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
VTTA Summer School
Luxembourg, Sept 2010

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
- UP Psala + AALborg = UPPAAL
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 [FORTES, Wang et al]

Example: Fischer’s Protocol

Example: the Vikings Problem
Real time scheduling

Solution

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

MODELING

How to construct Model?

Modeling Real Time Systems

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

UPPAAL - A model checker for real-time systems

System Model
(Modeling)

Questions
(specification)

No!
(Debugging Information)

Yes
(Debugging Information)

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

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 field bus protocol
Computation Tree Logic, CTL
Clarks & Emerson 1980

Syntax
\[ \phi ::= P \mid \neg \phi \mid \phi \vee \phi \mid \text{EX} \phi \mid E[\phi U \psi] \mid A[\phi U \psi] \]
where \( P \) \( \in \) AP (atomic propositions)

Derived Operators

Liveness: \( p \rightarrow q \) 
"p leads to q"

Specification: Examples

- Safety
  - AG \((P1.CS1 \& P2.CS2)\)
  - AG \((\text{temp} > 10 \& \text{speed} < 120)\)
  - EF (time>60 imply viking1.safe) 
- Invariant
  - AG \((\text{temp} > 10 \& \text{speed} < 120)\)
- Reachability
  - EF (time>60 imply viking1.safe)
- Liveness
  - AF (speed > 100)
  - AG (P1.try imply AF P1.CS1)
- Eventually
  - AG (P1.try imply AF P1.CS1)
- Leads to

Verification

- Semantics of a system
  - all states + state transitions
  - all possible executions

- Verification
  - state space exploration + examination

Verification

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

Two basic verification algorithms
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