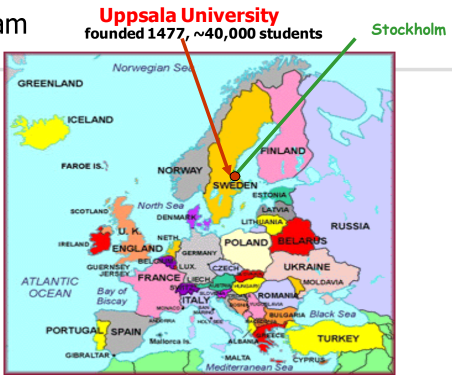
Here I am



Plan for today

- Part 1: Model Checking of Timed Systems
  - A UPPAAL Tutorial
- Part 2: Multicore Real-Time Systems
  - Challenges
  - The Timing Analysis Problems and Solutions

PART 1

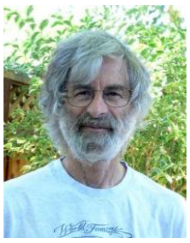
A UPPAAL Tutorial

Model-Checking of Timed Systems

Wang Yi
Uppsala University, Sweden

VTSA Summer School
Luxembourg, Sept 2010

This is simple, simple, simple … …



LESLIE LAMPORT

The main goal of this lecture

What's inside the tool: UPPAAL

UPPAAL: www.uppaal.com

- Developed jointly by
  - Uppsala university, Sweden
  - Aalorg university, Denmark
- UPPsala + AALborg = UPPAAL

UPPAAL: www.uppaal.com

- Developed jointly by
  - Uppsala university, Sweden
  - Aalorg university, Denmark
- UPPsala + AALborg = UPPAAL
  - SWEDEN + DENMARK = SWEDEN
  - SWEDEN + DENMARK = DENMARK

Main Authors/Contributors of UPPAAL

- Gerd Behrman
- Johan Bengtsson
- Alexandre David
- Kim G Larsen
- Fredrik Larsson
- Paul Pettersson
- Wang Yi

THANKS!

OUTLINE

- Model Checking in a Nutshell
- Timed automata and TCTL
- A UPPAAL Tutorial
  - Data stuctures & central algorithms
  - UPPAAL input languages

Main references

- **Temporal Logics (CTL)**
  - 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) "
- **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).
  - **On-line help of UPPAAL**: www.uppaal.com

Model-Checking

in a Nutshell

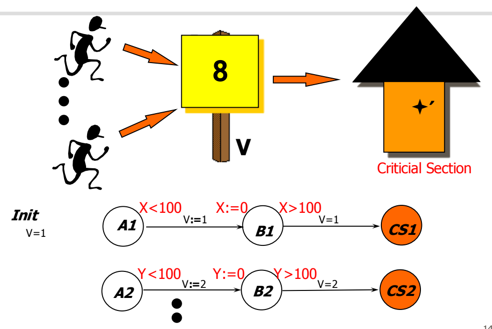
Merits of model checking …

- Checking simple properties (e.g. deadlock-free) is already extremely useful!
  - It is not to prove that a system is completely correct (bug-free)
- The goal is to have tools that can help a developer find errors and improve the quality of her/his design.
  - It is to complement testing
- Now widely used in hardware design, protocol design, and hopefully soon, embedded systems!

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

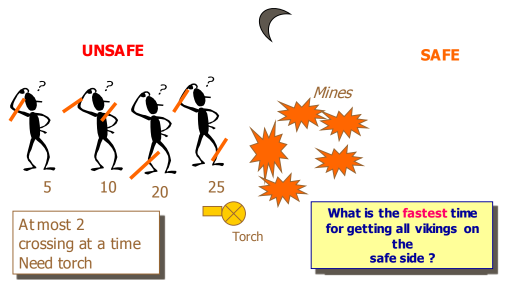
- Models of timed systems
  - Timed automata, [Alur&Dill 1990]
  - Timed process algebras, Timed CSP, Timed CCS
- Extension of model checking to consider time quantities
  - Timed variants of temporal logics e.g TCTL
- Tools
  - KRONOS, Hytech: 1993 --
  - UPPAAL 1995 –
    - TAB 1993/Prototype of UPPAAL [FORTE94, Wang et al]

Example: Fischer's Protocol

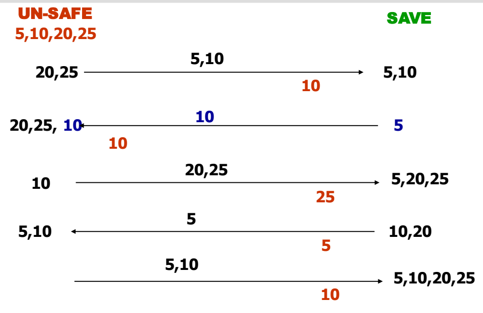


Example: the Vikings Problem

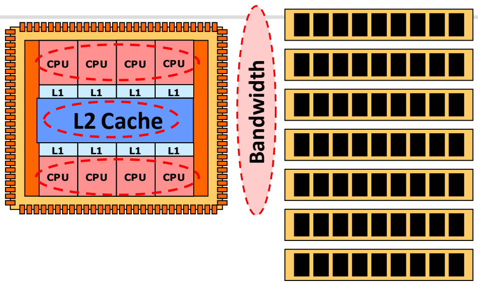
Real time scheduling


UNSAFE
SAFE
Mines
5
10
20
25
Torch
At most 2 crossing at a time
Need torch
What is the fastest time for getting all vikings on the safe side ?

Solution

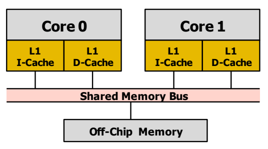


Multicore Challenges



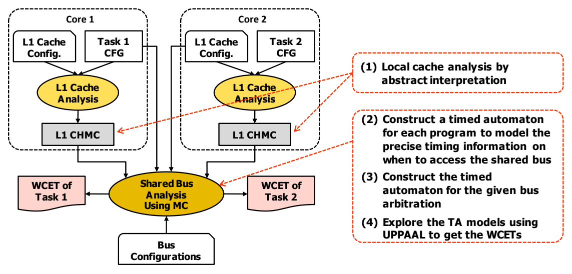
Shared Resources -- cpu's, caches, bandwidth, energy budget etc.

Worst-Case Execution Time Analysis of Concurrent Programs on Multicores



A duo-core processor with private L1 cache and shared memory bus

Combining Static Analysis & Model-Checking [RTSS 2010]



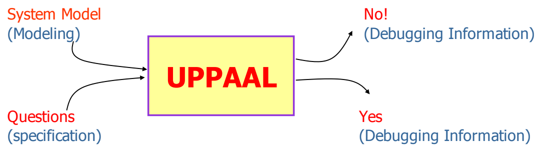
(1) Local cache analysis by abstract interpretation

(2) Construct a timed automaton for each program to model the precise timing information on when to access the shared bus

(3) Construct the timed automaton for the given bus arbitration

(4) Explore the TA models using UPPAAL to get the WCETs

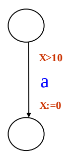
UPPAAL *A model checker for real-time systems*



MODELING

How to construct Model ?

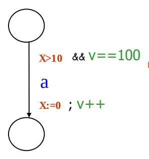
Modeling Real Time Systems



Timed Automaton

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

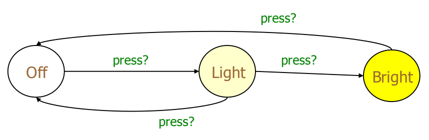
Modeling Real Time Systems



Timed Automaton in UPPAAL

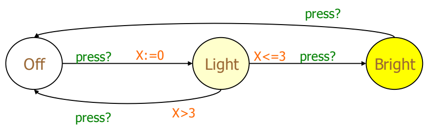
- Events
  - synchronization
  - interrupts
- Timing constraints
  - specifying event arrivals
  - e.g. Periodic and sporadic
- Data variables & C-subset
  - Guards
  - assignments

A Light Controller



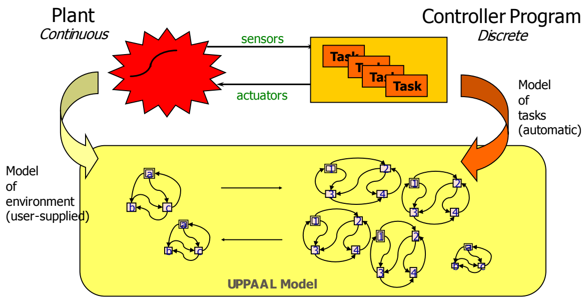
WANT: if press is issued twice quickly then the light will get brighter; otherwise the light is turned off.

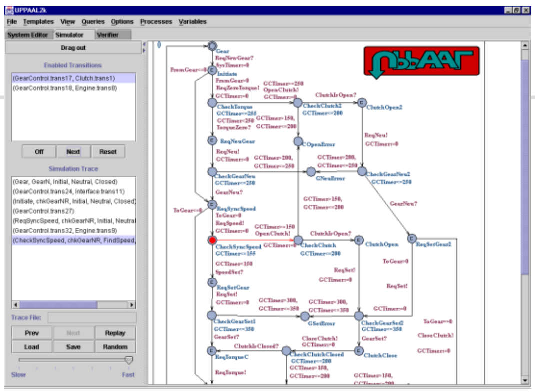
A Light Controller (with timer)



Solution: Add real-valued clock x

Construction of Models: Concurrency





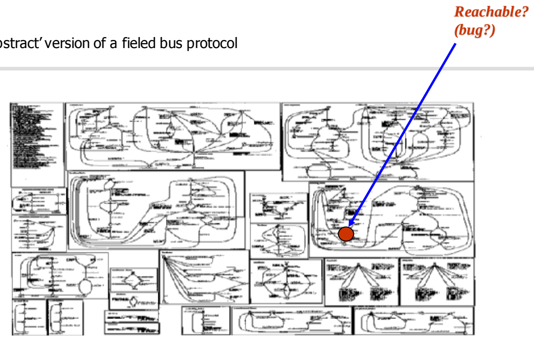
SPECIFICATION

How to ask questions: Specs ?

Specification=Requirement, Lamport 1977

- Safety
  - Something (bad) should not happen
- Liveness
  - Something (good) must happen/should be repeated

An 'abstract' version of a fieled bus protocol



Computation Tree Logic, CTL

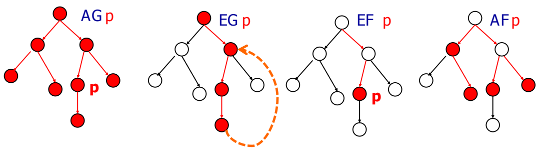
Clarke & Emerson 1980

Syntax

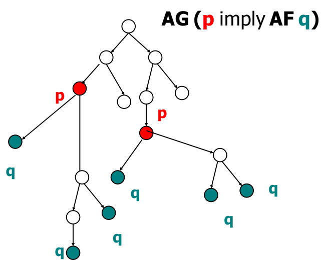
ϕ :: = P | ¬ ϕ | ϕ ∨ ϕ | EX ϕ | E[ϕ U ϕ] | A[ϕ U ϕ]

where **P** ∈ AP (atomic propositions)

Derived Operators



Liveness: p - -> q *"p leads to q"*



Specification: Examples

- Safety
  - AG ¬(P1.CS1 & P2.CS2) **Invariant**
  - AG (temp > 10 & speed < 120)
  - EF (time>60 imply viking4.safe) **Reachability**
  - EF (viking1.safe & viking2.safe & viking3.safe & viking4.safe)
- Liveness
  - AF (speed >100) **Eventually**
  - AG (P1.try imply AF P1.CS1) **Leads to**

VERIFICATION

Model meets Specs ?

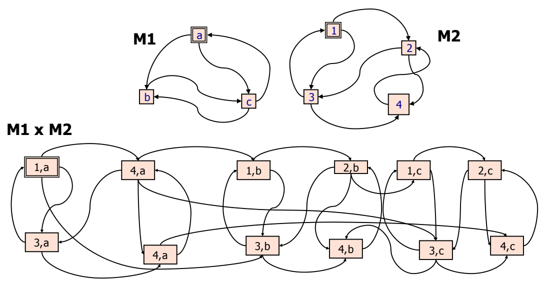
Verification

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

Two basic verification algorithms

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

Problem with verification: 'State Explosion'



All combinations = exponential in no. of components

EXAMPLE

13 components and each with 1 clock & 10 states

# of states = 10,000,000,000,000 =10,000 G

Each needs (10 * 10)* 4Bytes = 400 Bytes

Worst case memory usage >> 4,000,000GB

UPPAAL DEMO