Model Checking

My 27 year quest to overcome the state explosion problem

Edmund Clarke

Computer Science Department

Carnegie Mellon University

Intel Pentium FDIV Bug

- Try 4195835 – 4195835 / 3145727 * 3145727.
  - In 94’ Pentium, it doesn’t return 0, but 256.
- Intel uses the SRT algorithm for floating point division. Five entries in the lookup table are missing.
- Cost: $500 million
- Xudong Zhao’s Thesis on Word Level Model Checking

Recent Rumor: New AMD TLB Bug??

- AMD Family 10h revision B2 processors suffer from an issue in the processor TLB (Translation Lookaside Buffer).
- Launch date of these processors was delayed in September, 2007.
- AMD doesn’t have official announcement yet, but you can google “AMD Barcelona bug” for plenty of discussion.

Temporal Logic Model Checking

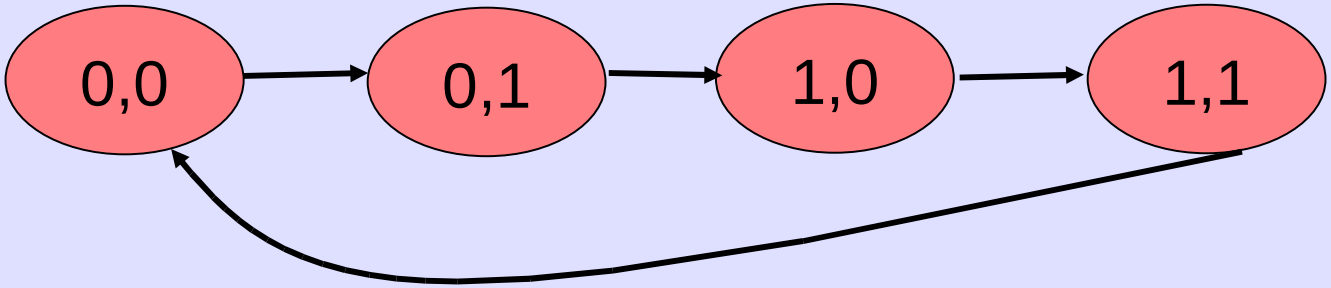
- Model checking is an automatic verification technique for finite state concurrent systems.
- Developed independently by Clarke and Emerson and by Queille and Sifakis in early 1980's.
- Specifications are written in propositional temporal logic.
- Verification procedure is an exhaustive search of the state space of the design.

Advantages of Model Checking

- No proofs!!!
- Fast (compared to other rigorous methods such as theorem proving)
- Diagnostic counterexamples
- No problem with partial specifications
- Logics can easily express many concurrency properties

Main Disadvantage

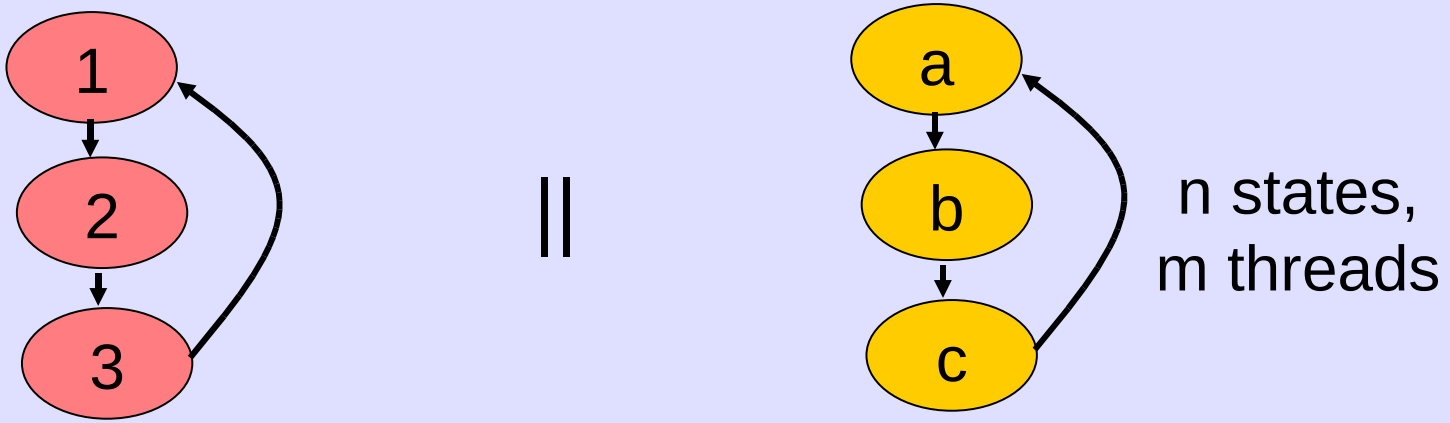
State Explosion Problem:

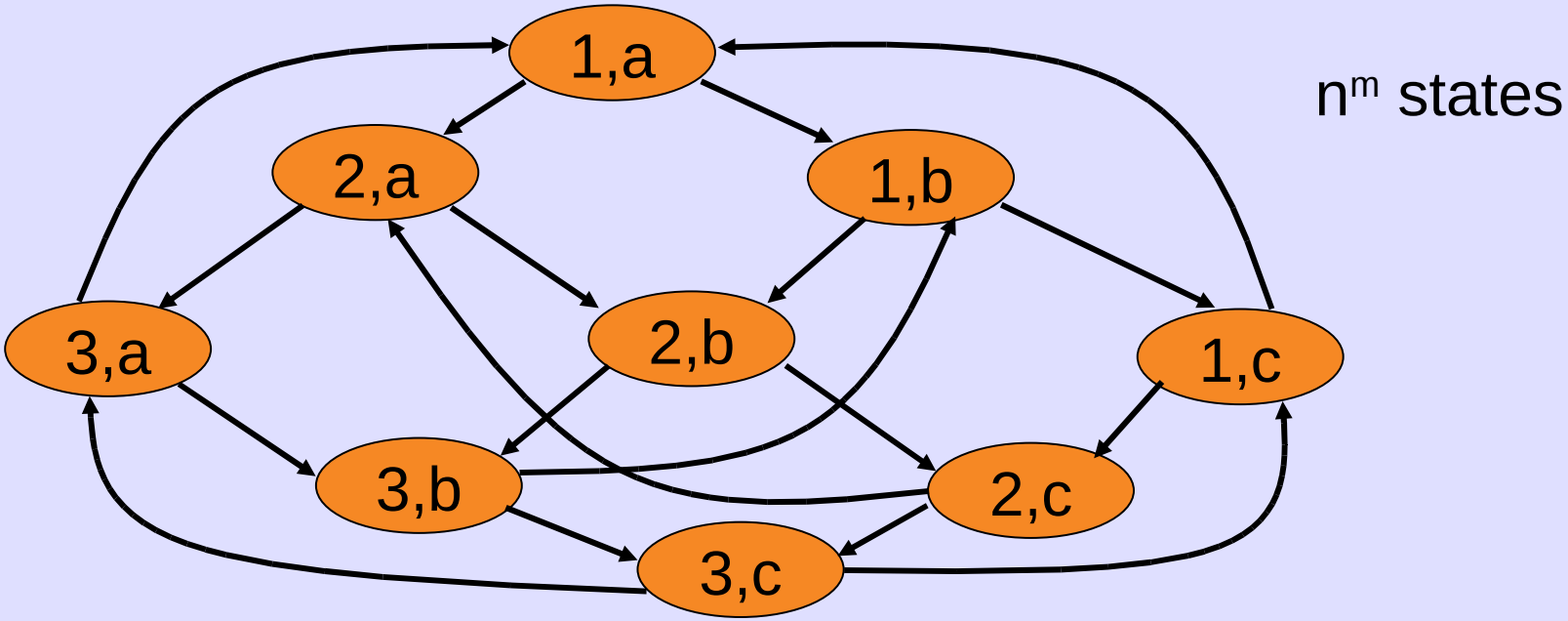


2-bit counter

n-bit counter has 2^n states

Main Disadvantage Contd.





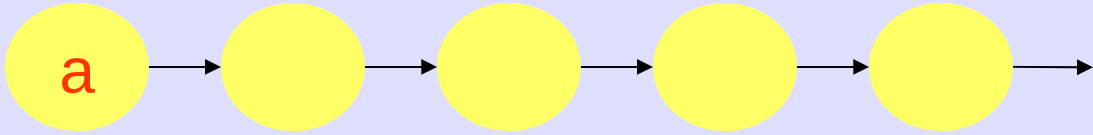
Main Disadvantage Contd.

State Explosion Problem:

Unavoidable in worst case, but steady progress over the past 27 years using clever algorithms, data structures, and engineering

LTL - Linear Time Logic

Determines Patterns on Infinite Traces



Atomic Propositions

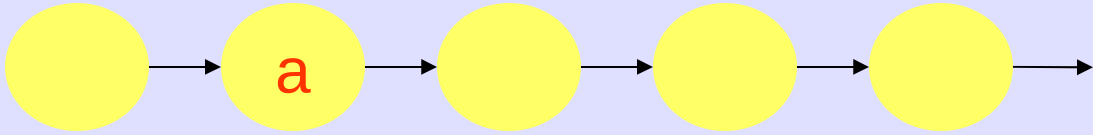
Boolean Operations

Temporal operators

a	**“a is true now”**
X a	**“a is true in the neXt state”**
Fa	**“a will be true in the Future”**
Ga	**“a will be Globally true in the future”**
a U b	**“a will hold true Until b becomes true”**

LTL - Linear Time Logic

Determines Patterns on Infinite Traces



Atomic Propositions

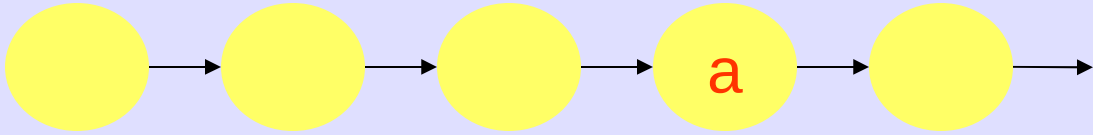
Boolean Operations

Temporal operators

a	**“a is true now”**
X a	**“a is true in the neXt state”**
Fa	**“a will be true in the Future”**
Ga	**“a will be Globally true in the future”**
a U b	**“a will hold true Until b becomes true”**

LTL - Linear Time Logic

Determines Patterns on Infinite Traces



Atomic Propositions

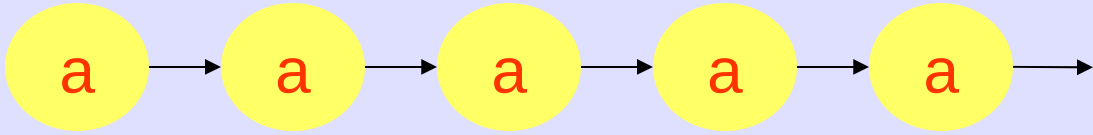
Boolean Operations

Temporal operators

a	**“a is true now”**
X a	**“a is true in the neXt state”**
Fa	**“a will be true in the Future”**
Ga	**“a will be Globally true in the future”**
a U b	**“a will hold true Until b becomes true”**

LTL - Linear Time Logic

Determines Patterns on Infinite Traces



Atomic Propositions

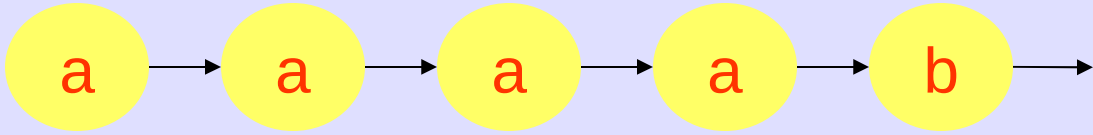
Boolean Operations

Temporal operators

a	**“a is true now”**
X a	**“a is true in the neXt state”**
Fa	**“a will be true in the Future”**
Ga	**“a will be Globally true in the future”**
a U b	**“a will hold true Until b becomes true”**

LTL - Linear Time Logic

Determines Patterns on Infinite Traces



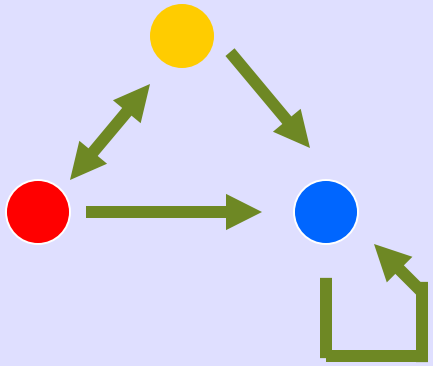
Atomic Propositions

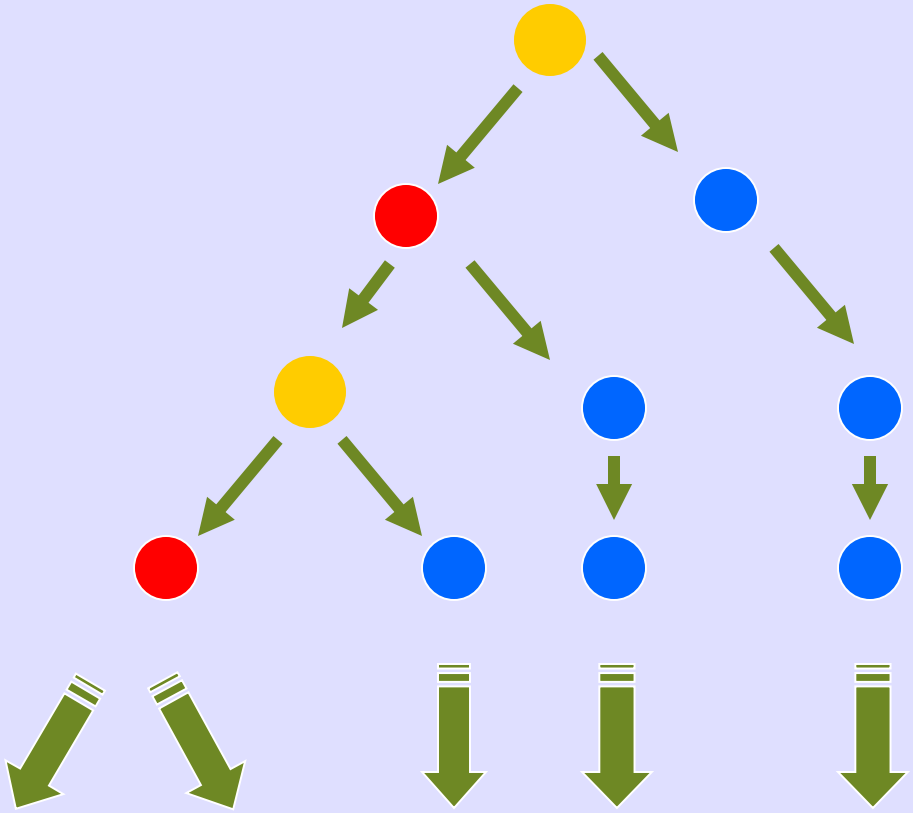
Boolean Operations

Temporal operators

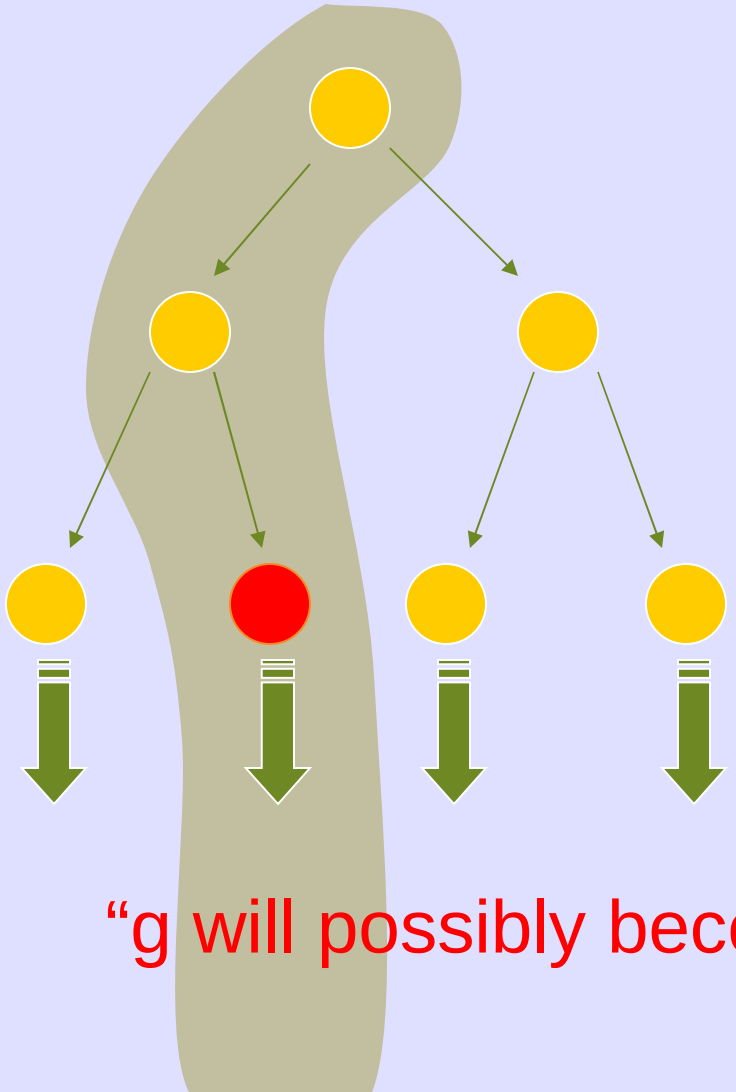
a	**“a is true now”**
X a	**“a is true in the neXt state”**
Fa	**“a will be true in the Future”**
Ga	**“a will be Globally true in the future”**
a U b	**“a will hold true Until b becomes true”**

Branching Time



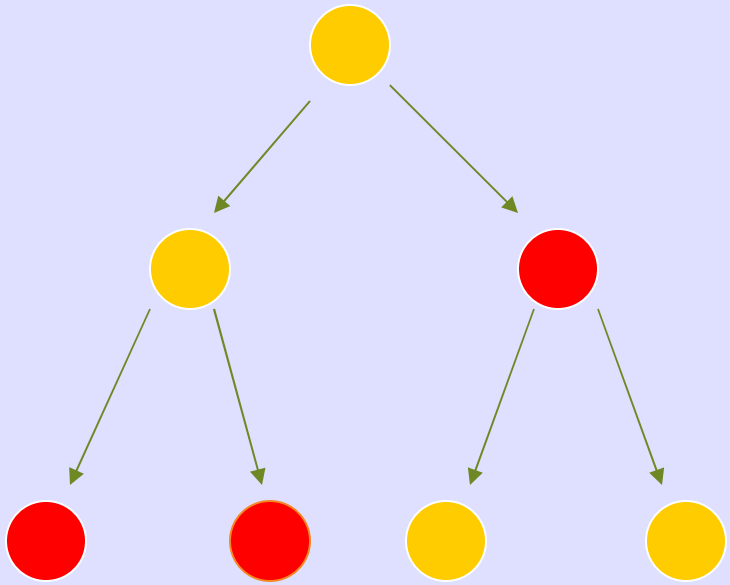


CTL: Computation Tree Logic



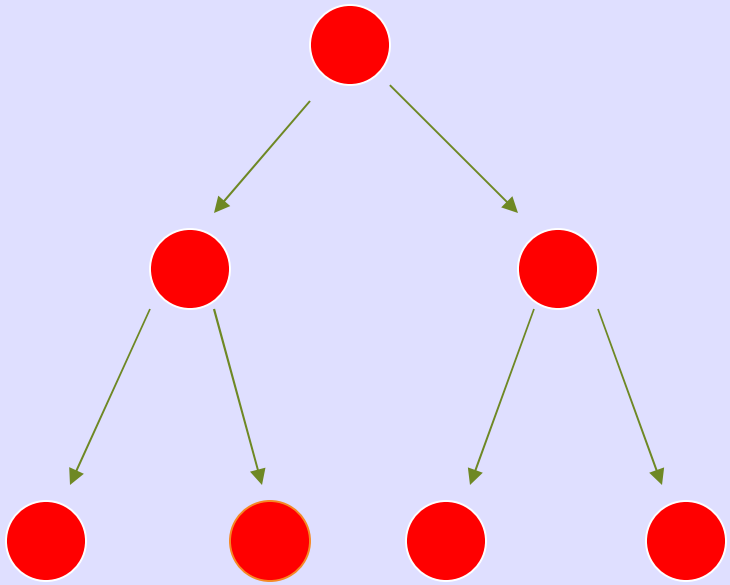
EF g “g will possibly become true”

CTL: Computation Tree Logic



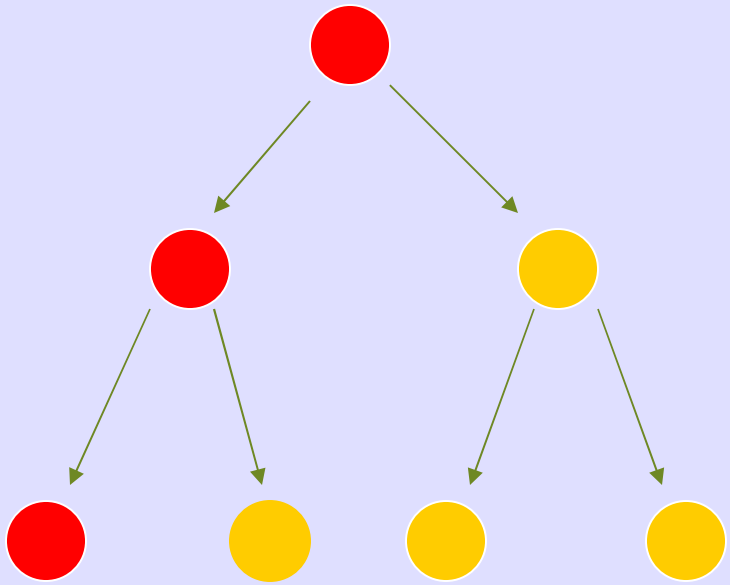
AF g “g will necessarily become true”

CTL: Computation Tree Logic



AG g “g is an invariant”

CTL: Computation Tree Logic



EG g “g is a potential invariant”

CTL: Computation Tree Logic

CTL uses the temporal operators

AX, AG, AF, AU

EX, EG, EF, EU

CTL* allows complex nestings such as

AXX, AGX, EXF, ...

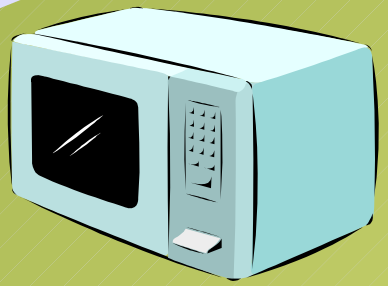
CTL: linear model checking algorithm !

Model Checking Problem

- Let *M* be a state-transition graph.
- Let *f* be the specification in temporal logic.
- Find all states *s* of *M* such that *M, s* |= *f*.

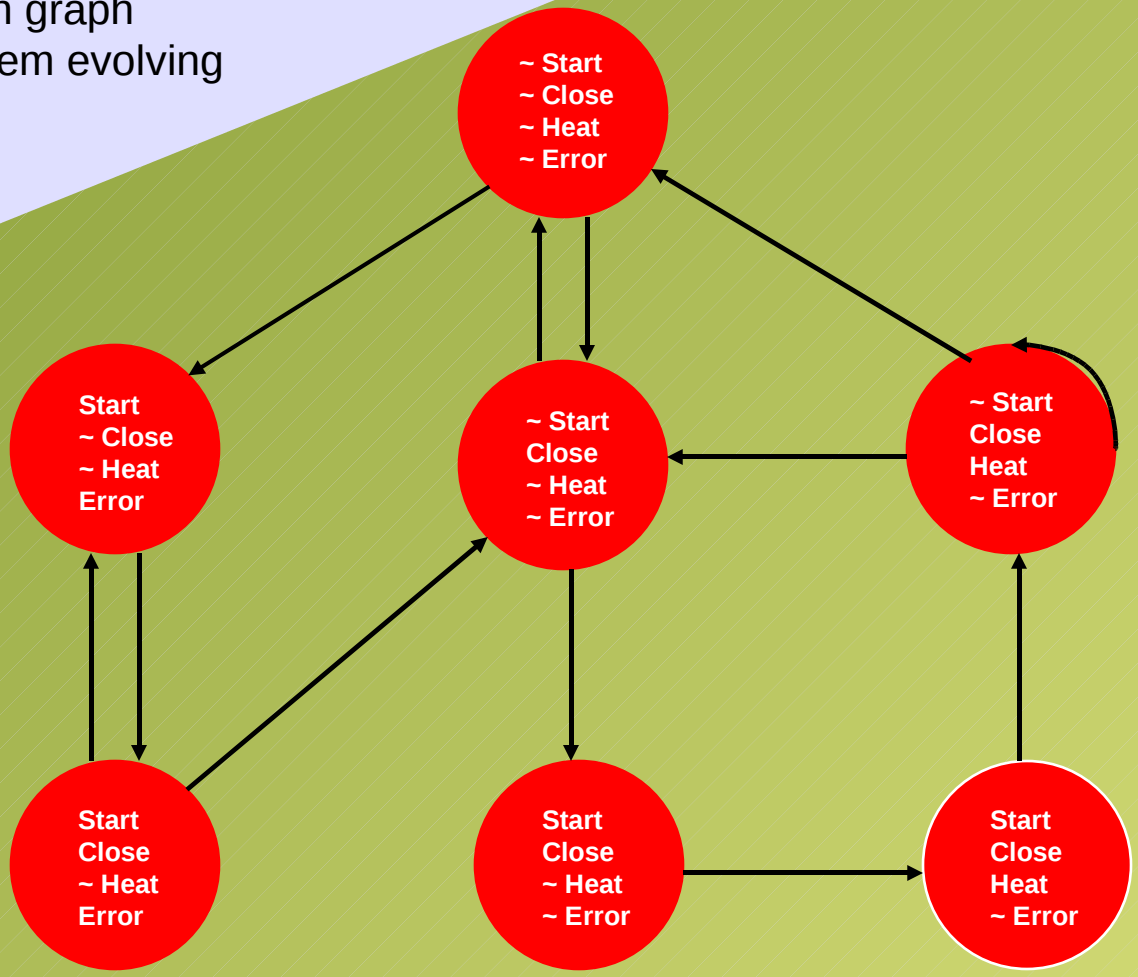
- CTL Model Checking: CE 81; CES 83/86; QS 81/82.
- LTL Model Checking: LP 85.
- Automata Theoretic LTL Model Checking: VW 86.
- CTL* Model Checking: EL 85.

Model of computation



Microwave Oven Example

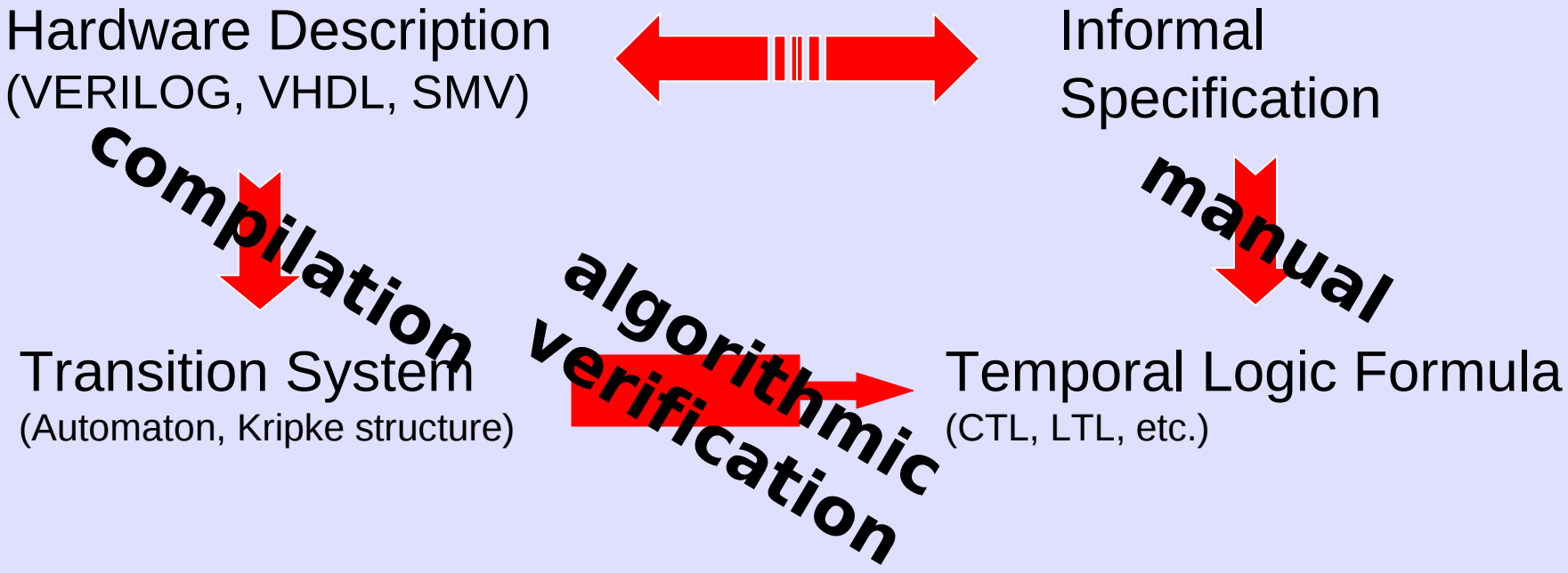
State-transition graph describes system evolving over time.



Temporal Logic and Model Checking

- The oven doesn't heat up until the door is closed.
- Not heat_up holds until door_closed
- (~ heat_up) U door_closed

Model Checking



Hardware Example: IEEE Futurebus+

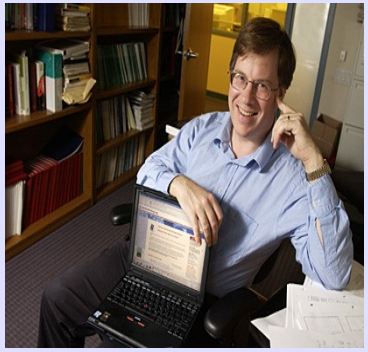
- In 1992 we used Model Checking to verify the IEEE Future+ cache coherence protocol.
- Found a number of previously undetected errors in the design.
- First time that formal methods were used to find errors in an IEEE standard.
- Development of the protocol began in 1988, but previous attempts to validate it were informal.

Four Big Breakthroughs on State Space Explosion Problem!

- **Symbolic Model Checking**

  Burch, Clarke, McMillan, Dill, and Hwang 90;

  Ken McMillan's thesis 92


- **The Partial Order Reduction**

  Valmari 90

  Godefroid 90

  Peled 94



Four Big Breakthroughs on State Space Explosion Problem (Cont.)

- **Bounded Model Checking**
  - Biere, Cimatti, Clarke, Zhu 99
  - Using Fast SAT solvers
  - Can handle thousands of state elements



Can the given property fail in k-steps?

$$I(V_0) \text{ Æ } T(V_0,V_1) \text{ Æ } \ldots \text{ Æ } T(V_{k-1},V_k) \text{ Æ } (: P(V_0) \text{ Ç} \ldots \text{Ç}: P(V_k))$$

Initial state

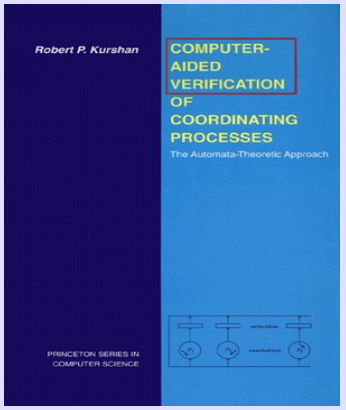
k-steps

Property fails in some step

BMC in practice: Circuit with 9510 latches, 9499 inputs
BMC formula has $4 \text{ £ } 10^6$ variables, $1.2 \text{ £ } 10^7$ clauses
Shortest bug of length 37 found in 69 seconds

Four Big Breakthroughs on State Space Explosion Problem (Cont.)

- **Localization Reduction**
  - Bob Kurshan 1994



- **Counterexample Guided Abstraction Refinement (CEGAR)**
  - Clarke, Grumberg, Jha, Lu, Veith 2000
  - Used in most software model checkers

From Hardware to Software:

Natural Question: Is it possible to model check software?

According to Wired News on Nov 10, 2005:

"When Bill Gates announced that the technology was under development at the 2002 Windows Engineering Conference, he called it the holy grail of computer science"

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

What Makes Software Model Checking Different ?

- Templates/generics/include files
- Interrupts/exceptions/callbacks
- Use of secondary storage: files, databases
- Absent source code for: libraries, system calls, mobile code
- Esoteric features: continuations, self-modifying code
- Size (e.g., MS Word = 1.4 MLOC)

What Does It Mean to Model Check Software?

1. Combine static analysis and model checking

Use **static analysis** to extract a **model K** from a boolean abstraction of the program.

Then check that f is true in K ($K \models f$), where f is the specification of the program.

- SLAM (Microsoft)
- Bandera (Kansas State)
- MAGIC, SATABS (CMU)
- BLAST (Berkeley)
- F-Soft (NEC)

What Does It Mean to Model Check Software?

1. **Simulate program along all paths in computation tree**

   - Java PathFinder (NASA Ames)
   - Source code + backtracking (e.g., Verisoft)
   - Source code + symbolic execution + backtracking (e.g., MS/Intrinsa Prefix)

- **Use finite-state machine to look for patterns in control-flow graph** [Engler]

What Does It Mean to Model Check Software?

1. Design with Finite-State Software Models

Finite state software models can act as "missing link" between transition graphs and complex software.

- **Statecharts**
- **Esterel**

What Does It Mean to Model Check Software?

- **Use Bounded Model Checking and SAT [Kroening]**

  - **Problem: How to compute set of reachable states? Fixpoint computation is too expensive.**

  - **Restrict search to states that are reachable from initial state within fixed number n of transitions**

  - **Implemented by unwinding program and using SAT solver**

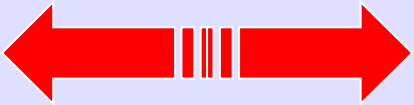
Key techniques for Software Model Checking

- **Counterexample Guided Abstraction Refinement**
  - Kurshan, Yuan Lu, Clarke et al JACM, Ball et al
  - Uses counterexamples to refine abstraction

- **Predicate Abstraction**
  - Graf and Saidi, Ball et al, Chaki et al, Kroening
  - Keeps track of certain predicates on data
  - Captures relationship between variables

Counterexamples

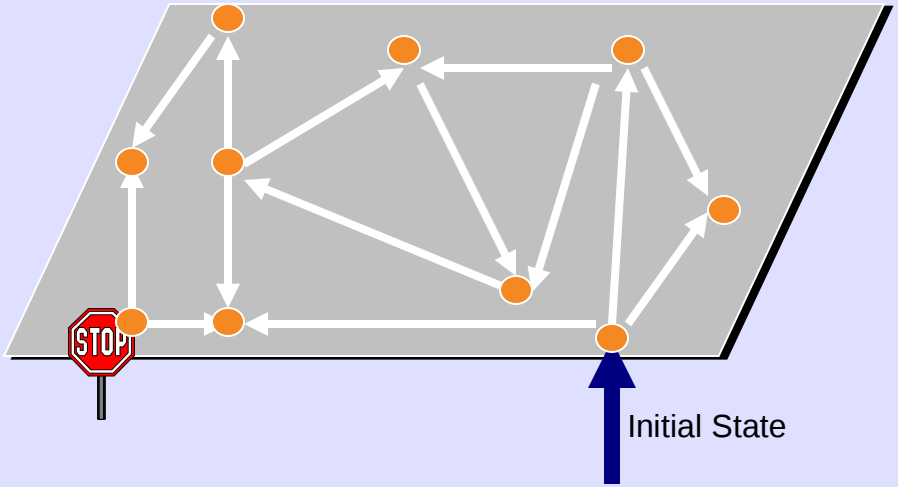
Program



Informal Specification

Transition System

Temporal Logic Formula
(CTL, LTL, etc.)



Safety Property:
bad state 

unreachable:

satisfied

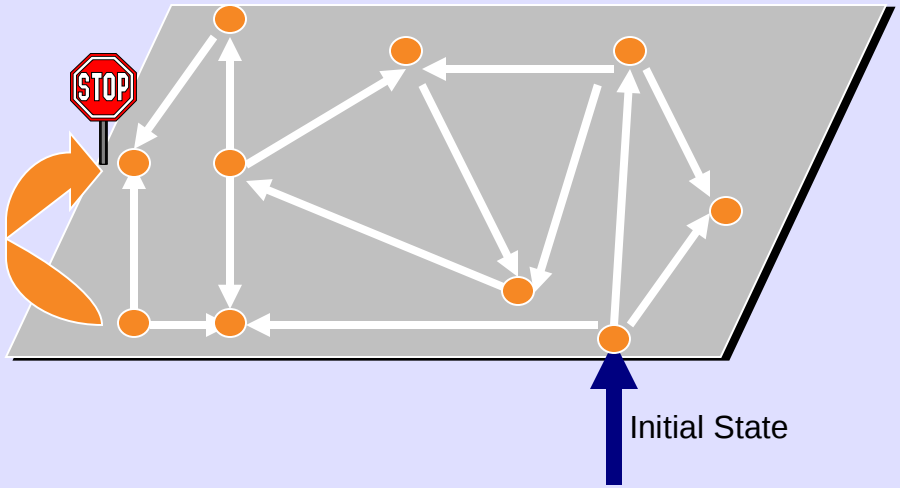
Counterexamples

Program

Informal Specification

Transition System

Temporal Logic Formula
(CTL, LTL, etc.)



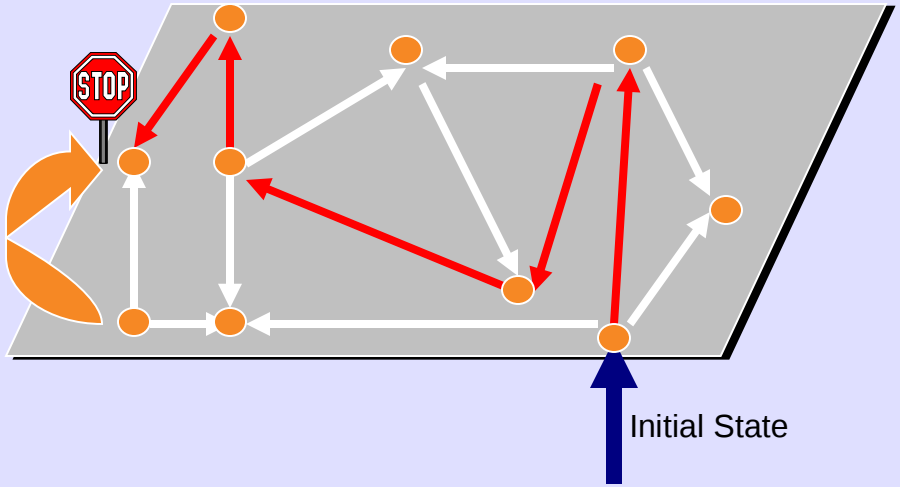
Safety Property:
bad state STOP unreachable

Counterexample

Counterexamples

Program

Transition System



Informal
Specification

Temporal Logic Formula
(CTL, LTL, etc.)

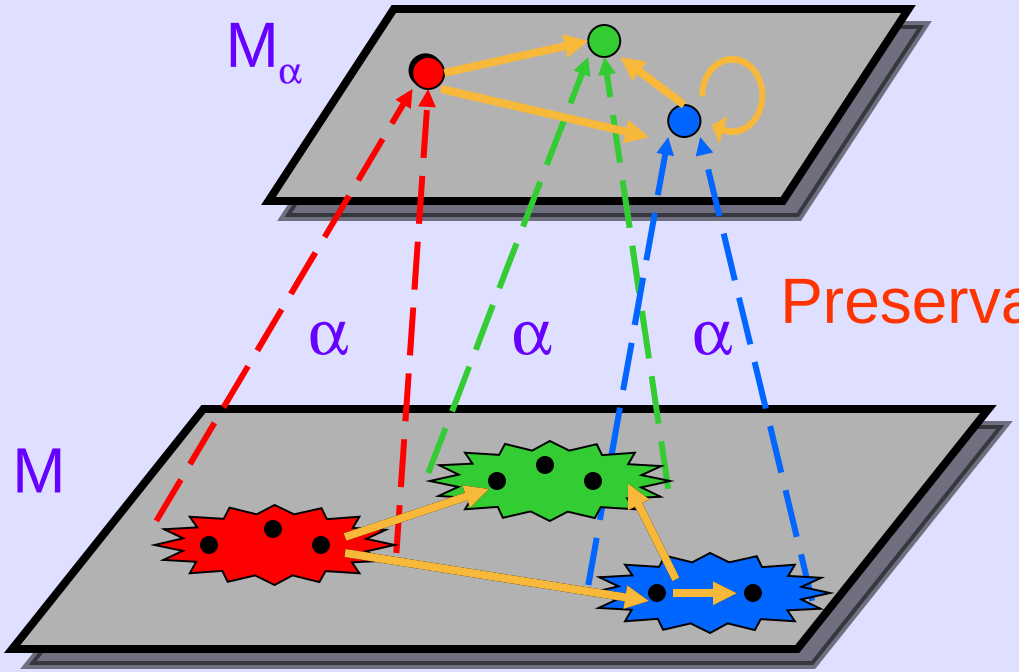
Safety Property:
bad state 

unreachable

Counterexample

Existential Abstraction

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



Preservation Theorem ?

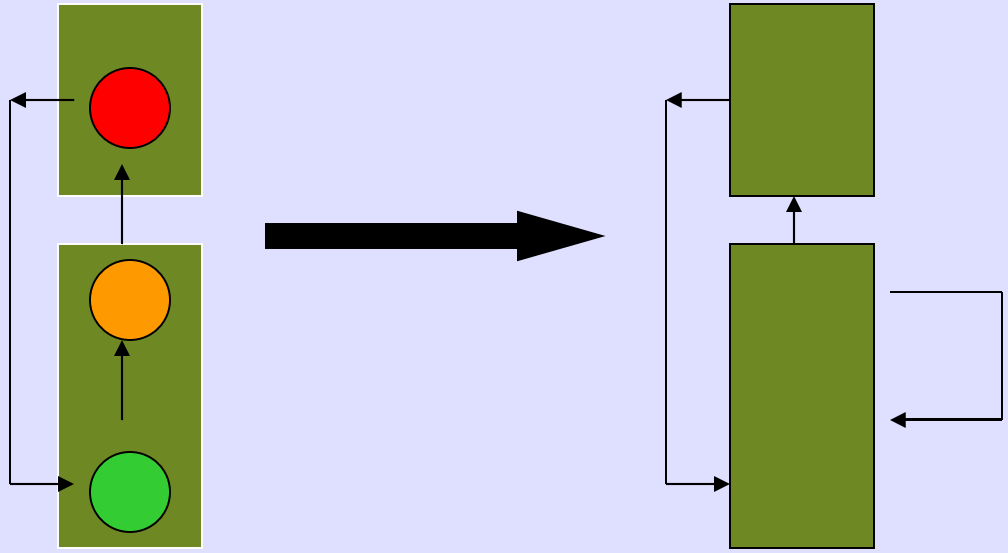
Preservation Theorem

- **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 valid !

Spurious Behavior



“red”

“go”

AGAF red

“Every path necessarily leads back to red.”

Spurious Counterexample:

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

Artifact of the abstraction !

How to define Abstraction Functions?

Abstraction too fine

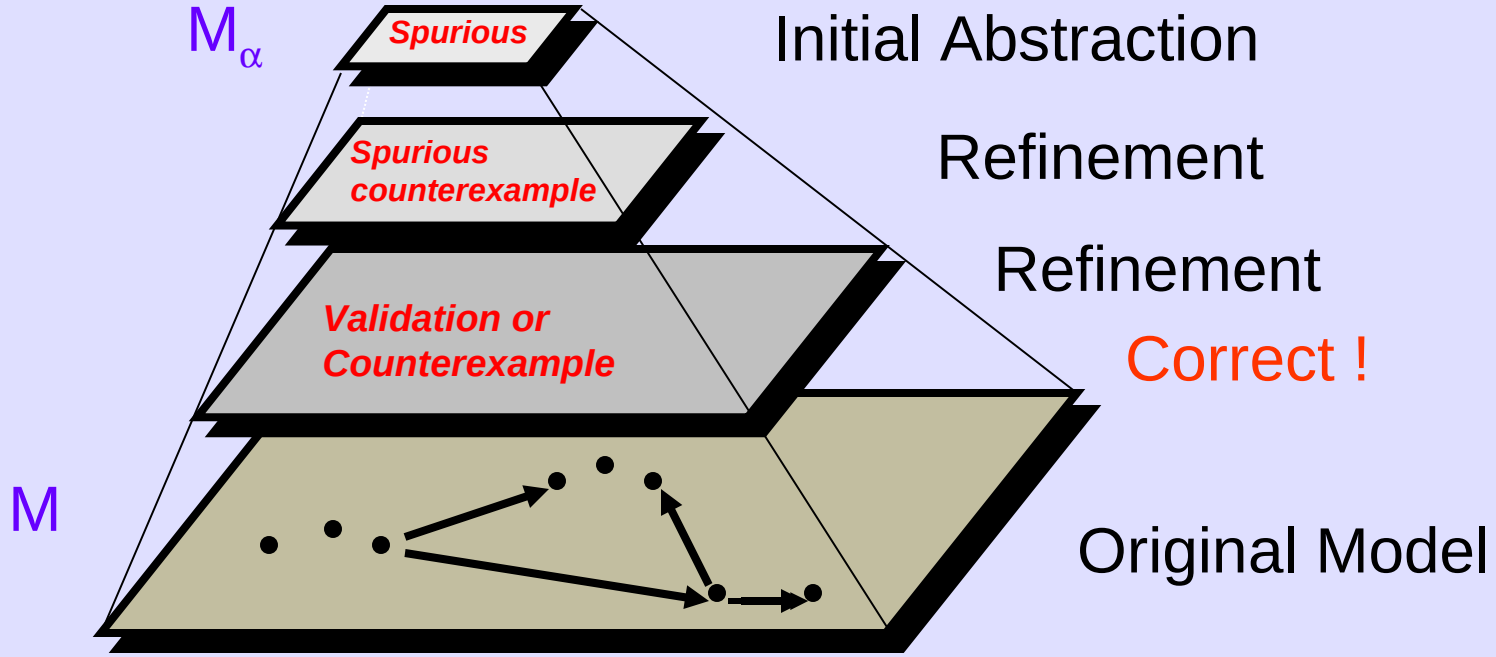
- **State Explosion**

Abstraction too coarse

- **Information Loss**

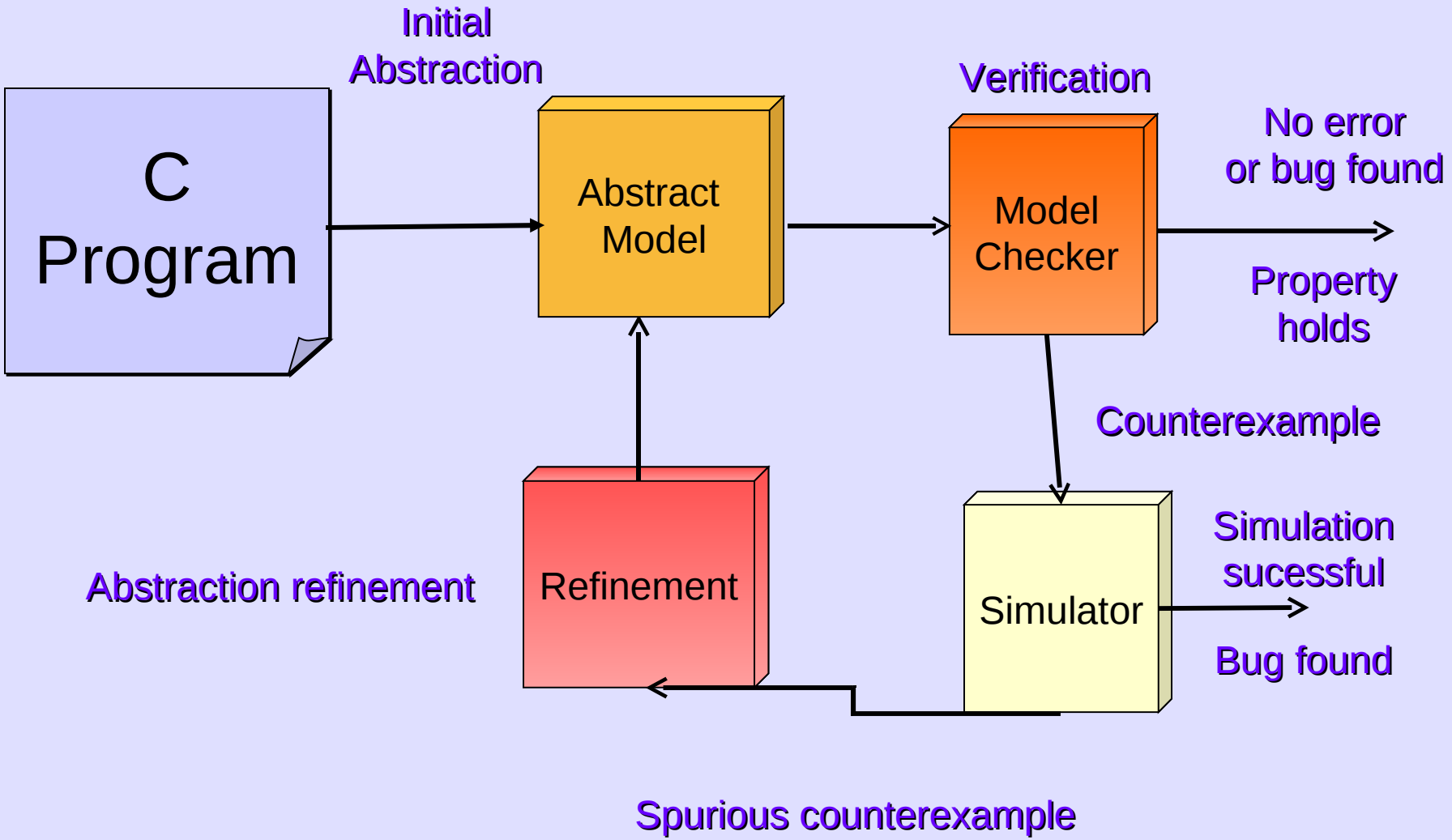
Automatic Abstraction Methodology

Automatic Abstraction



CEGAR

CounterExample-Guided Abstraction Refinement

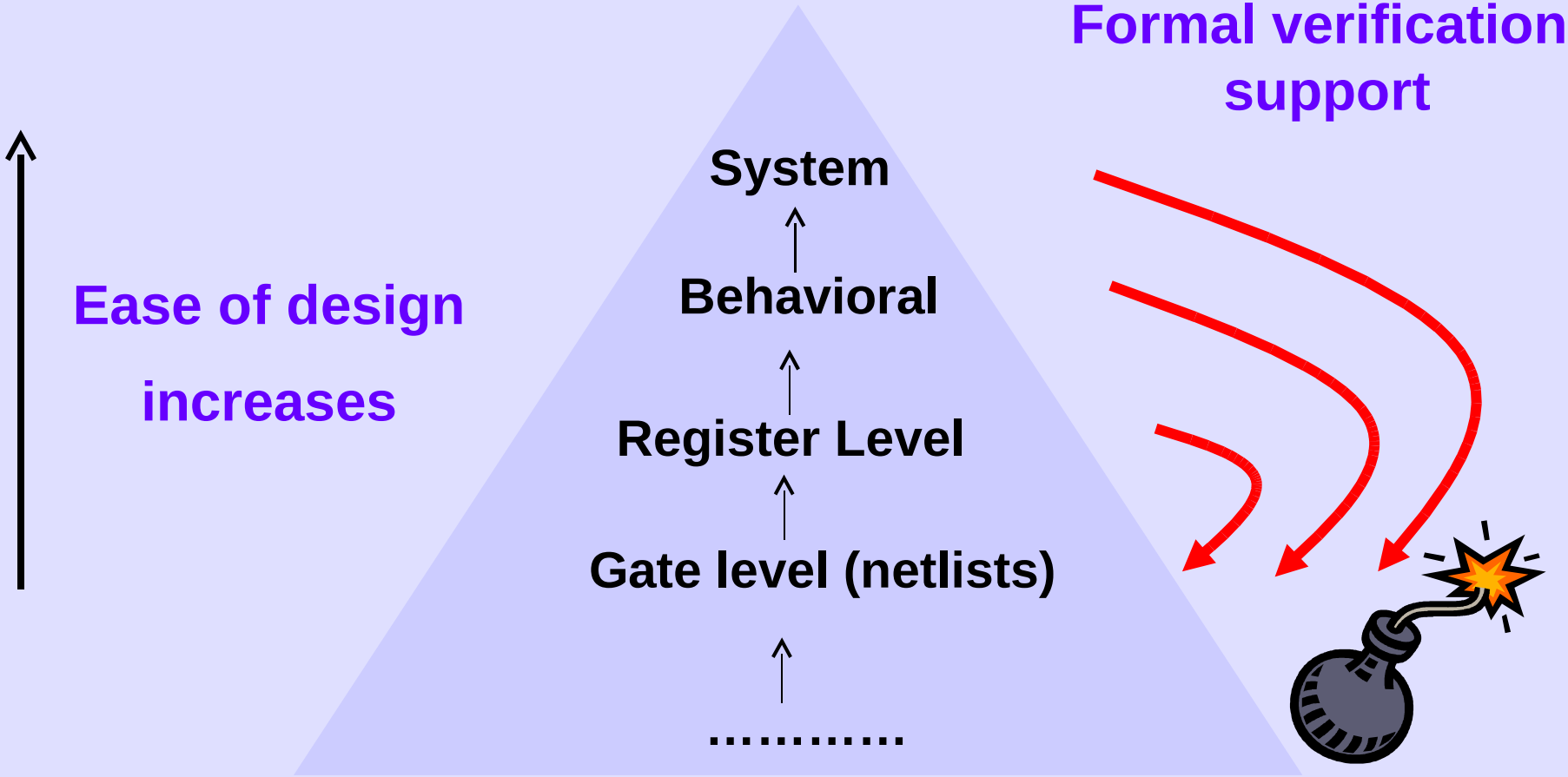


Software Example: Device Driver Code

Also according to Wired News:

"Microsoft has developed a tool called Static Device Verifier or SDV, that uses 'Model Checking' to analyze the source code for Windows drivers and see if the code that the programmer wrote matches a mathematical model of what a Windows device driver should do. If the driver doesn't match the model, the SDV warns that the driver might contain a bug."

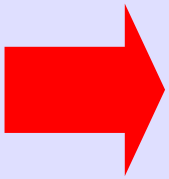
Back to Hardware!



Register Level Verilog:

```
module counter_cell(clk, carry_in,
                                carry_out);
input clk;
input carry_in;
output carry_out;
reg value;
assign carry_out = value & carry_in;
initial value = 0;

always @(posedge clk) begin
// value = (value + carry_in) % 2;
        case(value)
                0: value = carry_in;
                1: if (carry_in ==0)
                        value = 1;
                else value = 0;
        endcase
end
endmodule
```



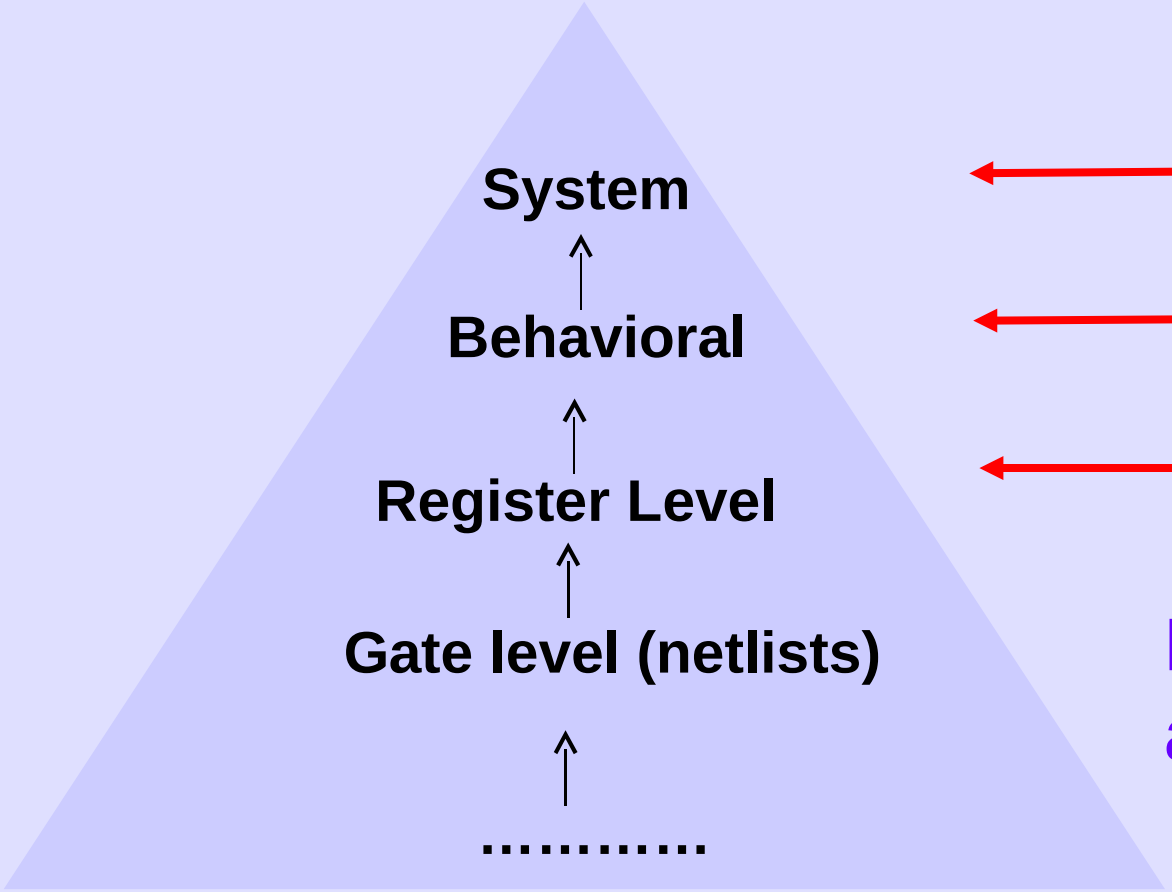
Gate Level (netlist):

```
.model counter_cell
.inputs carry_in
.outputs carry_out

.names value carry_in _n2
.def 0
1 1 1
.names _n2 carry_out$raw_n1
- =_n2
.names value$raw_n3
0
.names _n6
0
.names value _n6 _n7
.def 0
0 1 1
1 0 1
.r value$raw_n3 value
0 0
1 1
```

….. (120 lines)

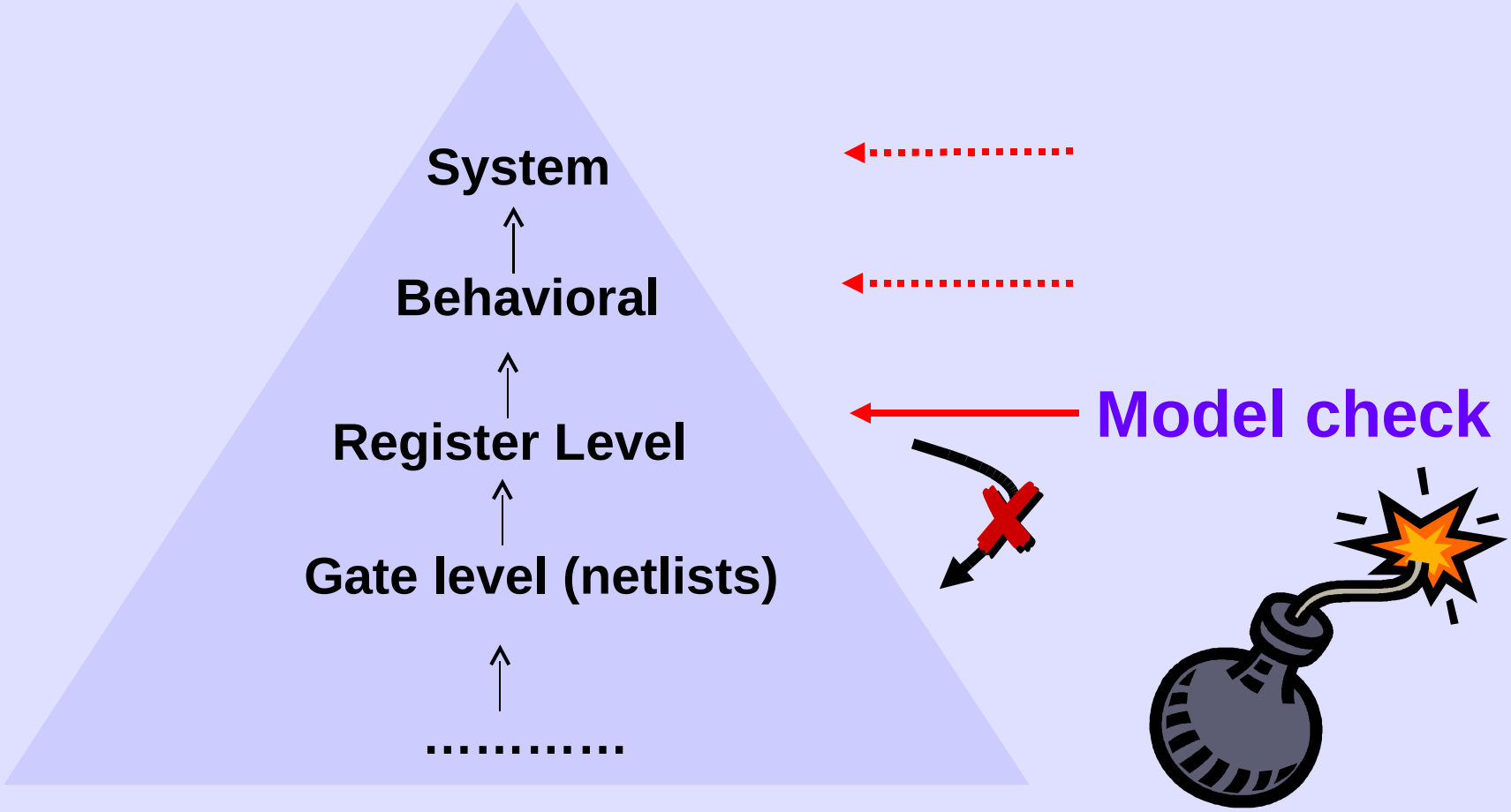
Lack of verification support



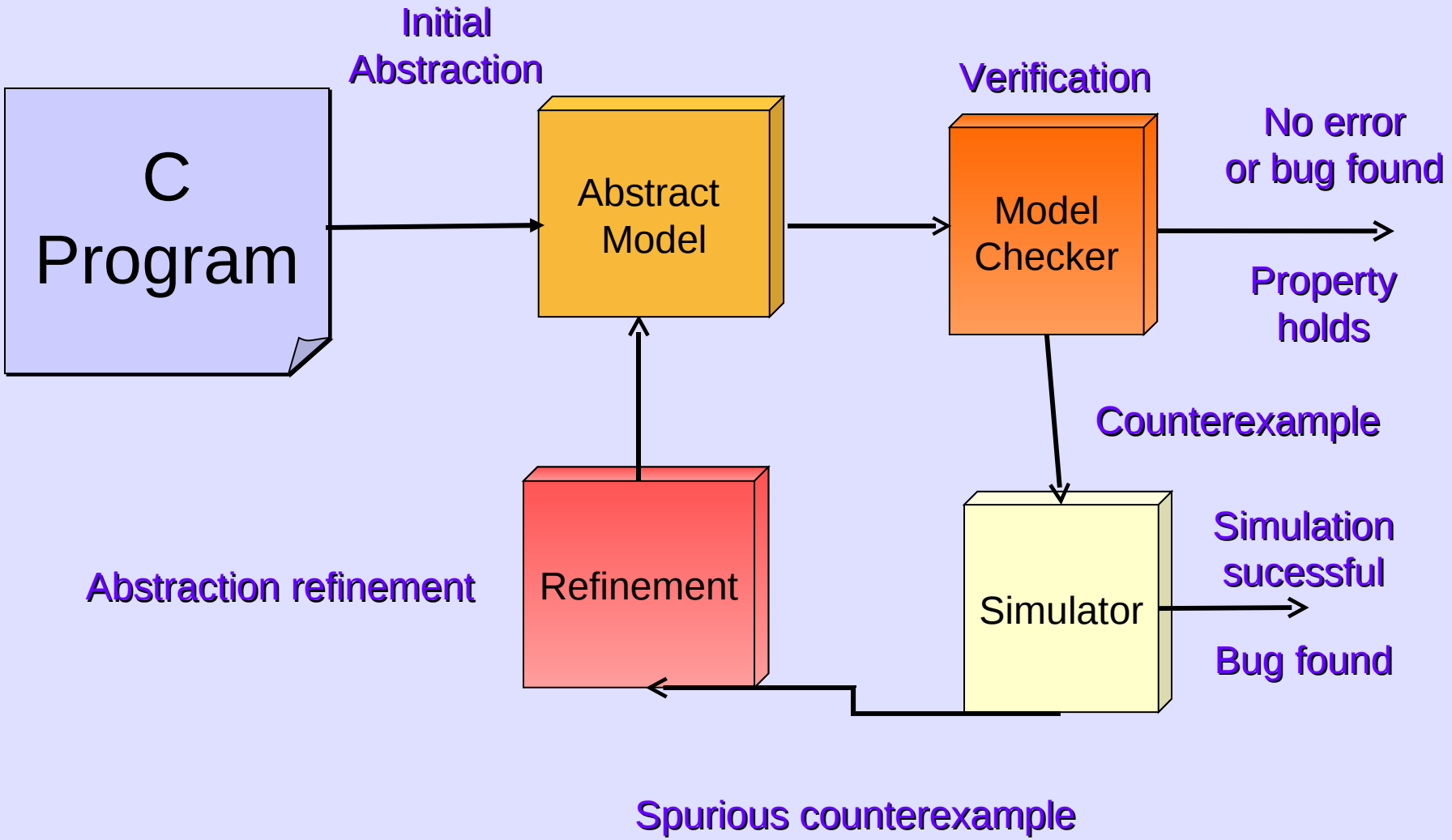
use techniques from **software verification**

Must be automatic and scalable!!

Model Checking at the Register Level

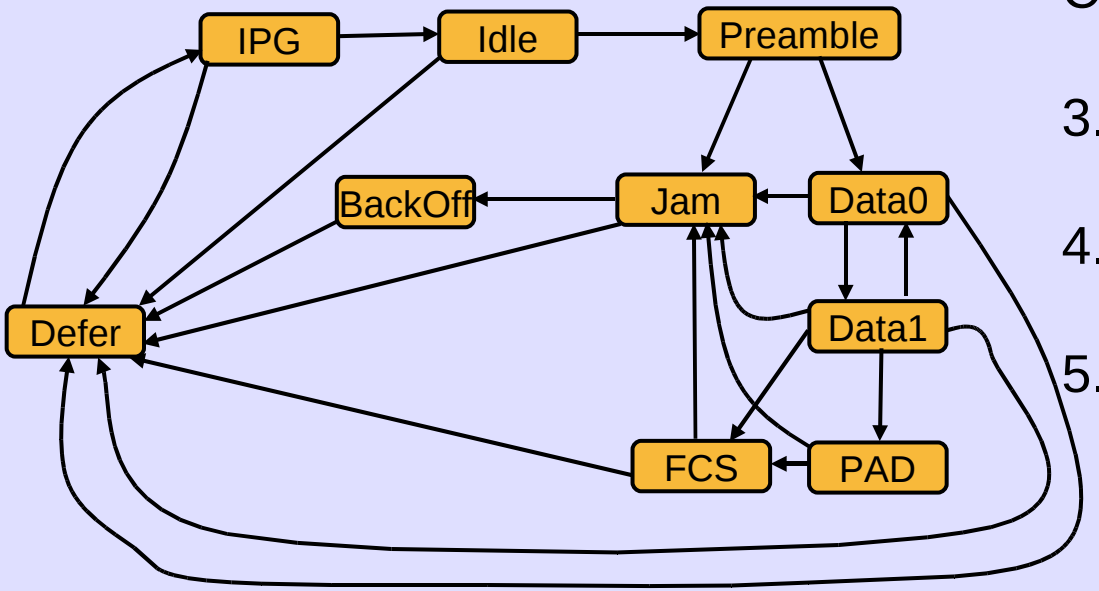


Abstraction-Refinement loop (CEGAR)



Benchmarks

- **Ethernet MAC from opencores.org**
- **5000 lines of RTL Verilog**



Transmit Module In Ethernet MAC
(self-loop on each state not shown)

Checked three properties:

3. Transmit module simulates state machine on left. (ETH0)
4. Checks transitions out of state BackOff (ETH1)
5. Checks transitions out of state Jam (ETH2)

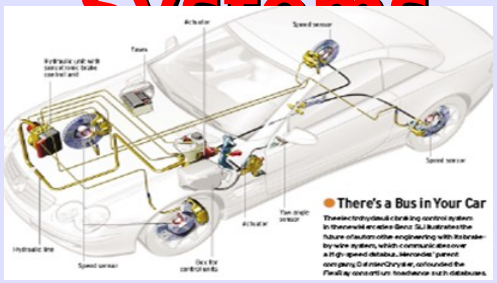
Experimental Results

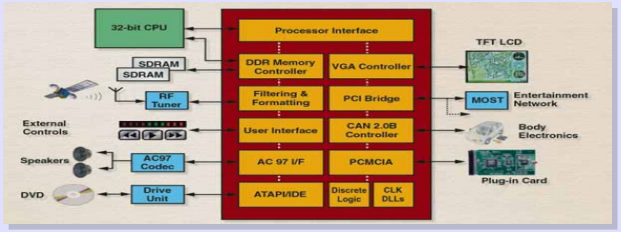
Benchmark	Latches	Time (sec)	#Preds	#Iters
ETH0	359	44	21	55
ETH1	359	127	93	51
ETH2	359	161	94	111

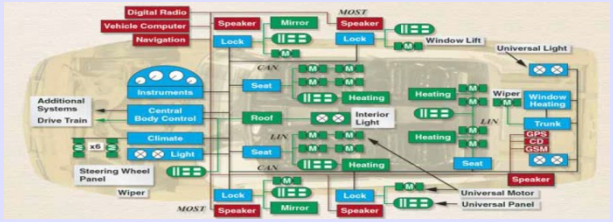
Challenges for the Future

- Exploiting the Power of SAT, Satisfiability Modulo Theories (SMT)
- Compositional Model Checking of both Hardware and Software
- Software Model Checking, Model Checking and Static Analysis
- Verification of Embedded Systems (Timed and Hybrid Automata)
- Model Checking and Theorem Proving (PVS, STEP, SyMP, Maude)
- Probabilistic and Statistical Model Checking
- Interpreting Counterexamples
- Scaling up even more!!

My goal:
Verification of Safety-Critical Embedded Systems







Do you trust your car?

Embedded Systems are as important in Europe as Computer Security is in the U.S.!

Students, Post-docs, and Visitors

Ph.D. Students:

- Sergey Berezin
- Michael Browne
- Jerry Burch
- Sergio Campos
- Sagar Chaki
- Pankaj Chauhan
- David Dill
- Allen Emerson
- Alex Groce
- Anubhav Gupta
- Vicki Hartonas-Garmhausen
- Himanshu Jain
- Sumit Jha
- William Klieber
- David Long
- Yuan Lu
- Dong Wang
- Will Marrero
- Ken McMillan
- Marius Minea
- Bud Mishra
- Christos Nikolaou
- Nishant Sinha
- Prasad Sistla
- Muralidhar Talupur
- Xudong Zhao

Post-docs:

- Constantinos Bartzis
- Armin Biere
- Lei Bu
- David Deharbe
- Alexandre Donze
- Azadeh Farzan
- Ansgar Fehnker
- Wolfgang Heinle
- Tamir Heyman
- James Kapinski
- Daniel Kroening
- Axel Legay
- Daniel Milam
- Alaexandar Nanevski
- Joel Ouaknine
- Karsten Schmidt
- Subash Shankar
- Ofer Strichman
- Prasanna Thati
- Micheal Theobald
- Tayssir Touili
- Helmut Veith
- Silke Wagner
- Karen Yorav
- Haifeng Zhu
- Yunshan Zhu

Visitors:

- Y. Chen
- Y. Feng
- T. Filkorn
- M. Fujita
- P. Granger
- O. Grumberg
- H. Hamaguchi
- H. Hiraishi
- S. Kimura
- S. Krischner
- G.H. Kwon
- X. Li
- A. Platzer
- R. Raimi
- H. Schlingloff
- S. Shanker
- Y.Q. Sun
- T. Tang
- F. Tiplea
- Y. Tsay
- J.P. Vidal
- B. Wang
- F. Wang
- P. Williams
- W. Windsteiger
- Kwang Yi
- T. Yoneda

Questions?