Formal Methods – Just Enough

Yogananda Jeppu

$$\exists a\,[I(a) \wedge \neg R(a)]$$

$$\forall b\,[D(b) \Rightarrow \neg R(b)]$$

Copyright Notice

- Formal Methods Just Enough by Yogananda Jeppu is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License.

- You are free:
- to Share — to copy, distribute and transmit the work
- to Remix — to adapt the work
- Under the following conditions:
- Attribution —You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
- Noncommercial —You may not use this work for commercial purposes.
- Share Alike — If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.
- For details please visit the website.

Background

- I am Yogananda Jeppu. I have a PhD in safety critical control system testing. I have 29 years experience in control system design, 6DOF simulation, Model Based Verification and Validation, System Testing.
- I have worked on the Indian Light Combat Aircraft (LCA) control system and the Indian SARAS aircraft. I have worked on model based commercial aircraft flight control law programs of Boeing, Airbus, Gulfstream and Comac.
- Currently I am working at Honeywell Technology Solutions , on Formal Methods, and Model Based System Engineering.

This is very true

Teaching to unsuspecting youngsters the effective use of formal methods is one of the joys of life because it is so extremely rewarding.

- Edsger Dijkstra

Errors in software

Patriot veered off course

… omission of a hyphen in coded computer instructions

Mull of Kintyre (Video)

29 DEAD!!

Why something different?

US aviation authority: Boeing 787 bug could cause 'loss of control'

More trouble for Dreamliner as Federal Aviation Administration warns glitch in control unit causes generators to shut down if left powered on for 248 days

The Boeing 787 has four generator-control units that, if powered on at the same, could fail simultaneou a complete electrical shutdown. Photograph: Elaine Thompson/AP

SUMMARY: We are adopting a new airworthiness directive (AD) for all The Boeing Company Model 787 airplanes. This AD requires a repetitive maintenance task for electrical power deactivation on Model 787 airplanes. This AD was prompted by the determination that a Model 787 airplane that has been powered continuously for 248 days can lose all alternating current (AC) electrical power due to the generator control units (GCUs) simultaneously going into failsafe mode. This condition is caused by a software counter internal to the GCUs that will overflow after 248 days of continuous power. We are issuing this AD to prevent loss of all AC electrical power, which could result in loss of control of the airplane.

Video

Airbus too ...

Glitch found in engine software requires immediate checks after issue-plagued fleet is grounded

The Airbus A400M has been plagued by technical faults and now software glitches that reportedly caused a crash. Photograph: Julio Munoz/EPA

Airbus has issued a critical alert calling for immediate checks on all its A400M aircraft after a report identified a software bug as having caused a fatal crash in Spain earlier this month.

In a statement Airbus said it was "devastated to confirm" the loss of four crew members, adding that another two are in hospital in a serious condition.

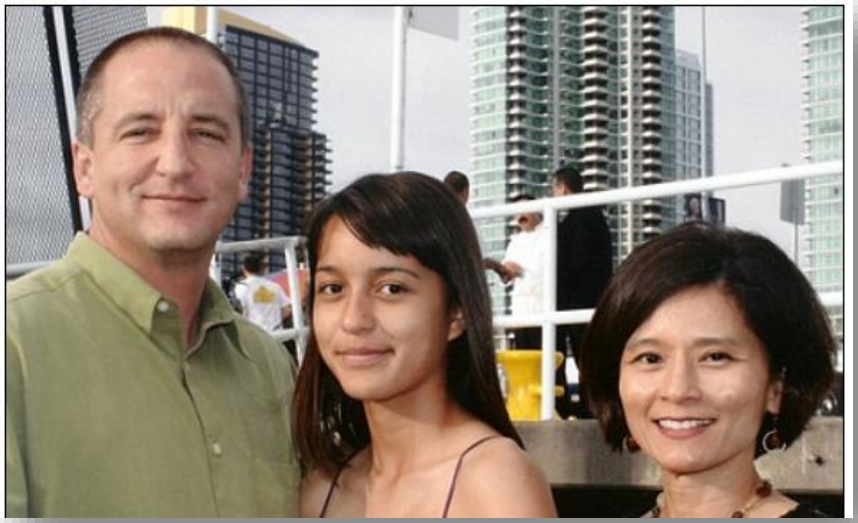
The plane crashed in a rural area near Seville airport (Pic: Airlive.net)

Four crew dead!!

Automotive domain

'There's no brakes... hold on and pray': Last words of man before he and his family died in Toyota Lexus crash

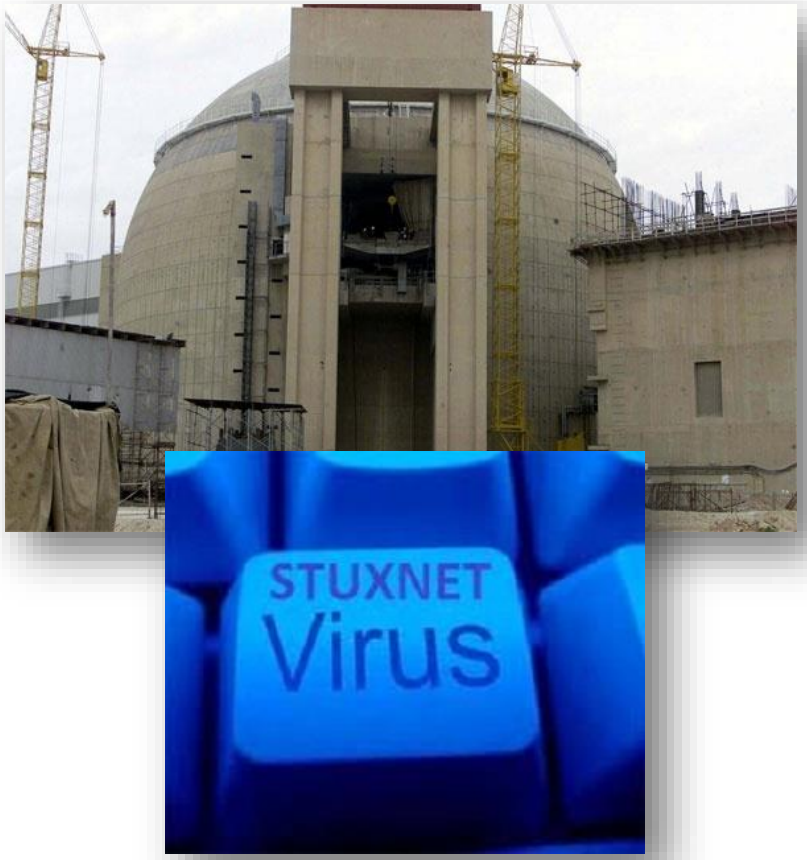
By MAIL FOREIGN SERVICE
UPDATED: 15:07 EST, 3 February 2010



- http://www.dailymail.co.uk/news/article-1248177/Toyota-recall-Last-words-father-family-died-Lexus-crash.html

Nuclear

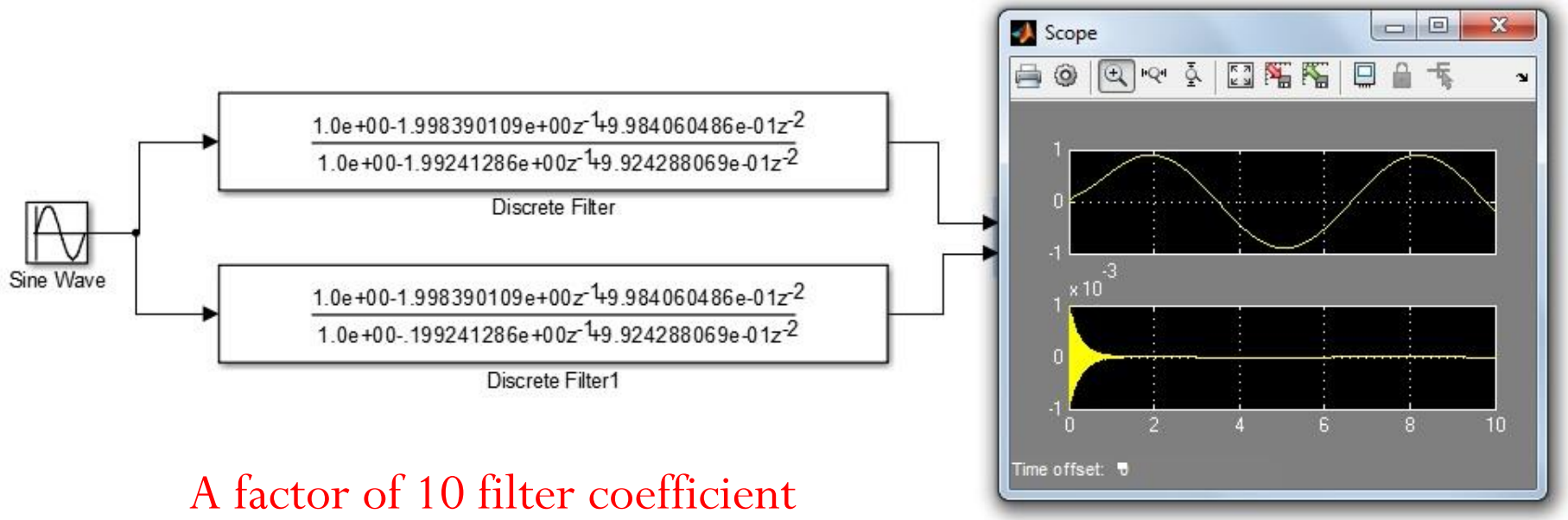
- Iran's first nuclear power plant has suffered a serious cyber-intrusion from a sophisticated worm that infected workers' computers, and potentially plant systems. Virus designed to target only Siemens supervisory control and data acquisition (SCADA) systems that are configured to control and monitor specific industrial processes (Wiki) - September 27, 2010



Space

- The Accident Investigation Board concluded the root cause of the Titan IV B-32 mission mishap was due to the failure of the software development, testing, and quality/mission assurance process used to detect and correct a human error in the manual entry of a constant. The entire mission failed because of this, and the cost was about $1.23 billion.

Titan IV B-32 filter problem



A factor of 10 filter coefficient error made the output to zero

2016

Radar glitch requires F-35 fighter jet pilots to turn it off and on again

Troubled warplane that has yet to see any cyber security testing hit with yet another bug affecting flight performance requiring software update

https://www.theguardian.com/technology/2016/mar/08/radar-glitch-requires-f-35-fighter-jet-pilots-to-turn-it-off-and-on-again

Defense One

NEWS THREATS POLITICS MANAGEMENT TECH

AA FONT SIZE + PRINT AP / RICK BOWMER

The F-35's Terrifying Bug List

FEBRUARY 2, 2016 BY PATRICK TUCKER

The Pentagon's top testing official has weighed and measured the F-35 and found it wanting. Technology

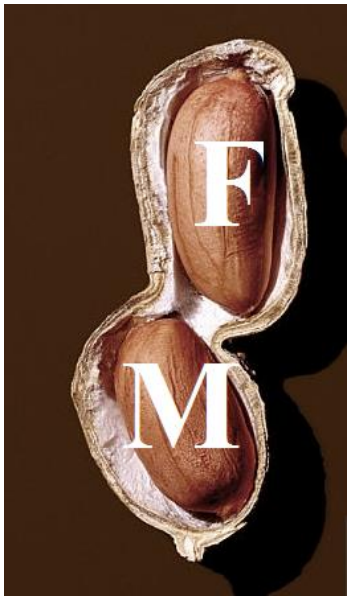
The F-35 Joint Strike Fighter programs the **most expensive military program** in the world, is even more broken than previously thought. The jet can't tell old parts from new ones, randomly prevents user logins into the logistics information

http://www.defenseone.com/technology/2016/02/f-35s-terrifying-bug-list/125638/

We require something different!

Formal Methods – In a nutshell

- Formal methods are techniques used to model complex systems as mathematical entities. (They are more precise)
- The complex system behavior is broken down into smaller units and each one of these is defined as a mathematical equations. This is the TRUTH.
- Defining systems formally allows the engineer to validate the system (mathematically correct behavior - mostly safety criteria) using other means than testing – like a proof of correctness



Why formal methods

- Program testing can be used to show the presence of bugs, but never to show their absence!
- If you want more effective programmers, you will discover that they should not waste their time debugging, they should not introduce the bugs to start with.
- The software engineer's task is to produce several models or descriptions of a system for an abstract machine, with accompanying proofs that models at lower levels of abstraction correctly implement higher-level models. Only this design process can ensure high levels of quality, not testing.

Edsger W. Dykstra

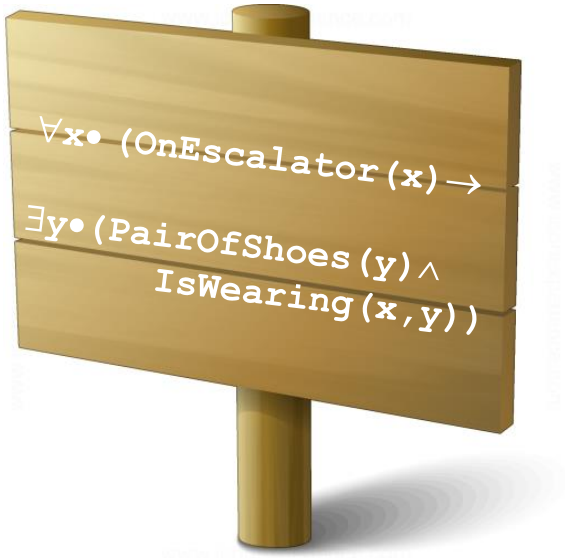
Why formal methods

- There is a lack of clarity when we use natural language
- It is sometimes difficult to use language in a precise and unambiguous way without making the document wordy and difficult to read (I prefer to use a block diagram)
- Several different requirements may be expressed together as a single requirement (This happens all the time even with guidelines)
- You can say the same thing in completely different ways. (We found that Indian and US read the same requirement in different ways and test them)

English is Ambiguous

$$\forall x \bullet (OnEscalator(x) \rightarrow \exists y \bullet (PairOfShoes(y) \wedge IsWearing(x,y)))$$

$$\forall x \bullet ((OnEscalator(x) \wedge IsDog(x)) \rightarrow IsCarried(x))$$



BUT thank God English signs are better – though ambiguous

History of Formal Methods

Panini should be thought of as the forerunner of the modern formal language theory used to specify computer languages. The Backus Normal Form was discovered independently by John Backus in 1959, but Panini's notation is equivalent in its power to that of Backus and has many similar properties.

http://www-groups.dcs.st-and.ac.uk/history/Biographies/Panini.html

History of Formal Methods

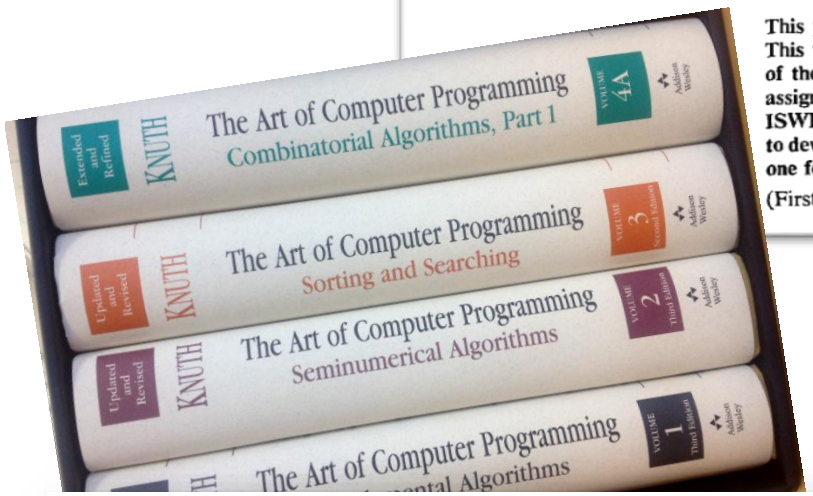
41

Proving properties of programs by structural induction

By R. M. Burstall*

This paper discusses the technique of structural induction for proving theorems about programs. This technique is closely related to recursion induction but makes use of the inductive definition of the data structures handled by the programs. It treats programs with recursion but without assignments or jumps. Some syntactic extensions to Landin's functional programming language ISWIM are suggested which make it easier to program the manipulation of data structures and to develop proofs about such programs. Two sample proofs are given to demonstrate the technique, one for a tree sorting algorithm and one for a simple compiler for expressions.

(First received April 1968 and in revised form August 1968)



J. McCarthy. *Towards a mathematical science of computation. Information processing 1962, Proceedings of IFIP Congress 62, organized by the International Federation for Information Processing, Munich, 27 August–1 September 1962,* edited by Cicely M. Popplewell, North-Holland Publishing Company, Amsterdam 1963, pp. 21–28.

History of Formal Methods

- 1969

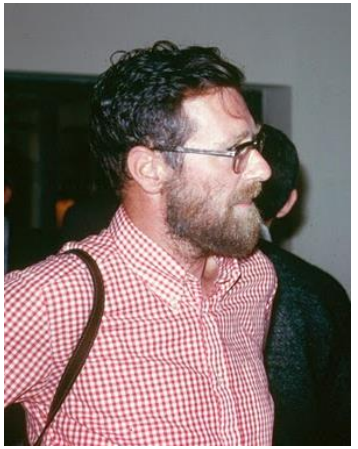
An Axiomatic Basis for Computer Programming

C. A. R. Hoare
The Queen's University of Belfast, Northern Ireland*

- 1975

Guarded Commands, Nondeterminacy and Formal Derivation of Programs

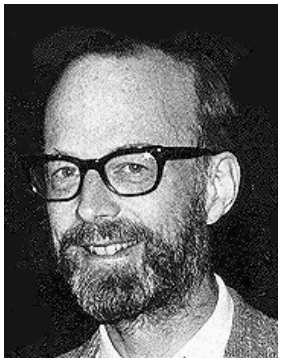
Edsger W. Dijkstra
Burroughs Corporation



History of Formal Methods

- 1940's: Turing annotated the properties of program states to simplify the logical analysis of sequential programs.

- 1960's: Floyd, Hoare and Naur recommended using axiomatic techniques to prove programs meet their specifications.



- 1970's:Dijkstra used formal calculus to aid to develop of non-deterministic programs.

Why have we taken so long …

- perhaps caused by the expectation of mathematics …
- perhaps caused by apprehension of the unknown

There is much fear of formal methods! I have gone through the phase ….

http://quoteaddicts.com/topic/fear-cartoon/

Mathematical roots

Discrete Mathematics

Mathematical Logic

Algorithms

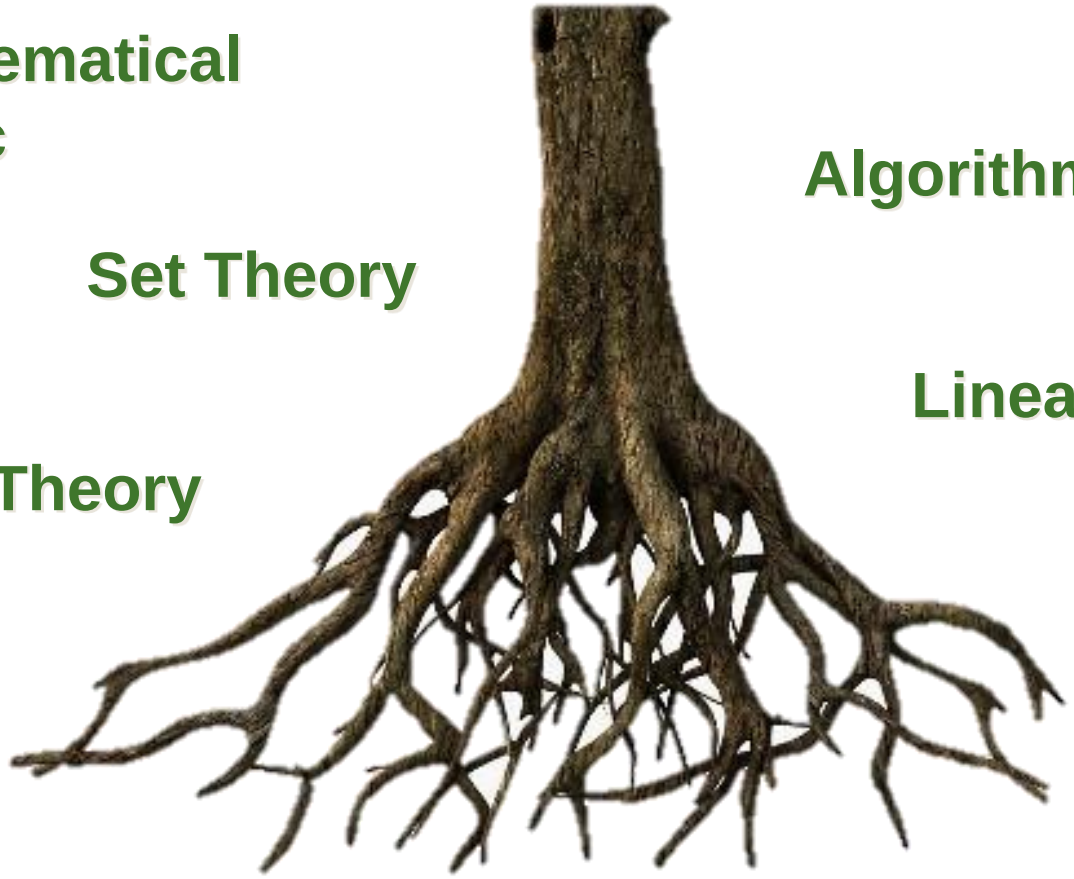
Set Theory

Linear Algebra

Graph Theory

Proof Theory

Number Theory



$\exists \quad \forall \quad \supset \quad \neg \quad \Rightarrow \quad \Leftrightarrow$

Standard & Definitions

NASA/CR–2014-218244

Formal Methods Case Studies for DO-333

Darren Cofer and Steven P. Miller
Rockwell Collins, Inc., Cedar Rapids, Iowa

Definitions

- Formal Methods (FM.B.1.3 DO 333)

  Formal methods are mathematically based techniques for the specification, development, and verification of software aspects of digital systems. The mathematical basis of formal methods consists of formal logic, discrete mathematics, and computer-readable languages. The use of formal methods is motivated by the expectation that, as in other engineering disciplines, performing appropriate mathematical analyses can contribute to establishing the correctness and robustness of a design.

- Formal Models (FM.B.1.3.1 DO 333)

  Establishing a formal model of the software artifact of interest is fundamental to all formal methods. In general a model is an abstract representation of a given set of aspects of the software that is used for analysis, simulation, and/or code generation. In the context of this supplement, to be formal, a model should have an unambiguous, mathematically defined syntax and semantics. This makes it possible to use automated means to obtain guarantees that the model has certain specified properties.

What are the benefits?

- DO 333 clearly defines that the following are the usefulness of formal methods

  - Unambiguously describing requirements of software systems.
  - Enabling precise communication between engineers.
  - Providing verification evidence such as consistency and accuracy of a formally specified representation of software.
  - Providing verification evidence of the compliance of one formally specified representation with another.

What can it find?

- DO 333 indicates the following errors can be found

  - Freedom from exceptions.
  - Freedom from deadlock.
  - Non-interference between different levels of criticality.
  - Worst case execution time.
  - Bounds on stack size during execution.
  - Freedom from unintended function.
  - Correct synchronous or asynchronous behavior.

Older myths of formal methods

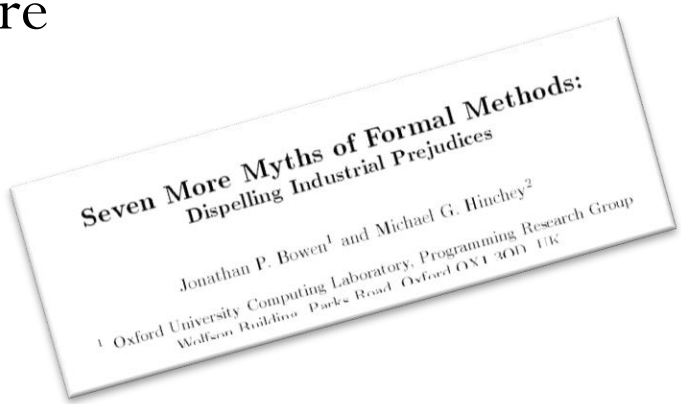
- Formal methods can guarantee that software is perfect.
- Work by proving that programs are correct.
- Only highly critical systems benefit from their use.
- They involve complex math.
- They increase the cost of development.
- They are incomprehensible to clients.
- Nobody uses them for real projects.

A. Hall. “Seven myths of formal methods". In: Software, IEEE 7.5 (Sept. 1990), pp. 11

http://languageartsgames.4you4free.com/Fmyths.gif

Newer myths of formal methods

- Formal Methods delay the development process
- Formal Methods are not supported by tools
- Formal Methods mean forsaking traditional engineering design methods
- Formal Methods only apply to software
- Formal Methods are not required
- Formal Methods are not supported
- Formal Methods people always use Formal Methods

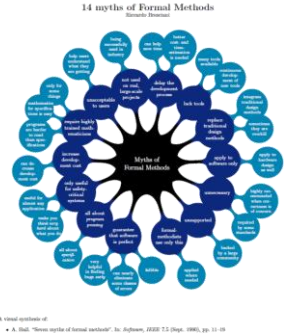

Seven More Myths of Formal Methods:
Dispelling Industrial Prejudices

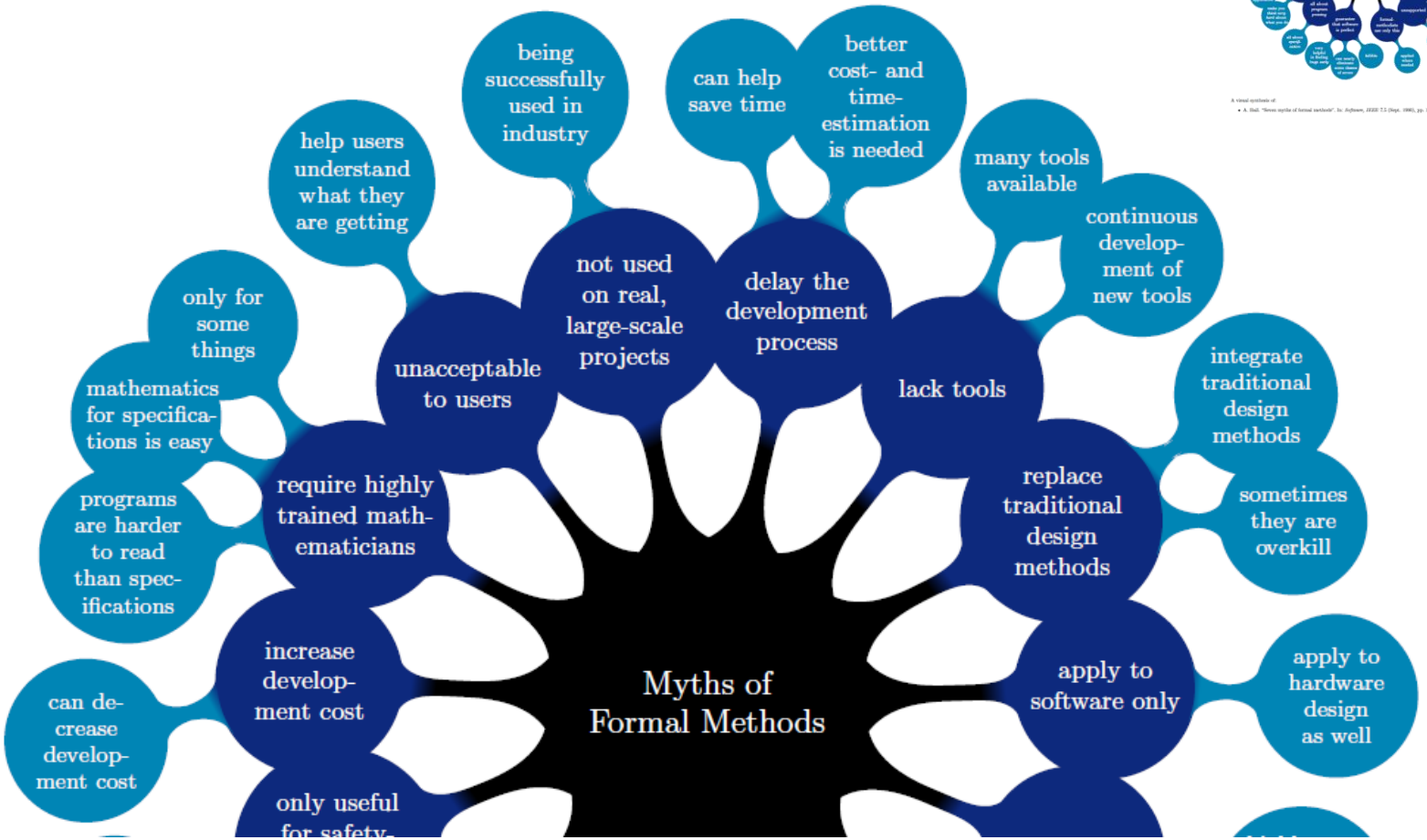
Jonathan P. Bowen[1] and Michael G. Hinchey[2]

[1] Oxford University Computing Laboratory, Programming Research Group
Wolfson Building, Parks Road, Oxford OX1 3QD, UK

J.P. Bowen and M.G. Hinchey. "Seven more myths of formal methods". In: Software, IEEE 12.4 (1995), pp. 34

14 myths of FM

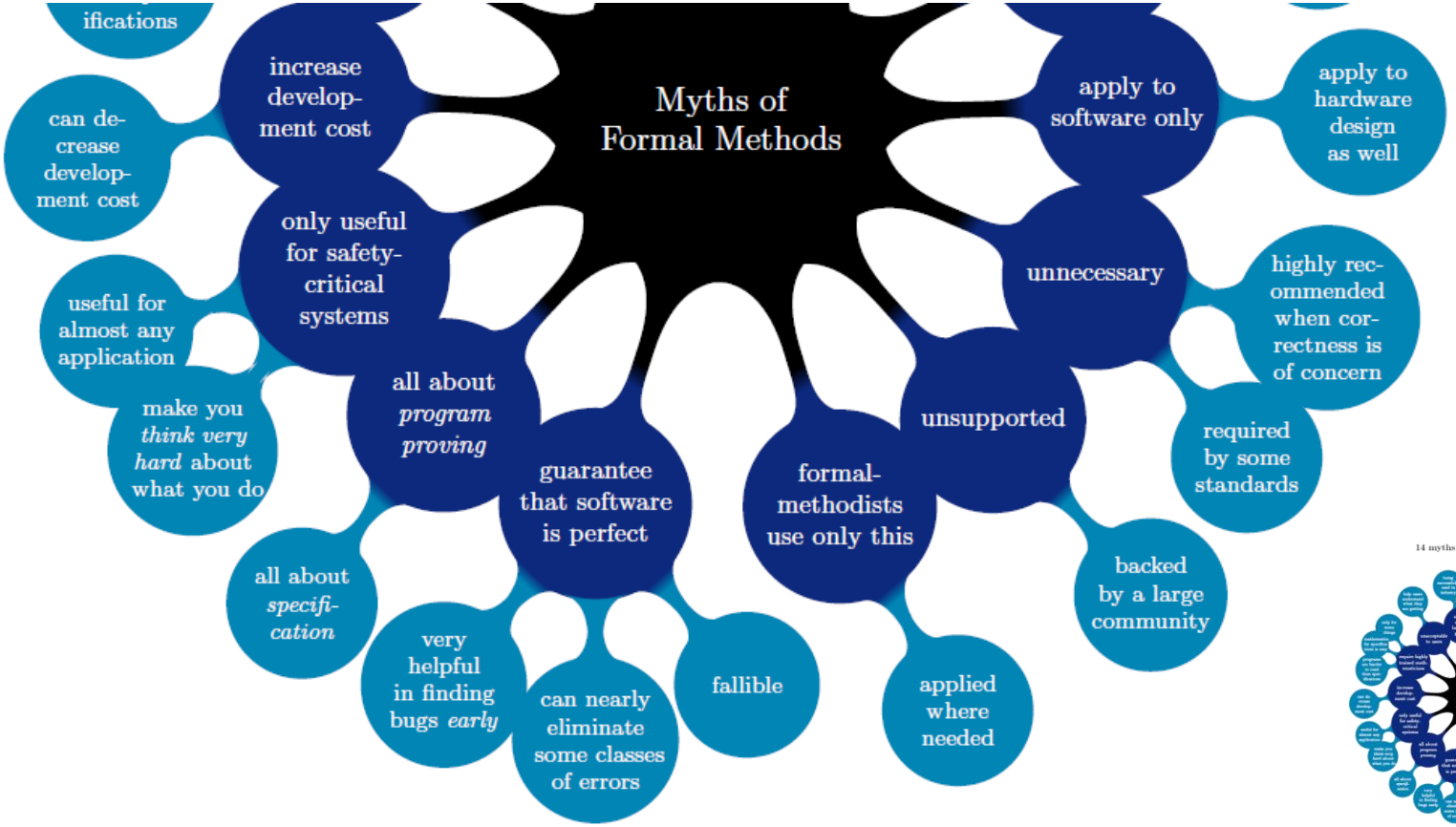


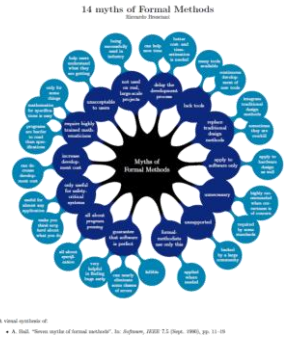


14 myths of Formal Methods

Riccardo Bresciani

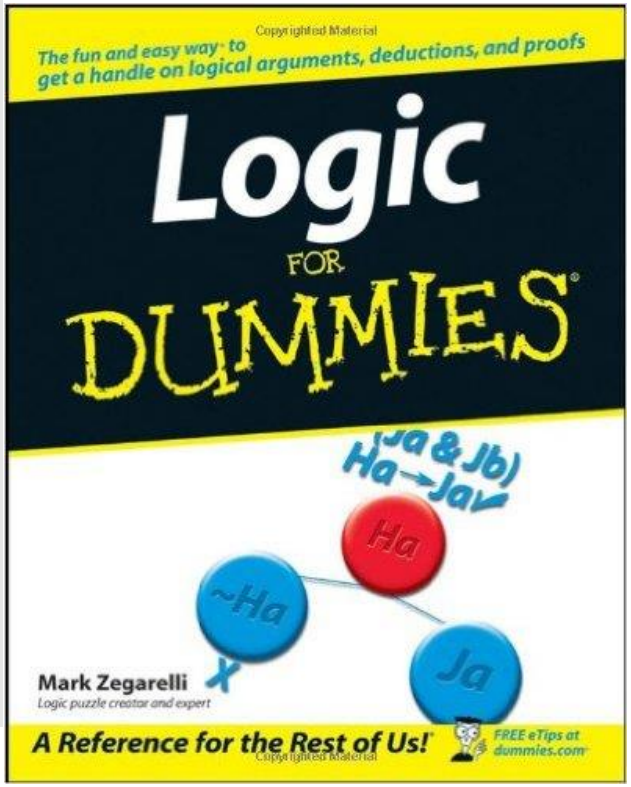
14 myths of FM





Please refer to the original for clarity
https://www.scss.tcd.ie/disciplines/software_systems/fmg/fmg_web/alumni/Riccardo.Bresciani/14mythsFM.pdf

Logic



Predicate logic

- The first order predicate calculus is a formal language for expressing propositions. (https://en.wikipedia.org/wiki/First-order_logic)
- This is a good read if you have patience. See the example at the end
- Predicate logic is the way to define TRUTH about anything. This is used for the properties of the system
- Predicate logic consists of A set of terms, A set of predicates, The connectives of propositional logic $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$, Quantifiers $\forall$, $\exists$ and lots of brackets

- Disclaimer: This is all the MATHS you are going to get from me so enjoy

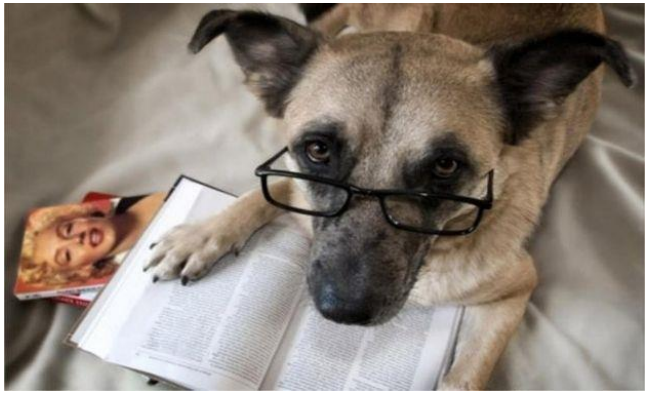
Predicate logic symbols

- A set of terms
  - a, b, c, d, … John, Mary, Pavarotti, Loren .. x, y, z, x0, x1…
- A set of predicates
  - One-place predicates: is happy, is boring H(a)
  - Two-place predicates: like, hate, love, hit L(a,b)
  - Three-place predicates: introduce, give I(a,b,c)
- Connectives of propositional logic: ¬ (NOT),∧ (AND),∨ (OR),→ (IMPLIES),↔ (EQUIVALENCE)
- Quantifiers: ∀ (for all), ∃ (there exists)

Example – a dog's day

Dogs are not literate,
$\forall x [D(x) \rightarrow \neg L(x)]$
for all x, Dog(x) implies not Literate(x)

Whoever can read is literate.
$\forall x [R(x) \rightarrow L(x)]$
for all x, Read(x) implies that Literate(x)



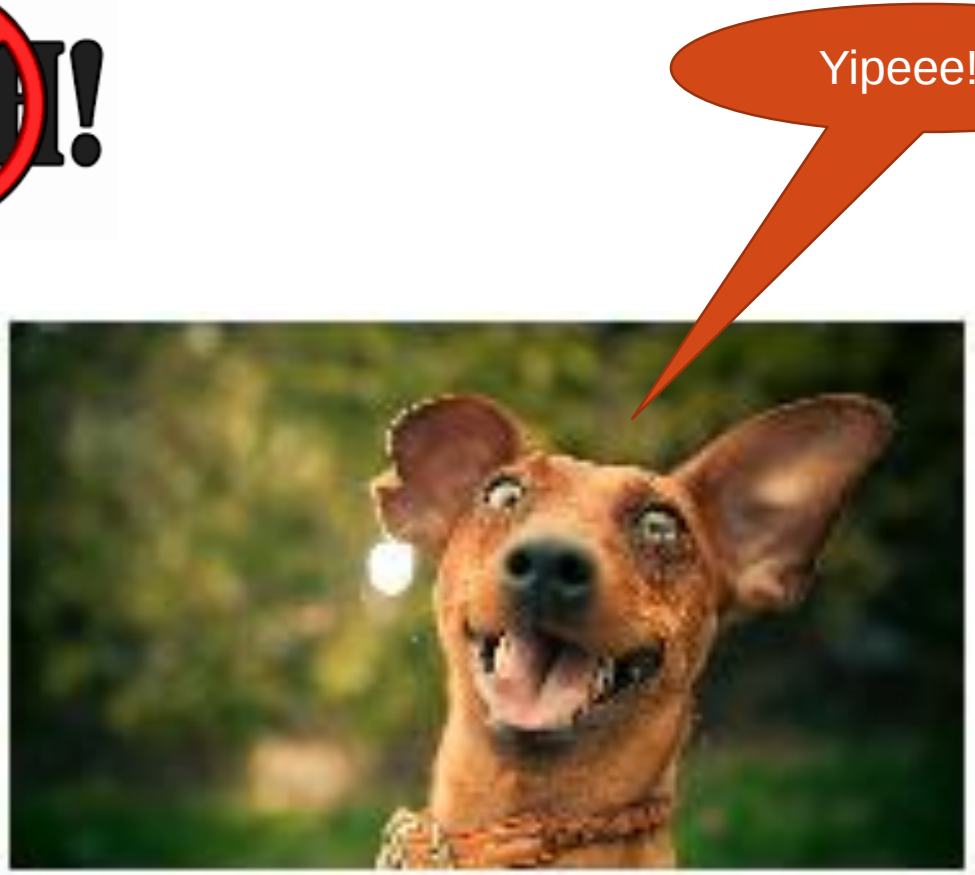
http://www.telegraph.co.uk/pets/quiz-how-intelligent-is-your-dog/
https://www.pinterest.com/pin/484981453598332655/

A dog's day continued

Some who are intelligent cannot read
$\exists$ x [I(x) $\wedge$ $\neg$ R(x)]
there exists x such that [Intelligent(x) AND NOT Read(x)] is TRUE

Some dogs are intelligent
$\exists$ x [D(x) $\wedge$ I(x)]
there exists x such that [Dog(x) AND Intelligent(x)] is TRUE

Some dogs don’t like Maths



Formal Methods - Levels

- Descriptive methods or **Formal Specification**
  - use mathematics-based languages that provide precise, unambiguous descriptions of requirements and other development artifacts
- **Formal Verification**
- Deduction (Theorem Proving)
  - rely on a discipline that requires the explicit enumeration of all assumptions and reasoning steps

Formal Methods - Levels

- Model Checking
  - the process of automatically checking whether a given finite model of an artifact satisfies a given property.
- Abstract Interpretation
  - analysis based on a formal abstract model of all program computations in all possible execution environments

Formal Specifications

- Syntax and semantics rigorously defined
- Precise form, perhaps mathematical
- Eliminate imprecision and ambiguity
- Provide basis for mathematically verifying
- Backus-Naur notation (more commonly known as BNF or Backus-Naur Form) is a formal mathematical way to describe a language, which was developed by John Backus (and possibly Peter Naur as well) to describe the syntax of the Algol 60 programming language.

Formal Specification

- History-based specification
  - behavior based system histories
  - assertions are interpreted over time
- State-based Specification
  - behavior based on system states, series of sequential steps, (e.g. a financial transaction)
- Transition-based specification
  - behavior based on transitions from state-to-state of the system
  - best used with a reactive system
- Functional and Operation Specification

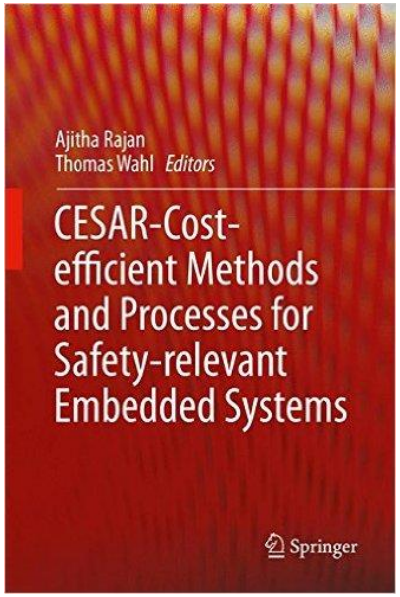
https://en.wikipedia.org/wiki/Formal_specification

References

- Axel van Lamsweerde. 2000. Formal specification: a roadmap. In *Proceedings of the Conference on The Future of Software Engineering* (ICSE '00). ACM, New York, NY, USA, 147-159. DOI=http://dx.doi.org/10.1145/336512.336546-
  http://homepage.divms.uiowa.edu/~tinelli/classes/181/Spring03/Readings/vLam00.pdf

- https://ifs.host.cs.st-andrews.ac.uk/Books/SE9/WebChapters/PDF/Ch_27_Formal_spec.pdf

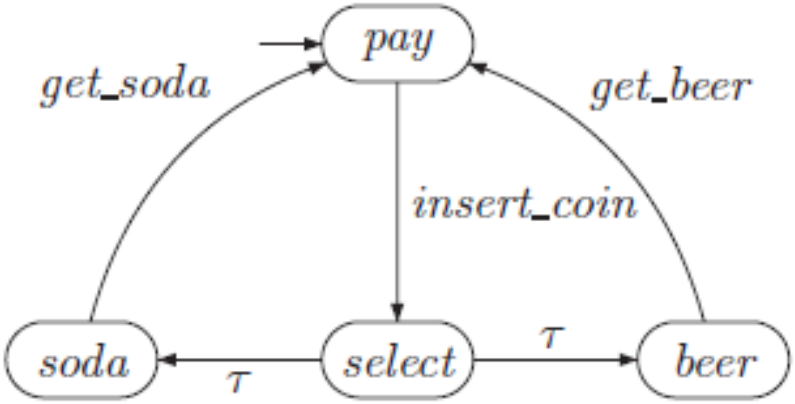
CESAR project did a lot of work on formal requirements .



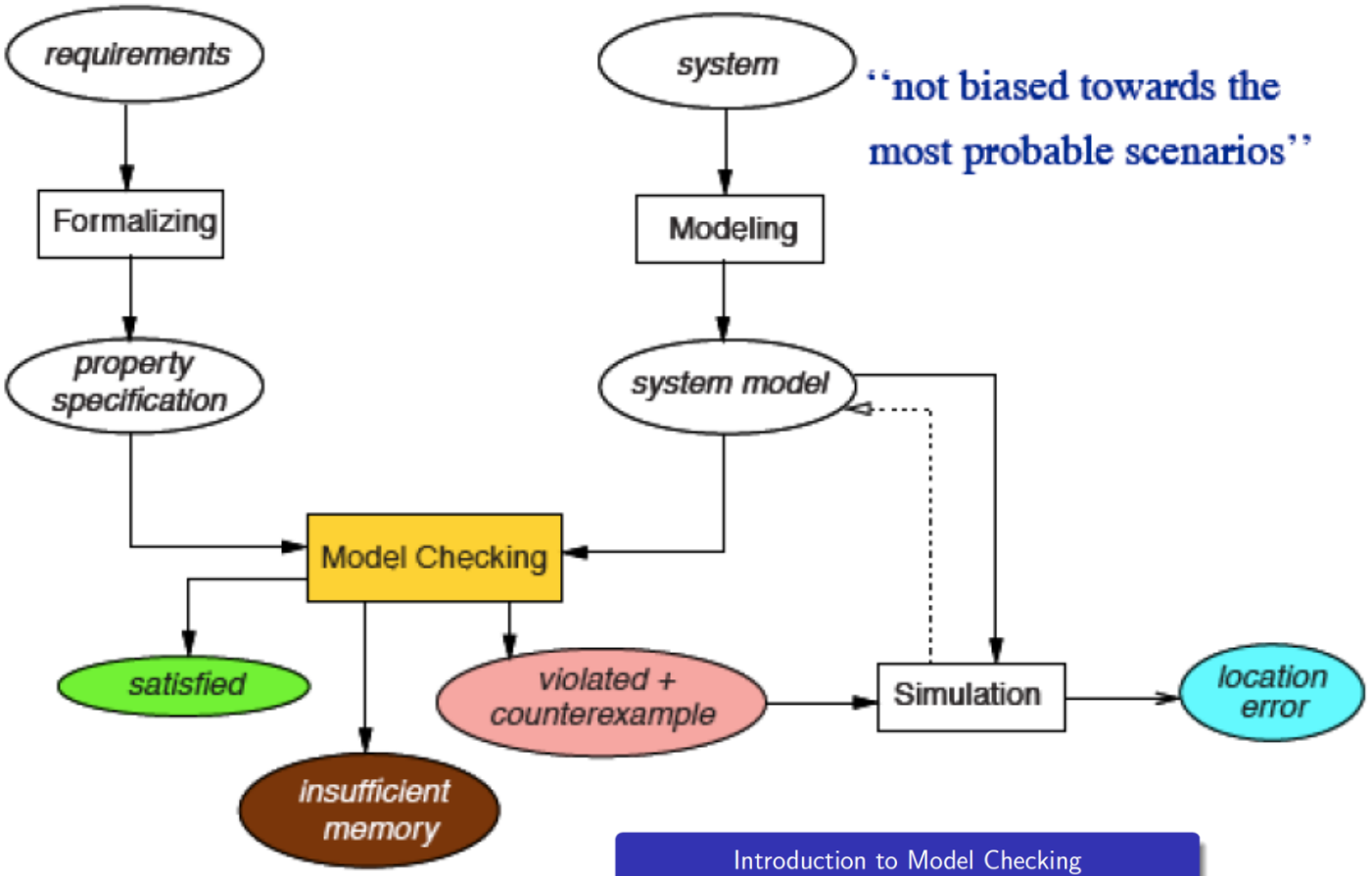
Disclaimer: I am not advising you to buy these books

Model Based Specifications

- Model-based specification is an approach to formal specification where the system specification is expressed as a system state model

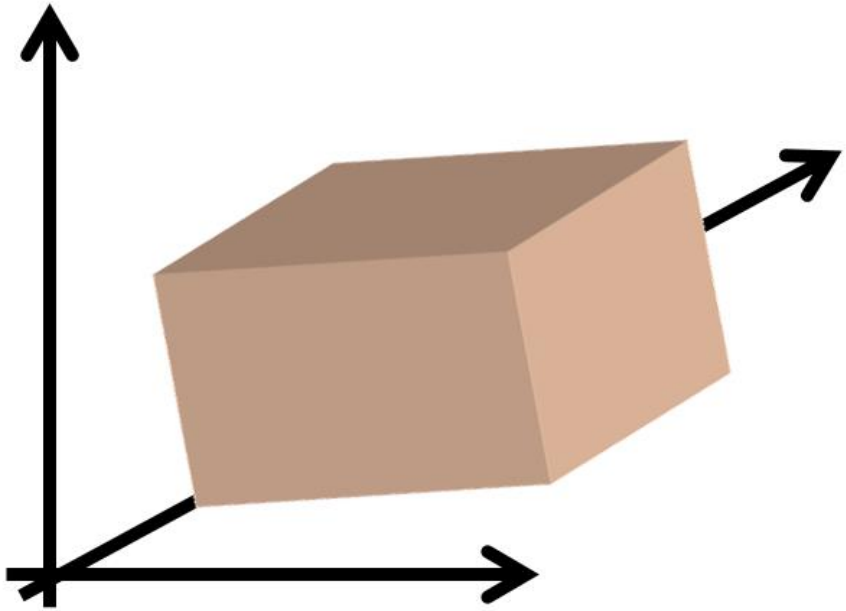


Model Checking

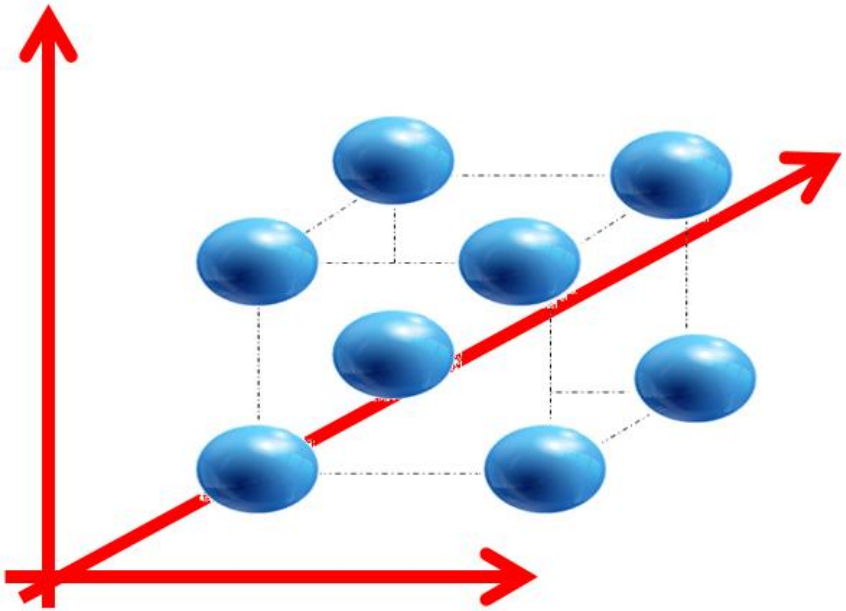


Formal Verification vs Testing

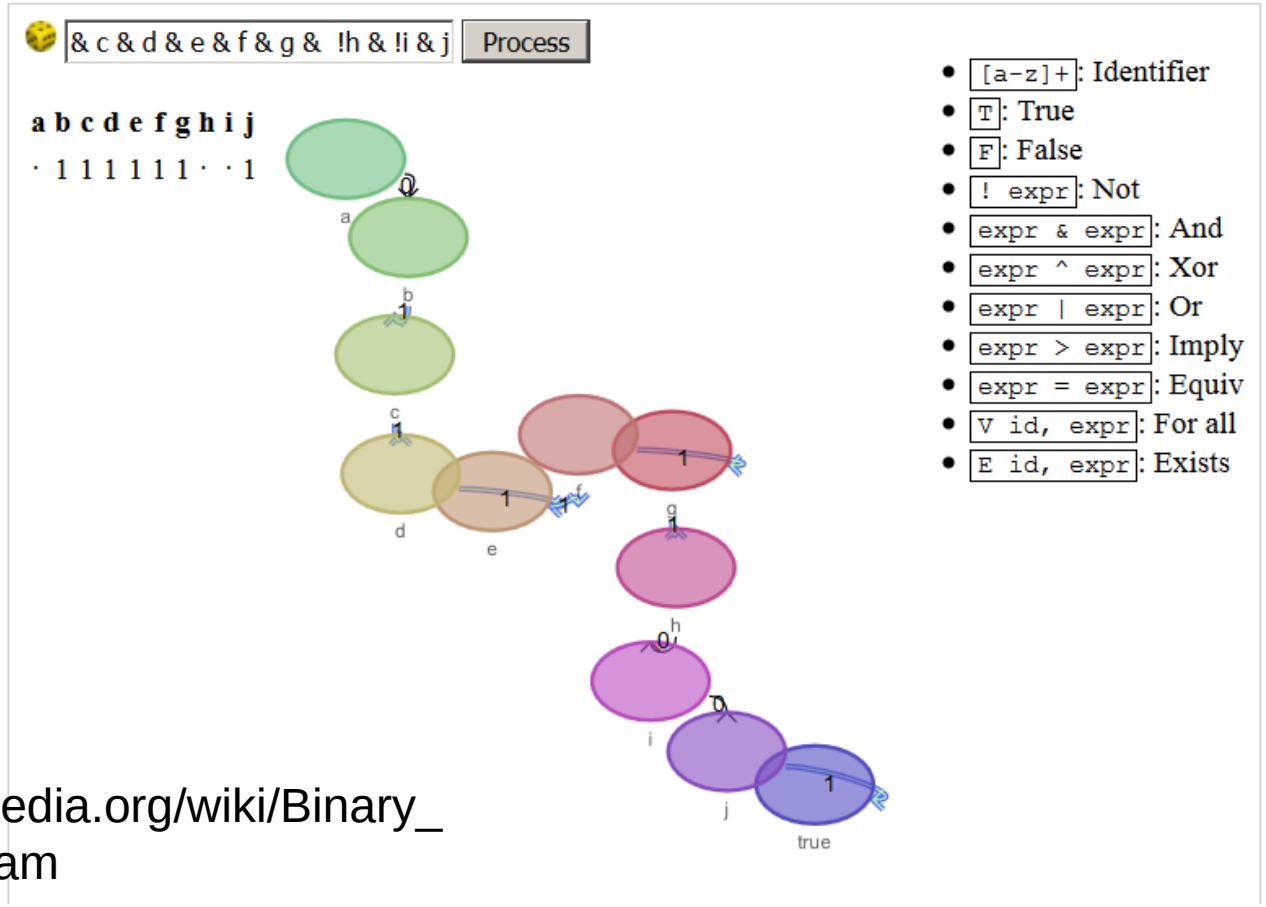
Formal Methods Proof attempts to cover all possible execution scenarios of the system



Testing will look at very specific points as designed by the test engineer



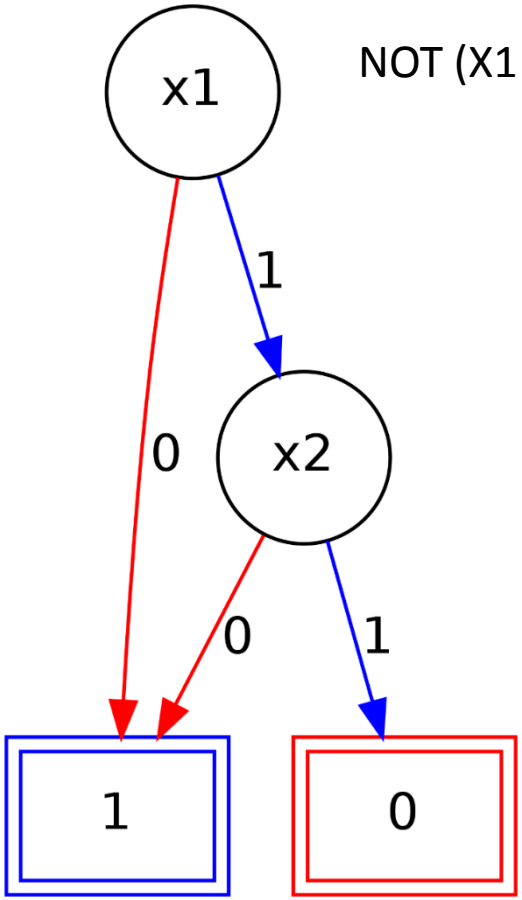
Binary Decision Diagrams



https://en.wikipedia.org/wiki/Binary_decision_diagram

http://blog.vjeux.com/2011/project/javascript-binary-decision-diagram.html

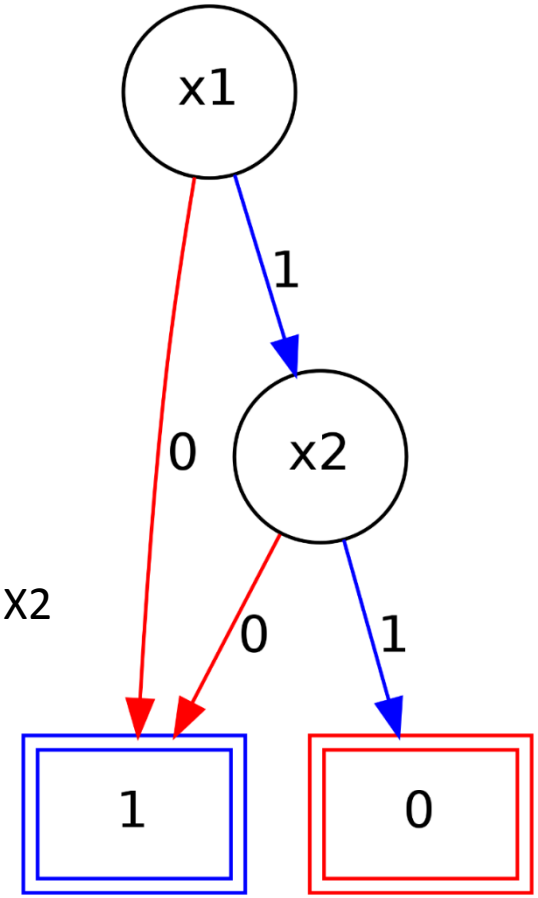
Binary Decision Diagrams



NOT (X1 AND X2)

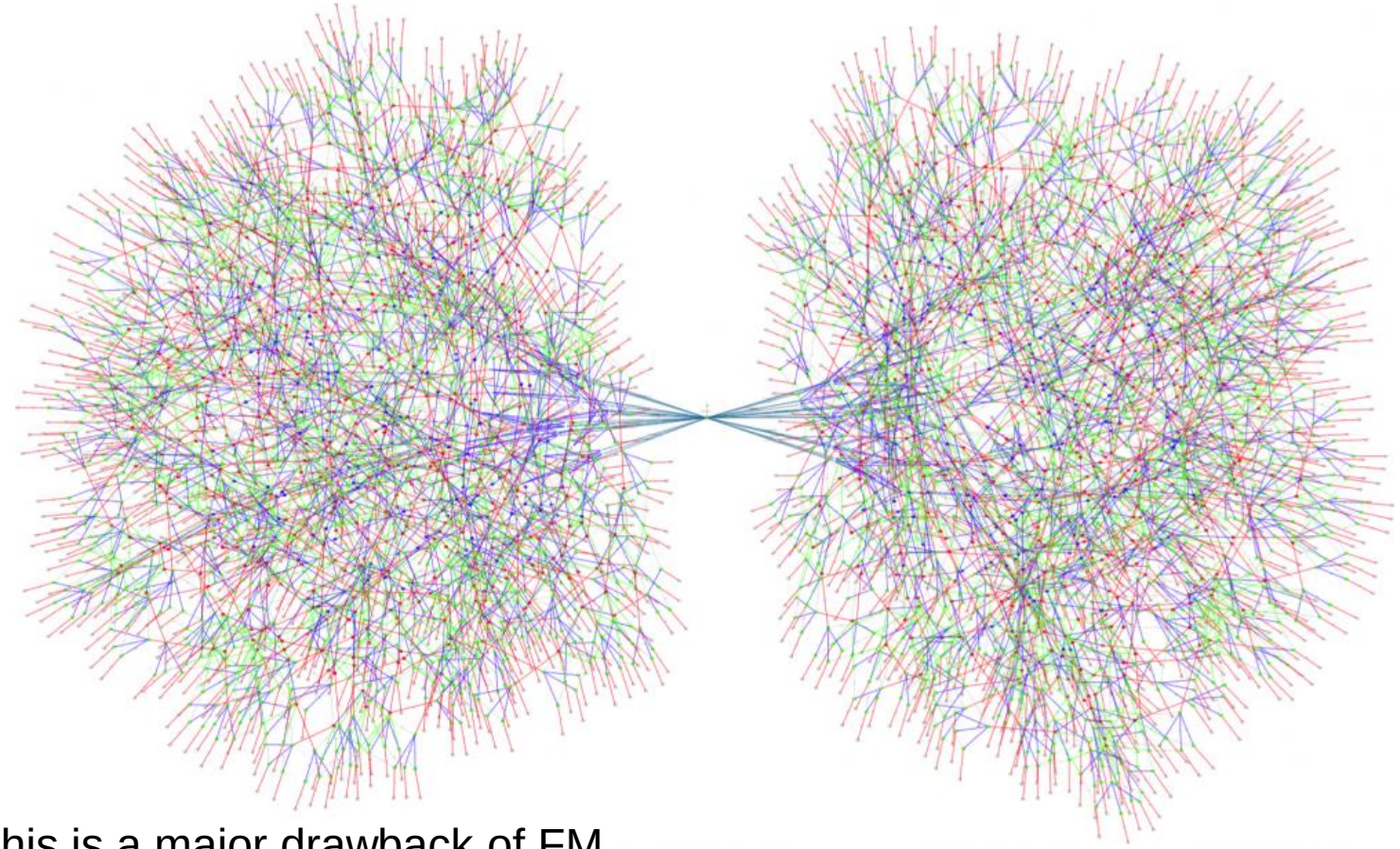
If Design and Assertion BDD match then design and assertions are the same!

NOT X1 OR NOT X2



http://formal.cs.utah.edu:8080/pbl/BDD.php

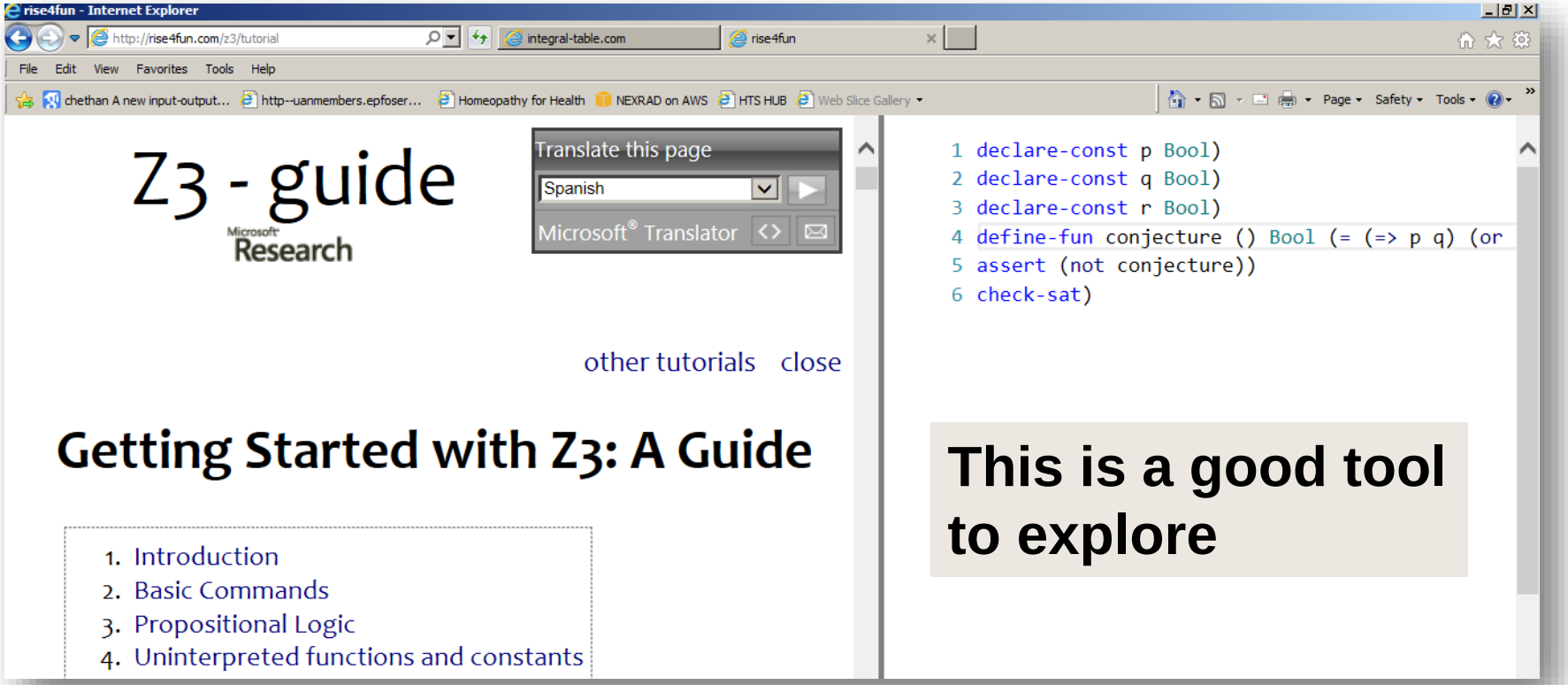
State bloat up



This is a major drawback of FM

https://www3.hhu.de/stups/prob/index.php/State_space_visualization_examples

Theorem proving



http://rise4fun.com/z3/tutorial

https://github.com/Z3Prover/z3

An example

- If **I have money** I will go to the movies

- P is I have money
- Q is I will go to the movies

- If I have money I will go to the movies is P => Q

- What is ~ (P => Q) ?

An example

P	Q	P => Q	~(P => Q)	P	~Q	P and ~Q
T	T	T	F	T	F	F
T	F	F	T	T	T	T
F	T	T	F	F	F	F
F	F	T	F	F	T	F

What is ~ (P => Q) ?

P and ~Q

I have money AND I (will NOT) go to the movies

http://www.professorserna.com/

An example with Z3

$$\sim (p \implies q) \iff (p \wedge \sim q)$$

(define-fun conjecture () Bool (= (not (=> p q)) (and p (not q))))

$$(p \implies q) \iff (\sim p \vee q)$$

(define-fun conjecture () Bool (= (=> p q) (or (not p) q)))

You can copy paste this in the Z3 website and check

An example with Z3

$$p \vee (q \wedge r) \iff (p \vee q) \wedge (p \vee r)$$

(define-fun conjecture () Bool (= (and (or p q) (or p r)) (or p (and q r))))

$$p \wedge (q \vee r) \iff (p \wedge q) \vee (p \wedge r)$$

(define-fun conjecture () Bool (= (or (and p q) (and p r)) (and p (or q r))))

You can copy paste this in the Z3 website and check

CVC4

http://cvc4.cs.nyu.edu/web/

http://church.cims.nyu.edu/wiki/About_CVC4

An example with CVC4

$$p \vee (q \wedge r) \iff (p \vee q) \wedge (p \vee r)$$

```
CVC4>
CVC4> p, q, r : BOOLEAN;
CVC4> QUERY (p OR (q AND r)) = ((p OR q) AND (p OR r));
valid
CVC4> QUERY (p OR (q AND r)) = ((p OR q) AND (p AND r));
invalid
CVC4> COUNTERMODEL;
p : BOOLEAN = FALSE;
q : BOOLEAN = TRUE;
r : BOOLEAN = TRUE;
CVC4>
CVC4>
```

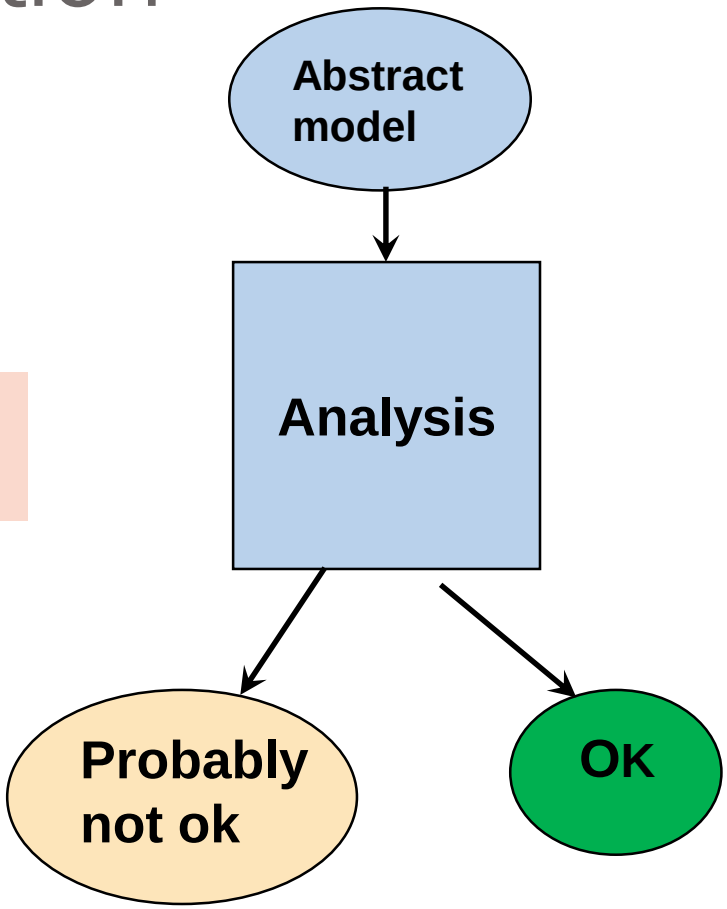
A counter example is also provided when there is an error or invalid proof.

Abstract interpretation

Example of static analysis (output)

```
{n0 = ?}
n := n0;
{n0=n, n0 = ?}
i := n;
{n0=i,n0=n,n0 = ?}
while (i > 0 ) do
{n0=n, i>=1, n0 = ?}
j := 0;
{n0=n, j=0,i>=1, n0>=i}
while (j <> i) do
{n0=n, j>=0,i>=j+1,n0>=i}
j := j + 1
{n0=n, j>=1, i>=j,n0>=i}
od;
{n0=n, i=j,i>=1,n0>=i}
i := i - 1
{i+1=j, n0=n,i>=0,n0>=i+1}
od
{n0=n,i=[0, ?],n0 = ?}
```

? – means any value

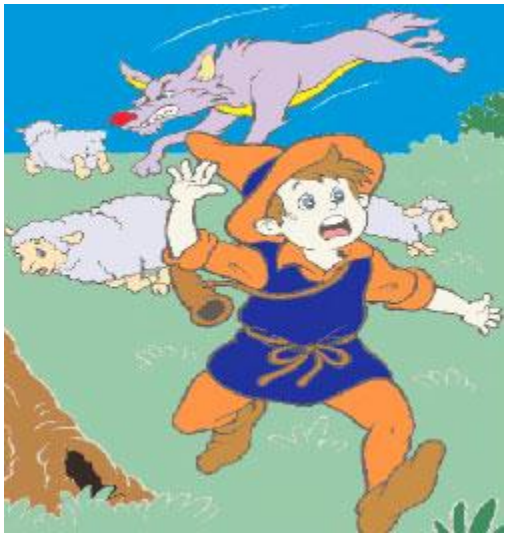


Need to determine if error or false alarm

The code is not executed but analyzed

False Positives

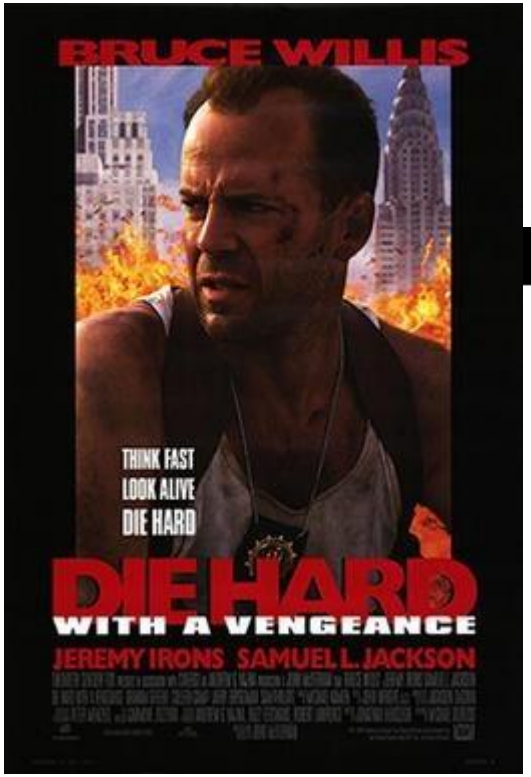
False positives: operations that are safe in reality but which cannot be decided safe or unsafe from the properties inferred by static analysis. **"Wolf-Wolf syndrome"**



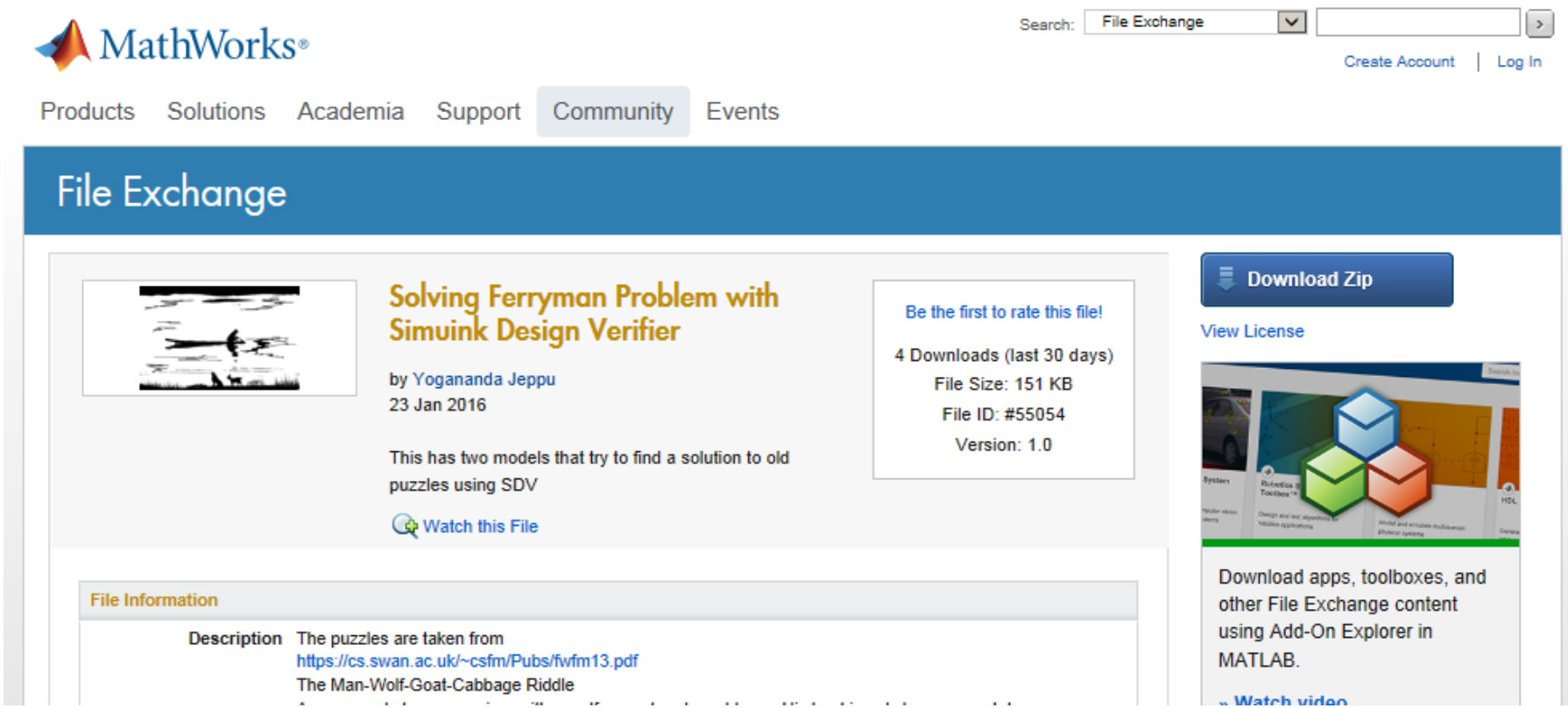
Abstract Interpretation can call out for the wolf sometimes. Analyze with care.

False Negative - A common example is a guilty prisoner freed from jail. This is dangerous!

Games with SLDV

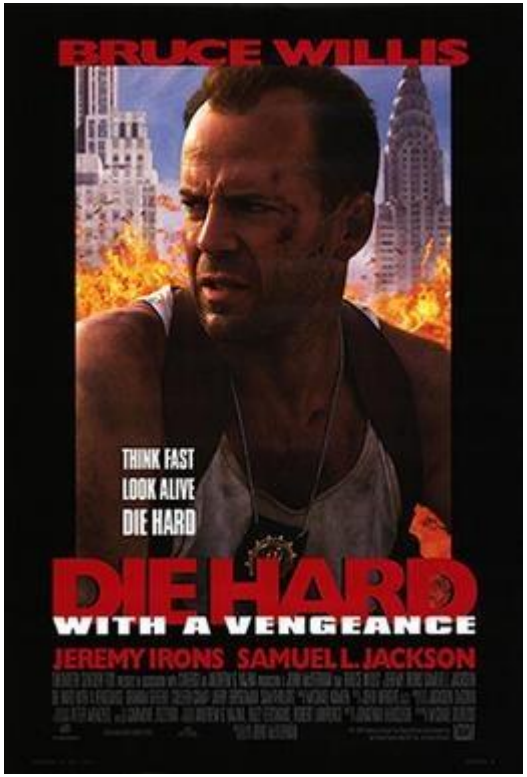


Die Hard



http://in.mathworks.com/matlabcentral/fileexchange/55054-solving-ferryman-problem-with-simuink-design-verifier

Die Hard

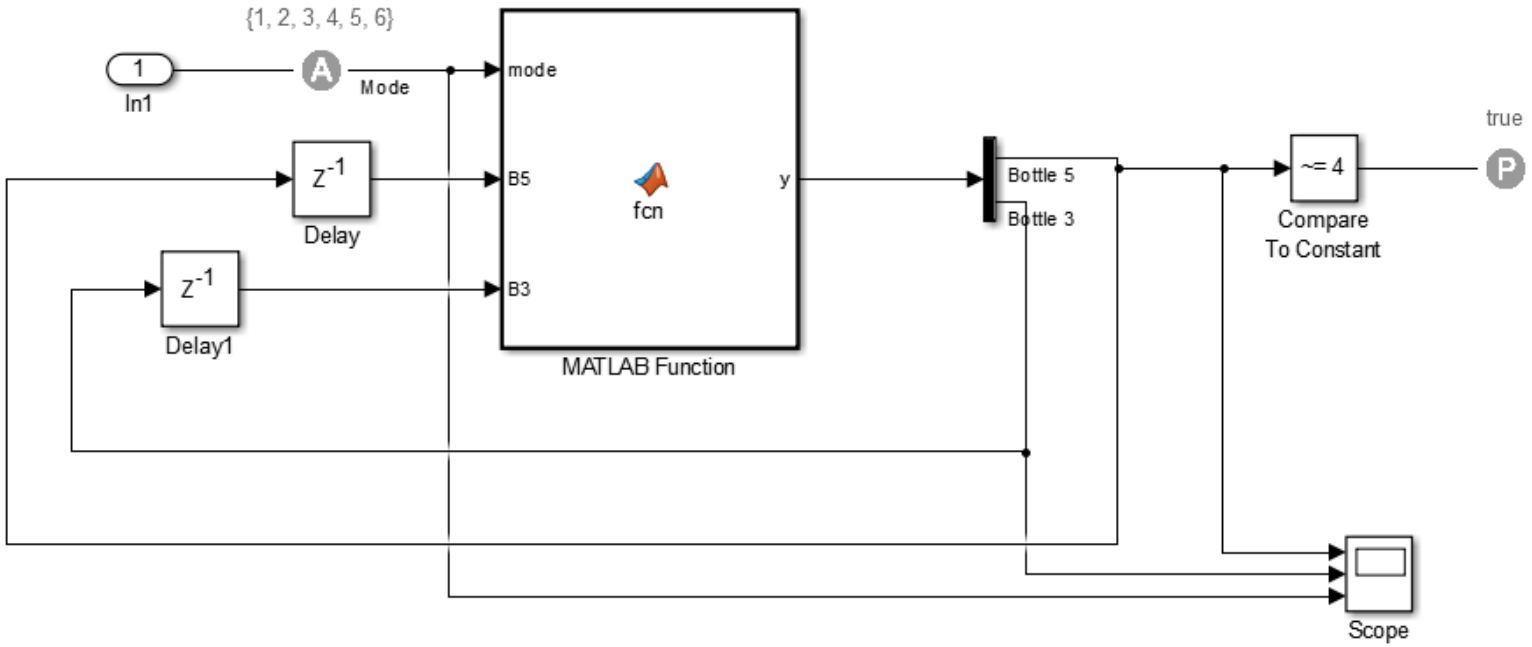


- You are give two cans a 5 lt and a 3 lt. You have to take water from the fountain and have exactly 4 lt of water so that the bomb does not go off

- How about solving it using SDV?

SLDV model

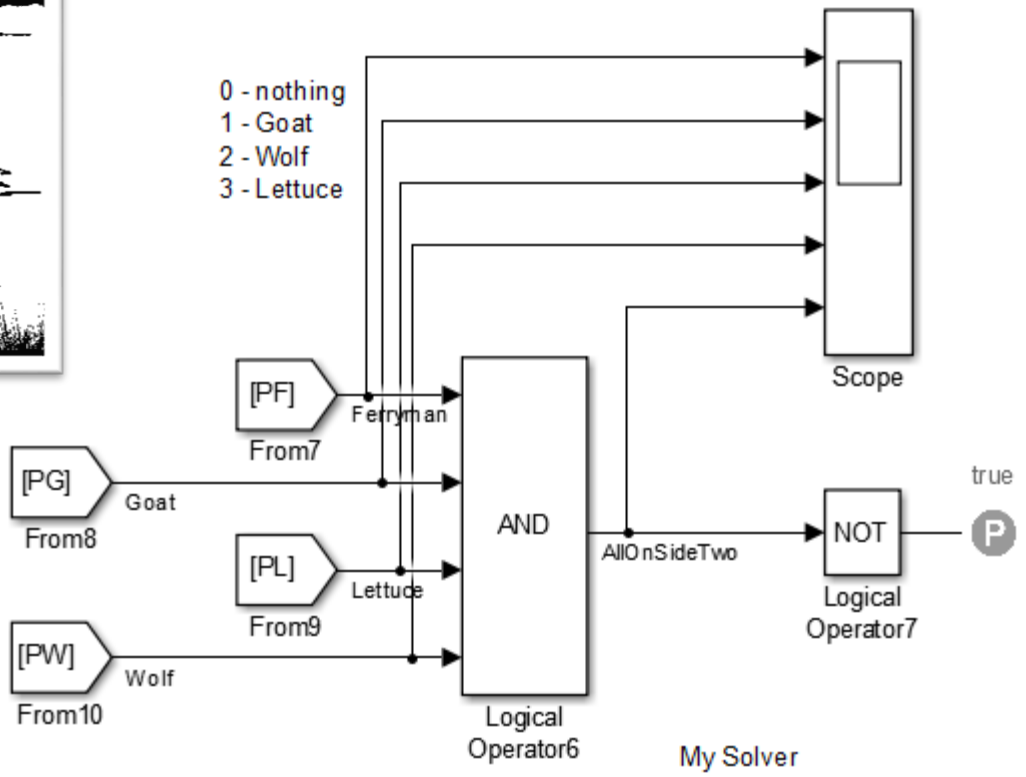
```
if mode == 1  % Fill 5 litre bottle
    B5 = 5;
elseif mode == 2  % Fill 3 litre bottle
    B3 = 3;
elseif mode == 3 % Empty 5 lt bottle
    B5 = 0;
elseif mode == 4 % Empty 3 lt bottle
    B3 = 0;
elseif (mode == 5) && (B5 > 0) && (B3 < 3) % Transfer B5 to B3  if i>0 and j < 3
    % max(0, i+j 3),min(3, i+j
    temp = max(0, B5+B3-3);
    B3 = min(3, B5+B3);
    B5 = temp;
elseif (mode == 6) && (B5 < 5) && (B3 > 0)    % Transfer B3 to B5 if i<5 and j>0
    % min(5, i+j),max(0, i+j 5)
    temp = min(5, B5+B3);
```



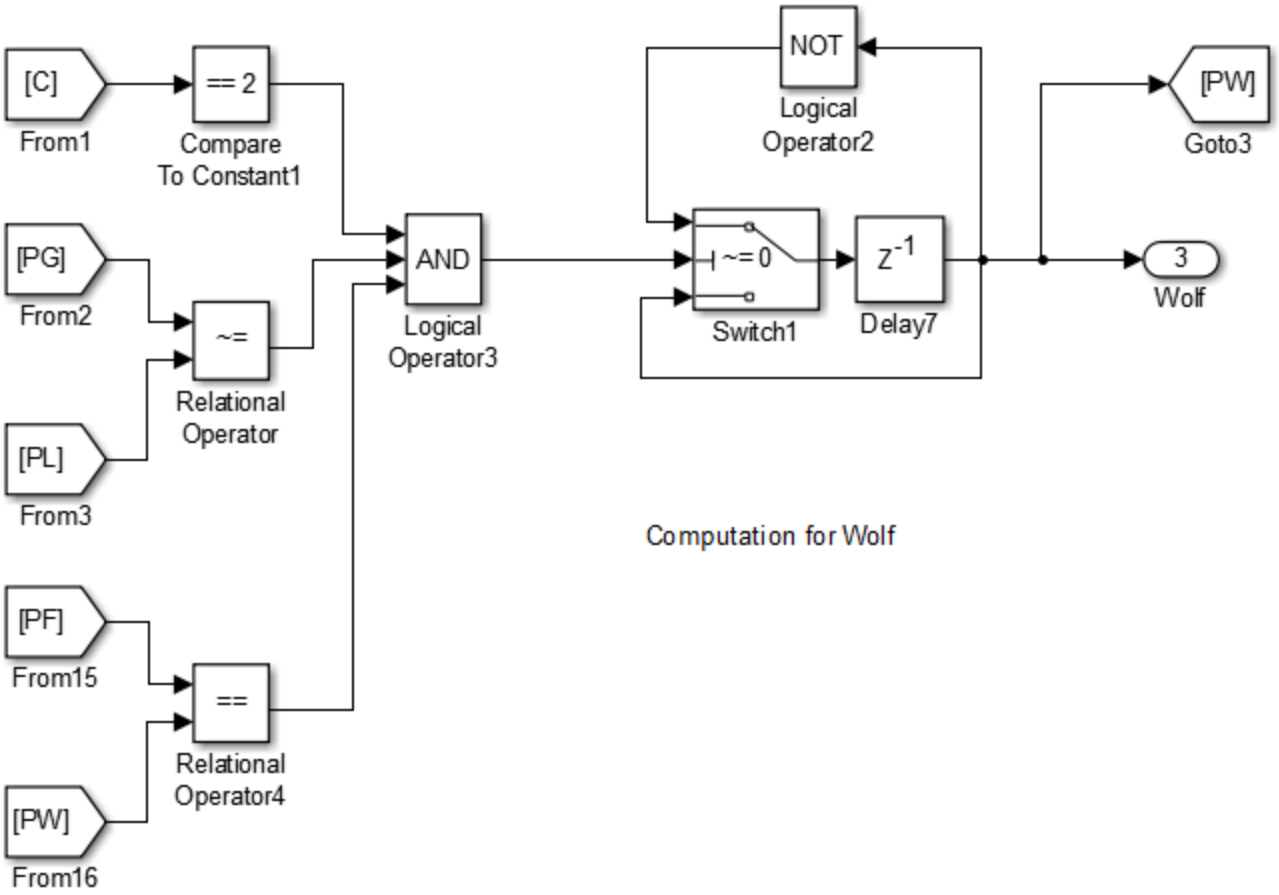
Ferryman problem

- A ferry man has to take his goat, cabbage and wolf to the other side of the river. He can take only one of them with him.
- If he takes the cabbage and leaves the wolf and goat together - a fat wolf will be left when he comes back.
- If he takes the wolf a fat goat will be left behind when he comes back.
- The animals behave themselves when he is around.
- How does he solve this problem? SLDV can do this …

SLDV model



SDV Model



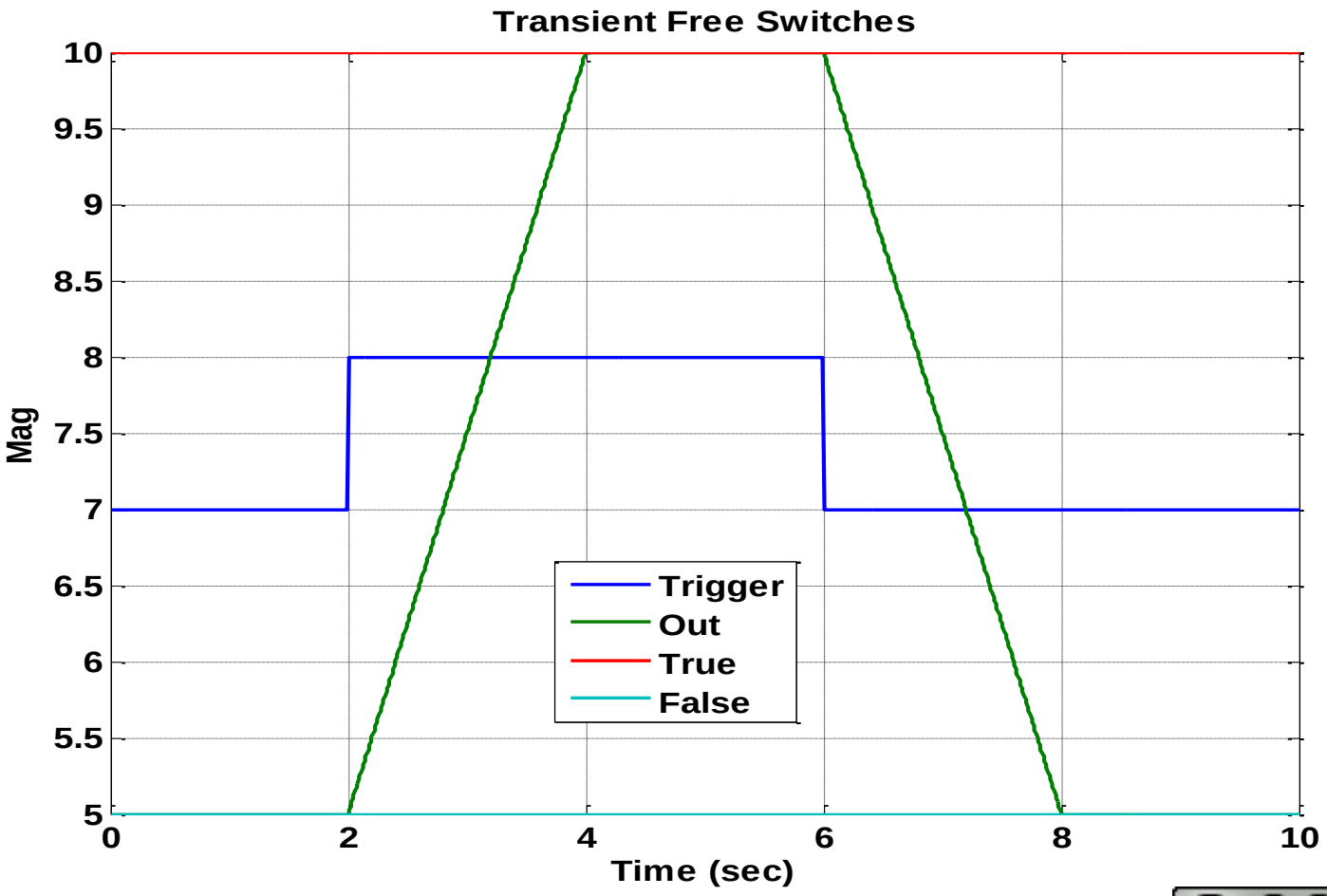
Example India Program

LCA Slat Failure

- The pilot was switching through very fast. The standby gain was selected, the slat was being retracted and the undercarriage was coming down
- In this dynamic situation the Slat failure warning came on.
- This was due to an error in the algorithm of the transient free switch. The behavior was not as expected
- Testing had brought out the failure but no changes were made to the design
- No design change was made even after the incident. This led to another incident

Erratic Fader Logic

The normal behavior of the fader logic. Output fades from 5 to 10 and back from 10 to 5 in 2 seconds based on trigger.

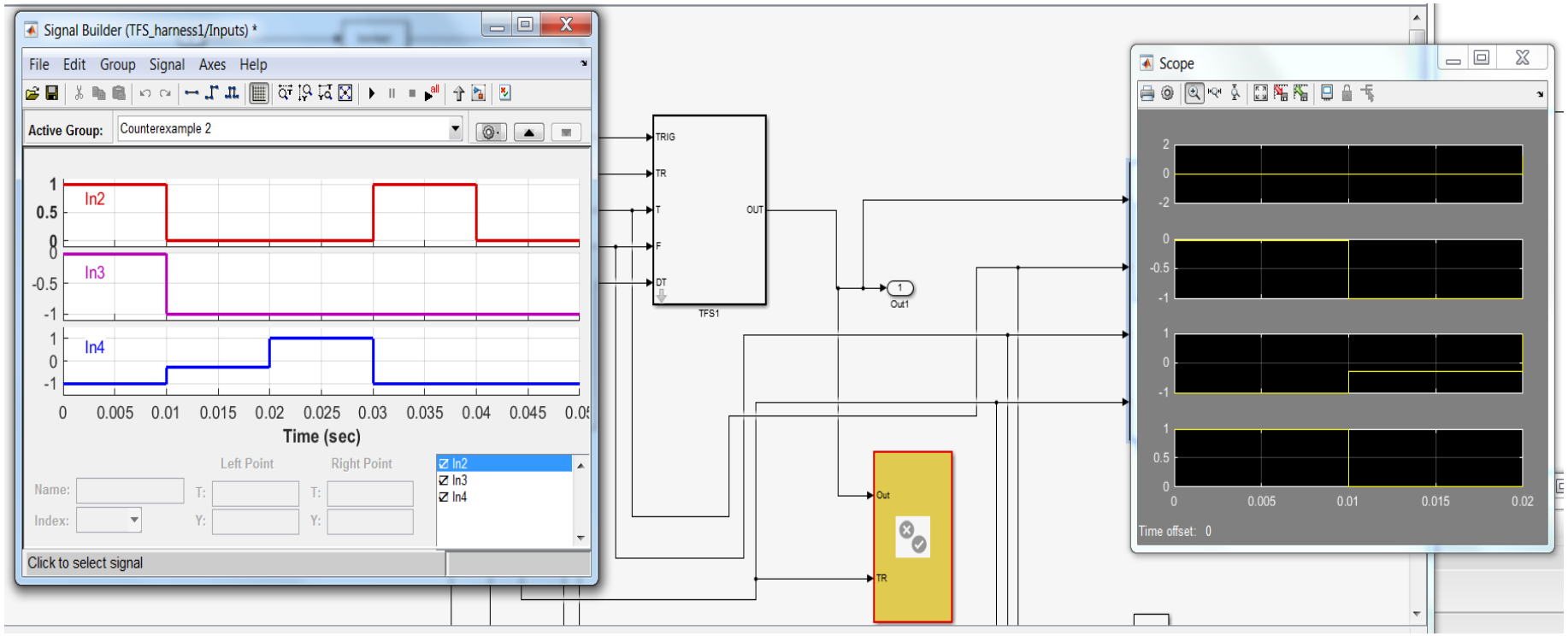


Erratic Fader Logic

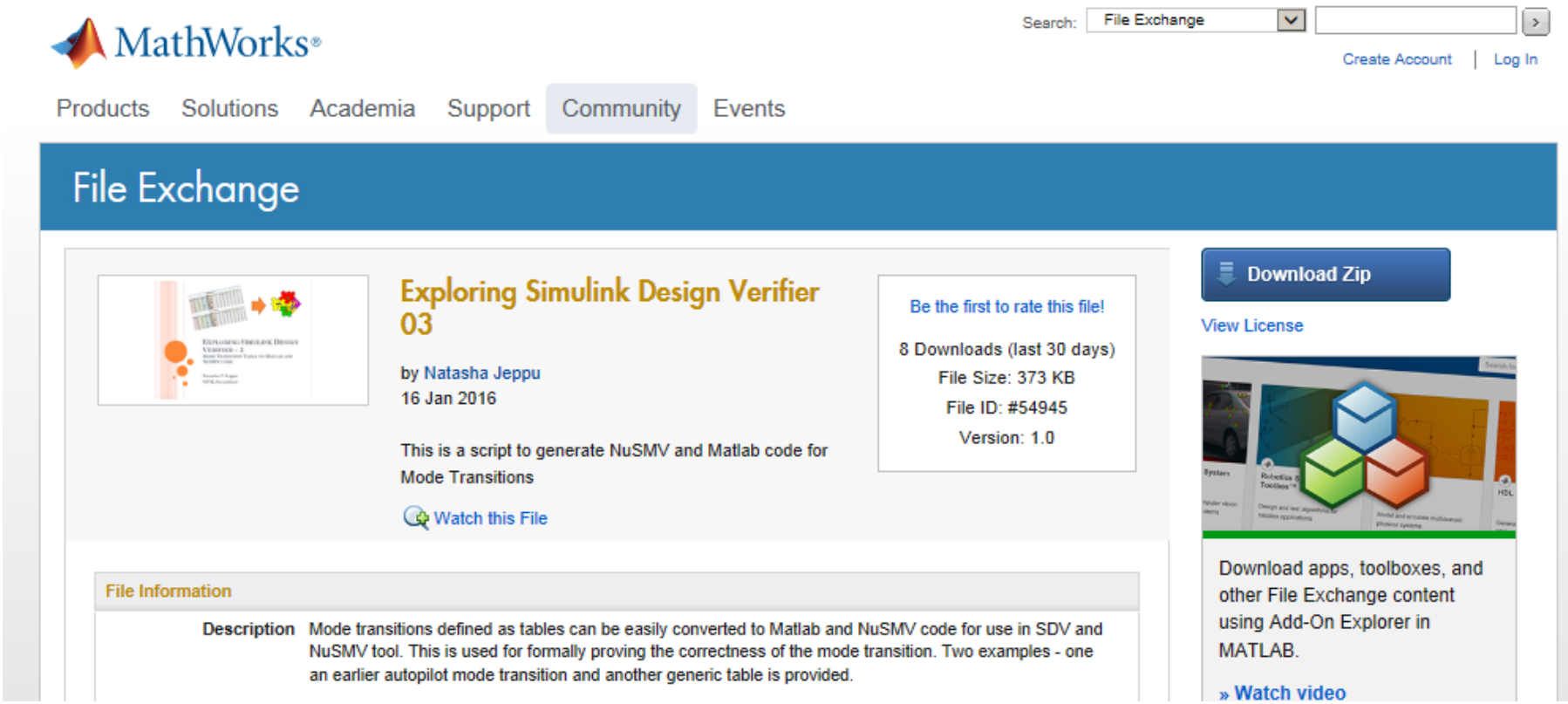
The output signal (green) is greater (nearly double) than the inputs (amplitude 1.0)

Formal Method Counter Example

The test case makes the output go greater than 1.0!

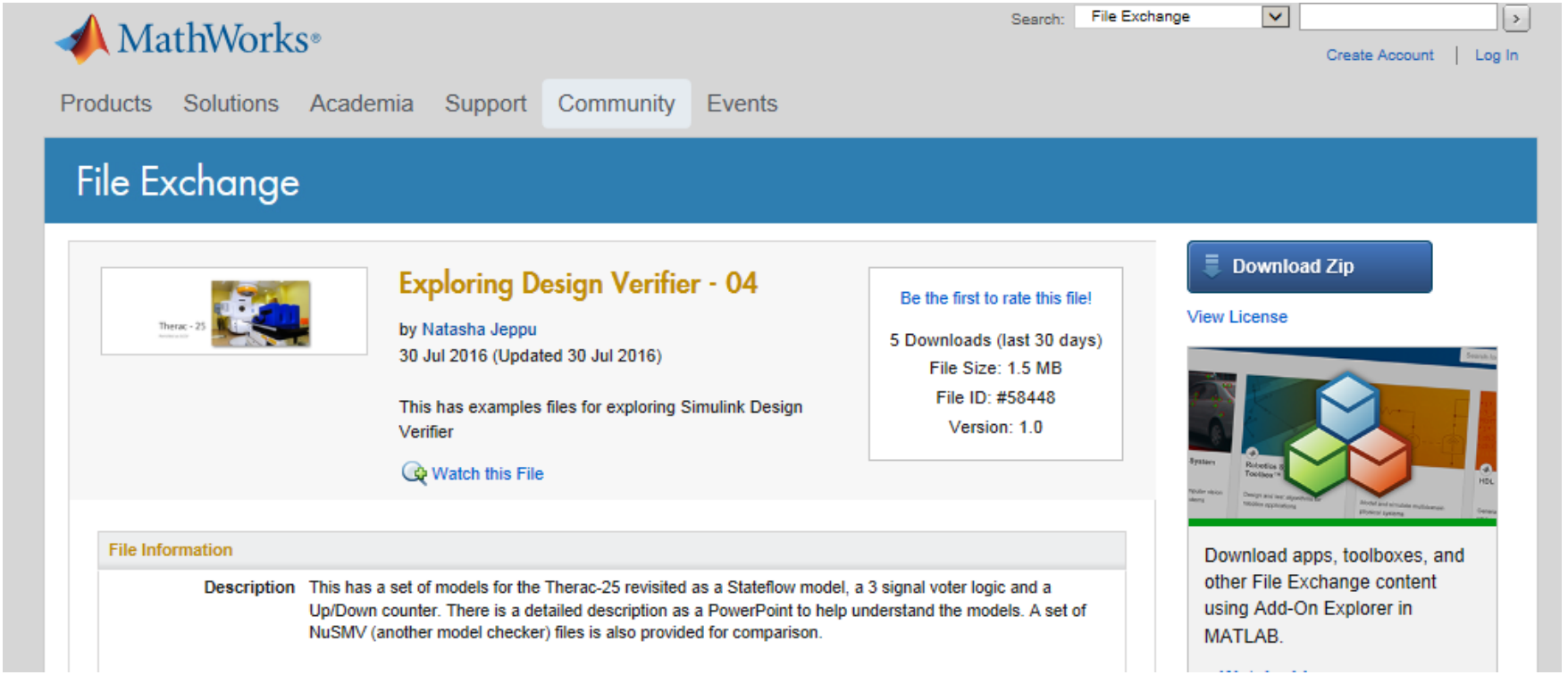


Autopilot problem



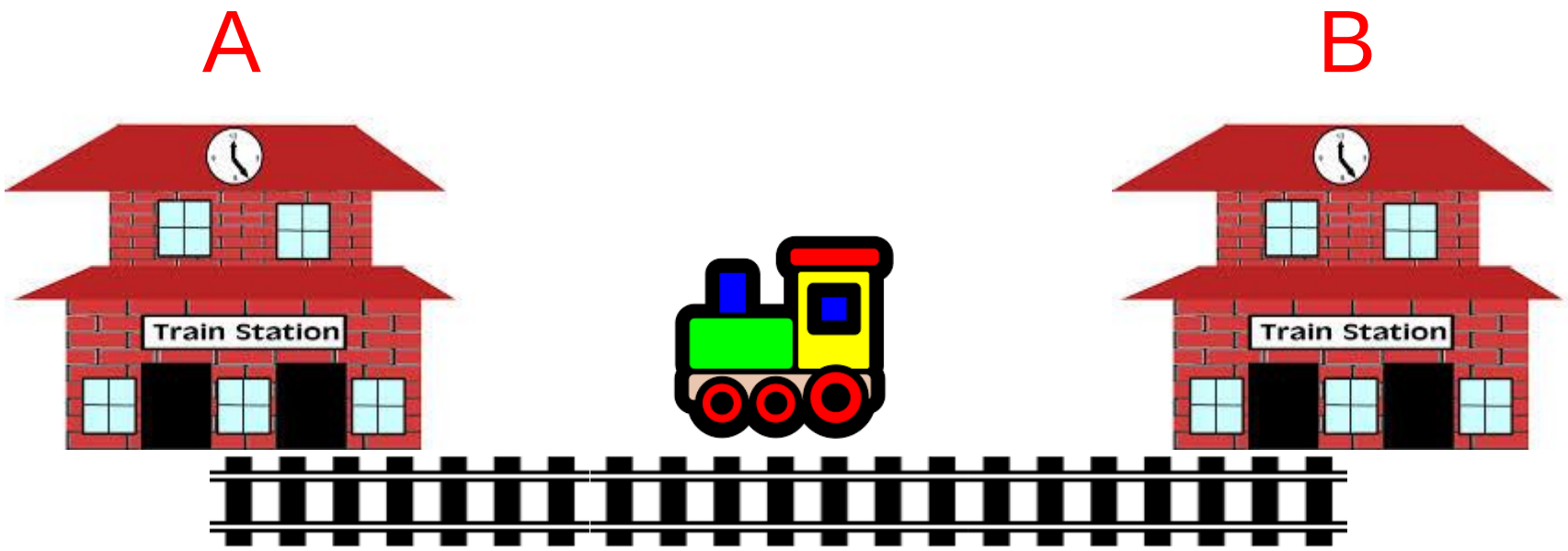
http://in.mathworks.com/matlabcentral/fileexchange/54945-exploring-simulink-design-verifier-03

Therac - 25



http://in.mathworks.com/matlabcentral/fileexchange/58448-exploring-design-verifier-04

A Simple Train Problem



Train Requirements

- This is a design error in an aerospace system camouflaged as a railway problem. The error was found during the Verification and Validation phase.
- The train can be AT_A or AT_B or IN_BET (in between) stations A and B or IN_DET (indeterminate state if there is an error). It can be in only one state at a time.

- Let us look at this problem in all the different formal methods applications like theorem proving, model checking, abstract interpretation

Train Requirements

- The system shall set AT_A to TRUE when sensor A is TRUE
- The system shall set AT_B to TRUE when sensor B is TRUE
- The system shall set IN_BET to TRUE when sensor A is FALSE AND sensor B is FALSE
- The system shall set IN_DET to TRUE when (sensor A is TRUE AND sensor B is TRUE) OR FAIL is TRUE

- Only one output flag out of {AT_A, AT_B, IN_BET, IN_DET} shall be set TRUE in a frame of computation

Theorem proving CVC4

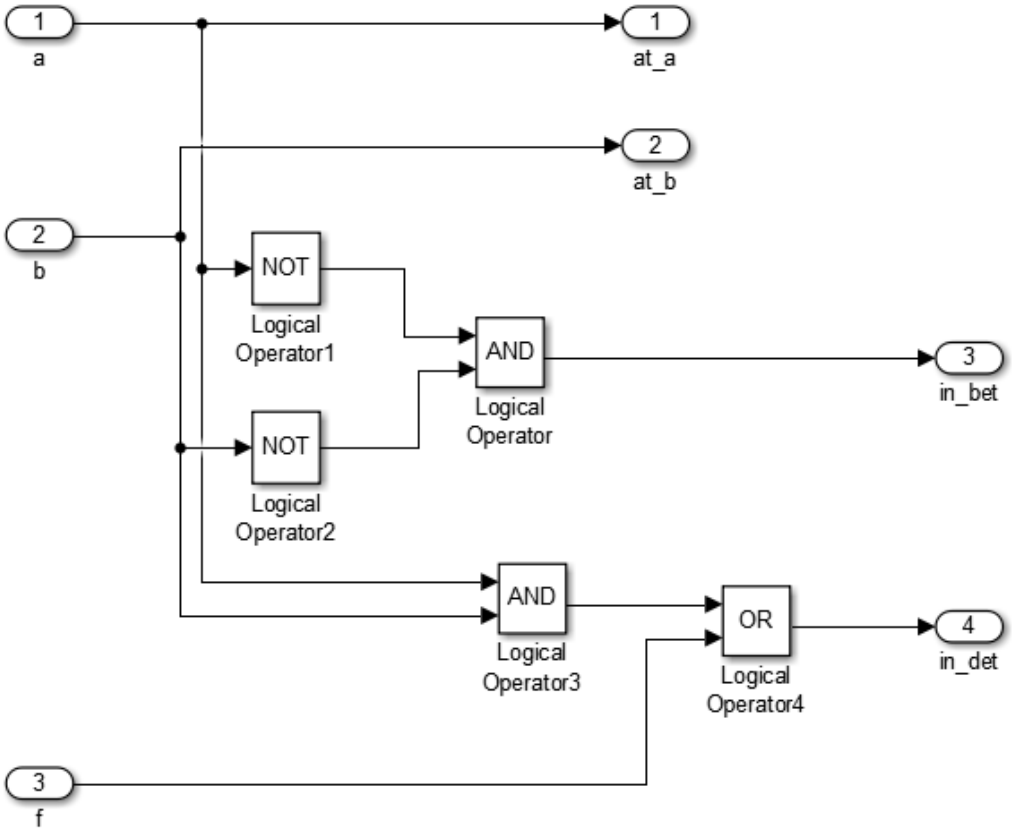
- OPTION "incremental";
- OPTION "produce-models";
- a, b, in_bet, in_det, f, at_a, at_b: BOOLEAN;
- ASSERT at_a = a;
- ASSERT at_b = b;
- ASSERT in_bet = ((NOT a) AND (NOT b));
- ASSERT in_det = ((a AND b) OR f);
- QUERY at_a => (NOT at_b) AND (NOT in_bet) AND (NOT in_det);
- COUNTERMODEL;

Note the assertion at_a TRUE implies that all other flags are FALSE

This gives an error and a counter example is created

Model Checking using SLDV

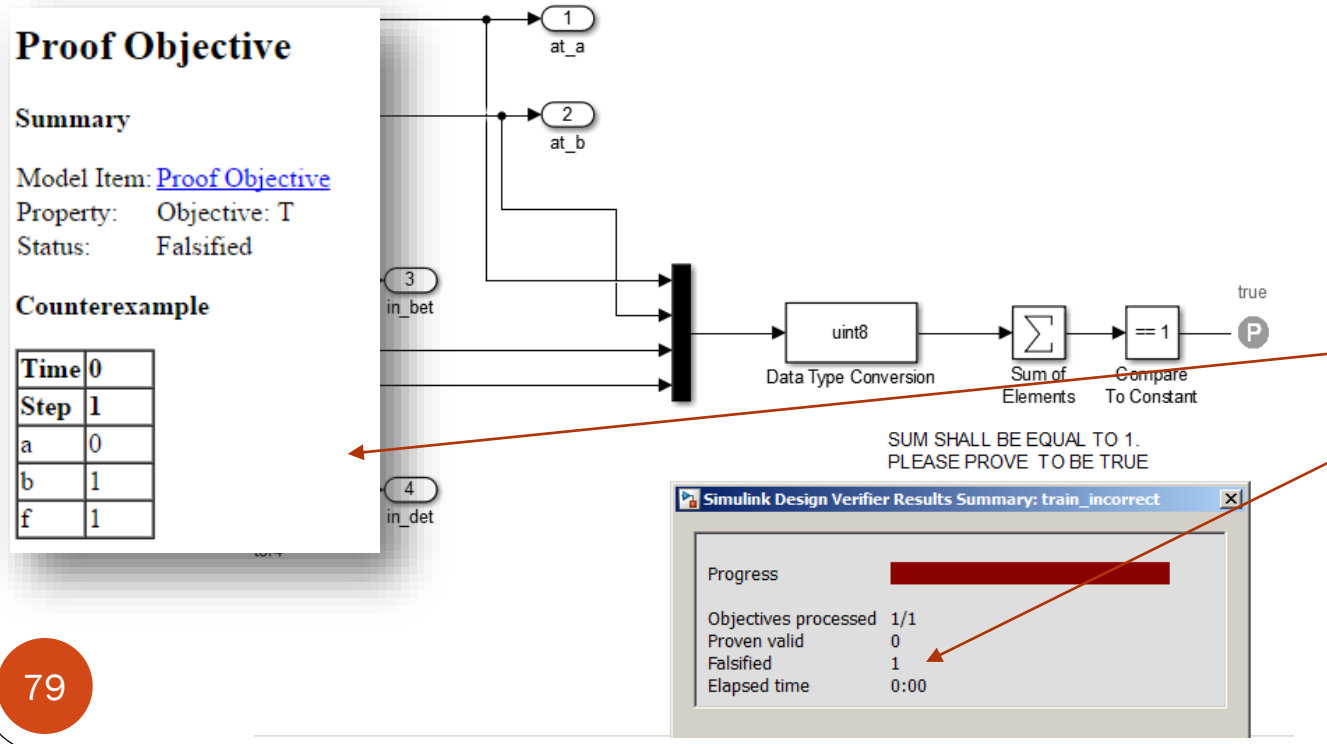
- A model is made in Simulink and the model is checked against the assertion using Simulink Design Verifier.



The requirements are modeled. This is the design model

Model Checking using SLDV

- The design model is cheeked against the assertion that only one output shall be true. In other words if converted to integer and summed up the sum shall be equal to 1 exactly



The assertion is falsified like in the case of the theorem proving. A counter example is also provided.

Model Checking using NuSMV

- NuSMV is an open source model checker obtained from http://nusmv.fbk.eu/
- The problem can be modeled in NuSMV and the assertion put into the code to check the correctness of the model
- All tools behave on the same premise. Each one has its nuances, difficulties in understanding and execution speed.
- **But they are all correct!**

Model Checking using NuSMV

```
MODULE main
VAR
a : boolean;
b : boolean;
f : boolean;
at_a : boolean;
at_b : boolean;
in_bet : boolean;
in_det : boolean;

ASSIGN
at_a := a;
at_b := b;
in_bet := !a & !b;
in_det := (a & b) | f;

LTLSPEC G (at_a -> (!at_b & !in_bet & !in_det));
LTLSPEC G ((toint(at_a) + toint(at_b) + toint(in_bet) + toint(in_det)) = 1);
```

This is the NuSMV code. You have a declaration section, assignment and the assertion as a LTL (Linear Temporal Logic) specification. Two different types of LTL specifications are shown here.

https://en.wikipedia.org/wiki/Linear_temporal_logic

Model Checking using NuSMV

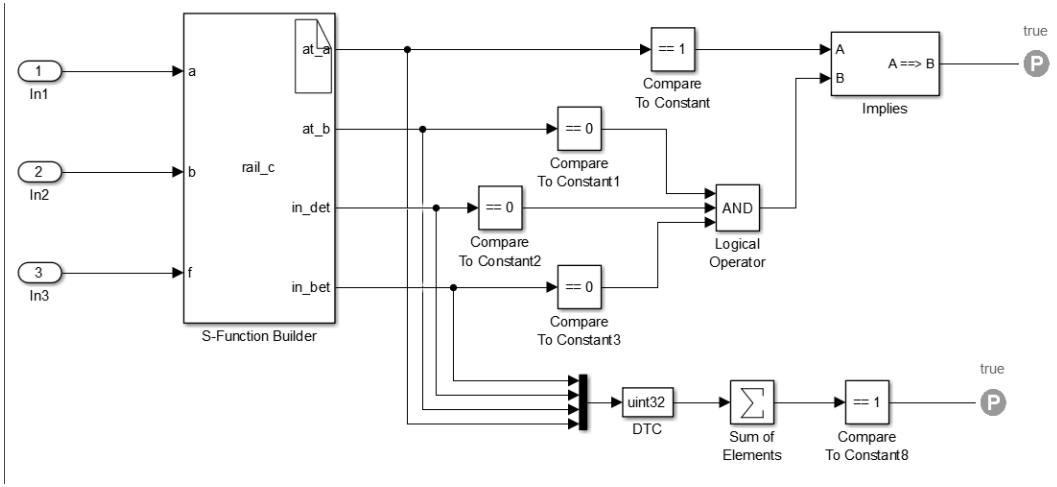
```
-- specification  G ((toint(at_a) + toint(at_b)) + toint(in_bet)) + toint(in_det) = 1  is false
-- as demonstrated by the following execution sequence
Trace Description: LTL Counterexample
Trace Type: Counterexample
  -> State: 2.1 <-
    a = TRUE
    b = TRUE
    f = FALSE
    at_a = TRUE
    at_b = TRUE
    in_bet = FALSE
    in_det = TRUE
  -- Loop starts here
  -> State: 2.2 <-
    b = FALSE
    at_b = FALSE
    in_det = FALSE
  -> State: 2.3 <-
```

NuSMV provides a counter example that can be used for testing and debug.

https://en.wikipedia.org/wiki/Linear_temporal_logic

C code checking using SLDV

```
in_bet[0]=0;
in_det[0]=0;

at_a[0] = a[0];
at_b[0] = b[0];

if ((a[0] == 0) && (b[0] == 0))
{
   in_bet[0] = 1;
}

if ((a[0] == 1 && b[0] == 1) || (f[0] == 1))
{
    in_det[0] = 1;
}
```



Simulink Design Verifier enables you to compile your C code and check it in the Simulink environment. This is still model checking of the C code as a model.

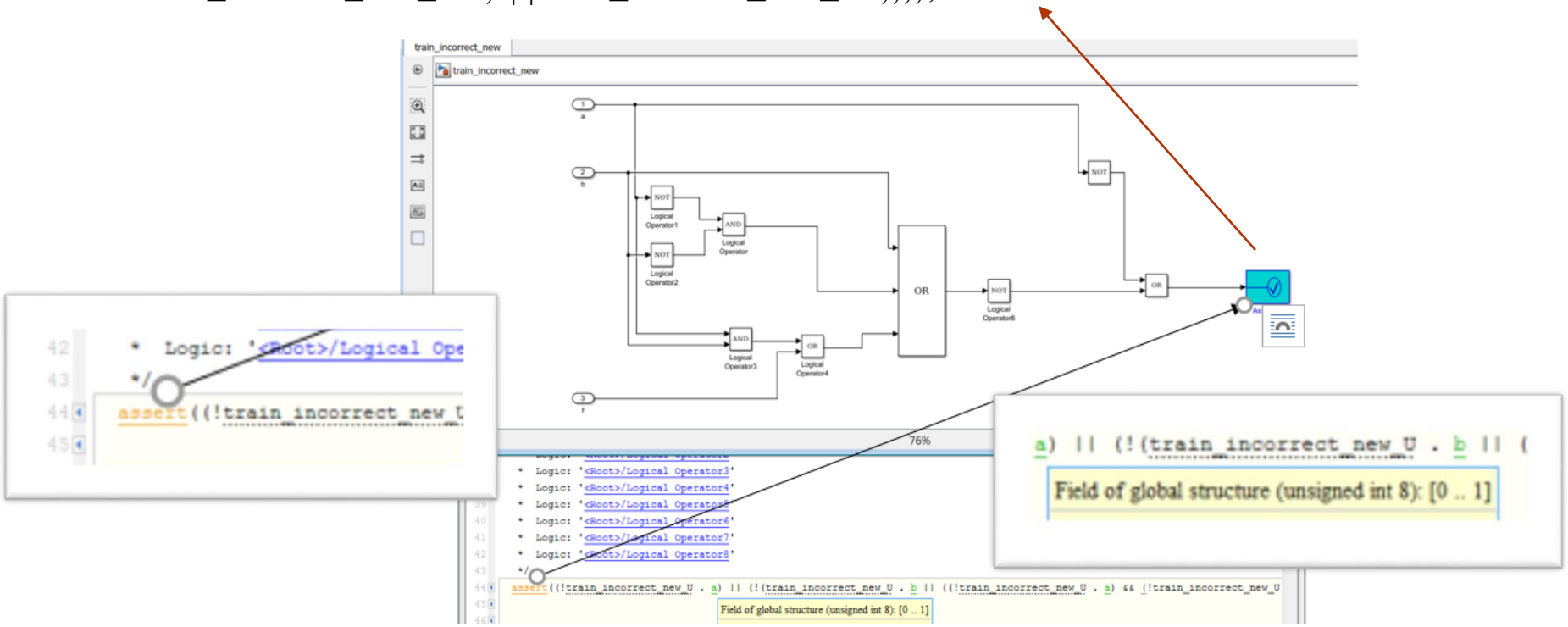
Abstract Interpretation in Polyspace

- A Simulink model can be autocoded and checked using Polyspace for correctness. This works on a legacy hand code also.
- It is possible to insert assertions into the code like in the case of the model checking
- Polyspace indicates that there is a possibility of an error in the assertions using abstract interpretation.

https://in.mathworks.com/products/polyspace.html

Abstract Interpretation in Polyspace

utAssert((!train_incorrect_new_U.a) || (!(train_incorrect_new_U.b || ((!train_incorrect_new_U.a) && (!train_incorrect_new_U.b)) || ((train_incorrect_new_U.a && train_incorrect_new_U.b) || train_incorrect_new_U.f))));



Abstract interpretation looks at the variable value ranges and checks for correctness.

Train Correct Requirements

- The system shall set AT_A to TRUE when sensor A is TRUE AND sensor B is FALSE AND FAIL is FALSE
- The system shall set AT_B to TRUE when sensor B is TRUE AND sensor A is FALSE AND FAIL is FALSE
- The system shall set IN_BET to TRUE when sensor A is FALSE AND sensor B is FALSE AND FAIL is FALSE
- The system shall set IN_DET to TRUE when (sensor A is TRUE AND sensor B is TRUE) OR FAIL is TRUE

Train Correct Requirements – CVC4

```
OPTION "incremental";
OPTION "produce-models";
a, b, in_bet, in_det, f,  at_a, at_b: BOOLEAN;
ASSERT at_a = (a AND (NOT b) AND (NOT f));
ASSERT at_b = (b AND (NOT a) AND (NOT f));
ASSERT in_bet = ((NOT a) AND (NOT b) AND (NOT f));
ASSERT in_det = ((a AND b) OR f);
QUERY at_a => ((NOT at_b) AND (NOT in_bet) AND (NOT in_det));
QUERY at_b => ((NOT at_a) AND (NOT in_bet) AND (NOT in_det));
QUERY in_bet => ((NOT at_b) AND (NOT at_b) AND (NOT in_det));
QUERY in_det => ((NOT at_b) AND (NOT at_b) AND (NOT in_bet));
COUNTERMODEL;
```

```
C:\yoga_personal\training\CVC3>cvc4 < train_prob_c.txt
valid
valid
valid
valid
```

All queries are valid and satisfied.

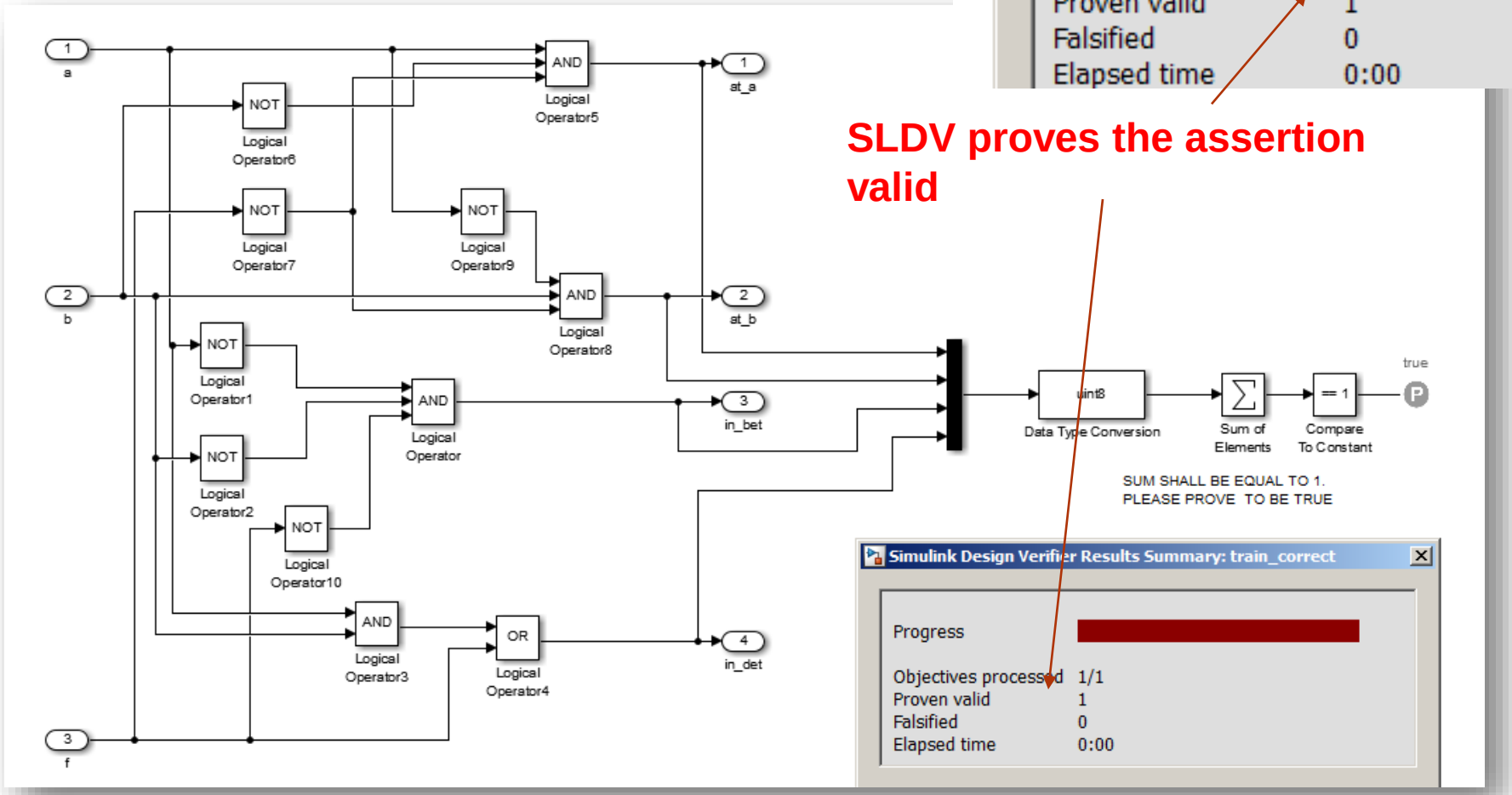
Train Correct – NuSMV

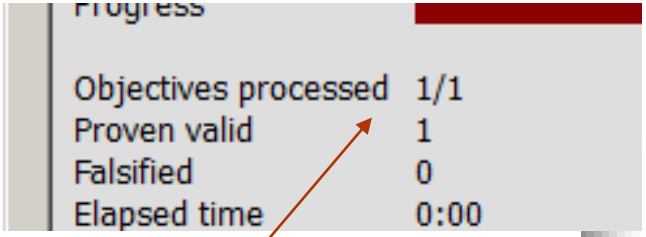
```
ASSIGN
at_a := a & !b & !f;
at_b := b & !a & !f;
in_bet := !a & !b & !f;
in_det := (a & b) | f;

LTLSPEC G (at_a -> (!at_b & !in_bet & !in_det));

LTLSPEC G ((toint(at_a) + toint(at_b) + toint(in_bet) + toint(in_det)) = 1);
```

```
-- specification  G (at_a -> ((!at_b & !in_bet) & !in_det))  is true
-- specification  G ((toint(at_a) + toint(at_b)) + toint(in_bet)) + toint(in_det) = 1  is true
```

All LTL specifications are found TRUE.

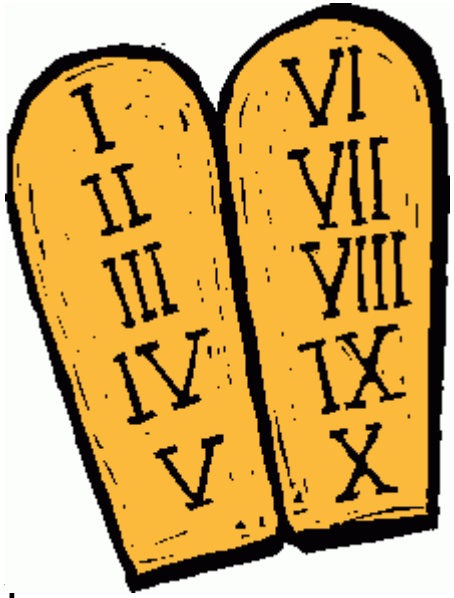
Train Correct – SLDV





SLDV proves the assertion valid

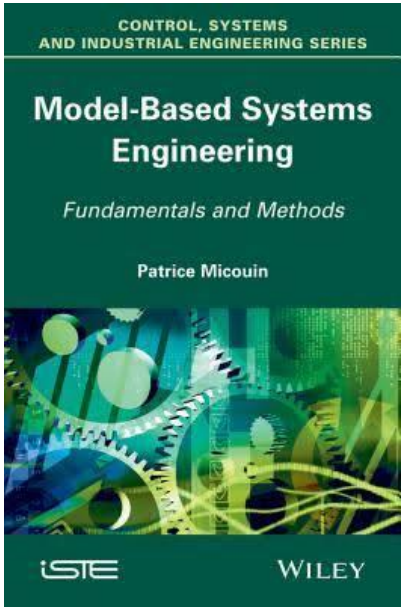
10 Commandments

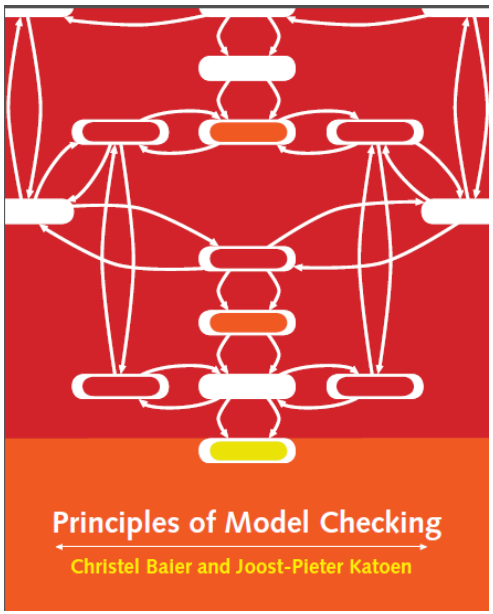


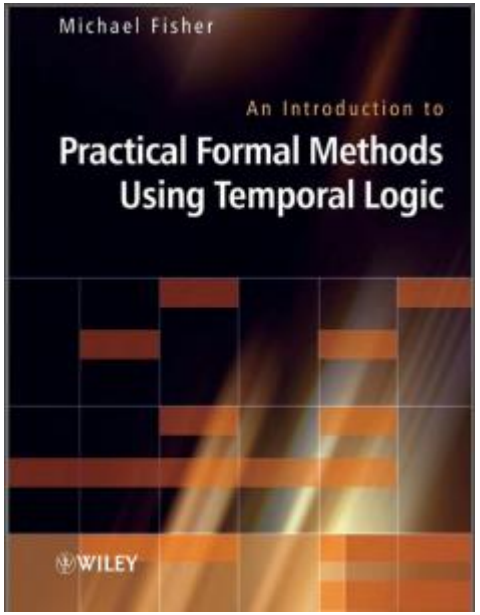
- Thou shalt choose an appropriate notation
- Thou shalt formalize but not over-formalize
- Thou shalt estimate costs
- Thou shalt have a formal methods guru on call
- Thou shalt not abandon thy traditional development methods
- Thou shalt document suffciently
- Thou shalt not compromise thy quality standards
- Thou shalt not be dogmatic
- Thou shalt test, test, and test again
- Thou shalt reuse

These will help you

- Wiki is a very good place for preliminary. But it can be also complicated. These books are good







http://www.cs.utexas.edu/users/EWD/ **especially 498**

Disclaimer: I am not advising you to buy these books

Thank you

- Contact details
- Yogananda Jeppu
- yvjeppu@gmail.com

Always Remember

Our system is only as good as the test cases we have designed to prove it correct.