Syntax-Guided Program Synthesis

Rajeev Alur

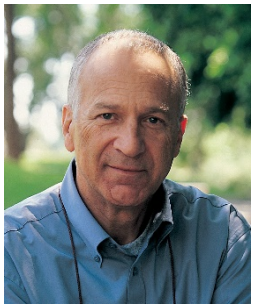
University of Pennsylvania

ExCAPE
Expeditions in Computer Augmented Program Engineering

Penn Engineering

Goal: Programming computers easier than communicating with people

Can programming be liberated, period.
David Harel, IEEE Computer, 2008



Enabling Technologies

- More computing power
- Mature software analysis/verification tools
- Better human-computer interfaces
- Data mining tools for code repositories


Foster


Hartmann


Lafortune


Kavraki


Kress-Gazit


Loo


Madhusudan


Bodik


Martin

Expeditions in Computer Augmented Program Engineering

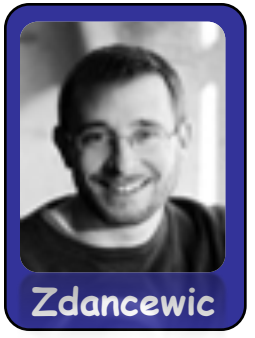
http://excape.cis.upenn.edu/

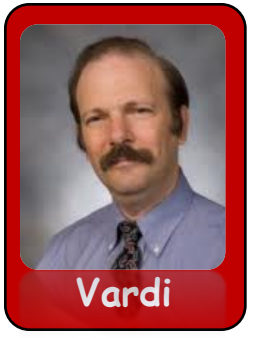
Cornell, Maryland, Michigan, MIT, Penn, Rice, UC Berkeley, UCLA, UIUC

2012--2018

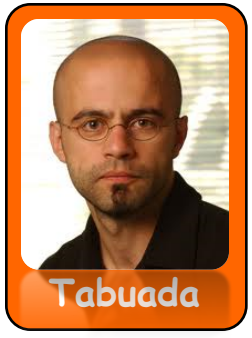

Alur


Pappas


Zdancewic


Vardi


Tripakis


Tabuada


Solar-Lezama


Seshia


Sangiovanni

End-User Programming

Can non-programmers communicate intent intuitively?

- People commanding robots
- Analysts harvesting data from the web
- Network operators configuring switches

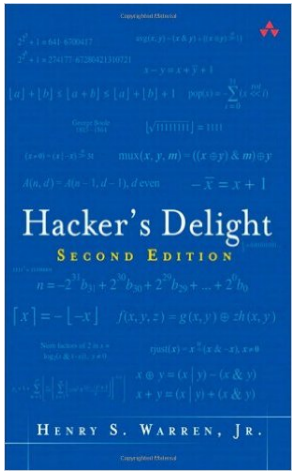
Opportunity: Logic to be programmed is simple

Possible Solution: Programming by Examples (or by Demonstration)

Programming By Examples (PBE)

Desired program P: bit-vector transformation that resets rightmost substring of contiguous 1's to 0's

1. P should be constructed from standard bit-vector operations
   |, &, ~, +, -, <<, >>, 0, 1, …
2. P specified using input-output examples
   00101 → 00100
   01010 → 01000
   10110 → 10000



Desired solution:

x & (1 + (x | (x-1))

FlashFill: PBE in Practice

Ref: Gulwani (POPL 2011)

Input	Output
(425)-706-7709	425-706-7709
510.220.5586	510-220-5586
1 425 235 7654	425-235-7654
425 745-8139	425-745-8139

Wired: Excel is now a lot easier for people who aren't spreadsheet- and chart-making pros. The application's new Flash Fill feature recognizes patterns, and will offer auto-complete options for your data. For example, if you have a column of first names and a column of last names, and want to create a new column of initials, you'll only need to type in the first few boxes before Excel recognizes what you're doing and lets you press Enter to complete the rest of the column.

Program Optimization

Can regular programmers match experts in code performance?

- Improved energy performance in resource constrained settings
- Adoption to new computing platforms such as GPUs

Opportunity: Semantics-preserving code transformation

Possible Solution: Superoptimizing Compiler

- Structure of transformed code may be dissimilar to original

Superoptimization Illustration

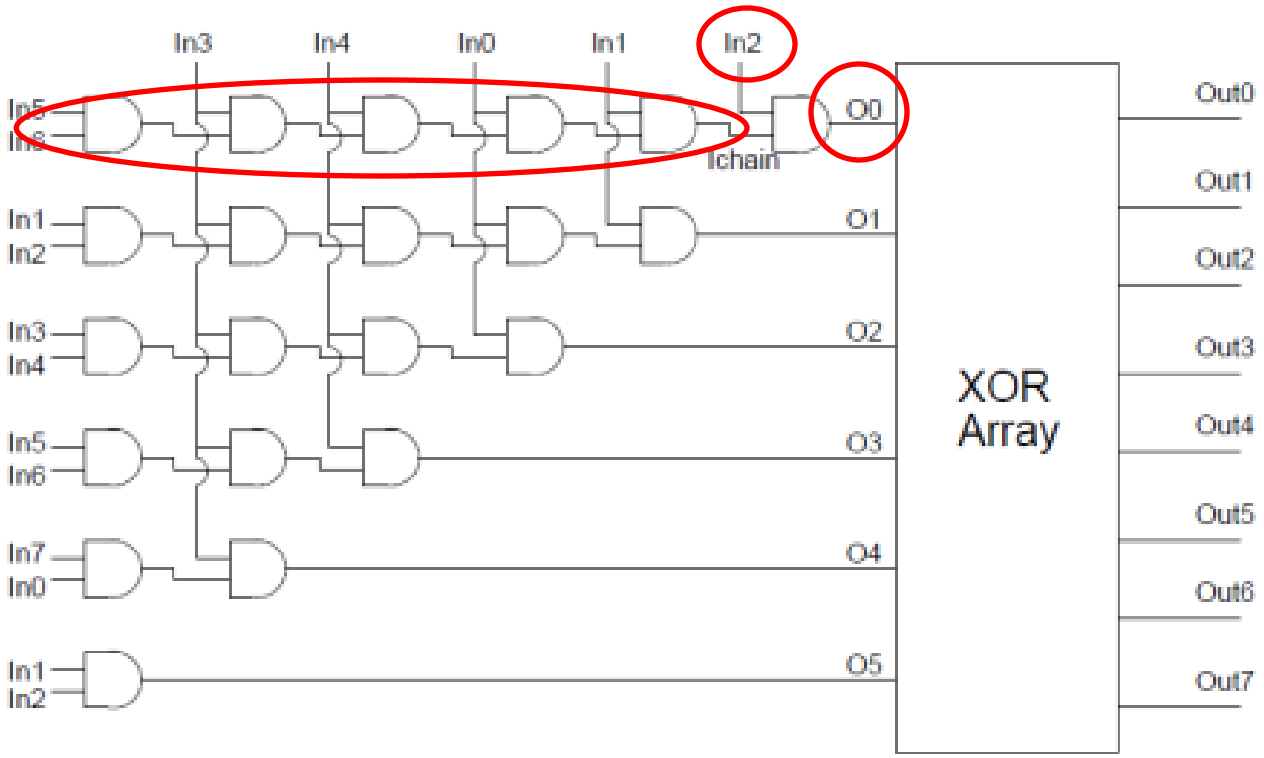
Given a program P, find a "better" equivalent program P'

```
average (bitvec[32] x, y) {
  bitvec[64] x1 = x;
  bitvec[64] y1 = y;
  bitvec[64] z1 = (x1+y1)/2;
  bitvec[32] z = z1;
  return z
}
```

Find equivalent code without extension to 64 bit vectors

```
average (x, y) =
  (x and y) + [(x xor y) shift-right 1 ]
```

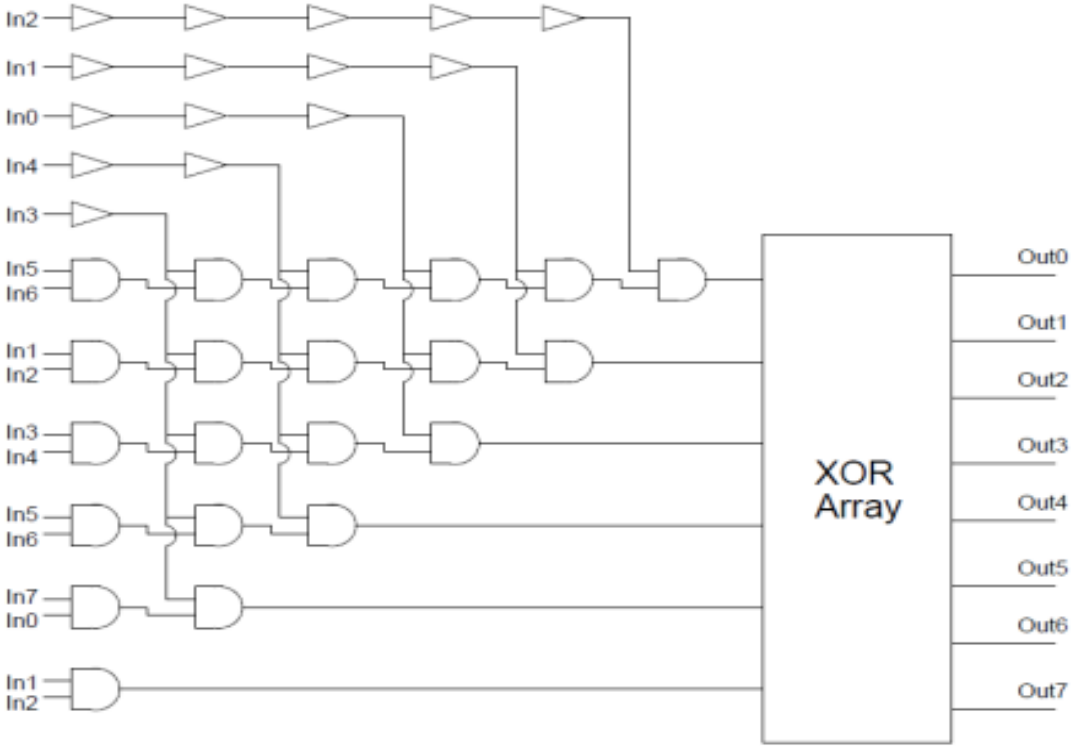
Side Channel Attacks on Cryptographic Circuits



PPRM1 AES S-Box implementation [Morioka and Satoh, 2002]

Vulnerability: Timing-based attack can reveal secret input In2

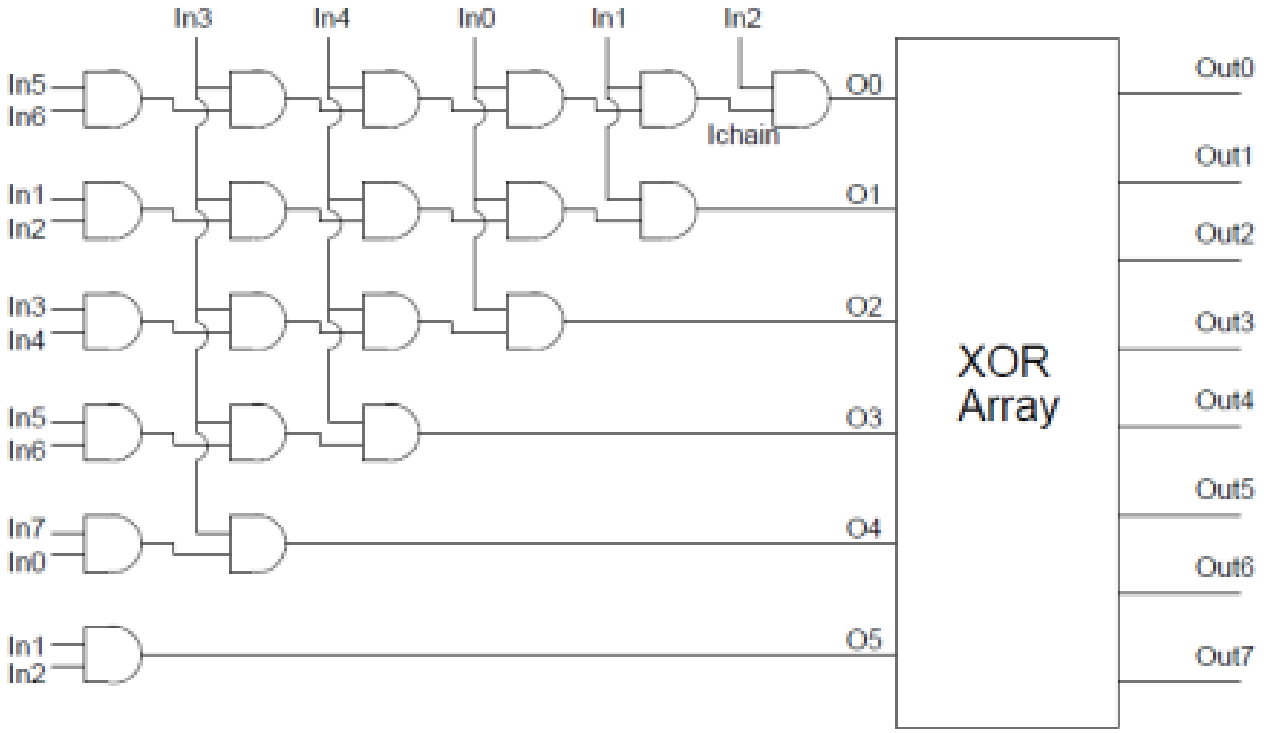
Countermeasure to Attack



FSA attack resilient ckt: All input-to-output paths have same delays

Manually hand-crafted solution [Schaumont et al, DATE 2014]

Synthesis of Attack Countermeasures



Given a circuit *C*, automatically synthesize a circuit *C'* such that

1. *C'* is functionally equivalent to *C* [sematic constraint]
2. All input-to-output paths in *C'* have same length [syntactic constraint]

Existing EDA tools cannot handle this synthesis problem

Syntax-Guided Program Synthesis

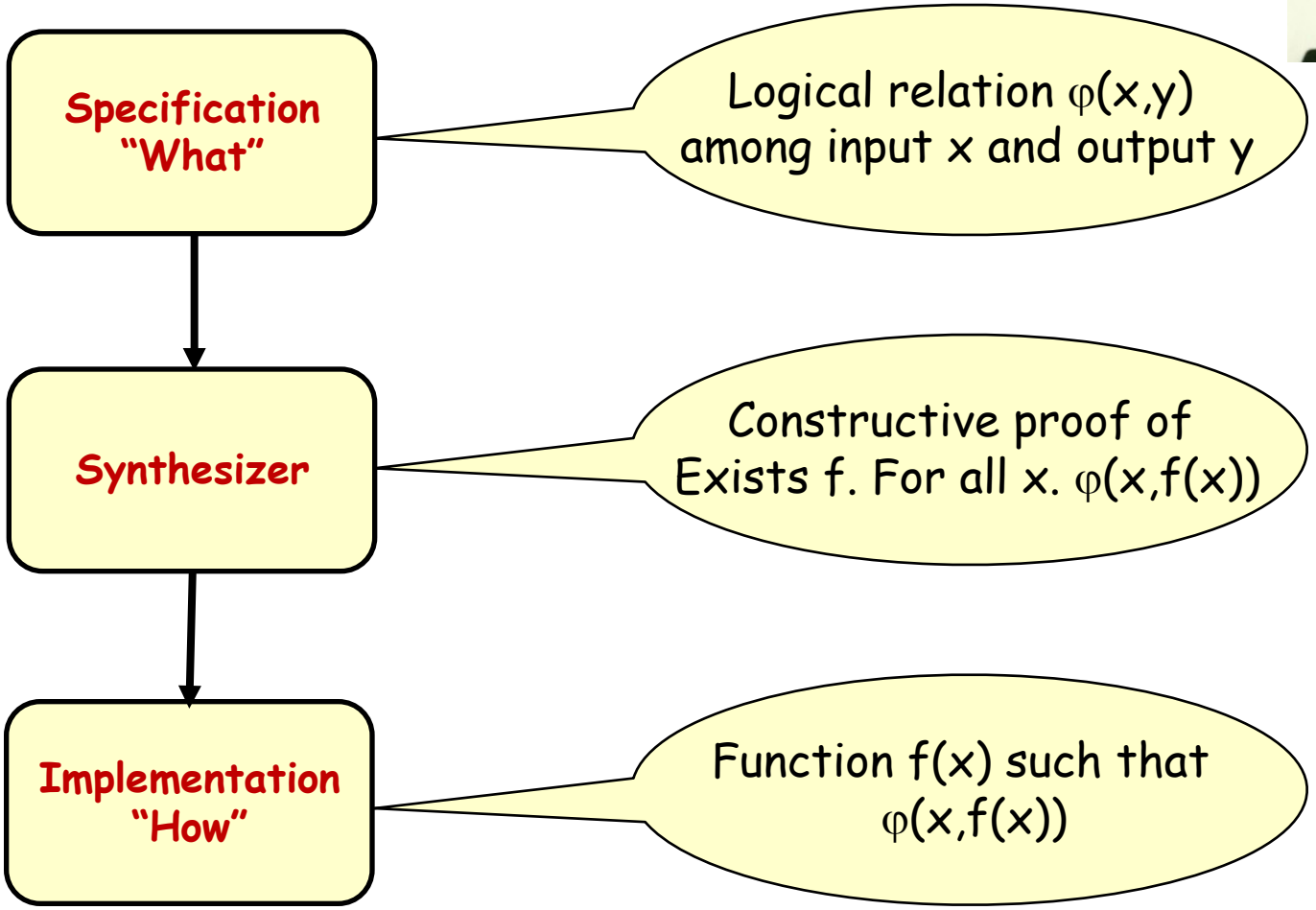
Rich variety of projects in programming systems and software engineering

- Programming by examples
- Automatic program repair
- Program superoptimization
- Template-guided invariant generation
- Autograding for programming assignments
- Synthesis of patches against security vulnerabilities
- Extracting SQL queries corresponding to Java code fragments

Computational problem at the core of all these synthesis projects:
Find a program that meets given syntactic and semantic constraints

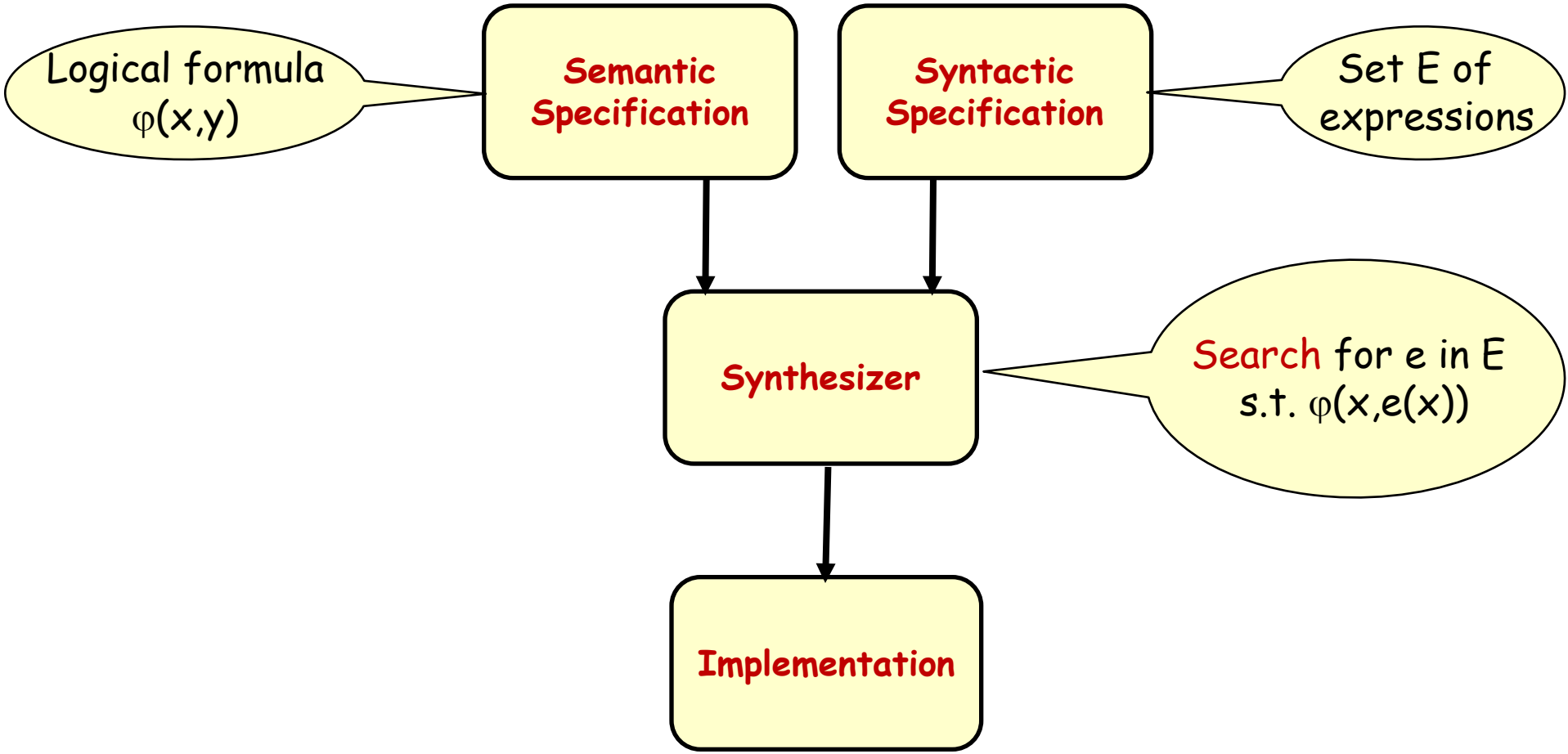
Classical Program Synthesis

Church (1957)



Syntax-Guided Program Synthesis

www.sygus.org



Talk Outline

- Formalization of SyGuS
- Solving SyGuS
- SyGuS Competition and Recent Progress
- Conclusions

Syntax-Guided Program Synthesis

www.sygus.org

- Find a program snippet e such that
  1. e is in a set E of programs (syntactic constraint)
  2. e satisfies logical specification φ (semantic constraint)

- Core computational problem in many synthesis tools/applications

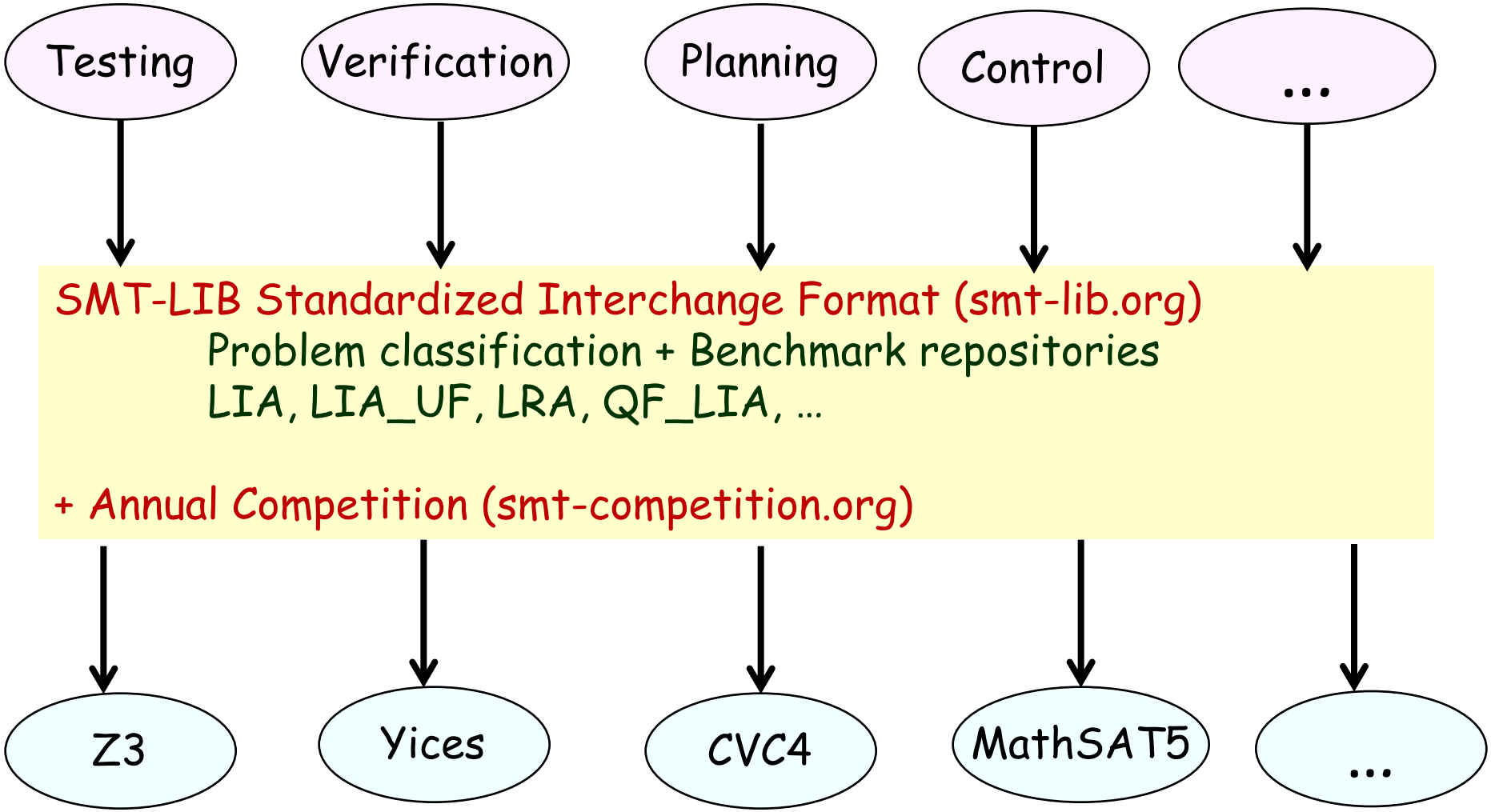
Can we formalize and standardize this computational problem?

Inspiration: Success of SMT solvers in formal verification

SMT: Satisfiability Modulo Theories

- Computational problem: Find a satisfying assignment to a formula
  - Boolean + Int types, logical connectives, arithmetic operators
  - Bit-vectors + bit-manipulation operations in C
  - Boolean + Int types, logical/arithmetic ops + Uninterpreted functs
- "Modulo Theory": Interpretation for symbols is fixed
  - Can use specialized algorithms (e.g. for arithmetic constraints)

SMT Success Story



Syntax-Guided 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)

Syntax-guided synthesis; FMCAD'13

with Bodik, Juniwal, Martin, Raghothaman, Seshia, Singh, Solar-Lezama, Torlak, Udupa

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 (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: Linear expressions
  - LinExp := x_1 | x_2 | Const | LinExp + LinExp | LinExp - LinExp

- No solution exists

SyGuS Example 2

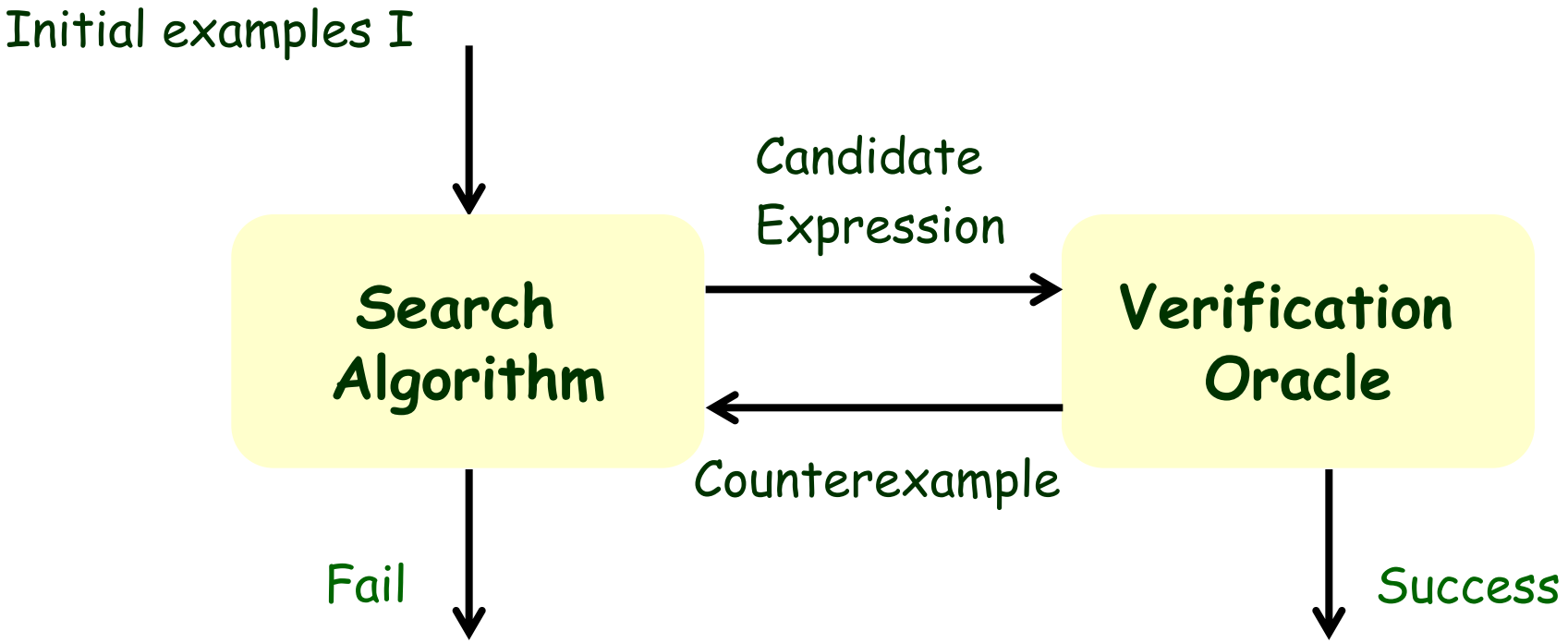
- Theory QF-LIA
- Function to be synthesized: f (int x_1 , x_2) : int
- Specification: $(x_1 \leq f(x_1, x_2)) \;\&\; (x_2 \leq f(x_1, x_2))$
- Candidate Implementations: Conditional expressions without +

  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)

SyGuS as Active Learning



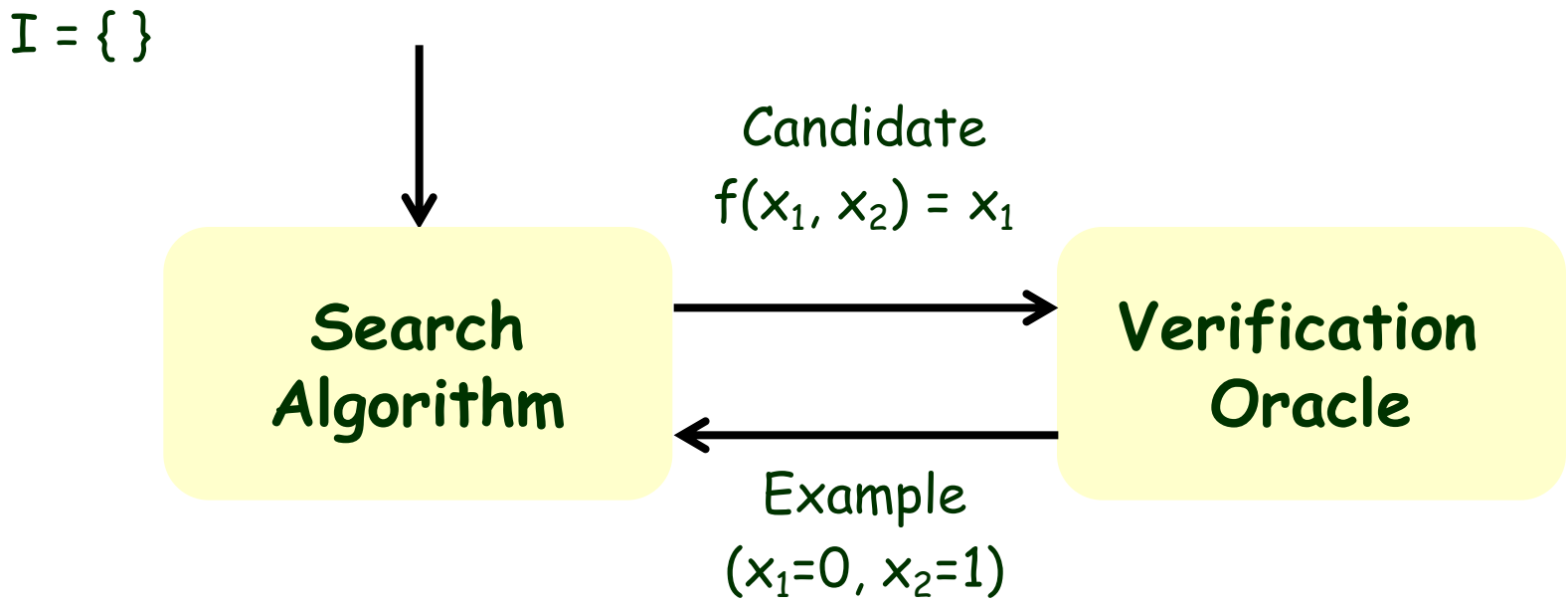
Concept class: Set E of expressions

Examples: Concrete input values

Counterexample-Guided Inductive Synthesis

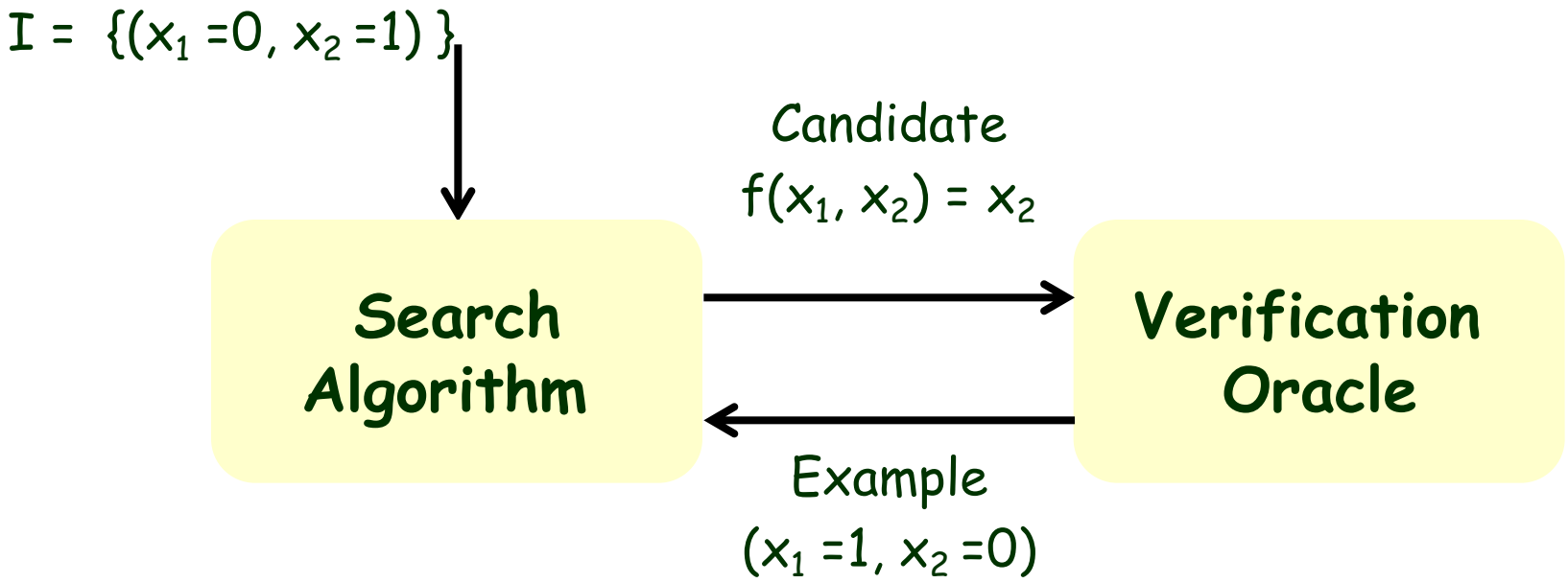
Solar-Lezama et al (ASPLOS'06)

- Specification: $(x_1 \le f(x_1, x_2)) \;\&\; (x_2 \le f(x_1, x_2))$
- Set E: All expressions built from x_1, x_2,0,1, Comparison, If-Then-Else



CEGIS Example

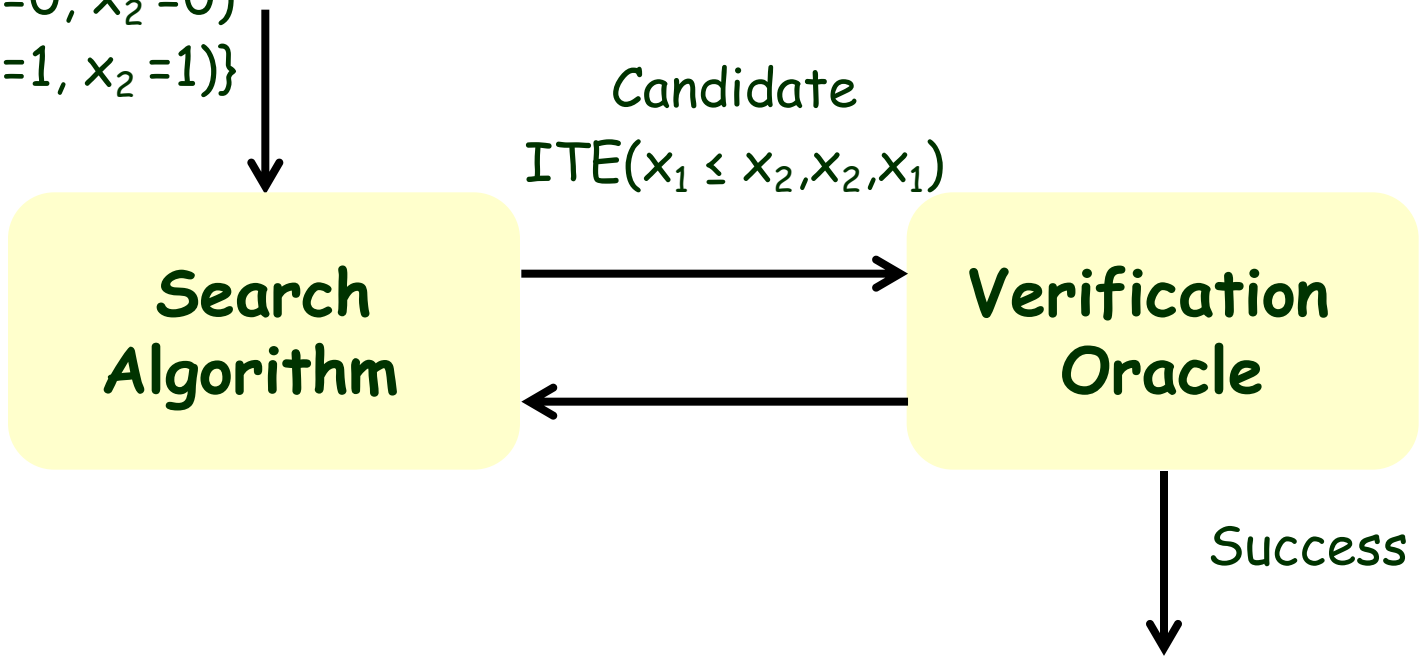
- Specification: $(x_1 \le f(x_1, x_2)) \;\&\; (x_2 \le f(x_1, x_2))$
- Set E: All expressions built from x_1, x_2,0,1, Comparison, If-Then-Else



CEGIS Example

- Specification: $(x_1 \le f(x_1, x_2)) \;\&\; (x_2 \le f(x_1, x_2))$
- Set E: All expressions built from x_1, x_2, 0, 1, Comparison, If-Then-Else

$\{(x_1 = 0, x_2 = 1)$
$(x_1 = 1, x_2 = 0)$
$(x_1 = 0, x_2 = 0)$
$(x_1 = 1, x_2 = 1)\}$



Enumerative Search

- Given:
  - Specification $\varphi(x, f(x))$
  - Grammar for set E of candidate implementations
  - Finite set I of inputs

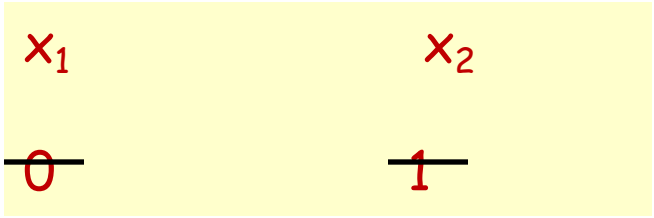
  Find an expression e(x) in E s.t. $\varphi(x,e(x))$ holds for all x in I

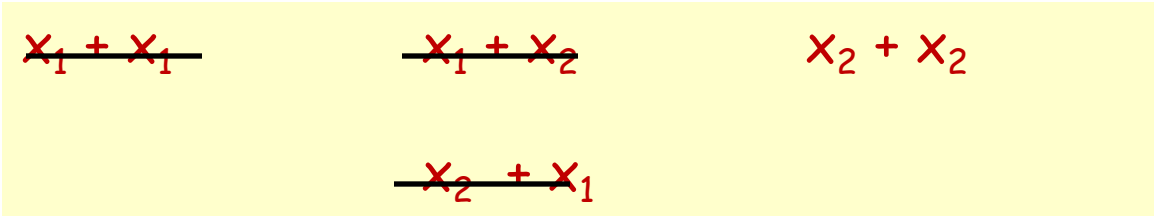
- Attempt 0: Enumerate expressions in E increasing size till you find one that satisfies φ for all inputs in I

- Attempt 1: Pruning of search space based on:
  - Expressions e_1 and e_2 are equivalent
    - if $e_1(x)=e_2(x)$ on all x in I
  - Only one representative among equivalent subexpressions needs to be considered for building larger expressions

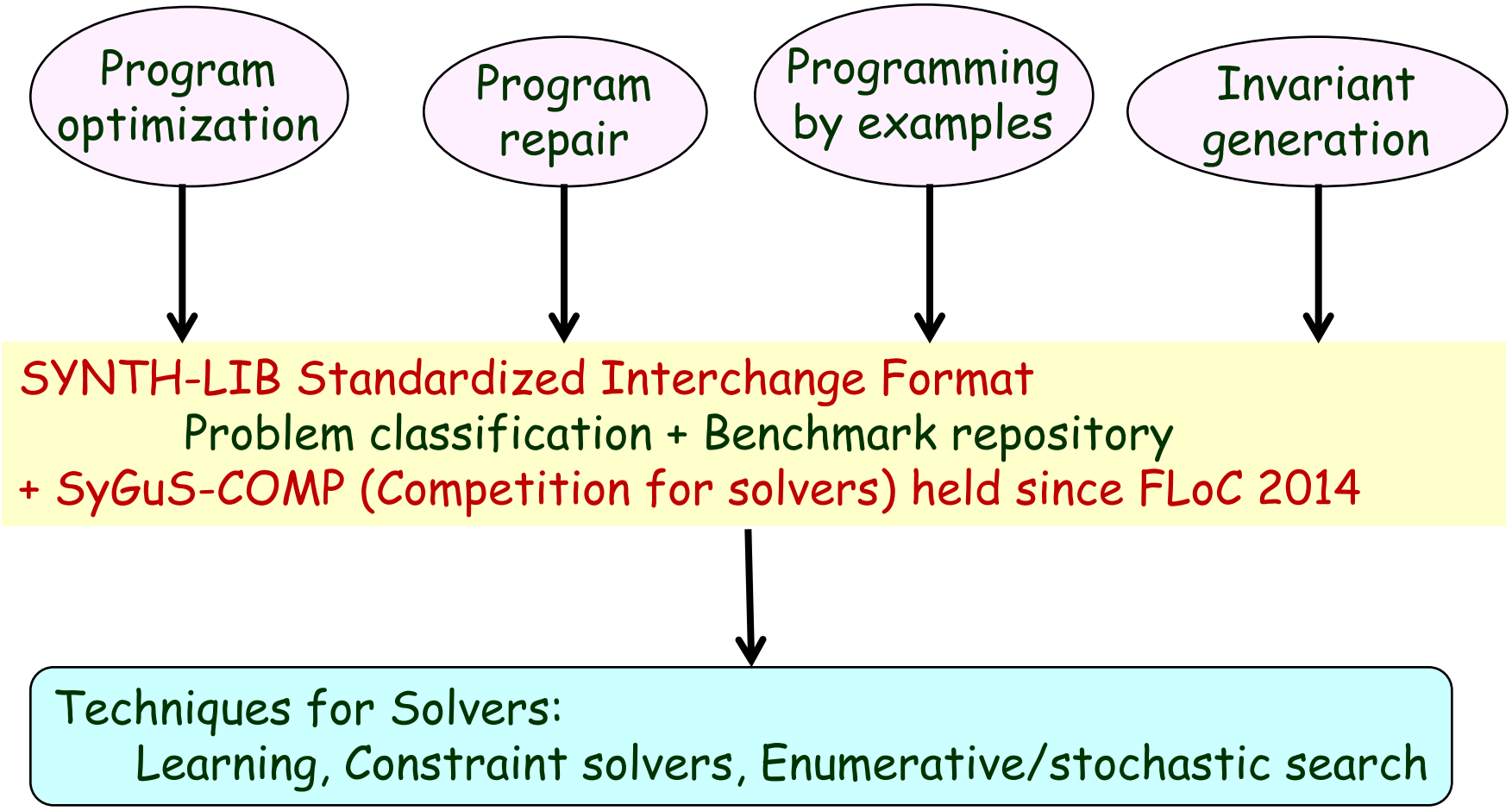
Illustrating Pruning

- Spec: $(x_1 < f(x_1, x_2)) \;\&\; (x_2 < f(x_1, x_2))$
- Grammar: E := x_1 | x_2 | 0 | 1 | E + E
- I = { (x_1=0, x_2=1) }
- Find an expression f such that (f(0,1) > 0) & (f(0,1) > 1)





SyGuS Competition



Collaborators: Fisman, Singh, Solar-Lezama

SyGuS Progress

www.sygus.org

- Over 1500 benchmarks
  - Hacker's delight
  - Invariant generation (based on verification competition SV-Comp)
  - FlashFill (programming by examples system from Microsoft)
  - Synthesis of attack-resilient crypto circuits
  - Program repair
  - Motion planning
  - ICFP programming competition
- Special tracks for competition
  - Invariant generation
  - Programming by examples
  - Conditional linear arithmetic
- New solution strategies and applications

Scaling Enumerative Search by Divide & Conquer

- For the spec $(x_1 \leq f(x_1, x_2))$ & $(x_2 \leq f(x_1, x_2))$ the answer is
  If-Then-Else $(x_1 \leq x_2, x_2, x_1)$

- Size of expressions in conditionals and terms can be much smaller than the size of the entire expression!

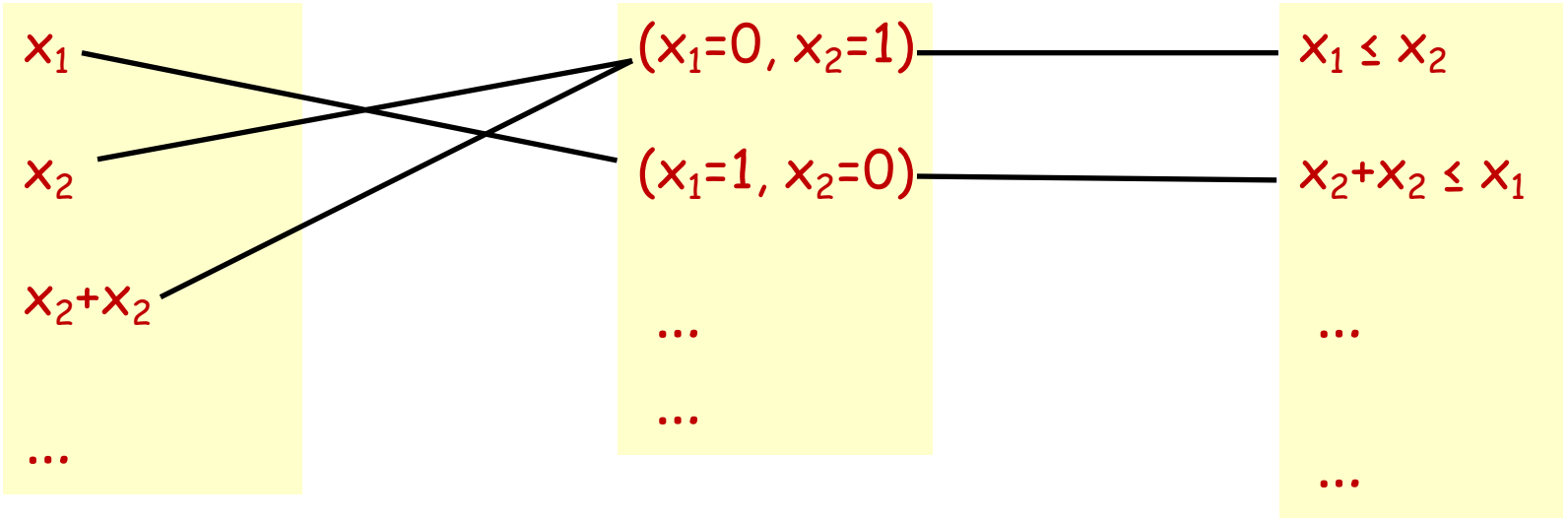
- $f(x_1, x_2) = x_2$ is correct when $x_1 \leq x_2$ and $f(x_1, x_2) = x_1$ is correct otherwise

- Key idea:
  - Generate partial solutions that are correct on subsets of inputs and combine them using conditionals
  - Enumerate terms and tests for conditionals separately
  - Terms and tests are put together using decision tree learning

With A. Radhakrishna and A. Udupa (TACAS 2017)

Enumerative Search with Decision Tree Learning

Expressions / Labels | Inputs / Data points | Predicates / Attributes



Input x labeled with expression e
if $\varphi(x, e(x))$ holds

Input x has attribute p
if p(x) holds

Desired decision tree:

Internal nodes: predicates + Leaves : expressions

Acceleration Using Learned Probabilistic Models

- Can we bias the search towards likely programs?

- Step 1: Mine existing solutions to convert given grammar into a probabilistic higher-order grammar
  - Weighted production rules
  - Conditioned on parent and sibling context
  - Transfer learning used to avoid overfitting

- Step 2: Enumerative search to generate expressions in decreasing likelihood
  - Use A* with cost estimation heuristic
  - Integrated with previous optimizations (equivalence-based pruning…)

With W. Lee, K. Heo, and M. Naik (PLDI 2018)

Experimental Evaluation

- 2017 SyGuS Competition
  - Over 1500 benchmarks in different categories
  - Solution size:
    - about 20 AST nodes in string manipulation programs
    - upto 1000 AST nodes in bitvector manipulation programs
  - Number of participating solvers: 8
- State of the art solver: Euphony
  - Enumerative + Decision trees + Learned probabilistic models
- Evaluation of Euphony
  - 70% of all benchmarks solved with a time limit of 1 hour
  - Average time ~ 10 min
  - Median time ~ 2 min

2018 Winner : CVC4 (Reynolds et al):
Integration of enumerative search with constraint solving !!

Emerging Applications of SyGuS

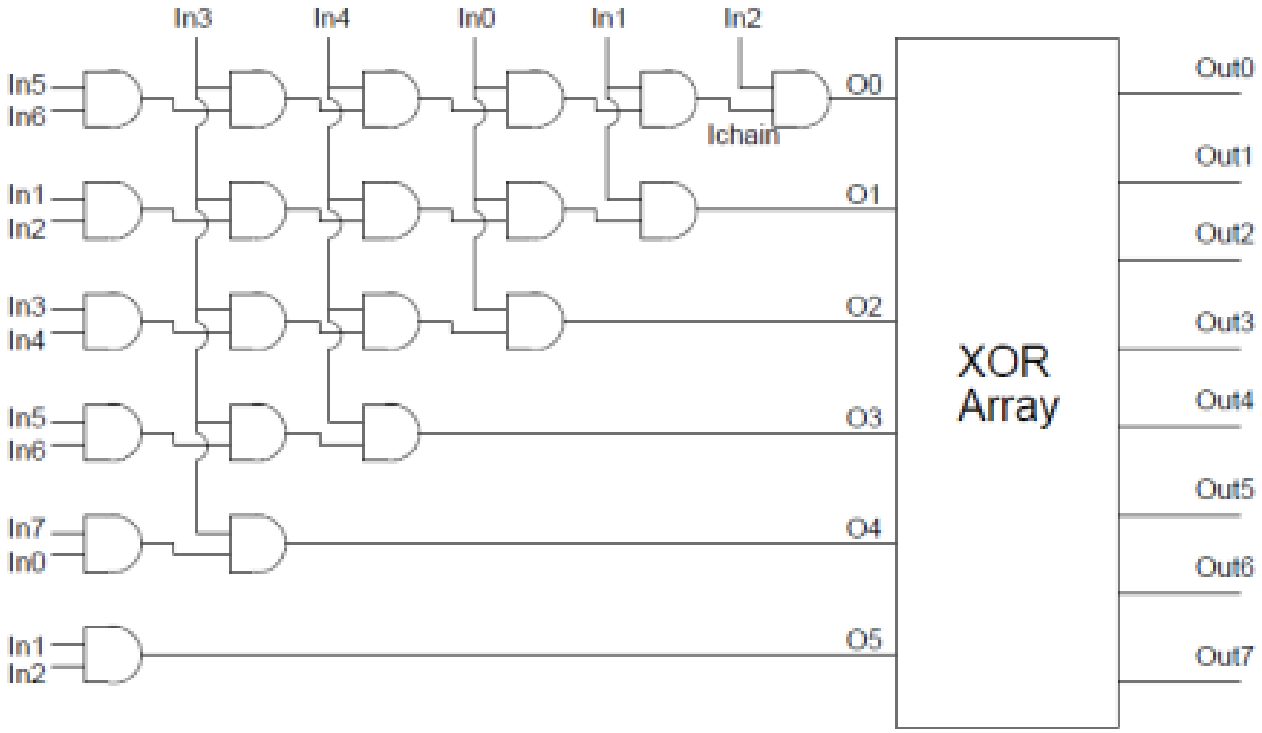
- Synthesis of crypto-circuits resilient to timing attack
  (Wang et al, CAV 2016)

- Solving of quantified formulas in SMT solvers
  (Biere et al, TACAS 2017)
  To solve For all x. Exists y. $\varphi(x,y)$
  synthesize Skolem function f(x) such that For all x. $\varphi(x,f(x))$

- Improved solver for bit-vector arithmetic in CVC4
  (Barrett et al, CAV 2018)
  Automatic generation of side conditions for bit-vector rewriting

- Automatic inversion of list manipulating programs
  (Hu and D'Antoni, PLDI 2018)
  Modeled as symbolic transducers and applied to string encoders

Back to Synthesis of Attack Countermeasures

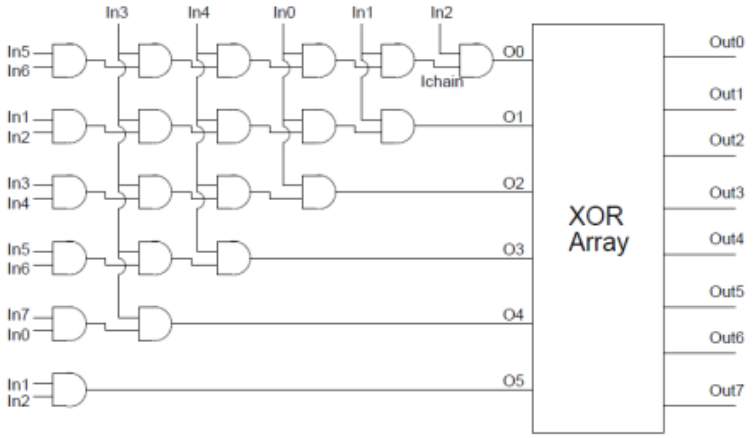


Given a circuit *C*, automatically synthesize a circuit *C'* such that

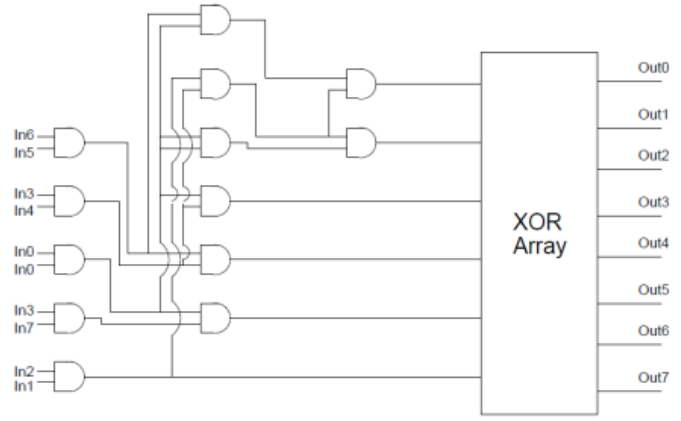
1. *C'* is functionally equivalent to *C* [sematic constraint]
2. All input-to-output paths in *C'* have same length [syntactic constraint]

Can be encoded directly as a SyGuS problem (Wang et al, CAV'16)

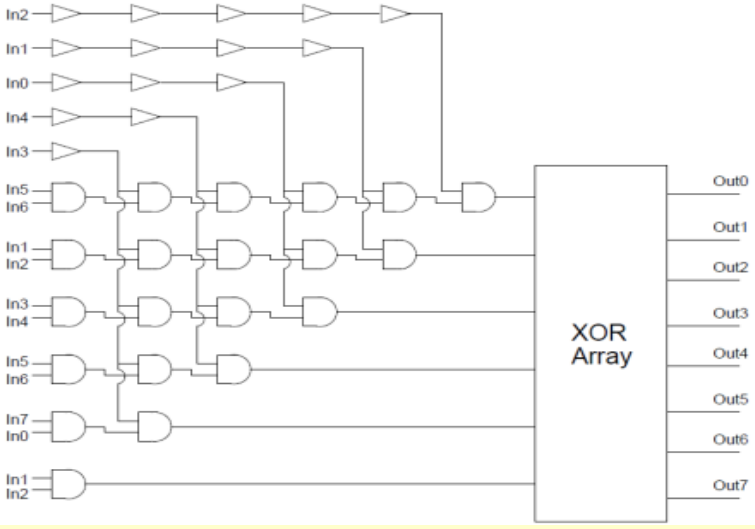
SyGuS Result



Original ckt prone to attack



SyGuS-generated Attack resilient ckt



Hand-crafted attack resilient ckt

Fully automatic
Smaller size
Shorter delays

SyGuS Conclusions

www.sygus.org

- Problem definition
  - Syntactic constraint on space of allowed programs
  - Semantic constraint given by logical formula
- Solution strategies
  - Counterexample-guided inductive synthesis
  - Search in program space + Verification of candidate solutions
- Applications
  - Programming by examples
  - Program optimization with respect to syntactic constraints
- Annual competition (SyGuS-comp)
  - Standardized interchange format + benchmarks repository

Program Synthesis: Future

- Can search-based synthesis scale?
  - Many unexplored opportunities to exploit program structure
  - Highly parallelizable
  - Computationally hard analysis problems such as model checking, constraint solving were considered hopeless at the beginning
- How to integrate synthesis in programming environments ?
  - Synthesis tool can suggest code completions
  - User interaction model is key
  - Integration in next-generation compilers
- Relationship to machine learning ?

Learning to Program

- How can machine learning help program synthesis ?
  - Already discussed: decision trees, probabilistic models of code
- Programming by examples: can we train a neural network ?
  - Challenges: very few examples, program space far from continuous
  - Illustrative effort: Neural Flashfill (Microsoft)
- Can we mine code bases to suggest program completions
  - DARPA MUSE program
  - Illustrative effort: Bayou (Chaudhuri et al) for prediction of API usage in Java code via Bayesian inference

Program Synthesis to Aid ML

- Can program synthesis help in design of ML systems?
  - Illustrative effort (Google Brain): Use syntax-guided synthesis to generate script of API calls for TensorFlow programs

- Can program verification/synthesis contribute to "explainable AI"?
  - Synthesize logical input-output relationships for trained neural networks
  - Synthesize adversarial test inputs to check robustness of neural networks

Goal: Programming computers easier than communicating with people