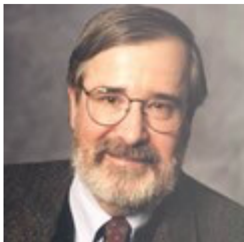
Model Checking

Ren Hao, Yue Guan, Yunchao Liu, Guangyi Cao

2007 Turing Award



Edmund Melson Clarke won the **2007 Turing award** for their role in **developing Model-Checking** into a highly effective verification technology that is widely adopted in the hardware and software industries.

Dr. Clarke is the FORE Systems Professor of Computer Science and Professor of Electrical and Computer Engineering at Carnegie Mellon University. He is a Fellow of **ACM** and the IEEE Computer Society, and was elected to the National Academy of Engineering in 2005.

Demo

Tennis Game

Two players: A and B

Rules

- The order of winning scores is 0, 15, 30, 40.
- Who first won 40 means who won this game.

Demo

Combine the tennis game with model checking.

Under the practice, we used:

- *CSP(Communicating Sequential Process, a kind of formal language)*
- *FDR3(a CSP Refinement Checker, which who ACM system software award on 2001)*

to implement model checking.

Initial definition of this tennis game:

```
IncA(AdvantageA) = gameA -> Game(NUM.(0,0))
IncA(NUM.(40,_)) = gameA -> Game(NUM.(0,0))
IncA(AdvantageB) = Game(Deuce)
IncA(Deuce) = Game(AdvantageA)
IncA(NUM.(30,40)) = Game(Deuce)
IncA(NUM.(x,y)) =  Game(NUM.(next(x),y))

IncB(AdvantageB) = gameB -> Game(NUM.(0,0))
IncB(NUM.(_,40)) = gameB -> Game(NUM.(0,0))
IncB(AdvantageA) = Game(Deuce)
IncB(Deuce) = Game(AdvantageB)
IncB(NUM.(40,30)) = Game(Deuce)
IncB(NUM.(x,y)) = Game(NUM.(x,next(y)))
```

Test

1. assert Scorer [T= STOP ***(Test succeed)***

Let’s See the Result

2. assert Scorer [T= Game(NUM.(15,0)) ***(Test failed)***

Let’s See the Result

Model Checking Content

Introduction of Model Checking

Formal Verification

Transition Systems

Temporal Logic

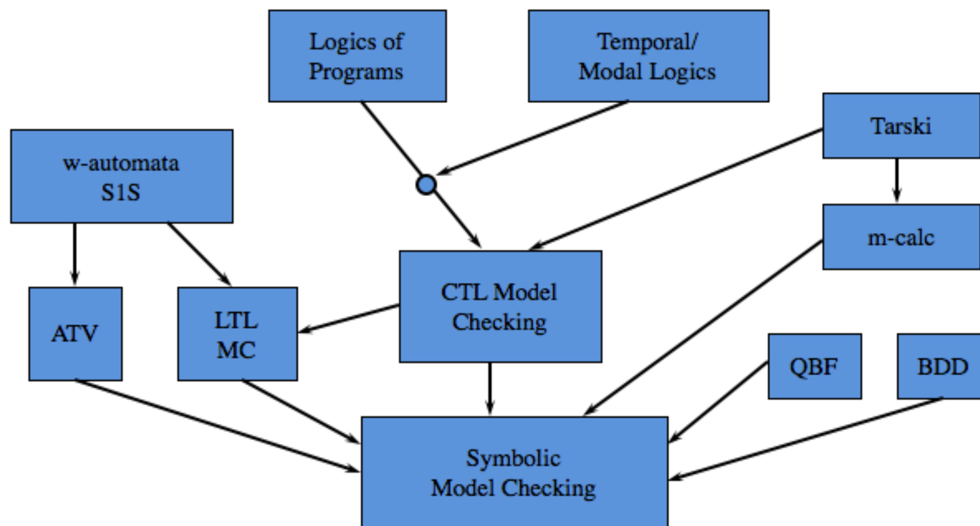
ω-Automata

Symbolic Model Checking

Implementation, Issues and Enhancement

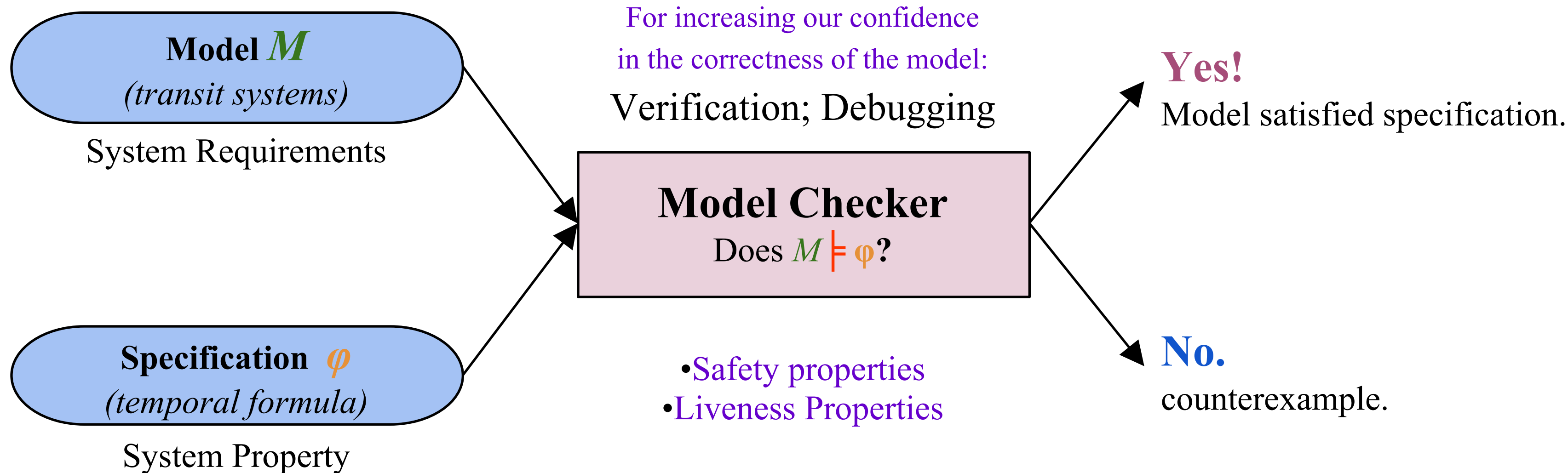
Introduction to Model Checking

- A successful approach of *verifying* requirements
- *Automatic* model based and property verification
- Used for *concurrent and reactive* systems



Model Checking Process

Process satisfy system requirement (model) and property (specification) of final system and generate outputs “Yes” if satisfy or counterexample if not.



Formal Verification

Model checking is one method of formal verification

Given

- a model of a (hardware or software) system
- a formal specification

does the system model satisfy the specification?

Not decidable!

To enable automation, we restrict the problem to a decidable one:

- **Finite-state** reactive systems
- **Propositional** temporal logics

Formal Verification

Why formal verification?

Safety-critical applications: Bugs are unacceptable!

–*Air-traffic controllers*

–*Medical equipment*

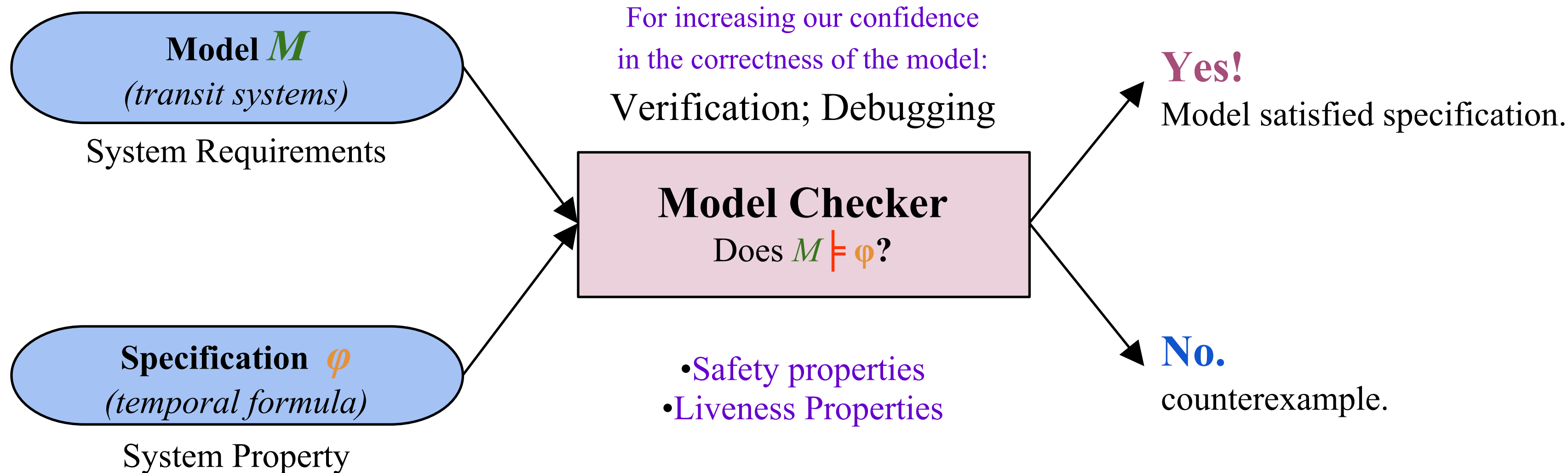
–*Cars*

- Bugs found in later stages of design are expensive, e.g. Intel's Pentium bug in floating-point division
- Hardware and software systems grow in size and complexity: Subtle errors are hard to find by testing
- Pressure to reduce time-to-market

Automated tools for formal verification are needed

Model Checking Process

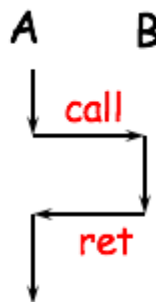
Process satisfy system requirement (model) and property (specification) of final system and generate outputs “Yes” if satisfy or counterexample if not.

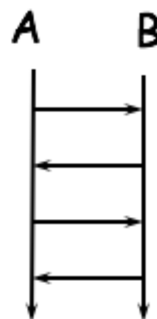


Transit System

concurrent and reactive systems
(classified as distributed systems)

- *Communicate by message passing*
- *Concurrent systems-shared variables*
- *Concurrent processes-shared clock*



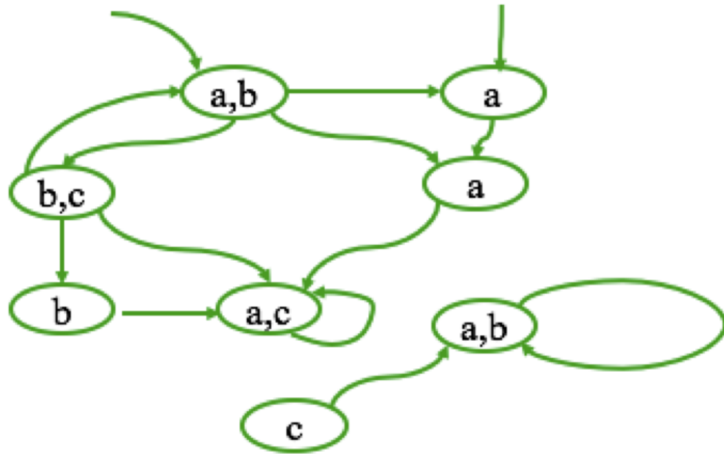


Execute in:

- lock-step *(time-synchronously systems)* or operate *asynchronously*
- sharing a common processor

Defining Model (Kripke Structure/transit system)

Kripke structure / transition system



•Kripke structure $M = (S, I, R(\delta), L)$

–S : finite set of possible global states

–I : set of initial states

–$R(\delta)$: set of transitions relations

–L: labeling function, associates each state with a subset of atomic propositions AP

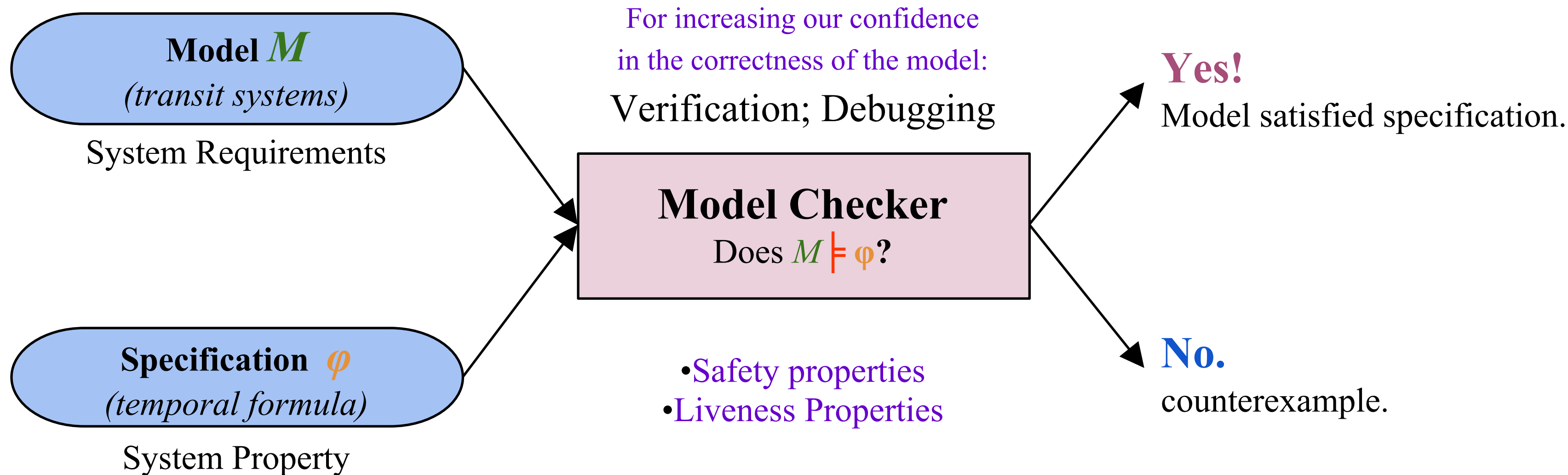
Defining Model (Kripke Structure/transit system)

Model checking problem: A ***model checker*** **checks** whether a system, interpreted as an automaton, is a (Kripke) **model** of a property expressed as a temporal logic formula.

$$\text{Does } M \models \varphi ?$$

Model Checking Process

Process satisfy system requirement (model) and property (specification) of final system and generate outputs "Yes" if satisfy or counterexample if not.



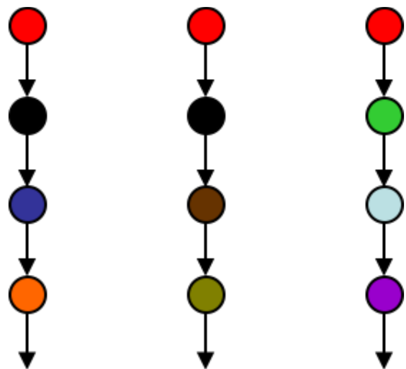
Temporal Logic

- Undesired states such as deadlock, a violation of mutual exclusion etc.
- Some “desired” state is never reached
- Some action never executed
- Initial system reachable? Able to reset

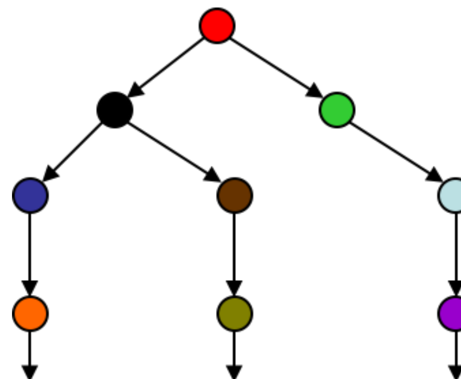
Properties of transition systems are expressed in temporal logic.

Temporary Logic

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



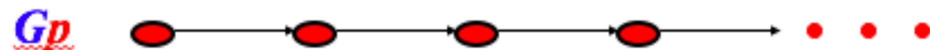
Branching Time
Every moment has several successors
Infinite tree
Computation Tree Logic(CTL)

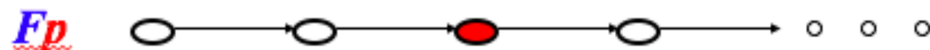


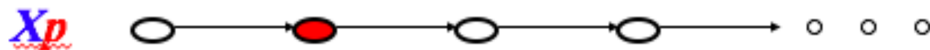
Proportional LTL

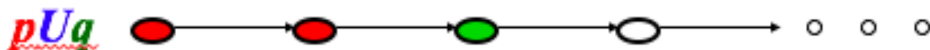
AP – a set of atomic propositions

Temporal operators:









Path quantifiers: ***A*** for **all** path

E there **exists** a path

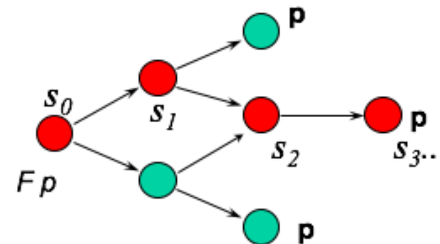
A path in model $M = (S,R,L)$ is a sequence

$$\sigma = s_0, s_1, s_2 \ldots \in S^*$$

$$M, s_0 \models f$$

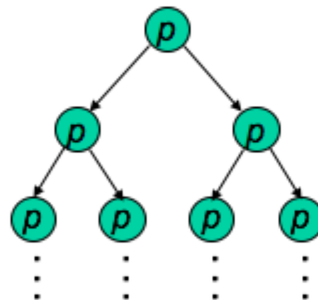
iff

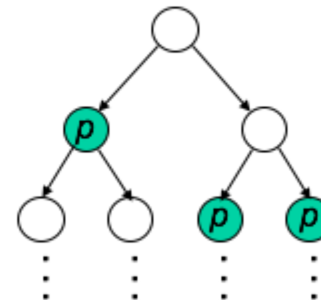
for all paths $\sigma = s_0, s_1, s_2 \ldots$ of $\sigma, s_0 \models f$

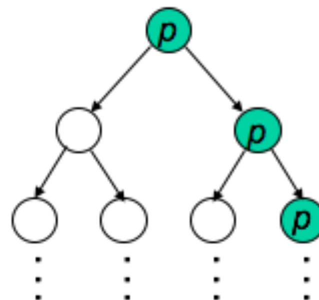


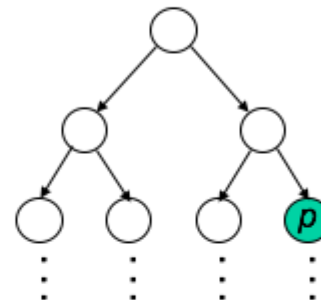
CTL

Every operator F, G, X, U preceded by A or E









LTL

Formulas are of the form ***Af***, where f can include any **nesting** of temporal operators but **no** path quantifiers

Example: LTL formula which is not CTL

$$A\ GF\ p$$

Meaning, along every path, infinitely often *p*

CTL formulas:

- mutual exclusion: **AG** Ø(cs_1 Ù cs_2)
- non starvation: **AG** (request Þ **AF** grant)
- "sanity" check: **EF** request

LTL formulas:

- fairness: **A**(**GF** enabled Þ **GF** executed)
- **A**(x=a Ù y=b Þ **XXXX** z=a+b)

PLTL, CTL, LTL

Contains both CTL and LTL

–*path formulas* *p U q, G p, Fp, Xp, Øp, p Ù q*

–*state formulas* *A p, E p*

p in LTL - *A p* in CTL

Framework for comparing expressiveness

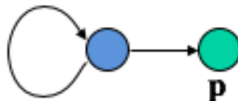
–*Existential properties not expressible in PLTL*

e.g., AG EF p

–*Fairness assumptions not expressible in CTL*

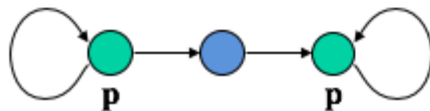
e.g., A (GF p ® GF q)

– ***AG EF p*** is weaker than ***G F p***



Good for finding bugs

– ***AF AG p*** is stronger than ***F G p***

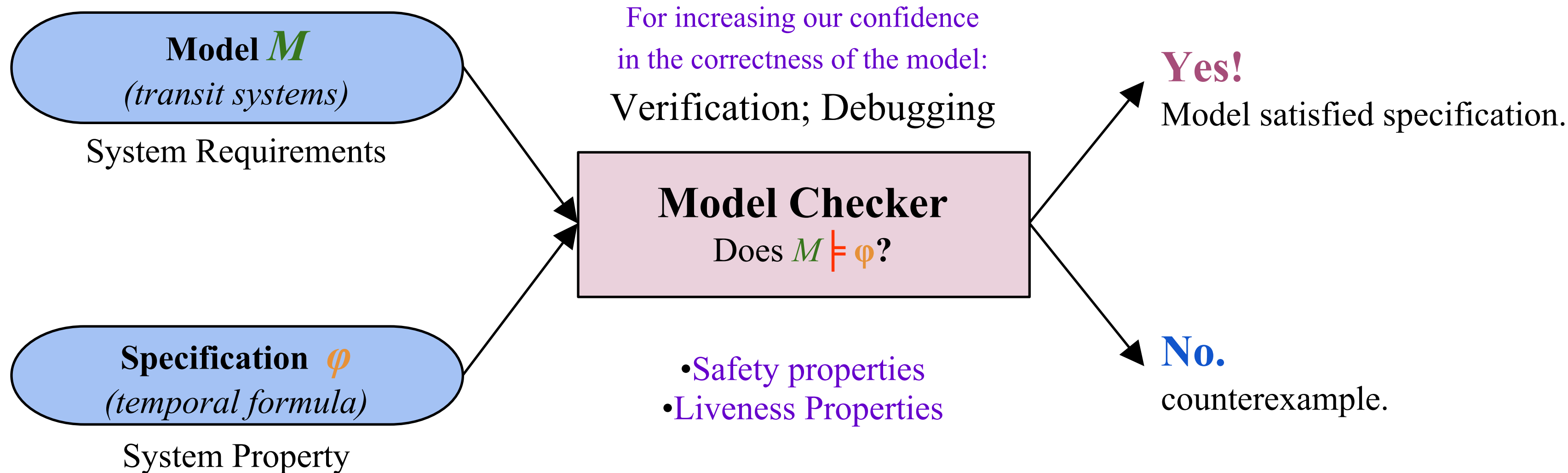


Good for verifying

CTL formulas easier to verify

Model Checking Process

Process satisfy system requirement (model) and property (specification) of final system and generate outputs "Yes" if satisfy or counterexample if not.



ω-Automata

Formally, a Bu ̈chi Automaton is a finite automaton *A = (Q, Σ, δ, s, F),* and the language accepted by such an automaton *is L(A) = {σ | there is a run ρ over σ such that inf(ρ) ∩ F ≠ ∅}*. A language *L ⊆ Σω* is said to be *ω-regular* if it is accepted by some Bu ̈chi automaton.

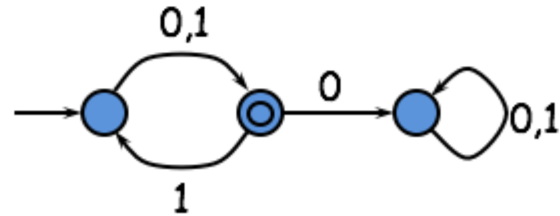
Q-A none-empty states set

Σ-A finite input alphabet

*δ-Q*Σ*

s-The beginning state

F-The set of acceptance states



Symbolic Model Checking

A solution to the state explosion problem

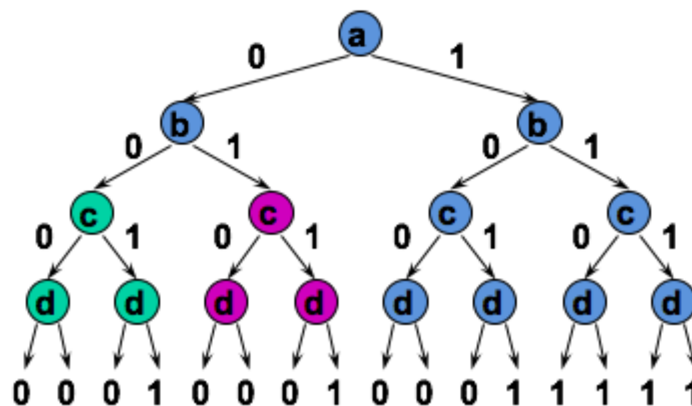
- ***Binary Decision Diagrams (BDDs)*** are used to represent the ***model*** and ***sets of states***.
- It can handle systems with ***hundreds*** of Boolean variables.

BDD-based Symbolic Model Checking

Binary Decision Diagram

Ordered decision tree for $f = ab + cd$

- Data structure for representing
- Boolean functions
- Often ***concise*** in memory
- ***Canonical*** representation
- Most ***Boolean operations*** can be performed on BDDs in ***polynomial time*** i[n] the BDD size



Implementation

Hardware Design

–Encore Gigamax

–Intel instruction decoder

–SGI cache protocol chip

Other areas

–Avionics (TCAS)

–Chemical plant control

–Nuclear storage facilities

Commercial tools

–Cadence, IBM, Synopsys

Software model checking

Model checking by itself cannot deal with the complexity of software

- *Techniques from static analysis are required*

Abstract interpretation, slicing, alias & shape analysis, symbolic execution

- *Even then, we need to borrow some more!*

Heuristic search, constraint solving, etc.

- *Abandon soundness*

Aggressive heuristics

Runtime analysis and runtime monitoring

Software model checking techniques

- *Program Verification*

For example, ESC/Java from Compaq
http://research.compaq.com/SRC/esc/

- *Static analysis for runtime errors*

For example, PolySpace for C, Ada and Java
http://www.polyspace.com/

- *Requirements and Design Analysis*

Analysis for SCR, RSML, Statecharts, etc.

- *Runtime analysis*

See Runtime Verification Workshops
http://ase.arc.nasa.gov/rv2002/

- *Analysis Toolsets*

IF (Verimag), SAL (SRI), etc.

Model Checking Issues

- Temporal logic: (heavy) can be hard to work with
- Translations of requirements models to the input language of model checking engines often times not straightforward.
- If no bugs are detected, does this mean that we have achieved verification, or just got too crude a model or property?
- Number of states typically grows exponentially in the number of processes: cannot be efficiently checked, due to state space explosion
- Counter-examples: do not mean anything to the stakeholders; need to be translated back into the original modeling language.
- Deals only with *state-oriented behavioral requirements models*

(Or, is it more like P, M |= S *with Promela?;*

Or, is it more like state-oriented behavioral S |= descriptive S*?)*

The future software model checking

- *Abstraction based approaches*

Combine object abstractions *(e.g. shape analysis)* with predicate abstraction

Automation is crucial

- *Symbolic Execution*

Solving structural (object) and numerical constraints

Acceleration techniques (e.g. widening)

- *Model checking as a companion to testing*

Test-case generation by model checking

Runtime monitoring and model checking

- *Modular model checking for software*

Exploiting the interface between components

Interface automata (de Alfaro & Henzinger)

- *Environment generation*

How to derive a “test-harness” for a system to be model checked

- *Result representation*

Much overlooked, but without this we are nowhere!

“Analysis is necessary, but not sufficient” – Jon Pincus

Reference

- *Model checking E. Clarke, O. Grumberg, D. Peled, MIT Press, 1999.*
- *The Temporal Logic of Programs A. Pnueli, FOCS 1977 R. E. Bryant*
- *Graph-based Algorithms for Boolean Function Manipulation, IEEE transactions on Computers, 1986*
- *Symbolic Model Checking J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, L.J. Hwang,: 10^20 States and Beyond, LICS'90*