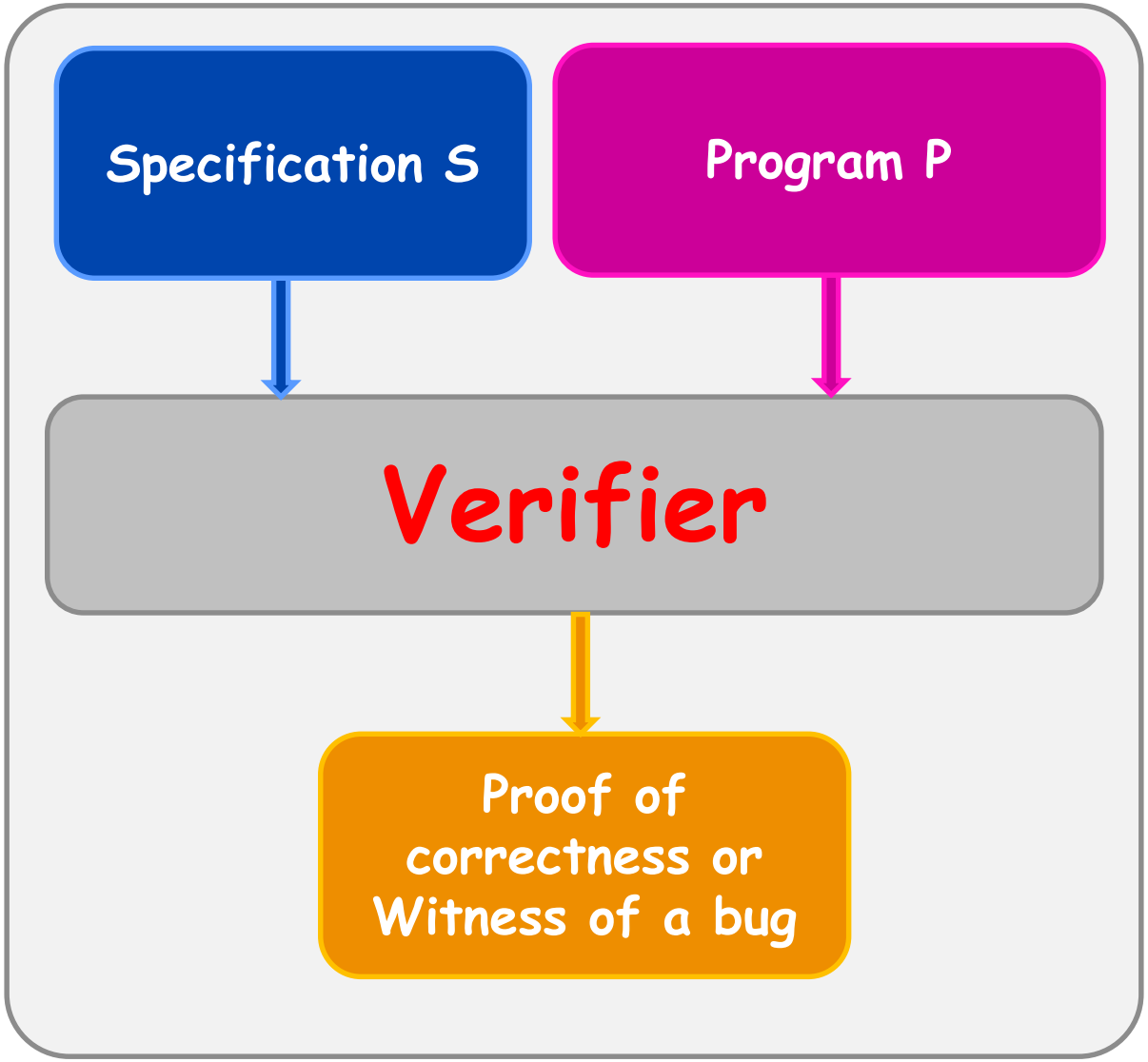
Syntax-Guided Synthesis

Rajeev Alur

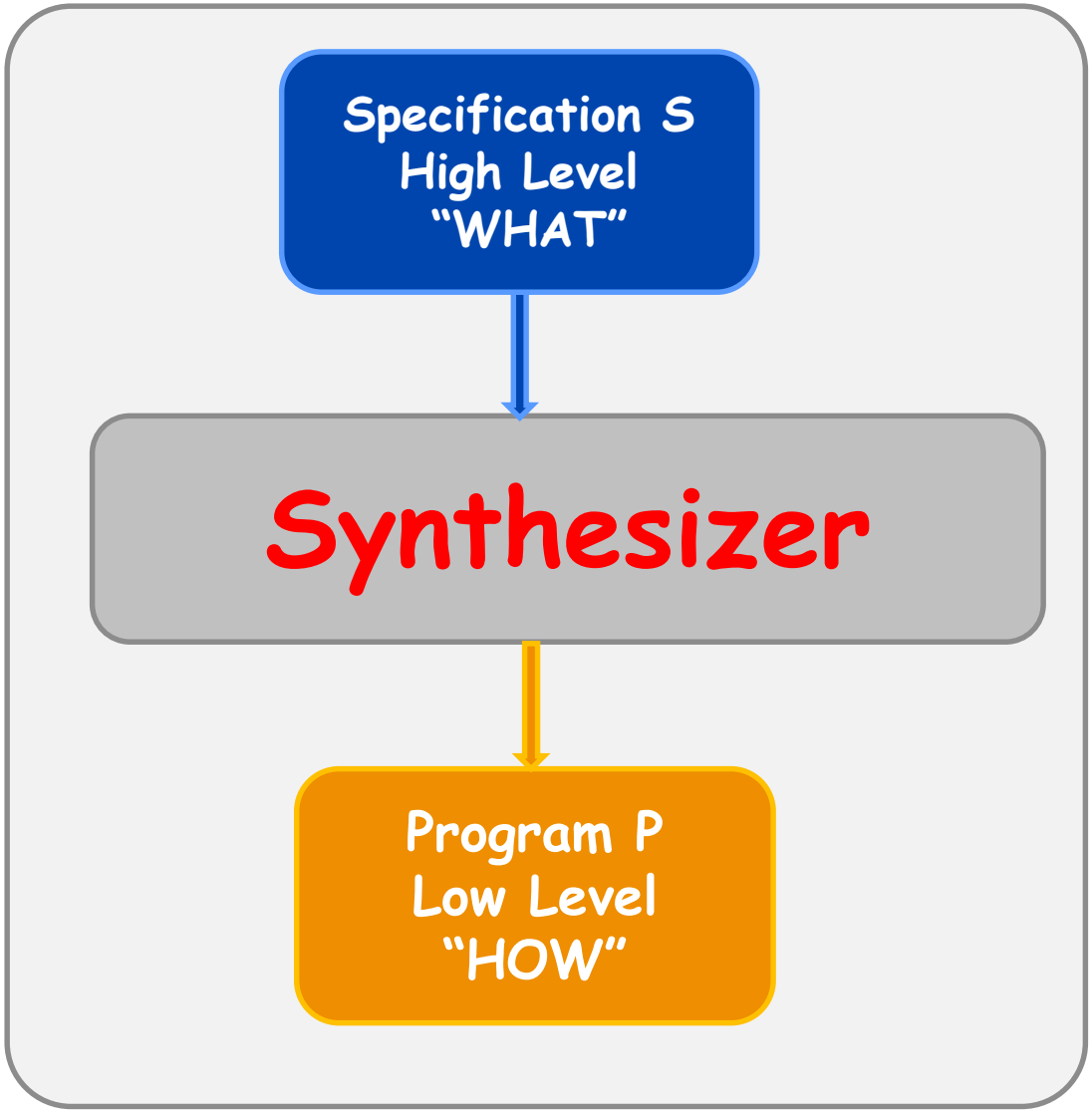
University of Pennsylvania

ExCAPE
Expeditions in Computer Augmented Program Engineering

Program Verification

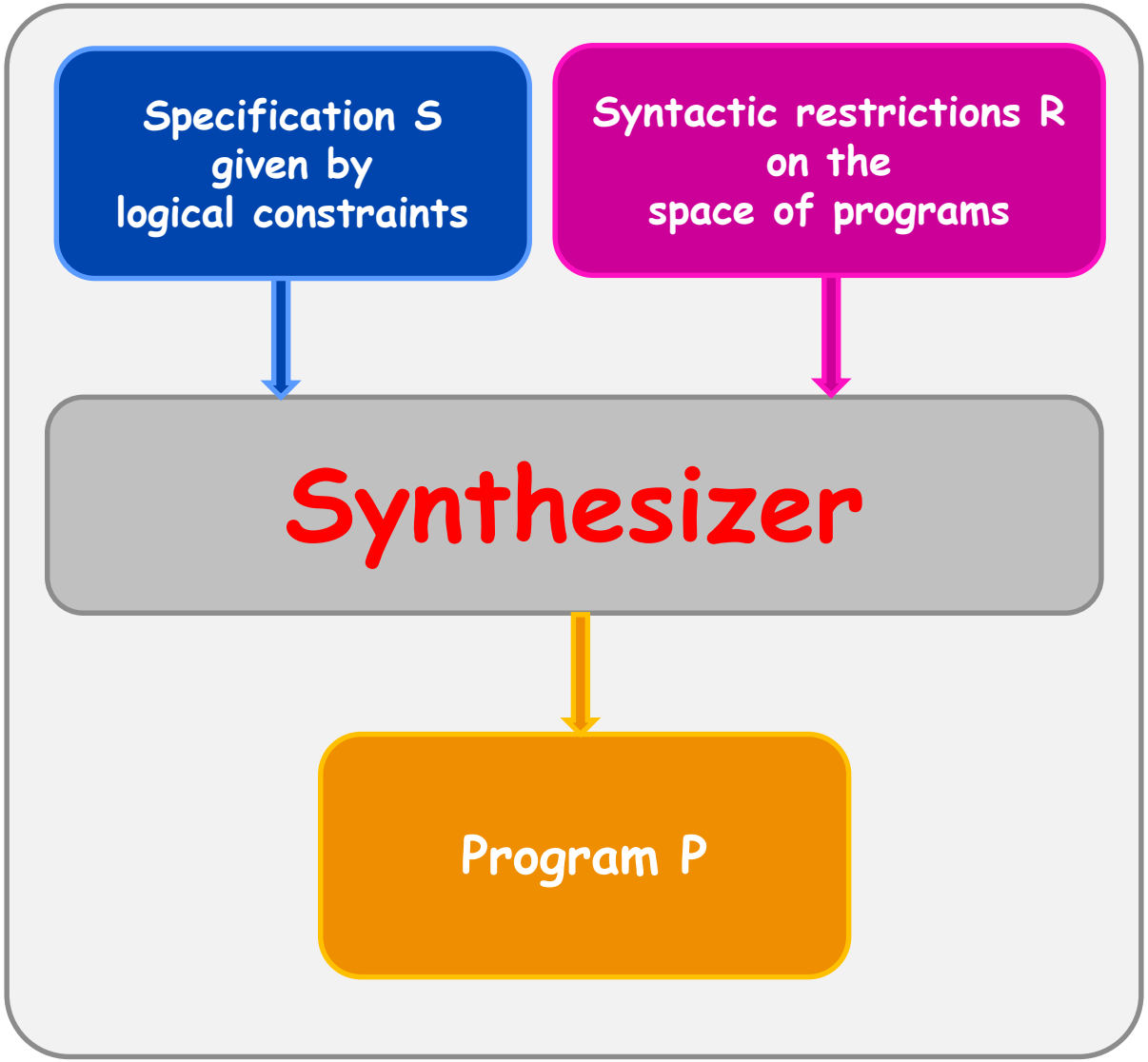


Classical Program Synthesis



Syntax-Guided Synthesis

www.sygus.org



Outline

- ❖ Motivating Examples
- ❑ Formalization of SyGuS
- ❑ Solving SyGuS
- ❑ SyGuS Competition and Conclusions

Syntax-Guided Program Synthesis

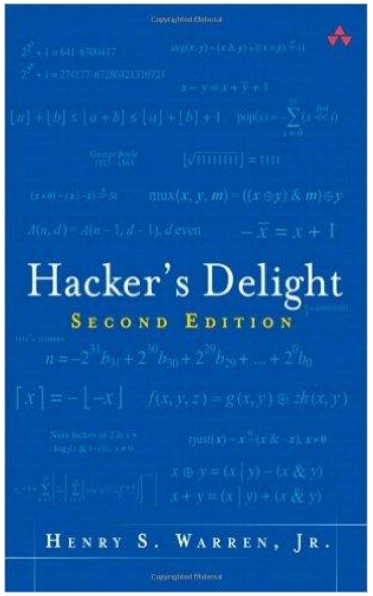
www.sygus.org

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

- Core computational problem with many applications
  - Programming by examples
  - Automatic program repair
  - Program superoptimization
  - Template-guided invariant generation
  - Autograding for programming assignments
  - Synthesis of FSA-attack-resilient cryptographic circuits

Programming By Examples

- Find a program P for bit-vector transformation such that
  1. P is constructed from standard bit-vector operations
     |, &, ~, +, -, <<, >>, 0, 1, …
  2. P is consistent with the following input-output examples
     00101 → 00100
     01010 → 01000
     10110 → 10000



- Resets rightmost substring of contiguous 1's to 0's

- Desired solution:
  x & (1 + (x | (x-1))

FlashFill: Programming by Examples

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.

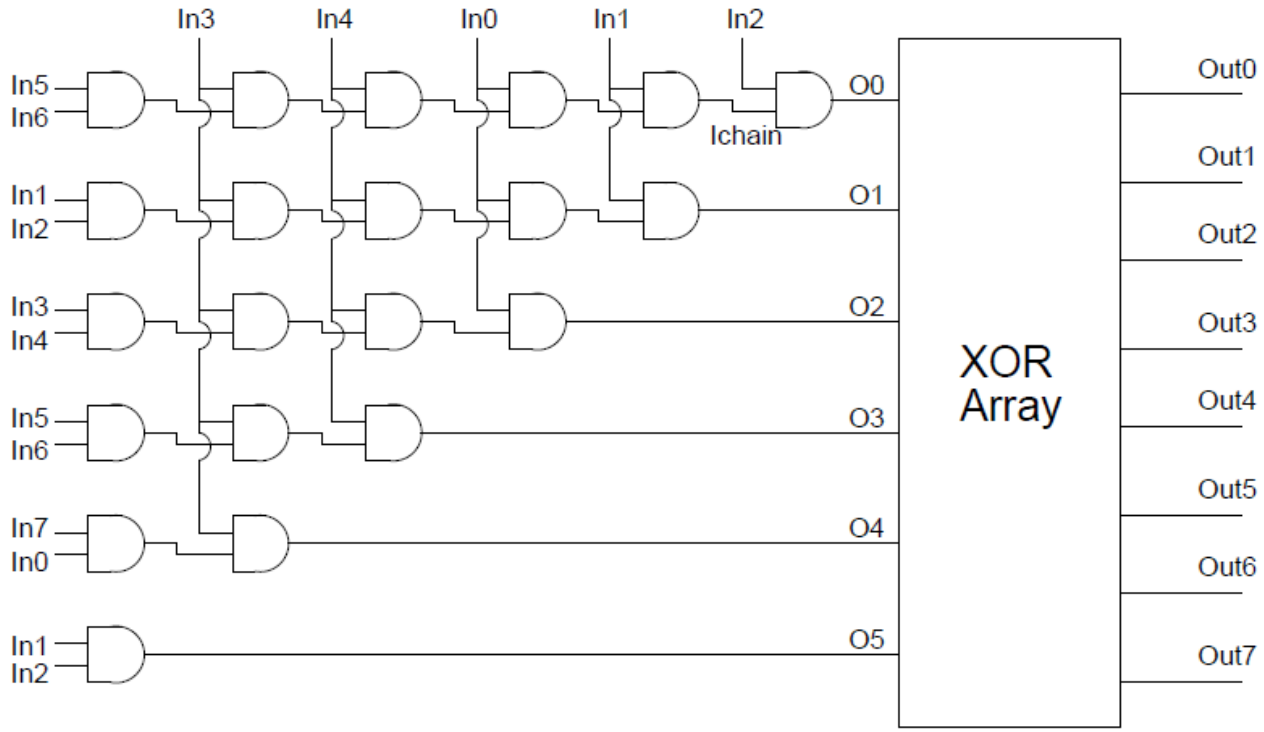
Superoptimizing Compiler

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

```
multiply (x[1,n], y[1,n]) {
  x1 = x[1,n/2];
  x2 = x[n/2+1, n];
  y1 = y[1, n/2];
  y2 = y[n/2+1, n];
  a = x1 * y1;
  b = shift( x1 * y2, n/2);
  c = shift( x2 * y1, n/2);
  d = shift( x2 * y2, n);
  return ( a + b + c + d)
}
```

Replace with equivalent code with only 3 multiplications

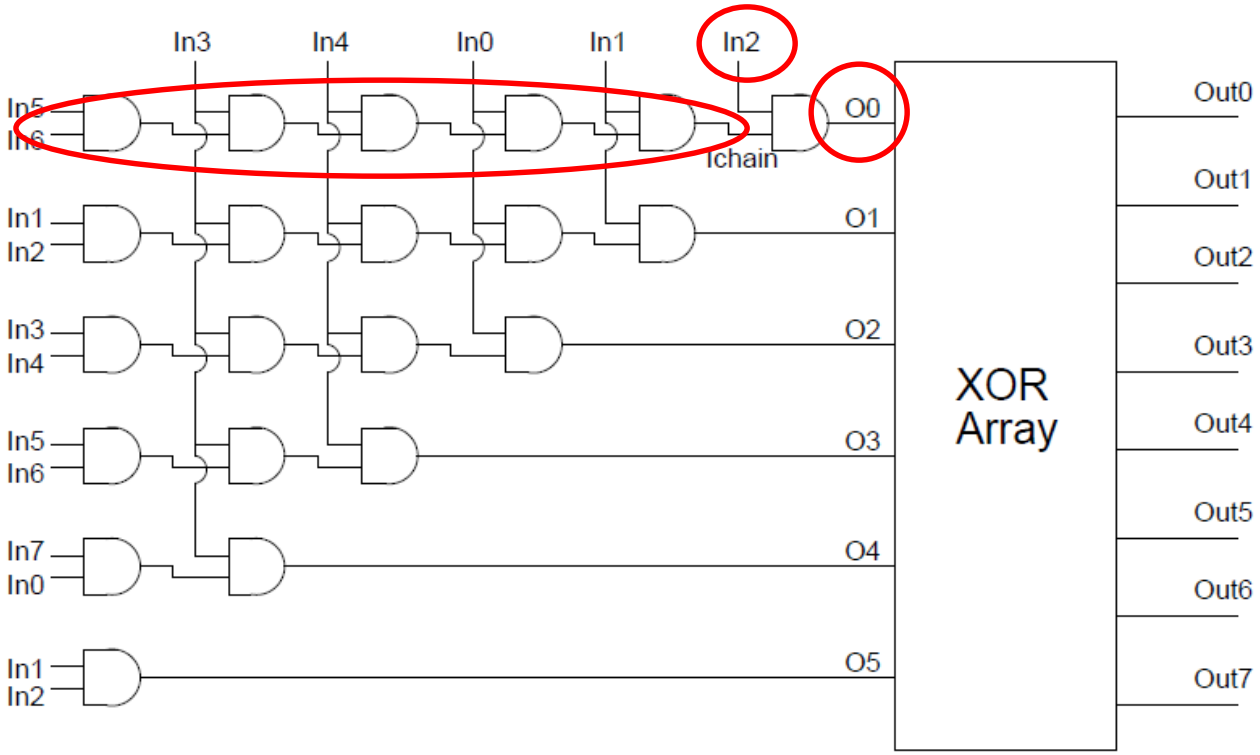
Side Channel Attacks on Cryptographic Circuits



PPRM1 AES S-box implementation [Morioka & Satoh, in CHES 2002]

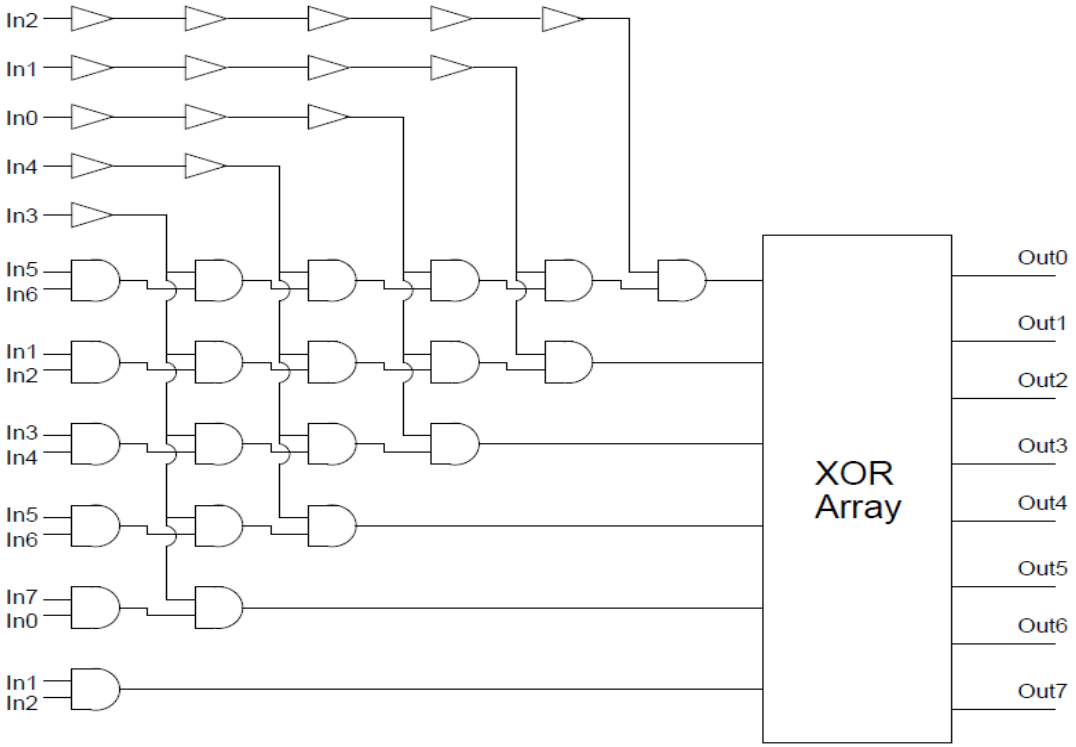
1. **The only non-linear function in Advanced Encryption Standard algorithm**
2. **Vulnerable to Fault Sensitivity Analysis attack**

Side Channel Attacks on Cryptographic Circuits



Time at which O_0 changes is different when In2=0 vs In2=1
Consequence: Timing-based attack can reveal secret input In2

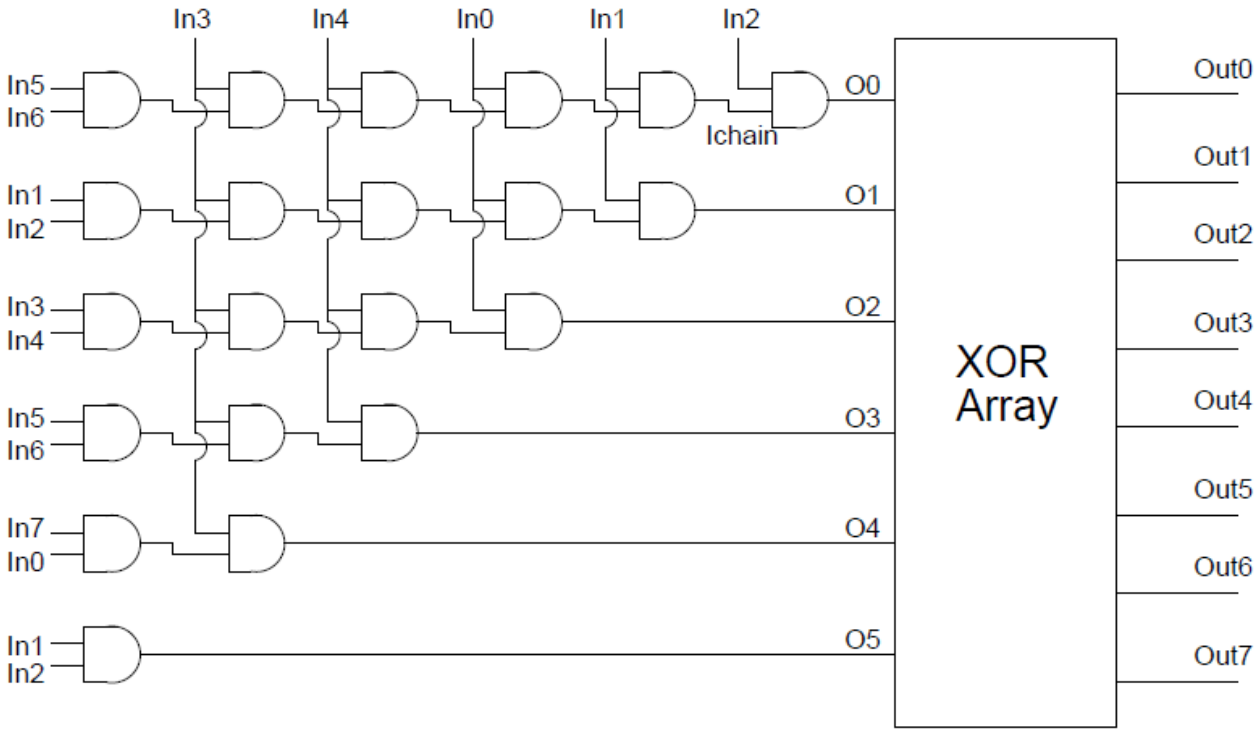
Countermeasure to Attack



FSA attack resilient ckt: All input-to-output paths have same delays
Manually hand-crafted solution [Schaumont et al, DATE 2014]

Verification problem: Is attack resilient ckt equivalent to original?

Synthesis of Attack Countermeasures



Given a ckt C, automatically synthesize a ckt 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]

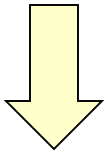
Existing EDA tools cannot handle this synthesis problem

Autograder: Feedback on Programming Homeworks

Singh et al (PLDI 2013)

```
1  def computeDeriv(poly):
2      deriv = []
3      zero = 0
4      if (len(poly)==1):
5          return deriv
6      for e in range(0,len(poly)):
7          if(poly[e]==0):
8              zero += 1
9          else:
10             deriv.append(poly[e]*e)
11
12     return deriv
```

Student Solution P
+ Reference Solution R
+ Error Model



The program requires **3** changes:

- In the return statement **return deriv** in **line 5**, replace **deriv** by **[0]**.
- In the comparison expression **(poly[e] == 0)** in **line 7**, change **(poly[e] == 0)** to **False**.
- In the expression **range(0, len(poly))** in **line 6**, replace **0** by **1**.

Find min no of edits to P so as to make it equivalent to R

Automatic Invariant Generation

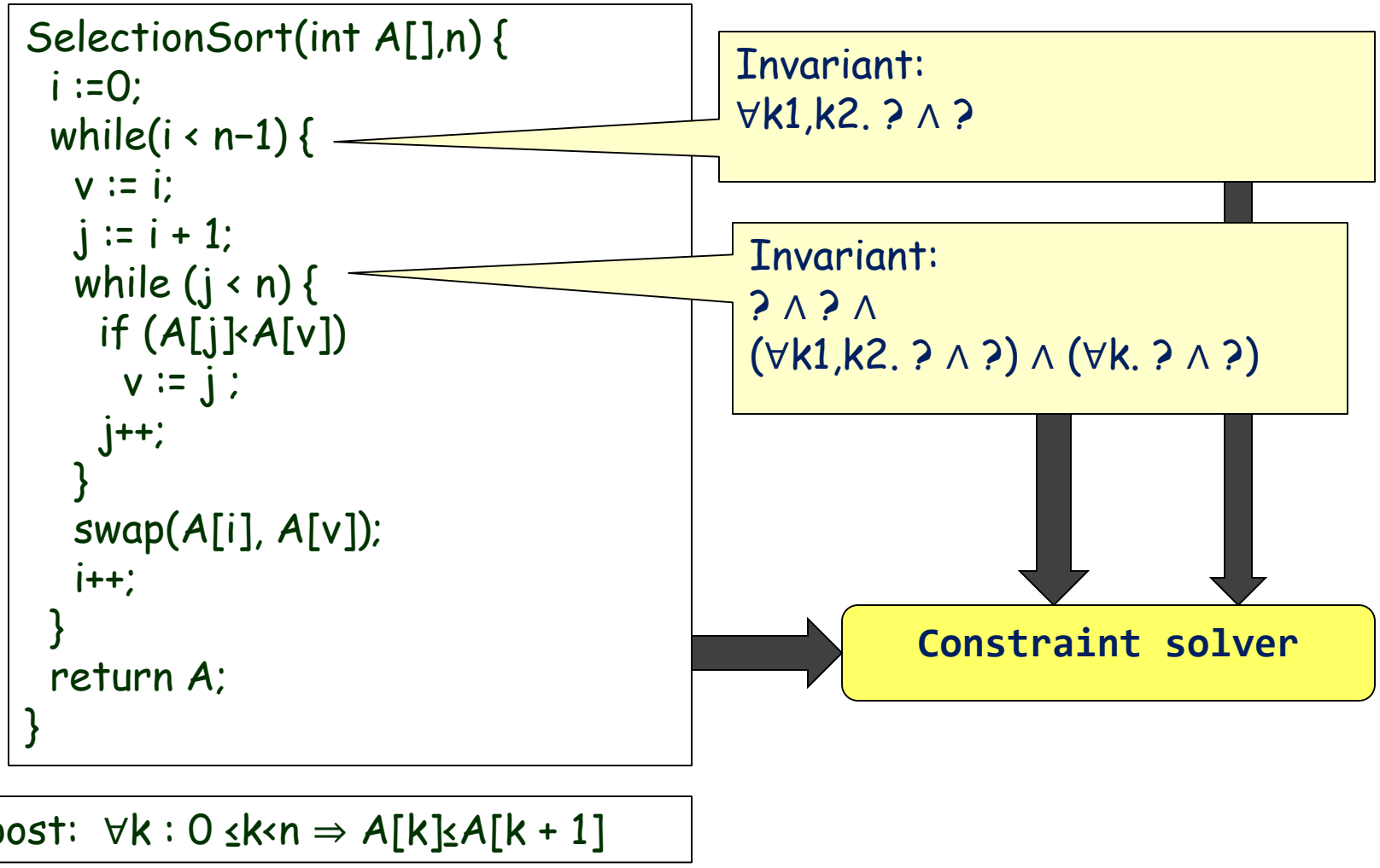
```
SelectionSort(int A[],n) {
  i := 0;
  while(i < n−1) {
    v := i;
    j := i + 1;
    while (j < n) {
      if (A[j]<A[v])
        v := j ;
      j++;
    }
    swap(A[i], A[v]);
    i++;
  }
  return A;
}
```

Invariant: ?

Invariant: ?

post: $\forall k : 0 \leq k < n \Rightarrow A[k] \leq A[k+1]$

Template-based Automatic Invariant Generation



Template-based Automatic Invariant Generation

```
SelectionSort(int A[],n) {
  i :=0;
  while(i < n−1) {
    v := i;
    j := i + 1;
    while (j < n) {
      if (A[j]<A[v])
        v := j ;
      j++;
    }
    swap(A[i], A[v]);
    i++;
  }
  return A;
}
```

Invariant:
$\forall k1,k2.\ 0 \le k1 < k2 < n \wedge$
$k1 < i \Rightarrow A[k1] \le A[k2]$

Invariant:
$i < j \wedge$
$i \le v < n \wedge$
$(\forall k1,k2.\ 0 \le k1 < k2 < n \wedge$
$k1 < i \Rightarrow A[k1] \le A[k2]) \wedge$
$(\forall k.\ i1 \le k < j \wedge$
$k \ge 0 \Rightarrow A[v] \le A[k])$

post: $\forall k : 0 \le k < n \Rightarrow A[k] \le A[k+1]$

Syntax-Guided Program Synthesis

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

- Core computational problem with many applications
  - Programming by examples
  - Automatic program repair
  - Program superoptimization
  - Template-guided invariant generation
  - Autograding for programming assignments
  - Synthesis of FSA-attack-resilient cryptographic circuits

Can we formalize and standardize this computational problem?

Inspiration: Success of SMT solvers in formal verification

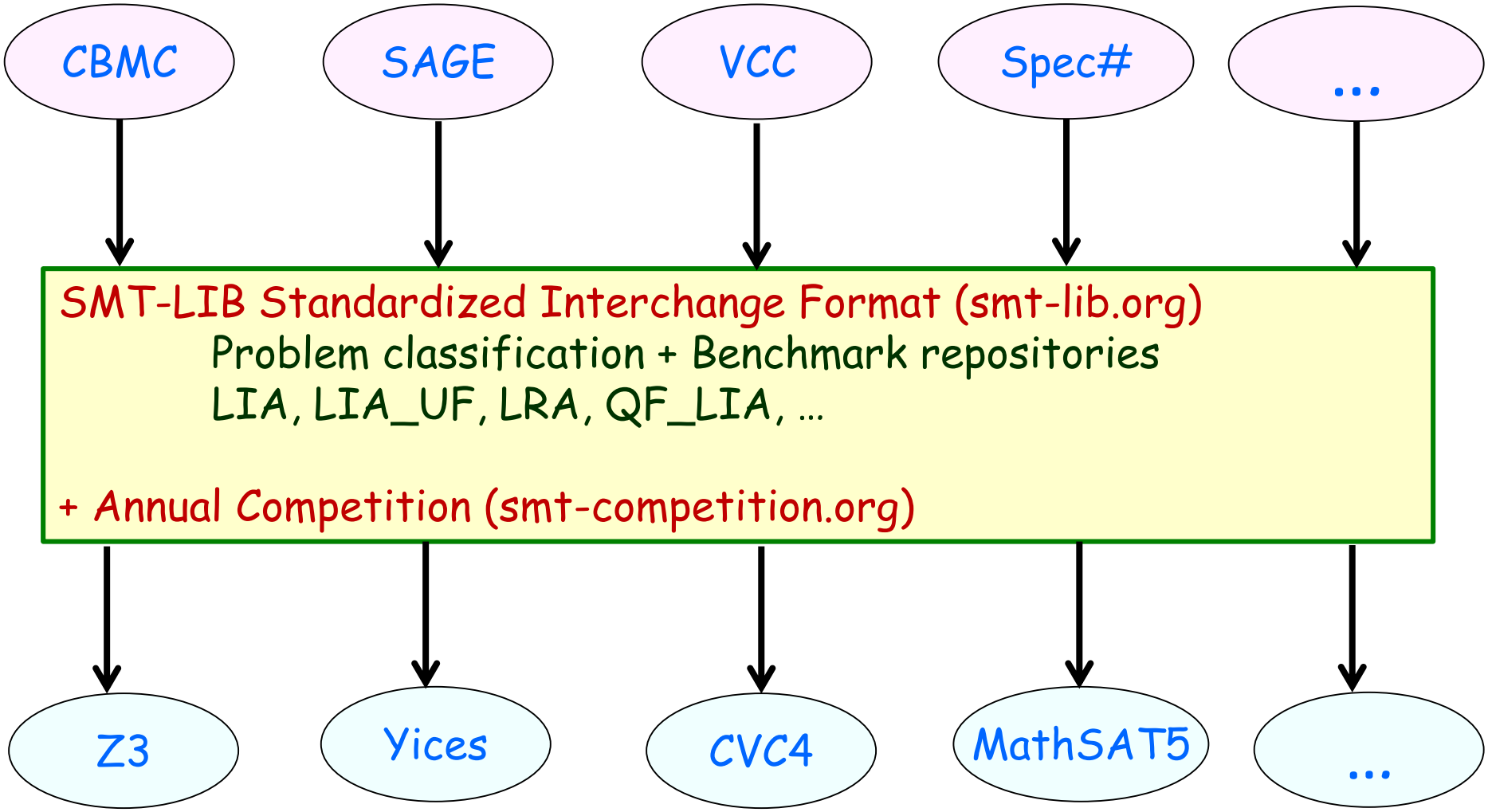
SMT: Satisfiability Modulo Theories

- 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)

Little Engines of Proof

SAT; Linear arithmetic; Congruence closure

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 φ
    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

- 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, int y) : int

- Specification: $(x \leq f(x,y)) \ \& \ (y \leq f(x,y))$

- Candidate Implementations: Linear expressions
  - LinExp := x | y | Const | LinExp + LinExp | LinExp - LinExp

- No solution exists

SyGuS Example

- Theory QF-LIA
- Function to be synthesized: f (int x, int y) : int
- Specification: (x ≤ f(x,y)) & (y ≤ f(x,y))
- Candidate Implementations: Conditional expressions without +

  Term := x | y | Const | If-Then-Else (Cond, Term, Term)
  Cond := Term <= Term | Cond & Cond | ~ Cond | (Cond)

- Possible solution:

  If-Then-Else (x ≤ y, y, x)

From SMT-LIB to SYNTH-LIB

www.sygus.org

```
(set-logic LIA)
(synth-fun max2 ((x Int) (y Int)) Int
    ((Start Int (x y 0 1
                 (+ Start Start)
                 (- Start Start)
                 (ite StartBool Start Start)))
     (StartBool Bool ((and StartBool StartBool)
                      (or StartBool StartBool)
                      (not StartBool)
                      (<= Start Start))))
 (declare-var x Int)
 (declare-var y Int)
 (constraint (>= (max2 x y) x))
 (constraint (>= (max2 x y) y))
 (constraint (or (= x (max2 x y)) (= y (max2 x y))))
 (check-synth)
```

Let Expressions and Auxiliary Variables

- Synthesized expression maps directly to a straight-line program
- Grammar derivations correspond to expression parse-trees
- How to capture common subexpressions (which map to aux vars) ?
- Solution: Allow "let" expressions
- Candidate-expressions for a function f(int x, int y) : int

  T := (let [z = U] in z + z)

  U := x | y | Const | (U) | U + U | U - U

Invariant Generation as SyGuS

```
bool x, y, z
int  a, b, c

while( Test ) {
   loop-body
   ....
}
```

- Goal: Find inductive loop invariant automatically
- Function to be synthesized

  Inv (bool x, bool z, int a, int b) : bool
- Compile loop-body into a logical predicate

  Body(x,y,z,a,b,c, x',y',z',a',b',c')
- Specification:

  (Inv & Body & Test') ⇒ Inv'

  & Pre ⇒ Inv & (Inv & ~Test ⇒ Post)
- Template for set of candidate invariants

  Term := a | b | Const | Term + Term | If-Then-Else (Cond, Term, Term)

  Cond := x | z | Cond & Cond | ~ Cond | (Cond)

Program Optimization as SyGuS

- Type matrix: 2x2 Matrix with Bit-vector[32] entries
  Theory: Bit-vectors with arithmetic

- Function to be synthesized f(matrix A, B) : matrix

- Specification: f(A,B) is matrix product
  f(A,B)[1,1] = A[1,1]*B[1,1] + A[1,2]*B[2,1]
  …
- Set of candidate implementations
  Expressions with at most 7 occurrences of *
  Unrestricted use of +
  let expressions allowed

- Benefit of saving this one multiplication: Strassen's $O(n^{2.87})$ algorithm for matrix multiplication

- Can we use only 6 multiplication operations?

Optimality

- Specification for f(int x) : int

  $x \leq f(x)$ & $-x \leq f(x)$

- Set E of implementations: Conditional linear expressions

- Multiple solutions are possible

  If-Then-Else ($0 \leq x$, x, 0)

  If-Then-Else ($0 \leq x$, x, $-x$)

- Which solution should we prefer?

  Need a way to rank solutions (e.g. size of parse tree)

Synthesis Puzzle: Cinderella v. stepmother

There are five buckets arranged in a circle. Each bucket can hold upto B liters of water. Initially all buckets are empty. The wicked stepmother and Cinderella take turns playing the following game:

Stepmother brings 1 liter of additional water and splits it into 5 buckets.

If any of the buckets overflows, stepmother wins the game.

If not, Cinderella gets to empty two adjacent buckets. If the game goes on forever, Cinderella wins.

Find B* such that if B < B* the stepmother has a winning strategy, and if B = B*, Cinderella has a winning strategy.

And give a proof that your strategies work!

Reference: Bodlaender et al, IFIP TCS 2012

Stepmother wins if B<2

Round 1:

Stepmother: Add 0.5 lit to buckets 1 and 3

Cinderella: Empty one of the buckets, say third

Round 2:

Stepmother: Add 0.25 lit to bucket 1 and 0.75 lit to bucket 3

Cinderella: Empty bucket 3

…

After n rounds, bucket 1 contains $1 - 1/2^n$ lit of water

If B < 2, then after some N rounds bucket 1 contains more than B-1 lit of water, stepmother can win in $(N+1)^{th}$ round by adding 1 lit to it

Cinderella wins if B=2

Cinderella maintains the following invariant:

(a1 + a3 < 1) & (a2 <= 1) & (a4 = 0) & (a5 = 0)

a1, a2, a3, a4, a5: water quantities starting at some bucket

If this condition holds after n rounds, stepmother cannot win in the next round. Thus, if this is an invariant, then Cinderella wins.

Invariant holds initially.

Assume the invariant holds at the beginning of a round.

Goal: Cinderella can enforce the invariant, no matter what the stepmother does, after her own turn.

Cinderella wins if B=2

At the beginning of the round, we have:

$(a1 + a3 < 1) \;\&\; (a2 <= 1) \;\&\; (a4 = 0) \;\&\; (a5 = 0)$

b1, b2, b3, b4, b5: water quantities after stepmother's turn

Claim: $b1 + b3 + b4 + b5 < 2$

Either $(b1 + b4 < 1)$ or $(b3 + b5 < 1)$

Suppose $(b1 + b4 < 1)$. Other case similar.

Cinderella strategy: empty buckets 2 and 3.
We have: $(b4 + b1 < 1) \;\&\; (b5 <= 1) \;\&\; (b2 = 0) \;\&\; (b3 = 0)$

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 φ
    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)

Solving SyGuS

- Is SyGuS same as solving SMT formulas with quantifier alternation?

- SyGuS can sometimes be reduced to Quantified-SMT, but not always
  - Set E is all linear expressions over input vars x, y

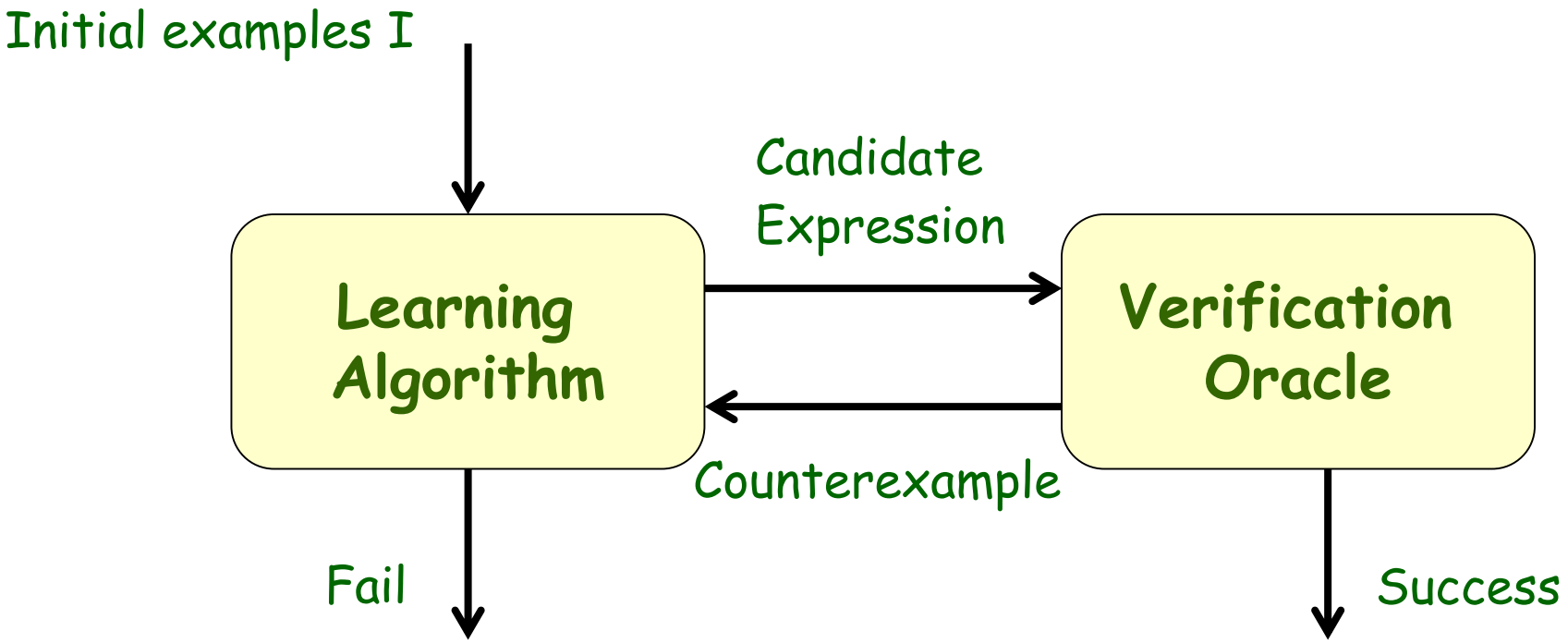
    SyGuS reduces to Exists a,b,c. Forall X. φ [f/ ax+by+c]
  - Set E is all conditional expressions

    SyGuS cannot be reduced to deciding a formula in LIA

- Syntactic structure of the set E of candidate implementations can be used effectively by a solver

- Existing work on solving Quantified-SMT formulas suggests solution strategies for SyGuS

SyGuS as Active Learning

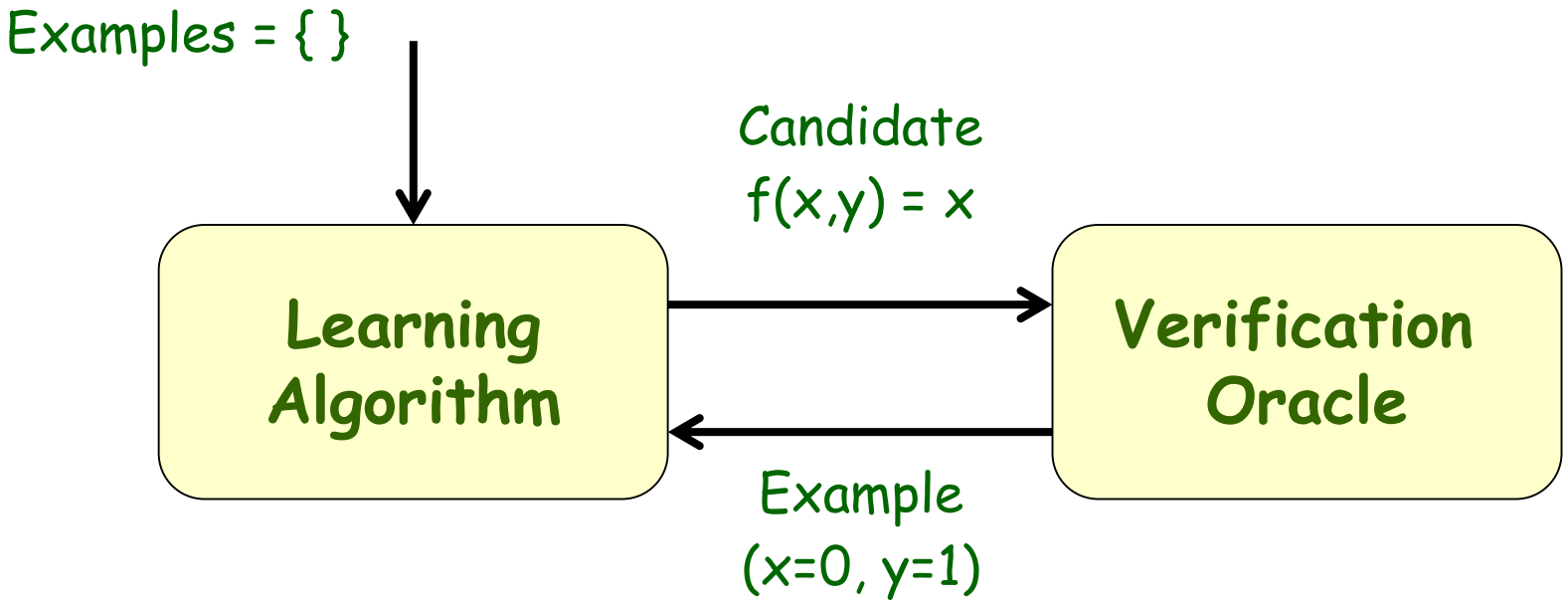


Concept class: Set E of expressions

Examples: Concrete input values

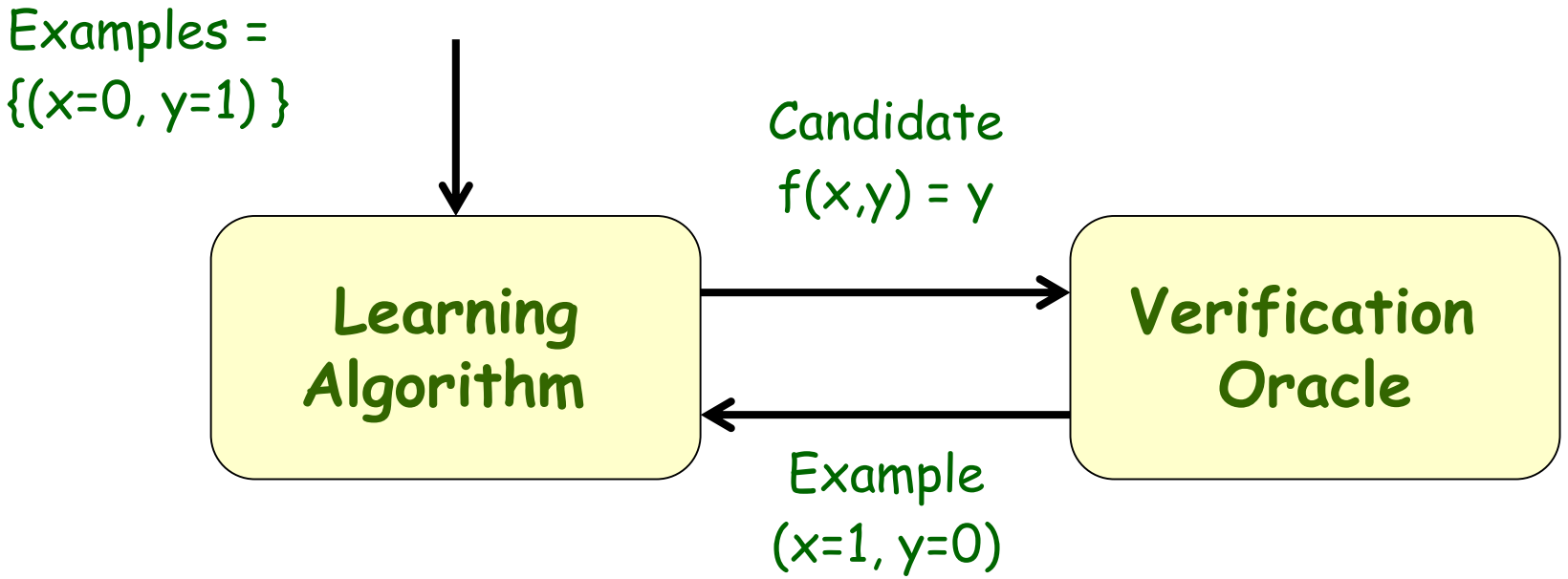
Counterexample-Guided Inductive Synthesis

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



CEGIS Example

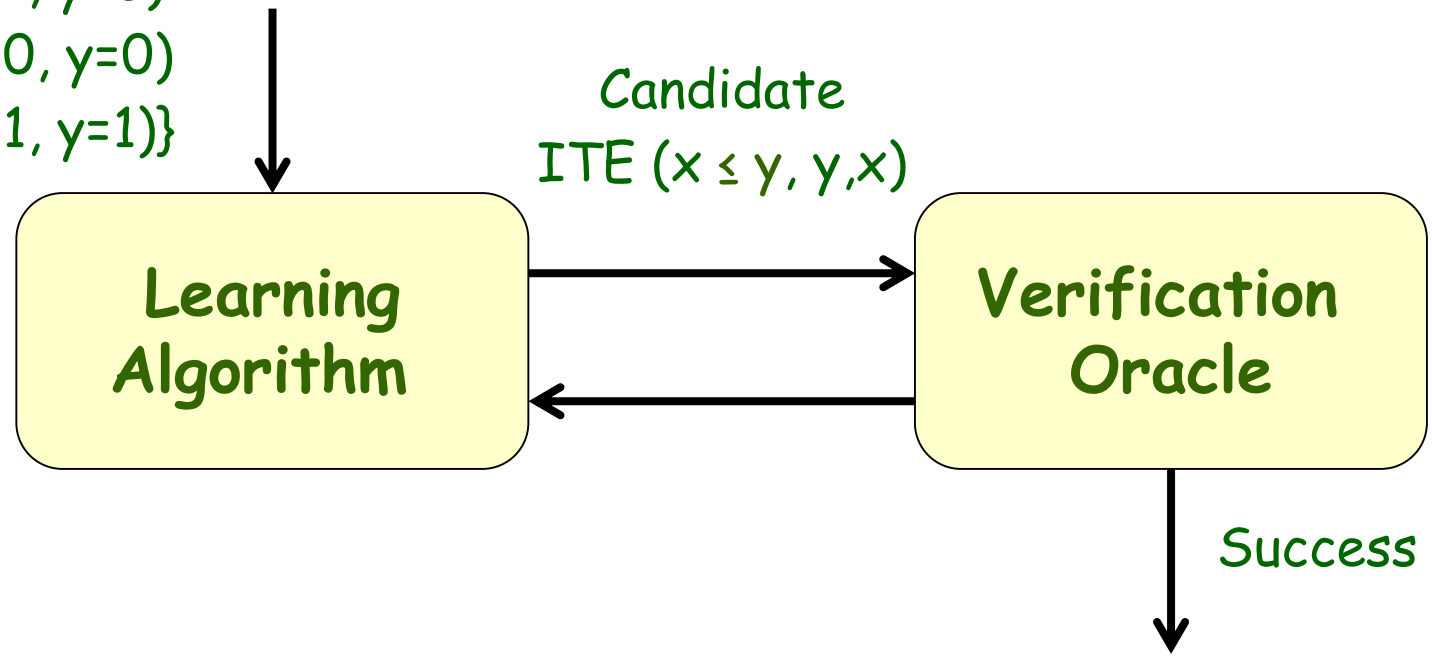
- Specification: $(x \le f(x,y)) \;\&\; (y \le f(x,y))$
- Set E: All expressions built from x,y,0,1, Comparison, +, If-Then-Else



CEGIS Example

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

Examples =
{(x=0, y=1)
(x=1, y=0)
(x=0, y=0)
(x=1, y=1)}



Counterexample-guided Inductive Synthesis (CEGIS)

Goal: Find f such that for all x in D, $\varphi(x, f)$ holds

I = { }; /* Interesting set of inputs */
Repeat
    Learn: Find f such that for all x in I, $\varphi(f, x)$ holds
    Verify: Check if for all x in D, $\varphi(f, x)$ holds
            If so, return f
            If not, find x such that ~ $\varphi(f, x)$ holds, and add x to I

SyGuS Solutions

- CEGIS approach (Solar-Lezama et al, ASPLOS'08)
- Similar strategies for solving quantified formulas and invariant generation
- Learning strategies based on:
  - Enumerative (search with pruning): Udupa et al (PLDI'13)
  - Symbolic (solving constraints): Gulwani et al (PLDI'11)
  - Stochastic (probabilistic walk): Schkufza et al (ASPLOS'13)

Enumerative Learning

- Find an expression consistent with a given set of concrete examples
- Enumerate expressions in increasing size, and evaluate each expression on all concrete inputs to check consistency
- Key optimization for efficient pruning of search space:
  - Expressions e_1 and e_2 are equivalent
    - if $e_1(a,b)=e_2(a,b)$ on all concrete values (x=a,y=b) in Examples
  - Only one representative among equivalent subexpressions needs to be considered for building larger expressions
- Fast and robust for learning expressions with ~ 20 nodes

Enumerative Search Example

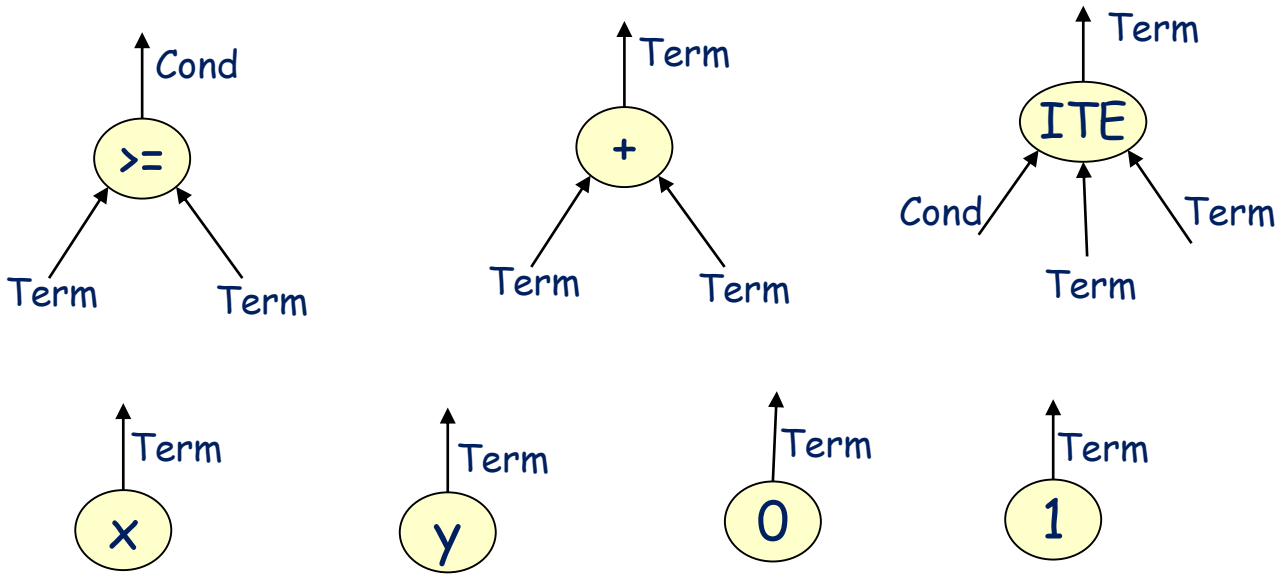
- Spec: (f(x,y) > x) & (f(x,y) > y)
- Grammar: E := x | y | 0 | 1 | E + E
- Examples = { (x=0, y=1) }
- Find an expression f such that (f(0,1) > 0) & (f(0,1) > 1)

- Expressions of size 1: x, y, 0, 1
- But x is equivalent to 0 for all points in Examples
- Also y is equivalent to 1, so only interesting expressions of size 1: x, y
- Neither f=x nor f=y satisfies the spec (f(0,1) > 0) & (f(0,1) > 1)
- So we need to enumerate expressions of larger size
- Expressions of size 3: x+x, x+y, y+x, y+y
- Can discard x+x as it is equivalent to x (for points in current Examples)
- Expressions x+y and y+x are equivalent to y
- Only interesting expression of size 3: y+y
- f(x,y)=y+y does satisfy (f(0,1)>0) & (f(0,1)>1), so return y+y

Symbolic Learning

- Use a constraint solver for both the synthesis and verification step.
- Each production in the grammar is thought of as a component.

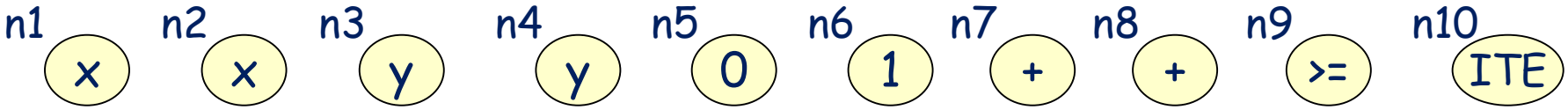
  Input and Output ports of every component are typed.



- A well-typed loop-free program comprising these component corresponds to an expression DAG from the grammar.

Symbolic Learning

- Start with a library consisting of some number of occurrences of each component.



- Synthesis Constraints:

  Shape is a DAG, Types are consistent

  Spec $\varphi[f/e]$ is satisfied on every concrete input values in I

- Use an SMT solver (Z3) to find a satisfying solution.

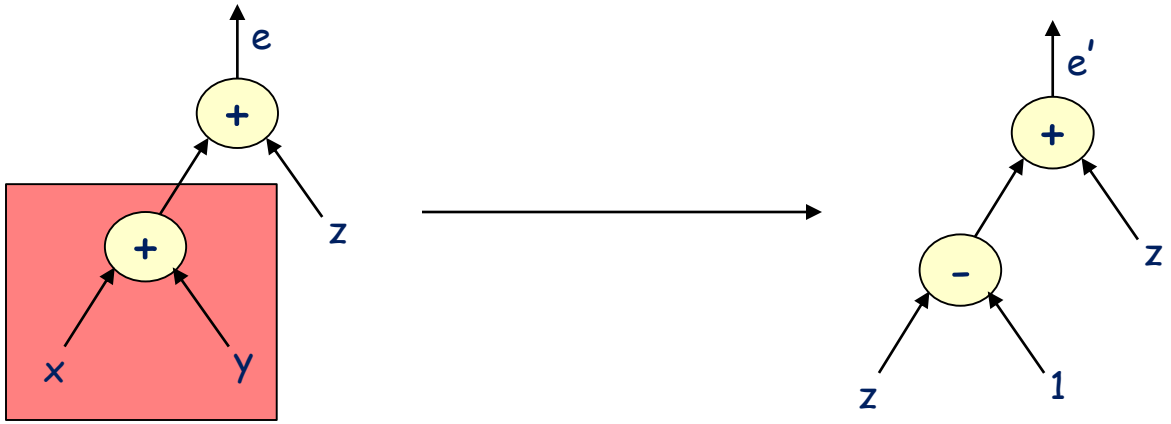
- If synthesis fails, try increasing the number of occurrences of components in the library in an outer loop

Stochastic Learning

- Idea: Find desired expression e by probabilistic walk on graph where nodes are expressions and edges capture single-edits
- Metropolis-Hastings Algorithm: Given a probability distribution P over domain X, and an ergodic Markov chain over X, samples from X
- Fix expression size n. X is the set of expressions E_n of size n. P(e) $\propto$Score(e) ("Extent to which e meets the spec φ")
- For a given set Examples, Score(e) = exp(- 0.5 Wrong(e)), where Wrong(e) = No of inputs in Examples for which ~ φ [f/e]
- Score(e) is large when Wrong(e) is small. Expressions e with Wrong(e) = 0 more likely to be chosen in the limit than any other expression

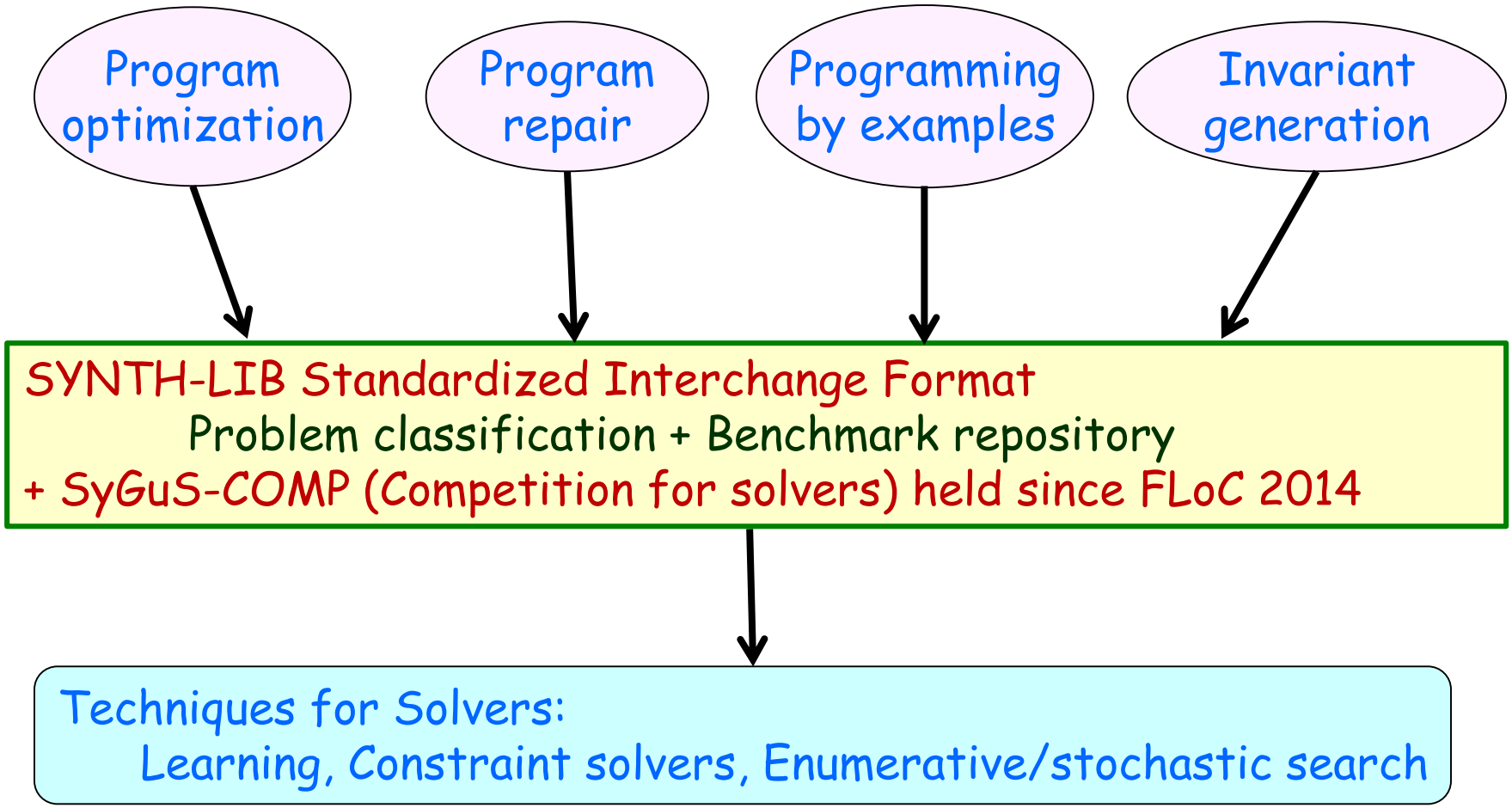
Stochastic Learning

- Initial candidate expression e sampled uniformly from E_n
- When Score(e) = 1, return e
- Pick node v in parse tree of e uniformly at random. Replace subtree rooted at e with subtree of same size, sampled uniformly



- With probability min{ 1, Score(e')/Score(e) }, replace e with e'
- Outer loop responsible for updating expression size n

SyGuS Solvers ⟷ Synthesis Tools



Collaborators: Fisman, Singh, Solar-Lezama

SyGuS Progress

www.sygus.org

- Over 1500 benchmarks
  - Hacker's delight: Programming by examples for bit-vector
  - Invariant generation (based on verification competition SV-Comp)
  - FlashFill (programming by examples system for string manipulation programs 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

SyGuS Progress

www.sygus.org

- Solution strategies
  - Enumerative (search with pruning) (Udupa… PLDI'13)
  - Symbolic (solving constraints) (Gulwani… PLDI'11)
  - Stochastic (probabilistic walk) (Schkufza… ASPLOS'13)
  - Implication counterexamples for invariant learning (Garg… POPL'15)
  - CVC4: integrating synthesis with SMT solver (Reynolds… CAV'15)
  - Decision trees + Enumerative search (Radhakrishna… 2016)
  - Guiding search based on learnt probabilistic models (Lee… 2017)
- Increasing interest in PL/FM/SE communities
  - Synthesis of Fault-Attack Countermeasures for Cryptographic Circuits (Wang… CAV'16)
  - Counterexample-guided model synthesis (Biere… TACAS'17)
  - Syntax-Guided Optimal Synthesis for Chemical Reaction Networks; (Cardelli… CAV'17)

Scaling Enumerative Search by Divide&Conquer

- For the spec (f(x,y) >= x) & (f(x,y) >= y), the answer is

  if-then-else (x>=y, x, y)

- Size of expressions in conditionals and terms can be much smaller than the size of the entire expression!
- f(x,y) = x is correct when x >= y and f(x,y) = y is correct when x <= y

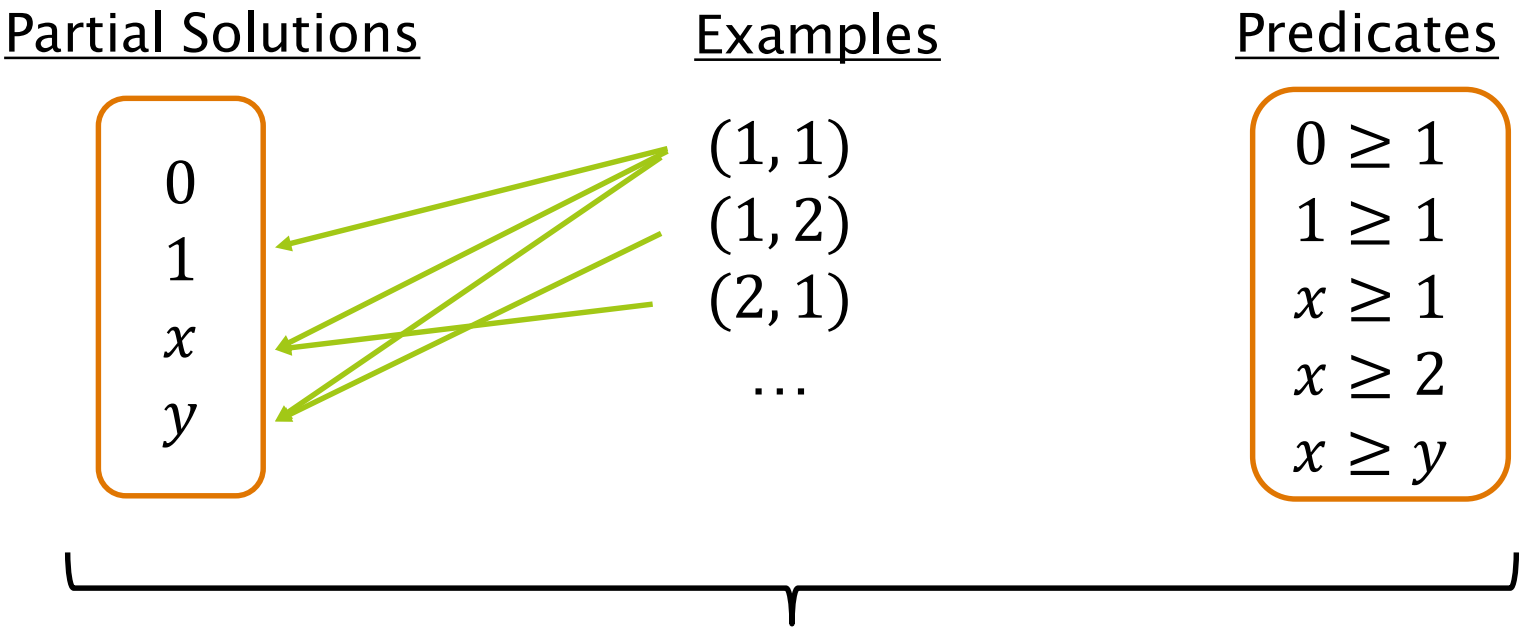
- Key idea:
  - Generate partial solutions that are correct on subsets of inputs and combine them using conditionals
  - Enumerate terms and tests for conditionals separately

- Does not work in general
  - Correctness of outputs for different inputs can be determined independently
  - Plainly separable specifications: Each conjunct of the specification contains a unique invocation of the unknown function

Divide & Conquer Overview

Step 1: Propose terms until all points covered

Step 2: Generate predicates



Step 3: Combine! $if\ (x \geq y)\ then\ x\ else\ y$

Conditional Expression Grammars

```
Expr ::=  T  | if (C) Expr else Expr
T      ::= x | y | 0 | 1 | T + T | T - T
C      ::= T >= T | T = T | T <= T | …
```

Conditional Linear Expressions

```
T      ::= x | y | 0 | 1 | T + T | T - T
```

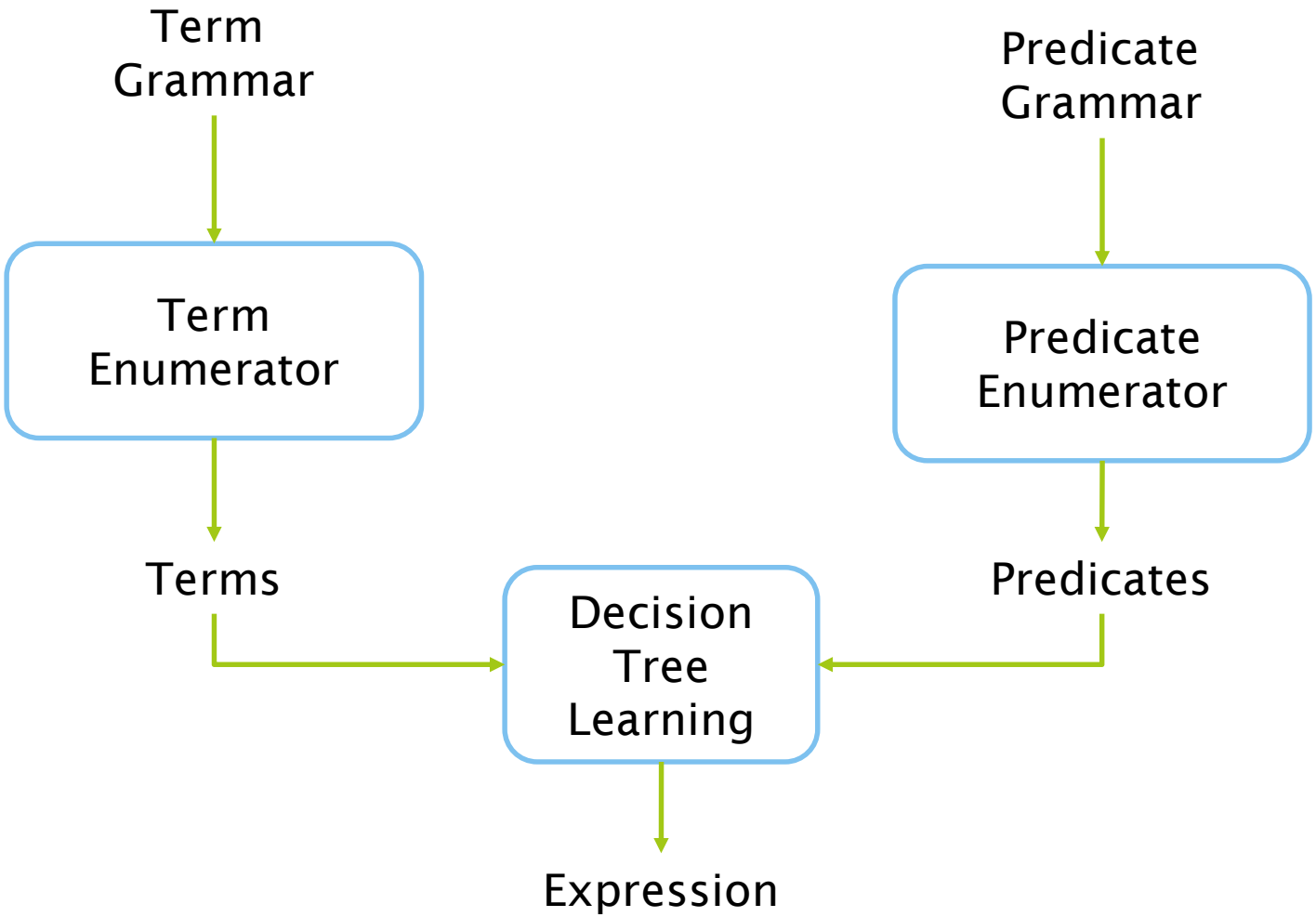
Term Grammar

```
C      ::= T >= T | T = T | T <= T | …
```

Predicate Grammar

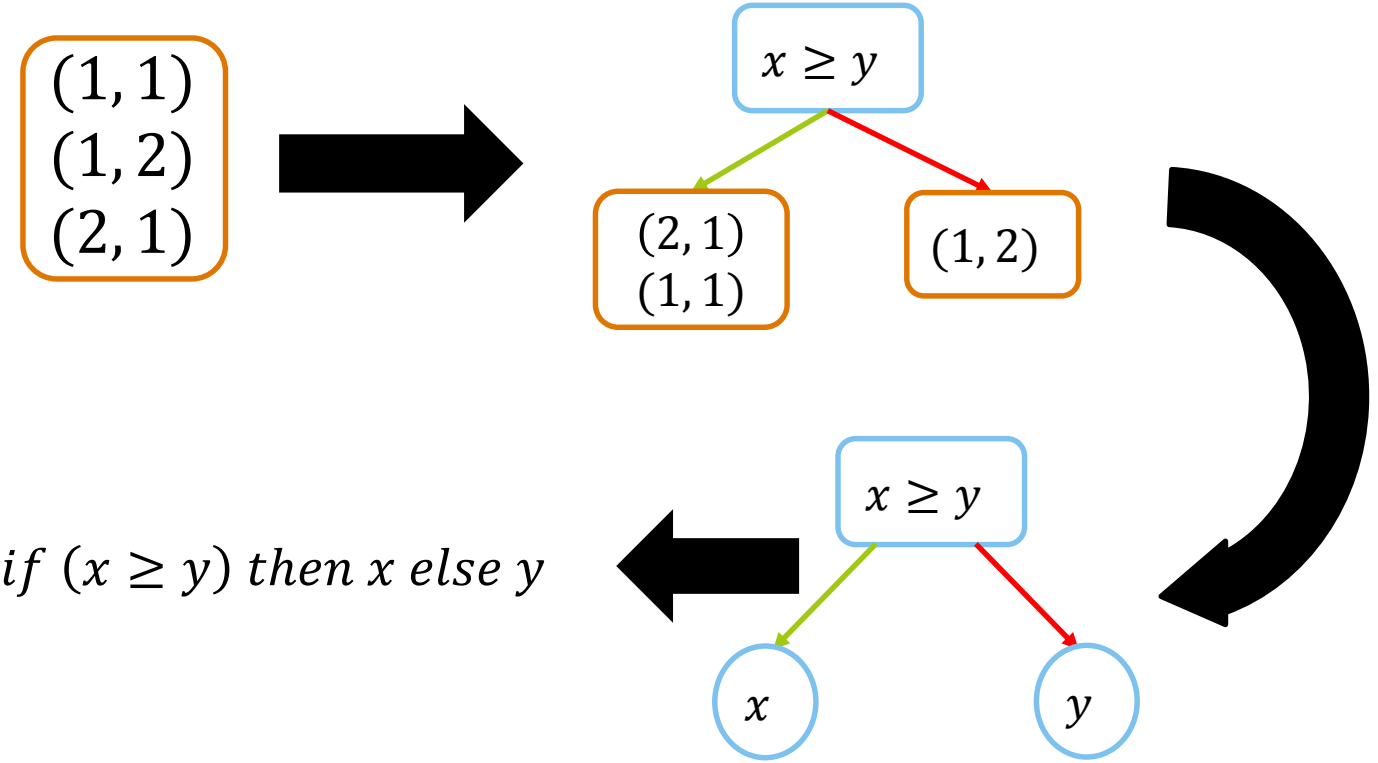
Separate grammars for predicates and terms
Can be automatically for typical grammars

Algorithm Overview



Decision Tree Learning

0	1	2	x	y
$0 \geq 1$				
$1 \geq 1$				
$x \geq 1$				
$x \geq 2$				
$x \geq y$				



Check if each set of examples can be covered by one term
If not, pick a predicate and split the set

How to choose a splitting predicate ?

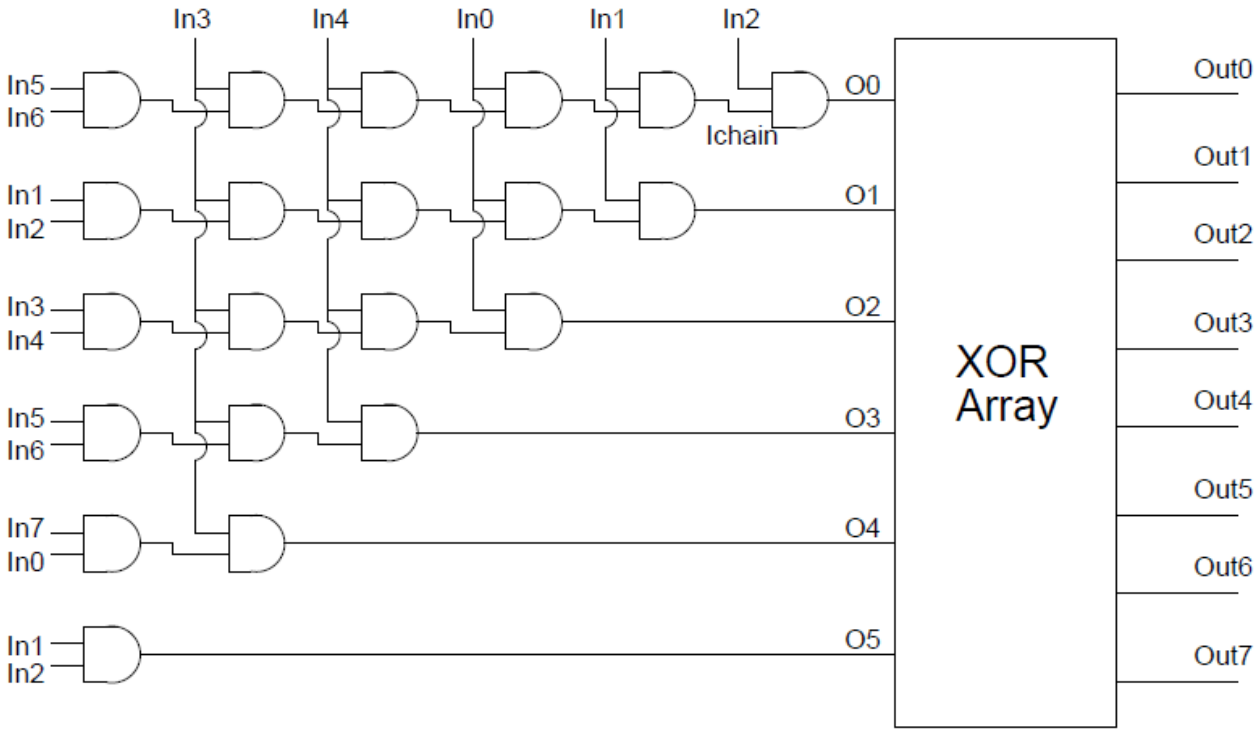
- Given:
  - a set P of predicates/attributes (e.g. x <=y, …)
  - a set T of terms/labels (e.g. x, y, …)
  - a set X of examples/points (e.g. (x=0,y=1), …)
  - a specification (e.g. f(x,y)>= x & f(x,y)>=y)

- A point x has label t, if setting f=t satisfies specification for x
- For each predicate p, let X_p be points in X that satisfy p
- For a subset Y of points, H(Y) is the "entropy" of Y and depends on how points in Y are labeled with terms in T
- For a predicate p,

  $Gain(p) = |X_p|/|X| * H(X_p) + |X_{\sim p}|/|X| * H(X_{\sim p})$

- Split X using the predicate p in P for which Gain(p) is maximum

Back to Synthesis of Attack Countermeasures

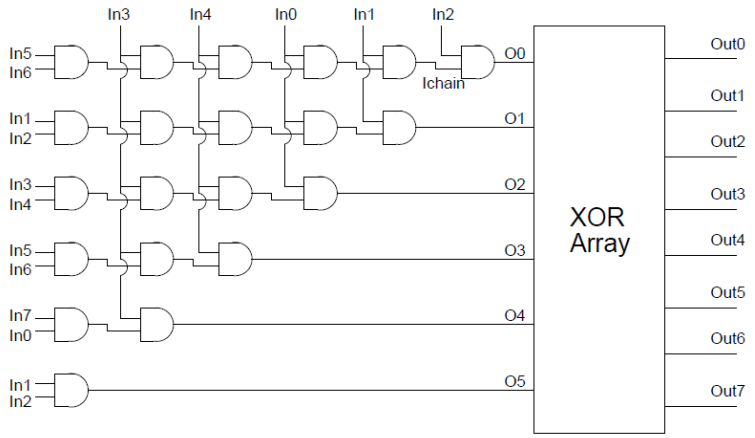


Given a ckt C, automatically synthesize a ckt C′ such that

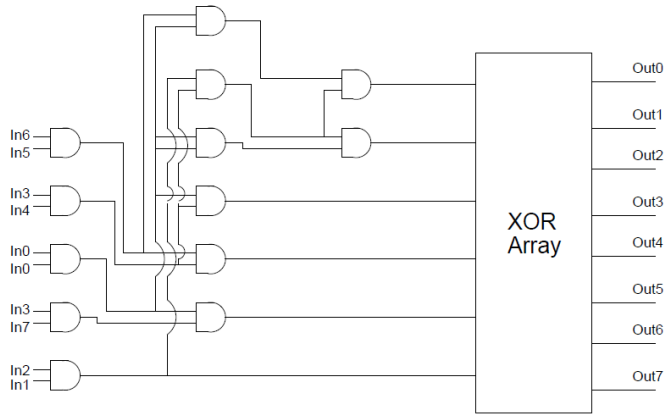
1. C′ is functionally equivalent to C [sematic constraint]
2. All input-to-output paths in C′ have same length [syntactic constraint]

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

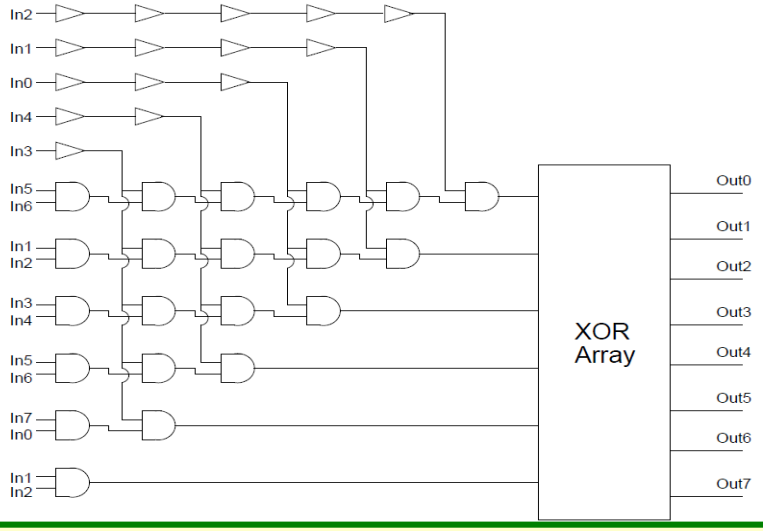
SyGuS Result



Original ckt prone to attack



SyGuS-generated Attack resilient ckt



Hand-crafted attack resilient ckt

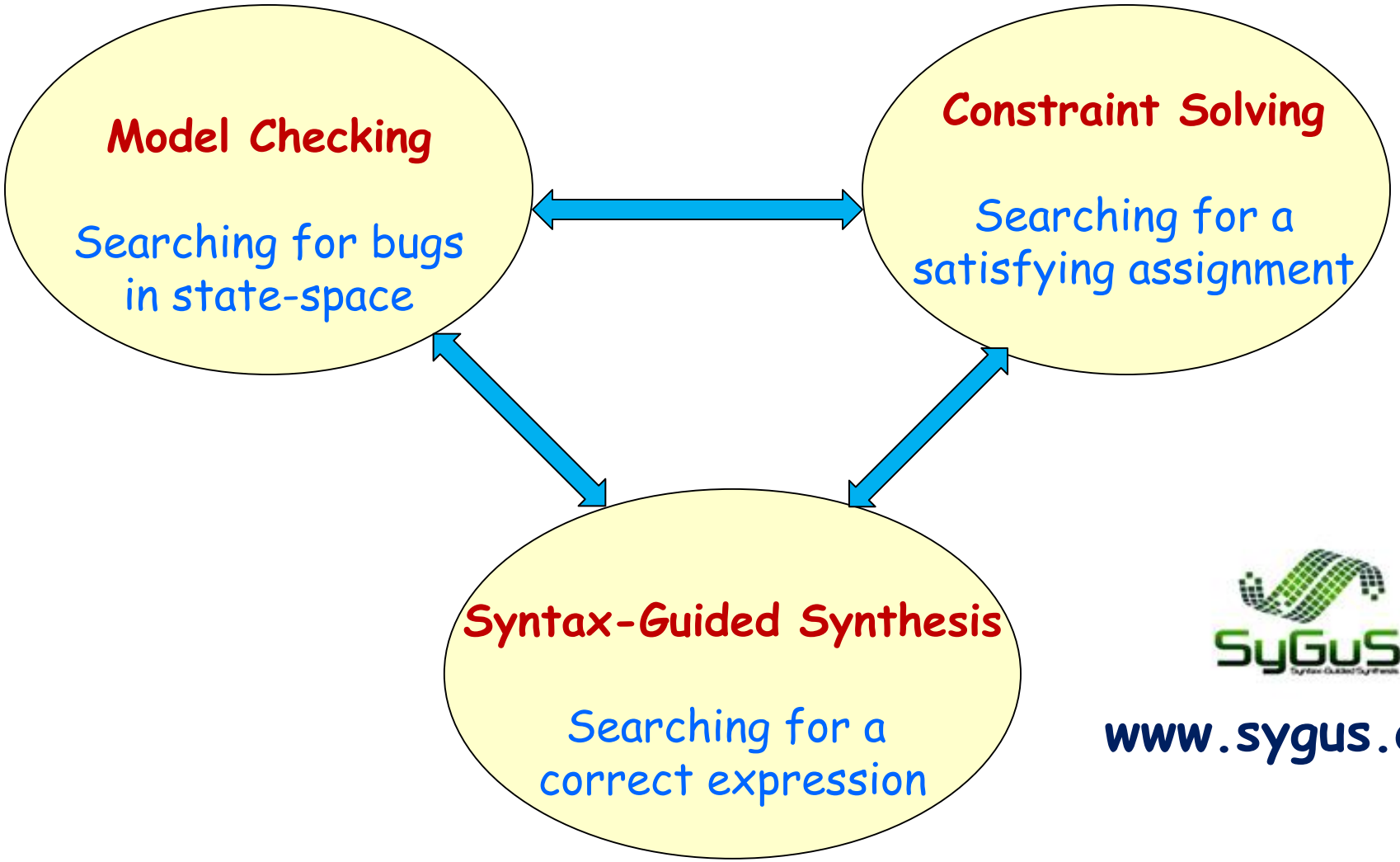
Fully automatic
Smaller size
Shorter delays

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

CAV: A Story of Battling Exponentials



SyGuS

www.sygus.org