Is the system correct?

Design Complexity

Exponential Growth – doubling of transistors every couple of years

Measuring SW Complexity

- Source Lines of Code (SLOC)
  - Measures how many lines (statements) in a program
  - Useful as a measure of software complexity

SOME SLOC Estimates:

<table>
<thead>
<tr>
<th>System</th>
<th>SLOC Estimates</th>
</tr>
</thead>
<tbody>
<tr>
<td>NASA Space Shuttle Flight Control</td>
<td>629 thousand (shuttle) + 1.4 million (ground)</td>
</tr>
<tr>
<td>Sun Solaris (1999-2000)</td>
<td>7-8 Million</td>
</tr>
<tr>
<td>Microsoft Windows 3.1 (1992)</td>
<td>3 Million</td>
</tr>
<tr>
<td>Microsoft Windows 95</td>
<td>15 Million</td>
</tr>
<tr>
<td>Microsoft Windows 98</td>
<td>18 Million</td>
</tr>
<tr>
<td>Microsoft Windows 2000</td>
<td>20 Million</td>
</tr>
<tr>
<td>Microsoft Windows XP (2002)</td>
<td>40 Million</td>
</tr>
<tr>
<td>Red Hat Linux 6.2 (2006)</td>
<td>20 Million</td>
</tr>
<tr>
<td>Red Hat Linux 7.1 (2001)</td>
<td>30 Million</td>
</tr>
</tbody>
</table>

Notable examples of system failure

In December 1996, the Ariane 5 rocket exploded 40 seconds after take off.
Cost: $400 million software failure

Mars, December 3, 1999
Crashed due to uninitialized variable
Pentium 4 Bugs Breakdown

- Intel Pentium chip, released in 1994 produced error in floating point division
- Cost: $475 million

Therac-25 Accident:
- A software failure caused wrong dosages of x-rays.
- Cost: Human Loss.

Systems are Unreliable

Checking System Correctness

- Interactive theorem proving: Formulate system correctness as a theorem in a suitable logic
  - Requires manual proofs
  - May require to test several cases
- Testing: Run the system on select inputs
  - May require to test a large amount of data
  - Used only at an advanced phase of a project
Another Approach: Formal verification

**Formal Verification:**
- System $\rightarrow$ A mathematical model $M$
- Desired behavior $\rightarrow$ A formal specification $\psi$
- Correctness $\rightarrow$ A formal technique to check that $M$ meets $\psi$

**Advantages:**
- Apply to system models
- Using them at a very early stage of a project
- Based on robust mathematical theories
- System analysis relies on the solution of some decision problems:
  - Reachability
  - Automata emptiness and containment
  - Satisfiability of logic formulas
  - Model checking
  - Module checking and games

Outline of the talk

**Model Checking**
- Discrete System Model
- Temporal logics (LTL, CTL, CTL*)

**Satisfiability of temporal logic formulas**

**Automata-theoretic approach to solve the model checking and the satisfiability problems**

**Automata on infinite objects**

**Module checking**
- Discrete Module
- Temporal logics

Model Checking

- Let $S$ be a finite-state system and $P$ its desired behavior
- $S \rightarrow$ labelled state-transition graph (automaton) $M$
- $P \rightarrow$ a temporal logic formula $\psi$

The system has the required behavior

$M$ satisfies $\psi$

An example

- A scheduler should be designed so that jobs of two users are not printed simultaneously, and whenever a user sends a job, the job is printed eventually.

- Build a mathematical model of the system:
  - what are possible behaviors?
- Write correctness requirements in a specification language:
  - what are desirable behaviors?
- Model Checking: (Automatically) check that the model satisfies the specification
An Automata-theoretic Approach to System Verification [Vardi and Wolper]

- Let $A$ describe the system $S$
- Let $\psi$ describe the specification of $S$ and $B_{\neg \psi}$ accept the computations that violate $\psi$
- $S$ is correct with respect to $\psi$ if

$$L(A) \cap L(B_{\neg \psi}) = \emptyset$$

- What we need?
  - Efficient system specification
  - Efficient automata closed under intersection
  - Efficiently decidable emptiness problem

### Decision Problems in Formal Verification

- **SYSTEM MODEL**
  - Automata on infinite objects
- **SYSTEM VERIFIER**
  - Model checking
  - Games
  - Module checking
  - Satisfiability
- **REQUIREMENTS**
  - Automata
  - Temporal Logic
  - Real time temporal logic
- **Automata-theoretic Approach**

### Finite Automata on Finite Words

$$A = \langle \Sigma, Q, Q_0, \delta, F \rangle$$

- with $F \subseteq Q$
- A run $r$ of $A$ on a finite word $\sigma$ is a finite sequence of states.
- $A$ accepts a word $\sigma$ if there exists a run $r$ of $A$ on $\sigma$ ending in a final state.
- $A$ accepts the language of the regular expression $\varepsilon + (a+b)^*a$

### Finite Automata on Infinite Words

$$A = \langle \Sigma, Q, Q_0, \delta, F \rangle$$

- $F$ may (or may not) be a subset of $Q$
- A run $r$ on an $\omega$-word $\sigma$ is an $\omega$-sequence of states.
- A run $r$ is accepting if the states occurring infinitely many times in $r$ ($\text{Inf}(r)$) satisfies $F$.
- Büchi condition: $F$ is a set of final states ($F \subseteq Q$) and a run is accepting if $\text{Inf}(r) \cap F \neq \emptyset$. As a Büchi automaton, $A$ accepts the $\omega$-language $(b^*a)^\omega$. 
Automata on Infinite Trees

- A infinite (binary) tree is a function $t: \{0,1\}^* \rightarrow \Sigma$
- Elements in $\{0,1\}^*$ are nodes
- Empty word $\varepsilon$ is the root

Tree Automata

- $A = \langle \Sigma, Q, Q_0, \delta, F \rangle$
  - $\delta$ - Transition relation on trees
  - $F$ - Acceptance condition.

A run of a tree automata over a tree is also a tree where labels are elements of $Q$ in accordance with $\delta$ and the root is labelled with an initial state.

Example.

$\delta(q_0,a) = \{(q_1,q_3),(q_2,q_4)\}$

- A tree $t$ is accepted by $A$ if there exists a run $r(t)$ of $A$ on $t$ such that all paths $\pi$ of $r(t)$ "infinitely often" satisfy $F$
- $L(A)$ : Language accepted by $A$

Temporal Logic

SYSTEM MODEL

- Discrete Automata
- Timed Automata

SYSTEM ANALYSIS

- Model-checking
- Games - Module Checking
- Satisfiability

REQUIREMENTS

- Automata
- Temporal logic
- Real time temporal logic
Temporal Logic

- Correctness requirements for open (reactive) systems

- Mostly used:
  - **LTL** (Linear Temporal Logic) 
    [Pnueli 1977]
  - **CTL** (Branching Temporal Logic) 
    [Emerson and Clarke 1982]
  - **CTL** (Full Branching Temporal Logic) 
    [Emerson and Halpern 1986]

Temporal logic (LTL)

- A logical notation that allows to:
  - specify relations in time
  - conveniently express finite control properties

- Syntax
  
  \[ \varphi ::= p \mid \neg \varphi \mid \varphi \land \varphi \mid X \varphi \mid F \varphi \mid G \varphi \mid \varphi U \varphi \]

- Temporal operators
  
  - \(X p\) "p at the next time"
  - \(F p\) "eventually p"
  - \(G p\) "henceforth p"
  - \(p U q\) "p until q"

- Semantics:

  \[ p U q: p p p q \]

Types of temporal properties

- Safety (nothing bad happens)
  - \(G \neg(\text{ack1} \land \text{ack2})\) "mutual exclusion"
  - \(G \neg(\text{req} \land \neg\text{ack})\) "req must hold until ack"

- Liveness (something good happens)
  - \(G (\text{req} \Rightarrow F \text{ack})\) "if req, eventually ack"

- Fairness
  - \(GF \neg(\text{req}) \Rightarrow GF \neg(\text{ack})\) "if infinitely often req, infinitely often ack"

A branching time temporal logic: CTL

[Emerson and Clarke 1982]

- Syntax
  
  \[ \varphi ::= p \mid \neg \varphi \mid \varphi \lor \varphi \mid \exists X \varphi \mid \forall X \varphi \mid \exists ! \varphi \mid \forall ! \varphi \mid \exists ! \varphi \mid \forall ! \varphi \]

- where \(p\) is an atomic proposition

- Semantics: with respect to a 2AP-labelled tree

- Example: \(\exists p U q\)

- Some abbreviations
  
  - \(\exists ! p = \exists \text{True} U p\)
  - \(\exists ! p = \exists ! \neg p\)
  - \(\forall ! p = \forall ! \text{True} U p\)
  - \(\forall ! p = \forall ! \neg p\)
Example: traffic light controller

- Guarantee no collisions
- Guarantee eventual service

Specifications

- Safety (no collisions)
  \[ \forall G \neg (E_{Go} \land (N_{Go} \lor S_{Go})) ; \]

- Liveness
  \[ \forall G (\neg N_{Go} \land N_{Green} \Rightarrow \forall F N_{Go}) ; \]
  \[ \forall G (\neg S_{Go} \land S_{Green} \Rightarrow \forall F S_{Go}) ; \]
  \[ \forall G (\neg E_{Go} \land E_{Green} \Rightarrow \forall F E_{Go}) ; \]

Decision problems in Temporal Logics

**Satisfiability**

- Given a CTL formula \( \phi \), is there a tree satisfying \( \phi \)?
  - Examples:
    - \( \forall p U q \) is satisfiable
    - \( \exists p \land \neg p \) is not satisfiable
  - Given a CTL formula \( \phi \), it is possible to build a BTA \( A_\phi \) (generator for CTL) with \( O(2^{18}) \) states accepting all infinite trees that satisfy \( \phi \) [Vardi and Wolper 1986]
  - A CTL formula \( \phi \) is satisfiable iff \( L(A_\phi) \neq \Phi \).
  - The emptiness problem for BTA is LOGSPACE-complete for PTIME [Vardi and Wolper 1986]
  - The satisfiability problem can be solved in exponential time

Decision Problems Using Automata

**Model Checking**

- Given a system \( S \) and a specification \( \phi \), using a tree \( t \) as model of \( S \), we determine whether \( t \) satisfy \( \phi \) (\( t \models \phi \)).

- Automata-theoretic approach: using an automaton \( A_S \) as model of \( S \) and an automaton \( A_{\neg \phi} \) describing the complementation of \( \phi \). \( S \) is correct with respect to \( \phi \) iff

\[ L(A_S) \cap L(A_{\neg \phi}) = \Phi \]
### Complexity Results

<table>
<thead>
<tr>
<th>Class</th>
<th>Model Checking</th>
<th>Satisfiability</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTL</td>
<td>Linear Time [3]</td>
<td>EXPTime-Complete</td>
</tr>
</tbody>
</table>

1. [Sistla and Clarke 1984]
2. [Emerson and Lei 1985]
3. [Clarke, Emerson, and Sistla 1986]
4. [Emerson, Sistla 1984]
5. [Emerson and Jutla 1988]