Model Checking

Accomplishments and Opportunities

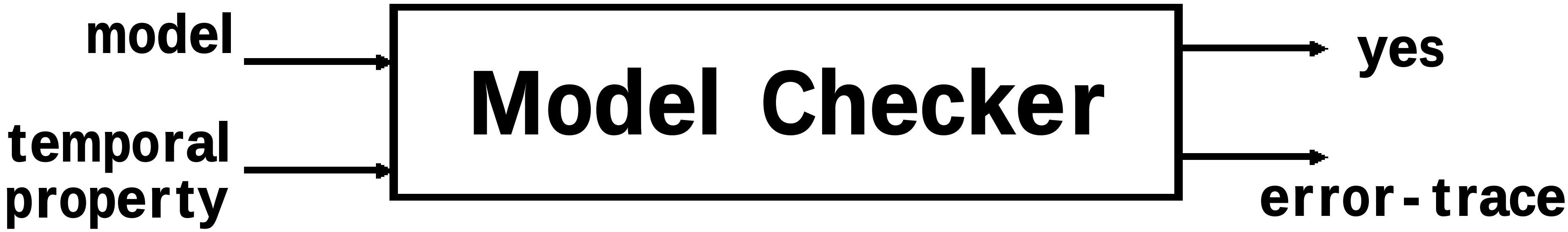
Rajeev Alur
Systems Design Research Lab
University of Pennsylvania
www.cis.upenn.edu/~alur/

Debugging Tools

- **Program Analysis**
  - Type systems, pointer analysis, data-flow analysis
- **Simulation**
  - Effective in discovering bugs in early stages
- **Testing**
  - Expensive!
- **Formal Verification**
  - Mathematical proofs, Not yet practical

Quest for Better Debugging

- Bugs are expensive!
  - Pentium floating point bug, Arian-V disaster
- Testing is expensive!
  - More time than design and implementation
- Safety critical applications
  - Certification mandated



Advantages

Automated formal verification, Effective debugging tool

Moderate industrial success

In-house groups: Intel, Microsoft, Lucent, Motorola...

Commercial model checkers: FormalCheck by Cadence

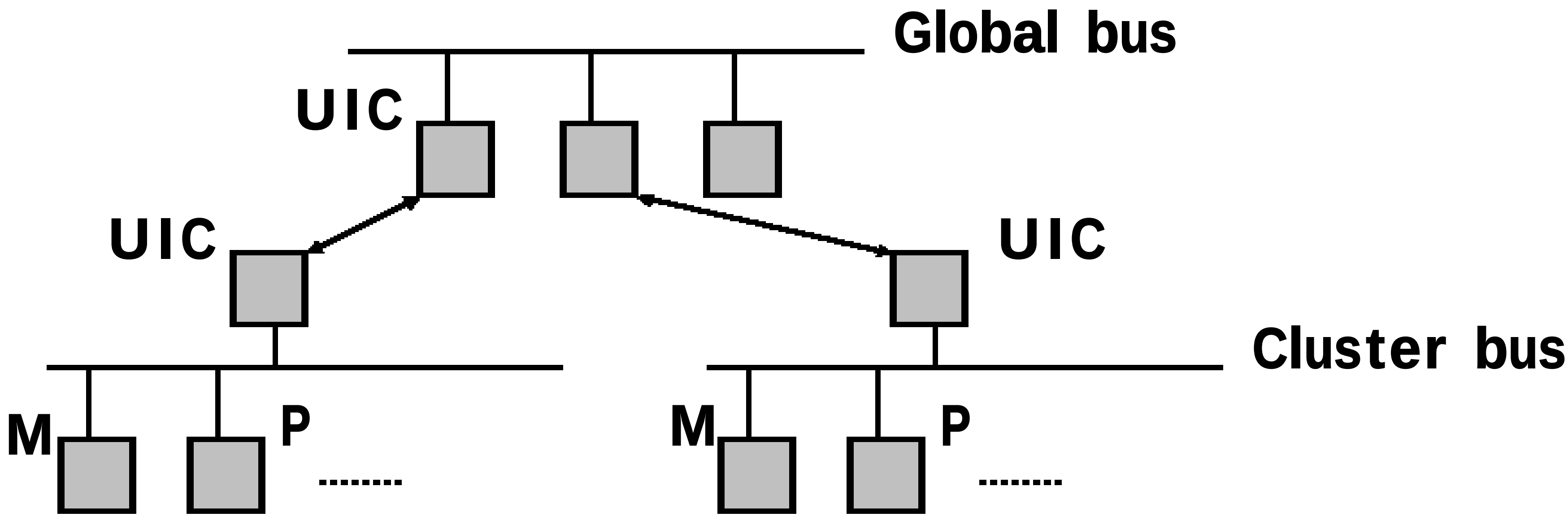
Obstacles

Scalability is still a problem (about 100 state vars)

Effective use requires great expertise

Cache consistency: Gigamax

Real design of a distributed multiprocessor



Read-shared/read-owned/write-invalid/write-shared/...

Deadlock found using SMV

Similar successes: IEEE Futurebus+ standard, network RFCs

Talk Outline

- ✓ Introduction
- ➲ Foundations
- ❑ MOCHA
- ❑ Current Trends and Future

Components of a Model Checker

- **Modeling language**
  - Concurrency, non-determinism, simple data types
- **Requirements language**
  - Invariants, deadlocks, temporal logics
- **Search algorithms**
  - Enumerative vs symbolic + many optimizations
- **Debugging feedback**

Reachability Problem

Model variables X ={x1, ... xn}

- Each var is of finite type, say, boolean

Initialization: I(X) condition over X

Update: T(X,X')

- How new vars X' are related to old vars X as a result of executing one step of the program

Target set: F(X)

Computational problem:

- Can F be satisfied starting with I by repeatedly applying T ?

Graph Search problem

Symbolic Solution

Data type: region to represent state-sets

R:=I(X)

Repeat

If R intersects T report “yes”

Else if R contains Post(R) report “no”

Else R := R union Post(R)

Post(R(X))= (Exists X. R(X) and T(X,X’))[X’ -> X]

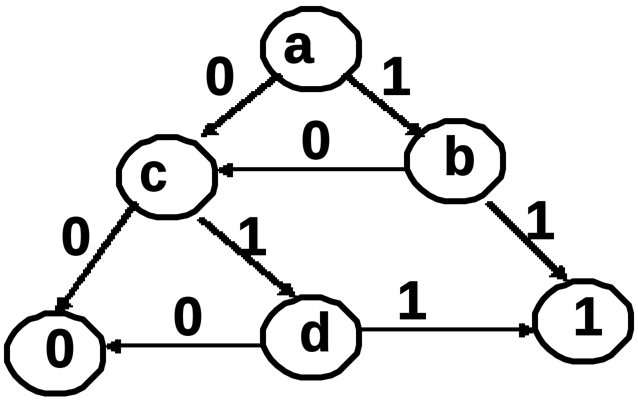
Termination may or may not be guaranteed

Symbolic Representations

- **Necessary operations on Regions**
  - Union
  - Intersection
  - Negation
  - Projection
  - Renaming
  - Equality/containment test
  - Emptiness test
- **Different choices for different classes**
  - BDDs for boolean variables in hardware verification
  - Size of representation as opposed to number of states

Binary Decision Diagrams

Popular representations for Boolean functions



Like a decision graph
No redundant nodes
No isomorphic subgraphs
Variables tested in fixed order

Function: (a and b) or (c and d)

Key properties:
- Canonical!
- Size depends on choice of ordering of variables
- Operations such as union/intersection are efficient

Battling Complexity

- State-space search is expensive!
  - Typical computational complexity: PSPACE
- Symbolic search is a partial solution
  - Running out of memory is the norm
- Secret of success
  - Great flexibility in setting up the problem
  - Abstract many details, and simplify
- Cache coherence
  - Test with 2 processors, 1 bus, 1-bit memory

Requirements

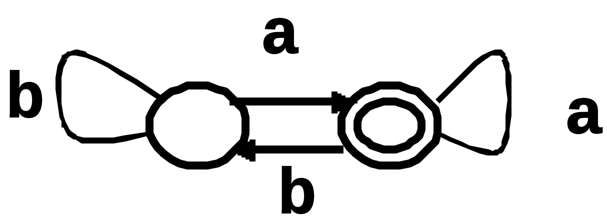
- **Safety properties**
  - Mutual exclusion
  - Deadlock freedom
- **Liveness properties**
  - Every request is followed by response
  - Every reachable state has a path to reset state
- **Temporal logic**
  - Linear-time (LTL) vs Branching-time (CTL)
  - Sample formulas:

    [] (pc1=cs -> pc2!=cs)

    [] (req -> <> response)

Liveness Properties

Beautiful theory of w-regular languages



Buchi automata: Automata accepting infinite words

L(A) = All infinite words over {a,b} with infinitely many a's

Verification of liveness properties:

- Find a reachable cycle satisfying certain properties
- Analysis of strongly connected components
- Nested fixpoint computation

Talk Outline

- ✓ Introduction
- ✓ Foundations
- ➲ MOCHA
- ❑ Current Trends and Future

MOCHA

Goals:

- Exploit design structure for scalable model checking
- Coherent integration of techniques

Key features

- Compositional modeling language: Reactive Modules
- Game-based requirements of open systems: ATL
- Refinement checking by assume-guarantee rules
- Hierarchical reduction algorithms
- Java-based implementation with extensive GUI

Joint project with UC Berkeley, Funded by DARPA/SRC

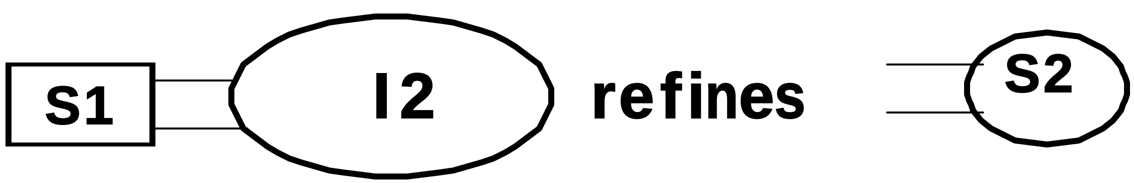
Visit www.cis.upenn.edu/~mocha/

Assume-Guarantee Rule

To prove

It suffices to prove

and

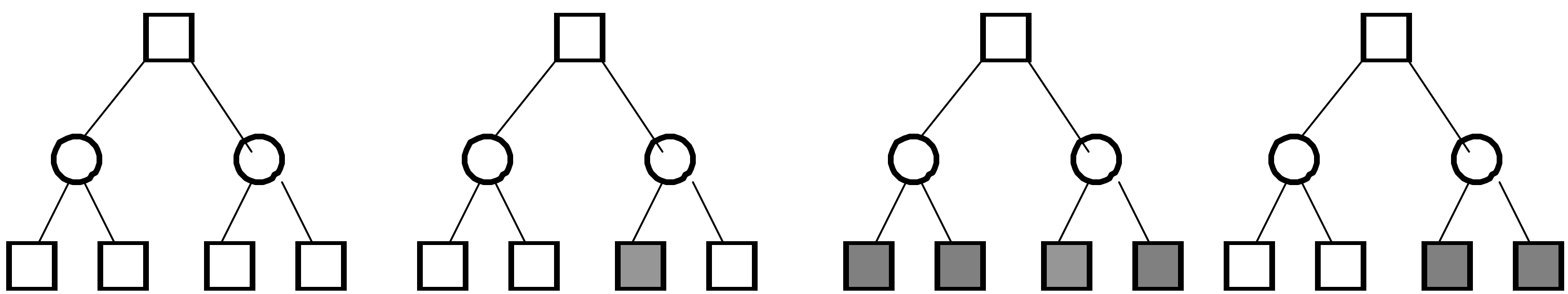


Alternating Temporal Logic

Suitable for requirements of open systems

- explicit distinction between choices of system vs env

Sample game: system and env take turns



EF p **AF p** **<sys> F p**

Alternating Temporal Logic

- In Mocha, multiple players that execute concurrently
- Sample property <A,B> G p
  - can agents A and B collaborate to maintain invariant p?
  - existential over choices of A & B, universal over others
- Can specify games and controllability
- More expressive than CTL
  - model checking via symbolic fixpoint computation

Talk Outline

- ✓ Introduction
- ✓ Foundations
- ✓ MOCHA
- ➲ Current Trends and Future

Current Research Trends

- **Compositional model checking**

  Exploit modularity and hierarchy for efficient analysis

- **Abstraction of programs**

  Automatic extraction of finite-state machines from code (C/Java): Bandera, JavaPathFinder

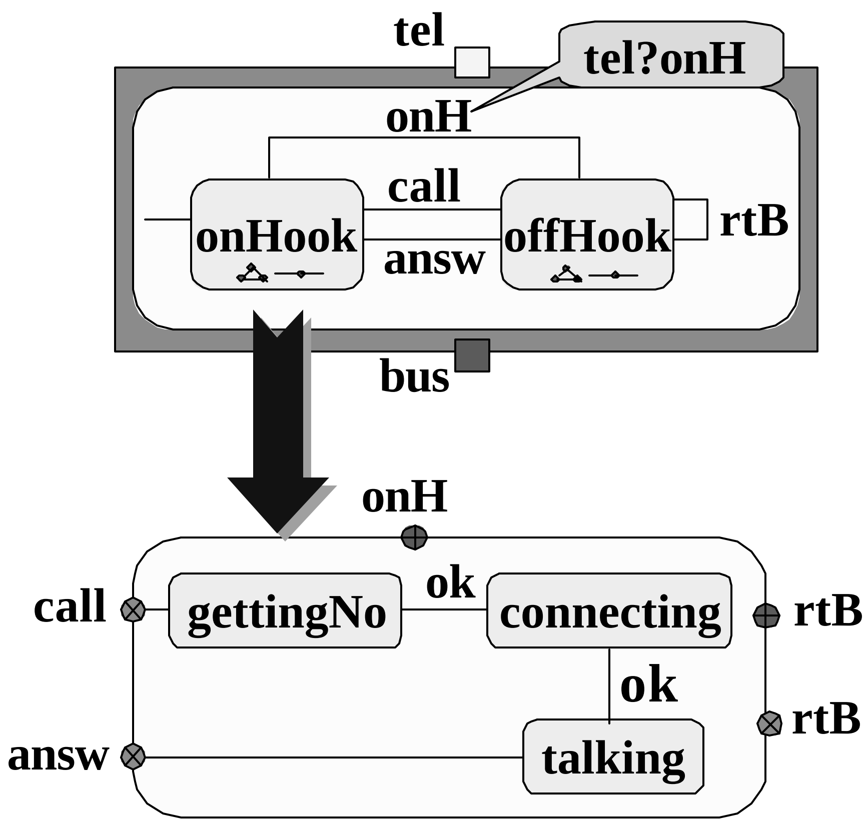
- **Beyond finite-state systems**

  Hybrid systems, Recursive programs…

- **Better Search Technology**

  BDDs + SAT solvers, Decision procedures for other logics (theory of uniterpreted functions with equality)

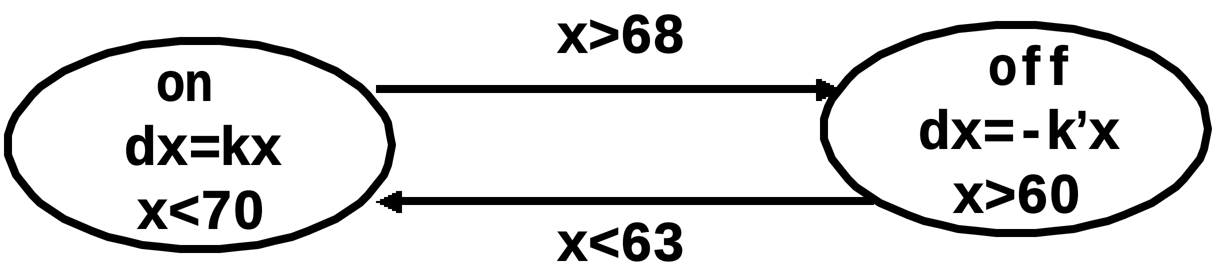
Hierarchical State Machines



HeRMes: How to exploit hierarchy during search?
Use scoping/typing information about variables

Hybrid Systems

State machines + Dynamical systems



Embedded software interacting with physical processes

Analysis of Hybrid Systems

- **Timed Automata**
  - Only continuous variables are timers
  - Can express lower/upper bounds on delays
  - Reachability analysis is decidable
  - Representation for state-sets: Matrices (DBMs)
  - Tools: Cospan, Kronos, Uppaal
- **Linear Hybrid Automata**
  - Dynamics approximated by differential inclusions
  - Expressions in guards/assignments are linear
  - Representation for state-sets: polyhedra
  - Tools: HyTech

Program Abstraction

```
int x, y;
if x>0 {
…………
y:=x+1
……….}
else {
…………
y:=x+1
……….}
```

Predicate Abstraction

bx: x>0; by : y>0

```
bool bx, by;
if bx {
…………
by:=true
……….}
else {
…………
by:={true,false}
……….}
```

Successful applications:

Lucent: Pathstar switch

NASA: Space shuttle control

Emerging Trends

- Past success: hardware and protocols
  - Model-based/principled design methodology in place
- Improved computing technology
  - Greater speed, more memory
- Model-based software design
  - UML
- Embedded software
  - Small and critical

Long-Term Future

- **Problem is REAL!!**

  System design methodology will constantly evolve

- **Model-based design of Systems-on-chip**

  Precise specs of interface behavior

- **Next-generation programming languages**

  Will be designed with model checking as a concern, and will support some checks based on it

- **Embedded software**

  Key app with special-purpose tools

Perspectives on Model Checking

- **Theoreticians**

  Automata + Logic + Graphs

- **Tool Builders**

  Optimizations + Memory management

- **Verification Engineers**

  Abstractions + Expertise + Frustration

- **Enterpreneurs**

  Tools don’t sell, Cost-benefits tradeoff unclear