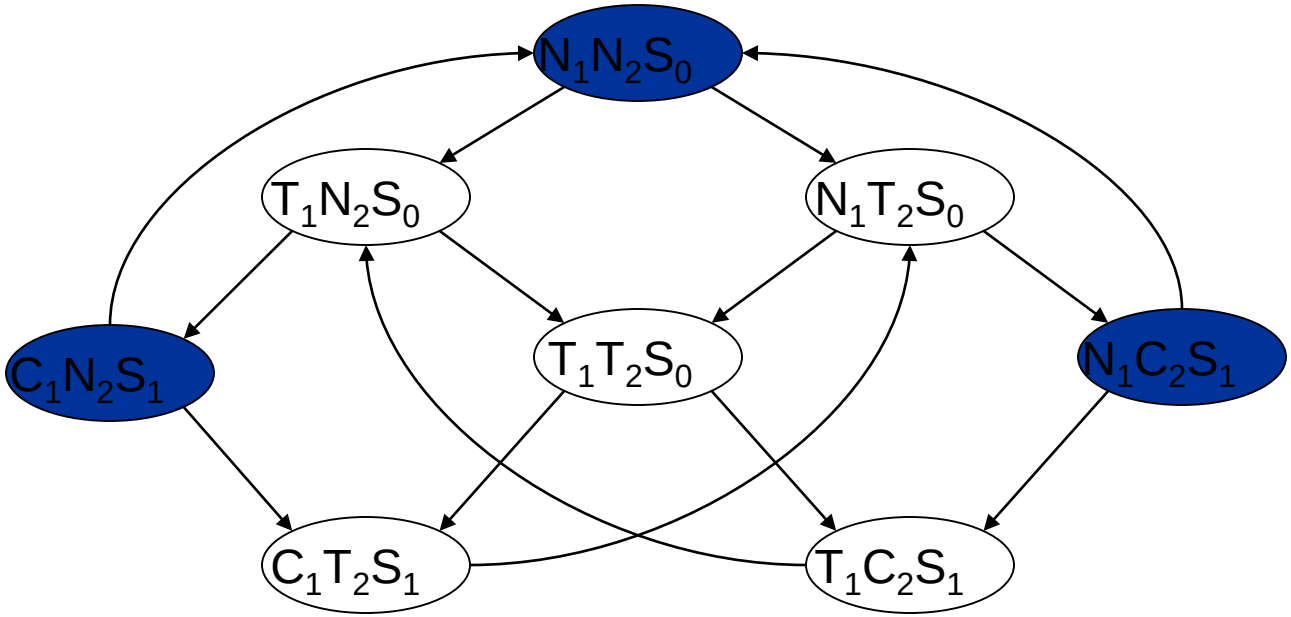
No matter where you are there is always a way to get to the initial state

Mutual Exclusion Example



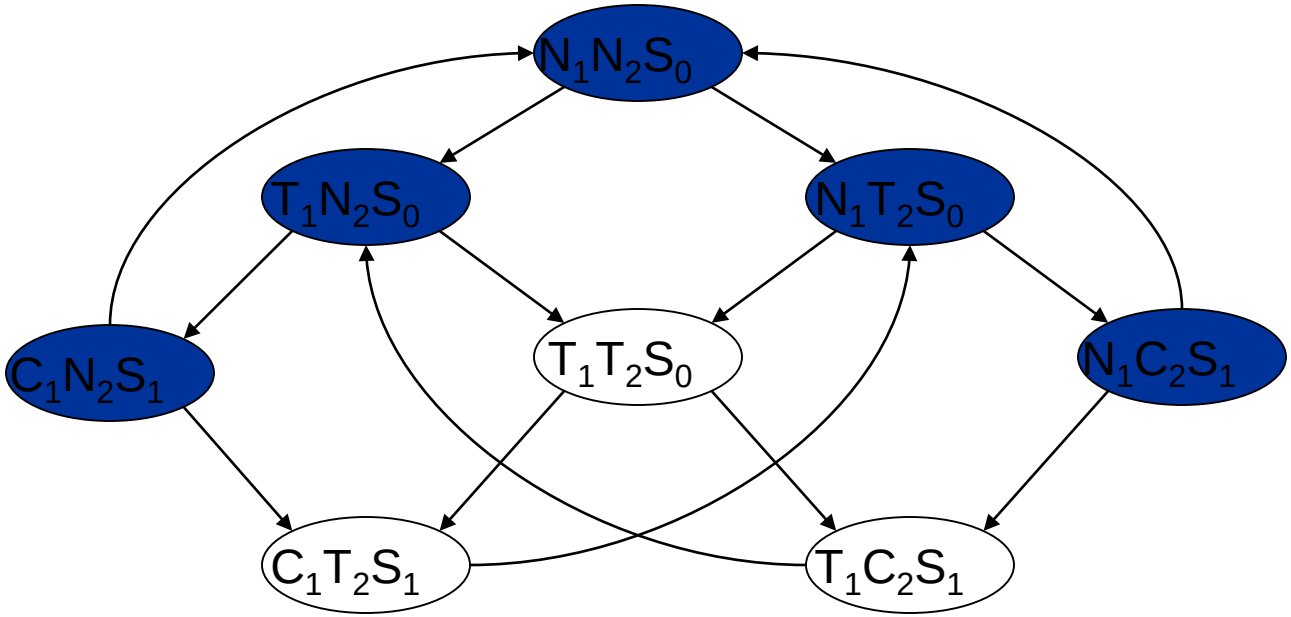
$K \models AG\ EF\ (N_1 \wedge N_2 \wedge S_0)$

Mutual Exclusion Example



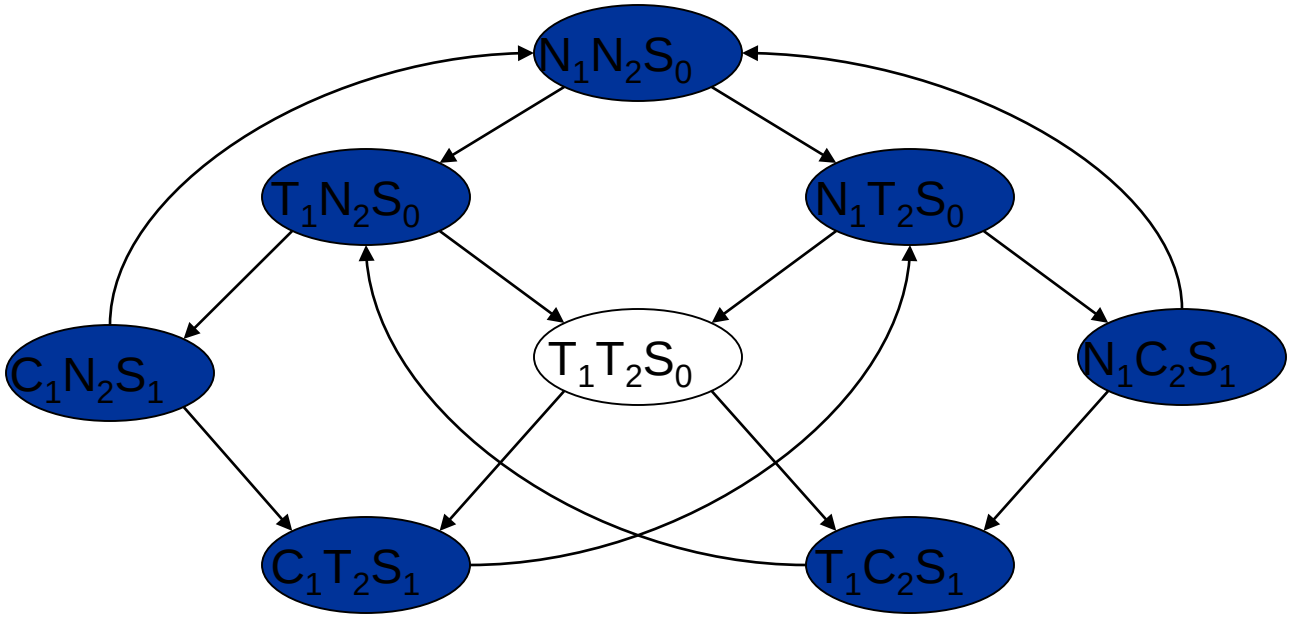
$K \models AG\ EF\ (N_1 \wedge N_2 \wedge S_0)$

Mutual Exclusion Example



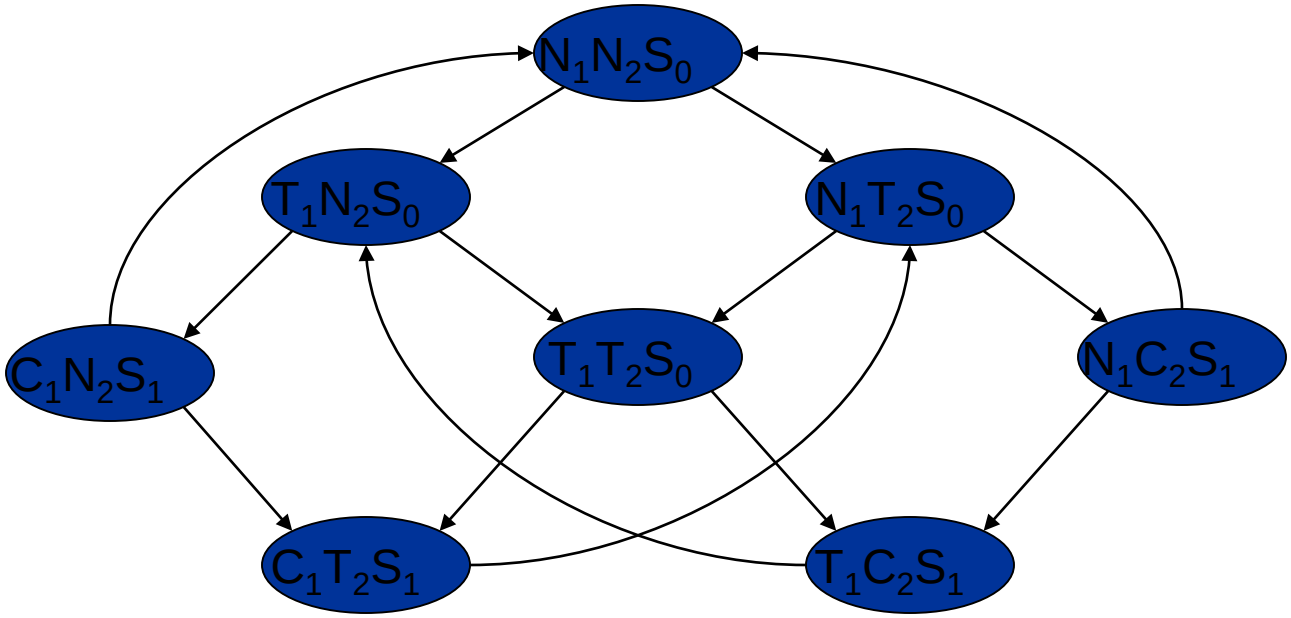
$K \models AG\ EF\ (N_1 \wedge N_2 \wedge S_0)$

Mutual Exclusion Example



$K \models AG\ EF\ (N_1 \wedge N_2 \wedge S_0)$

Mutual Exclusion Example



$K \models AG\ EF\ (N_1 \wedge N_2 \wedge S_0)$

Model Checking

- Given a Kripke structure *M = (S,R,L)* that represents a finite-state concurrent system and a temporal logic formula *f* expressing some desired specification, find the set of states in *S* that satisfy *f*:

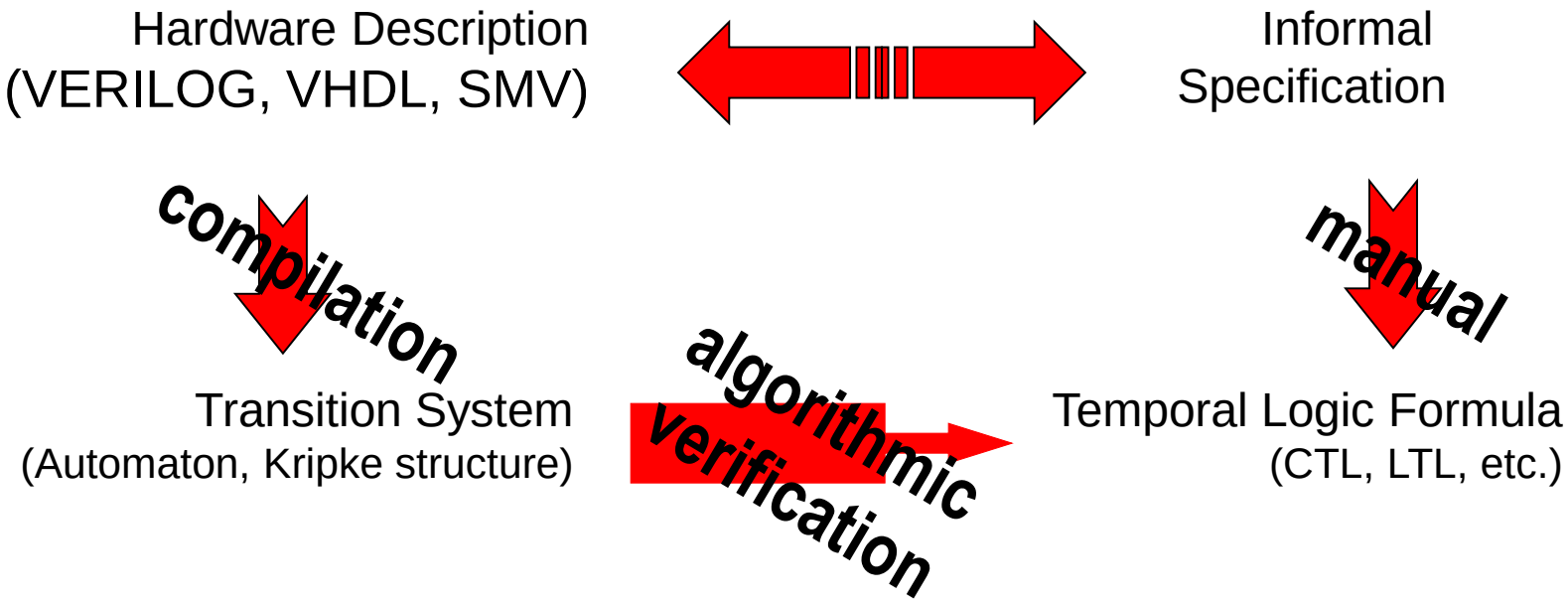
$$\{ s \in S \mid M,s \models f \}$$

- Normally, some states of the concurrent system are designated as initial states. The system satisfies the specification provided all the initial states are in the set. We often write: $M \models f$

Model Checking in the hardware world

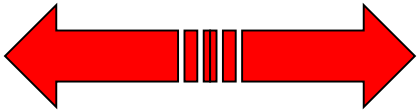
- Hardware is typically synchronous and regular, hence the transition relation can be encoded efficiently
  - Execution paths are typically very short
- The Intel Pentium bug, was the “disaster” that got model checking on the map in the hardware industry
  - What is it going to take in the software world?
- Intel, IBM, Motorola, etc. now employ hundreds of model checking experts

Model Checking

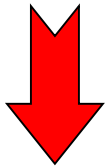


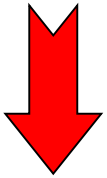
Counterexamples

Program or circuit



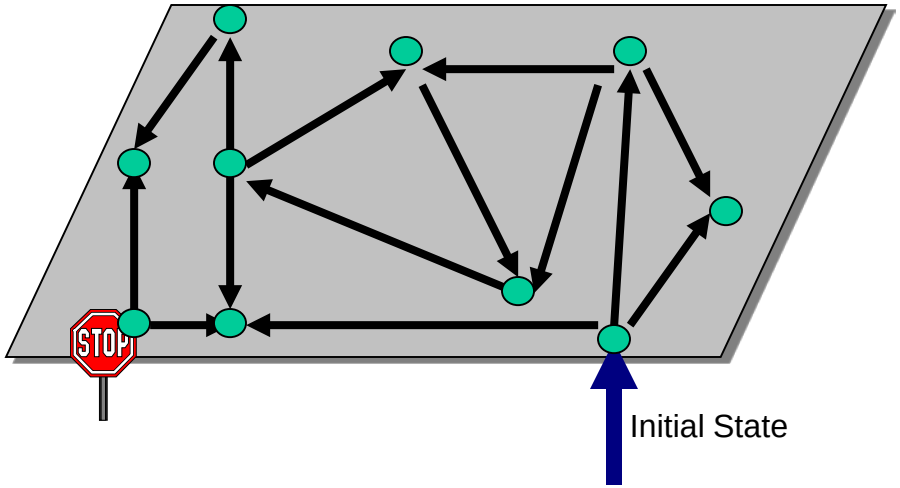
Informal Specification





Transition System

Temporal Logic Formula
(CTL, LTL, etc.)



Safety Property:

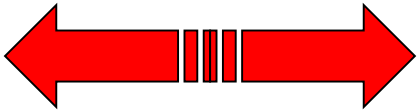
bad state 

unreachable:

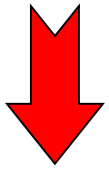
satisfied

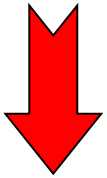
Counterexamples

Program or circuit



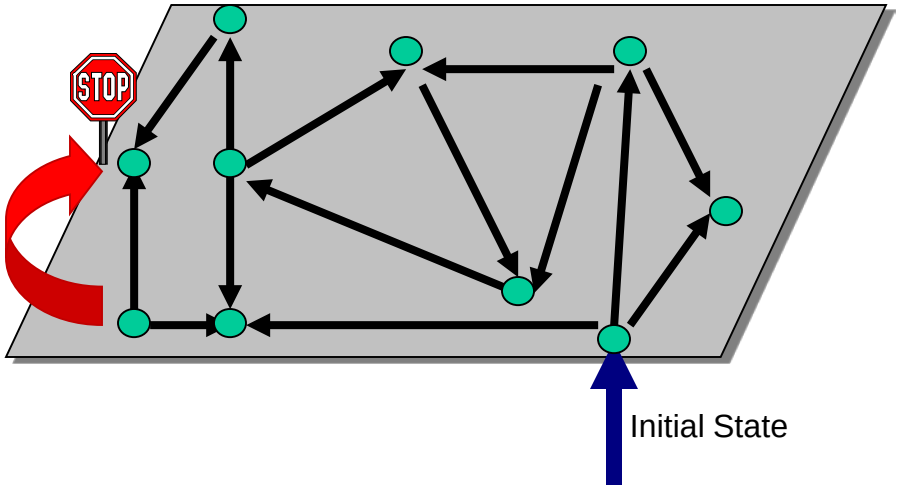
Informal Specification





Transition System

Temporal Logic Formula
(CTL, LTL, etc.)



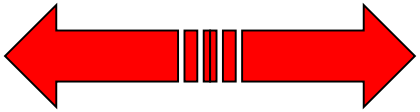
Safety Property:
bad state 

unreachable

Counterexample

Counterexamples

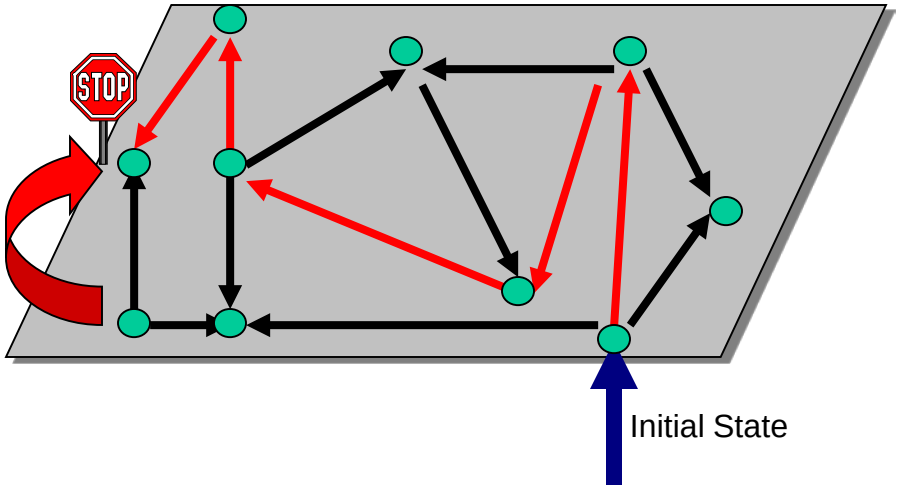
Program or circuit



Informal Specification

Transition System

Temporal Logic Formula
(CTL, LTL, etc.)



Safety Property:
bad state 

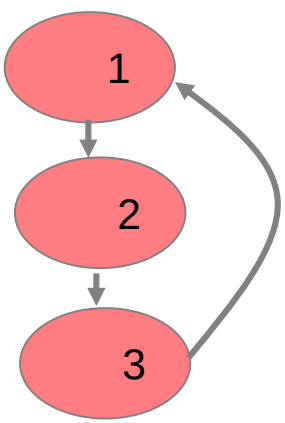
unreachable

Counterexample

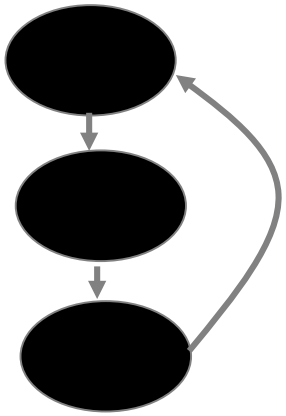
Advantages of Model Checking

- **No proofs!!!**
- **Fast (compared to other rigorous methods)**
- **Diagnostic counterexamples**
- **Logics can easily express many concurrency properties**

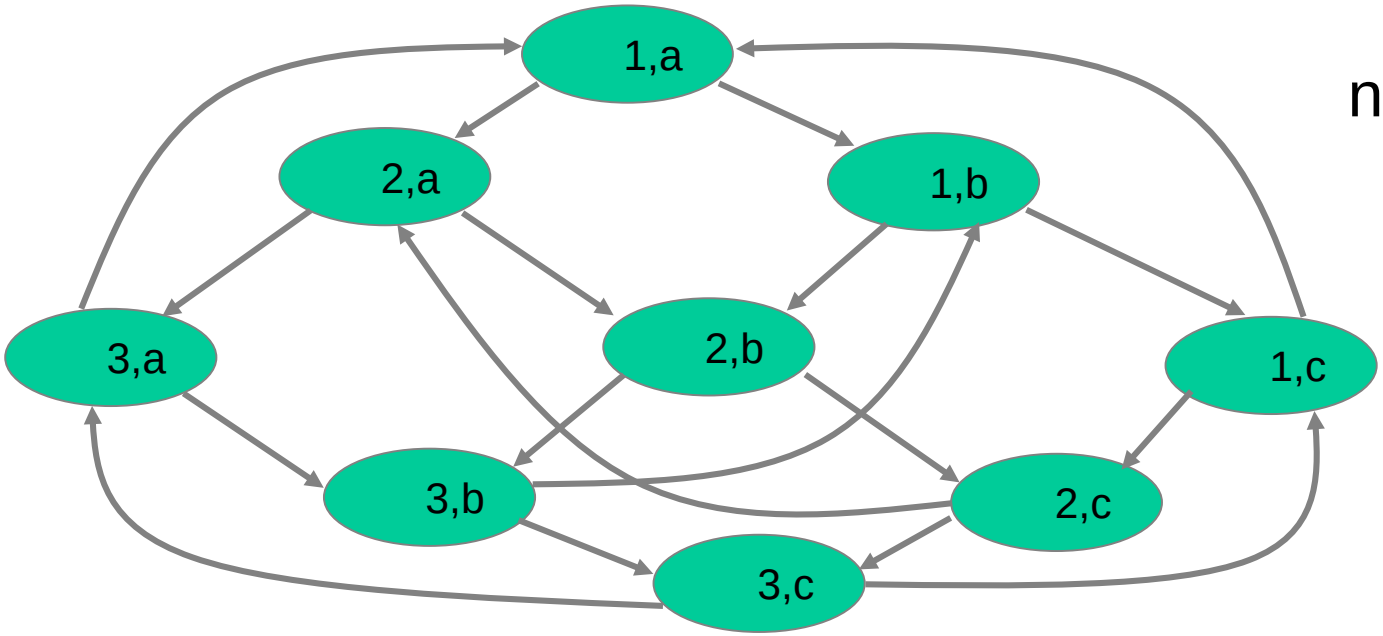
Main Disadvantage (Cont.)



||



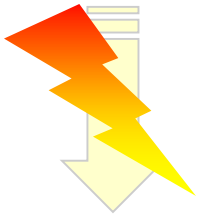
n states,
m processes



n^m states

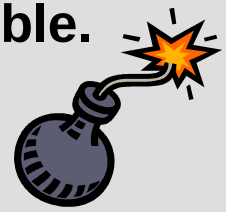
The State Explosion Problem

System Description



State Transition Graph

Combinatorial explosion of system states renders explicit model construction infeasible.



Exponential Growth of …

… global state space in number of concurrent components.

… memory states in memory size.

Feasibility of model checking inherently tied to handling state explosion.

Model Checking since 1981

1981 Clarke / Emerson: CTL Model Checking
Sifakis / Quielle
1982 EMC: Explicit Model Checker
Clarke, Emerson, Sistla

10^5

1990 Symbolic Model Checking
Burch, Clarke, Dill, McMillan
1992 SMV: Symbolic Model Verifier
McMillan

10^{100}

1990s: Formal Hardware Verification in Industry: Intel, IBM, Motorola, etc.

1998 Bounded Model Checking using SAT
Biere, Clarke, Zhu
2000 Counterexample-guided Abstraction Refinement
Clarke, Grumberg, Jha, Lu, Veith

10^{1000}

Grand Challenge:
Model Check Software !

What makes Software Model Checking different ?

What Makes Software Model Checking Different ?

- Large/unbounded base types: **`int`**, **`float`**, **`string`**
- User-defined types/classes
- Pointers/aliasing + unbounded #'s of heap-allocated cells
- Procedure calls/recursion/calls through pointers/dynamic method lookup/overloading
- Concurrency + unbounded #'s of threads

Grand Challenge:
Model Check Software !

Early attempts in the 1980s failed to scale.

2000s: renewed interest / demand:
Java Pathfinder: NASA Ames
SLAM: Microsoft
Bandera: Kansas State
BLAST: Berkeley
SPIN
…
SLAM shipped to Windows device driver developers.

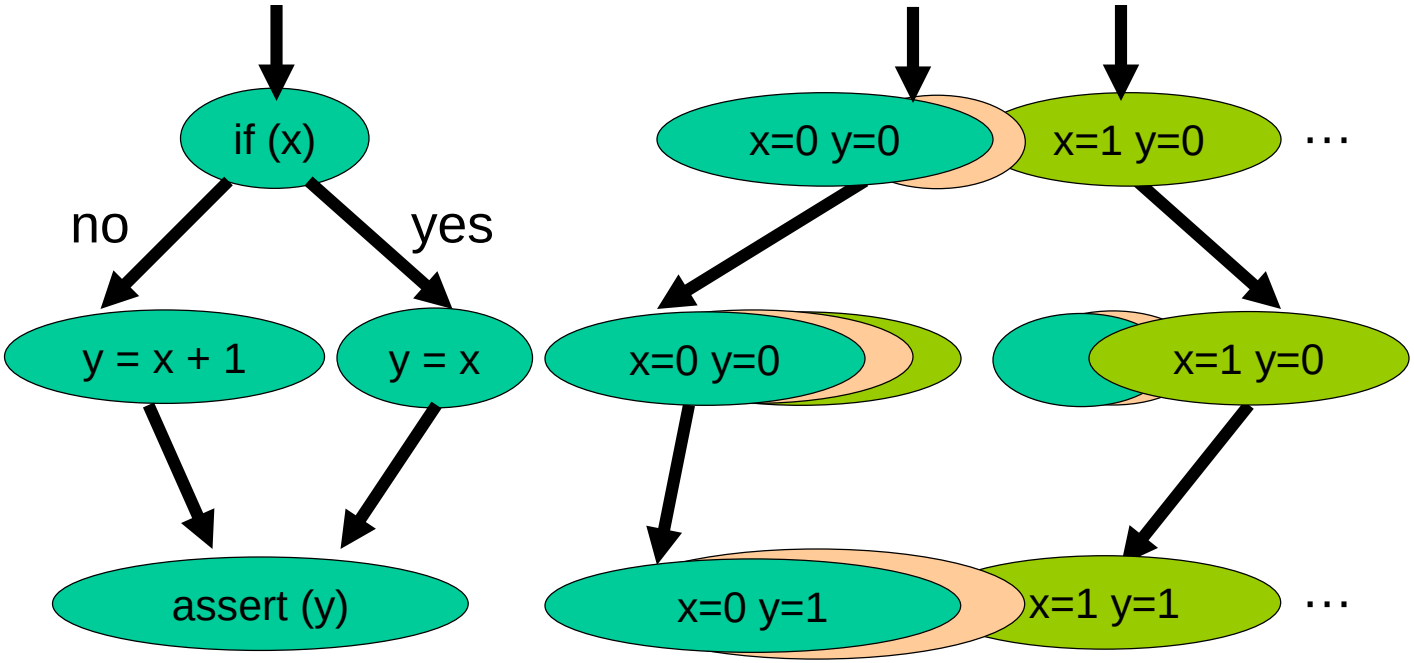
In general, these tools are unable to handle complex data structures and concurrency.

Model Checking

- Algorithm for answering queries about behaviors of state machines
  - Given a state machine M and a query ϕ does M satisfy ϕ ?
- Standard formulation:
  - M is a Kripke structure
  - ϕ is a temporal logic formula
    - Computational Tree Logic (CTL)
    - Linear Temporal Logic (LTL)
- Discovered independently by Clarke & Emerson and Queille & Sifakis in the early 1980's

Models of C Code

```
if (x) {
  y = x;
} else {
  y = x + 1;
}
assert (y);
```

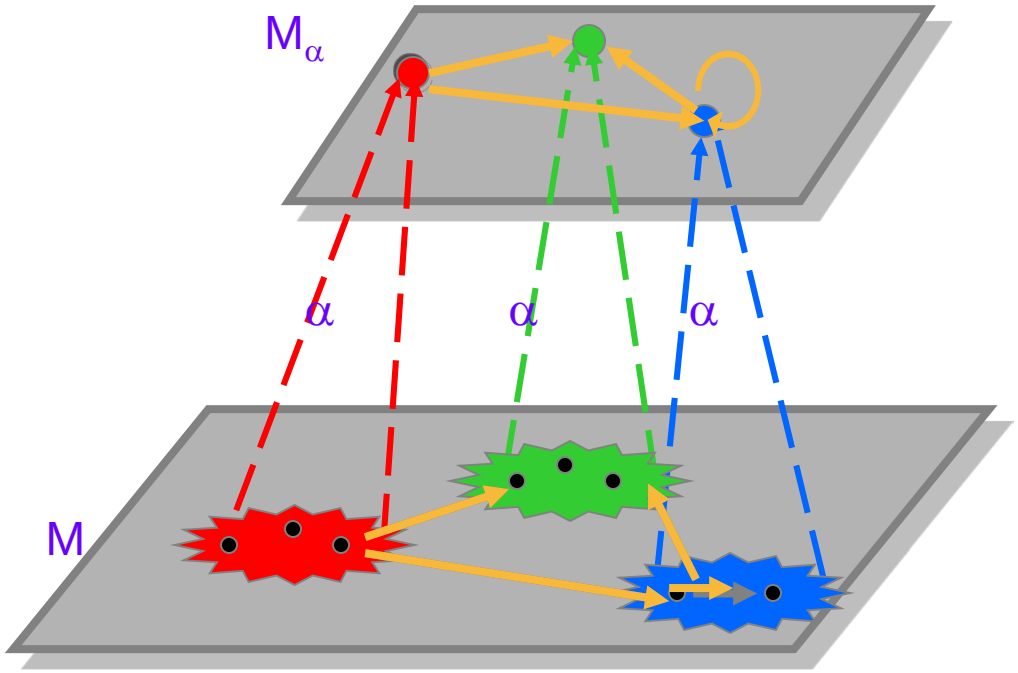


Program: Syntax Control Flow Graph Model: Semantics

Infinite State

Existential Abstraction

Given an abstraction function $\alpha : S \rightarrow S_\alpha$, the concrete states are grouped and mapped into abstract states:

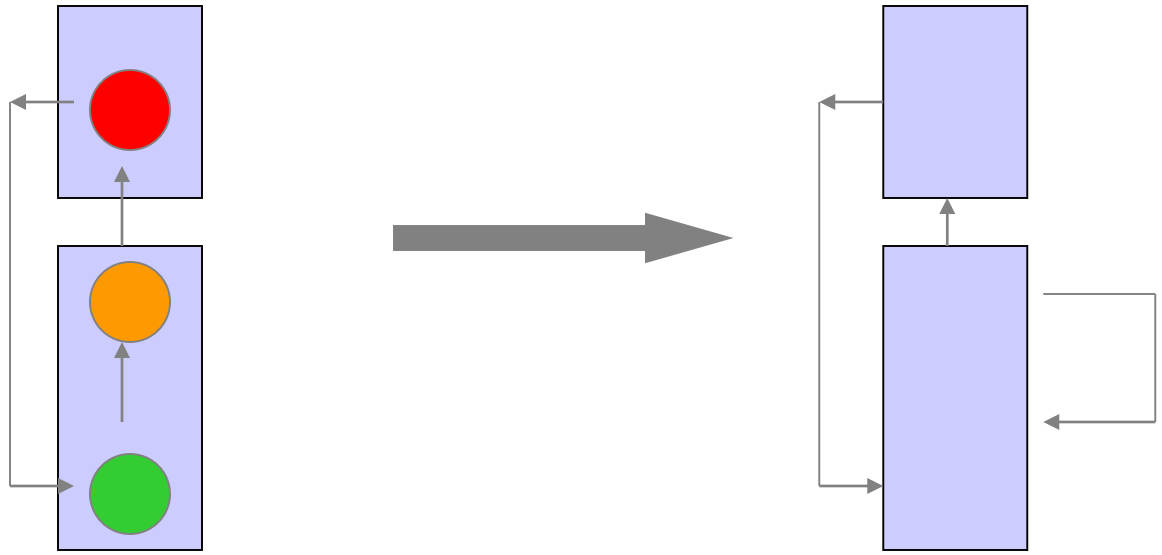


Preservation Theorem

- Theorem (Clarke, Grumberg, Long) If property holds on abstract model, it holds on concrete model

- Technical conditions
  - Property is universal i.e., no existential quantifiers
  - Atomic formulas respect abstraction mapping
- Converse implication is not true !

Spurious Behavior



AGAF red

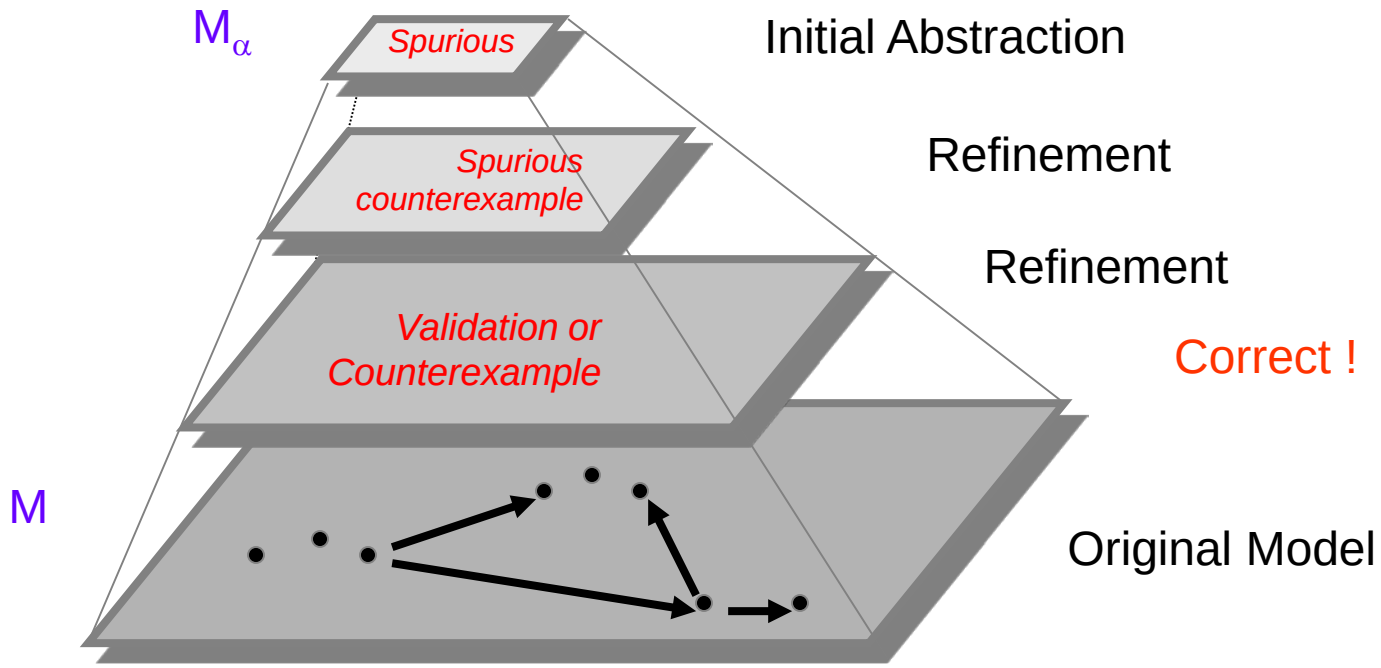
“Every path necessarily leads back to red.”

Spurious Counterexample:

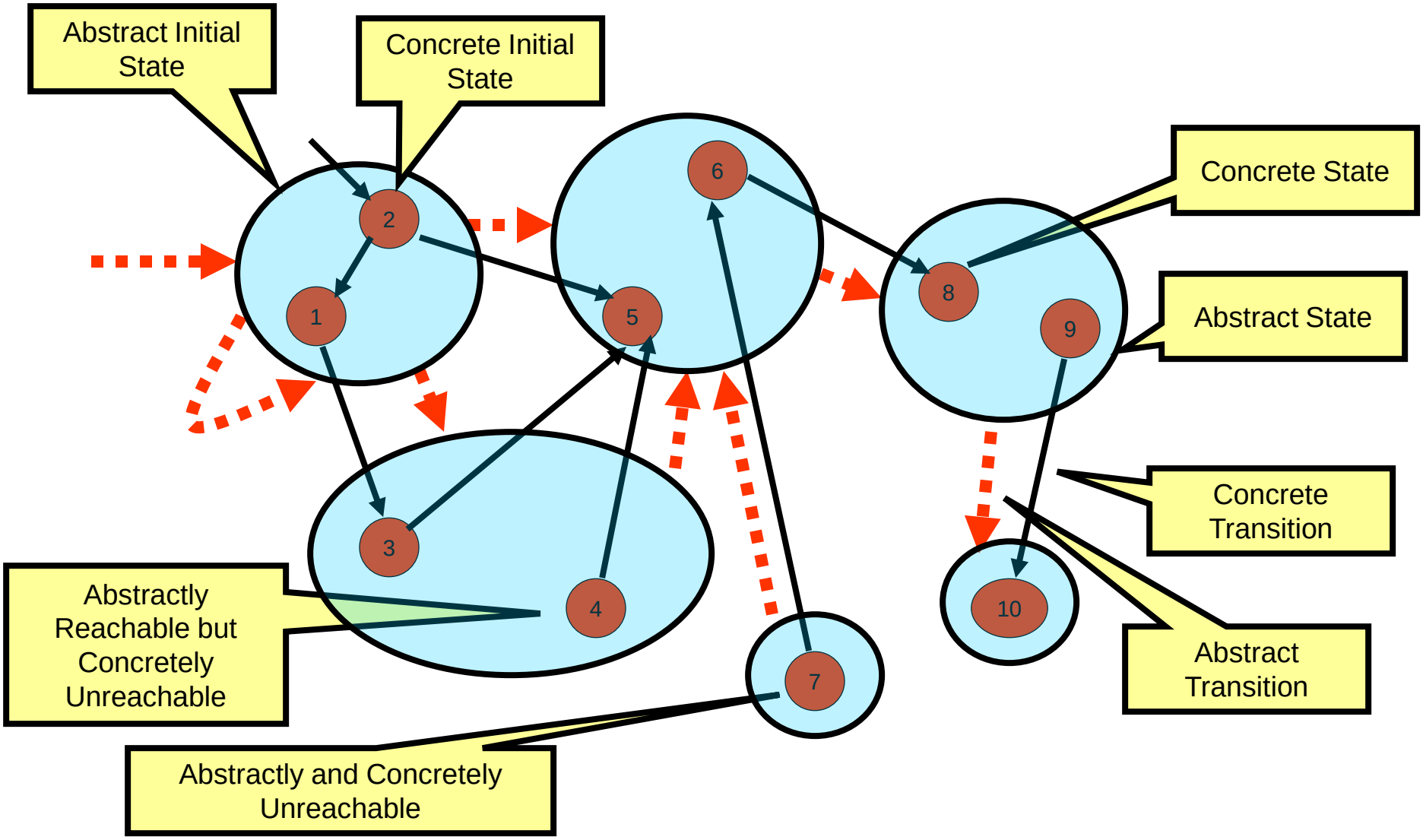
<go><go><go><go> ...

Artifact of the abstraction !

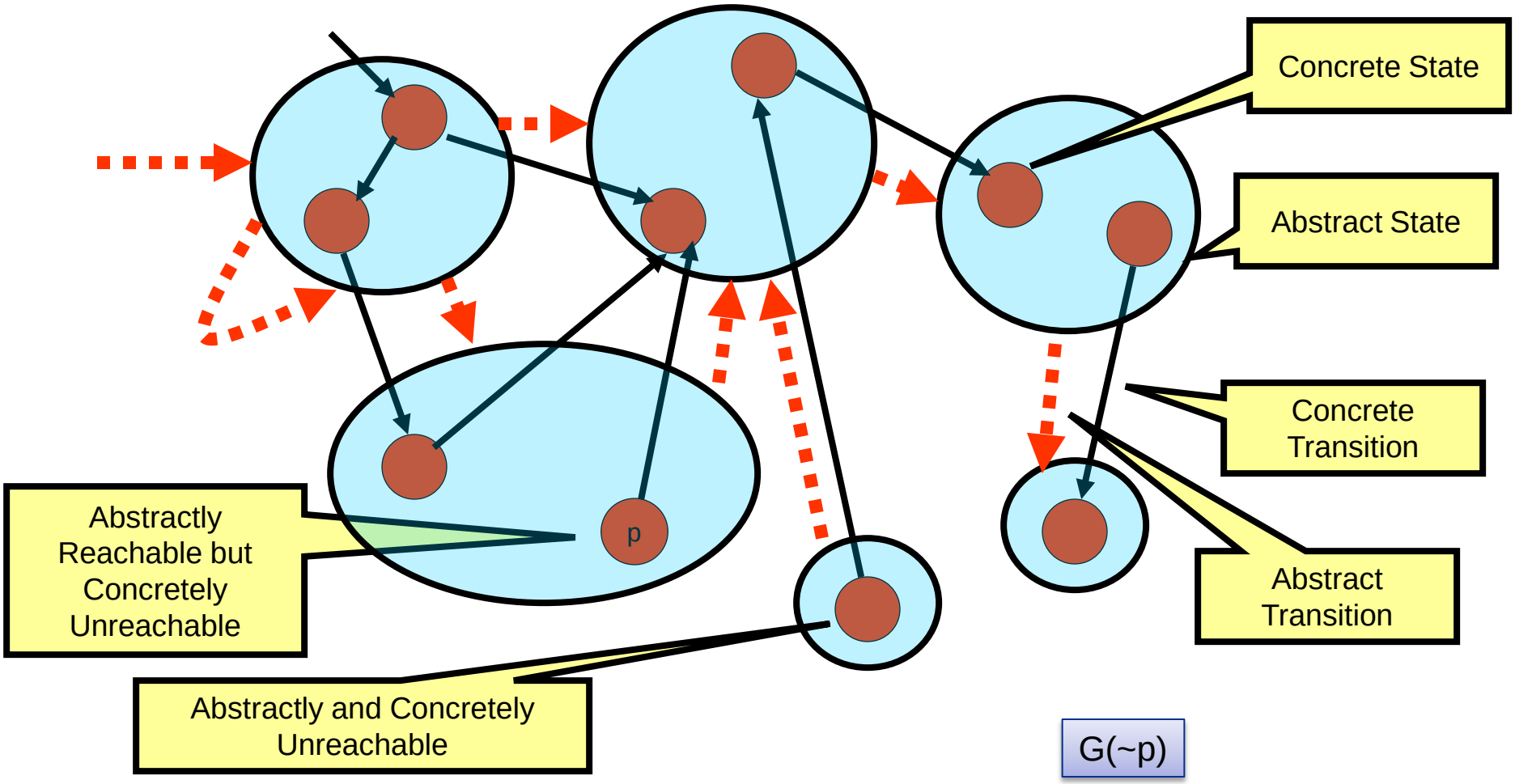
Automatic Abstraction



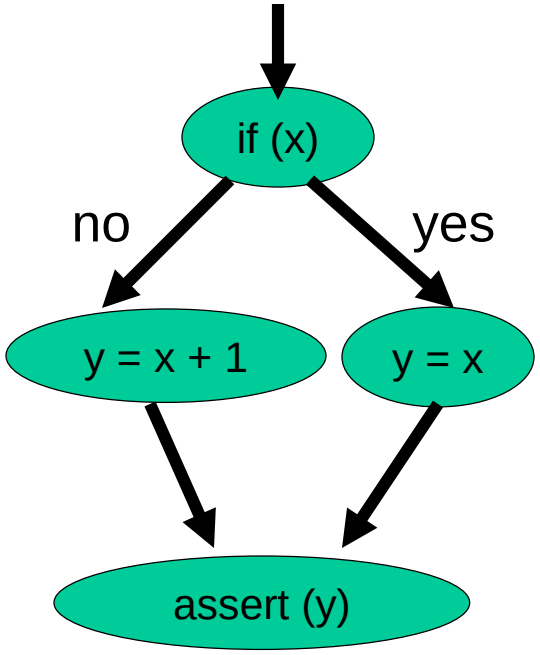
Example of Existential Abstraction



Example of Existential Abstraction

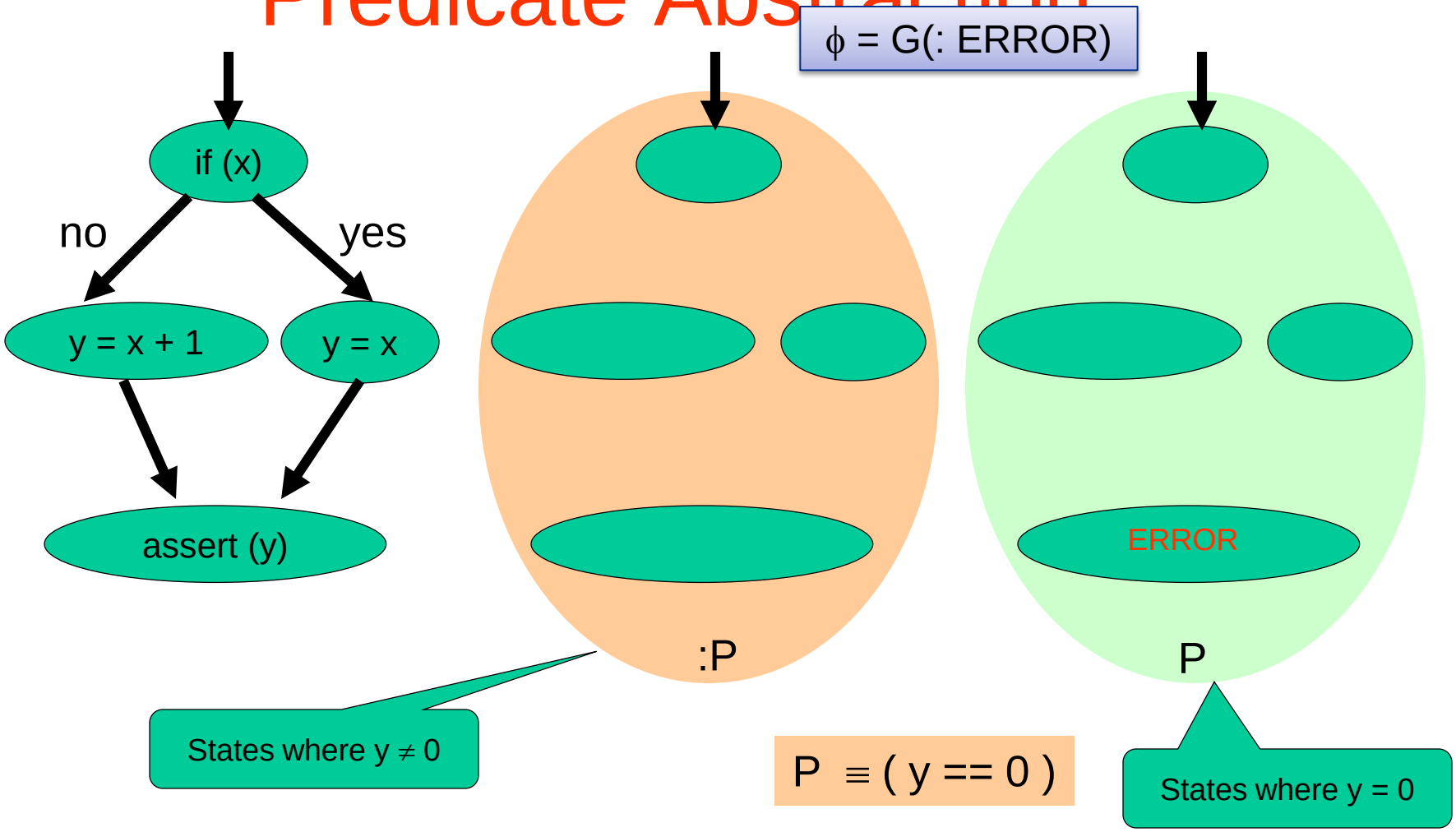


Predicate Abstraction



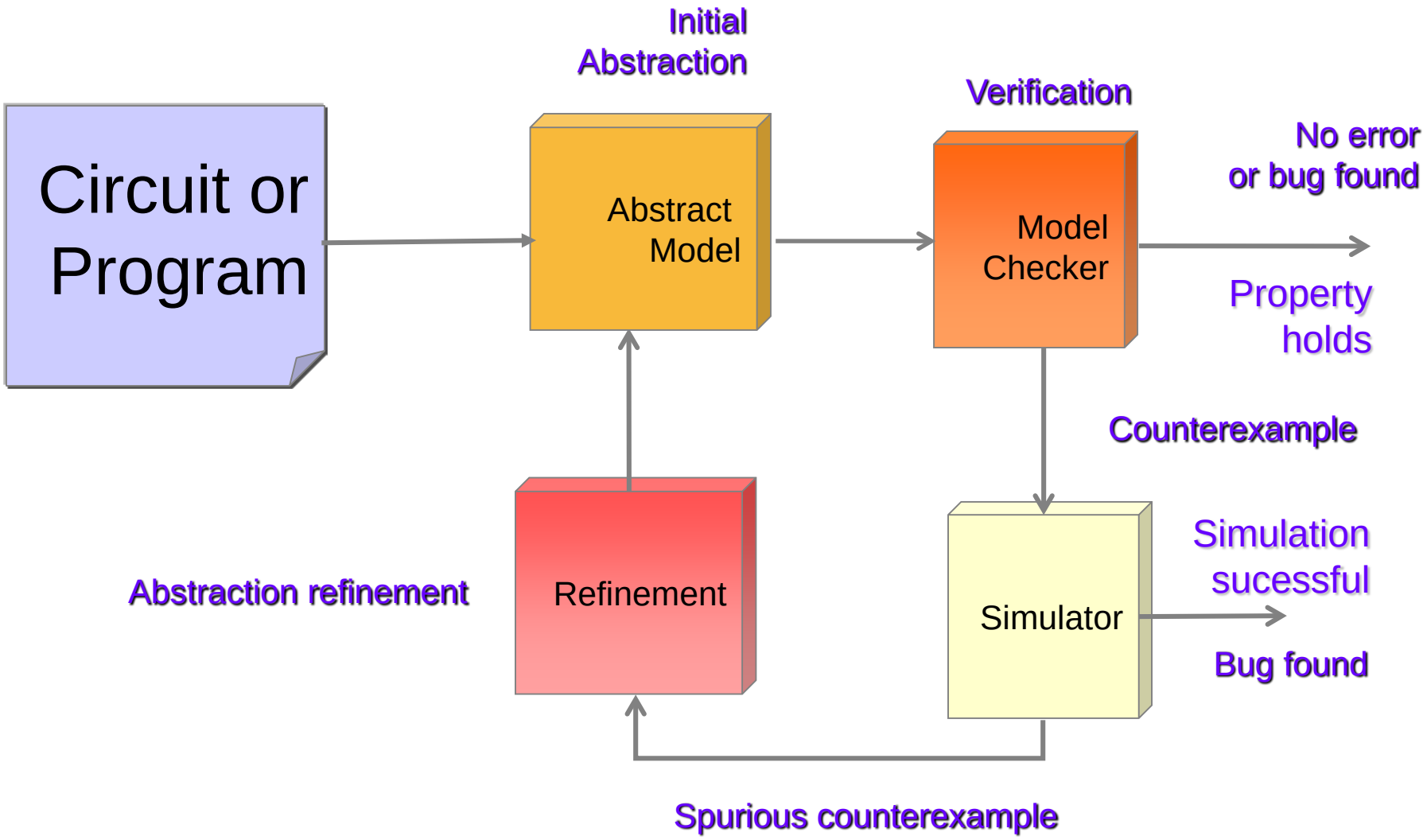
Partition the statespace based on values of a finite set of predicates on program variables

Predicate Abstraction



CEGAR

CounterExample-Guided Abstraction Refinement



Formal Verification: The security context

Correctness vs Security

- Program or system **correctness:**

  program satisfies specification
  - For reasonable input, get reasonable output
- Program or system **security**:

  program properties preserved in face of attack
  - For unreasonable input, output not completely disastrous
- Main differences
  - Active interference from adversary
  - Refinement techniques may fail
    - Abstraction is very difficult to achieve in security:
      what if the adversary operates below your level of abstraction?

Security Analysis

❶ Model system

❷ Model adversary

❸ Identify security properties

❹ See if properties preserved under attack

Theme #1: there are many notions of what it means for a protocol to be "secure"

Theme #2: there are many ways of looking for security flaws

- Result
  - Under given assumptions about system, no attack of a certain form will destroy specified properties
  - There is no "absolute" security

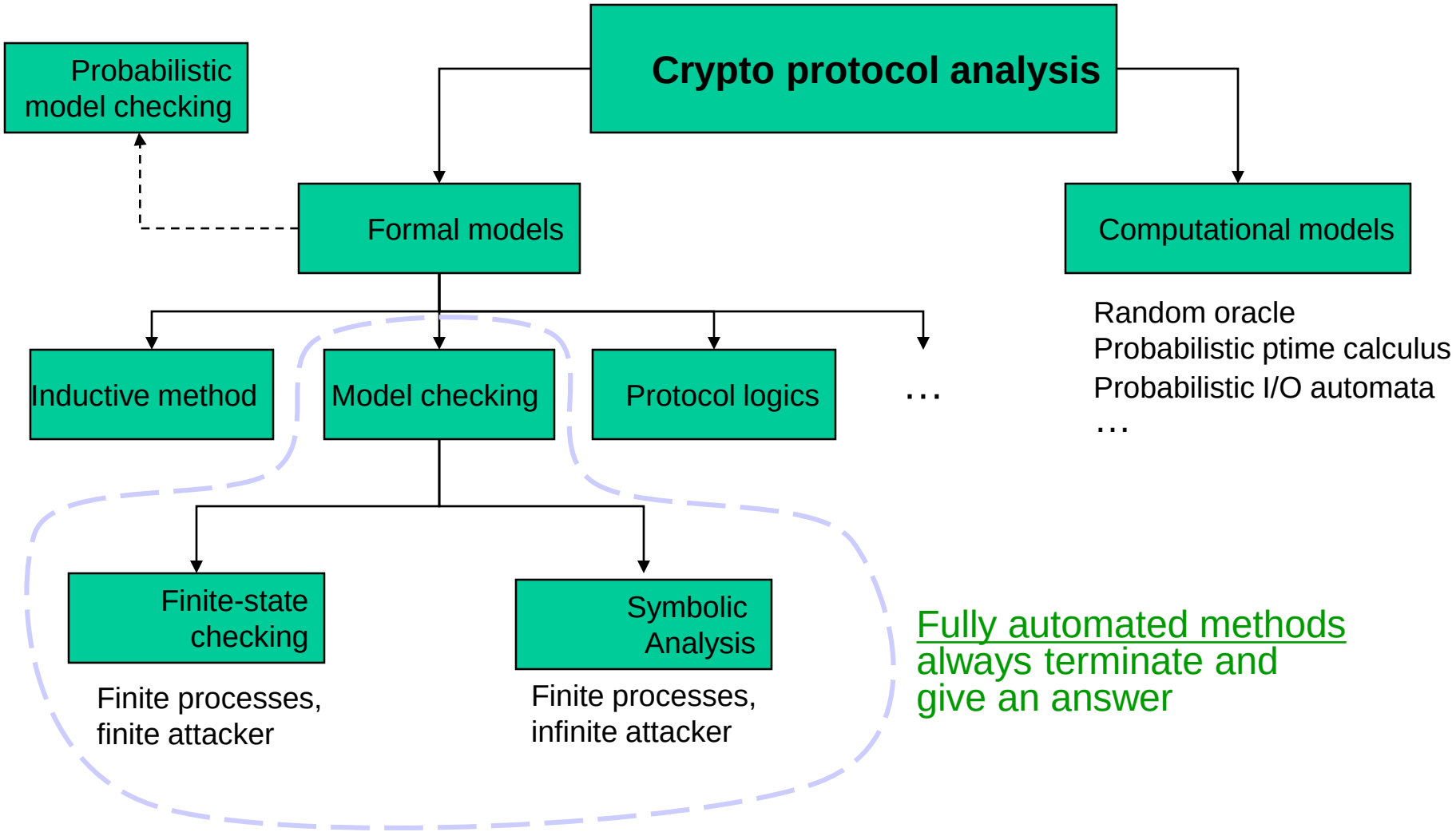
Theme #1: Protocols and Properties

- Authentication
  - Needham-Schroeder, Kerberos
- Key establishment
  - SSL/TLS, IPSec protocols
- Secure group protocols
- Anonymity
- Electronic payments, wireless security, fair exchange, privacy…

Theme #2: Formal Analysis Methods

- Focus on special-purpose security applications
  - Some techniques are very different from those used in hardware verification
  - In all cases, the main difficulty is modeling the attacker
- Simple, mechanical models of the attacker

Protocol Analysis Techniques



Variety of Tools and Techniques

Secrecy
Authentication
Authorization

- Explicit finite-state checking
  - Murφ model checker
- Infinite-state symbolic model checking
  - SRI constraint solver
- Process algebras
  - Applied pi-calculus

Anonymity

- Probabilistic model checking
  - PRISM probabilistic model checker

Fairness

- Game-based verification
  - MOCHA model checker

Slide sources

- Edmund Clarke's course:
  http://www.cs.cmu.edu/~emc/15414-f11/lecture/
- Vitaly Shmatikov's course:
  http://www.cs.utexas.edu/~shmat/courses/cs395t_fall04/cs395t_home.html
- Tom Chotia's course:
  http://www.cs.bham.ac.uk/~tpc/cwi/Teaching/index.html