Introduction to verification

Anca Muscholl, Marc Zeitoun

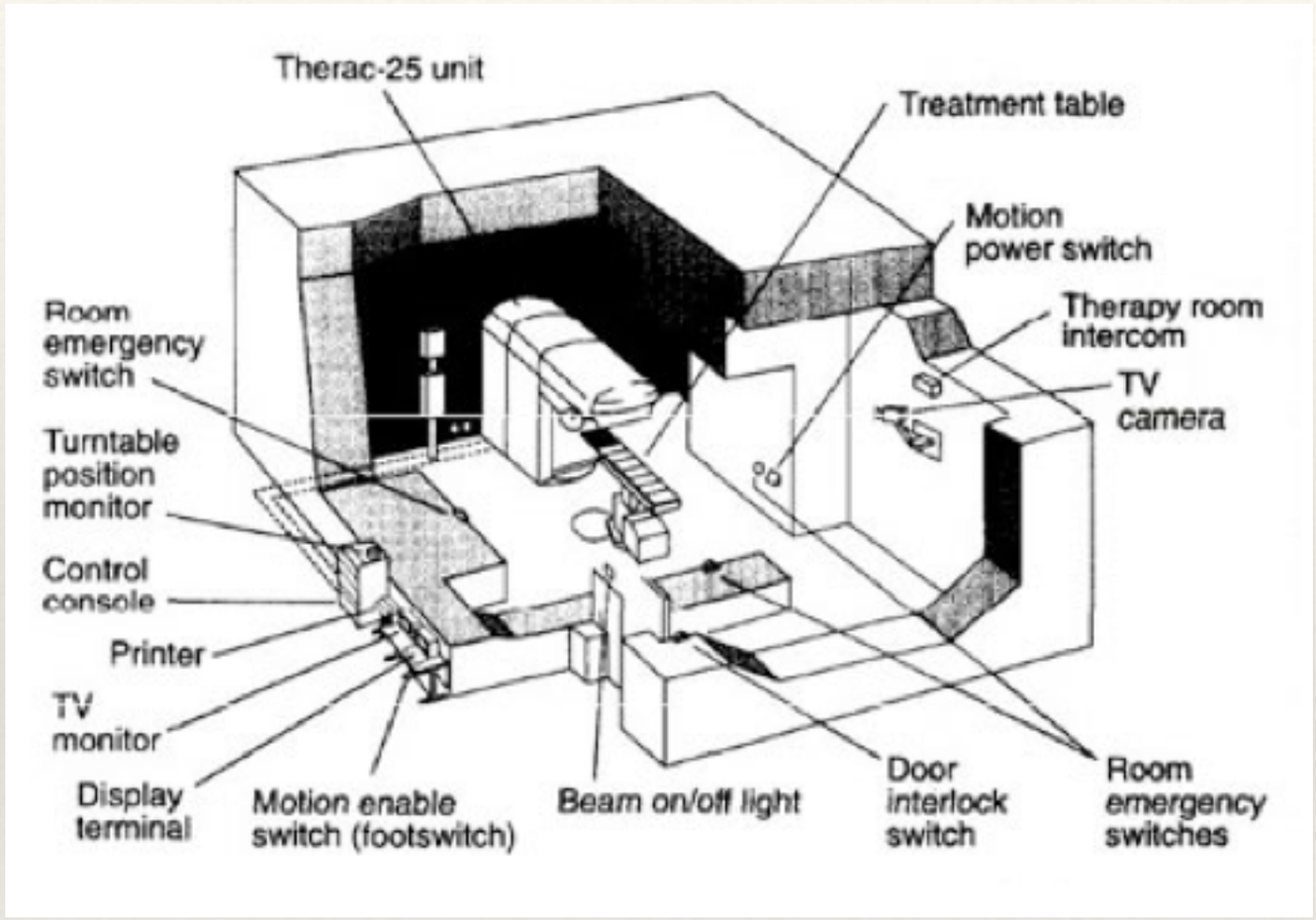
Bibliography: Principles of Model-Checking, C. Baier and J.-P. Katoen, MIT Press 2008

Factorial(x) := if x=0 then 1 else x*Factorial(x-1)

Is this program correct?

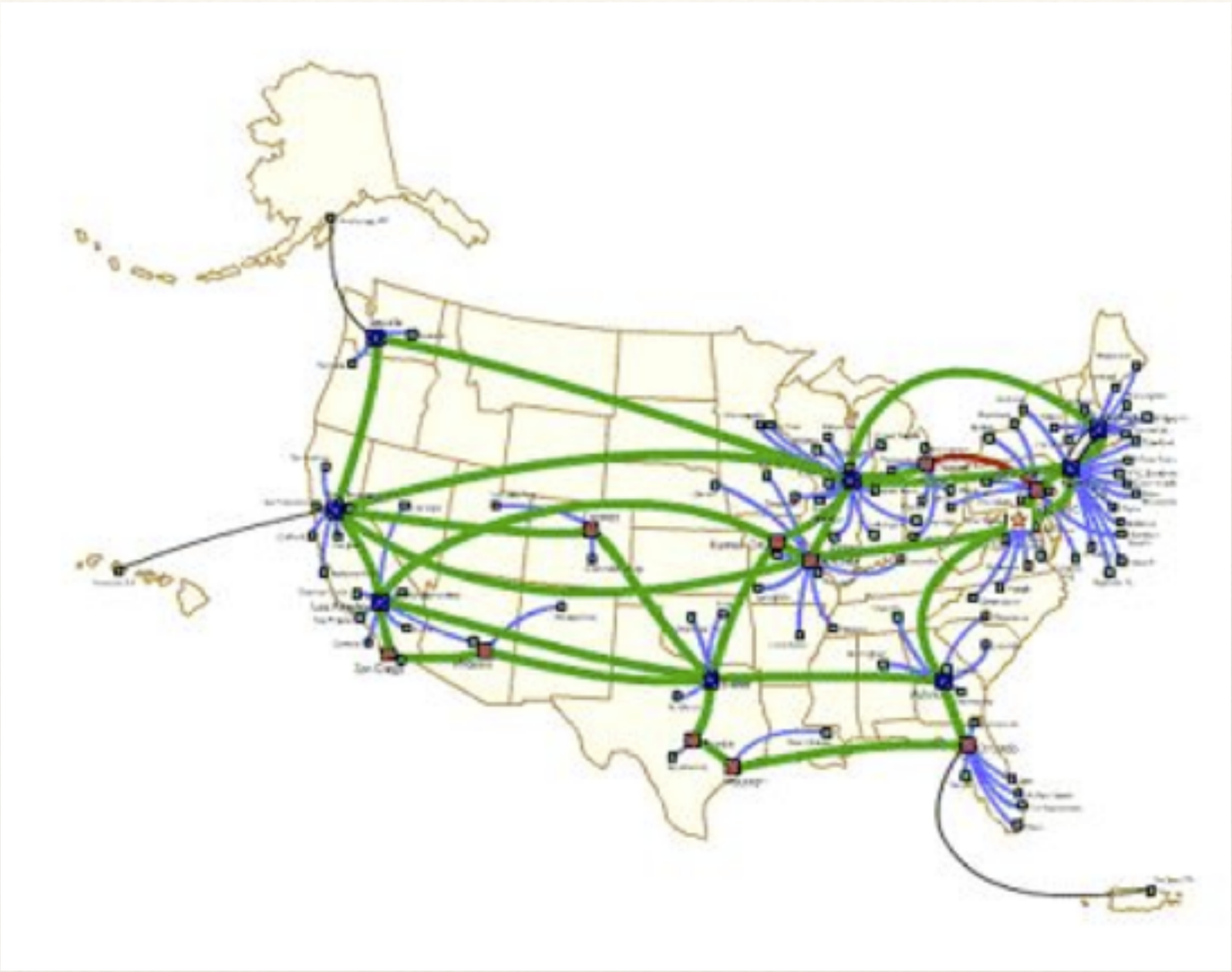
Some examples where it matters

Therac-25 Radiation Overdosing (1985-87)



- Radiation machine for treatment of cancer patients
- At least 6 cases of overdoses in period 1985-1987
- Three death cases
- Source: Design error in the control software (race condition)

AT&T Telephone Network Outage (1990)

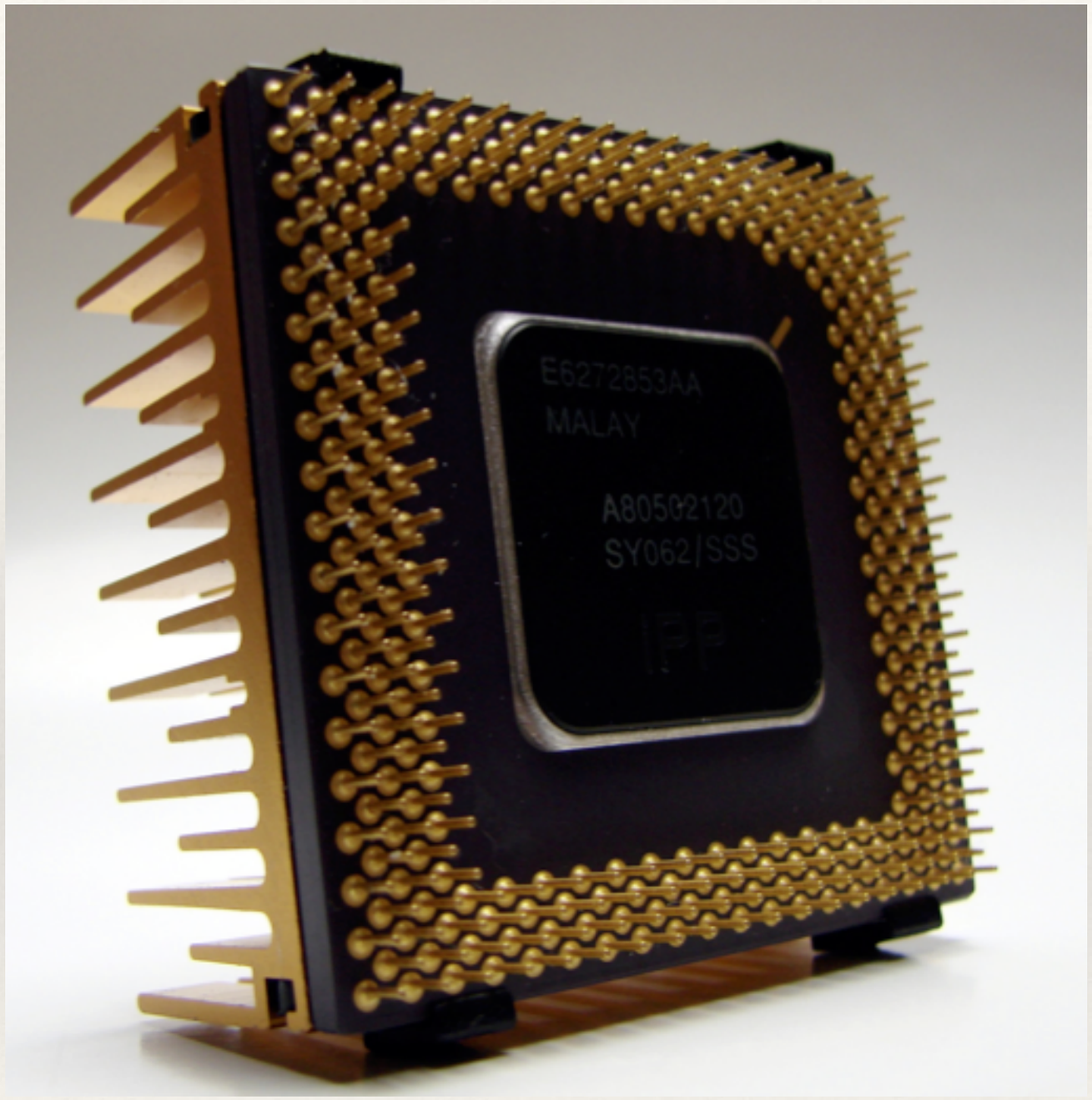


- 9 hours outage of large parts of US telephone network
- Cost: several 100 million $
- Source: software flaw (wrong interpretation of break statement in C)

Ariane 5 Crash (1996)

- Crash of Ariane 5 missile in June 1996
- Cost: more than 500 million $
- Source: software flaw
- A data conversion from 64-bit floating to 16-bit signed integer

Pentium FDIV Bug (1994)



- FDIV= floating point division unit
- 1 in 9 billion floating point dividers would produce inaccurate results
- Cost: 500 million $ in replaced processors
- Source flaw in a division table

Why it is difficult to verify computer systems?

- Analog systems are continuous
- Digital systems are discrete
- Big number of components interacting together

Some analysis

"The role of software in recent Aerospace Accidents" (2001)

Nancy G. Leveson
Aeronautic and Astronautic Department
MIT

Engineers often underestimate the complexity of software and overestimate the effectiveness of testing.

Increasingly: *system accidents* that result from dysfunctional interactions among components, not from individual component failure.

Accidents analyzed

- Explosion of Ariane 5
- Loss of Mars Climate Orbiter
- Destruction of Mars Polar Lander
- Placing Milstar satellite in an incorrect orbit
- American Airlines B-757 crash into a mountain near Cali
- Collision of Lufthansa A320 into earth bank in Warsaw
- Crash of China Airlines A320 near Nagoya

- Overconfidence on digital automation
- Not understanding the risks associated to software
- Almost all errors were due to flaws in specification and not in coding
- Reliability techniques (like redundancy) not effective for software
- Assuming the risk decreases over time
- Inadequate specifications
- Flawed review process
- Inadequate safety engineering
- 50%-70% safety decisions are made in early stages of development
- Software reuse without safety analysis
- Unnecessary complexity of software (“keep it simple!”)

Some analysis

Why does cryptographic software fail? A case study and open problems

David Lazar, Haogang Chen, Xi Wang, and Nickolai Zeldovich
MIT CSAIL, 2014

83% of bugs are misuses of cryptographic libraries by individual applications

Bugs analyzed

• Apple's goto bug in its SSL/TLS implementation (additional goto statement)

• Goto bug in GnuTLS's certificate validation code (secure communications library implementing SSL, TLS, DTLS)

• 269 cryptographic vulnerabilities reported in the CVE database (2011 - 2014).

Nature of bugs

- Plaintext disclosure (not using HTTPS for login, storing passwords in plaintext)

- Man-in-the-middle attacks (authentication errors, see Apple and OS X)

- Brute-force attacks (low encryption strength, insufficient randomness)

- Side-channel attacks (information leakage)

- Testing: high code coverage difficult to achieve (ex: test vectors use 7-bit ASCII, not sufficient for Unicode)

- Static analysis (catching errors at compiler time): do not offer strong guarantees, as they do not catch behavioral errors

- Formal verification: relies on SAT solvers (but cannot handle inputs of variable length)

Formal verification at work

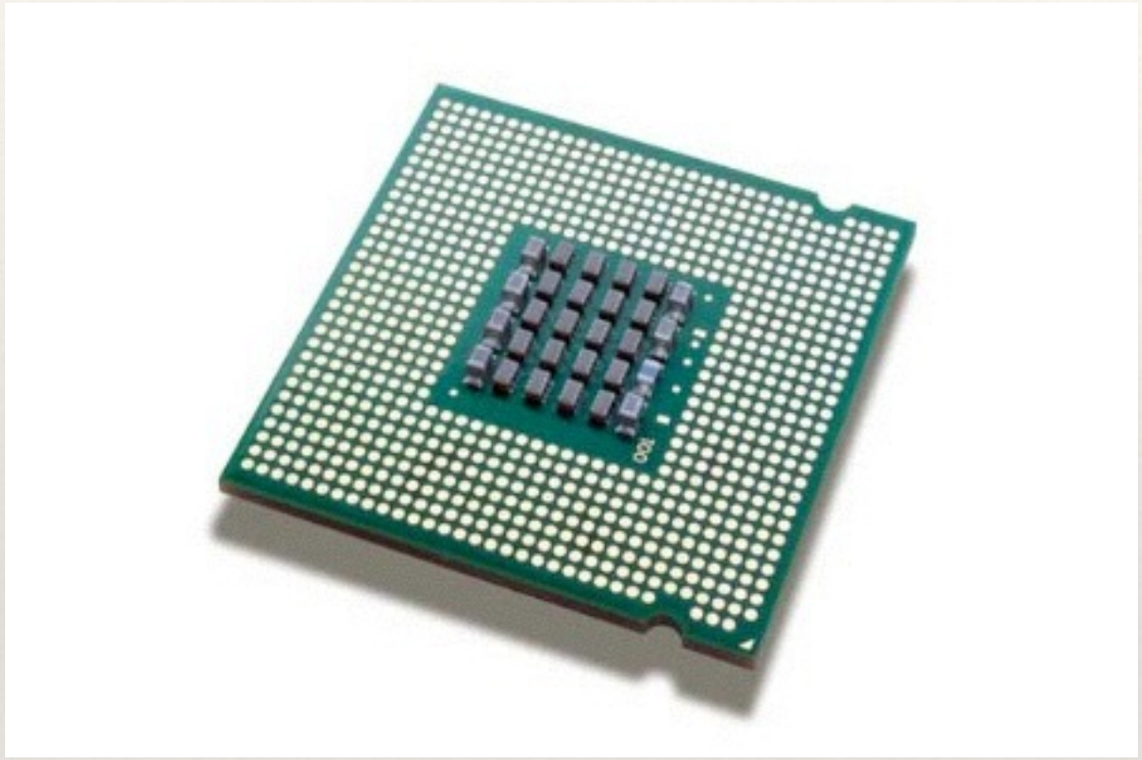
Signalling system for RER

- Increase traffic by 25% preserving safety levels
- 21K of Modula-2 code have been formally specified and verified using B method
- Later the same method has been used for line 14 and Roissy Shuttle
- No unit test were preformed, just some global tests

B method

	Paris Métro Line 14	Roissy Shuttle
Line length (km)	8.5	3.3
Number of stops	8	5
Inter-train time (s)	115	105
Speed (km/hr)	40	26
Number of trains	17	14
Passengers/day	350,000	40,000
Number of lines of Ada	86,000	158,000
Number of lines of B	115,000	183,000
Number of proofs	27,800	43,610
Interactive proof percentage	8.1	3.3
Interactive proof effort (person-months)	7.1	4.6

AAMP Microprocessor



- AAMP5 widely used processor (Rockwell Collins)
- .5 M transistors
- Completely verified in PVS (300 hours per instruction)
- Later verified AAMP-FV showing dramatic reduction in verification costs
- National Security Agency certification for use in cryptographic applications.

Airbus

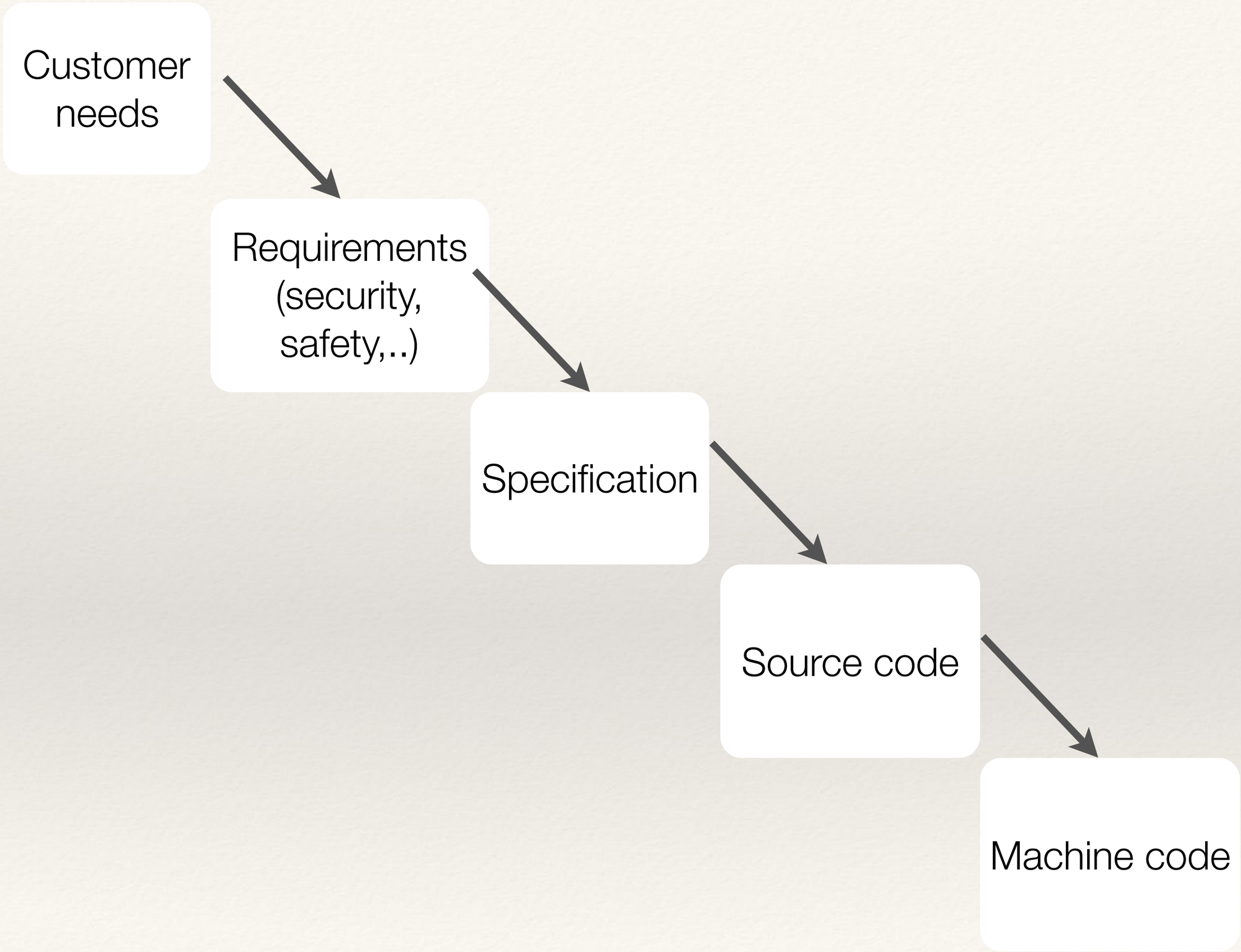
- Development of Level A controllers for A340/600 series (Esterel technologies)
- 70% of code generated automatically
- Quick management of requirements changes
- Major productivity improvement (each new project requires twice as much software as its predecessor)
- SCADE has been adopted for A380 for most of on-board computers.

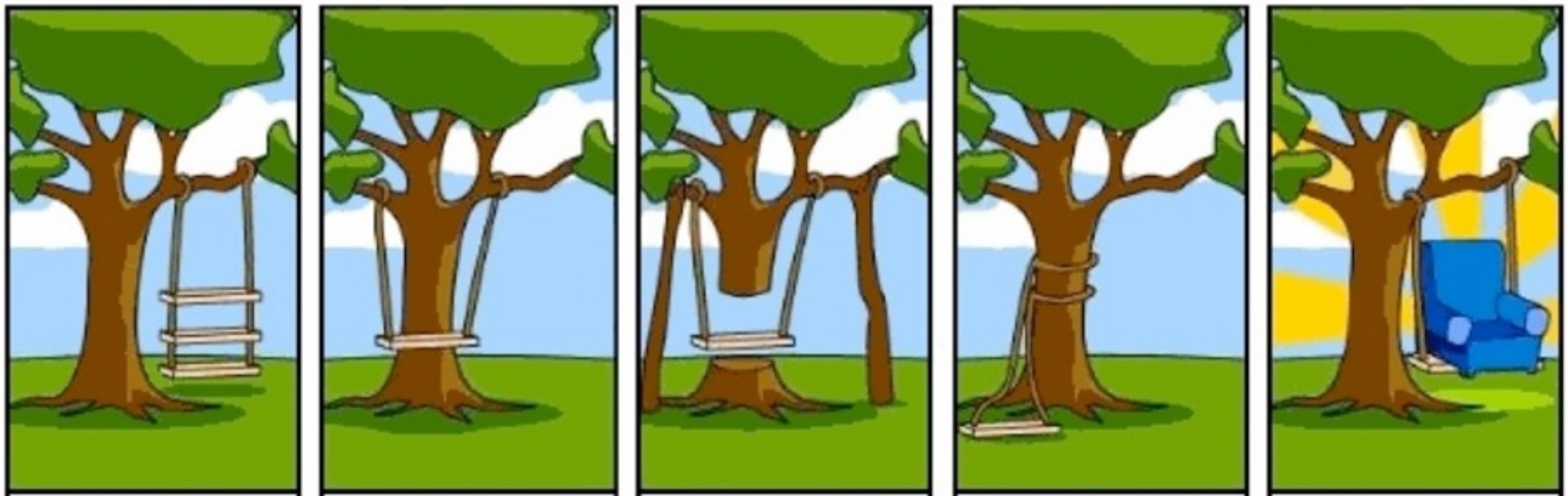
What are formal methods

Specification

+

Analysis of the system


Customer needs
Requirements (security, safety,..)
Specification
Source code
Machine code


How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed

Specificaion

Giving precise statement of what the system has to do, while avoiding constraints on how it is achieved.

Ex: No deadlock, termination, no crash etc

Limits: Component based software development

One of the most widely recognized problems in software development is the difficulty of clearly specifying expected software behavior.

Ensuring that component-based software is reliable is difficult also because source code is often not available for components that have been bought.

Methods of system verification

Verification is impossible (algorithmically)

Halting problem for Turing machines

Alan Turing (1912-1954)
Mathematician, Logician, crypto-specialist
Computational model: Turing machine

Program termination is not decidable:
There is no algorithm to decide if a TM stops.

Verification is impossible (algorithmically)

But we have no choice

Peer reviewing

- Manual code inspection.
- On average 60% of errors caught.
- Subtle errors (concurrency, algorithm defects) hard to catch.
- Used in 80% of all software engineering projects.
- Refinement of this method: parallel development

Testing

30%-50% of development cost.

Programmers have to provide insights what to test, and what should be system response.

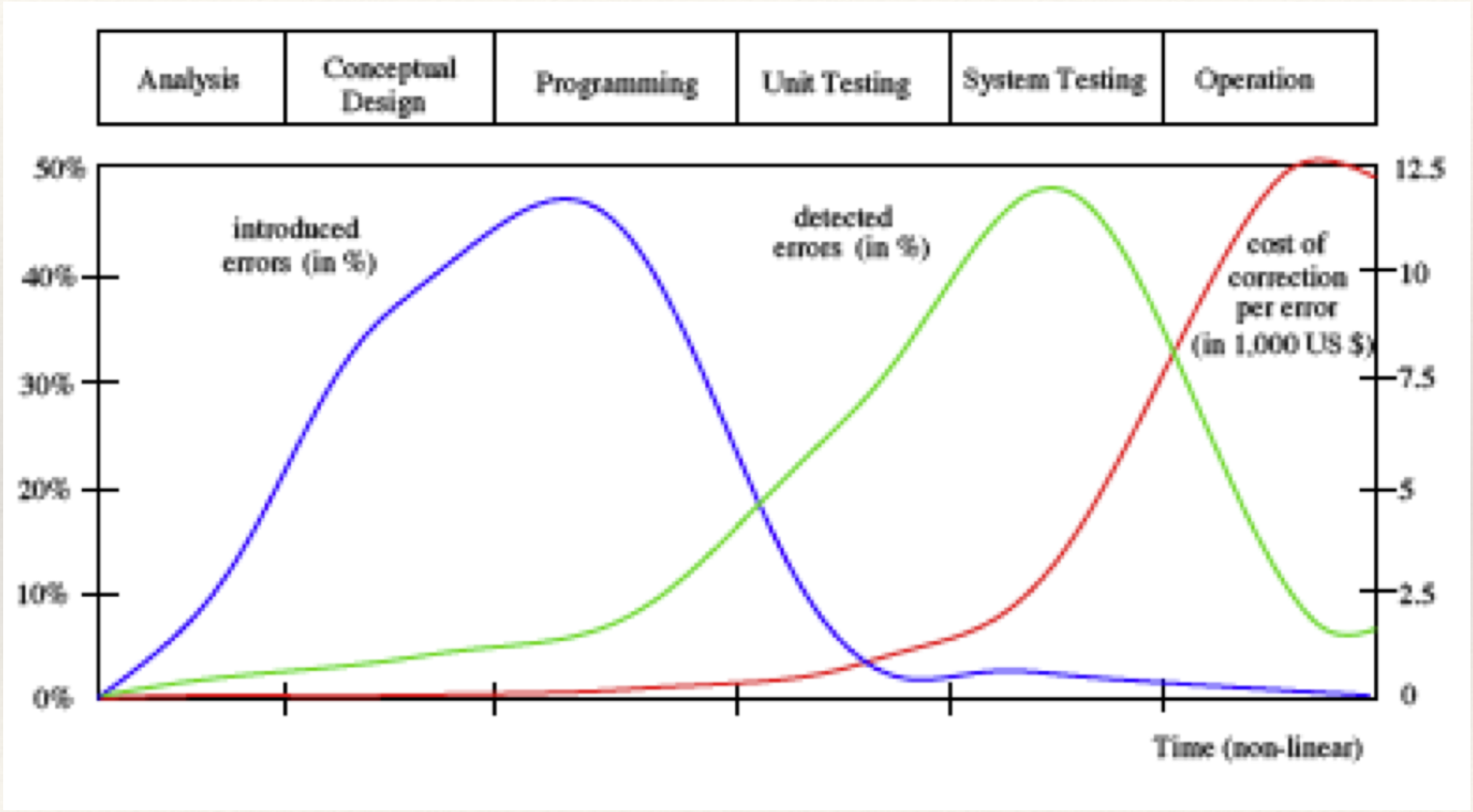
When to stop testing?

Formal specifications help here:

One of the most cost-effective uses of specifications

New tools provide as good coverage as manual test cases. They avoid programming test cases

Get them as soon as you can



Theorem proving

Doing large proofs semi-automatically

Backward axiom

$$\frac{}{\{\mathcal{A}[e/x]\}\ x := e\ \{\mathcal{A}\}}$$

Invariant rule

$$\frac{\{\mathcal{I} \wedge b\}\ P\ \{\mathcal{I}\}}{\{\mathcal{I}\}\ \textbf{while}\ b\ \textbf{do}\ P\ \{\mathcal{I} \wedge \neg b\}}$$

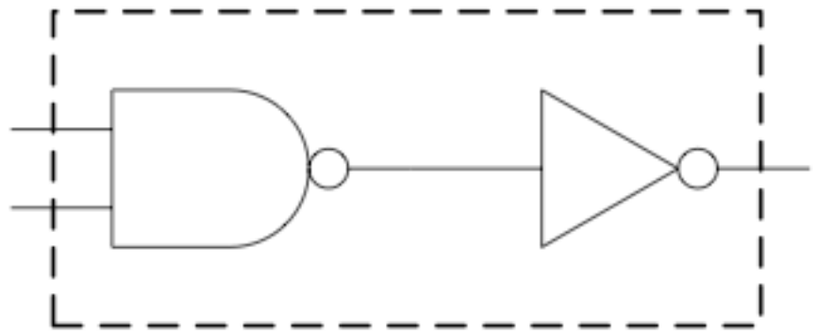
Cut rule

$$\frac{\{\mathcal{A}\}\ P\ \{\mathcal{B}\} \qquad \{\mathcal{B}\}\ Q\ \{\mathcal{C}\}}{\{\mathcal{A}\}\ P; Q\ \{\mathcal{C}\}}$$

Logical rule

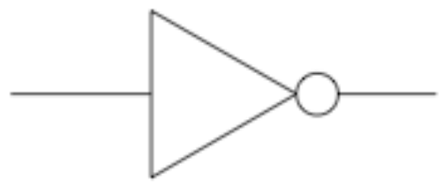
$$\frac{\mathcal{A} \Rightarrow \mathcal{A}' \quad \{\mathcal{A}'\}\ P\ \{\mathcal{B}'\} \quad \mathcal{B}' \Rightarrow \mathcal{B}}{\{\mathcal{A}\}\ P\ \{\mathcal{B}\}}$$

Constructive logics (type theory): PVS, COQ,Isabelle



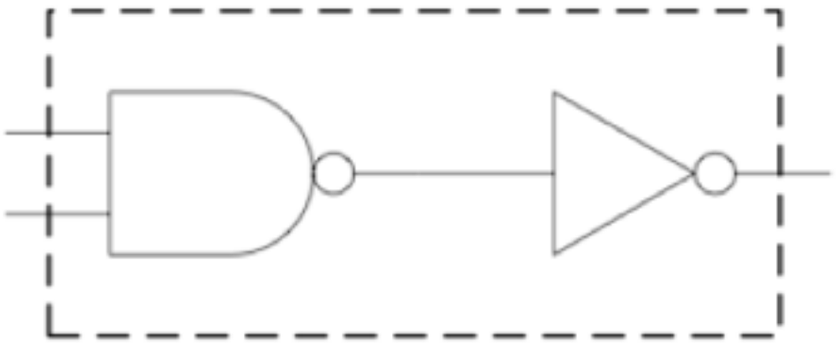
- **AND Specification:**
  - AND_SPEC (i1,i2,out) := out = i1 $\wedge$ i2

- **NAND specification:**
  - NAND (i1,i2,out) := out = $\neg$(i1 $\wedge$ i2)



- **NOT specification:**
  - NOT (i, out) := out = $\neg$ I

- **AND Implementation:**
  - AND_IMPL (i1,i2,out) := $\exists$x. NAND (i1,i2,x) $\wedge$ NOT (x,out)



- **Proof Goal:**
  - ∀ i1, i2, out. AND_IMPL(i1,i2,out) ⇒ ANDSPEC(i1,i2,out)

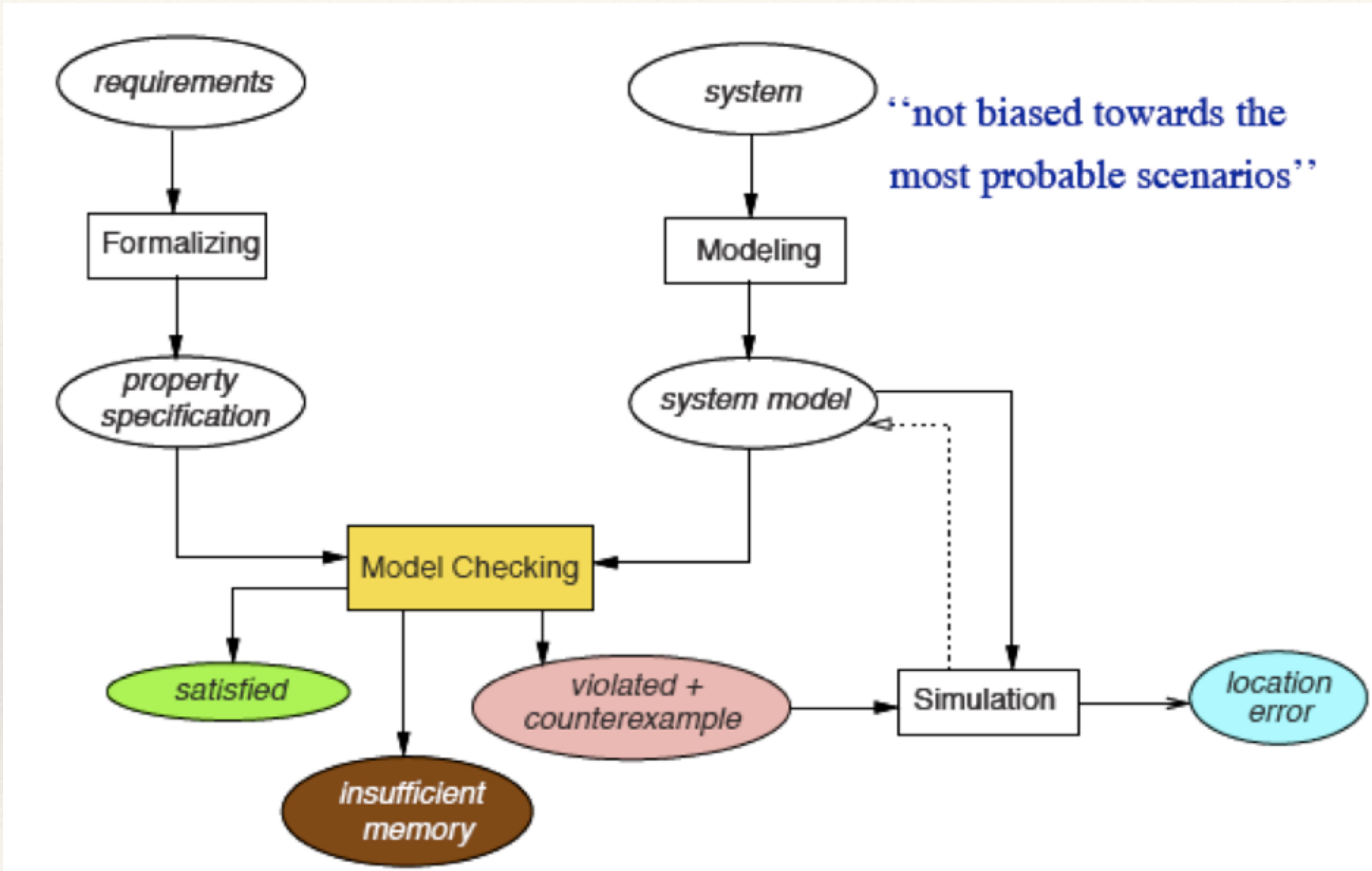
- **Proof (forward)**

AND_IMP(i1,i2,out) {from above circuit diagram}
- ⊢ ∃ x. NAND (i1,i2,x) ∧ NOT (x,out) {by def. of AND impl}
- ⊢ NAND (i1,i2,***x***) ∧ NOT(***x***,out) {strip off "∃ x."}
- ⊢ NAND (i1,i2,***x***) {left conjunct of line 3}
- ⊢ x =¬ (i1 ∧ i2) {by def. of NAND}
- ⊢ NOT (*x*,out) {right conjunct of line 3}
- ⊢ out = ¬ x {by def. of NOT}
- ⊢ out = ¬(¬(i1 ∧ i2) {substitution, line 5 into 7}
- ⊢ out =(i1 ∧ i2) {simplify, ¬¬ t=t}
- ⊢ AND (i1,i2,out) {by def. of AND spec}
- ⊢ AND_IMPL (i1,i2,out) ⇒ AND_SPEC (i1,i2,out)

Q.E.D.

Model Checking

Model checking flow-graph



The ACM Turing Award in 2007 for model-checking

Some Turing Award Winners

- Edsger Dijkstra (1972)
- Donald Knuth (1974)
- Michael Rabin and Dana Scott (1976)
- Tony Hoare (1980)
- Thompson & Ritchie (1983)
- Hopcroft & Tarjan (1986)
- Rivest, Shamir, Adleman (2002)

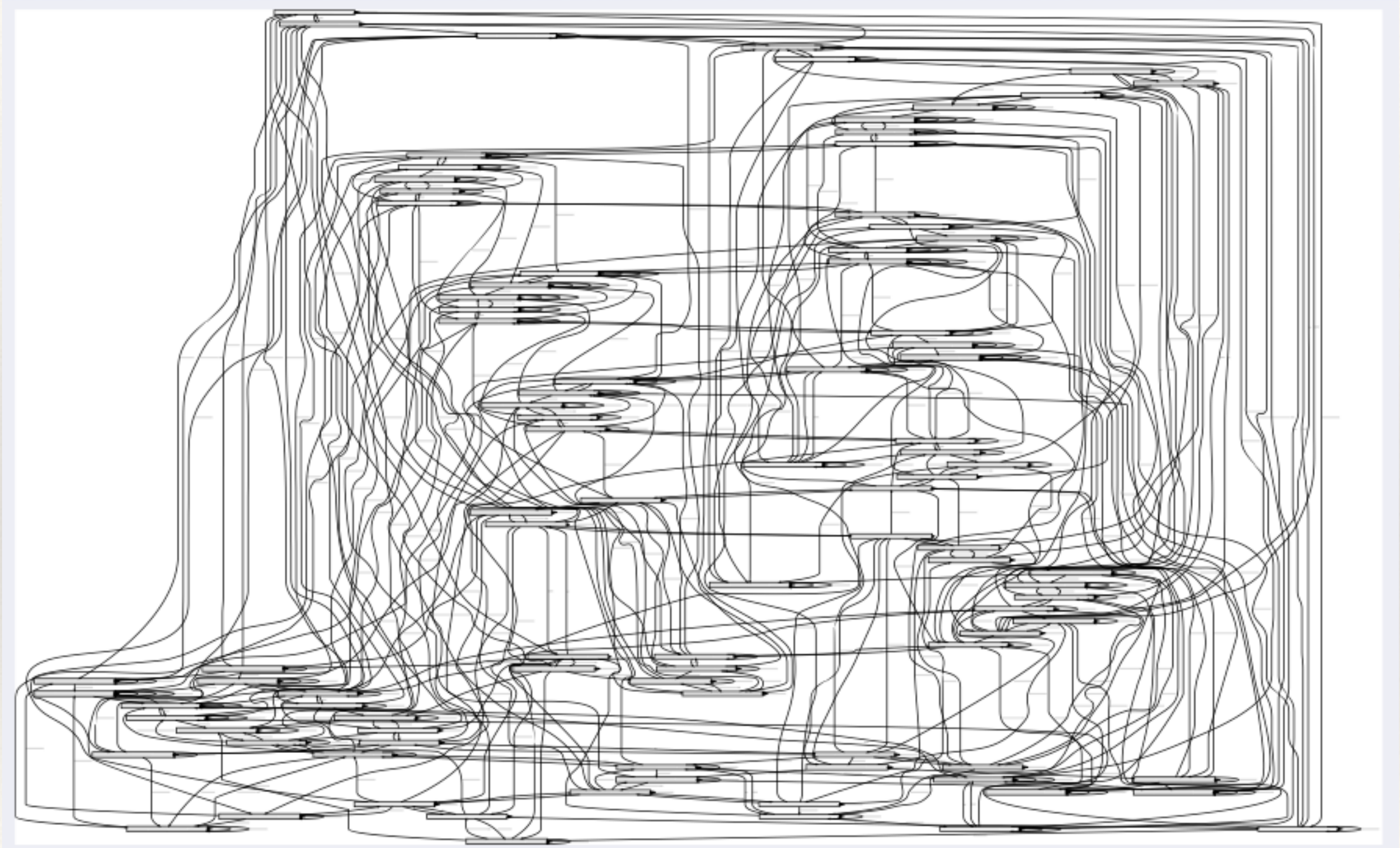
The ACM Turing Award in 2007

Edmund M. Clarke Jr. (CMU USA)
Allen E. Emerson (U. of Texas at Austin, USA)
Joseph Sifakis (IMAG, Grenoble)

Jury justification:

For their roles in developing Model-Checking into a highly effective verification technology, widely adopted in the hardware and software industries

A transition system (model)



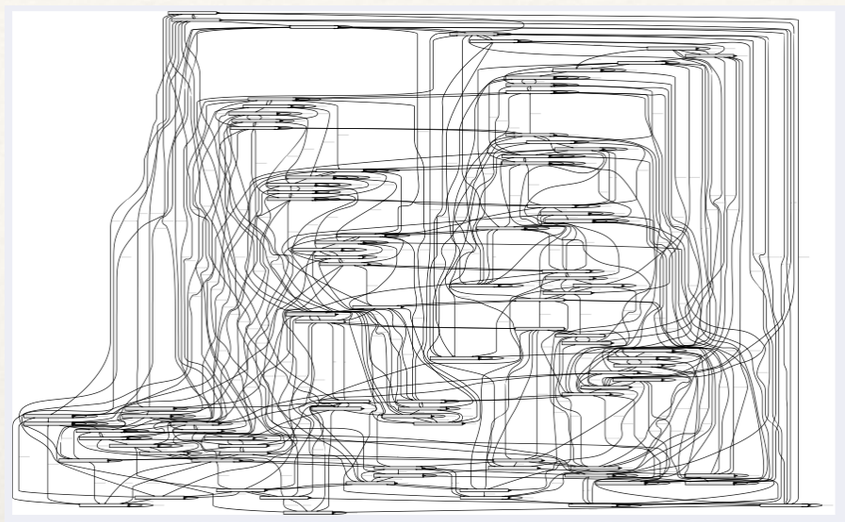
A property

A sequence of events is correct

Mutual exclusion

No deadlock

No starvation



No starvation

Model Checker

Yes

No

Counterexample

Insufficient memory

Advantages

General verification approach

Makes formal techniques available to broad audience:
not much training required

Provides diagnostic information

Automatic procedure (“push-button”) taking as input:
a finite state model and a set of required properties

Disavantages

Suitable for control-intensive applications, less for data.

Outcome depends on the quality of the model obtained from the real system - good abstractions are crucial.

Suffers from state-explosion problem - if decidable at all.

Overview

- Transition systems
- Linear-time temporal logic
- Automata-theoretic model checking
- Bounded model checking
- Computation tree logic
- Symbolic model checking
- Equivalences and abstraction
- Partial order reduction
- Communicating automata
- Timed automata
- Probabilistic systems