Modeling & Analysis of Timed Systems

Wang Yi
Uppsala University, Sweden

CUGS May 15-16, 2007

Modified by Insup Lee for CIS 480, Spring 2009

OUTLINE

- Model checking
- Timed automata and verification problems
- UPPAAL tutorial: data structures & algorithms
Main references (Papers)

- **Temporal Logics (CTL,LTL)**
- **Timed Systems (Timed Automata, TCTL)**

Main references (Books)

- Edmund M. Clarke, Orna Grumberg and Doron A. Peled, *Model Checking*
Lecture 1
Motivation and Sketch of Verification History

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 [Holzmann 89]
  - SPIN, COSPAN, CAESAR, KRONOS, UPPAAL etc
- SAT-based techniques [Clarke et al, McMillan, ...]
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

Model Checking

Model: \( M \)

Property: \( \varphi \)

Promela/Temporal Logic

Model Checker

Yes!

No!

Error trace

Promela

SPIN
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 undoutedly 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!

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
Traditional software development

The Waterfall Model

- Analysis
- Design
- Implementation
- Testing

Problem Area

Introducing, Detecting and Correcting errors

30-50% of development time/money for testing

Errors detected: the late the more expensive

<table>
<thead>
<tr>
<th>Analysis</th>
<th>Conceptual Design</th>
<th>Programming</th>
<th>Design Test</th>
<th>System Test</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

30% introduced errors in %
30% detected errors in %
25 kDM cost of correction percent in DM
Model-Checking may complement testing to find (design) Bugs as early as possible

Motivation: Model Verification

- Requirements
  - High level design
  - Detailed design
- Build model of the design.
  - Analyze it thoroughly
- Coding
- Testing
- Deployment

- Testing concentrates more on low-level issues
  - And conformance to model
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.

An ‘abstract’ version of a fielded bus protocol

Reachable? (bug?)
Model-Checking in a Nutshell

EXAMPLE: Petersson’s algorithm

- 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?
Example: Fischer’s Protocol

Init

\[ V = 1 \]

\[ A_1 \quad X < 100 \quad V = 1 \quad B_1 \quad X > 100 \quad V = 1 \]

Critical Section

\[ B_2 \quad Y > 100 \quad V = 2 \quad C_{S2} \]

\[ C_{S1} \]

UPPAAL  
A model checker for real-time systems

System Model (Design)

Questions (specification)

No! (Debugging Information)

Yes (Debugging Information)
MODELING

How to construct Model?

Program as State Machine!
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 $x$
Modeling Real Time Systems

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

Data variables & C-subset

Guards

Assignments

---

\[
X > 10 \\
X := 0
\]

---

\[
X > 10 \land v == 100 \\
a \\
X := 0 \land v++
\]
Construction of Models: Concurrency

Plant
Continuous

Controller Program
Discrete

Model of tasks (automatic)

Model of environment (user-supplied)

UPPAAL Model

Model of environment (user-supplied)

UPPAAL Model
SPECIFICATION

How to ask questions: Specs?

Specification=Requirement, Lamport 1977

- Safety
  - Something (bad) will not happen
- Liveness
  - Something (good) must happen
Specification=Requirement  [Lamport 1977]

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

Specification: Examples

- **Safety**
  - AG  
    (P1.CS1 & P2.CS2)  
    **Always Globally**
  - AG  
    (m< 100)
  - EF  (5<6)  
    Possibly in **Future**
    - construct the whole state space
    - Report deadlocks etc.
  - EF  (viking1.safe & viking2.safe & viking3.safe & viking4.safe)
  - AG  
    (time>60 imply viking4.safe)

- **Liveness**
  - AF  
    (m>100)
  - AG  
    (P1.try imply AF P1.CS1)  
    Leads to
VERIFICATION

Model meets Specs?

(Formal) Verification

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

- Verification
  = state space exploration + examination
Verificatioin = Searching

State-Space of a system

(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)?

Approaches to Verification

- Manual: Proof systems, paper and pen
  - Find invariants (difficult !)
  - Induction: Assume nth-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
Two basic verification algorithms

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

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?
Verification: example

\[(C_1, C_2)\] is not reachable!

UPPAAL Demo
Problem with verification: ‘State Explosion’

All combinations = exponential in no. of components

EXAMPLE

13 components and each with 1 clock & 10 states

# of states = $10,000,000,000,000 = 10,000 \text{ G}$
Each needs $(10 \times 10) \times 4 \text{Bytes} = 400 \text{ Bytes}$

Worst case memory usage $>> 4,000,000 \text{GB}$
A Protocol by Philips for Audio Products

- 6 months for manual proof in 1993
- 24 hours for Hytech in 1994
- 50 sec for Uppaal in 1995
- 0.2 sec for Uppaal now!

Every 9 month 10 times better performance!

---

The dream goes on ... ...

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

End of introduction