Machine Learning and Invariant Synthesis

Arie Gurfinkel
Dept. of Electrical and Computer Engineering
University of Waterloo

Waterloo ML + Security + Verification Workshop
August 26 – 30, 2019
Waterloo, Ontario, Canada
Automated (Software) Verification

Program and/or model

Alan M. Turing. 1936: “Undecidable”

Alan M. Turing. ”Checking a large routine” 1949

How can one check a routine in the sense of making sure that it is right? The programmer should make a number of definite assertions which can be checked individually, and from which the correctness of the whole programme easily follows.
Symbolic Reachability Problem

\[ P = (V, \text{Init}, \text{Tr}, \text{Bad}) \]

\( P \) is UNSAFE if and only if there exists a number \( N \) s.t.

\[ \text{Init}(X_0) \land \left( \bigwedge_{i=0}^{N-1} \text{Tr}(X_i, X_{i+1}) \right) \land \text{Bad}(X_N) \not\Rightarrow \bot \]

\( P \) is SAFE if and only if there exists a safe inductive invariant \( \text{Inv} \) s.t.

\[ \text{Init} \Rightarrow \text{Inv} \]

\[ \text{Inv}(X) \land \text{Tr}(X, X') \Rightarrow \text{Inv}(X') \]

\[ \text{Inv} \Rightarrow \neg \text{Bad} \]
Inductive Invariants

System S is safe iff there exists an inductive invariant $\text{Inv}$:

- **Initiation:** $\text{Initial} \subseteq \text{Inv}$
- **Safety:** $\text{Inv} \cap \text{Bad} = \emptyset$
- **Consecution:** $\text{TR}(\text{Inv}) \subseteq \text{Inv}$

i.e., if $s \in \text{Inv}$ and $s \rightsquigarrow t$ then $t \in \text{Inv}$
System S is safe iff there exists an inductive invariant Inv:

- **Initiation:** Initial $\subseteq$ Inv
- **Safety:** Inv $\cap$ Bad = $\emptyset$
- **Consecution:** TR(Inv) $\subseteq$ Inv i.e., if s $\in$ Inv and s $\leadsto$ t then t $\in$ Inv

System S is safe if Reach $\cap$ Bad = $\emptyset$
Program Verification with HORN(LIA)

\[
\begin{align*}
z &= x; \ i = 0; \\
\text{assume} \ (y > 0); \\
\text{while} \ (i < y) \text{ \{} \\
\quad z &= z + 1; \\
\quad i &= i + 1; \\
\text{\}} \\
\text{assert} (z == x + y);
\end{align*}
\]

\[
\begin{array}{c}
z = x \land i = 0 \land y > 0 \\
\text{Inv}(x, y, z, i) \land i < y \land z1=z+1 \land i1=i+1 \\
\text{Inv}(x, y, z, i) \land i \geq y \land z \neq x+y \\
\end{array}
\rightarrow
\begin{array}{c}
\text{Inv}(x, y, z, i) \\
\text{Inv}(x, y, z1, i1) \\
\text{false}
\end{array}
\]
In SMT-LIB

(set-logic HORN)

;; Inv(x, y, z, i)
(declare-fun Inv (Int Int Int Int) Bool)

(assert
(forall ( (A Int) (B Int) (C Int) (D Int))
  (=> (and (> B 0) (= C A) (= D 0))
       (Inv A B C D))))

(assert
(forall ( (A Int) (B Int) (C Int) (D Int) (C1 Int) (D1 Int) )
  (=>
    (and (Inv A B C D) (< D B) (= C1 (+ C 1)) (= D1 (+ D 1)))
    (Inv A B C1 D1)
  )
)

(assert
(forall ( (A Int) (B Int) (C Int) (D Int))
  (=> (and (Inv A B C D) (>= D B) (not (= C (+ A B))))
       false
  )
)

(check-sat)
(get-model)
Spacer: Solving SMT-constrained CHC

Spacer: SAT procedure for SMT-constrained Horn Clauses
- now the default CHC solver in Z3
  - https://github.com/Z3Prover/z3
  - dev branch at https://github.com/agurfinkel/z3

Supported SMT-Theories
- Linear Real and Integer Arithmetic
- Quantifier-free theory of arrays
- Universally quantified theory of arrays + arithmetic
- Best-effort support for many other SMT-theories
  - data-structures, bit-vectors, non-linear arithmetic

Support for Non-Linear CHC
- for procedure summaries in inter-procedural verification conditions
- for compositional reasoning: abstraction, assume-guarantee, thread modular, etc.
(Un)Decidability Barrier

The problem of finding a safe inductive invariant is highly undecidable

- In many cases, even whenever the problem of finding a finite counterexample is decidable, the inductive invariant problem remains undecidable
- In particular, in this talk, we assume that all components of the transition system are in linear arithmetic (LIA or LRA)

The problem of validating whether a candidate formula (or set of states) is an inductive invariant is (often) decidable

- In particular, decidability of the counterexample problem implies decidability of validating candidate invariants
- In particular, validating inductive invariants is decidable for transition systems over LRA and LIA

The problem of finding inductive invariant is decidable for transition system over propositional logic

- a.k.a, the Finite State Model Checking
Machine Learning for (Software) Verification

Treat invariant discovery as a machine learning problem

The object being learned is an inductive invariant
  • described in some language or data structure

Samples are various artifacts from program execution
  • e.g., a program state is a vector in $\mathbb{R}^n$

An invariant is a classifier that separates good and bad states
  • A state is good if it is reachable state of the program
  • A state is bad if it can reach a state that violates the property
  • An invariant (if it exists) contains all good states, no bad states, and can classify other states arbitrarily
ML for Verification: The Old Guard

There is a long history of applications of “machine learning” in software verification

• after all, the problem is undecidable and no solution is perfect

For the purpose of this talk, the most relevant are:

Daikon

• Daikon is an implementation of dynamic detection of likely invariants, by M. Ernst, A. Czeislery, W. Griswoldz, and D. Notkin. International Conference on Software Engineering (ICSE) 2000.

Houdini

• Cormac Flanagan, K. Rustan M. Leino: Houdini, an Annotation Assistant for ESC/Java. FME 2001: 500-517
Daikon: Overview

Determined Invariants

1.) \( n \geq 0 \)
2.) \( s = \text{SUM}(B) \)
3.) \( 0 \leq i \leq n \)
Houdini: Maximal Inductive Subset

Let $L$ be a set of formulas, $P=(V, \text{Init}, \text{Tr}, \text{Bad})$ a program. A subset $X$ of $L$ is a maximal inductive subset iff it is the largest subset of $X$ such that

$$\text{Init}(u) \Rightarrow \bigwedge_{\ell \in X} \ell(u)$$

$$\bigwedge_{\ell \in X} \ell(u) \land \text{Tr}(u, v) \Rightarrow \bigwedge_{\ell \in X} \ell(v)$$

A Maximal Inductive Subset is unique

• inductive invariants are closed under conjunction
Houdini: Algorithm Sketch

Start with a set of candidates S (the hypothesis space)

Check whether S is inductive (using some decision procedure)

• Yes: terminate
• No: there is s in S that is not preserved by the transition relation; remove s and repeat

Guarantees to find the maximal inductive subset of S
ML for Verification: The Newcomers

ICE-DT:
• Pranav Garg, Daniel Neider, P. Madhusudan, Dan Roth: Learning invariants using decision trees and implication counterexamples. POPL 2016: 499-512

Data-driven CHC

FreqHorn
• Grigory Fedyukovich, Samuel J. Kaufman, Rastislav Bodík: Sampling invariants from frequency distributions. FMCAD 2017: 100-107

HOICE
• Adrien Champion, Naoki Kobayashi, Ryosuke Sato: Holce: An ICE-Based Non-linear Horn Clause Solver. APLAS 2018

Loop invariants
• Xujie Si, Hanjun Dai, Mukund Raghothaman, Mayur Naik, Le Song: Learning Loop Invariants for Program Verification. NeurIPS 2018
LEARNING INDUCTIVE INVARIANTS
Finding an Inductive Invariant

Discovering an inductive invariants involves two steps

**Step 1**: find a candidate inductive invariant Inv

**Step 2**: check whether Inv is an inductive invariant

Invariant Inference is the process of automating both of these phases
Finding an Inductive Invariant

Two popular approaches to invariant inference:

Machine Learning based Invariant Synthesis (MLIS)
- referred to as a Black-Box approach

SAT-based Model Checking (SAT-MC)
- e.g. IC3: Aaron R. Bradley: SAT-Based Model Checking without Unrolling. VMCAI 2011: 70-87
- referred to as a White-Box approach
Our Goal

Understand the relationship between SAT-MC and MLIS

What is the fundamental difference between White-Box and Black-Box?
Our Goal

Understand the relationship between SAT-MC and MLIS

What is the fundamental difference between White-Box and Black-Box?

• Study two state-of-the-art algorithms: ICE and IC3
• In other words: can we describe IC3 as an instance of ICE?

Yakir Vizel, Arie Gurfinkel, Sharon Shoham, Sharad Malik: IC3 - Flipping the E in ICE. VMCAI 2017
Reachability Analysis

\[ R_1 = \text{post(INIT, } Tr) \]
\[ R_2 = \text{post}(R_1, Tr) \]
... \[ R_n = \text{post}(R_{n-1}, Tr) \]

INIT

Bad
Reachability Analysis

Computing states reachable from a set of states $S$ using the post operator

\[
\begin{align*}
\text{post}^0(S) &= S \\
\text{post}^{i+1}(S) &= \text{post}^i(S) \cup \{t \mid s \in S \land (s, t) \in Tr\}
\end{align*}
\]

Computing states reaching a set of states $S$ using the pre operator

\[
\begin{align*}
\text{pre}^0(S) &= S \\
\text{pre}^{i+1}(S) &= \text{pre}^i(S) \cup \{t \mid s \in S \land (t, s) \in Tr\}
\end{align*}
\]

Transitive closure is denoted by post* and pre*
Machine Learning-based Invariant Synthesis

MLIS consists of two entities: Teacher and Learner

Learner comes up with a candidate $Inv$

- Agnostic of the transition system
- Uses machine learning techniques

Learner asks the Teacher if $Inv$ is a safe inductive invariant

If not, Teacher replies with a witness: positive or negative

- Teacher knows the transition system

Referred to as Black-Box
Machine Learning-based Invariant Synthesis

Diagram:
- Teacher
- Learner
- Candidate \( \text{Inv} \)
- NO
- YES
- A witness \( S \)
Machine Learning-based Invariant Synthesis

Teacher

Learner

aware of Tr

not aware of Tr

candidate \( Inv \)

YES

NO

witness \( S \)
ICE: MLIS Framework

(Garg et al. CAV 2014)

Given a transition system \( T = (\text{INIT}, \text{Tr}, \text{Bad}) \) and a candidate \( \text{Inv} \) generated by the Learner

When the Teacher determines \( \text{Inv} \) is not a safe inductive invariant, a witness is returned:

- E-example: \( s \in \text{post}^*(\text{INIT}) \) but \( s \notin \text{Inv} \)
- C-example: \( s \in \text{pre}^*(\text{Bad}) \) and \( s \in \text{Inv} \)
- I-example: \( (s,t) \in T \) such that \( s \in \text{Inv} \) but \( t \notin \text{Inv} \)

Given a set of states \( S \), the triple \( (E, C, I) \) is an ICE state

- \( E \subseteq S \), \( C \subseteq S \), \( I \subseteq S \times S \)

A set \( J \subseteq S \) is consistent with ICE state iff

- \( E \subseteq J \) and \( J \cap C = \emptyset \)
- for \( (s,t) \in I \), if \( s \in J \) then \( t \in J \)
Inductive Invariants

System S is safe iff there exists an **inductive invariant** $\text{Inv}$:

- **Initiation**: $\text{Initial} \subseteq \text{Inv}$
- **Safety**: $\text{Inv} \cap \text{Bad} = \emptyset$
- **Consecution**: $\text{TR}(\text{Inv}) \subseteq \text{Inv}$ i.e., if $s \in \text{Inv}$ and $s \leadsto t$ then $t \in \text{Inv}$
Input: A transition system $T = (V, \text{Init}, Tr, Bad)$

$Q \leftarrow \emptyset \quad \text{LEARNER}(T) ; \text{TEACHER}(T)$;

repeat

$J \leftarrow \text{LEARNER.SYNCandidate}(Q)$;
$\varepsilon \leftarrow \text{TEACHER.IsInd}(J)$;
if $\varepsilon = \bot$ then return SAFE;

$Q \leftarrow Q \cup \{\varepsilon\}$;

until $\infty$;
ICE

(Garg et al. CAV 2014)

Input: A transition system $T = (V, A)$,

\[ Q \leftarrow \emptyset; \text{LEARNER}(T); \text{TEACHER}(T); \]

repeat

\[ J \leftarrow \text{LEARNER}.\text{SYNCANDIDATE}(Q); \]
\[ \varepsilon \leftarrow \text{TEACHER}.\text{ISIND}(J); \]

if $\varepsilon = \bot$ then return SAFE;

\[ Q \leftarrow Q \cup \{\varepsilon\}; \]

until $\infty$;

The Learner is passive - has no control over the Teacher

No requirement for incrementality

J must be consistent with $Q$
SAT/SMT-based Model Checking

Search for a counterexample for a specific length
- using Bounded Model Checking with a SAT solver

If a counterexample does not exist, generalize the bounded proof into a candidate Inv
- using interpolation with the help of a SAT solver

Check if Inv is a safe inductive invariant
- using a SAT solver, like in Houdini

Referred to as White-Box: Rely on a close interaction between the main algorithm and the decision procedure (SAT/SMT solver) used
SMT-based Model Checking
Generalizing from bounded proofs

\[
\text{counterexample of length } N \text{ exists?} \quad \text{No, } N := N + 1
\]

\[
\text{No + bounded proof} \quad \text{Yes, } N = 0
\]

\[
\text{Generalize proof} \quad \text{Yes}
\]

\[
\text{Is safe inductive invariant?} \quad \text{YES}
\]

\[
\text{SMT}
\]

\[
\text{Inv}
\]
Key IC3 Data Structure: Inductive Trace $\vec{F}$

A sequence of state formulas called frames

Properties of a trace:

- Inductive: $F_i \land Tr \rightarrow F'_{i+1}$
- Monotone: $\forall i \ F_i \rightarrow F_{i+1}$
- Safe: $\forall i \ F_i \rightarrow \neg Bad$
- Closed: $\exists i \ F_i \rightarrow \bigvee_{j=1}^{i-1} F_j$

Frame $F_i$ over-approximates states reachable in $i$ steps
PDR/IC3 – SAT Queries

Trace $[F_0,\ldots,F_N]$, and $Q \subseteq \text{pre}^*(\text{Bad})$, a state $s \in Q \cap F_{i+1}$

Strengthening

• SAT query: $\text{is SAT} \ (F_i \land \neg s) \land T \land s'$
• Checking whether $(F_i \land \neg s) \land T \rightarrow \neg s'$ is valid

If the above is satisfiable then there exists a state $t$ in $F_i$ that can reach Bad

• This looks like a C-example

In order to "fix" $F_i$ the state $t$ must be removed

Now check

• $(F_{i-1} \land \neg t) \land T \land t'$
PDR/IC3 – SAT Queries

Trace $[F_0,...,F_N]$, try to push a lemma $c \in F_i$ to $F_{i+1}$

Pushing

- $(F_i \land c) \land T \land \neg c'$
- is $(F_i \land c) \land T \rightarrow c'$ valid?

If this is satisfiable then there exists a pair $(s,t) \in T$ s.t. $s \in F_i$ and $t \notin F_{i+1}$

- It looks like an I-example
  - Also, can be either an E- or C-example

In order to ”fix” $F_i$, either $s$ is removed from $F_i$ or $t$ is added to it

- Strengthening vs Weakening
The Problem of Connecting ICE and IC3

IC3 reasons about relative induction

F is inductive relative to G when:

- \( \text{INIT} \rightarrow F, \) and
- \( G(V) \land F(V) \land T(V,V') \rightarrow F(V') \)

But, in ICE, the Learner (Teacher) asks (answers) about induction

and, the Learner in ICE is passive

- cannot control the Teacher in any way
- No guarantee for incrementality
Input: A transition system $T = (V, \text{Init}, Tr, Bad)$

$Q \leftarrow \emptyset$ ;

\textbf{Learner}(T) ; \textbf{Teacher}(T) ;$

repeat

$\langle F, G \rangle \leftarrow \text{Learner.SyncAndAndBase}(Q)$ ;

$\varepsilon \leftarrow \text{Teacher.IsRelInd}(F, G)$ ;

\textbf{if} $\varepsilon = \bot \land G = \text{true}$ \textbf{then} \textbf{return} \text{SAFE}$;$

$Q \leftarrow Q \cup \{\varepsilon\}$ ;

until $\infty$ ;

\textbf{G} allows the \textbf{Learner} to have some control over the \textbf{Teacher}$.$

\textbf{When G} is true it is a regular inductive check$.$
RICE – ICE + Relative Induction

The Teacher in RICE reacts to queries about relative induction

The Learner can “manipulate” the Teacher using relative induction

RICE is a generalization of ICE where the Learner is an active learning algorithm
RICE – ICE + Relative Induction

The Teacher in RICE reacts to queries about relative induction

Is F inductive relative to G?

If not, a witness is returned:

• E-example: $s \in \text{post}^*(\text{INIT})$ but $s \notin F$
• C-example: $s \in \text{pre}^*(\text{Bad})$ and $s \in F$

• I-example: $(s,t) \in T$ such that $s \in F \land G$ but $t \notin F$
IC3 AS AN INSTANCE OF RICE
IC3 Learner

The IC3 Learner is active and incremental

Maintains the following:
- a trace \([F_0, \ldots, F_N]\) of candidates
- RICE state \(Q=(E, C, I)\)

The Learner must be consistent with the RICE state

E-examples and C-examples may exist when F is inductive relative to G
- The Teacher may return an E-example or C-example when F is inductive relative to G
IC3 Learner - Strengthening

Strengthening:

• a C-example s in F_i
• \((F_i \land \neg s \land \neg C(Q)) \land T \land (s \lor C(Q))'\)

INIT \rightarrow F, and
\[ G(V) \land F(V) \land T(V,V') \rightarrow F(V') \]

**E-example:** a cex exists

**C-example:** add to Q

**I-example:** treat like C-example
IC3 Learner - Pushing

Pushing:
- a lemma \( c \) in \( F_i \)
- \( (F_i \land c \land \neg C(Q) \land F_{i+1}) \land T \land (\neg c \lor C(Q) \lor \neg F_{i+1})' \)

is \( (c \land \neg C(Q) \land F_{i+1}) \)
inductive relative to 
\( F_i \) ?

- E-example: do not push and add to \( Q \)
- C-example: do not push and add to \( Q \)
- I-example: do not push and add to \( Q \)

\[ \text{INIT} \rightarrow F, \text{ and} \]
\[ G(V) \land F(V) \land T(V,V') \rightarrow F(V') \]
Pushing:
- a lemma $c$ in $F_i$
- $(F_i \land c \land \neg C(Q) \land F_{i+1}) \land T \land (\neg c \lor C(Q) \lor \neg F_{i+1})'$

**E-example:** do not push and add to $Q$

**C-example:** do not push and add to $Q$

**I-example:** do not push and add to $Q$

E- and C-examples may exist even when relative induction holds.
IC3 Teacher

Using a general Teacher, the described Learner computes a trace \([F_0, \ldots, F_N]\) such that

- \(\text{post}^*(\text{INIT}) \rightarrow F_i \rightarrow \neg\text{pre}^*(\text{Bad})\)

General Teacher is infeasible

- required to look arbitrary far into the future (for E-examples)
- required to look arbitrary far into the past (for C-examples)

Solution: add restrictions on E- and C-examples
IC3 Teacher

Is F inductive relative to G?

If not, a witness is returned:

- C-example: $s \in \text{pre}^m(\text{Bad})$ and $s \in F$
- I-example: $(s,t) \in T$ such that $s \in F \land G$ but $t \notin F$
- E-example: $s \in \text{post}^0(\text{INIT})$ but $s \notin F$

Claim: Using this IC3 Teacher and the IC3 Learner results in an algorithm that behaves like (simulates) IC3
What Can We Learn?

Can we lift the restriction that requires E-example to be in INIT only?

• Yes, a variant of IC3, called Quip, does that

There is no “real” weakening mechanism in IC3

• (Not) Pushing is a form of weakening
• But no ‘active’ weakening of candidates
• IC3 is incremental and never restarts

RICE – a fundamentally different framework for MLIS

• exponentially more effective learning (Y. Feldman et al.)
Conclusions

Program analysis is a difficult (undecidable) problem
  • many more solutions/techniques are needed!

Program Analysis is well suited for ML-based solutions
  • Rich space of heuristics
  • Easy definition of ‘ground truth’

But much better benchmarks / data sets are needed!
  • existing benchmarks are not well suited for empirical research

Is program analysis harder / different than image recognition?
  • 5 year olds are amazingly good at recognizing animals
  • Not so good at distinguishing good and bad programs
  • (are experts really that much better?)
Puppy?