A selective view of AI Trustworthiness methods: How far can we go?

Zakaria Chihani
Zakaria.chihani@cea.fr

July 25th, 2022
Formal Methods

- **What are formal methods?**
  - Math-based techniques with rigorously established theoretical foundations
  - Used for the specification, development, test and verification of software and hardware

- **Why use formal methods?**
  - Non validated software can have dire consequences and mathematical analysis can contribute to the reliability and robustness
  - Some certification standards call for (e.g., DO-178C for avionics) or even mandate (e.g., ISO/IEC 15408) the use of formal methods
Formal Methods: Verification

Program

set of inputs

Specification

Verification problem

Formal verification

Certified answer
\[ \forall x \in D \]
Not verified
(+ Counterexample)
Formal Methods: Verification

Program

\[
\forall x \in D
\]

Certified answer

Not verified
(+ Counterexample)

set of inputs

\(D\)

Specification

Formal verification
Formal Methods : Property-based Testing

D

Specification ➔ Property-based testing ➔ Generate inputs ➔ Program

Zakaria Chihani
Formal Methods: Property-based Testing

\[ D \]

Property-based testing

Generate inputs

Specification  \rightarrow  \rightarrow  Program

Commissariat à l’Énergie Atomique et aux Énergies Alternatives

Zakaria Chihani
In the FM history...
In 1979:
“[P]rogram verification is bound to fail. We can't see how it's going to be able to affect anyone's confidence about programs”

“Social processes and proofs of theorems and programs”, Communications of ACM. By Richard De Millo, Richard Lipton, and Alan Perlis.
In 1979:
“[P]rogram verification is bound to fail. We can't see how it's going to be able to affect anyone's confidence about programs”

“Social processes and proofs of theorems and programs”, Communications of ACM. By Richard De Millo, Richard Lipton, and Alan Perlis.

- Distinguished Professor of Computing at the Georgia Tech
- VP and CTO of Hewlett-Packard
- Yale, Berkeley, Princeton, Georgia Tech
- Knuth Prize winner
- ACM, Carnegie Mellon, Yale, Purdue
- The first recipient of the Turing Award
The influence

Language used

University courses

Choice of PhD subjects

Credibility of projects

Funding opportunities
Sounds familiar?

Been there…
That was a close call!

Credit: Csaba Martinák
The Ariane 5 reused the inertial reference platform from the Ariane 4, but the Ariane 5's flight path differed considerably from the previous models.

The greater horizontal acceleration caused a data conversion from a 64-bit floating point number to a 16-bit signed integer value to overflow and cause a hardware exception.

The subsequent automated analysis of the Ariane code (written in Ada) was the first example of large-scale static code analysis by abstract interpretation (one of the FM that we will be seeing shortly).
The Ariane 5 reused the inertial reference platform from the Ariane 4, but the Ariane 5's flight path differed considerably from the previous models.

The greater horizontal acceleration caused a data conversion from a 64-bit floating point number to a 16-bit signed integer value to overflow and cause a hardware exception.

The subsequent automated analysis of the Ariane code (written in Ada) was the first example of large-scale static code analysis by abstract interpretation (one of the FM that we will be seeing shortly).

Good example to scare students: “If you don’t use formal methods, your failure will be on every presentation of every FM slideshow.”
We have come a long way
Fast forward a few decades
Fast forward a few decades
We have been here before
We have been here before

- The first solvers and analyzers were **not** efficient or scalable.
- For example, today’s SAT solvers can automatically solve problem instances involving **tens of thousands of variables and millions of constraints**.
- But it wasn’t always the case! We needed to invent DPLL, CDCL, Symmetry breaking, two-watched literals, WalkSAT, adaptive branching, random restarts, portfolio, divide-and-conquer, parallel local search...
Much faster evolution (e.g., Reluplex → Marabou)

Parallelization Techniques for Verifying Neural Networks

2020- Parallelism

The Marabou Framework for Verification and Analysis of Deep Neural Networks

2019- Dedicated simplex etc.

Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks

2017- inception

Efficient Neural Network Analysis with Sum-of-Infeasibilities

2022- SoI

An Abstraction-Based Framework for Neural Network Verification

2020- Abstractions
But same trend for others

- PyRAT
- ReluVal
- CNN-Cert
- Neurify
- Planet
- Sherlock
- Eran
- MIPVerify
- Sherlock

Commissariat à l’énergie atomique et aux énergies alternatives

Zakaria Chihani
But we are evolving much faster for AI

- Artificial Intelligence Safety Engineering (WAISE, at SafeComp)
- AISafety (at IJCAI)
- Safe AI (at AAAI)
- Verification of Neural Networks (VNN, at AAAI or CAV)
- Formal Methods for ML-Enabled Autonomous Systems (FoMLAS, at CAV)
- Machine Learning with Guarantees (ML with Guarantees, at NeurIPS)
- Safe Machine Learning (SafeML, at ICLR)
- Privacy in Machine Learning (PriML, at NeurIPS)
- Security and Safety in Machine Learning Systems (AISecure, at ICLR)
- Dependable and Secure Machine Learning (DSML, at DSN)

In addition to dedicated tracks in FM venues
Global view of the discipline
Validation: a three-players game

<table>
<thead>
<tr>
<th>Developer's side</th>
<th>Object to certify</th>
</tr>
</thead>
<tbody>
<tr>
<td>• What is the architecture of the software, how can it be modified to be more amenable to verification, will these modifications cost too much? (Activation functions of NN, kernel function of SVM, etc.)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Validator's side</th>
<th>Properties to verify</th>
</tr>
</thead>
<tbody>
<tr>
<td>• What to verify, how to formally specify it, how is it decomposed in smaller bits? (Robustness, metamorphism, behavior specification, etc.)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Methods and tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>• How to verify, what methods fit my problem, can the tools be helped with heuristics? (Abstract interpretation, SMT solving, symbolic execution, Constraint programming, etc.)</td>
<td></td>
</tr>
</tbody>
</table>
The Property

What to verify,
how to formally specify it,
how is it decomposed into smaller parts
Formally specifiable (structured data)

- **Semantics** (directly related to the domain) can be described through math and logic.
- Can be proved on all possible inputs.

Description: If the intruder is near and approaching from the left, the network advises “strong right”.

Input constraints: $250 \leq \rho \leq 400$, $0.2 \leq \theta \leq 0.4$, $-3.141592 \leq \psi \leq -3.141592 + 0.005$, $100 \leq v_{\text{own}} \leq 400$, $0 \leq v_{\text{int}} \leq 400$. 
Properties

Not directly specifiable (unstructured data)

- Semantics of the input cannot be described through math and logic (e.g. an image containing a jacket)
- Only abstracted properties can be described through math and logic (e.g. “distance” between two images)
- Cannot realistically be proved on all possible inputs
Ideally the selected data to build and validate the model is representative of the intended distribution.
Ideally the selected data to build and validate the model is representative of the intended distribution.
Properties

Ideally the selected data to build and validate the model is representative of the intended distribution.

What we want to avoid is a model that only knows what it was shown.
Ideally the selected data to build and validate the model is representative of the intended distribution.

What we want to avoid is a model that only knows what it was shown.

How does it behave with the neighborhood of selected data?
Ideally the selected data to build and validate the model is representative of the intended distribution.

What we want to avoid is a model that only knows what it was shown

How does it behave with the neighborhood of selected data?
- Formally **verify**: guarantee, but can be costly
Ideally the selected data to build and validate the model is representative of the intended distribution.

What we want to avoid is a model that only knows what it was shown.

How does it behave with the neighborhood of selected data?
  - Formally **verify**: guarantee, but can be costly
  - **Test a lot**: no guarantee, but faster
Ideally the selected data to build and validate the model is representative of the intended distribution.

What we want to avoid is a model that only knows what it was shown

How does it behave with the neighborhood of selected data?
- Formally **verify**: guarantee, but can be costly
- **Test a lot**: no guarantee, but faster
Ideally the selected data to build and validate the model is representative of the intended distribution.

What we want to avoid is a model that only knows what it was shown.

How does it behave with the neighborhood of selected data?

What is it good for?
Ideally the selected data to build and validate the model is representative of the intended distribution.

What we want to avoid is a model that only knows what it was shown.

How does it behave with the neighborhood of selected data?

What is it good for?

Can detect this…
Ideally the selected data to build and validate the model is representative of the intended distribution.

What we want to avoid is a model that only knows what it was shown

How does it behave with the neighborhood of selected data?

**What is it good for?**

... But doesn’t imply this
Properties

Can also generate other data from acceptable transformations
Can also generate other data from acceptable transformations
Can also generate other data from acceptable transformations

To see if the network can handle a wider distribution.
Can also generate other data from acceptable transformations

To see if the network can handle a wider distribution.

And we can combine methods
Side note: what phase?

Post-conception

Intervals around data

Transformed data

Train

Validate
Side note: what phase?

Post-conception

- Intervals around data
- Transformed data

At conception

- Intervals around data
- Transformed data
The Methods
How to verify
what methods fit my problem
Abstract interpretation

We would like to verify a property on the all possible values of inputs $x \in [a,b]$ and $y \in [c,d]$ in some program.

e.g.:

$$x + y \in [a+c, b+d]$$

Do the same for all operations in the program.

Use other types of domain for more precision (not just intervals).
Abstract interpretation

precise analysis

\[ A \subseteq S \implies P \subseteq S \]
Conservative overapproximation

precise analysis
\[ A \subseteq S \implies P \subseteq S \]

false alarm
\[ A \not\subseteq S \text{ but } P \subseteq S \]

Credit: Antoine Miné
Abstract interpretation on NN

Input

A shape that abstracts all possible perturbations

Convolution

A shape that abstracts all possible outputs

Dense

Guaranteed to classify to label 8

Not guaranteed to classify to label 8

Crédit: SRI lab, ETH Zurich;
Still a long way ahead
Not the complete picture…

“Traditional” human-written software

General purpose provers and platforms (Why3, Alt-Ergo, Z3, Colibri, TLA+, K-framework,...)

<table>
<thead>
<tr>
<th>Software</th>
<th>Machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-Method</td>
<td>aiT WCET analyzers</td>
</tr>
<tr>
<td>Krakatoa, OpenJML (Java programs)</td>
<td>seL4 (proven microkernel)</td>
</tr>
<tr>
<td>SPARK (Ada programs)</td>
<td>BINSEC</td>
</tr>
<tr>
<td>Frama-C, Astrée Polyspace (C/C++ programs)</td>
<td>BAP</td>
</tr>
<tr>
<td>CompCert (Compilation of C)</td>
<td>Miasm</td>
</tr>
<tr>
<td>GATeL (Lustre programs)</td>
<td>CoVaC (Compiler validation)</td>
</tr>
<tr>
<td>Polyspace (C/C++ programs)</td>
<td>Unisim</td>
</tr>
<tr>
<td>High level (model... source code...)</td>
<td>... low level</td>
</tr>
<tr>
<td>SCADE</td>
<td>Miasm</td>
</tr>
<tr>
<td>Papyrus</td>
<td>BAP</td>
</tr>
<tr>
<td>CompCert</td>
<td>seL4</td>
</tr>
<tr>
<td>seL4 (proven microkernel)</td>
<td>BAP</td>
</tr>
<tr>
<td>SCADE</td>
<td>Miasm</td>
</tr>
</tbody>
</table>

Machine generated models
“Traditional” human-written software

General purpose provers and platforms (Why3, Alt-Ergo, Z3, Colibri, TLA*, K-framework, ...)

<table>
<thead>
<tr>
<th>Software</th>
<th>Machine generated models</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-Method</td>
<td>aiT WCET analyzers</td>
</tr>
<tr>
<td>Krakatoa, OpenJML (java programs)</td>
<td></td>
</tr>
<tr>
<td>Frama-C, Astrée</td>
<td></td>
</tr>
<tr>
<td>Polyspace (C/C++ programs)</td>
<td></td>
</tr>
<tr>
<td>CompCert (Compilation of C)</td>
<td></td>
</tr>
<tr>
<td>seL4 (proven microkernel)</td>
<td></td>
</tr>
<tr>
<td>BAP</td>
<td></td>
</tr>
<tr>
<td>BINSEC</td>
<td></td>
</tr>
<tr>
<td>Papyrus</td>
<td></td>
</tr>
<tr>
<td>SCADE</td>
<td></td>
</tr>
<tr>
<td>SPARK (Ada programs)</td>
<td></td>
</tr>
<tr>
<td>GATeL (Lustre programs)</td>
<td></td>
</tr>
<tr>
<td>CoVaC (Compiler validation)</td>
<td></td>
</tr>
<tr>
<td>Unisim</td>
<td></td>
</tr>
<tr>
<td>Miasm</td>
<td></td>
</tr>
<tr>
<td>High level (model... source code...)</td>
<td>......</td>
</tr>
<tr>
<td>Machine</td>
<td></td>
</tr>
<tr>
<td>SAVer</td>
<td></td>
</tr>
<tr>
<td>CNN-Cert</td>
<td></td>
</tr>
<tr>
<td>Neurify</td>
<td></td>
</tr>
<tr>
<td>Planet</td>
<td></td>
</tr>
<tr>
<td>ReluVal</td>
<td></td>
</tr>
<tr>
<td>PyRAT</td>
<td></td>
</tr>
<tr>
<td>Sherlock</td>
<td></td>
</tr>
<tr>
<td>MIPVerify</td>
<td></td>
</tr>
</tbody>
</table>

Not the complete picture...
"Traditional" human-written software

Software

High level (model... source code...) ...... ... low level

Machine

General purpose provers and platforms (Why3, Alt-Ergo, Z3, Colibri, TLA+, K-framework,...)

- B-Method
- Krakatoa, OpenJML (Java programs)
- SPARK (Ada programs)
- GATEl (Lustre programs)
- Fram-C, Astére
- Polyspace (C/C++ programs)
- CompCert (Compilation of C)
- aiT WCET analyzers
- seL4 (proven microkernel)
- BINSEC
- BAP
- Miasm
- Unisim
- Polyspace
- GATEl
- Fram-C
- Astére
- SPARK
- B-Method

New Passive and Active Attacks on Deep Neural Networks in Medical Applications
Invited talk
Cheng Gongye, Hongjie Li, Xiang Zhang, Majid Sadjough, Geng Yuan, Xue Lin, Thomas Wahl, and Yunyi Fei

Exploiting Verified Neural Networks via Floating Point Numerical Error
Kai Ji
M. U. U.S.
jianali@mit.edu

Bit-Flip Attack: Crushing Neural Network with Progressive Bit Search
Adnan Sirai Rakin1, Zhezhi He2 and Deliang Fan

Machine generated models

Adam Betts1 Nathan Chong1 Alastair F. Donaldson1 Shaz Qadeer2 Paul Thomson1

Commissariat à l’énergie atomique et aux énergies alternatives
Zakaria Chihani
"Traditional" human-written software

General purpose provers and platforms (Why3, Alt-Ergo, Z3, Colibri, TLA⁺, K-framework,...)

B-Method
SPARK
Krakatoa, OpenJML
GATeL
Frama-C, Astrée
Polyspace
CompCert (Compilation of C)
CoVaC (Compiler validation)
aiT WCET analyzers

Machine generated models

Software

High level (model... source code...)

Not the complete picture...

Not NN

SAVer

CNN-Cert
Neurify
Sherlock

MIPVerify
ReluVal
Planet
PyRAT

Software

BAP
BINSEC
Unisim
Miasm

Machine

New Passive and Active Attacks on Deep Neural Networks in Medical Applications
Invited Talk
Cheng Gongye, Hongjie Li, Xiang Zhang, Majid Sahlagh, Geng Yuan, Xue Lin, Thomas Wahl, and Yumi Foi

Exploiting Verified Neural Networks via Floating Point Numerical Error
Kai Ju
MIT CSAIL
juank@mit.edu

Martin Riedard
MIT CSAIL
rzaang@mit.edu

Bit-Flip Attack: Crushing Neural Network with Progressive Bit Search
Adnan Siraj Rakin¹, Zhezhi He¹ and Deliang Fan

GPUVerify: A Verifier for GPU Kernels *
Adam Betts¹  Nathan Chong¹  Alastair F. Donaldson¹  Shaz Qadeer²  Paul Thomson¹
Challenges
Challenges

“Traditional” software

Symbolic AI

Machine Learning
Challenges

“Traditional” software

Symbolic AI

- Spec
- Part
- Spec
- Part
- Spec
- Part
- Spec
- Part

- Spec
- Constraint
- Spec
- Domain
- Spec
- Propagator
- Spec
- Labelling
- Spec
- Constraint
- Spec
- Domain
- Spec
- Propagator
- Spec
- Labelling

- Spec
- Labelling

Machine Learning

✓
✓
✓
✓
✓
✓
✓
✓
• Scalability: clever heuristics? Parallelisation?
  • Need for more expressive languages to describe more specifications and tools able to handle them
• Coverage of all compilation steps until the embedded software.

• Community building: compare to the SMT-Comp, runtime verification comp, CADE ATP System Competition...
  • Common language, common benchmark, common infrastructure (VNN comp)
• Encourage formation: university masters courses, tutorial in conferences, summer schools...
Conclusion
Conclusion

- FM need adaptation but the challenge is not new
- FM are advancing faster than ever before
- Very dynamic and fast moving environment
- FM are only one tool in what should be a diversified panoply
- FM can intervene at different stages of development