Decision Tree Based Learning of Program Invariants

Deepak D'Souza

Department of Computer Science and Automation
Indian Institute of Science, Bangalore.

FM Update Meeting
IIT Mandi
17 July 2017

What this talk is about

Paper titled

> *Learning invariants using decision trees and implication counterexamples,*

by Garg, Neider, Madhusudan, and Roth, in POPL 2016.

- A way to automate deductive-style program verification.
- Extends the Decision Tree classification technique in Machine Learning, to handle implication samples, with applications to finding proofs of programs.

Also talk about some directions to extend this work.

Outline of this talk

Proving assertions in programs

```
// Pre: 10 <= y

y := y + 1;
z := x + y;

// Post: x <= z
```

```
// Pre: true

if (a <= b)
  min = a;
else
  min = b;

// Post: min <= a && min <= b
```

```
// Pre: 0 <= n

int a = m;
int x = 0;
while (x < n) {
  a = a + 1;
  x = x + 1;
}

// Post: a = m + n
```

Proving assertions in programs

```
// Pre: 10 <= y

y := y + 1;
z := x + y;

// Post: x <= z
```

```
// Pre: true

if (a <= b)
  min = a;
else
  min = b;

// Post: min <= a && min <= b
```

```
// Pre: 0 <= n

int a = m;
int x = 0;
while (x < n) {
  a = a + 1;
  x = x + 1;
}

// Post: a = m + n
```

Model-checking vs Deductive Reasoning.

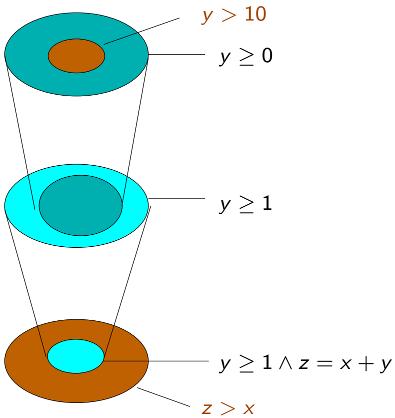
Floyd-Hoare Style of Program Verification

Robert W. Floyd: "Assigning meanings to programs" *Proceedings of the American Mathematical Society Symposia on Applied Mathematics* (1967)

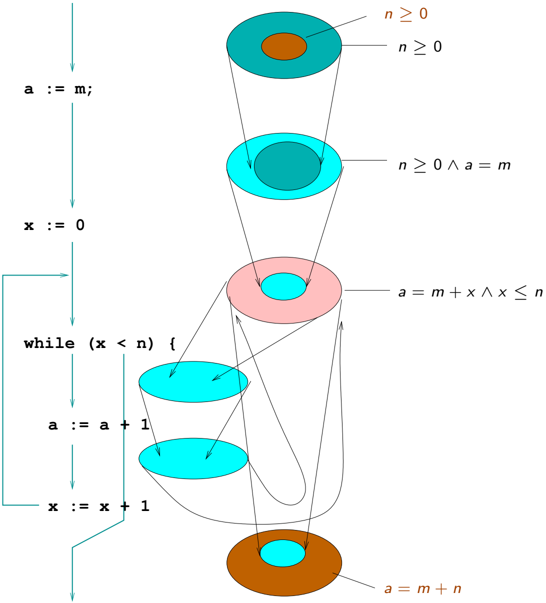
C A R Hoare: "An axiomatic basis for computer programming", *Communications of the ACM* (1969).

Example proof

```
y := y + 1

z := x + y
```



Example proof of `add` program



Problems with automating such proofs

To check:

$\{y > 10\}$

```
y := y + 1;
z := x + y;
```

$\{x < z\}$

Use the weakest precondition rules to generate the verification condition:

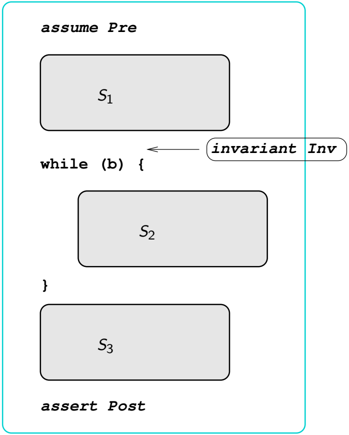
$$(y > 10) \implies (y > -1).$$

Check the verification condition by asking a theorem prover / SMT solver if the formula

$$(y > 10) \wedge \neg(y > -1).$$

is satisfiable.

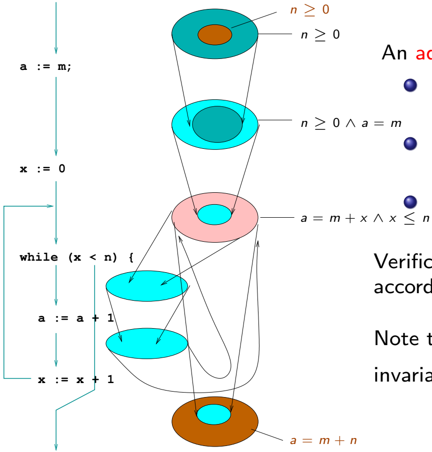
What about Programs with loops?



Find an adequate and inductive invariant *Inv*:

1. $Pre \implies WP(S_1, Inv)$ ("inductive invariant")
2. $(Inv \wedge b) \implies WP(S_2, Inv)$ ("inductive invariant")
3. $Inv \wedge \neg b \implies WP(S_3, Post)$ ("adequate").

Adequate loop invariant



An adequate loop invariant needs to satisfy:

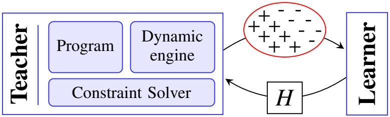
- $\{n \geq 0\}$ `a := m; x := 0` $\{a = m + x \wedge x \leq n\}$.
- $\{a = m + x \wedge x \leq n \wedge x < n\}$ `a := a+1; x := x+1` $\{a = m + x \wedge x \leq n\}$.
- $\{a = m + x \wedge x \leq n \wedge x \geq n\}$ `skip` $\{a = m + n\}$.

Verification conditions are generated accordingly.

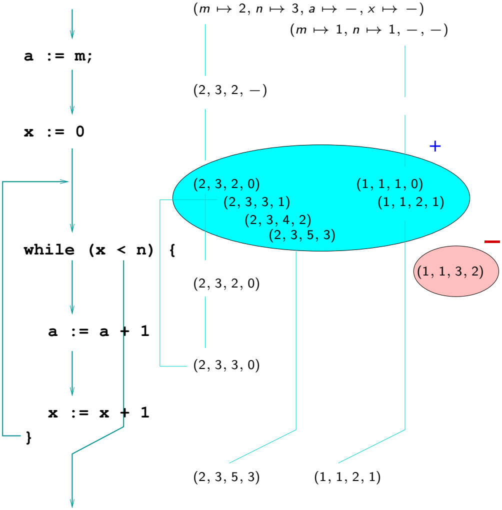
Note that $a = m + x$ is not an adequate loop invariant.

Learning loop invariants

- Main hurdle in automating program verification is coming up with adequate loop invariants.
- Several white-box approaches have been used (CEGAR, Lazy Annotation, using interpolation, and tools like Slam/Blast, Synergy).
- Instead explore a black-box approach, based on a Teacher-Learner model.



Black-box Learning for `add` program



Decision Tree Based Learning

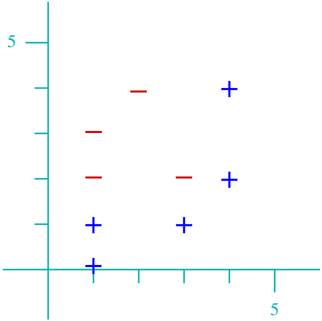
- Given a set of positive samples S^+ and negative samples S^-, learn a predicate H from a given concept class.
- Example concept class: Boolean combinations of atomic predicates of the form $x \leq c$, where x is a prog variable and $c \leq 10$.
- Or octagonal constraints $^\pm x \; ^\pm y \leq c$...
- A brute-force search is always possible, but we would like to be more efficient in practice.

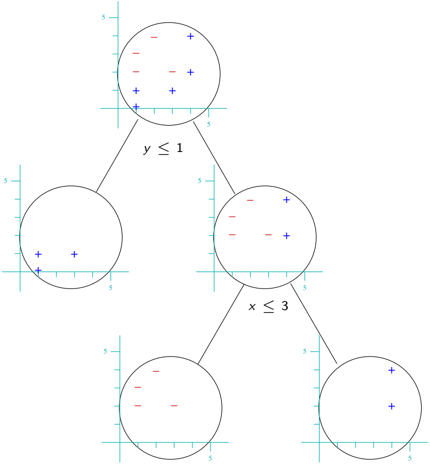
Decision Tree learning algorithm

Maintain a tree whose nodes correspond to subsets of the sample points

- Root node contains all given samples
- Choose a non-finished node n, and an attribute a to split on.
- Create two children n_a and $n_{\neg a}$ of n with corresponding subset of samples.
- If a node is "homogeneous", mark it pos/neg and finished.
- Recurse till all nodes are finished.
- Output predicate corresponding to disjunction of all positive nodes.

Decision Tree learning by example





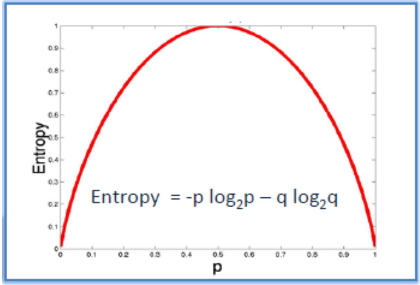
Predicate learnt: $y \leq 1 \vee (y > 1 \wedge x > 3)$.

Choosing attribute based on entropy

If n has P positive and N negative samples:

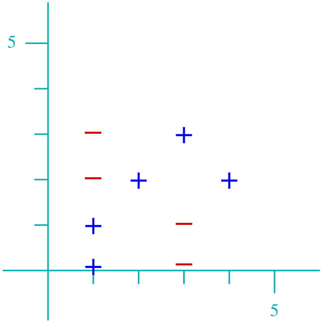
$$Entropy(n) = -\frac{P}{P+N} \cdot \log \frac{P+N}{N} - \frac{N}{P+N} \cdot \log \frac{P+N}{P}$$

- Entropy measures reduction in uncertainty in number of bits.
- Gives us a measure of the "impurity" of a node.
- Choose attribute a which maximizes $Entropy(n) - (Entropy(n_a) + Entropy(n_{\neg a}))$.



Entropy = -0.5 log$_2$0.5 – 0.5 log$_2$0.5 = 1

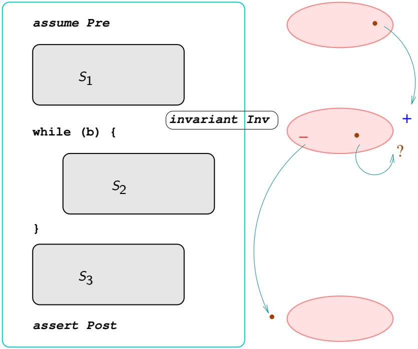
Decision Tree: Example where entropy does not do well



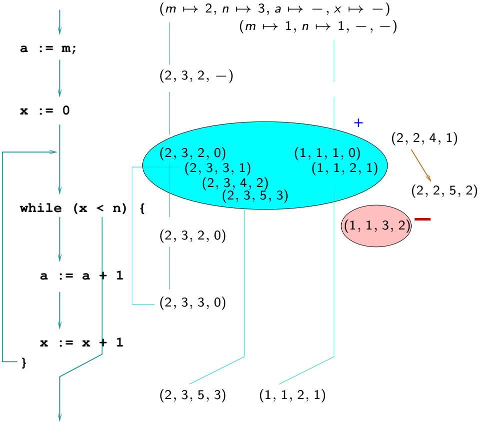
Best attribute would be $y \leq 1$ followed by $x \leq 1$, but entropy would choose $x \leq 3$ as first split.

ICE: The need for implication counterexamples

- Introduced by Garg, Löding, Madhusudan, and Neider, in a paper in CAV 2014.
- Just Examples (positive) and Counterexamples (negative) are not enough: the Teacher needs to give Implication samples as well.
- This way the Teacher is honest, not precluding some candidate invariant by an arbitrary answer.
- Leads to a robust learning framework.

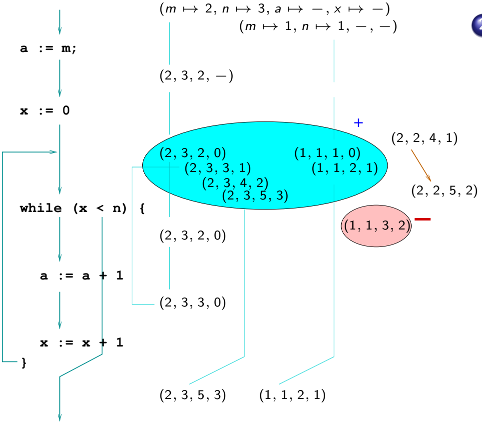


ICE learning by example



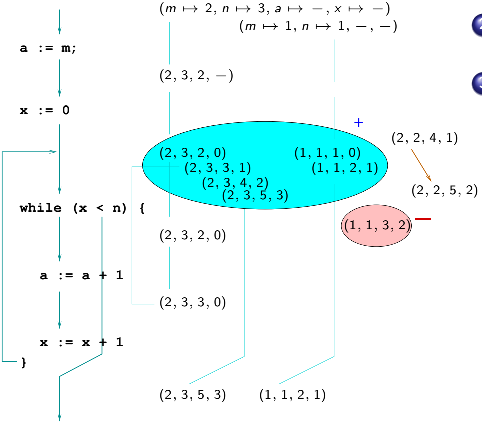
1. Learner conjectures H: $m \leq n \land x \leq a$

ICE learning by example



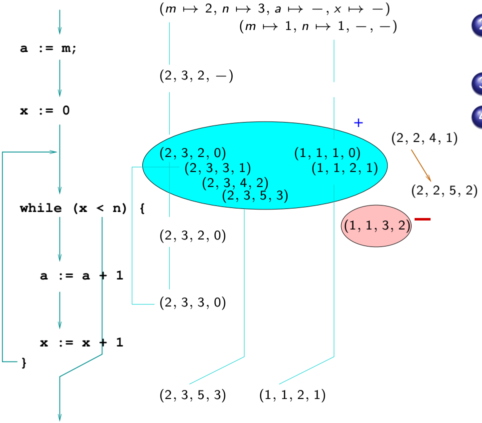
1. Learner conjectures H: $m \leq n \wedge x \leq a$
2. Teacher replies with Example: $(2, 1, 2, 0)$.

ICE learning by example



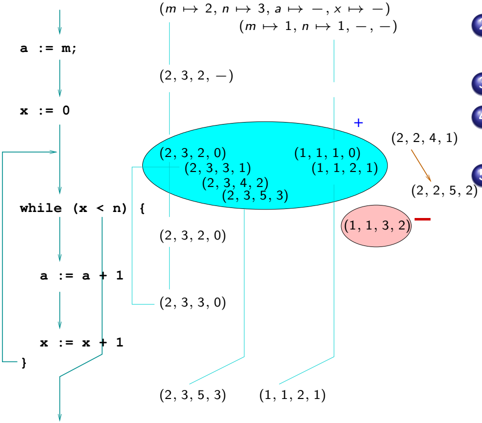
1. Learner conjectures H: $m \leq n \wedge x \leq a$
2. Teacher replies with Example: $(2, 1, 2, 0)$.
3. Learner conjectures: $a \leq m + n$

ICE learning by example



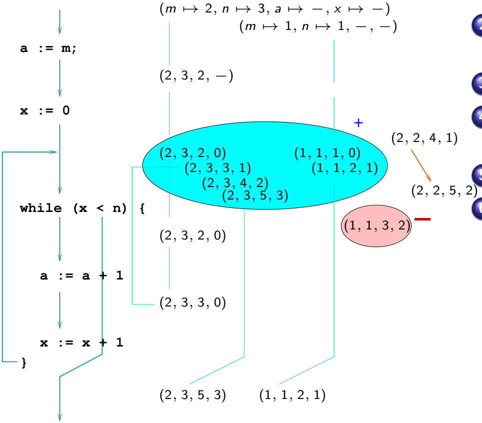
1. Learner conjectures H: $m \leq n \wedge x \leq a$
2. Teacher replies with Example: $(2, 1, 2, 0)$.
3. Learner conjectures: $a \leq m + n$
4. Teacher replies with Implication: $(2, 2, 4, 1) \implies (2, 2, 5, 2)$.

ICE learning by example



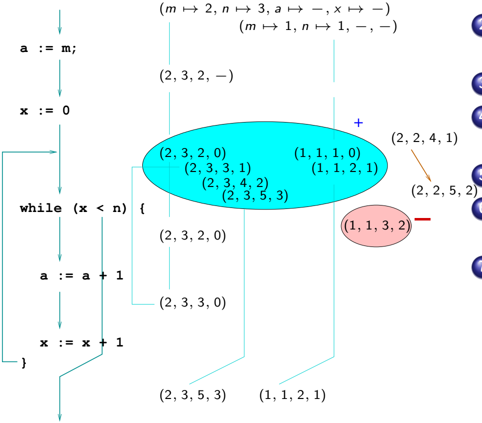
1. Learner conjectures H:
   $m \leq n \wedge x \leq a$
2. Teacher replies with Example:
   $(2, 1, 2, 0)$.
3. Learner conjectures: $a \leq m + n$
4. Teacher replies with Implication:
   $(2, 2, 4, 1) \implies (2, 2, 5, 2)$.
5. Learner conjectures: $a = m + x$

ICE learning by example



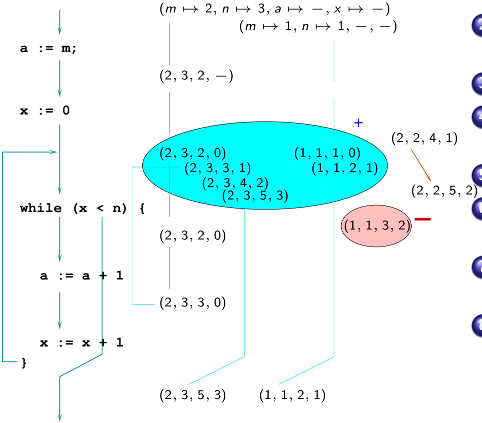
1. Learner conjectures H: $m \leq n \wedge x \leq a$
2. Teacher replies with Example: $(2, 1, 2, 0)$.
3. Learner conjectures: $a \leq m + n$
4. Teacher replies with Implication: $(2, 2, 4, 1) \implies (2, 2, 5, 2)$.
5. Learner conjectures: $a = m + x$
6. Teacher replies with Counterexample: $(1, 1, 3, 2)$

ICE learning by example



1. Learner conjectures H: $m \leq n \wedge x \leq a$
2. Teacher replies with Example: $(2, 1, 2, 0)$.
3. Learner conjectures: $a \leq m + n$
4. Teacher replies with Implication: $(2, 2, 4, 1) \implies (2, 2, 5, 2)$.
5. Learner conjectures: $a = m + x$
6. Teacher replies with Counterexample: $(1, 1, 3, 2)$
7. Learner conjectures: $a = m + x \wedge x \leq n$

ICE learning by example



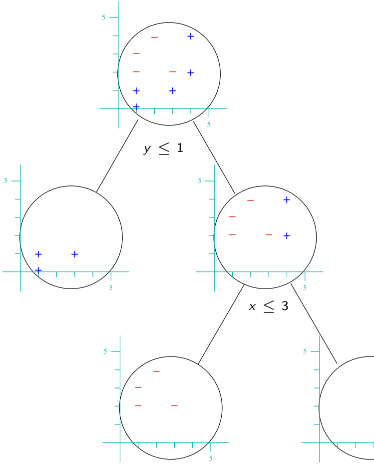
1. Learner conjectures H: $m \leq n \wedge x \leq a$
2. Teacher replies with Example: $(2, 1, 2, 0)$.
3. Learner conjectures: $a \leq m + n$
4. Teacher replies with Implication: $(2, 2, 4, 1) \implies (2, 2, 5, 2)$.
5. Learner conjectures: $a = m + x$
6. Teacher replies with Counterexample: $(1, 1, 3, 2)$
7. Learner conjectures: $a = m + x \wedge x \leq n$
8. Teacher replies: Thanks, I found a proof!

Extending Decision Tree Learning to handle implication samples

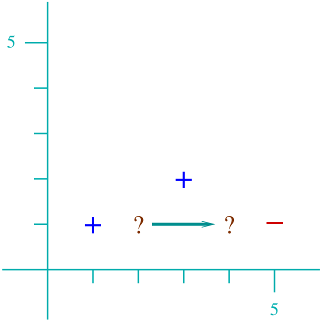
Now given S^+, S^-, and $S^{\implies}$. Learn a predicate (from a given concept class) consistent with given samples.

Challenges:

- Avoid having to backtrack or lookahead (to keep learning efficient).
- Can't recurse on sub-nodes independently.
- Entropy alone not a good gain hueristic.
- Avoid missing potential solutions.



Problem with using plain entropy when implications are there



Entropy would favour $x \leq 3$. However, $x \leq 4$ is clearly a better choice.

Proposed ICE Decision Tree Learning Algo

- Maintain a set partial classification G of the endpoints of implication pairs.
- Process nodes sequentially.
- Choose a split based on some hueristic (eg entroply $+$ penalty).
- If a node is turned into a finished node, propagate the classification to G.

Experimental evaluation

Table 1: Results comparing different invariant synthesis tools. χ_{TO} indicates that the tool times out (> 10 minutes); χ indicates that the t
incorrectly concludes that the program is buggy; χ_{MO} indicates that the tool runs out of memory; P, N, I are the number of positive, negat
examples and implications in the final sample of the resp. learner; *#R* is the number of rounds, and T is the time in seconds.

Program	White-box	Black-box											
	CPAchecker [12] (s)	Randomized Search [49]			ICE-CS [25]			ICE-DT-entropy			ICE-DT-penalty		
		Min.(s)	Max.(s)	Avg.(s) + TO	P,N,I	#R	T(s)	P,N,I	#R	T(s)	P,N,I	#R	T(s
SV-COMP programs and variants [2]													
array	2	0	123	18.5 + 3/10 TO	4,7,11	14	0.5	6,7,22	34	1.47	5,11,32	48	2.2
array2	2.4	0.1	384.5	105.7 + 4/10 TO	4,7,5	7	0.3	2,3,1	5	0.22	2,4,1	6	0.39
afnp	χ_{TO}	0.1	0.7	0.3 + 0/10 TO	1,19,15	29	3.6	1,3,7	11	0.48	1,2,7	10	0.47
cggmp	2	—	—	— + **10/10 TO**	1,36,50	71	51.1	1,18,45	64	3.48	1,17,42	60	3.01
countud	χ	—	—	— + **10/10 TO**	3,12,7	13	1	3,10,5	17	0.69	2,9,3	13	0.51
dtuc	χ_{TO}	4.9	190.4	62.8 + 2/10 TO	3,9,14	12	0.7	2,5,11	12	0.51	4,11,14	21	0.83
ex14	2.4	0	0.1	0.0 + 0/10 TO	2,5,1	7	0	1,1,0	2	0.12	1,1,0	2	0.11
ex14c	1.8	0.2	31.6	3.4 + 0/10 TO	2,2,1	4	0	2,2,0	3	0.12	2,2,0	3	0.14
ex23	5.4	0.1	127.5	21.8 + 1/10 TO	5,32,40	69	17.5	6,23,12	36	1.59	8,9,1	15	0.56
ex7	5.7	0	160.2	22.0 + 0/10 TO	1,2,1	2	0	1,1,0	2	0.12	1,1,0	2	0.09
matrixl1	3.3	—	—	— + **10/10 TO**	2,9,3	8	0.3	6,8,2	9	0.61	6,9,2	10	0.58
matrixl1c	3	—	—	— + **10/10 TO**	4,12,4	8	0.9	7,13,2	10	0.59	7,13,1	9	0.5
matrixl2	3.4	0.7	0.7	0.7 + 9/10 TO	8,19,13	27	22.9	8,11,8	23	1.25	9,11,6	22	1.06
matrixl2c	3.1	308	308	308.0 + 9/10 TO		χ_{TO}		15,26,10	44	2.61	20,35,22	66	3.95
nc11	2.1	0	0.1	0.1 + 0/10 TO	5,15,7	18	0.7	3,6,5	13	0.58	2,4,4	9	0.39
nc11c	2.1	0.1	46.1	6.3 + 2/10 TO	4,6,3	10	0.4	3,3,3	8	0.36	3,3,3	8	0.27
sum1	1.9	270.2	270.2	270.2 + 9/10 TO	2,15,10	17	2.3	3,11,2	14	0.58	3,11,2	14	0.56
sum3	2	0	0.1	0.1 + 0/10 TO	1,3,1	4	0.1	1,4,1	6	0.31	1,4,1	6	0.31
sum4	2.2	4.7	26.8	11.4 + 0/10 TO	1,23,31	44	3.5	1,9,41	51	2.42	1,8,41	50	2.46
sum4c	2	3.1	420.2	171.2 + 6/10 TO	6,29,21	34	11.6	4,14,7	22	1.05	4,13,4	18	0.86
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

What about proofs that require multiple annotations?

- Multiple (sequential or nested) while loops can be handled with ICE counterexamples.
- Some “modular” proofs of programs may need Horn implications
  - Programs with procedure calls
  - Owicki-Gries style proofs of concurrent programs
  - Rely-Guarantee proofs

Some proofs needing Horn implications

```
main() {
  x := y := 0;
  while (x < 10) {
    y := y + 1;
    f();
  }
  assert (x == 2y)
}

f() {
  x := x + 2;
}
```

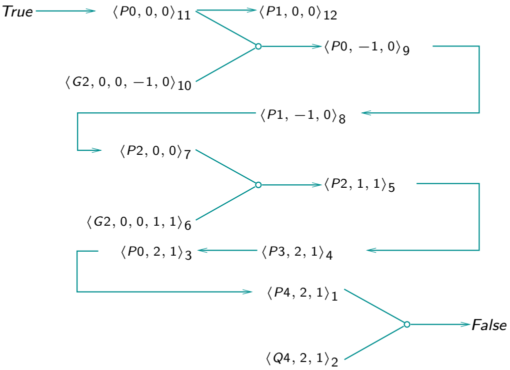
```
            Pre: x = y = 0

       T1           ||           T2

P0  while (*) {          Q0  while (*) {
P1    if (x < y)         Q1    if (y < 10)
P2      x := x + 1;      Q2      y := y + 3
P3  }                    Q3  }
P4                       Q4

             Post: x <= y
```

Example Rely-Gaurantee Proof

```
          Pre: x = y = 0

    T1         ||         T2

P0  while (*) {        Q0  while (*) {
P1    if (x < y)       Q1    if (y < 10)
P2      x := x + 1;    Q2      y := y + 3
P3  }                  Q3  }
P4                     Q4

          Post: x <= y
```

Adequacy		Inductiveness	
1.	$(x = 0 \land y = 0) \rightarrow P0$	1.	$P0 \rightarrow P1 \land P4$
2.	$P4 \land Q4 \rightarrow (x \leq y)$	2.	$P1 \land (x < y) \rightarrow P2$
		3.	$P2 \land [x := x + 1] \rightarrow P3'$
		4.	$P3 \rightarrow P0$
			...
Stability		**Guarantee**	
1.	$P0 \land G2 \rightarrow P0'$	1.	$P2 \land [x := x + 1] \rightarrow G1$
2.	$P1 \land G2 \rightarrow P1'$	2.	$Q2 \land [y := y + 3] \rightarrow G2$
	...		

Horn Counterexamples



How does one extend Decision Tree Learning to handle such a setting?

Conclusion

- Program verification is important if we want high assurance of the correctness of our programs.
- Coming up with adequate invariants is crucial to be able to automate Floyd-Hoare style verification.
- ICE framework for learning invariants.
- Extending popular Decision Tree Learning to ICE samples.
- Challenges in extending to multiple invariants.

Conclusion

- Program verification is important if we want high assurance of the correctness of our programs.
- Coming up with adequate invariants is crucial to be able to automate Floyd-Hoare style verification.
- ICE framework for learning invariants.
- Extending popular Decision Tree Learning to ICE samples.
- Challenges in extending to multiple invariants.

Thank you for your attention!