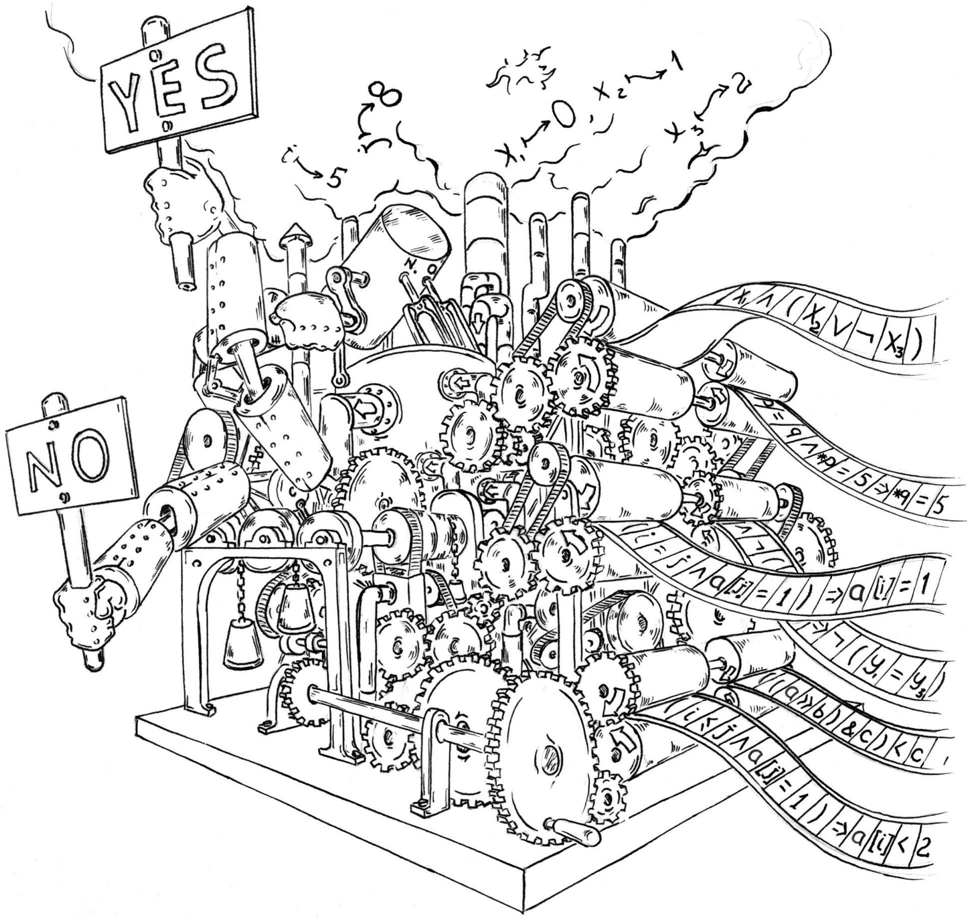
CSC 2547H: AUTOMATED REASONING WITH MACHINE LEARNING

Xujie Si

Assistant Professor

Department of Computer Science

University of Toronto



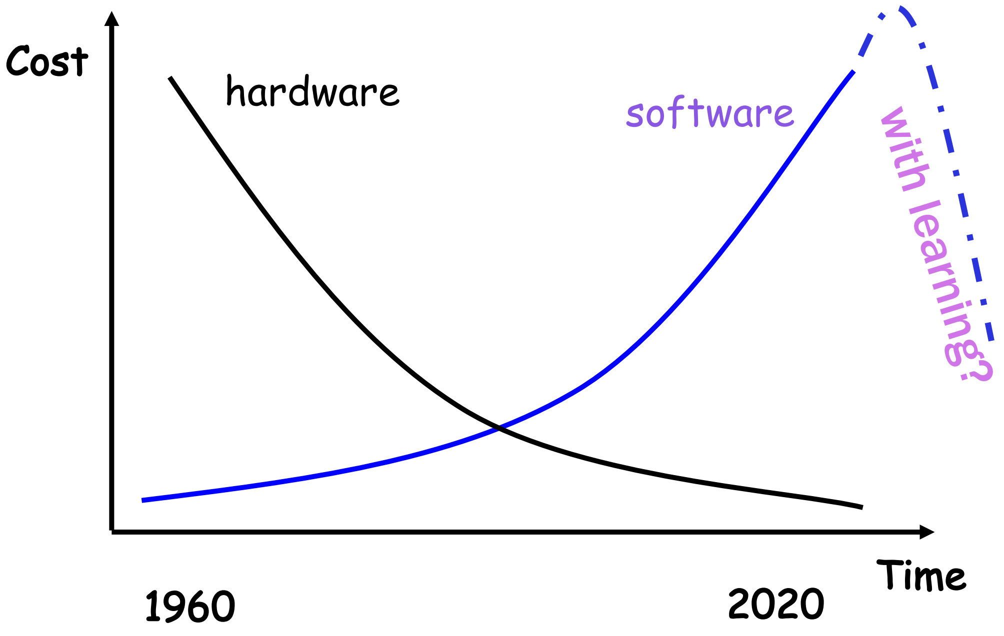
Paper presentation

- **Grading rubrics**
  - Preparation (15%)
    - Sign up on Ed 5%
    - Get feedback from TA 5%
    - Practice Recording 5%
  - Presentation (70% + 15% bonus)
    - Provide the necessary background 10%
    - Explain the problem and main challenges 10%
    - Illustrate the main ideas clearly 15%
    - Show the main results 15% + demo (15% bonus)
    - Limitations / related / future work discussion 10%
    - Finish under time 10% (around 20 minutes depending on the sign-ups)
  - Question Answering (15%)
    - In-class QA (10%)
    - Ed QA (5%)

Week	#Sign-ups
Week4: ml4sat	4
Week5: ml4smt	3
Week6: fm4ml	6
Week7: ml4code	2
Week8: dl4code	4
Week9: dl+logic	4
Week10: nv-sym	1

Lecture Overview

- **Program Analysis**
  - Dynamic Analysis
  - Static Analysis
- **Program Synthesis**
  - Programming by Examples / Demonstrations
  - Syntax-guided Program Synthesis



Program Analysis

- **Given a program, analyze whether it is “good”**
- **What is “good” or “bad”?**
    - Correctness
    - Performance
    - Energy efficiency
    - Memory footprint
    - Fault tolerance
    - Easy to read / maintain
    - Obscure enough to protect IP
    - Side channels (e.g., timing, cache miss, etc.)
    - Liveness, fairness, no crashes (e.g., deadlock, segment fault, etc.)

Dynamic Analysis

- **Software Testing**
  - Unit testing / Integration testing / System testing / Acceptance testing (alpha, beta)
  - Regression testing / compatibility testing
  - White-box testing / black-box testing / gray-box testing
  - Differential testing
  - Fuzzing
  - Mutation testing
  - Delta debugging
  - Code coverage

DBMS Testing: **400+ bugs** in widely-used DBMS
(SQLite, MySQL, MariaDB, PostgreSQL, CockroachDB, and TiDB)

Project Yin-Yang for SMT Solver Testing: [Z3/CVC4 bugs: **1,560** (total) / **1,061** (fixed)]
[Reports: YinYang, OpFuzz, TypeFuzz]

EMI & SPE Compiler Testing: [GCC/LLVM bugs: **1,634** (total) / **1,076** (fixed)]
[Reports: GCC (link1, link2, link3, link4, link5), LLVM (link1, link2, link3, link4, link5)]

https://people.inf.ethz.ch/suz/

https://lcamtuf.coredump.cx/afl/

https://google.github.io/oss-fuzz/

Symbolic Execution

- **Assuming there exists at least one input following a given execution path**
- **Keep track of symbolic constraints along the given path step by step**
- **Solve constraints**
  - SAT ➔ a concrete test case
  - UNSAT ➔ a proof that the given execution path is infeasible

```
def foo(x, y):
  if x < 10:
    y += x
    if y < 15:
      assert(False)   #crash 1
    else
      assert(x < 2*y) #crash 2
      return 100
  elif x == 1234:
    return 5678
  else:
      return x * y
```

$$x < 10 \land y_2 = x + y \land y_2 < 15$$

$$x < 10 \land y_2 = x + y \land y_2 \geq 15 \land x \geq 2 * y_2$$

Dynamic Symbolic Execution

- **When constraints become just too complicated**
  - Option-1: give up completely (not good ☹)
  - Option-2: Instantiate some symbolic values with concrete values, and move on

```python
def foo(x, y, z):
  if 2**x <= y * sin(y):
    w = x + z
    if w < y:
      assert (w > 0)
  else:
    assert (y <= z)
```

$$2^x \leq y * sin(y) \wedge w = x + z \wedge w < y \wedge w \leq 0$$

$$\frac{1}{2} \leq \frac{\pi}{2} * sin(\frac{\pi}{2}) \wedge w = (-1) + z \wedge w < \frac{\pi}{2} \wedge w \leq 0$$

Microsoft Zune

Zune bug explained in detail

Devin Coldewey @techcrunch / 9:58 PM EST • December 31, 2008

Comment

Earlier today, the sound of thousands of Zune owners crying out in terror made ripples across the blogosphere. The response from Microsoft is to wait until tomorrow and all will be well. You're probably wondering, what kind of bug fixes itself?

Well, I've got the code here and it's very simple, really; if you've taken an introductory programming class, you'll see the error right away.

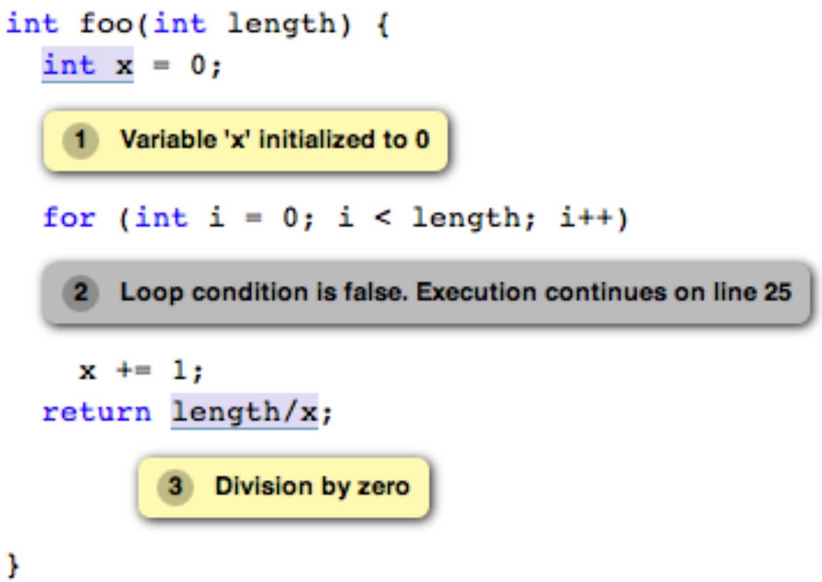
```
year = ORIGINYEAR;
while (days > 365)
{
    if (IsLeapYear(year))
    {
        if (days > 366)
        {
            days -= 366;
            year += 1;
        }
    }
    else
    {
        days -= 365;
        year += 1;
    }
}
```

```
static int IsLeapYear(int Year)
{
    int Leap;
    Leap = 0;
    if ((Year % 4) == 0) {
        Leap = 1;
        if ((Year % 100) == 0) {
            Leap = (Year%400) ? 0 : 1;
        }
    }
    return (Leap);
}
```

Static Analysis

- **Analyze programs without executing them**
  - Prevent bugs in the earliest stage (i.e., before any execution happens)
- **Compiler warnings and errors**
- **Linters, static checkers**
- **Type Checking and Inference**
- **Program Verification**

https://clang-analyzer.llvm.org/available_checks.html



Type Checking and Inference

- **Type safety**
  - Minimum requirement of non-buggy code
  - e.g. “hello” + 1.23 does not make sense
- **Dynamic typing vs static typing**
  - Python, JavaScript, PHP, Ruby, Racket, etc.
  - C/C++, C#, Java, Rust, Go, Standard ML, Ocaml, Haskell, etc.
- **Strongly statically typed languages tend to be safer, faster, less annotations, modular …**
- **Linear type**
  - Rust, Haskell, Idris, Linear ML, etc.
- **Dependent type**
  - Agda, Coq, Lean, Idris, Dependent ML, etc.

https://openai.com/blog/formal-math/

https://www.quantamagazine.org/lean-computer-program-confirms-peter-scholze-proof-20210728/

Program Verification

- **Symbolic Executions**
  - Explicitly enumerate and verify each path
  - How about loops / recursions?
- **Theorem Proving**
  - Hoare logic, separation logic, intuitionistic/constructive logic, etc.
- **Abstract Interpretation**
- **Software Model Checking**

Theorem Proving

- **Hoare Logic**

C. A. R. Hoare, *An Axiomatic Basis for Computer Programming*, CACM 1969

$$\{p\}\ \mathbf{skip}\ \{p\}$$

$$\{p[x \mapsto t]\}\ x := t\ \{p\}$$

$$\frac{\{p\}\ S_1\ \{r\} \qquad \{r\}\ S_2\ \{q\}}{\{p\}\ S_1;\ S_2\ \{q\}}$$

$$\frac{\{p \wedge b\}\ S_1\ \{q\} \qquad \{p \wedge \neg b\}\ S_2\ \{q\}}{\{p\}\ \mathbf{if}\ b\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2\ \{q\}}$$

$$\frac{\{p \wedge b\}\ S\ \{p\}}{\{p\}\ \mathbf{while}\ b\ \mathbf{do}\ S\ \{p \wedge \neg b\}}$$

Loop Invariant
(Fundamental Challenge)

$$\frac{\{p_1\}\ S\ \{q_1\}}{\{p\}\ S\ \{q\}}$$ provided $p \Rightarrow p_1$ and $q_1 \Rightarrow q$.

frama C
Software Analyzers

Dafny

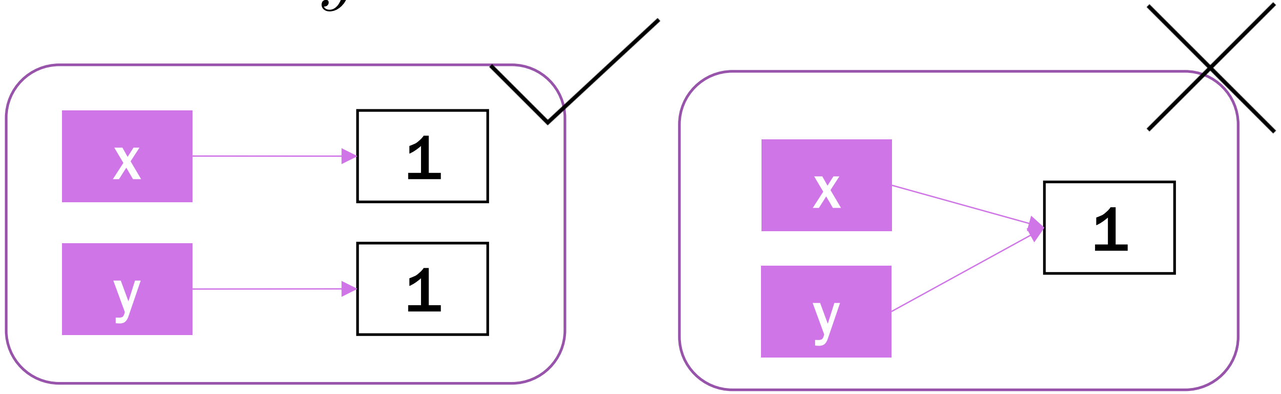
Theorem Proving

- **Separation Logic**
  - Reason about programs that manipulate pointer data structures
  - Split heaps into disjoint parts
  - Enable scalable compositional reasoning

$$P * Q$$

"and, separately"

$$x \mapsto 1 * y \mapsto 1$$



[Online demo]

Startups

Facebook Acquires Assets Of UK Mobile Bug-Checking Software Developer Monoidics

Josh Constine @joshconstine / 9:28 AM EDT • July 18, 2013

Frame Rule

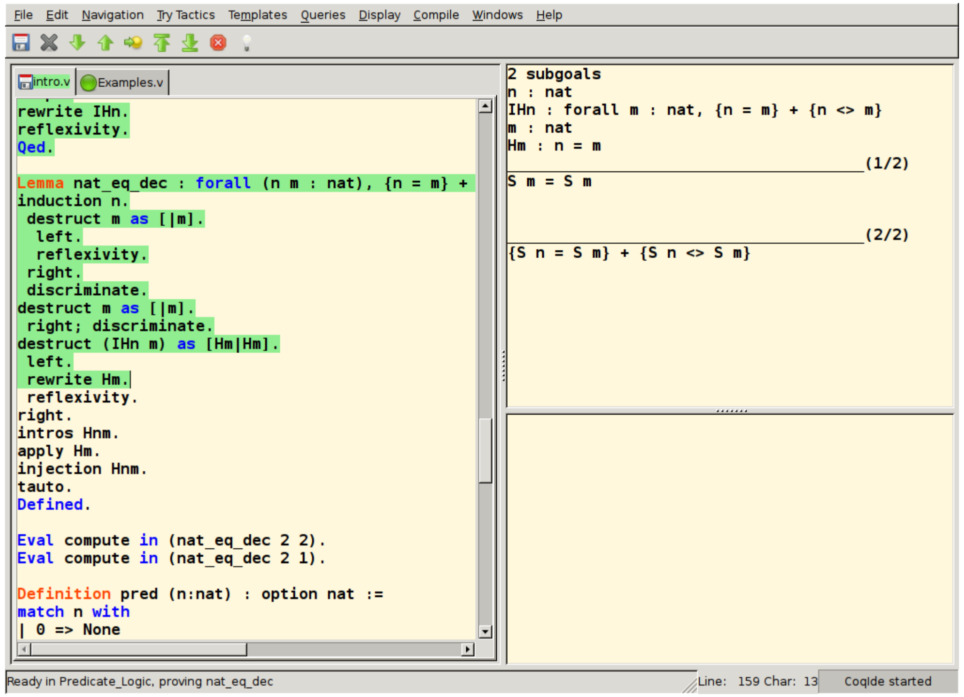
$$\frac{\{pre\}\,\mathrm{code}\,\{post\}}{\{pre * frame\}\,\mathrm{code}\,\{post * frame\}}$$

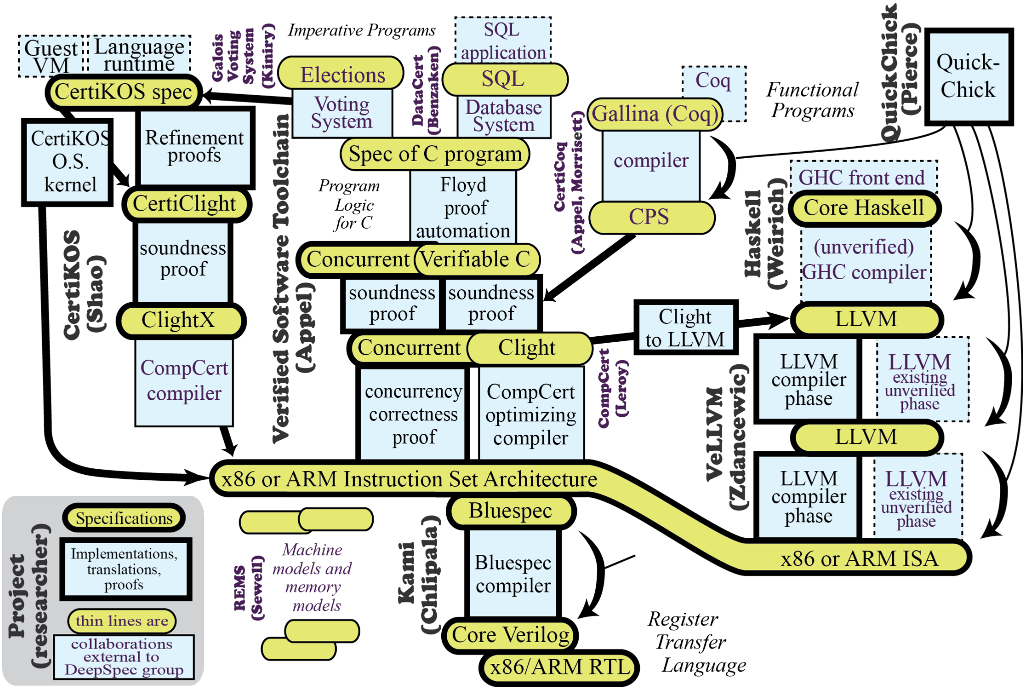
Peter O'hearn, *Separation logic*, CACM 2019

Theorem Proving

- **Intuitionistic/Constructive Logic/Dependent Types**
  - A theorem is a type
  - Building a proof is essentially constructing an object of that type

Curry–Howard correspondence





https://deepspec.org/main

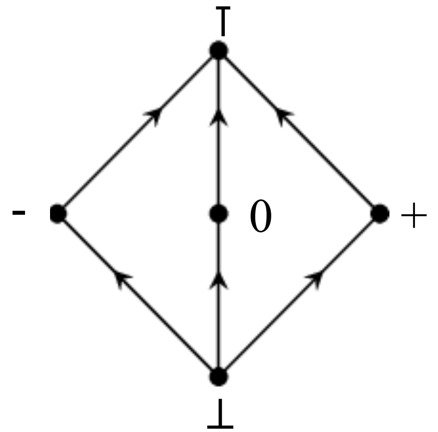
Abstract Interpretation

- **A theory of sound approximation of the semantics of programs**
  - Abstract domain $\mathbb{A}$, and Concrete domain $\mathbb{C}$
  - Abstraction function, $\alpha: \mathbb{C} \to \mathbb{A}$ $\qquad x \in \gamma(\alpha(x))$
  - Concretization function, $\gamma : \mathbb{A} \to 2^{\mathbb{C}}$ $\qquad x = \alpha(\gamma(x))$

```
def foo(x):
  if x <= 0:
    return 1 - x
  else:
    return x
```

$\mathbb{C} = \{-2147483648, \dots, -1, 0, 1, \dots, 2147483647\}$

$\mathbb{A} = \{+, 0, -, \top, \bot\}$



minus	+	0	-	⊤	⊥
+	⊤	+	+	⊤	⊥
0	-	0	+	⊤	⊥
-	-	-	⊤	⊤	⊥
⊤	⊤	⊤	⊤	⊤	⊥
⊥	⊥	⊥	⊥	⊥	⊥

Cousot & Cousot, *Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints*, POPL 1977

Successful Applications of A.I.

Astrée

https://www.astree.ens.fr/

AbsInt
Angewandte Informatik

AIRBUS

framatome

BOSCH

HELBAKO

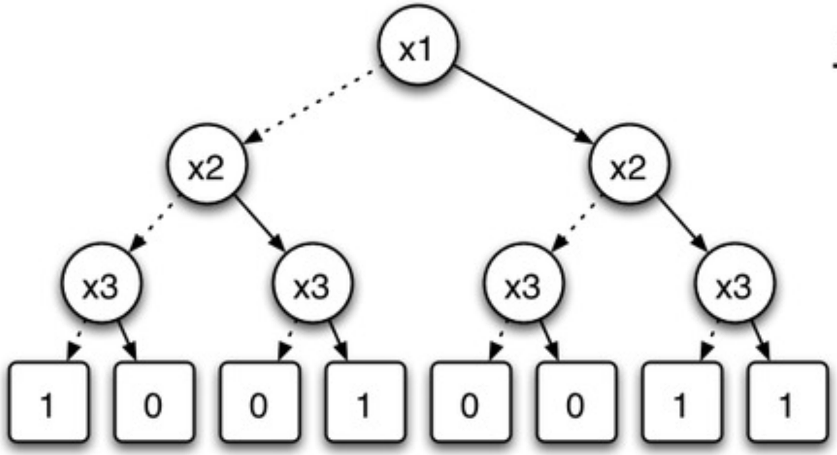
esa

mtu

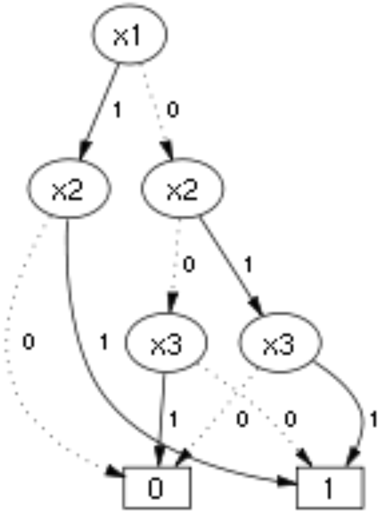
Program Verification

- **Model checking**
  - Check whether a finite-state model (FSM) meets a given specification (usually in temporal logic)
  - Earlier successes are in hardware design
- **Symbolic model checking**
  - Using Binary decision diagram (BDD) to represent all possible states

A hardware bug incurred $475 million loss

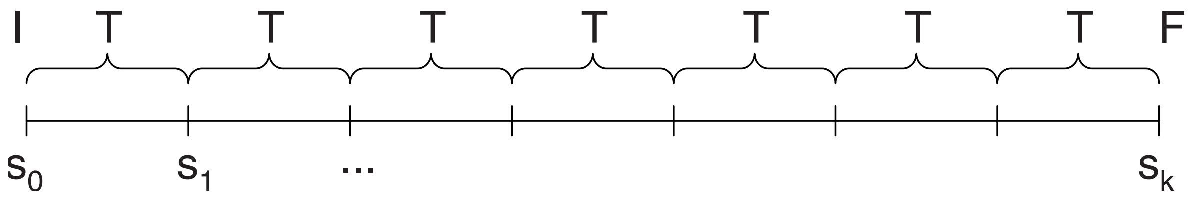


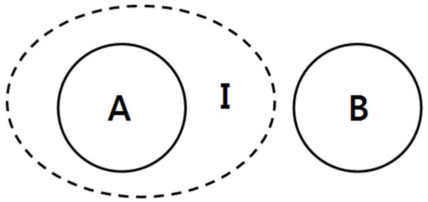
x1	x2	x3	*f*
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



Program Verification

- **Bounded model checking**
  - Unroll FSM up to a fixed number of steps
  - Check whether a property is violated using SAT solver





Craig's Interpolant

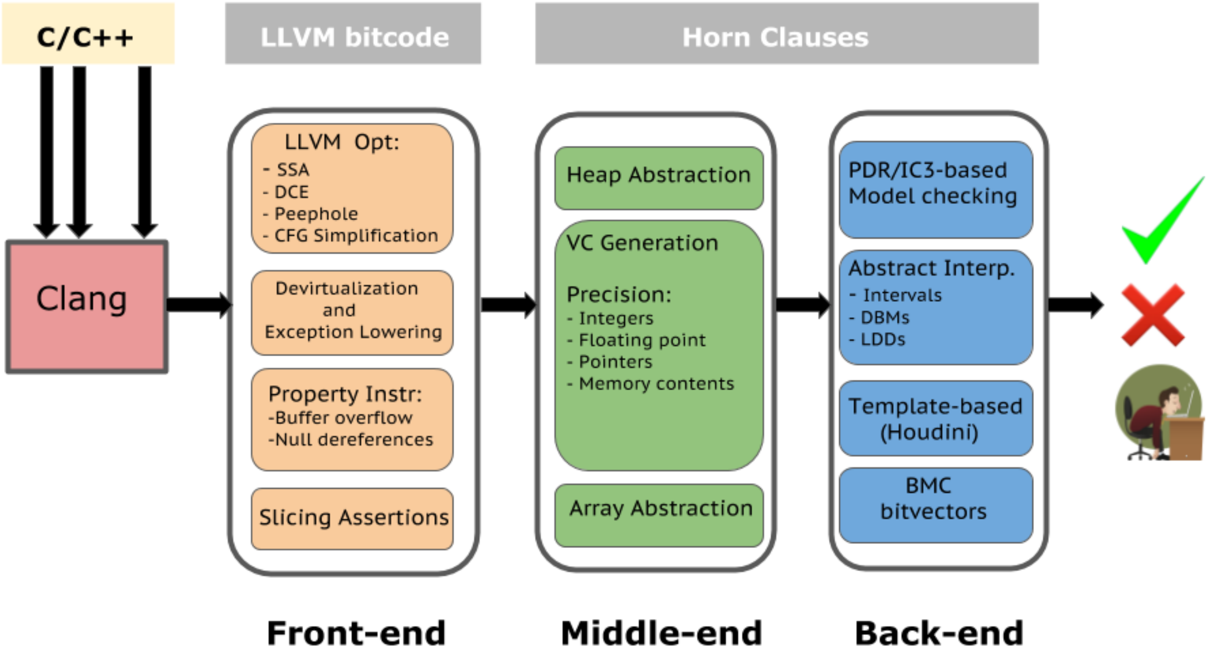
$A : \{(p), (\neg p \vee q)\}$

$B : \{(\neg q \vee r), (\neg r)\}$

$I: \{q\}$

Program Verification

- **Software Model Checking**
  - Extend model checking to software, which has infinite number of states



Program Verification

- Constrained Horn Clauses

$$\forall \mathcal{V} \cdot (\varphi \wedge p_1(X_1) \wedge \cdots \wedge p_k(X_k) \rightarrow h(X)),\ \text{for } k \geq 1$$

```
main(){
 int x,y;
 x=1; y=0;
 while(*){
  x=x+y;
  y++;}
 assert(x>=y);}
```

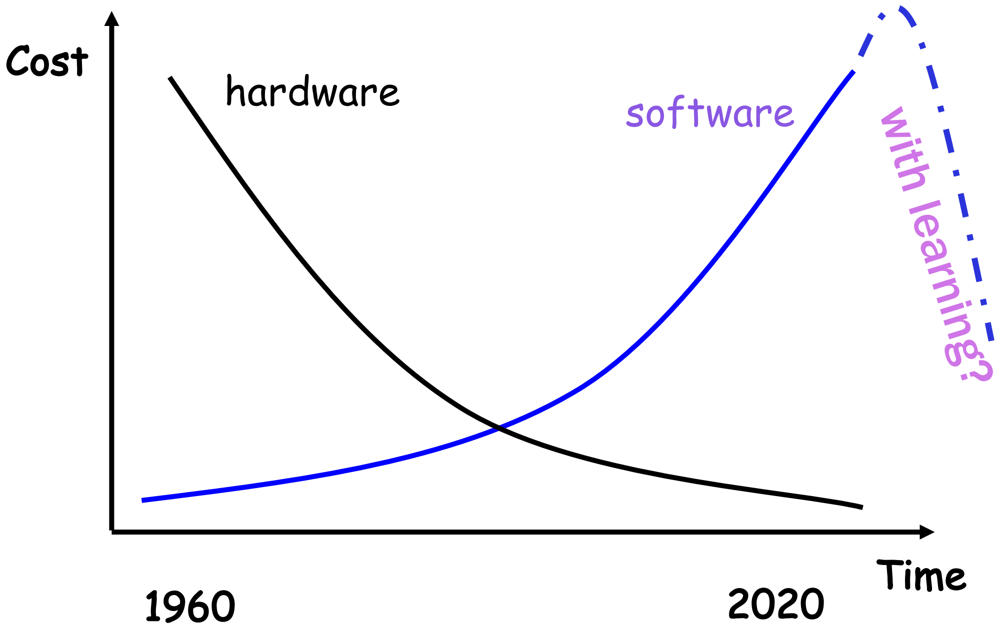
$$x = 1 \wedge y = 0 \rightarrow p(x, y) \tag{1}$$

$$p(x, y) \wedge x' = x + y \wedge y' = y + 1 \rightarrow p(x', y') \tag{2}$$

https://www.microsoft.com/en-us/research/blog/spacer-and-z3-accessible-reliable-model-checking-as-theorem-proving/

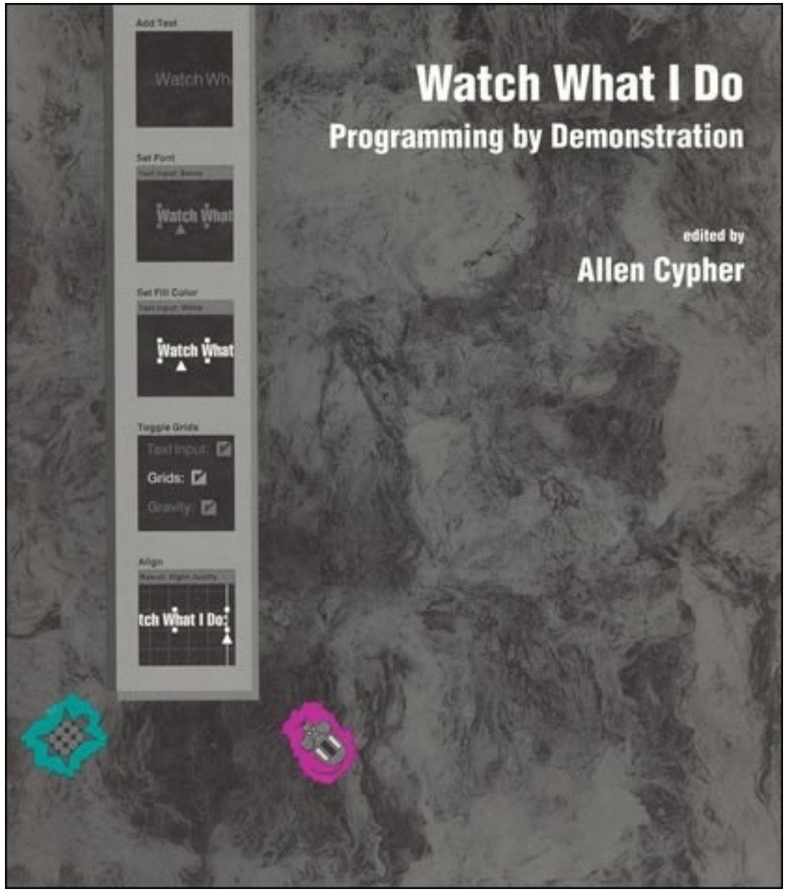
Lecture Overview

- **Program Analysis**
  - Dynamic Analysis
  - Static Analysis
- **Program Synthesis**
  - Programming by Examples / Demonstrations
  - Syntax-guided Program Synthesis



Programming by Examples

Watch What I Do: Programming by Demonstration, 1993

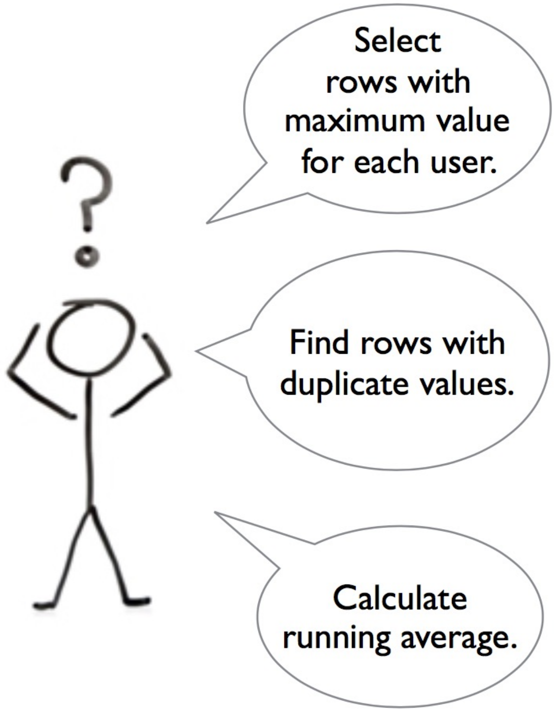


https://web.media.mit.edu/~lieber/PBE/PBE-Examples.html

Like neural networks, an old idea becomes fashionable again …

Programming by Examples

End User



stackoverflow

SQL

```sql
Select x.id, x.customer, x.total
From PURCHASES x
Join (Select p.customer,
             Max(total)
        From PURCHASES p
        Group By p.customer) y
On y.customer = x.customer
   And y.max_total = x.total
```

```sql
Select *
From  Users a
Where Exists
   (Select *
    From Users b
    Where (a.name = b.name
           Or  a.email = b.email)
          And a.ID <> b.id)
```

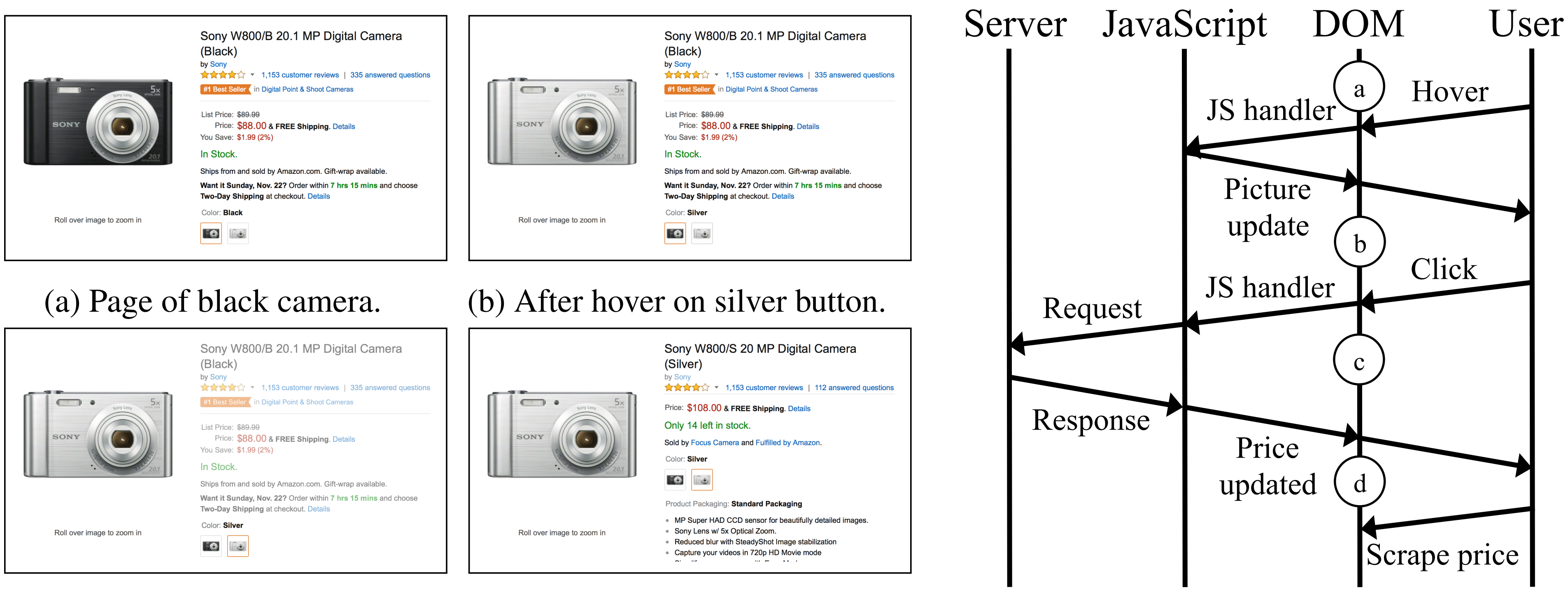
```sql
Select a.ord, a.val, Avg(b.val)
From t As a Join t As b
Where b.ord <= a.ord
Group By a.ord,a.val
Order By a.ord
```

Wang et al., *Synthesizing Highly Expressive SQL Queries from Input-Output Examples*, PLDI 2017

https://stackoverflow.com/questions/40015743/jpa-distinct-and-limiting-result-number

Programming by Examples

Let's say we want to find cheap silver cameras on Amazon …



(a) Page of black camera.

(b) After hover on silver button.

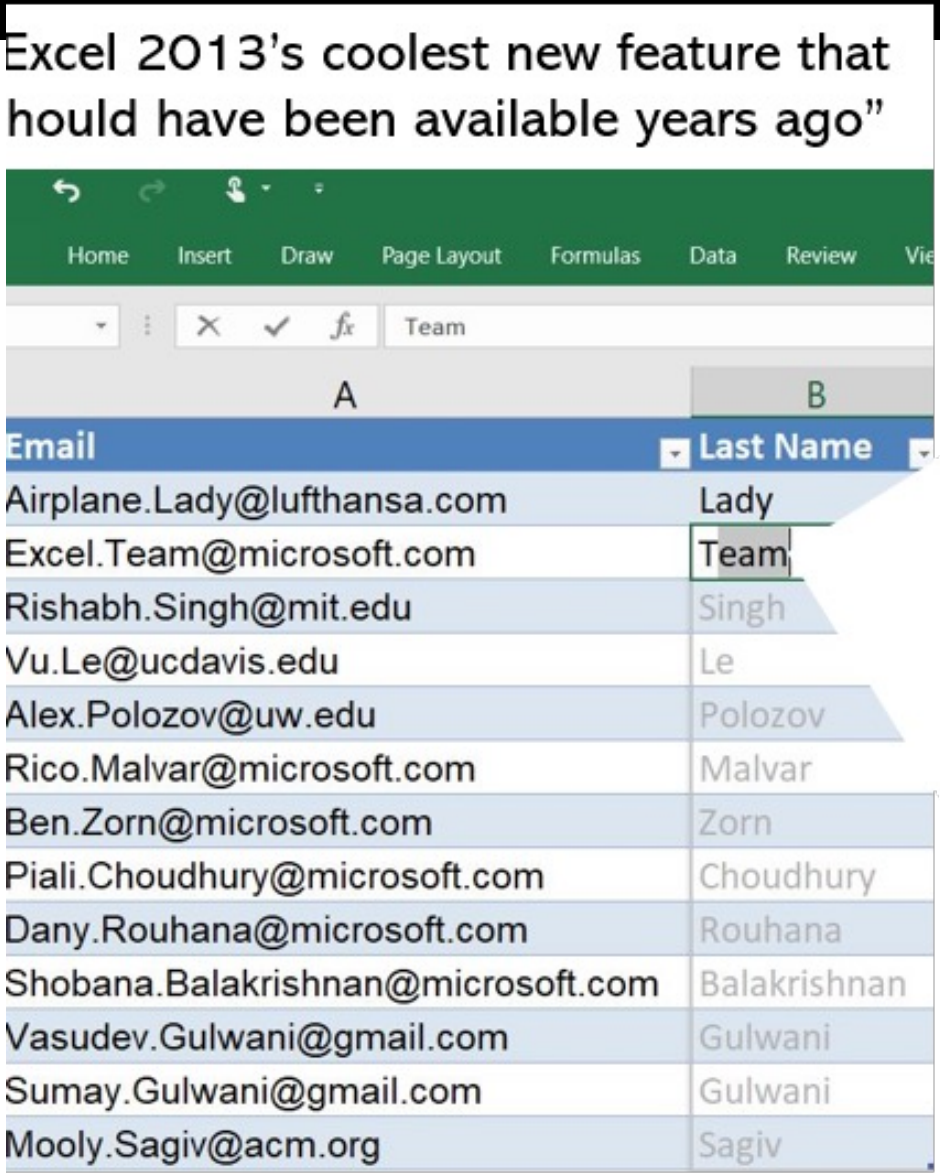
(c) After click (grayed page).

(d) Page of silver camera.

(e) Sequence of events.

Barman et al., *Ringer: Web Automation by Demonstration*, OOPSLA 2016

FlashFill demo



Automating String Processing in Spreadsheets Using Input-Output Examples

Sumit Gulwani
Microsoft Research, Redmond, WA, USA
sumitg@microsoft.com

Abstract

We describe the design of a string programming/expression language that supports restricted forms of regular expressions, conditionals and loops. The language is expressive enough to represent a wide variety of string manipulation tasks that end-users struggle with. We describe an algorithm based on several novel concepts for synthesizing a desired program in this language from input-output examples. The synthesis algorithm is very efficient taking a fraction of a second for various benchmark examples. The synthesis algorithm is interactive and has several desirable features: it can rank multiple solutions and has fast convergence, it can detect noise in the user input, and it supports an active interaction model wherein the user is prompted to provide outputs on inputs that may have multiple computational interpretations.

their task [9]. More significantly, programming is still required to perform tedious and repetitive tasks such as transforming entities like names/phone-numbers/dates from one format to another, data cleansing, extracting data from several text files or web pages into a single document, etc. Spreadsheet systems like Microsoft Excel allow users to write macros using a rich inbuilt library of string and numerical functions, or to write arbitrary scripts using a variety of programming languages like Visual Basic, or .Net. Since end-users are not proficient in programming, they find it too difficult to write desired macros or scripts.

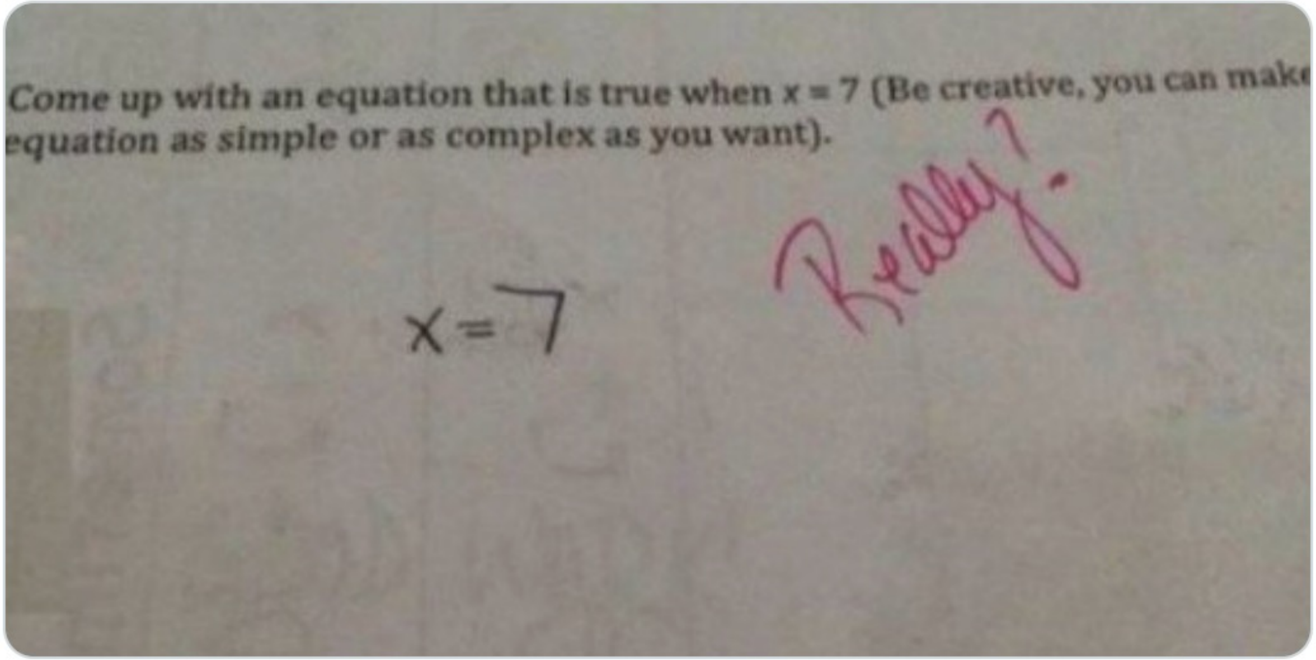
We have performed an extensive case study of spreadsheet help forums and identified that string processing is one of the m… common class of programming problems that end-users stru… with. This is not surprising given that languages like Perl, …

POPL Most Influential Paper Award 2021

Any concerns of PBE?

Kristopher Micinski
@krismicinski

Program synthesis in a nutshell



10:28 PM · Jan 21, 2017

874 10 Copy link to Tweet

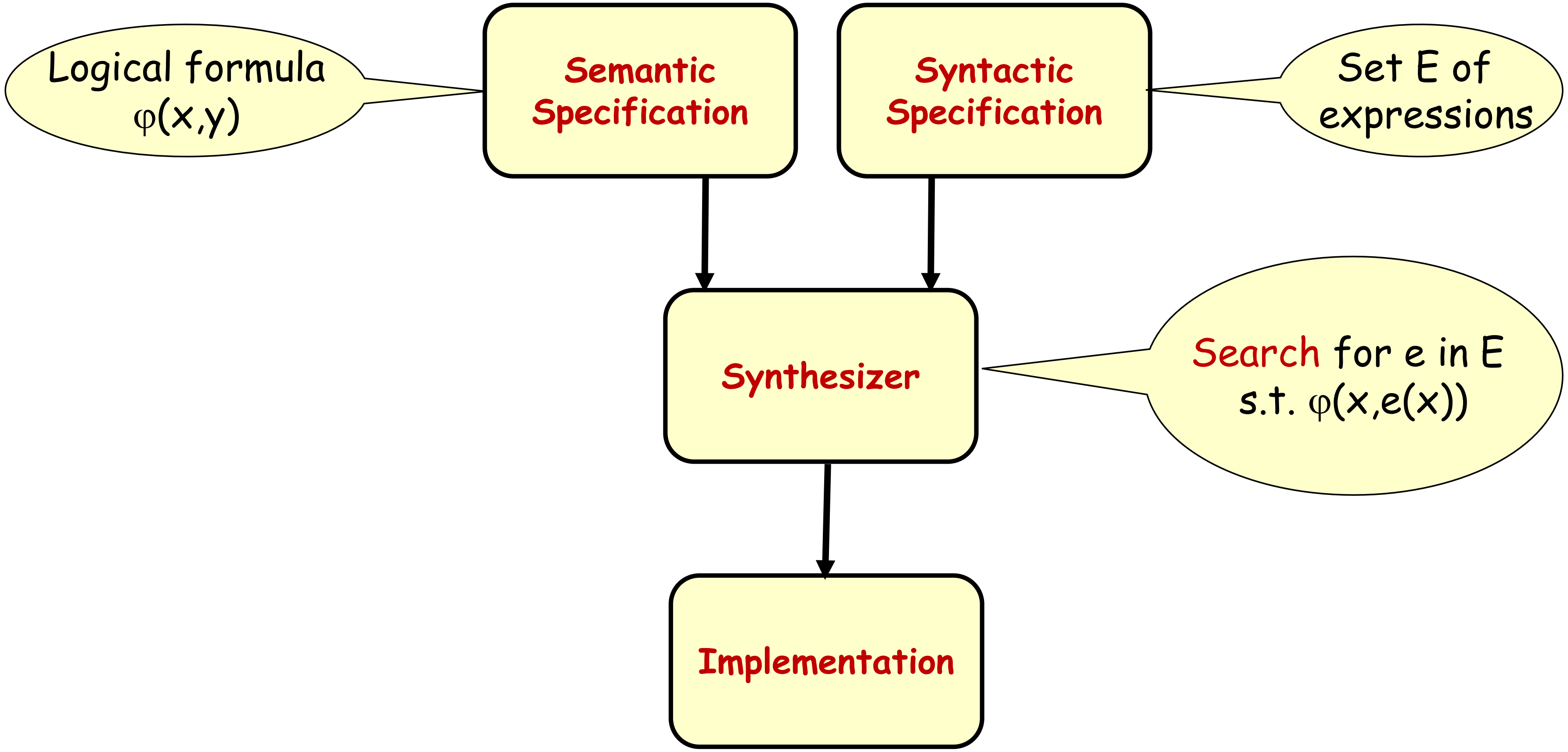
Given $\{(i_1, o_1), \dots, (i_n, o_n)\}$, synthesize a program

Easy!

$$\begin{gathered} if\ I = i_1, then\ return\ o_1 \\ if\ I = i_2, then\ return\ o_2 \\ \dots \\ if\ I = i_n, then\ return\ o_n \end{gathered}$$

Twitter link

Syntax-guided Program Synthesis



SyGuS Problem

- Fix a background theory T: fixes types and operations

- Function to be synthesized: name f along with its type
  - General case: multiple functions to be synthesized

- Inputs to SyGuS problem:
  - Specification $\varphi(x, f(x))$
    Typed formula using symbols in T + symbol f
  - Set E of expressions given by a context-free grammar
    Set of candidate expressions that use symbols in T

- Computational problem:
  Output e in E such that $\varphi[f/e]$ is valid (in theory T)

SyGuS Example 1

- Theory QF-LIA (Quantifier-free linear integer arithmetic)
  - Types: Integers and Booleans
  - Logical connectives, Conditionals, and Linear arithmetic
  - Quantifier-free formulas
- Function to be synthesized $f\ (\text{int}\ x_1, x_2) : \text{int}$
- Specification: $(x_1 \leq f(x_1, x_2))$ & $(x_2 \leq f(x_1, x_2))$
- Candidate Implementations: Linear expressions
  - LinExp := x_1 | x_2 | Const | LinExp + LinExp | LinExp - LinExp
- No solution exists

What is the solution?

SyGuS Example 2

- Theory QF-LIA
- Function to be synthesized: f (int x_1 , x_2) : int
- Specification: $(x_1 \leq f(x_1, x_2))$ & $(x_2 \leq f(x_1, x_2))$
- Candidate Implementations: Conditional expressions without +

  Term := x_1 | x_2 | Const | If-Then-Else (Cond, Term, Term)
  Cond := Term ≤ Term | Cond & Cond | ~ Cond | (Cond)

- Possible solution:
  If-Then-Else $(x_1 \leq x_2, x_2, x_1)$

How to solve SyGuS problems?

- Enumerative Search
- Constraint Solving (SMT)
- Stochastic Search

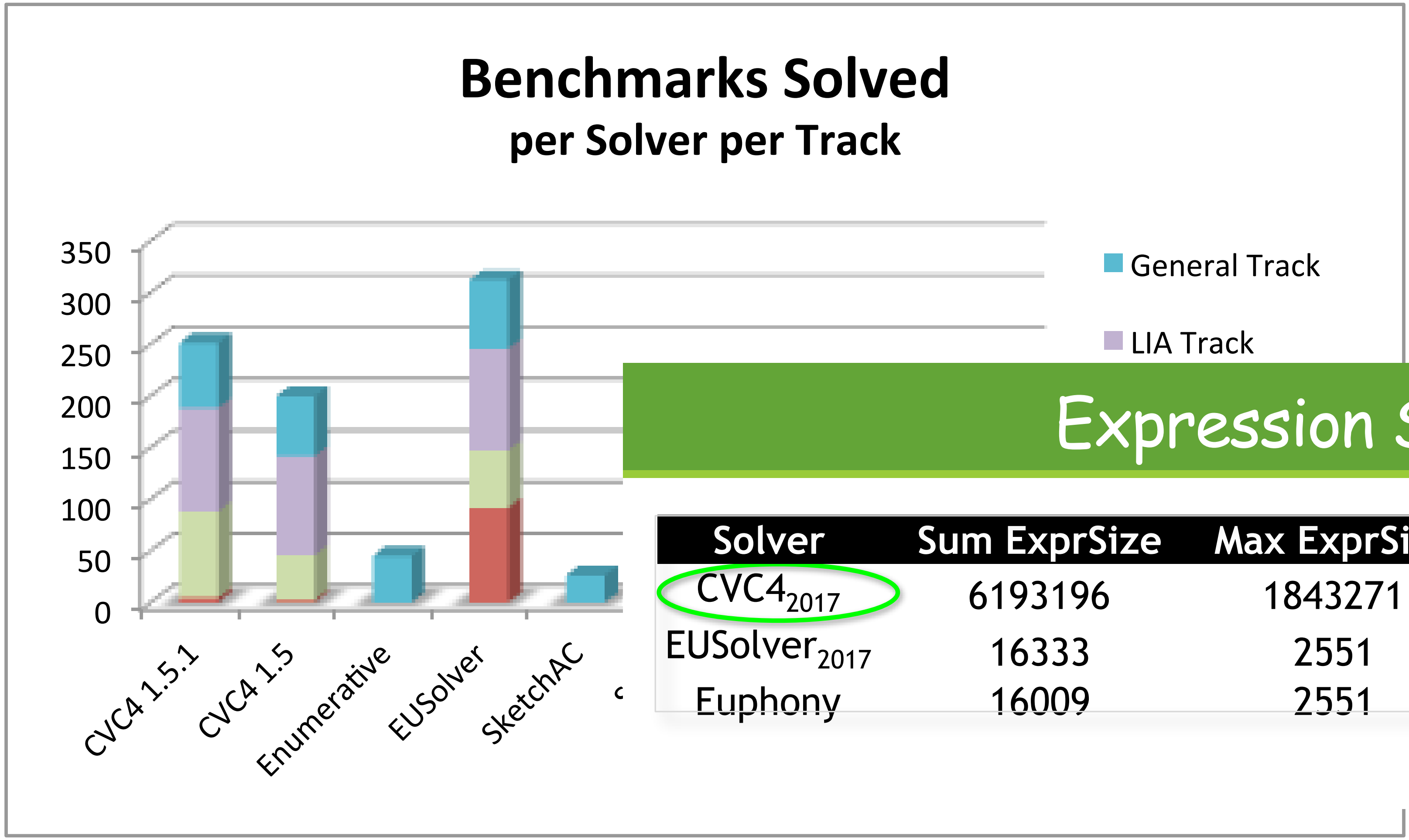
Enumerative Search

- **Enumerate programs from small to large**
- **Pruning is important**
- **Run test cases**
- **No need to enumerate equivalent sub-expressions**
  - But how to avoid that?
- **Use pre-defined rules**
  - E.g. A + B == B + A
- **Indistinguishability based on tests**

Udupa et al., *TRANSIT: specifying protocols with concolic snippets*, PLDI 2013

Alur et al., *Synthesis Through Unification*, CAV 2015

Is it effective?



Expression Sizes

Solver	Sum ExprSize	Max ExprSize	Avrg ExprSize
$CVC4_{2017}$	6193196	1843271	16559.34759
$EUSolver_{2017}$	16333	2551	40.62935323
Euphony	16009	2551	44.34626039

$CVC4_{2017}$ generates quite big expressions...

https://sygus.org/comp/2016/report.pdf

https://sygus.org/comp/2017/results-slides.pdf

Stochastic Search

- **Start with a random program**
- **Mutate randomly (MCMC sampling)**
- **Stop when finding the correct program**

Search(J: Initial candidate)
Returns: A candidate C with $c_V(C) = 0$.

$C := J$
while $c_V(C) \neq 0$ **do**
$\quad m := SampleMove(rand())$
$\quad C' := m(C)$
$\quad c_o := c_V(C)$, $c_n := c_V(C')$
$\quad$ **if** $c_n < c_o$ or $e^{-\gamma(c_n - c_0)} > \frac{rand()}{RANDMAX}$ **then**
$\qquad C := C'$
$\quad$ **end if**
end while
return C

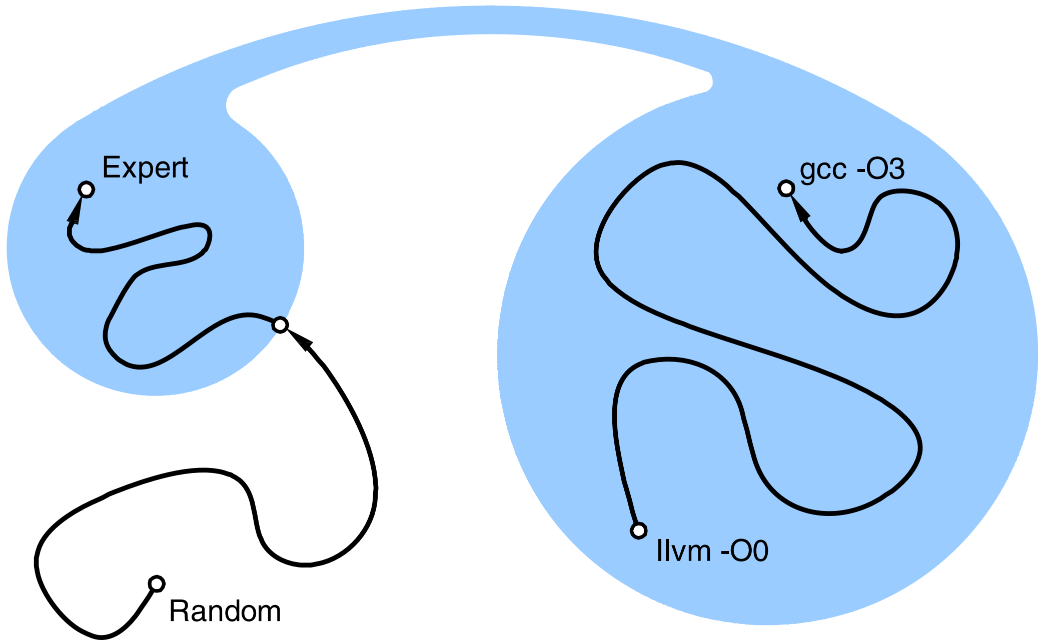
Success Example

```
# gcc -O3

.L0:
  movq rsi, r9
  movl ecx, ecx
  shrq 32, rsi
  andl 0xffffffff, r9d
  movq rcx, rax
  movl edx, edx
  imulq r9, rax
  imulq rdx, r9
  imulq rsi, rdx
  imulq rsi, rcx
  addq rdx, rax
  jae .L2
  movabsq 0x100000000, rdx
  addq rdx, rcx
.L2:
  movq rax, rsi
  movq rax, rdx
  shrq 32, rsi
  salq 32, rdx
  addq rsi, rcx
  addq r9, rdx
  adcq 0, rcx
  addq r8, rdx
  adcq 0, rcx
  addq rdi, rdx
  adcq 0, rcx
  movq rcx, r8
  movq rdx, rdi
```

```
# STOKE

.L0:
  shlq 32, rcx
  movl edx, edx
  xorq rdx, rcx
  movq rcx, rax
  mulq rsi
  addq r8, rdi
  adcq 0, rdx
  addq rdi, rax
  adcq 0, rdx
  movq rdx, r8
  movq rax, rdi
```



Schkufza et al, *Stochastic Superoptimization*, ASPLOS 2013

Success Example

Benchmark	Target LOC	Best LOC	Best Speedup	Search Time (min)	DDEC Time (min)
wcpcpy	40	13	48%	37	38
wcslen	43	47	97%	78	89
wmemset	47	47	0%	29	45
wcsnlen	94	51	2%	61	83
wmemcmp	91	77	47%	360	302
wcschr	87	28	2%	61	5
strxfrm	99	38	0%	81	414
wcscmp	108	29	47%	38	586
wmemchr	132	75	2%	67	30
wcscpy	35	40	25%	276	252
wcscat	89	90	26%	360	46
strcpy	70	63	30%	360	415
wcsrchr	178	178	0%	30	15

Churchill et al, *Sound Loop Superoptimization for Google Native Client*, ASPLOS 2017