SPIN
Model Checking Distributed Software Systems

M. Devi Prasad

School of Information Sciences
Manipal University, Manipal

devi.prasad@manipal.edu

August 14, 2016

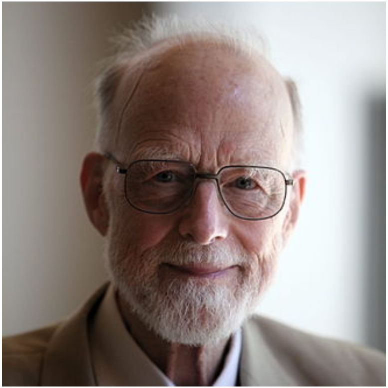
Goals

Goals

- Motivate the audience to formal verification of software
- Introduce model checking as a formal method
- Demonstrate ideas with SPIN
- Advocate the manifesto of the verified software initiative

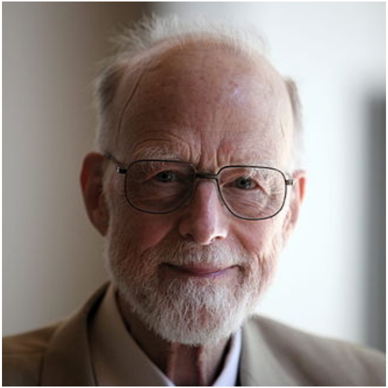
Background

Sir C. A. Richard Hoare



"Education in technology should begin at the undergraduate level. Would you be prepared to teach the use of specifications and assertions as a routine aid to program testing, as they are currently being used in Industry?"

Sir C. A. Richard Hoare



"Education in technology should begin at the undergraduate level. Would you be prepared to teach the use of specifications and assertions as a routine aid to program testing, as they are currently being used in Industry? Would you use them yourself as an aid to marking the students' practical assignments?"

Three Catastrophes

- Loss of lives due to Toyota unintended acceleration defect (2002 - 2014)
- Ethical bankruptcy in Volkswagen emission fraud (2015)
- Linux kernel vulnerability - keyring bug (2016)

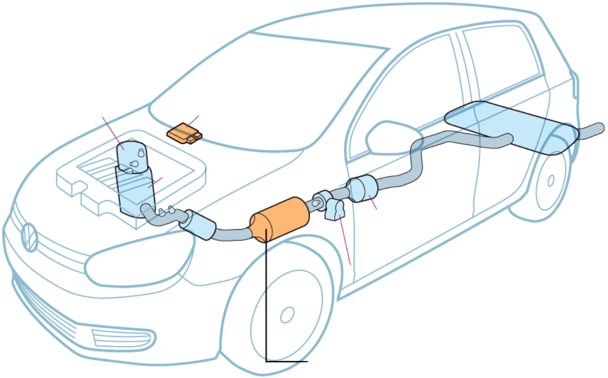
Toyota Unintended Acceleration

A horrifying story of corrupt leadership and pathetic engineering.

Pages 28 to 32 in Bookout_v_Toyota_Barr_REDACTED.

Volkswagen Emission Fraud

Dishonest and fraudulent practices of automotive industries.



Informal Methods - Testing

Edsger Dijkstra

"The first moral of the story is that program testing can be used very effectively to show the presence of bugs but never to show their absence."

Linux Kernel Keyring Bug

An embarrassing bug that confirms Dijkstra's incisive observations.

"The join_session_keyring function in security/keys/process_ keys.c in the Linux kernel before 4.4.1 mishandles object references in a certain error case, which allows local users to gain privileges or cause a denial of service (integer overflow and use-after-free) via crafted keyctl commands."

`http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=2016-0728`

Two Superior Solutions

- NASA - software for Mars rover *Curiosity* (2012)
- Amazon - formal verification of cloud services (2011)

SPIN Model Checker

Mars Code

CACM, Vol. 57 No. 2, Pages 64-73, February 2014

"Informally, Spin takes the role of a demonic process scheduler, trying to find system executions that violate user-defined requirements."

Verification of Amazon AWS

How Amazon Web Services Uses Formal Methods

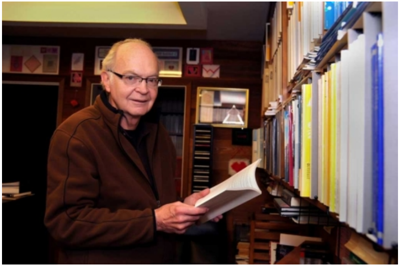
CACM, Vol. 58, No. 4, Pages 66-73. April 2015.

System	Components	Line count (excl. comments)	Benefit
S3	Fault-tolerant low-level network algorithm	804 PlusCal	Found 2 bugs. Found further bugs in proposed optimizations.
	Background redistribution of data	645 PlusCal	Found 1 bug, and found a bug in the first proposed fix.
DynamoDB	Replication & group-membership system	939 TLA+	Found 3 bugs, some requiring traces of 35 steps
EBS	Volume management	102 PlusCal	Found 3 bugs.
Internal distributed lock manager	Lock-free data structure	223 PlusCal	Improved confidence. Failed to find a liveness bug as we did not check liveness.
	Fault tolerant replication and reconfiguration algorithm	318 TLA+	Found 1 bug. Verified an aggressive optimization.

A Sage Thought

A Sage Thought

Donald Knuth



"I was surprised to learn that the writing of programs for TeX and for METAFONT proved to be much more difficult than all the other things I had done (like proving theorems and writing books)."

Donald Knuth

"The creation of good software demands a significantly higher standard of accuracy than those other things do, and it requires a longer attention span than other intellectual tasks."

Model Checking

– Theory and Practice –

Reactive Systems

- A reactive program's role is to maintains an ongoing interaction with its environment rather than to compute a final value and terminate.

Reactive Systems

- A reactive program's role is to maintains an ongoing interaction with its environment rather than to compute a final value and terminate.
- Concurrency is a fundamental element in reactive programs.

Reactive Systems

- A reactive program's role is to maintains an ongoing interaction with its environment rather than to compute a final value and terminate.
- Concurrency is a fundamental element in reactive programs.
- By definition, a reactive program runs concurrently with its environment.

Examples of Reactive Systems

- Web servers and Web browsers
- Embedded systems software
- Operating systems
- Control programs

Properties of Reactive Programs

- Safety properties

  "nothing bad ever happens"

Properties of Reactive Programs

- Safety properties

  "nothing bad ever happens"

- Liveness properties

  "something good will eventually happen"

Properties of Reactive Programs

- Safety properties

  "nothing bad ever happens"

  no deadlocks, no abnormal termination

Properties of Reactive Programs

- Safety properties

  "nothing bad ever happens"

  no deadlocks, no abnormal termination

- Liveness properties

  "something good will eventually happen"

  progress, fairness

Properties of Reactive Programs

- Safety properties

  "nothing bad ever happens"

- Liveness properties

  "something good will eventually happen"

These properties are defined only over infinite execution sequences of reactive programs.

Model Checking

- Define a formal model of the system.

Model Checking

- Define a formal model of the system.
- Formulate a rigorous specification of the properties.

Model Checking

- Define a formal model of the system.
- Formulate a rigorous specification of the properties.
- construct a verification algorithm.

Model Checking

- Define a formal model of the system.
- Formulate a rigorous specification of the properties.
- construct a verification algorithm.
    - detect violation of the properties.
    - produce an execution trace for a counter-example.

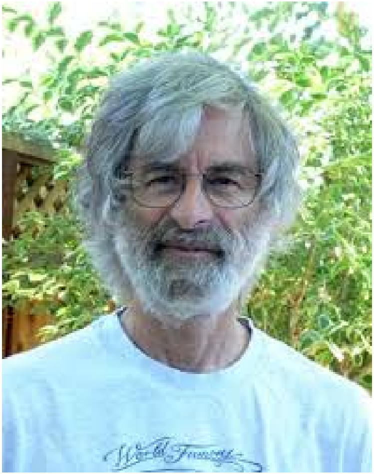
Amir Pnueli

"In mathematics, logic is static . . . When one designs a dynamic computer system that has to react to ever changing conditions, . . . one cannot design the system based on a static view. It is necessary to characterize and describe dynamic behaviors that connect entities, events, and reactions at different time points. Temporal Logic deals therefore with a dynamic view of the world that evolves over time."

Amir Pnueli

"In mathematics, logic is static . . . When one designs a dynamic computer system that has to react to ever changing conditions, . . . one cannot design the system based on a static view. It is necessary to characterize and describe dynamic behaviors that connect entities, events, and reactions at different time points. Temporal Logic deals therefore with a dynamic view of the world that evolves over time."

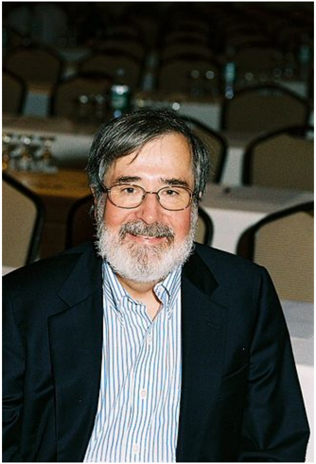
Leslie Lamport



What Good Is Temporal Logic?

"We want to specify not that the program *might* produce the right answer, but that it *must* do so. Because many formalisms cannot express liveness properties, thay have led people to consider such 'possibility' properties instead."

Edmund Clarke



- Engineered world's first Model Checker.
- Mentored the design of world's first symbolic model checker.
- Contributed to the design of highly efficient theorem provers

Joseph Sifakis Allen Emerson

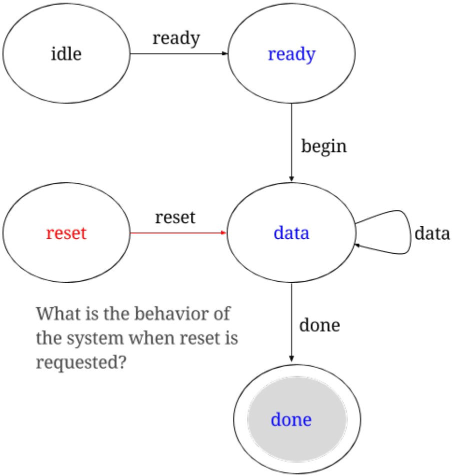
Automata and Logic

Automata and Logic

A finite state automaton is a quintuple $(\mathrm{S}, \mathrm{s}_0, \mathrm{L}, \mathrm{T}, \mathrm{F})$, where

- S is a finite set of *states*
- s_0 is the *initial state*, $s_0 \in S$
- L is a finite set of *labels*
- T is a set of *transitions*, $\mathrm{T} \subseteq (\mathrm{S} \times \mathrm{L} \times \mathrm{S})$, and
- F is a set of *final* states, $\mathrm{F} \subseteq \mathrm{S}$

Finite State Automaton - An Example



What is the behavior of the system when reset is requested?

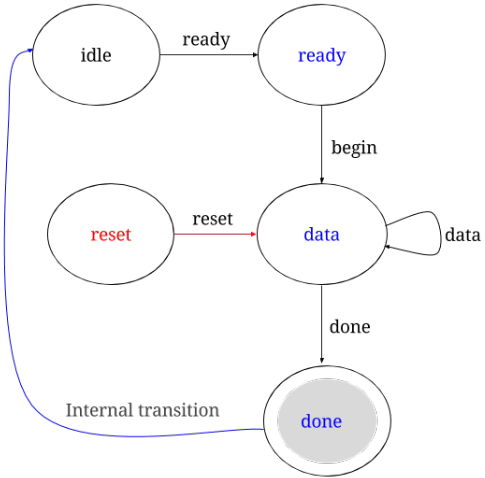
Omega Run

- A finite state machine models terminating executions
- A finite state machine cannot decide on acceptance or non-acceptance of ongoing, potentially infinite executions.

Omega Run

- A finite state machine models terminating executions
- A finite state machine cannot decide on acceptance or non-acceptance of ongoing, potentially infinite executions.
- An infinite run is called an ω-run or *omega run*

Omega Runs in an Automaton



Omega Acceptance

- if σ is an infinite run, σ^{ω} represents the set of states that appear infinitely often within σ's transitions

Omega Acceptance

- if σ is an infinite run, σ^ω represents the set of states that appear infinitely often within σ's transitions
- σ^+ represents the set of states that appear only finitely many times.

Omega Acceptance

- An accepting ω-run of finite state automaton $(\mathrm{S}, \mathrm{s}_0, \mathrm{L}, \mathrm{T}, \mathrm{F})$ is any infinite run σ such that $\exists \mathrm{s}_\mathrm{f}.\, \mathrm{s}_\mathrm{f} \in \mathrm{F} \wedge \mathrm{s}_\mathrm{f} \in \sigma^\omega$

Omega Acceptance

- An accepting ω-run of finite state automaton $(\mathrm{S}, \mathrm{s}_0, \mathrm{L}, \mathrm{T}, \mathrm{F})$ is any infinite run σ such that $\exists \mathrm{s}_\mathrm{f}.\, \mathrm{s}_\mathrm{f} \in \mathrm{F} \wedge \mathrm{s}_\mathrm{f} \in \sigma^\omega$
- An infinite run is accepted if and only if some state in F is visited infinitely often in the run.

Omega Acceptance

- An accepting ω-run of finite state automaton $(\mathrm{S}, \mathrm{s}_0, \mathrm{L}, \mathrm{T}, \mathrm{F})$ is any infinite run σ such that $\exists \mathrm{s}_\mathrm{f}.\, \mathrm{s}_\mathrm{f} \in \mathrm{F} \wedge \mathrm{s}_\mathrm{f} \in \sigma^\omega$
- An infinite run is accepted if and only if some state in F is visited infinitely often in the run.
- A Büchi automaton satisfies the ω-acceptance conditions.

Temporal Logic

- Automata provides the specification of the system behavior

Temporal Logic

- Automata provides the specification of the system behavior
- A verification model should contain a formulation of the *correctness requirements* of the system.

Temporal Logic

- Automata provides the specification of the system behavior
- A verification model should contain a formulation of the *correctness requirements* of the system.
- We need to be able to verify properties of *reactive* programs.

Temporal Logic

- Automata provides the specification of the system behavior
- A verification model should contain a formulation of the *correctness requirements* of the system.
- We need to be able to verify properties of *reactive* programs.
    - *ready* is invariantly true.

Temporal Logic

- Automata provides the specification of the system behavior
- A verification model should contain a formulation of the *correctness requirements* of the system.
- We need to be able to verify properties of *reactive* programs.
  - *ready* is invariantly true.
  - *send* always implies an eventual *ack* response.

Temporal Logic

- Automata provides the specification of the system behavior
- A verification model should contain a formulation of the *correctness requirements* of the system.
- We need to be able to verify properties of *reactive* programs.
    - *ready* is invariantly true.
    - *send* always implies an eventual *ack* response.
    - *reset* always eventually becomes *false* at least once more.

Frequently Used Linear Temporal Logic Formulae

Table: LTL Formulae

Formula	Read As	Template
$\Box$ p	always p	*invariance*
$\Diamond$ p	eventually p	*guarantee*
p $\rightarrow$ $\Diamond$ q	p implies eventually q	*response*
p $\rightarrow$ q U r	p implies q until r	*precedence*
$\Box\Diamond$ p	always eventually p	*progress*
$\Diamond\Box$ p	eventually always p	*stability*
$\Diamond$ p$\rightarrow$ $\Diamond$ q	eventually p implies eventually q	*correlation*

Specification of a Few Properties

Table: LTL Specification

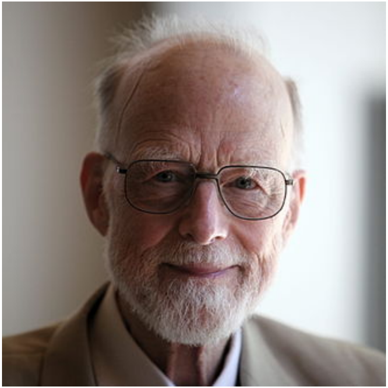
Formula	Description
$\square$ p	p is invariantly *true*
$\lozenge\square$!p	p eventually becomes invariantly false
$\square\lozenge$!p	p always eventually becomes false at least once
$\square$ (q $\rightarrow$!p)	q always implies !p
p $\rightarrow$ $\lozenge$ q	p always implies eventually q

Verified Software Initiative

Verified Software Initiative Manifesto

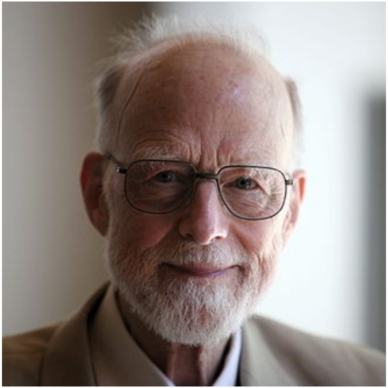
The Verified Software Initiative is a long-term research program directed at the challenge of verifying software to the highest levels of correctness.

Sir C. A. Richard Hoare



"The teams of experimental scientists will require education in the relevant theories, and training in the use of the best available tools. Would you be prepared to design and deliver Master's courses on program verification? Would it be appropriate to set Master's projects to verify small portions of the challenge material held in the repository?"

Sir C. A. Richard Hoare



"The teams of experimental scientists will require education in the relevant theories, and training in the use of the best available tools. Would you be prepared to design and deliver Master's courses on program verification? Would it be appropriate to set Master's projects to verify small portions of the challenge material held in the repository?"

Conclusion

Summary

- SPIN is a valuable model checking tool.
- It is time bring formal methods into our curriculum
- It is time we practice formal verification!

Questions?