Machine Learning and Invariant Synthesis

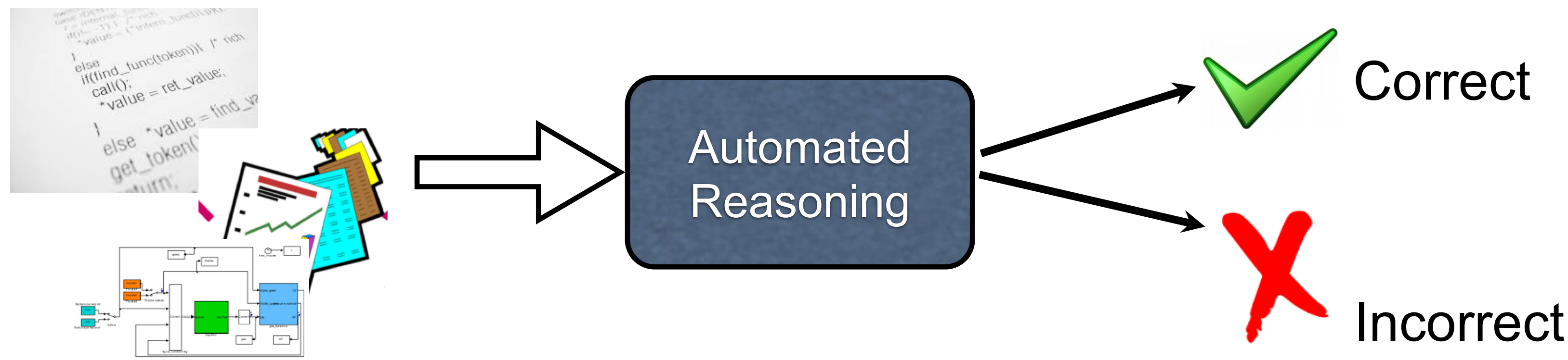
Arie Gurfinkel
Dept. of Electrical and Computer Engineering
University of Waterloo

Waterloo ML + Security + Verification Workshop
August 26 – 30, 2019
Waterloo, Ontario, Canada

UNIVERSITY OF WATERLOO

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?

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, Init, $\mathcal{T}r$, Bad)

P is UNSAFE if and only if there exists a number *N* s.t.

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

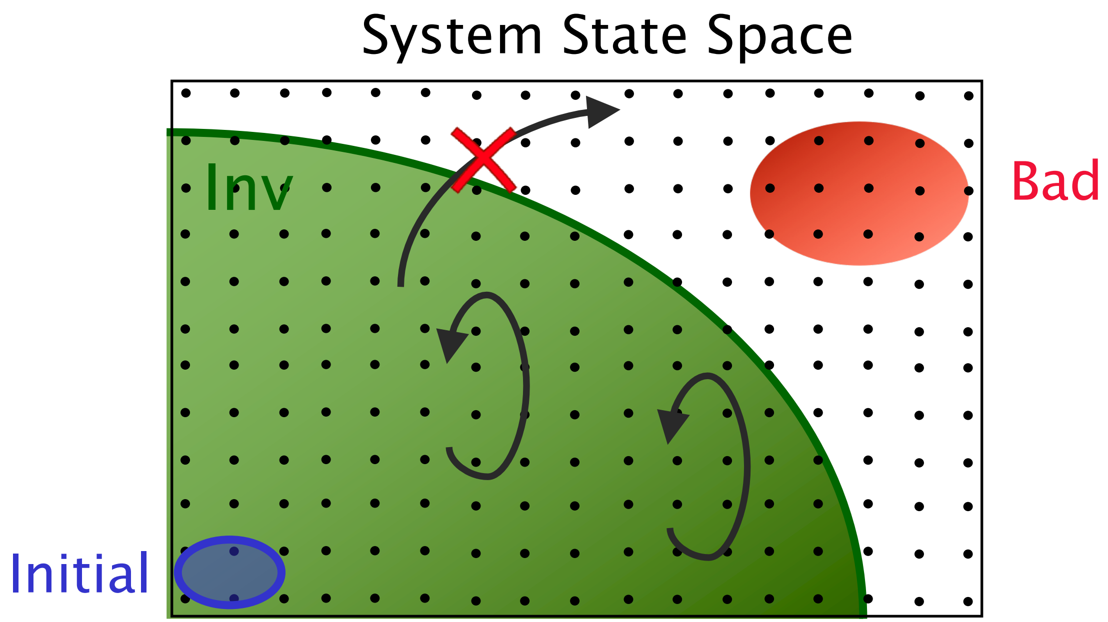
P is SAFE if and only if there exists a *safe inductive invariant Inv s.t.*

$$\left.\begin{array}{r} Init \Rightarrow Inv \\ Inv(X) \wedge Tr(X, X') \Rightarrow Inv(X') \end{array}\right\} \boxed{\text{Inductive}}$$

$$Inv \Rightarrow \neg Bad \quad \boxed{\text{Safe}}$$

UNIVERSITY OF WATERLOO

Inductive Invariants

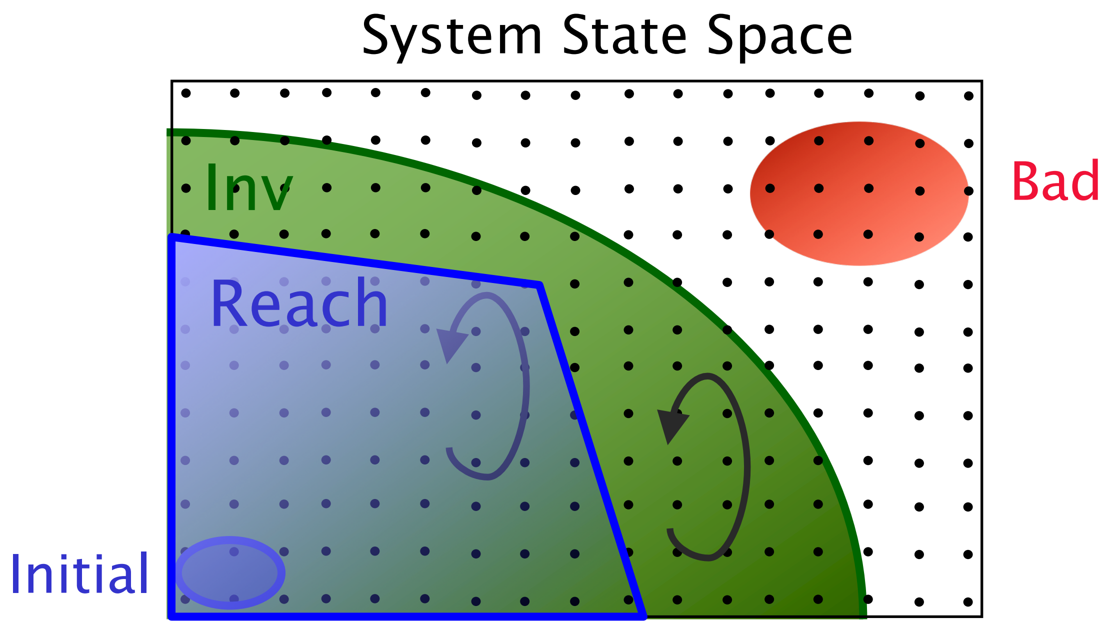


System S is safe iff there exists an **inductive invariant** Inv:

- Initiation: Initial ⊆ Inv
- Safety: Inv ∩ Bad = Ø
- Consecution: TR(Inv) ⊆ Inv

i.e., if s ∈ Inv and s↝t then t ∈ Inv

Inductive Invariants



System S is safe iff there exists an **inductive invariant** Inv:

- Initiation: Initial ⊆ Inv
- Safety: Inv ∩ Bad = ∅
- Consecution: TR(Inv) ⊆ Inv

i.e., if s ∈ Inv and s↝t then t ∈ Inv

System S is **safe** if Reach ∩ Bad = ∅

Program Verification with HORN(LIA)

```
z = x; i = 0;
assume (y > 0);
while (i < y) {
  z = z + 1;
  i = i + 1;
}
assert(z == x + y);
```

IS SAT?

```
z = x & i = 0 & y > 0                          ➔  Inv(x, y, z, i)
Inv(x, y, z, i) & i < y & z1=z+1 & i1=i+1      ➔  Inv(x, y, z1, i1)
Inv(x, y, z, i) & i >= y & z != x+y            ➔  false
```

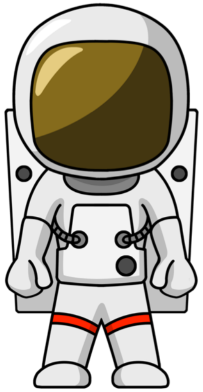
In SMT-LIB

```
(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)
```

```
$ z3 add-by-one.smt2
sat
(model
  (define-fun Inv ((x!0 Int) (x!1 Int) (x!2 Int) (x!3 Int)) Bool
    (and (<= (+ x!2 (* (- 1) x!0) (* (- 1) x!3)) 0)
         (<= (+ x!2 (* (- 1) x!0) (* (- 1) x!1)) 0)
         (<= (+ x!0 x!3 (* (- 1) x!2)) 0)))
)
```

```
Inv(x, y, z, i)
     z  = x + i
     z <= x + y
```

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 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

UNIVERSITY OF WATERLOO

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:

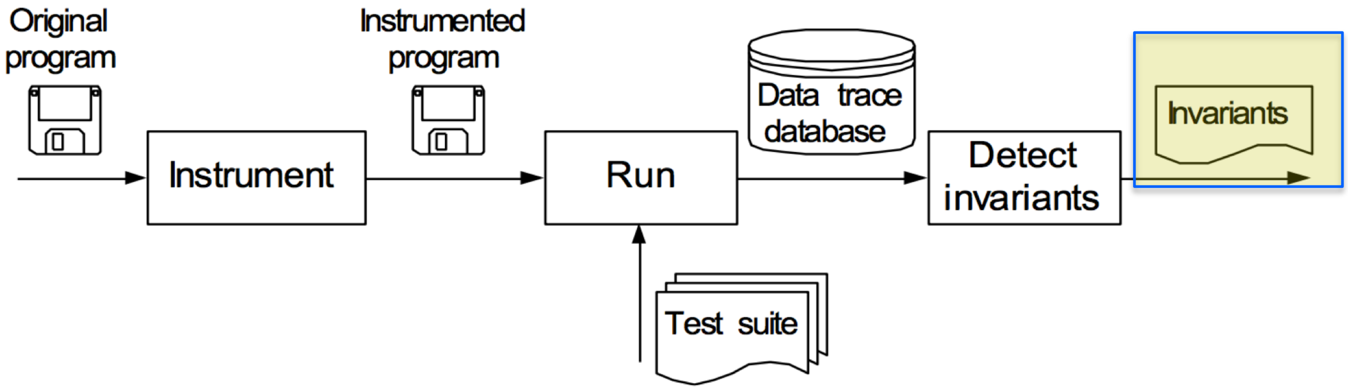
Daikon

- 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 >= 0
2.) s = SUM(B)
3.) $0 \leq i \leq n$

```
15.1.1:::ENTER
B = 92 56 -96 -49 76 92 -3 -88, modified
N = 8, modified

15.1.1:::LOOP
B = 92 56 -96 -49 76 92 -3 -88, modified
N = 8, modified
I = 0, modified
s = 0, modified

15.1.1:::LOOP
B = 92 56 -96 -49 76 92 -3 -88, unmodified
N = 8, unmodified
I = 1, modified
S = 92, modified
```

UNIVERSITY OF WATERLOO

Houdini: Maximal Inductive Subset

Let *L* be a set of formulas, *P=(V, Init, Tr, Bad)* a program

A subset *X* of *L* is a *maximal inductive subset* iff it is the largest subset of *X* such that

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

$$\wedge_{\ell \in X} \ell(u) \wedge Tr(u, v) \Rightarrow \wedge_{\ell \in X} \ell(v)$$

A Maximal Inductive Subset is unique

- inductive invariants are closed under conjunction

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

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

- He Zhu, Stephen Magill, Suresh Jagannathan:
  A data-driven CHC solver. PLDI 2018: 707-721

FreqHorn

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

HOICE

- Adrien Champion, Naoki Kobayashi, Ryosuke Sato:
  HoIce: 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**)

- e.g. ICE: Pranav Garg, Christof Löding, P. Madhusudan, Daniel Neider: ICE: A Robust Framework for Learning Invariants. CAV 2014: 69-87
- 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

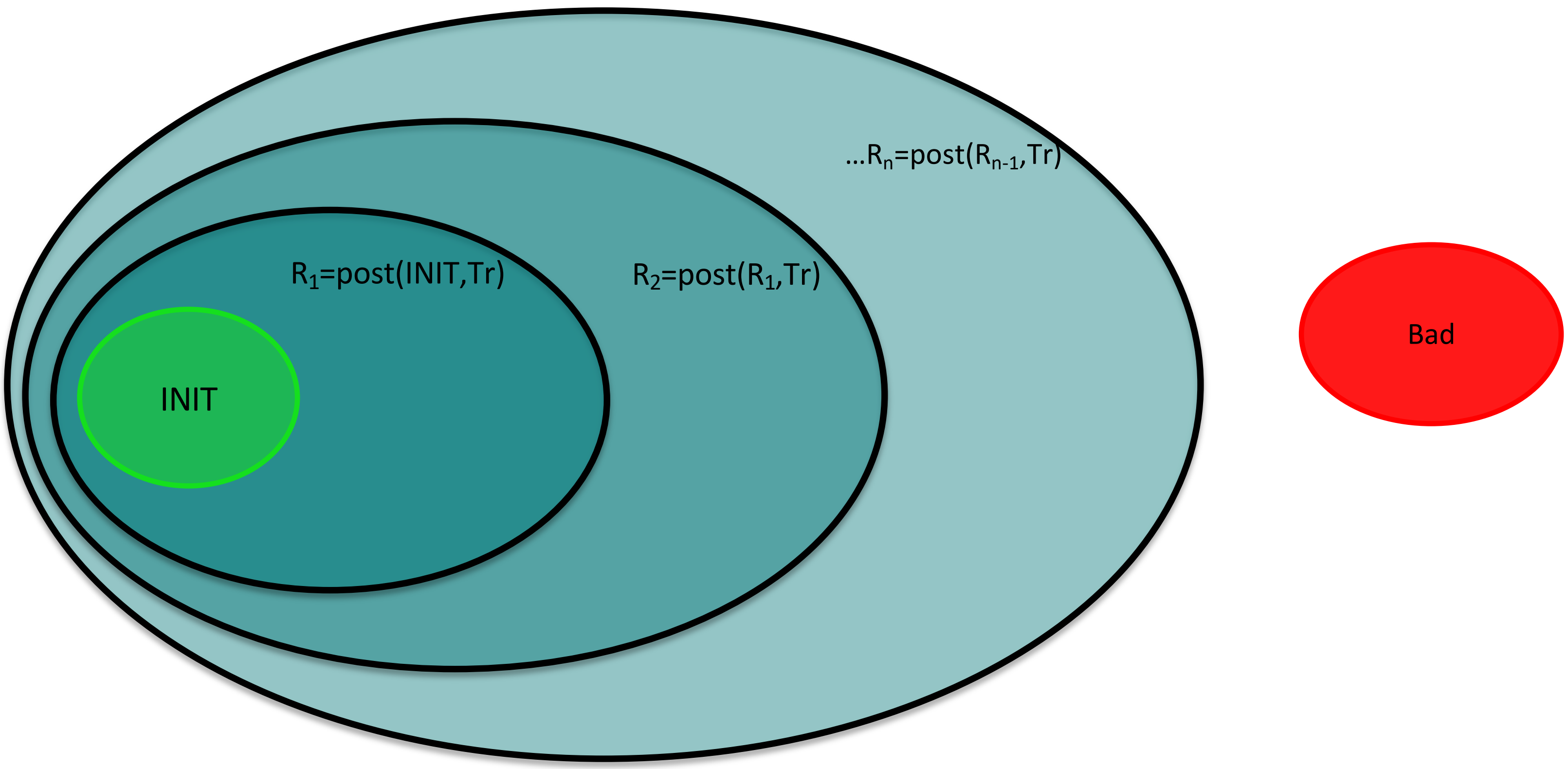
Understand the relationship between SAT-MC and MLIS

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



Reachability Analysis

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

$$\begin{cases} post^0(S) = S \\ post^{i+1} = post^i(S) \cup \{t \mid s \in S \wedge (s,t) \in Tr\} \end{cases}$$

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

$$\begin{cases} pre^0(S) = S \\ pre^{i+1} = pre^i(S) \cup \{t \mid s \in S \wedge (t,s) \in Tr\} \end{cases}$$

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

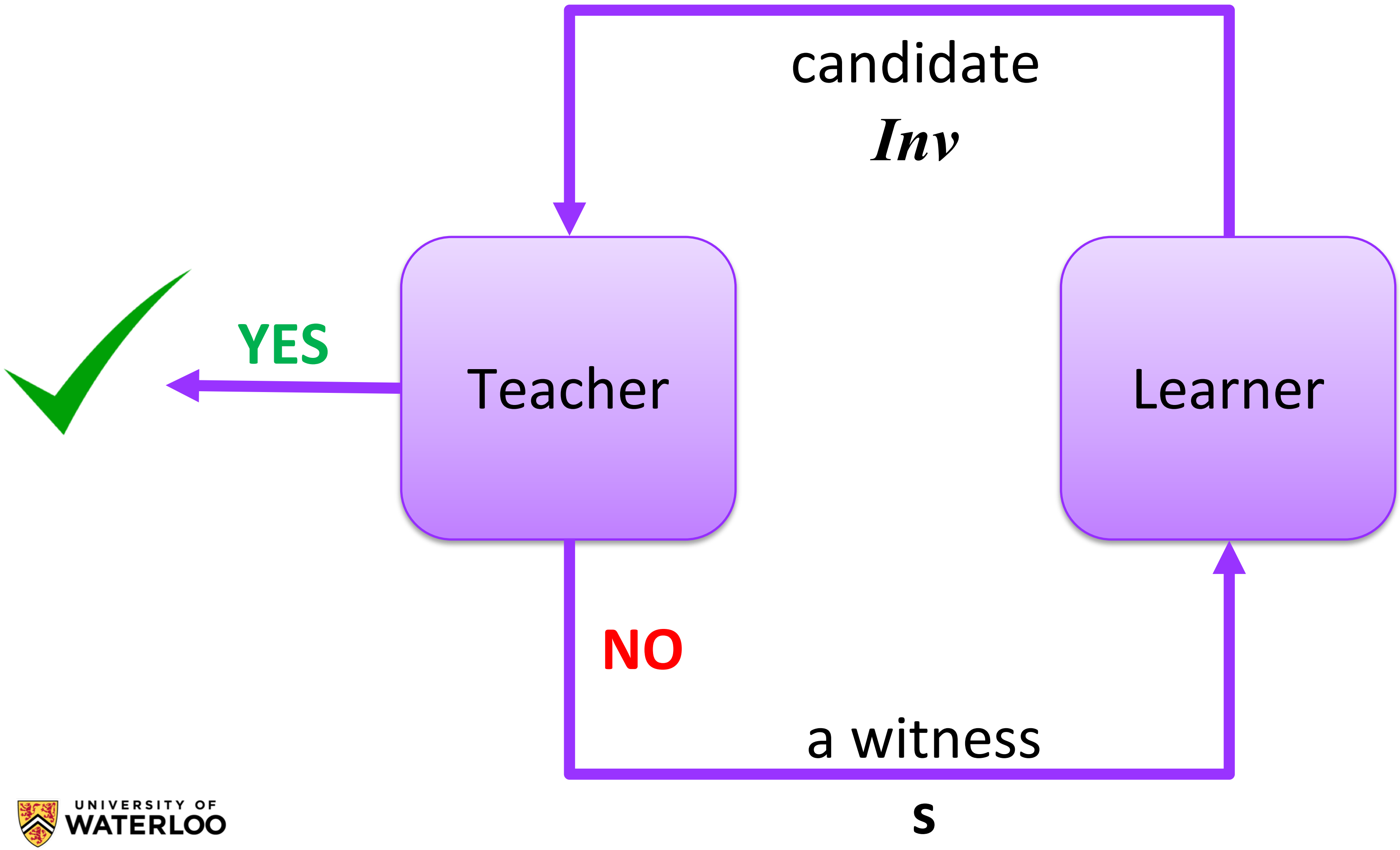
Learner asks the Teacher if *Inv* is a safe inductive invariant

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

- Teacher knows the transition system

Referred to as Black-Box

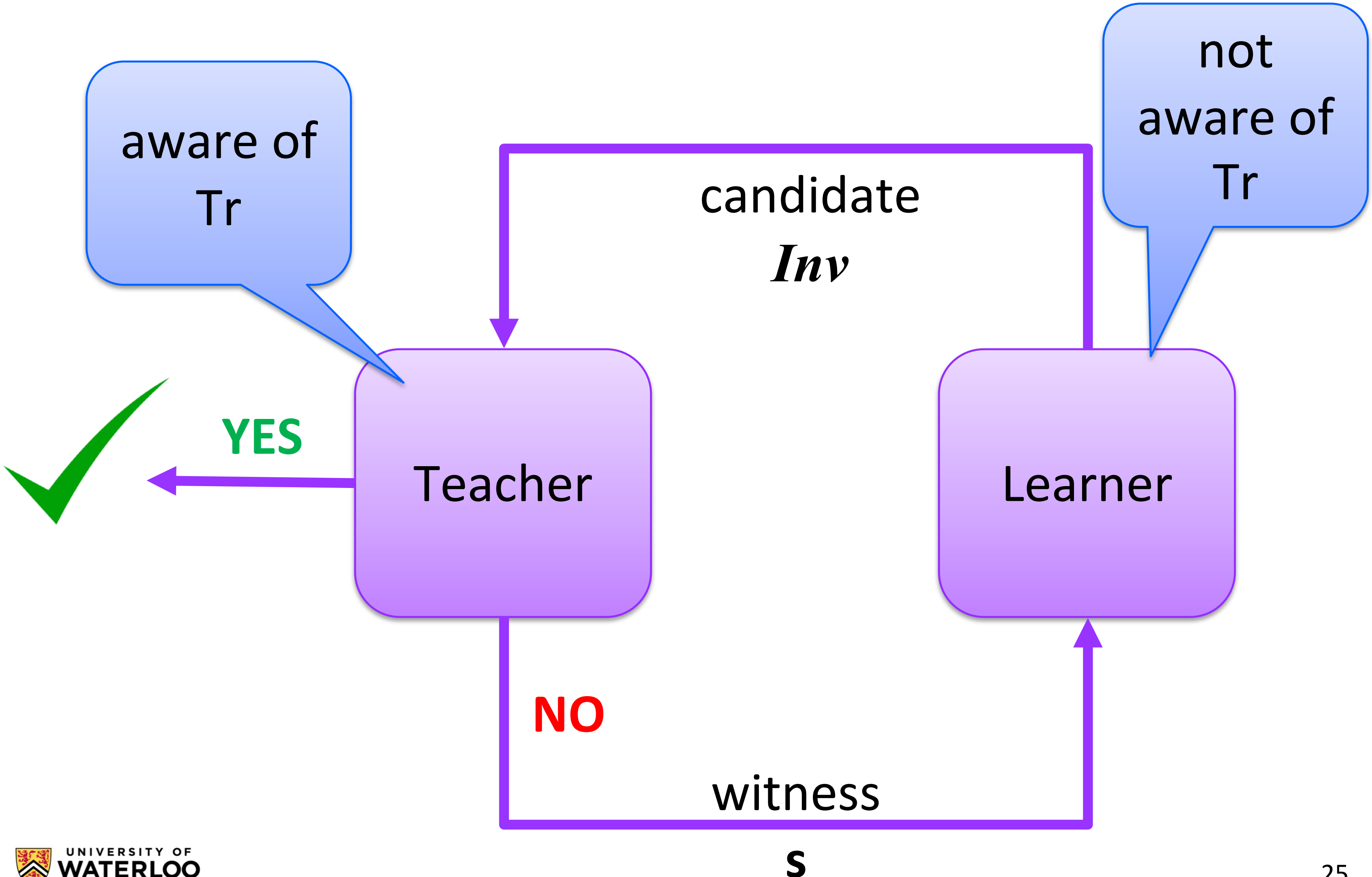
Machine Learning-based Invariant Synthesis



UNIVERSITY OF WATERLOO

5

Machine Learning-based Invariant Synthesis



ICE: MLIS Framework

(Garg et al. CAV 2014)

Given a transition system T=(INIT, Tr, Bad) and a candidate *Inv* generated by the Learner

When the Teacher determines *Inv* is not a safe inductive invariant, a witness is returned:

- E-example: s ∈ post*(INIT) but s ∉ Inv
- C-example: s ∈ pre*(Bad) and s ∈ Inv
- I-example: (s,t) ∈ T such that s ∈ Inv but t ∉ Inv

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

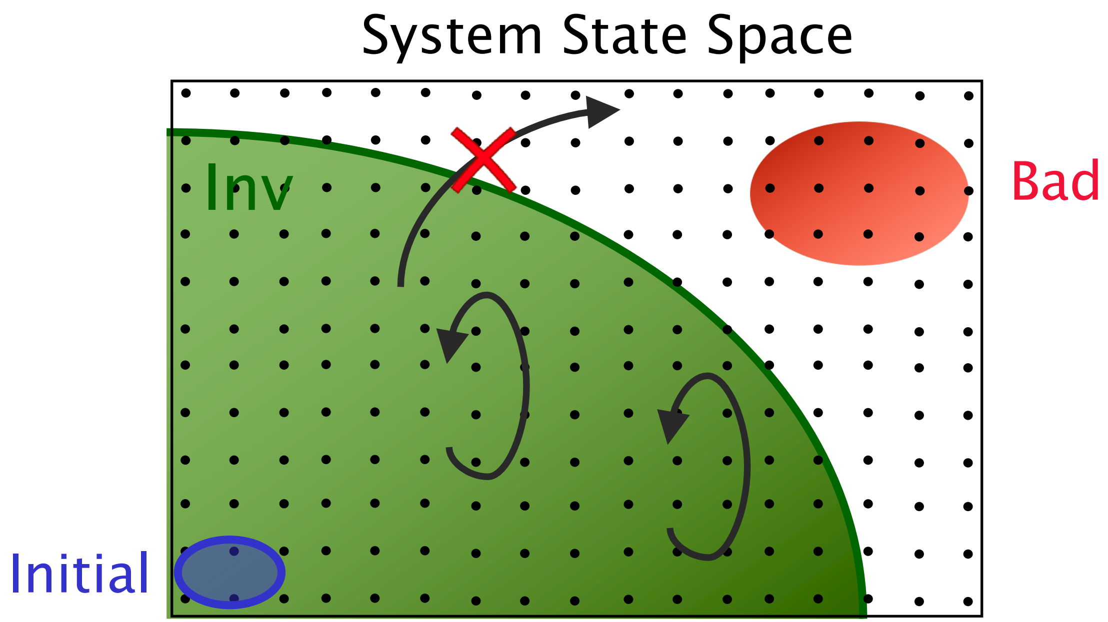
- E ⊆ S, C ⊆ S, I ⊆ S × S

A set J ⊆ S is **consistent** with ICE state iff

- E ⊆ J and J ∩ C = ∅
- for (s,t) ∈ I, if s ∈ J then t ∈ J

UNIVERSITY OF WATERLOO

Inductive Invariants



System S is safe iff there exists an **inductive invariant** Inv:

- Initiation: Initial ⊆ Inv
- Safety: Inv ∩ Bad = Ø
- Consecution: TR(Inv) ⊆ Inv

i.e., if s ∈ Inv and s↝t then t ∈ Inv

UNIVERSITY OF WATERLOO

ICE

(Garg et al. CAV 2014)

Input: A transition system $T = (\mathcal{V}, Init, Tr, Bad)$

$Q \leftarrow \emptyset$ $\textsc{Learner}(T)$; $\textsc{Teacher}(T)$;

repeat

$\quad J \leftarrow \textsc{Learner.SynCandidate}(Q)$;

$\quad \varepsilon \leftarrow \textsc{Teacher.IsInd}(J)$;

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

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

until ∞;

ICE

(Garg et al. CAV 2014)

Input: A transition system $T = (\mathcal{V},$

$Q \leftarrow \emptyset$ $\textsc{Learner}(T)$; $\textsc{Teacher}(T)$;

repeat

$\quad J \leftarrow \textsc{Learner.SynCandidate}(Q)$;

$\quad \varepsilon \leftarrow \textsc{Teacher.IsInd}(J)$;

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

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

until ∞;

No requirement for incrementality

J must be consistent with Q

The Learner is passive - has no control over the Teacher

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

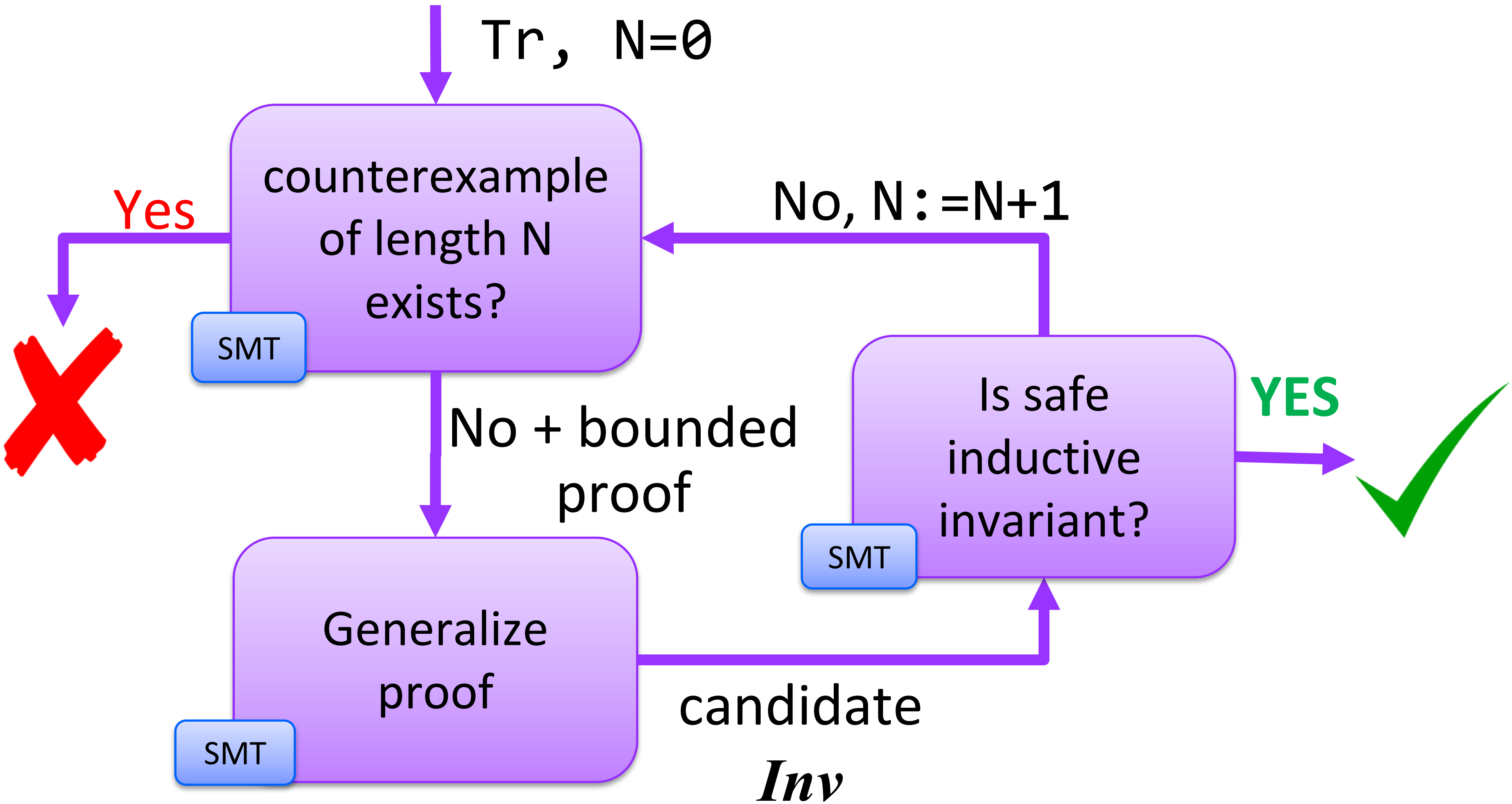
Check if Inv is a safe inductive invariant

- 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



Key IC3 Data Structure: Inductive Trace $\vec{F}$

A sequence of state formulas called frames

$Init$	F_1	....	F_{N-1}	F_N

Properties of a trace:

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

Frame F_i over-approximates states reachable in i steps

3

PDR/IC3 – SAT Queries

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

Strengthening

- SAT query: is SAT $(F_i \wedge \neg s) \wedge T \wedge s'$
- Checking whether $(F_i \wedge \neg s) \wedge T \rightarrow \neg s'$ 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

- $(F_{i-1} \wedge \neg t) \wedge T \wedge t'$

UNIVERSITY OF WATERLOO

PDR/IC3 – SAT Queries

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

Pushing

- $(F_i \wedge c) \wedge T \wedge \neg c'$
- is $(F_i \wedge c) \wedge T \rightarrow c'$ 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:

- INIT $\longrightarrow$ F, and
- $G(V) \land F(V) \land T(V,V') \longrightarrow F(V')$

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

6

RICE – ICE + Relative Induction

Input: A transition system $T = (\mathcal{V}, \mathit{Init}, \mathit{Tr}, \mathit{Bad})$

$Q \leftarrow \emptyset$;

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

repeat

$\quad (F, G) \leftarrow \textsc{Learner.SynCandAndBase}(Q)$;

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

$\quad$ **if** $\varepsilon = \bot \wedge G = \mathit{true}$ **then return** SAFE;

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

until ∞;

G allows the Learner to have some control over the Teacher

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 ∈ post*(INIT) but $s \notin F$
- C-example: s ∈ pre*(Bad) and $s \in F$

- I-example: (s,t) ∈ T such that $s \in F \land G$ but $t \notin F$

IC3 AS AN INSTANCE OF RICE

0

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

UNIVERSITY OF WATERLOO

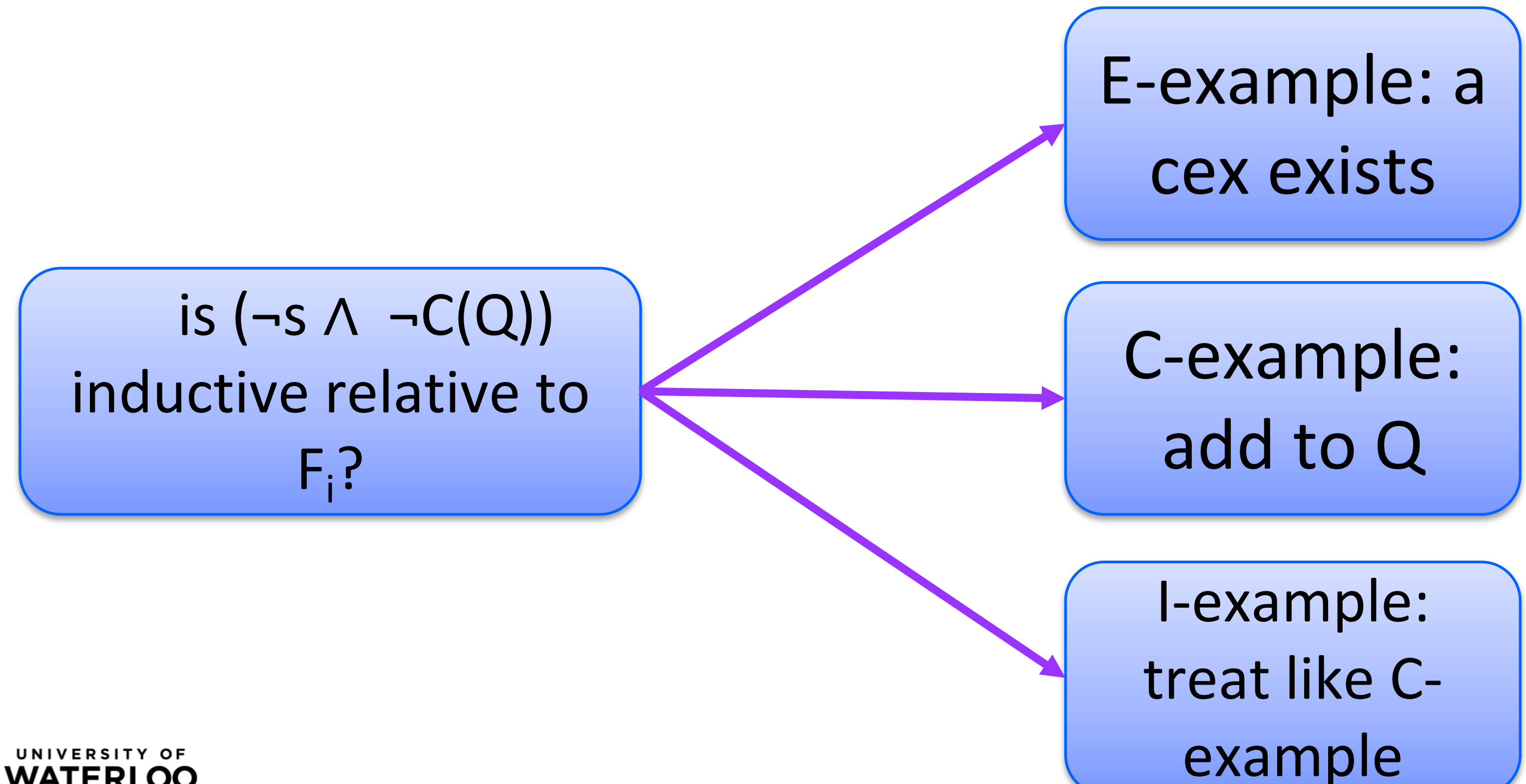
1

IC3 Learner - Strengthening

INIT $\longrightarrow$ F, and
$G(V) \wedge F(V) \wedge T(V,V') \longrightarrow F(V')$

Strengthening:

- a C-example s in F_i
- $(F_i \wedge \neg s \wedge \neg C(Q)) \wedge T \wedge (s \vee C(Q))'$



UNIVERSITY OF WATERLOO

2

IC3 Learner - Pushing

INIT → F, and
$G(V) \wedge F(V) \wedge T(V,V') \rightarrow F(V')$

Pushing:

- a lemma c in F_i
- $(F_i \wedge c \wedge \neg C(Q) \wedge F_{i+1}) \wedge T \wedge (\neg c \vee C(Q) \vee \neg F_{i+1})'$

is $(c \wedge \neg C(Q) \wedge F_{i+1})$ 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

UNIVERSITY OF WATERLOO

3

IC3 Learner - Pushing

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

Pushing:

- a lemma c in F_i
- $(F_i \wedge c \wedge \neg C(Q) \wedge F_{i+1}) \wedge T \wedge (\neg c \vee C(Q) \vee \neg F_{i+1})'$

is $(c \wedge \neg C(Q) \wedge F_{i+1})$ 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

IC3 Teacher

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

- post*(INIT) → F_i → ¬pre*(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 ∈ pre^m(Bad) and s ∈ F
- I-example: (s,t) ∈ T such that s ∈ $F \wedge G$ but t ∉ F
- E-example: s ∈ $post^0$(INIT) but s ∉ 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/technqiues 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?

UNIVERSITY OF WATERLOO