

Modeling & Analysis of Timed Systems

Wang Yi
Uppsala University, Sweden

CUGS May 15-16, 2007

Modified by Insup Lee for CIS 480, Spring 2009

1

OUTLINE

- Model checking
- Timed automata and verification problems
- UPPAAL tutorial: data structures & algorithms

2

Main references (Papers)

- Temporal Logics (CTL,LTL)
 - **Automatic Verification of Finite State Concurrent Systems Using Temporal Logic Specifications: A Practical Approach.** Edmund M. Clarke, E. Allen Emerson, A. Prasad Sistla, POPL 1983: 117-126, also as "Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications. ACM Trans. Program. Lang. Syst. 8(2): 244-263 (1986)"
 - **An Automata-Theoretic Approach to Automatic Program Verification,** Moshe Y. Vardi, Pierre Wolper: LICS 1986: 332-344. Also as "Reasoning About Infinite Computations. Inf. Comput. 115(1): 1-37 (1994)"
- Timed Systems (Timed Automata, TCTL)
 - **A Theory of Timed Automata.** Rajeev Alur, David L. Dill. Theor. Comput. Sci. 126(2): 183-235 (1994)"
 - **Symbolic Model Checking for Real-Time Systems,** Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. *Information and Computation* 111:193-244, 1994.
 - **UPPAAL in a Nutshell.** Kim Guldstrand Larsen, Paul Pettersson, Wang Yi. STTT 1(1-2): 134-152 (1997)
 - **Timed Automata – Semantics, Algorithms and Tools,** a tutorial on timed automata Johan Bengtsson and Wang Yi: (a book chapter in Rozenberg et al, 2004, LNCS).

3

Main references (Books)

- Edmund M. Clarke, Orna Grumberg and Doron A. Peled, **Model Checking**
- G.J. Holzmann, Prentice Hall 1991, Design and Validation of Computer Protocols (new book: **The SPIN MODEL CHECKER Primer and Reference Manual** , 2003)
- Joost-Pieter Katoen, **Concepts, Algorithms, and Tools for Model Checking** (draft book on the web)

4

Lecture 1

Motivation and Sketch of Verification History

5

History: Model-checking invented in 70's/80s

[Pnueli 77, Clarke et al 83, POPL83, Sifakis et al 82]

- **Restrict attention to finite-state programs**
 - Control skeleton + boolean (finite-domain) variables
 - Found in hardware design, communication protocols, process control
- **Temporal logic specification of e.g., synchronization pattern**
 - There are algorithms to check that **MODEL of program** satisfies: **SPEC**
 - e.g. Alternating Bit Protocol skeleton, around 140 states, 1984
- **BDD-based symbolic technique** [Bryant 86]
 - SMV 1990 Clarke, McMillan et al, state-space 10^{20}
 - Now powerful tools used in processor design
- **On-the-fly enumerative technique** [Holzman 89]
 - **SPIN**, **COSPAN**, **CAESAR**, **KRONOS**, **UPPAAL** etc
- **SAT-based techniques** [Clarke et al, McMillan, ...]

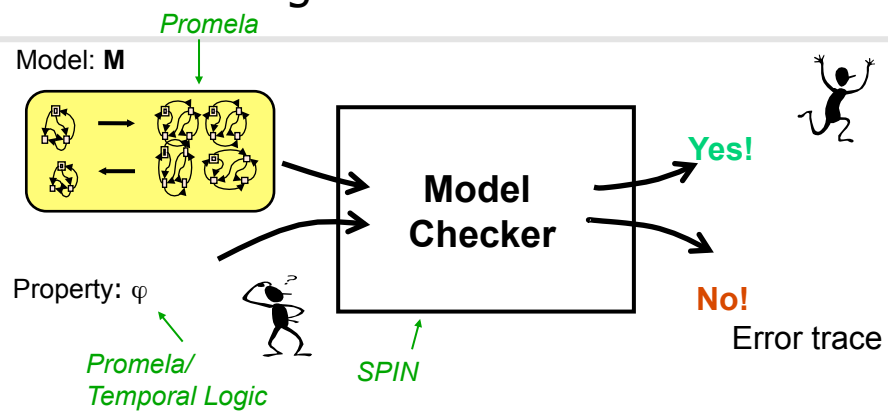
6

History: Model checking for real time systems, started in the 80s/90s

- Extension of model checking to consider time quantities
 - Models, specifications, and algorithms can be extended
- Timed automata, timed process algebras
[Alur&Dill 1990]
- Tools
 - KRONOS, Hytech, 1993-1995, IF 2000's
 - TAB 1993, UPPAAL 1995, TIMES 2002

7

Model Checking



8

Merits of this simpler approach

- Checking **simple properties** (e.g. deadlock freeness) is already extremely useful!
- The goal is **no longer** seen as proving that a system is completely, absolutely and undoubtedly correct (**bug-free**)
- The **objective** is to have tools that can help a developer **find errors** and gain confidence **in her/his design**. That is achievable
- Now widely used in hardware design, protocol design, and hopefully soon, **embedded systems!**

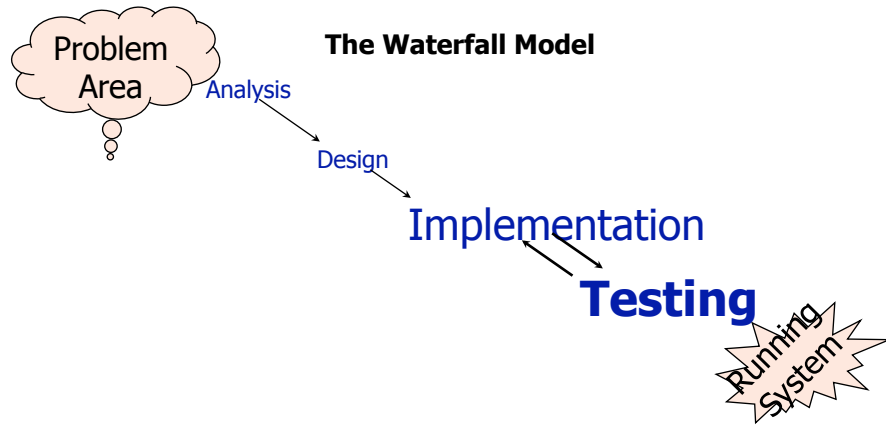
9

Why testing not good enough

- **Testing/simulation**: coverage problems, difficult to deal with non-determinism and concurrent computation
- **Formal verification/Model-Checking** (= **exhaustive testing** of software and hardware **design**) provides 100% coverage

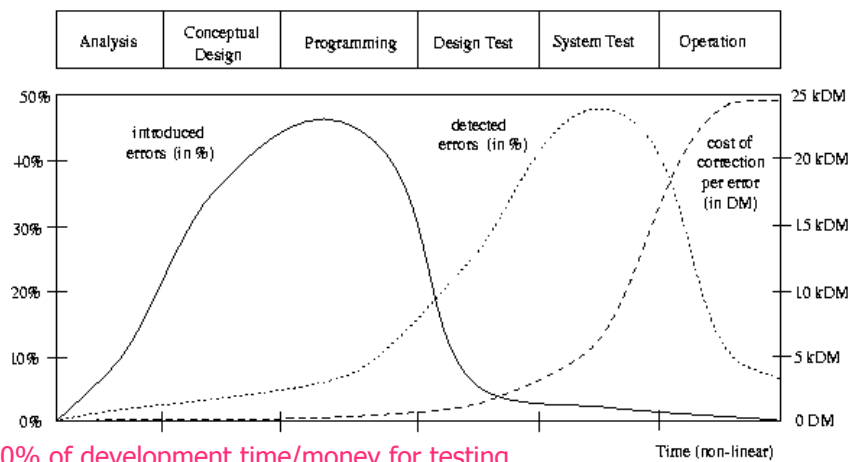
10

Traditional software development



11

Introducing, Detecting and Correcting errors



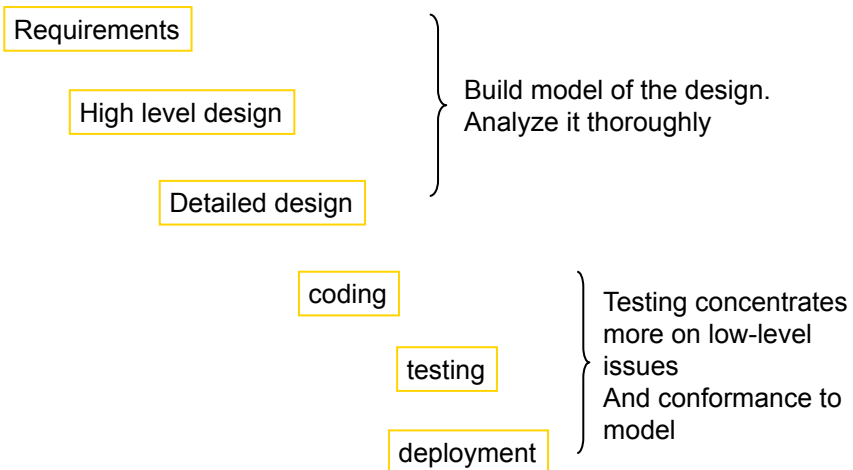
- ◆ 30-50% of development time/money for testing
- ◆ Errors detected: the late the more expensive

12

Model-Checking may complement testing to find (design) Bugs as early as possible

13

Motivation: Model Verification



14

Problems that can be addressed by Model Checking

Checking correctness of

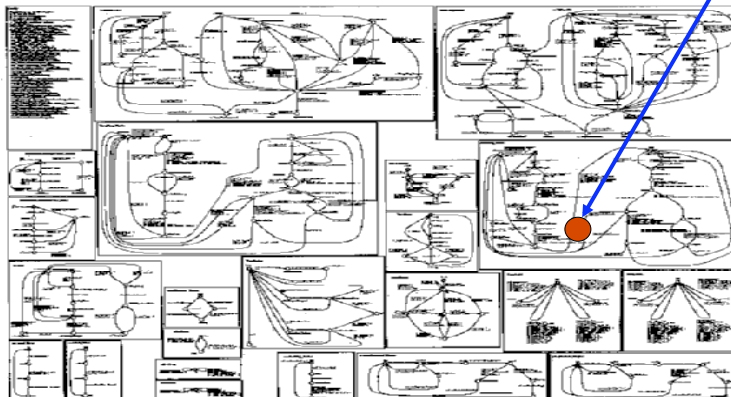
- Communication protocols
- Distributed Algorithms
- Controllers
- Hardware circuits
- Parallel and distributed software
- Embedded and real-time systems and software

e.g., Absence of race conditions, proper synchronization,

Model checking is the appropriate technique when there are many many different scenarios of interaction between components in a system

15

An 'abstract' version of a fielded bus protocol



*Reachable?
(bug?)*

16

Model-Checking

in a Nutshell

17

EXAMPLE: Petersson's algorithm

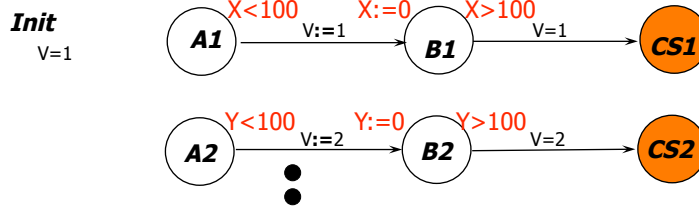
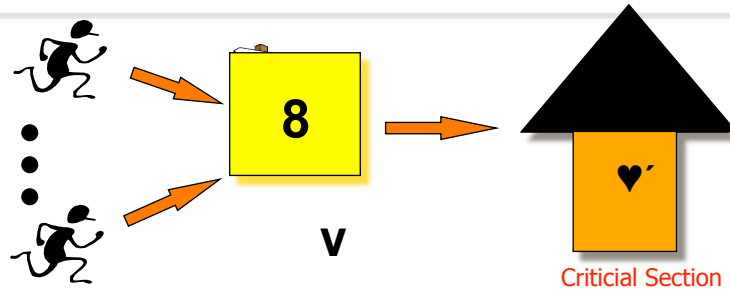
turn, flag1, flag2: shared variable

- Process 1
 - loop
 - flag1:=1; turn:=2
 - while (flag2 & turn=2) wait
 - **CS1**
 - flag1:=0
 - end loop
- Process 2
 - loop
 - flag2:=1; turn:=1
 - while (flag1 & turn=1) wait
 - **CS2**
 - flag2:=0
 - end loop

Question: can both run in CS simultaneously ?

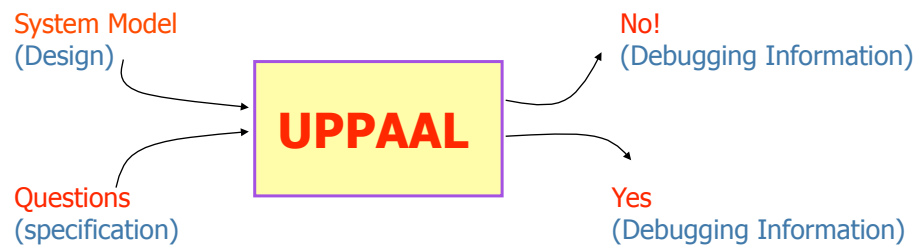
18

Example: Fischer's Protocol



19

UPPAAL *A model checker for real-time systems*



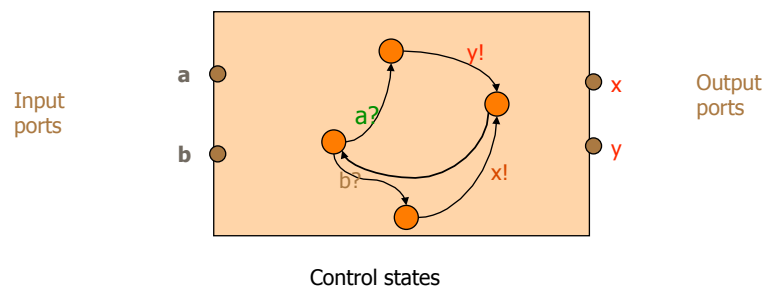
20

MODELING

How to construct Model ?

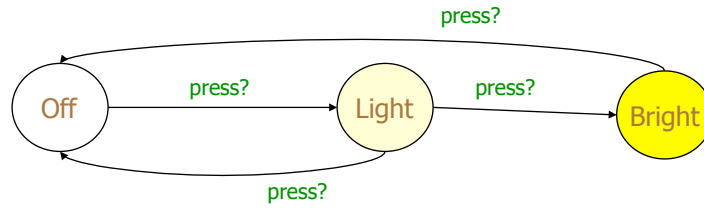
21

Program as State Machine!



22

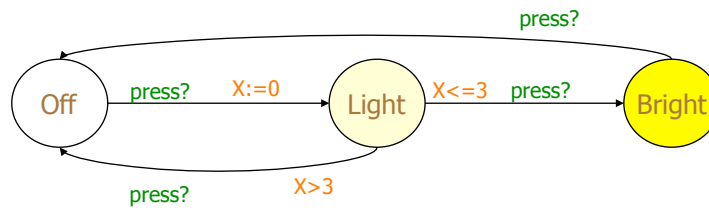
A Light Controller



WANT: if press is issued twice quickly then the light will get brighter; otherwise the light is turned off.

23

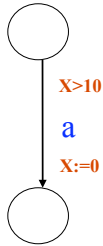
A Light Controller (with timer)



Solution: Add real-valued clock x

24

Modeling Real Time Systems

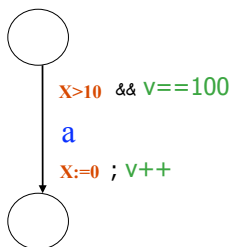


Timed Automaton

- Events
 - synchronization
 - interrupts
- Timing constraints
 - specifying event arrivals
 - e.g. Periodic and sporadic

25

Modeling Real Time Systems

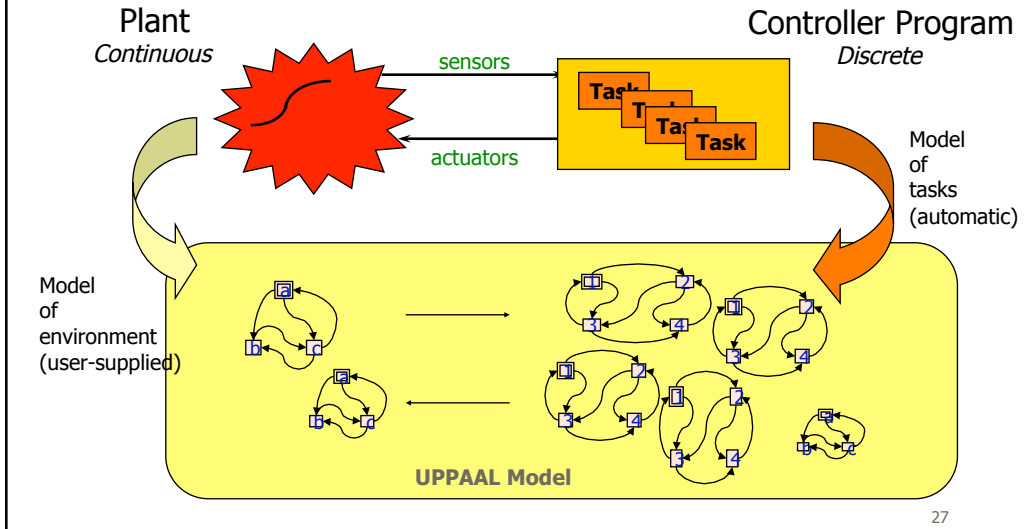


*Timed Automaton
in UPPAAL*

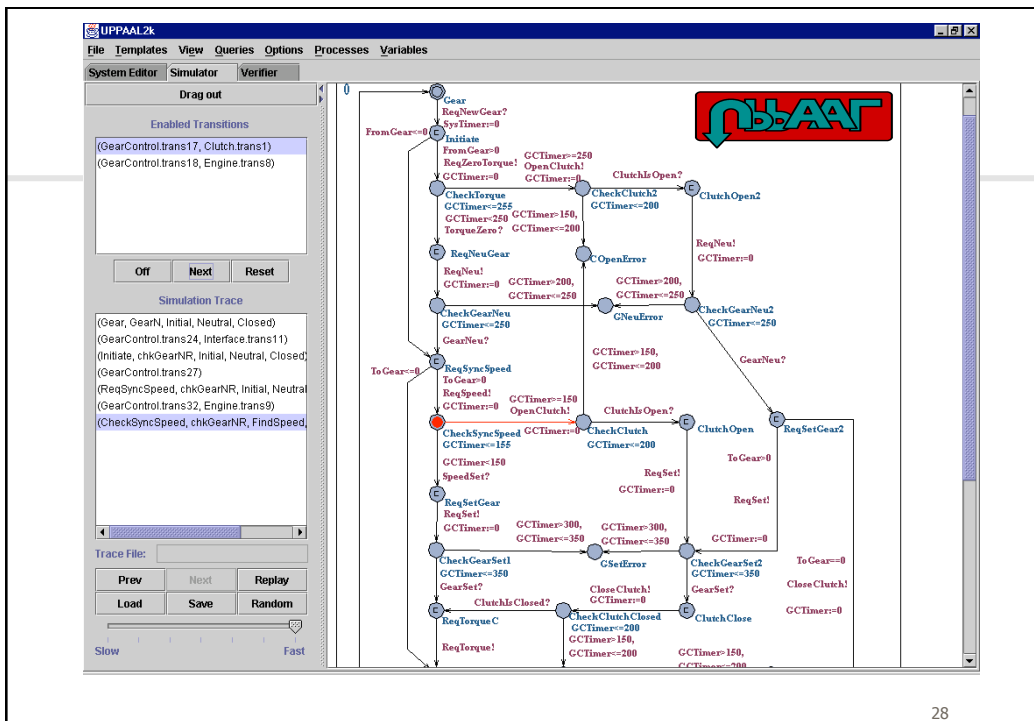
- Events
 - synchronization
 - interrupts
- Timing constraints
 - specifying event arrivals
 - e.g. Periodic and sporadic
- Data variables & C-subset
 - Guards
 - assignments

26

Construction of Models: Concurrency



27



28

SPECIFICATION

How to ask questions: Specs ?

29

Specification=Requirement, *Lamport 1977*

- Safety
 - Something (bad) will not happen
- Liveness
 - Something (good) must happen

30

Specification=Requirement [Lamport 1977]

- **Safety**
 - Something (bad) will not happen
- **Liveness**
 - Something (good) must happen
- **Realizability (for systems with limited resources)**
 - Schedulability, enough resources?

31

Specification: Examples

- **Safety**
 - AG $\neg(P1.CS1 \ \& \ P2.CS2)$ Always **G**lobally
 - AG $(m < 100)$
 - EF $(5 < 6)$ Possibly in **F**uture
 - construct the whole state space
 - Report deadlocks etc.
 - EF $(viking1.safe \ \& \ viking2.safe \ \& \ viking3.safe \ \& \ viking4.safe)$
 - AG $(time > 60 \ \text{imply} \ viking4.safe)$
- **Liveness**
 - AF $(m > 100)$ Eventually
 - AG $(P1.try \ \text{imply} \ AF \ P1.CS1)$ Leads to

32

VERIFICATION

Model meets Specs ?

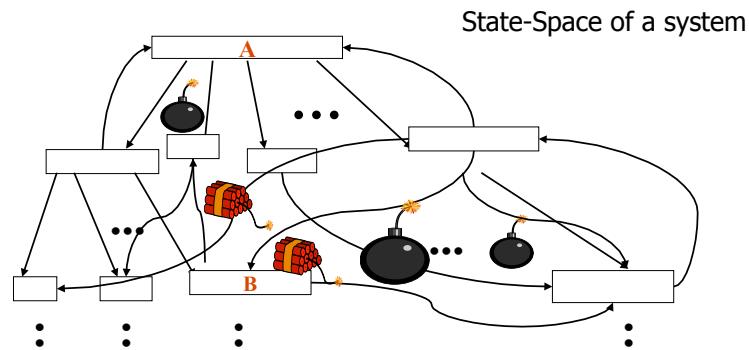
33

(Formal) Verification

- Semantics of a system
 - = all states + state transitions
 - (all possible executions)
- Verification
 - = state space exploration + examination

34

Verification = Searching



(1) SAFETY:

- Is it possible to fire the bombs?
- Is it possible to go from A to B within 10 sec?

(2) LIVENESS:

- Will B be executed eventually (no time bound given)?

35

Approaches to Verification

- **Manual: Proof systems, paper and pen**
 - Find invariants (difficult !)
 - Induction: Assume n th-state OK, check $(n+1)$ th OK
 - Boring ☹ (more fun with programming)
- **Semi-automatic: Theorem proving**
 - Use theorem provers to prove the induction step
 - e.g. PVS, HOL, ALF
 - Require too much expertise ☹
- **Automatic: Model-Checking ☺**
 - State-Space Exploration and Examination
 - e.g. SPIN, SMV, UPPAAL

36

Two basic verification algorithms

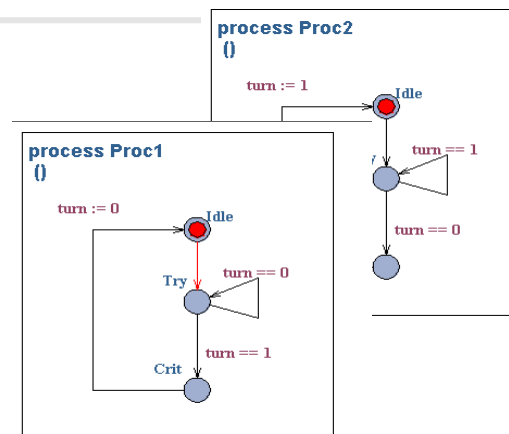
- Reachability analysis
 - Checking safety properties
- Loop detection
 - Checking liveness properties

37

Modelling in UPPAAL: example

```
P1 :: while True do
  T1 : wait(turn=1)
  C1 : CS1; turn:=0
endwhile
||
P2 :: while True do
  T2 : wait(turn=0)
  C2 : CS2; turn:=1
endwhile
```

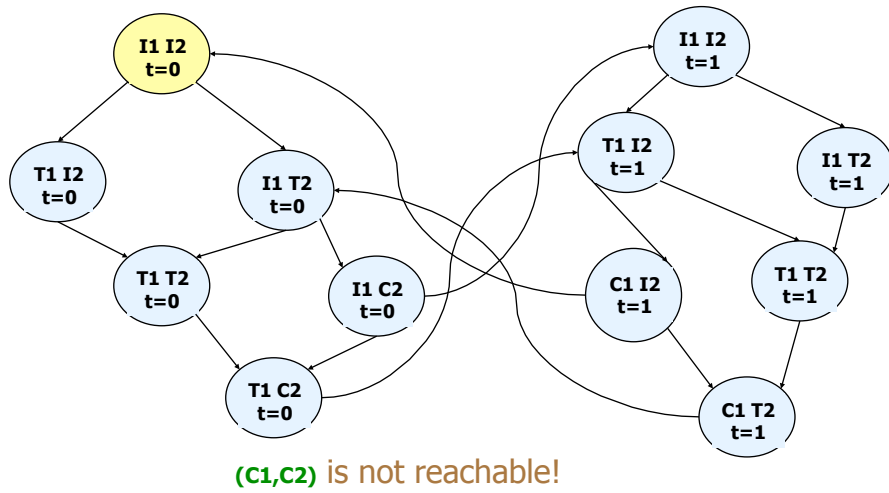
Mutual Exclusion Program



Is it possible that P1 and P2 run C1 and C2 simultaneously?

38

Verification: example



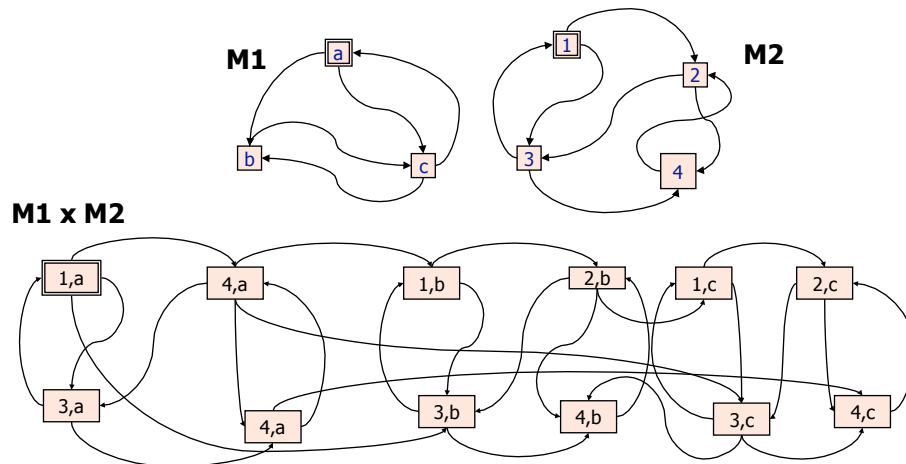
39

UPPAAL Demo

40



Problem with verification: 'State Explosion'



All combinations = exponential in no. of components

41

EXAMPLE

13 components and each with 1 clock & 10 states

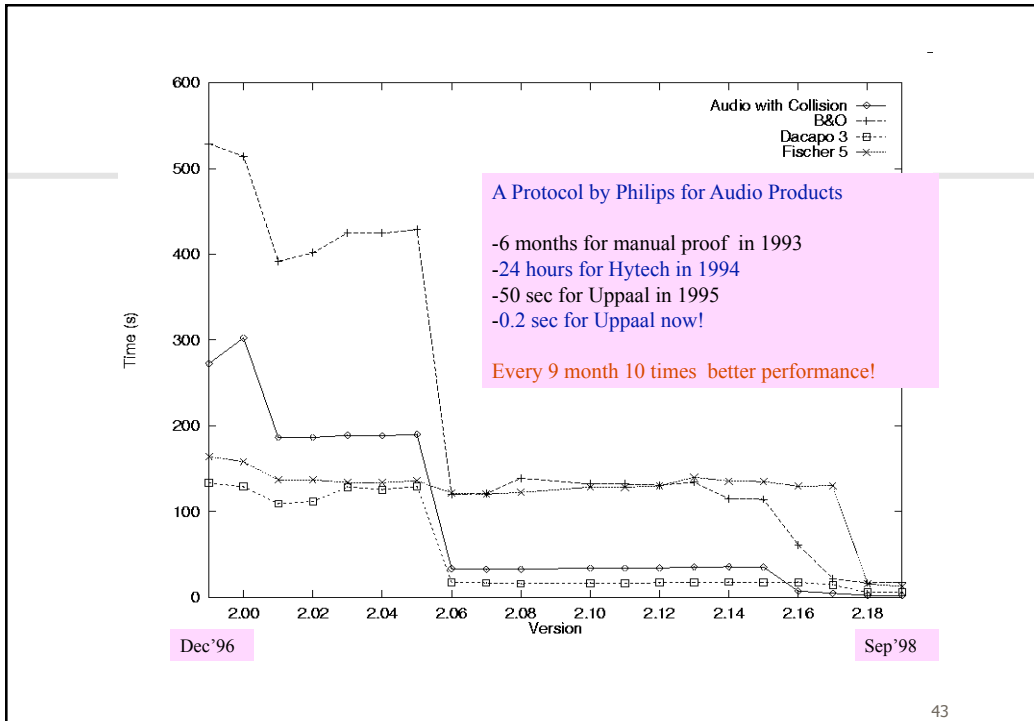
of states = $10,000,000,000,000 = 10,000 \text{ G}$

Each needs $(10 * 10) * 4\text{Bytes} = 400 \text{ Bytes}$

Worst case memory usage $\gg 4,000,000\text{GB}$



42



43

The dream goes on

- *Model Checking, a useful and applicable technique as compiler theory*

End of introduction

44