

Introduction to Formal Verification

Aniello Murano

Università degli studi di Napoli "Federico II"
Dipartimento di Scienze Fisiche
Sezione di Informatica

22 Maggio, 2006

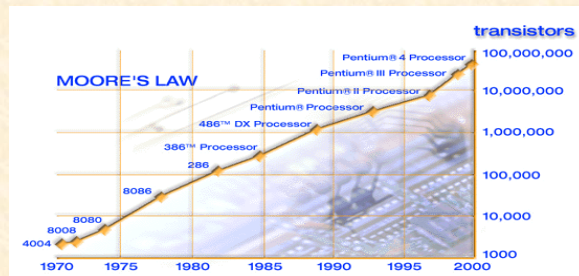
1

Is the system correct?



2

Design Complexity



Exponential Growth – doubling of transistors every couple of years

3

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:

NASA Space Shuttle flight Control	420 thousand (shuttle) + 1.4 million (ground)
Sun Solaris (1999-2000)	7-8 Million
Microsoft Windows 3.1 (1992)	3 Million
Microsoft Windows 95	15 Million
Microsoft Windows 98	18 Million
Microsoft Windows 2000	20 Million
Microsoft Windows XP (2002)	40 Million
Red Hat Linux 6.2 (2000)	20 Million
Red Hat Linux 7.1 (2001)	30 Million

Sources: D. Wheeler, "More Than A Gigabuck: Estimating GNU/Linux's Size", <http://www.dewheeler.com/sloc/>; Wikipedia (wikipedia.org).

4

System Failure

❑ Safety



❑ Money



❑ System Release



❑ Market reputation



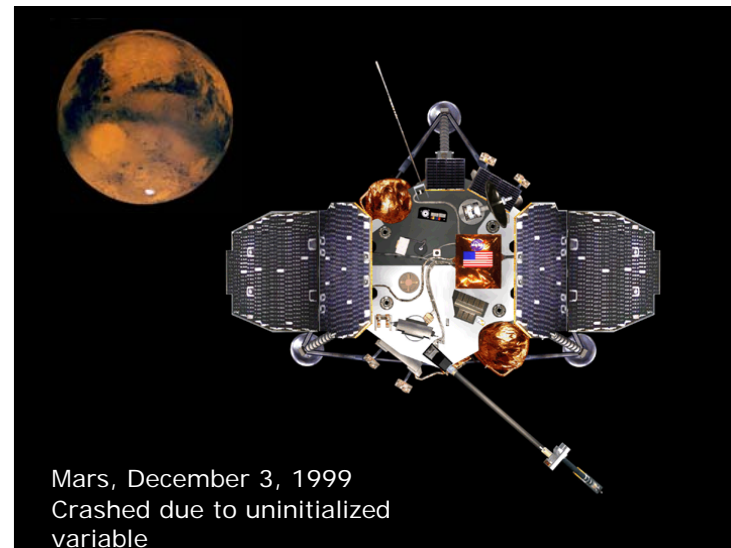
5

Notable examples of system failure

6

In December 1996, the Ariane 5 rocket exploded 40 seconds after take off.

Cost : \$400 million software failure

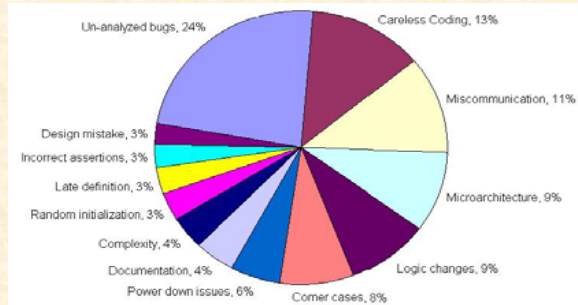


Mars, December 3, 1999
Crashed due to uninitialized variable

Pentium 4 Bugs Breakdown

❑ Intel Pentium chip, released in 1994 produced error in floating point division

❑ Cost : \$475 million



9

❑ **Therac-25 Accident :**

- ❖ A software failure caused wrong dosages of x-rays.
- ❖ Cost: Human Loss.

10

Windows

An exception 06 has occurred at 0028:C11B3ADC in %d DiskTSD(03) + 00001660. This was called from 0028:C11B40C8 in %d voltrack(04) + 00000000. It may be possible to continue normally.

- * Press any key to attempt to continue.
- * Press CTRL+ALT+RESET to restart your computer. You will lose any unsaved information in all applications.

Press any key to continue

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

12

Another Approach: Formal verification

□ Formal Verification:

- ◆ System → A mathematical model M
- ◆ Desired behavior → A formal specification ψ
- ◆ Correctness → A formal technique to check that M meets ψ

□ 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

13

Outline of the talk

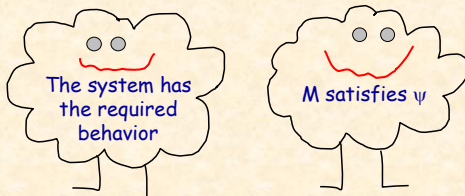
- Model Checking
 - ◆ Discrete System Model
 - ◆ Temporal logics (LTL, CTL, CTL*)
- Satisfiability of temporal logic formulas
- Automata-theoretic approach to solve the model checking and the satisfiability problems
- Automata on infinite objects
- Module checking
 - ◆ Discrete Module
 - ◆ Temporal logics

14

Model Checking

□ Let S be a finite-state system and P its desired behavior

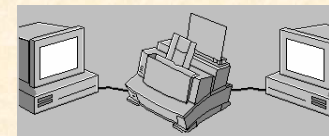
- S → labelled state-transition graph (automaton) M
- P → a temporal logic formula ψ



15

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.

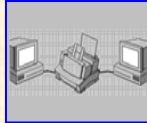


- Build a **mathematical model** of the system:
 - ◆ what are possible behaviors?
- Write **correctness requirements** in a specification language:
 - ◆ what are desirable behaviors?
- Model Checking: (Automatically) **check** that the model satisfies the specification

16

An Automata-theoretic Approach to System Verification [Vardi and Wolper]

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



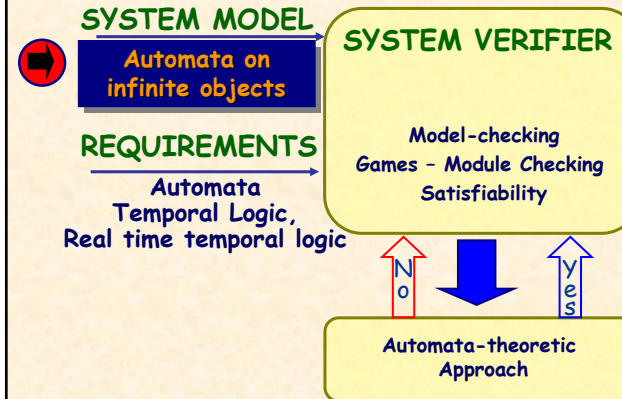
$$L(A) \cap L(B_{\neg\psi}) = \emptyset$$



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

17

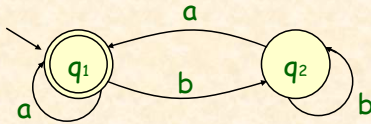
Decision Problems in Formal Verification



18

Finite Automata on Finite Words

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

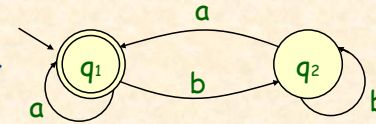


- A run r of A on a finite words σ is a finite sequence of states.
- A accepts a word σ if there exists a run r of A on σ ending in a final state.
- A accepts the language of the regular expression $\varepsilon + (a+b)^*a$

19

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 w -word σ is an w -sequence of states.
- A run r is accepting if the states occurring infinitely many times in r ($\text{Inf}(r)$) satisfies F .



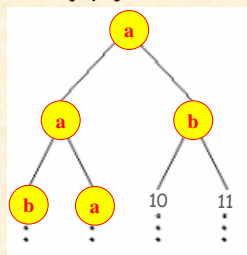
- ◆ Büchi condition: F is a set of final states ($F \subseteq Q$) and a run is accepting if $\text{Inf}(r) \cap F \neq \emptyset$. As a Buchi automaton, A accepts the w -language $(b^*a)^w$.

20




Automata on Infinite Trees

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




21

Tree Automata




- $A = \langle \Sigma, Q, Q_0, \delta, F \rangle$
 - δ - 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 δ and the root is labelled with an **initial state**.

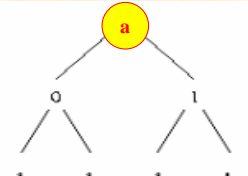


22

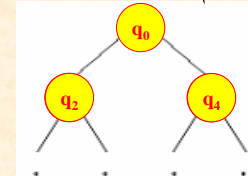
Example.



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



a tree t

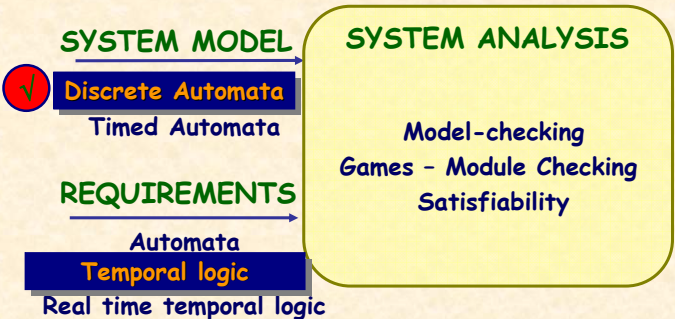


a run $r(t)$

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

23

Temporal Logic



24

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]

25

Temporal logic (LTL)

□ A logical notation that allows to:

- ◆ specify relations in time
- ◆ conveniently express finite control properties

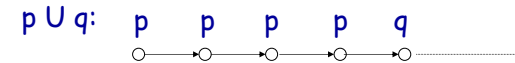
□ Syntax

$\phi := p \mid \neg \phi \mid \phi \vee \psi \mid \phi \wedge \psi \mid X\phi \mid F\phi \mid G\phi \mid \phi U \psi$

□ Temporal operators

- ◆ $X p$ "p at the next time"
- ◆ $F p$ "eventually p"
- ◆ $G p$ "henceforth p"
- ◆ $p U q$ "p until q"

□ Semantics:



Types of temporal properties

□ **Safety** (nothing bad happens)

- $G \neg(\text{ack1} \wedge \text{ack2})$ "mutual exclusion"
- $G(\text{req} U \text{ack})$ "req must hold until ack"

□ **Liveness** (something good happens)

- $G(\text{req} \Rightarrow F \text{ack})$ "if req, eventually ack"

□ **Fairness**

- $GF \text{req} \Rightarrow GF \text{ack}$ "if infinitely often req, infinitely often ack"

27

A branching time temporal logic: CTL [Emerson and Clarke 1982]

Syntax $\phi := p \mid \neg \phi \mid \phi \vee \psi \mid \exists X\phi \mid \forall X\phi \mid \exists \phi U \psi \mid \forall \phi U \psi$

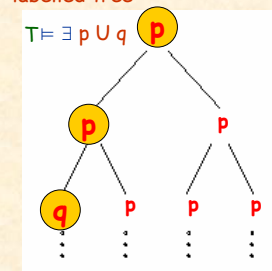
where p is an atomic proposition

Semantics: with respect to a 2^{AP} -labelled tree

Example: $\exists p U q$

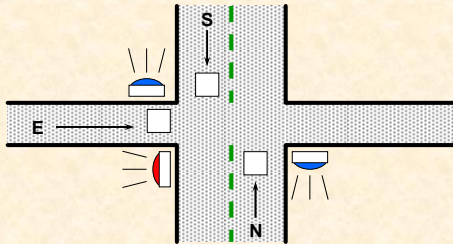
Some abbreviations

- $\exists F p = \exists \text{True} U p$
- $\exists G p = \neg \forall F \neg p$
- $\forall F p = \forall \text{True} U p$
- $\forall G p = \neg \exists F \neg p$



28

Example: traffic light controller



- Guarantee no collisions
- Guarantee eventual service

29

Specifications

- Safety (no collisions)

$$\forall G \neg (E_Go \wedge (N_Go \mid S_Go));$$

- Liveness

$$\forall G (\neg N_Go \wedge N_Green \Rightarrow \forall F N_Go);$$

$$\forall G (\neg S_Go \wedge S_Green \Rightarrow \forall F S_Go);$$

$$\forall G (\neg E_Go \wedge E_Green \Rightarrow \forall F E_Go);$$

30

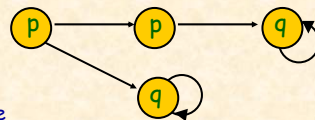
Decision problems in Temporal Logics

Satisfiability

- Given a CTL formula φ , is there a tree satisfying φ ?

Examples:

- ◆ $\forall p \ U \ q$ is satisfiable



- ◆ $\exists G(p \wedge \neg p)$ is not satisfiable

- Given a CTL formula φ , it is possible to build a BTA A_φ (generator for CTL) with $O(2^{|\varphi|})$ states accepting all infinite trees that satisfy φ [Vardi e Wolper 1986]
- A CTL formula φ is satisfiable iff $L(A_\varphi) \neq \Phi$.
- The **emptiness problem** for BTA is **LOGSPACE-complete** for **PTIME** [Vardi and Wolper 1986]
- The satisfiability problem can be solved in exponential time

31

Decision Problems Using Automata

Model Checking

- Given a system S and a specification φ , using a tree t as model of S , we determine whether t satisfy φ ($t \models \varphi$).
- Automata-theoretic approach: using an automaton A_S as model of S and an automaton $A_{\neg\varphi}$ describing the complementation of φ , S is correct with respect to φ **iff**

$$L(A_S) \cap L(A_{\neg\varphi}) = \Phi$$



32

Complexity Results

Class	Model Checking	Satisfiability
LTL	PSpace-Complete [1]	PSpace-Complete [1]
CTL	Linear Time [3]	EXPTIME-Complete
CTL*	PSpace-Complete [2]	2EXPTIME-Complete [4,5]

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]