# CDA 5416: Computer System Verification

## Introduction

Hao Zheng

Department of Computer Science and Engineering
University of South Florida
Tampa, FL 33620
Email: haozheng@usf.edu
Phone: (813)974-4757
Fax: (813)974-5456

# Contents

# About This Course

## Definition of Verification (Google)

The process of establishing the *truth, or validity* of *something*

**Objective**: learn *model checking*, an automated techniques for verifying computing systems

- Learn modeling computation and communication of concurrent systems
- Learn formal correctness specification using temporal logics,
- Understand the basic model checking algorithms
- Gain Hand-on experience with widely-used model checkers

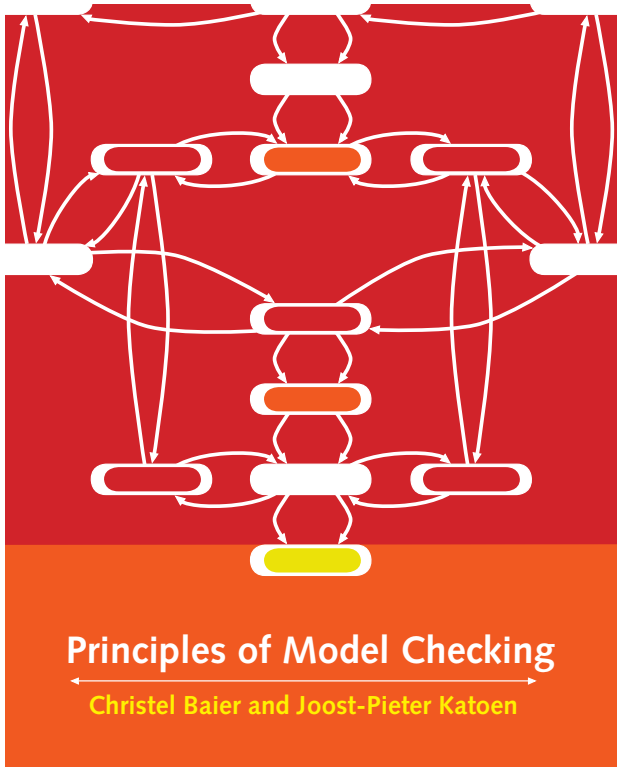
# Contact Information

| | |
|---|---|
| Office Location: | ENB 312 |
| Office Hours: | $1 - 2{:}30pm$, Mon & Wed, or by appointment |
| Course webpage: | Canvas<br>`http://www.cse.usf.edu/~haozheng/teach/cda5416/` |
| Email: | haozheng@usf.edu |
| Phone: | (813) 974-4757 |

# Background Requirements

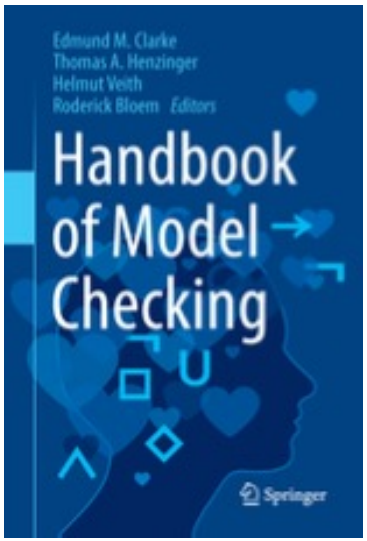
- Topics covered are for HW/SW verification.
  - Basic knowledge of how HW/SW works (logic design, computer architecture, OS, etc).
- Knowledge in automata/first-order logic (Discrete math) is desirable,
  - but we will review the basics as needed.
- Programming skills that might be needed for the final project.

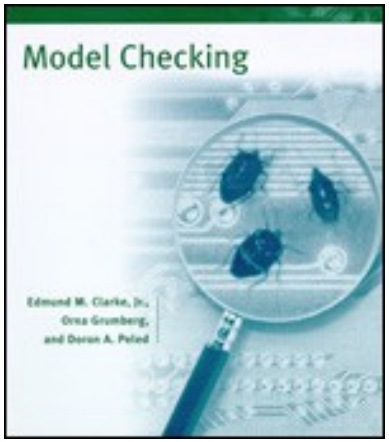
# Textbook

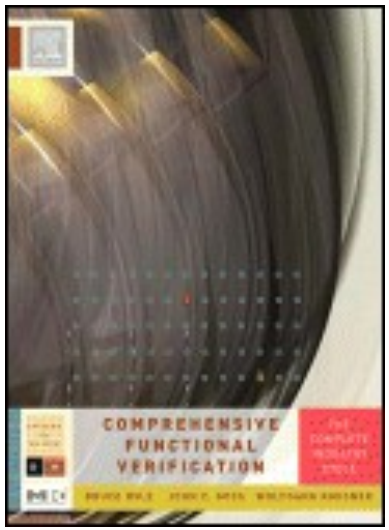


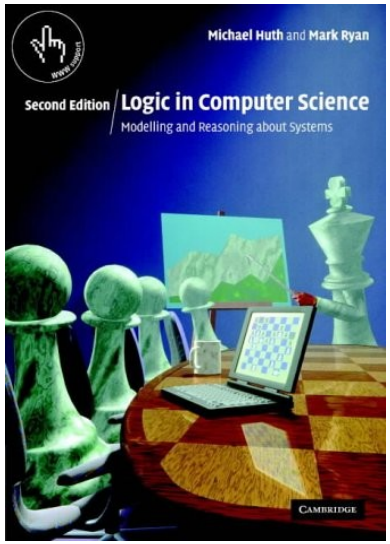

- **Principles of Model Checking** by Christel Baier and Joost-Pieter Katoen MIT Press 2008.
- Lectures borrow much material from the textbook.
- Free on-line access via USF Library

# Books for References

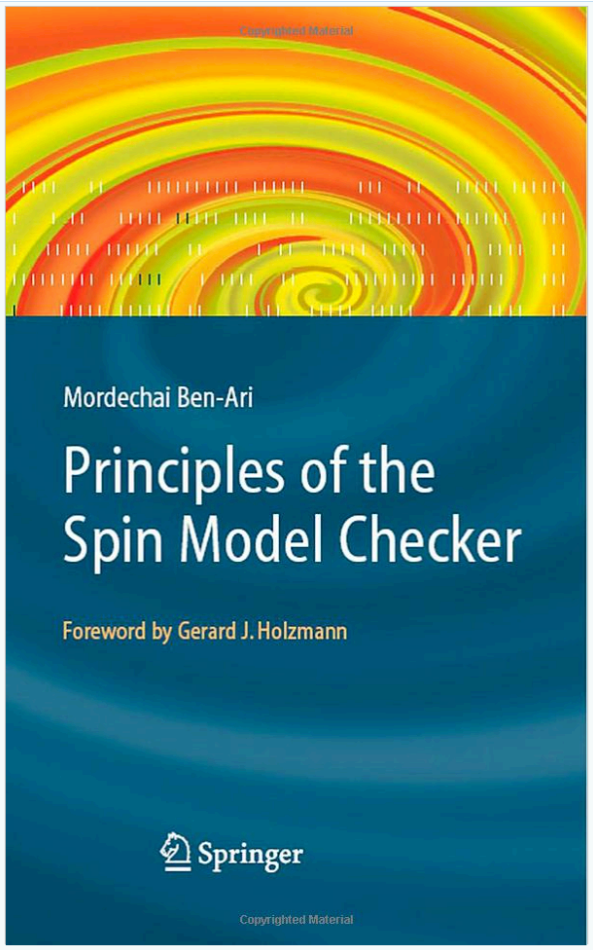


# Another Reference Book




A systematic introduction to the SPIN model checker

# Evaluation

- Grading policy:
  - Homeworks: $40\%$
  - Quizzes: $5\%$
  - Midterm: $25\%$
  - Final Project: $30\%$

- **Final Grade**: suppose your grade is $x\%$.

$$\begin{array}{rcl} 90\% \leq x & : & A \\ 80\% \leq x < 90\% & : & B \\ 70\% \leq x < 80\% & : & C \\ x < 70\% & : & D. \end{array}$$

# Course Communications

- Communications: Canvas at my.usf.edu.
  - Check out grades, announcements, handouts, etc
  - All submissions must be done via Canvas.
    - **Submission using other means will be ignored!**
  - HW solutions and other related information.
  - Additional information can be found on

    `http://www.cse.usf.edu/~haozheng/teach/cda5416/`
- **Clear your email inbox!**
  - You are responsible for not getting emails due the full inbox.
- Request for late submissions and makeup exam:
  - Granted only when proof showing emergency is provided.
  - Exceptions to homework or exam schedules for religious observance will be granted if you let me know at least **one** week ahead!

# Academic Integrity

- Students are expected to be honest and do not cheat.
  - More important, be honest to yourselves.
- Collaboration and discussions are highly encouraged.
- Copying each others work is forbidden.
- Read the university policy at http://www.ugs.usf.edu/catalogs/0809/adadap.htm
- The reward for cheating is **FF**.

# Contents

# A Flight Autopilot

- **Requirement**: The autopilot should avoid collision with other planes.
- **A solution**: When distance is 1km, give warning to other plane and notify the pilot. When distance is 300m, and no changes in the course of other plane were noticed, go up to avoid collision.
- Is this correct?

# A Flight Autopilot

- **Requirement**: The autopilot should avoid collision with other planes.
- **A solution**: When distance is 1km, give warning to other plane and notify the pilot. When distance is 300m, and no changes in the course of other plane were noticed, go up to avoid collision.
- Is this correct?
- *The same SW installed on both planes, and both may be directed to change to the same course again!*

# A Flight Autopilot

- **Requirement**: The autopilot should avoid collision with other planes.
- **A solution**: When distance is 1km, give warning to other plane and notify the pilot. When distance is 300m, and no changes in the course of other plane were noticed, go up to avoid collision.
- Is this correct?
- *The same SW installed on both planes, and both may be directed to change to the same course again!*
- **Deadlock** is a state where all parties are stuck and cannot make further progress.
  - Deadly consequences may occur if the control system deadlocks.

# A SW example

process Inc: **while** true **do if** $x < 200$ **then** $x := x + 1$ **od**
process Dec: **while** true **do if** $x > 0$ **then** $x := x - 1$ **od**
process Reset: **while** true **do if** $x = 200$ **then** $x := 0$ **od**

**Property**: is $x$ always between (including) $0$ and $200$?

# A SW example

process Inc: **while** true **do if** $x < 200$ **then** $x := x + 1$ **od**
process Dec: **while** true **do if** $x > 0$ **then** $x := x - 1$ **od**
process Reset: **while** true **do if** $x = 200$ **then** $x := 0$ **od**

**Property**: is $x$ always between (including) $0$ and $200$?

**Answer**: When $x = 200$, both Dec and Reset are active, ...

# A SW example: SPIN Model (1)

```
int x = 0;

proctype Inc() {
   do :: true -> if :: (x < 200) -> x = x+1 fi od
}
proctype Dec() {
   do :: true -> if :: (x > 0) -> x = x-1 fi od
}
proctype Reset() {
   do :: true -> if :: (x == 200) -> x = 0 fi od
}
```

# A SW example: SPIN Model (2)

```
proctype Check() {
   assert (x >= 0 && x <= 200)
}

init {
   atomic {
              run Inc();
              run Dec();
              run Reset();
              run Check();
} }
```

# A SW example: SPIN Output

```
pan:1: assertion violated ((x>=0)&&(x<=200)) (at depth 1805)
pan: wrote ex1.pml.trail

(Spin Version 6.2.5 -- 3 May 2013)
Warning: Search not completed

Full state space search for:
        never claim             - (none specified)
        assertion violations    +
        cycle checks            - (disabled by -DSAFETY)
        invalid end states      +

State-vector 52 byte, depth reached 4212, errors: 1
    12657 states, stored
    26470 states, matched
    39127 transitions (= stored+matched)
        3 atomic steps
```

# Some High-Profile Bugs

# Pentium FDIV Bug (1994)

- Intel Pentium chip, released in 1994 produced error in floating point division.
- Try $4195835 - \frac{4195835}{3145727} * 3145727$.
  - You would expect $0$.
  - In 94 Pentium, it returns $256$!
- **Cause**: Five entries in the lookup table used for the division algorithm are missing when implemented.
- Bugs only occur after 10th bits to the right of floating point.
- **Cost**: $\$475$ million (part replacements + reputation demage)

# Ariane 5 Explosion (1996)

- In December 1996, the Ariane 5 rocket exploded 40 seconds after take off.
- **Cause**: A software components threw an exception caused by a data conversion from 64-bit floating point to 16-bit signed number.
- The exception handler was not used.
- **Cost**: $\$400$ million payload.

# Thera-25 Radiation Overdose (1985-87)

- Therac-25: a radiation machine for treatment of cancer patients.
- **Cause**: A failure in the control SW caused wrong dosages of x-rays into patients.
- **Cost**: Three patients died as a direct result of this accident.

# AT&T Telephone Network Outage (1990)

- January 1990: problem in New York City leads to 9 hour outage of large parts of U.S. telephone network
- **Cause**: a flaw (wrong interpretation of break statement in C) in the SW embedded in the switches.
- **Cost**: hundreds of millions US$.

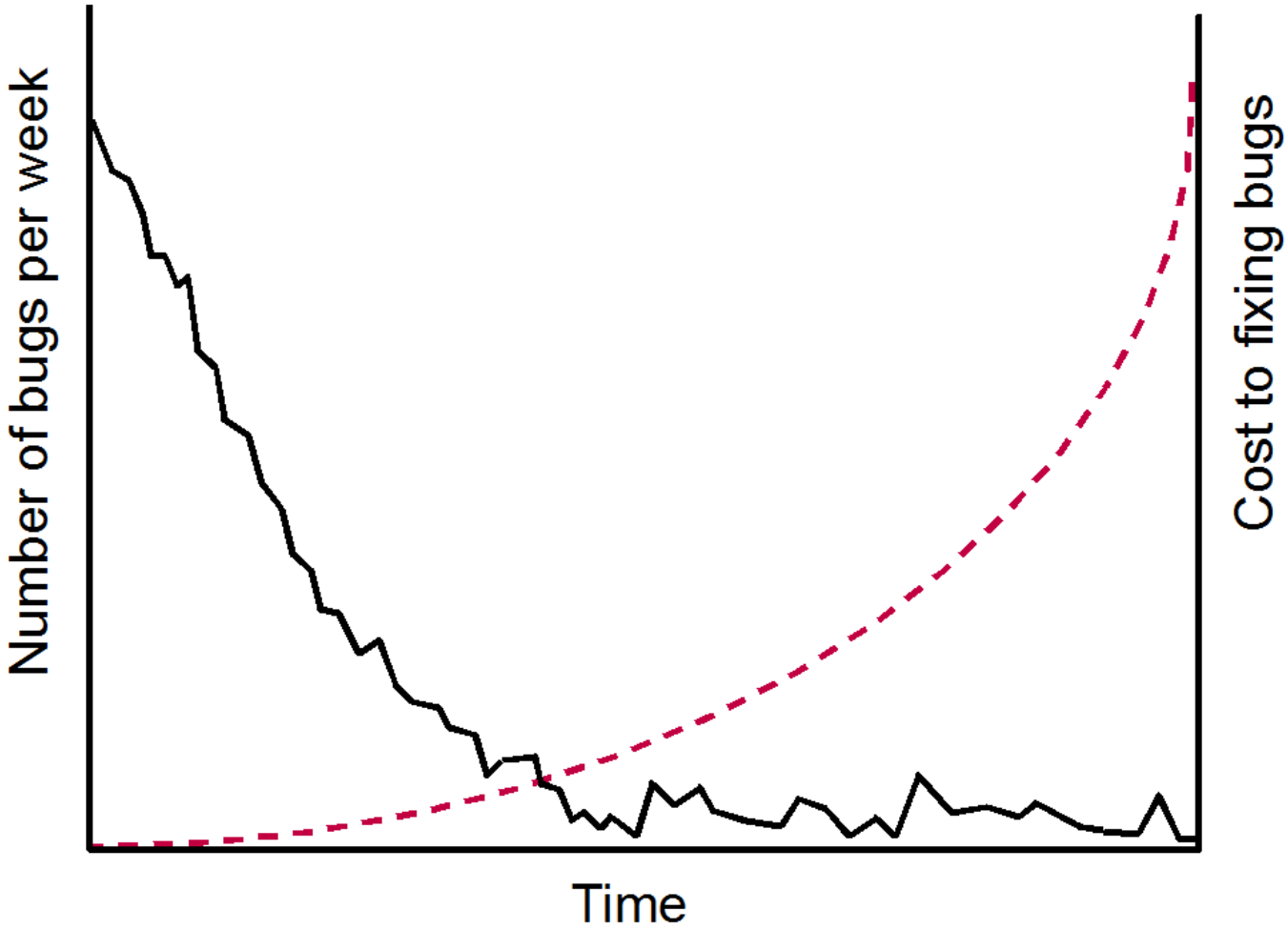
# Lessons from Previous Bugs

- Accidents are often not simple.
- Usually involve complex sequences of interactions among different components in the system, and the operating environment including human beings using the system.
  - Verifying a component is far from being enough.
  - The whole system must be thoroughly verified.
- Verification challenges
  - Huge space of behavior – impossible to verify them completely
  - External events – non-determinism

# Importance of System Correctness

- Computing integrated in various applications
  - Embedded systems
  - Communication protocols
  - transportation systems
  - Manufacturing/process control
- System reliability depends on correctness of HW/SW.
- Defects can be
  - Very expensive for mass-produced products – repair & replacements
  - Fatal for safety-critical systems – loss of human lives
- NIST (National Institute of Standards and Technology) reports software bugs cost $\$60$ billion annually

# Cost of Bugs




The number of design bugs and the cost to fixing them over the course of a design project.

# Verification in Reality

- Some numbers:
  - Verification engineers : design engineer = 3:1.
  - Verification takes $50\% - 70\%$ of design resources.
- The reasons:
  - The longer bugs undetected the costlier to fix them.
  - A bug found early incurs little fixing cost.
  - A bug found after being manufactured may require to repeat the whole design process.
  - A bug slipped into customer's hand can cost hundreds of millions in hardware and brand image.

# Contents

# Types of Verification

- A process that establishes or confirms that a system fulfills its requirements.
- Verification can be classified depending on the attributes:
  - **Functional**
  - Performance
  - Power
  - Reliability

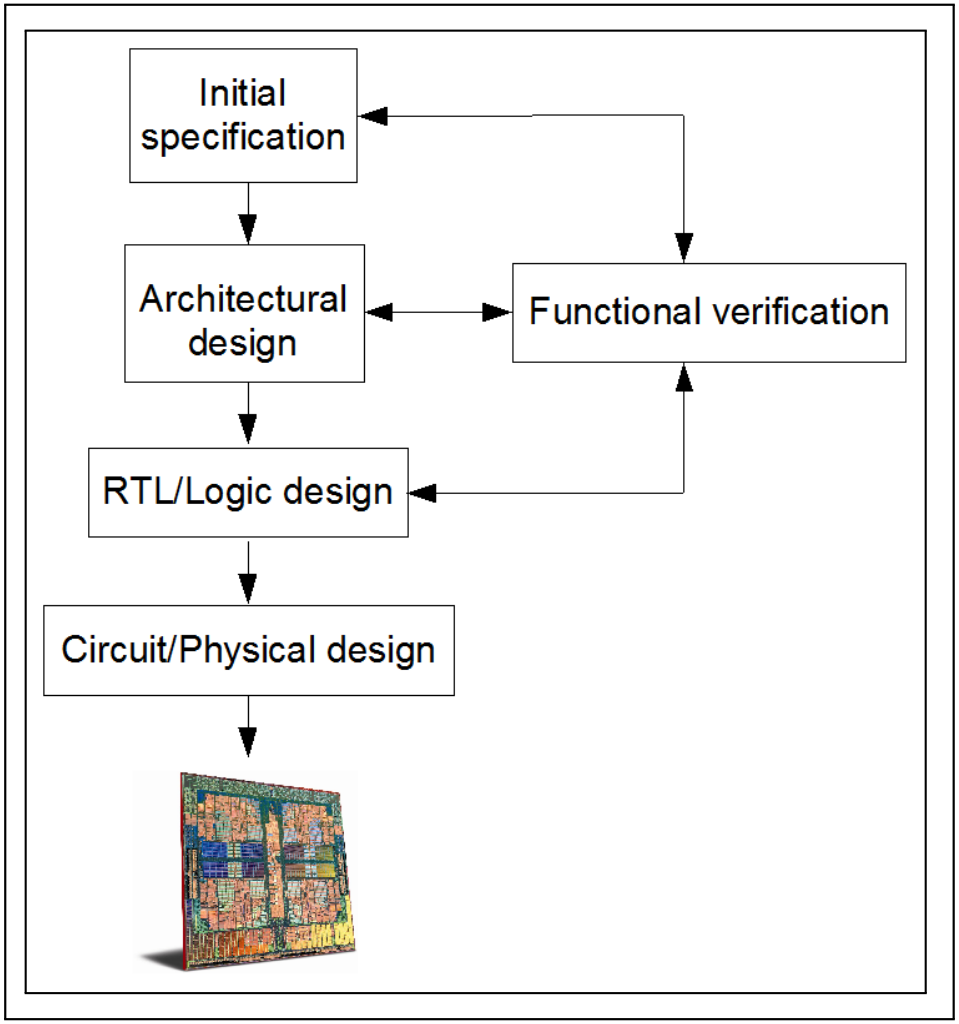
## Verification $\neq$ Validation

1. Verification = check that we are building the the thing right
2. Validation = check that we are building the the right thing

# What is Functional Verification?

- Verification to ensure that the logic behavior of a system meets requirements.
  - Also called **logic** verification.
- Target applications:
  - HW & SW & communication protocols
  - Sequential or concurrent systems
  - Can be found in many important applications such as
    - Digital logic designs,
    - Communication protocols
    - Embedded control systems
- Can be applied to finite or infinite systems
  - Abstraction can reduce infinite state systems to finite ones.

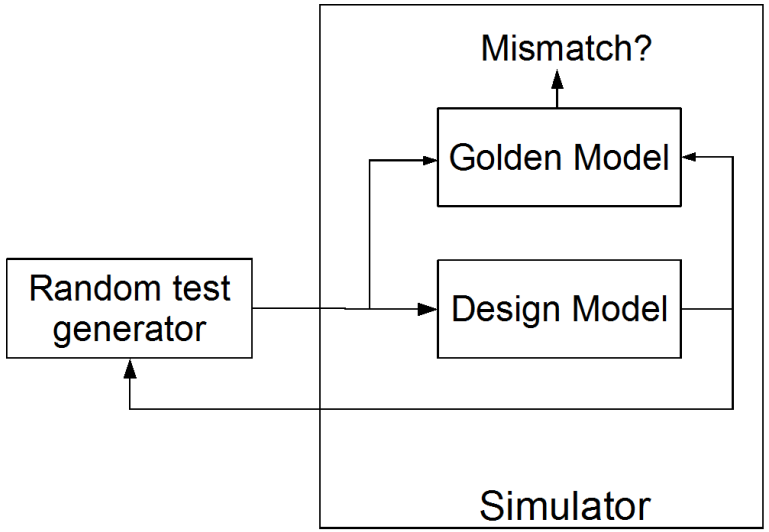
# When to Use Functional Verification




- A system is designed through a sequence of refinement steps.
- Different requirements at different levels.
- System at a lower level must conform to the one at a higher level.

# Functional Verification Techniques

- **Simulation**




- **Testing**: work on real stuff!
- **Logic Emulation**: a design is prototyped with FPGAs
  - Faster, more real testing, and easier for system integration.
  - Less flexible, hard to debug, etc.
- **Formal verification/model checking**
  - Based on mathematic logic foundation.

# Challenges to Verification

- System complexity grows exponentially over time
  - Moore's law says that number of transistors double in every $24$ months.
  - More functions are integrated on a single chip.
- Effectiveness of simulation/testing degrades exponentially.
  - Performance of simulation degrades linearly in system size and number of simulation vectors – too slow for large complex systems.
  - The state space to check grows exponentially at the same time.
  - Not enough input vectors can be simulated with reasonable amount of time.

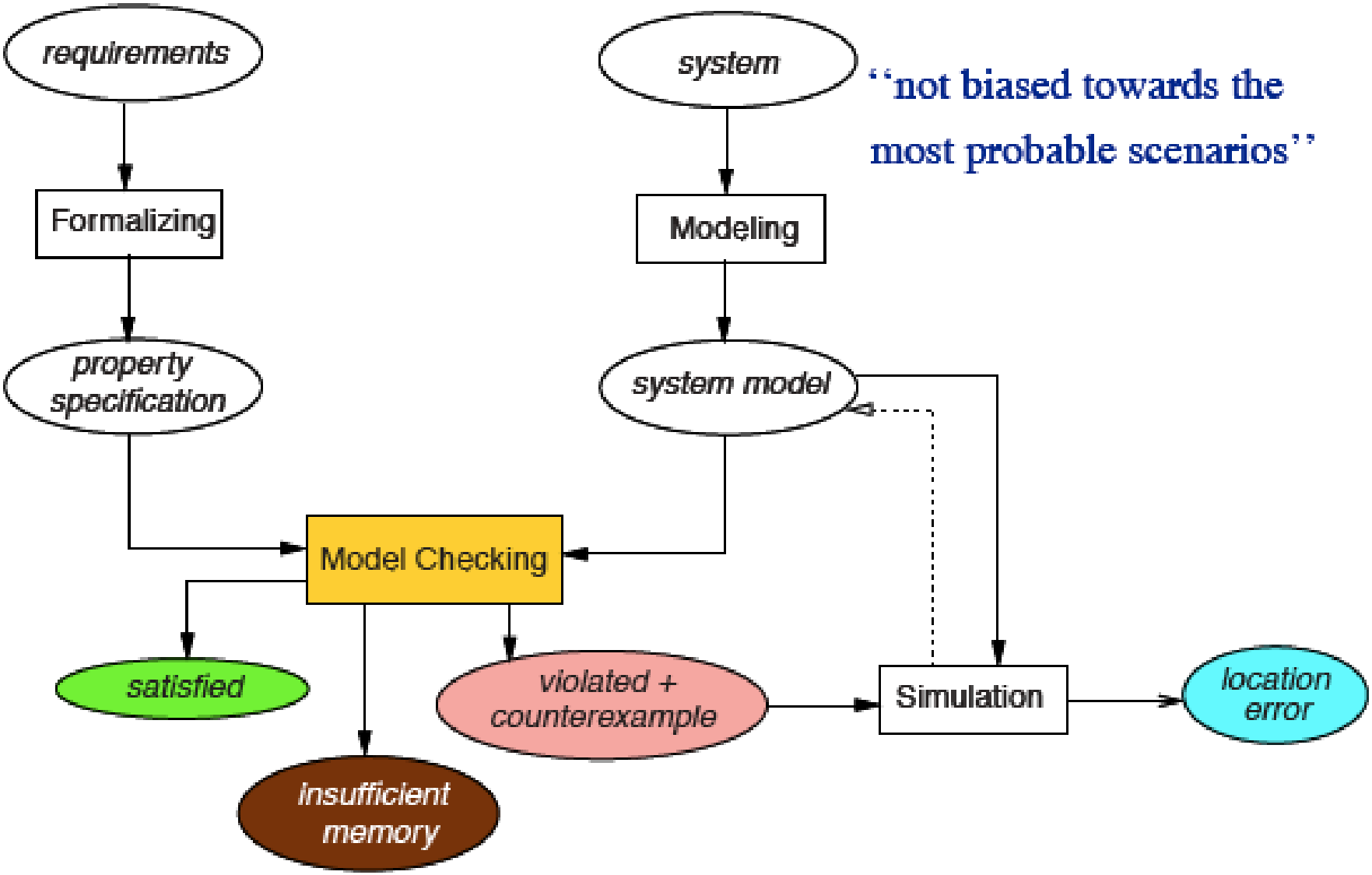
    $\Rightarrow$ Low confidence in system correctness.

# Contents

# Formal Verification

- Applied mathematic logic for modeling and analyzing computing systems.
  - Improve system quality, and reduce verification time.
- Highly recommended by FAA and NASA.
- **Formal Specification**: describe behavior accurately at higher abstraction level.
- **Models**: mathematical objects independent of implementations.
- Approaches
  - Theorem Proving: highly expressive
  - Logic equivalence checking: highly automated
  - Model Checking: automated

# Model Checking: Overview



# Model Checking: Definition

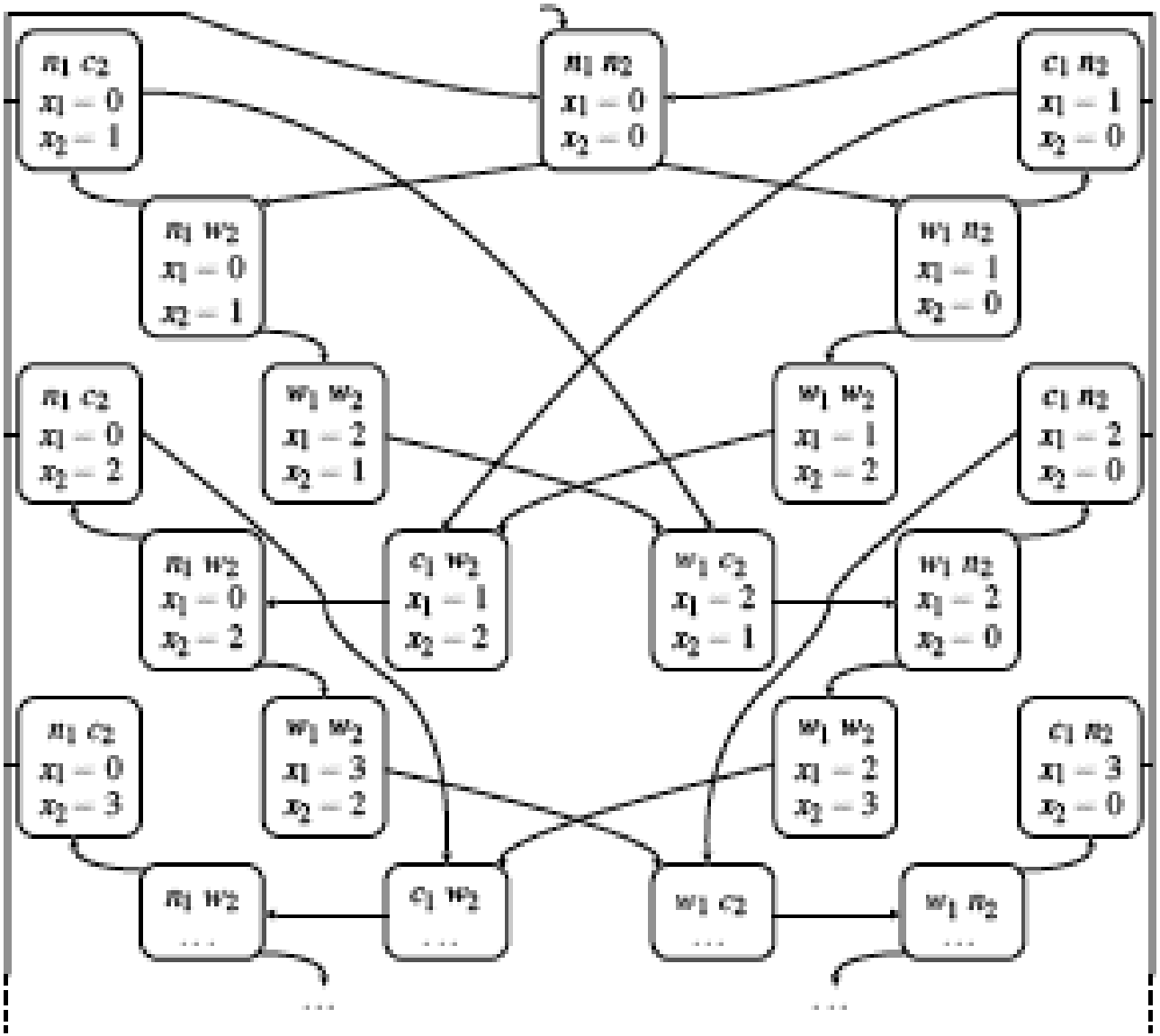
Model checking (MC) is an automatic verification technique that answers yes or no to the following question:

$$M \models f$$

where

- $M$ is a finite state model of the system under verification,
- $f$ is the set of formal properties specifying the correctness requirements.

# What are Models?



# What are Models?

## State transition systems

- States labeled with basic propositions.
- Transition relation between states.

## Generality

- Sequential programs
- Multi-threaded programs
- Communicating processes and protocols
- Hardware circuits
- Biologic systems
- ...

# What are Properties?

- Examples:
  - Can the system reach a deadlock state?
  - Can two processes access a shared resource at the same time?
  - Does the program in the correct state upon termination?

## Classification

- **Safety** properties: nothing bad ever happens.
- **Liveness** properties: good things eventually happen.
- **Fairness**: something happens infinitely often or repeatedly.

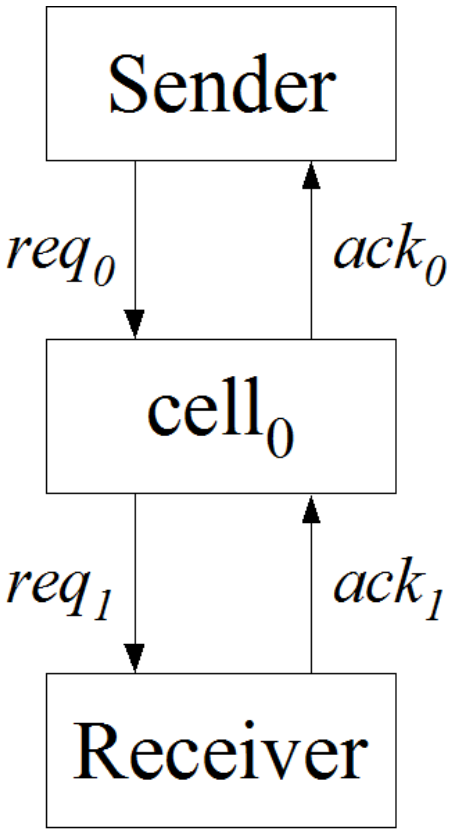
- Specification formalims
  - Temporal logic
  - Automata

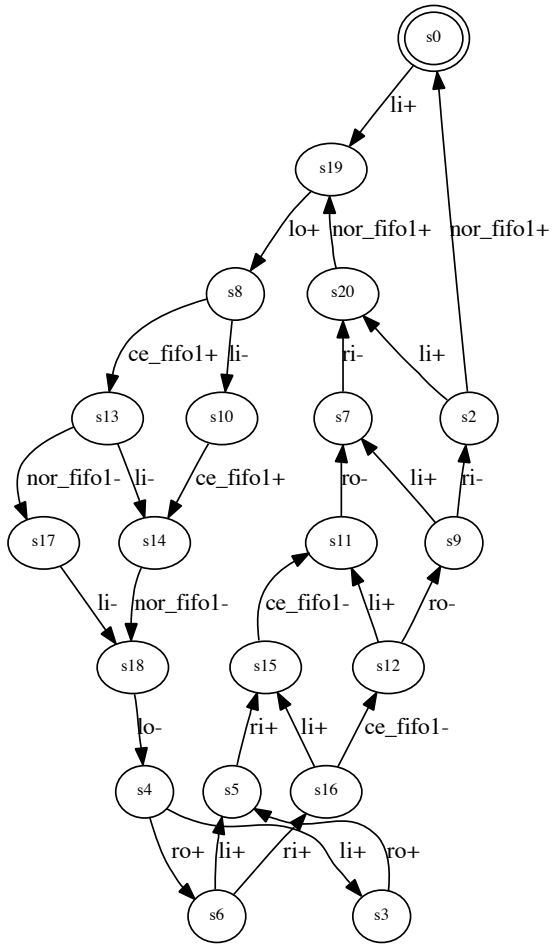
# Advantages of Model Checking

- **Exhaustiveness** (vs simulation)
  - All system states are checked, at least in theory.
  - Not biased to the most possible scenarios (as in testing).
- **Automated and fast** (vs theorem proving)
  - Allows easy integration into the existing design flow.
- **Diagnostic counter-examples** to speed debugging
  - Help to pinpoint source of the bug
- Specification logics easily express many concurrency properties.
  - Concise and rigorous.
- The process of modeling and specification itself can reveal a lot of
  - Incompleteness, ambiguities, and inconsistencies.
- No proofs: why a system is correct is not important, and does not reveal much useful information.
  - Often used as an enhanced debugger.

# State Explosion

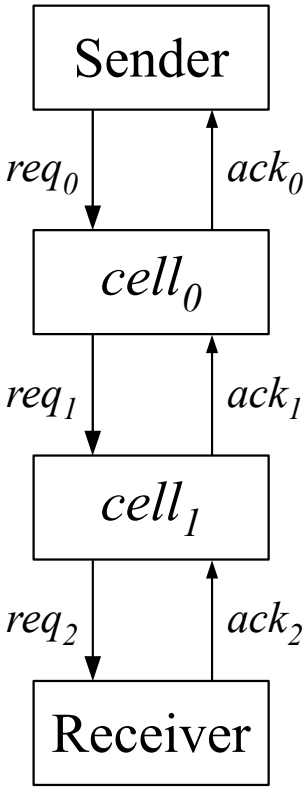
- State space grows exponentially as the size of system description.
  - System state space = product of component state space
  - Available memory cannot keep up with demand.

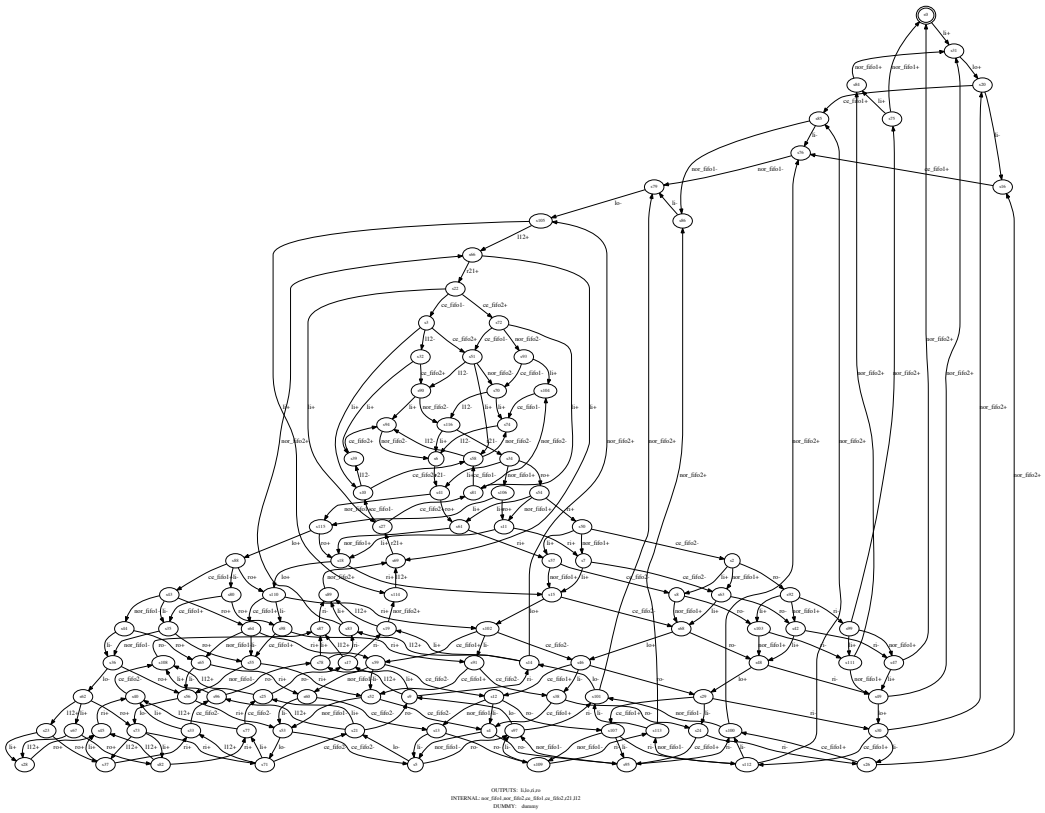


# State Explosion (1)

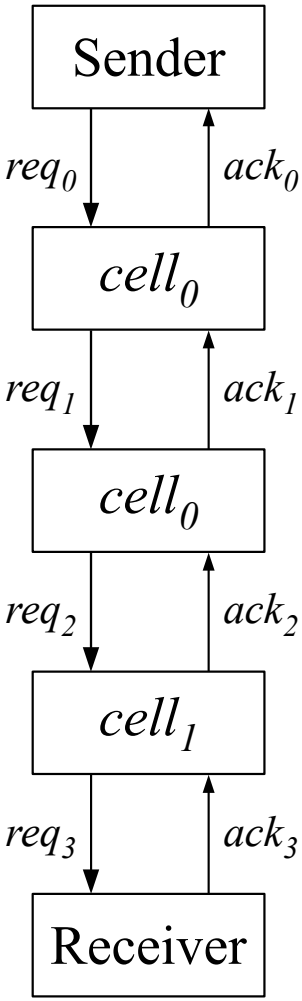
$$|S| = 116, \quad |R| = 240$$

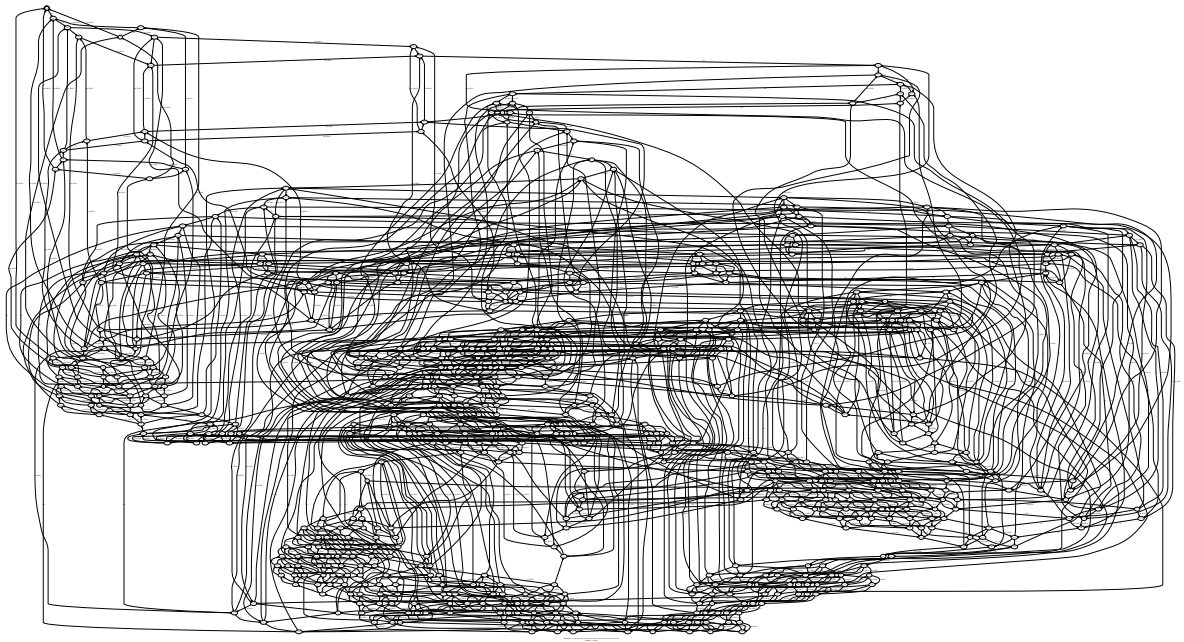


# State Explosion (2)

$$|S| = 644, \quad |R| = 1724$$



# Big Breakthroughs on State Explosion

- **Symbolic model checking**: Burch, Clarke, McMillan, Dill, and Hwang 90, Ken McMillans thesis 92.
  - Encode state space and MC algorithms using Boolean formulas and operations.
- **Partial order reduction**: Valmari 90, Godefroid 90, Peled 94.
  - Mainly used to reduce redundant states in asynchronous design verification.
- **Bounded model checking**: Biere, Cimatti, Clarke, Zhu 99
  - targeted to find bugs of fixed lengths.
  - Use fast SAT solvers at Boolean reasoning engine.
  - Can handle designs of thousands of state variables.
  - There are now many SAT-based unbounded methods.
- **Counter-example guided abstraction refinement (CEGAR)**: Bob Kurshan 1994, Clarke, Grumberg, Jha, Lu, Veith 2000.
  - Used in most software model checkers.

# Success Stories of Model Checking

- Security: Needham-Schroeder encryption protocol
  - Error that remained undiscovered for 17 years unrevealed
- Transportation systems
  - Train model containing $10^{476}$ states
- Model checkers for C, Java and C++
  - Used (and developed) by Microsoft, Digital, NASA
  - Successful application area: device drivers
- Dutch storm surge barrier in Nieuwe Waterweg
- Software in the current/next generation of space missiles
  - NASAs Mars Pathfinder, Deep Space-1, JPL LARS group
- An entire execution cluster in Intel Core i7 (CAV 2009).

# Model Checking Tools

- Industry (Intel, IBM, Motorola) has been using MC more widely for obvious reasons.
- **SMV**: first symbolic model checker, many variants.
- **VIS**: logic synthesis and verification for synchronous circuits.
- **SPIN/LTSA**: an explicit model checker for SW verification.
- **Uppaal/Kronos/ATACS**: real-time system verification.
- **HyTech**: hybrid system verification.
- **Cospan/FormalCheck**: $\omega$-automata/language inclusion.
- **SteP/PVS**: combination of model checking and theorem proving.
- **SLAM**: a project done at Microsoft for device driver verification.

# Contents

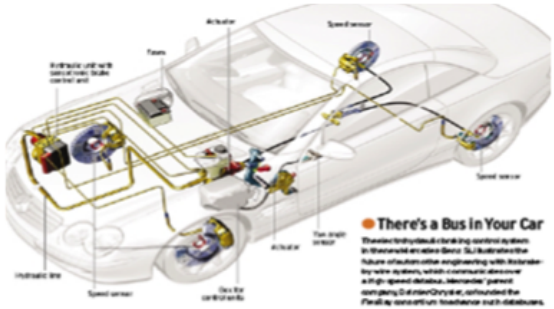
# Topics to be Covered (Tentative)

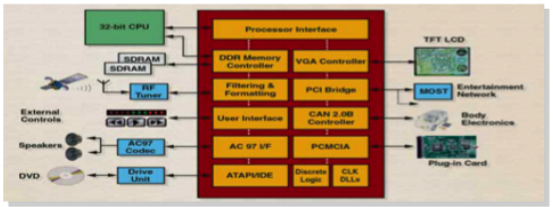
- Introduction to the model checker SPIN
- Modeling concurrent software and hardware systems
- Basics of Linear Time Logic and model checking algorithms
- Introduction to the model checker NuSMV
- Basics of computational-tree logic and symbolic model checking algorithms
- If there is time, introduce Boolean SAT solving and bounded model checking

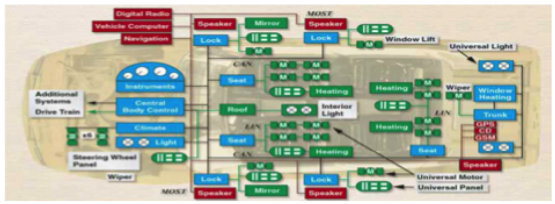
# My Research

Building **S**ecure, **T**rustworthy, **A**utonomous and **R**eliable Embedded (**STAR**) Systems

- Embedded system are pervasive: cars, aircrafts, medical devices, etc.





- **Can we trust cars and aircrafts?**