Verification of Timed Systems

Wang Yi
Uppsala University

ARTIST/China spring school
 Xi’an, China, April 03-15, 2006

OUTLINE

• A Brief Introduction
  • Motivation ... what are the problems to solve
• Temped Systems
  • Timed automata and verification problems
• UPPAAL tutorial (1): data structures & algorithms
• UPPAAL tutorial (2): input languages
• TIMES: From models to code “guaranteeing” timing constraints
• Further topics/Recent Work
  • Systems with buffers/queues [CAV 2006]

Main references (Books)

• Edmund M. Clarke, Orna Grumberg and Doron A. Peled, Model Checking

Main references (Papers)

• Temporal Logics (CTL/LTL)

Contributors

@UPPsala
  • John Håkansson
  • Pavel Kiral
  • Leonid Mokrushin
  • Paul Pettersson
  • ...

@AALborg
  • Kim G Larsen
  • Gerd Behrman
  • Alexandre David
  • ...

@Elsewhere
  • Johan Bangaåsen, Fredrik Larsson, Kåre J Kristoffersen, Thomas Hune, Oliver Nilsson, Evelin Atanasova, David Griftioen, Ansgar Fehnker, Frits Vaandrager, J-P Katoen, Marijn Heuleman, Marga Ulisse, John Hendriks...
Lecture 1
Introduction

The dream started 40 years ago in 1960’s aiming at “bug-free software” [Floyd 1967, Hoare 1969]

What does this program do?

start
y1:=x1,x2
print(y1)
y1:=y1-y2
y2:=y2-y1
y1>y2
y1==y2
Y
N
N
Y

y1=y2
y2=y2-y1
p(p(R(y1))
stop
It computes the Greatest Common Divisor (gcd) of \( x_1 \) and \( x_2 \) [Floyd 67]

**Specification (partial correctness)**

Hoare logic: \( \{ P \} \text{ program } \{ Q \} \) [Floyd 1967, Hoare 1969]

- Assume, initially (pre-condition)
  - \( x_1 > 0, x_2 > 0 \)
- After each iteration of the loop (invariant)
  - \( y_1 > 0, y_2 > 0, \gcd(y_1, y_2) = \gcd(x_1, x_2) \)
- When done (post-condition)
  - \( y_1 = \gcd(x_1, x_2) \)

---

**What does this program do?**

```
start
\( y_1, y_2 := x_1, x_2 \)
print(\( y_1 \))
stop
\( y_2 := y_2 - y_1 \)
\( y_1 := y_1 - y_2 \)
```

Can you check this?

---

Yes, you may prove it manually by induction on the number of iterations. Question: can you automate the proof?

---

**Software verification (now, a hot topic)**

---

**One more example (Total correctness)**

```
Function foo(n)
begin
  if \( n = 1 \) then 1
  else if even(n) then foo(n/2)
  else foo(3*n+1)
end
```

Does this program terminate for any \( n \)? (WCET?)

---

**Reality: 10 years later (1980’s)**

- The majority of programs are never proven correct! what went wrong?
  - Difficult to find and prove invariants: partial correctness
  - Difficult/impossible to prove termination: total correctness
  - Difficult to write complete specifications: what I really want?
- What to do?
  - Start another research program! In 20 years, the problems will be solved, hopefully
History: Model-checking invented in 70’s/80s

- Temporal logics/verification
  - Check the design/model: MODEL - TVEC (not the code)
  - Finite-state, non-terminating, control-intensive, less data
  - e.g. ABP ca 140 states, 1984
- BDD-based symbolic technique [Bryant 86]
- SMV 1989 Clarke, McMillan et al, state-space 10^12
- On-the-fly technique [Pnueli 89]
  - SPIN, COSPAN, CESAR, KRONOS, UPPAAL etc

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

- Timed automata, timed process algebras [Alur&Dill 1990]
- KRONOS, Hytech, 1993-1995, IF 2000’s

Reality: 40 years later, now

- Many extensions and improvements have been proposed, various tools exist: (non-)commercial
- Good complete specifications are still hard to obtain
- However this is not a real problem!

Reality: 40 years later, now

- 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 his/her 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

Introducing, Detecting and Correcting errors

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

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

Model-Checking
in a Nutshell

Example: Peterson'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

turn, flag1, flag2: shared variable

Question: can both run in CS simultaneously?

Example: Fischer's Protocol

Init

- X<100
- Y<100
- X=0
- Y=0

Critical Section

- X>100
- Y>100

- A1
- B1
- C1
- A2
- B2
- C2

- X<100
- Y<100
- X=0
- Y=0
Example: the Vikings Problem
Real time scheduling

UNSAFE SAFE

At most 2 crossing at a time need torch

Torch

What is the fastest time for getting all vikings on the safe side?

UPPAAL A model checker for real-time systems

System Model (Design) Questions (specification)

Yes (Debugging Information) No! (Debugging Information)

MODELING
How to construct Model?

Program as State Machine!

Input ports

Output ports

Control states

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

Construction of Models: Concurrency

Plant
- Continuous

Controller Program
- Discrete

Model of tasks (automatic)

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
- **Realizability (for systems with limited resources)**
  - Schedulability, enough resources?

Specification: Examples

- **Safety**
  - $\text{AG} \neg (P_1.CS_1 \& P_2.CS_2)$  
  - Always Globally
  - $\text{AG} \ (m< 100)$
  - $\text{EF} \ (5<6)$
  - Possibly in Future
  - construct the whole state space
  - Report deadlocks etc.
  - $\text{EF} \ (\text{viking}_1.\text{safe} \& \text{viking}_2.\text{safe} \& \text{viking}_3.\text{safe} \& \text{viking}_4.\text{safe})$
  - $\text{AG} \ (\text{time}>60 \implies \text{viking}_4.\text{safe})$
- **Liveness**
  - $\text{AF} \ (m>100)$  
  - Eventually
  - $\text{AG} \ (P_1.\text{try} \implies \text{AF} \ P_1.\text{CS}_1)$  
  - Leads to

(Proper) Verification

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

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

Verification = 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)?
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

(C1,C2) is not reachable!

Example: the Vikings Problem
Real time scheduling

UNSAFE

At most 2 crossing at a time
Need torch

SAFE

This sounds too good!
What’s the problem?
Problem with verification: ‘State Explosion’

13 components and each with 1 clock & 10 states

\[ \text{\# of states} = 10,000,000,000,000 = 10,000 \ G \]

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

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

The dream goes on ... ...

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

OUTLINE

- A Brief Introduction
  - Motivation ... what are the problems to solve
  - CTL, LTL and basic model-checking algorithms
- Timed Systems
  - Timed automata and verification problems
  - UPPAAL tutorial (1): data structures & algorithms
  - UPPAAL tutorial (2): input languages
  - TIMES: From models to code “guaranteeing” timing constraints
- Further topics/Recent Work
  - Systems with buffers/queues (CAV 2006)

Lecture 2

Model-Checking Untimed Systems
Transition Systems, Temporal Logics and Basic Verification Algorithms
Transition Systems

A transition system is a graph with
- a set of nodes (states)
- a set of edges (transitions)
where
- nodes may be labeled with propositions (state properties)
- edges may be labeled with action names (synchronization)

Example (with labeled nodes)

Example (with labeled nodes, and labeled edges)

EXAMPLE: a BUGGY machine

The (branching-time) semantic of BUGGY

“Properties” of BUGGY
"Properties" of BUGGY

Properties of Computation Trees

Properties of Computation Trees

CTL: Computation Tree Logics
defined on Computation Trees of Kripke structures

Computation Tree Logic, CTL
Clarke & Emerson 1980

Syntax

A CTL-model is a Kripke Structure
(=transition systems with labeled nodes)

Example
Computation Trees vs. STATES

The computation tree of state $s$

Computation Trees vs. STATES

The computation tree of state $s_2$

The computation tree of state $s_2$

Computation trees of STATES

Path (of computation tree)

A path is an infinite sequence of states

e.g. $\sigma = s \ s_1 \ s_2 \ s_3 \ldots$

Formal Semantics of CTL

$\models p \iff p \in \text{label}(s)$

$\models \neg \phi \iff \neg (s \models \phi)$

$\models \phi \lor \psi \iff (s \models \phi) \lor (s \models \psi)$

$\models \exists X \phi \iff \exists \sigma \in P_{\text{set}}(s) \exists j \geq 0 \sigma[j] \models \psi \land (\forall 0 \leq k < j \sigma[k] \models \phi)$

$\models \exists a \sigma \psi \iff \exists \sigma \in P_{\text{set}}(s) \exists j \geq 0 \sigma[j] \models \psi \land (\forall 0 \leq k < j \sigma[k] \models \phi)$

Where $P_{\text{set}}(s)$ denotes the set of paths starting from $s$ and $a[i]$ denotes $i$th element of $a$

$E[\phi \lor \psi]$ is valid in $s$ if some path from $s$ satisfies the above
$A[\phi \ U \ \psi]$ is valid in $s$ if all paths from $s$ satisfy the above.

**CTL, Derived Operators**

- $EF \phi \equiv E[true \ U \ \phi]$ (possible)
- $AF \phi \equiv A[true \ U \ \phi]$ (inevitable)

- $\text{EF} \ p$ in UPPAAL
- $\text{AF} \ p$ in UPPAAL

**CTL, Derived Operators (cont.)**

- $EG \phi \equiv \neg AF \neg \phi$ (potentially always)
- $AG \phi \equiv \neg EF \neg \phi$ (always)

- $\text{EG} \ p$ in UPPAAL
- $\text{AG} \ p$ in UPPAAL

**Theorem**

All operators are derivable from:

- $EX \ f$
- $EG \ f$
- $E[ f \ U \ g ]$

and boolean connectives:

$A[ f \ U \ g ] = \neg E[ \neg g \ U (\neg f \land \neg g)] \land \neg EG \neg g$

**Example**
Example

\[ \text{EX } p \]

\[ \begin{array}{c}
1 & 2 & 3 & 4 \\
p & q & p & q
\end{array} \]

Example

\[ \text{EX } p \]

\[ \begin{array}{c}
1 & 2 & 3 & 4 \\
p & q & p & q
\end{array} \]

Note: state 1 doesn't satisfy \( \text{AX } p \)

Example

\[ \text{AX } p \]

\[ \begin{array}{c}
1 & 2 & 3 \\
p & q & q
\end{array} \]

Example

\[ \text{AX } p \]

\[ \begin{array}{c}
1 & 2 & 3 \\
p & q & q
\end{array} \]

Example

\[ \text{EG } p \]

\[ \begin{array}{c}
1 & 2 & 3 & 4 \\
p & q & p & q
\end{array} \]

Example

\[ \text{EG } p \]

\[ \begin{array}{c}
1 & 2 & 3 & 4 \\
p & q & p & q
\end{array} \]
Properties of MUTEX example?

\[ \text{AG} \neg (C_1 \land C_2) \]
\[ \text{AG} (T_1 \Rightarrow AF(C_1)) \]
\[ \text{EG} [\neg C_2] \]
\[ \text{AG}[C_1 \Rightarrow A[C_1 U (\neg C_1 \land \neg C_2)]]] \]

\text{HOW TO DECIDE IN GENERAL}

CTL Model-Checking Algorithms
Labeling Methods [Clarke et al 81]

- $\text{Sat}(\phi) = \text{all states where } \phi \text{ is true}$
- Compute $\text{Sat}(\phi)$ recursively as follows:
  - For each sub-formula $\phi_i$ of $\phi$, compute $\text{Sat}(\phi_i)$
  - This is easier: e.g. $\text{Sat}(P) = \{s \mid P \in \text{Label}(s)\}$
  - Compose $\text{Sat}(\phi_i)$ to get $\text{Sat}(\phi)$
Properties of MUTEX example?

**Sat(\(T_1 \Rightarrow AF \, C_1\)) = all states!!**

How to Compute Sat(\(\phi U \psi\))

```
function Sat(\(\phi\) : Formula) : set of State;
(* precondition: true *)
begin
  if \(\phi\) = true then
    return \(\emptyset\);
  else
    if \(\phi\) = \(\phi_1 \lor \phi_2\) then
      return \(Sat(\phi_1) \cup Sat(\phi_2)\);
    else
      if \(\phi\) = \(\phi_1 \land \phi_2\) then
        return \(\{ s \in S | (s, s') \in R \land s' \in Sat(\phi_1) \cup Sat(\phi_2) \}\);
      else
        if \(\phi\) = \(\text{EX}\, \phi_1\) then
          return \(\{ s \in S | (s, s') \in R \land s' \in Sat(\phi_1) \}\);
        else
          return \(\{ s \in S | (s, s') \in R \land s' \in Sat(\phi_1) \}\);
      end
    end
  end
end
```

(* postcondition: Sat(\(\phi\)) = \{ s | M, s \models \phi \} *)
How to Compute Sat($E[\phi \cup \psi]$)

1. **New (not in Passed?)**
   - Run time: $Sat(E[\phi \cup \psi])$
   - Definition of $E[\phi \cup \psi]$
   - New state found
   - Passed

2. **Passed**
   - $\phi \in E[\phi \cup \psi]$
   - Passed

3. **No new state, Done!**
   - $\psi \in E[\phi \cup \psi]$
   - Passed
How to Compute $\text{Sat}(A[\phi U \psi])$?
How to Compute $\text{Sat}(A[\phi \cup \psi])$

$\phi \ A[\phi \cup \psi] \ \psi$

Passed

No new more state, Done!

$\phi \ E[\phi \cup \psi] \ \psi$

Passed

Fixpoint Characterizations (SMV)

$$EF \ p \equiv p \lor \text{EX} \ EF \ p$$

Let $A$ be the set of states satisfying $EF \ p$ then

$$A \equiv p \lor \text{EX} \ A$$

in fact $A$ is the smallest one of sets satisfying the equations (the least fixpoint)

Fixed points of monotonic functions

- Let $\tau$ be a function $S \rightarrow S$
- Say $\tau$ is monotonic when
  $$x \leq y \ implies \ \tau(x) \leq \tau(y)$$
- Fixed point of $\tau$ is $y$ such that
  $$\tau(y) \equiv y$$
- If $\tau$ monotonic, then it has
  - least fixed point $\mu: \tau(y)$
  - greatest fixed point $\nu: \tau(y)$

Iteratively computing fixed points

- Suppose $S$ is finite
  - The least fixed point $\mu: \tau(y)$ is the limit of
    $$\text{false} \subseteq \tau(\text{false}) \subseteq \tau(\tau(\text{false})) \subseteq A$$
  - The greatest fixed point $\nu: \tau(y)$ is the limit of
    $$\text{true} \supseteq \tau(\text{true}) \supseteq \tau(\tau(\text{true})) \supseteq A$$

Note, since $S$ is finite, convergence is finite
Example: \( \textit{EF} \ p \)

- \( \textit{EF} \ p \) is characterized by
  \[
  \textit{EF} \ p = \mu y. (p \lor \textit{EX} \ y)
  \]
- Thus, it is the limit of the increasing series...

\[\text{\ldots} p \lor \textit{EX} \ p \lor \textit{EX} \ p \lor \ldots\]

Example: \( \textit{EG} \ p \)

- \( \textit{EG} \ p \) is characterized by
  \[
  \textit{EG} \ p = \nu y. (p \land \textit{EX} \ y)
  \]
- Thus, it is the limit of the decreasing series...

\[\text{\ldots} p \land \textit{EX} \ p \land \textit{EX} \ p \land \ldots\]

Example, continued

\( \textit{EF} \ q \)

\[
\textit{EF} \ q = \mu y. (q \lor \textit{EX} \ y)
\]

\[
\begin{array}{c}
 p \\
 1 \\
 2 \\
 \text{q} \\
 3 \\
 4 \\
 A_0 = \emptyset \\
 A_1 = \{2,3\} \\
 A_2 = \{1,2,3\} \\
 A_3 = \{1,2,3\}
\end{array}
\]

Remaining operators

\[
\begin{align*}
\textit{AF} \ p &= \nu y. (p \lor \textit{AX} \ y) \\
\textit{AG} \ p &= \mu y. (p \land \textit{AX} \ y) \\
E(p \textit{U} q) &= \mu y. (q \lor (p \land \textit{EX} \ y)) \\
A(p \textit{U} q) &= \mu y. (q \lor (p \land \textit{AX} \ y))
\end{align*}
\]

Complexity

The worst-case time complexity of checking whether system-model sys satisfies the CTL formula \( \phi \) is \( O(|\text{sys}| \times |\phi|) \).

However, \( |\text{sys}| \) may be EXPONENTIAL in number of parallel components!

- FIXPOINT COMPUTATIONS may be carried out using ROBDD's (Reduced Ordered Binary Decision Diagrams)

LTL: Linear Time Logics

Defined on infinite traces of transition systems with (Buchi) accepting conditions
LTL, Linear-Time Logic

Syntax

\[ \phi ::= P | \neg \phi | \phi \lor \psi | \text{EX} \phi | \text{U} \phi \]

where \( P \in \text{AP} \) (atomic propositions)

EX pronounced “nEXT state”

U pronounced “Until”

EXAMPLE: a BUGGY machine

The linear-time behaviour of BUGGY

\( \phi \text{ U } \psi \) satisfied by a trace

\( \phi \text{ U } \psi \) satisfied by a system (def.)

Derived Operators

- \( \Box \phi \) denotes \( \text{true U } \phi \) inelitably
- \( \square \phi \) denotes \( \neg (\Box \neg \phi) \) invariantly/globally
Comparing CTL and LTL

- \( \Box p \) (LTL) similar to AF p (CTL)
- \( \Diamond p \) (LTL) similar to AG p (CTL)

However,
- LTL cannot express reachability properties: EF p in CTL
- CTL cannot express \( \Diamond \Box p \) in LTL

- CTL* = LTL + CTL

Comparing CTL and LTL (contn.)

Why?

No subtree where ok is true everywhere

The linear-time behaviour of BUGGY

Model Checking LTL

Wolper et al. 1986

- Given an automata \( M \) and a formula \( \phi \), to check \( M \text{ sat } \phi \)
  - Construct the formula automaton: \( A(\neg \phi) \)
  - Construct the product automaton \( M \parallel A(\neg \phi) \) (on-the-fly)
  - If \( M \parallel A(\neg \phi) \) is empty then \( M \text{ sat } \phi \) otherwise NO
  - Time-Complexity = \( |M|^2|\phi| \cdot \log |\phi| \)

The same idea can be used for CTL model checking using Tree-automata

END

(of Untimed Systems)