Syntax-Guided Program Synthesis

Rajeev Alur

University of Pennsylvania

Part I

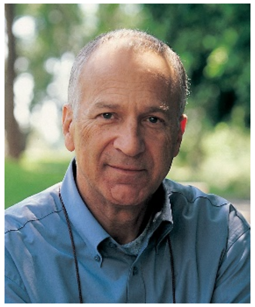
(Syntax-guided) Synthesis: Why and What ?

Formal Verification in Practice

Next Challenge: Beyond Verification and Testing ??

Vision of Program Synthesis

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

Applications in the near term …

1. Programming by examples (PBE)
2. Superoptimizing compilers
3. Program repair
4. Proof objects for verification

1. 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

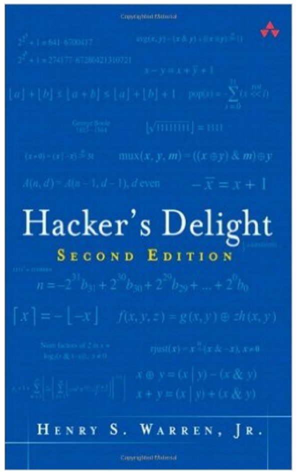
   |, &, ~, +, -, <<, >>, 0, 1, …

2. P specified using input-output examples

   00101 → 00100

   01010 → 01000

   10110 → 10000



Desired solution:

$x \mathbin{\&} (1 + (x \mid (x-1)))$

FlashFill: PBE in Practice

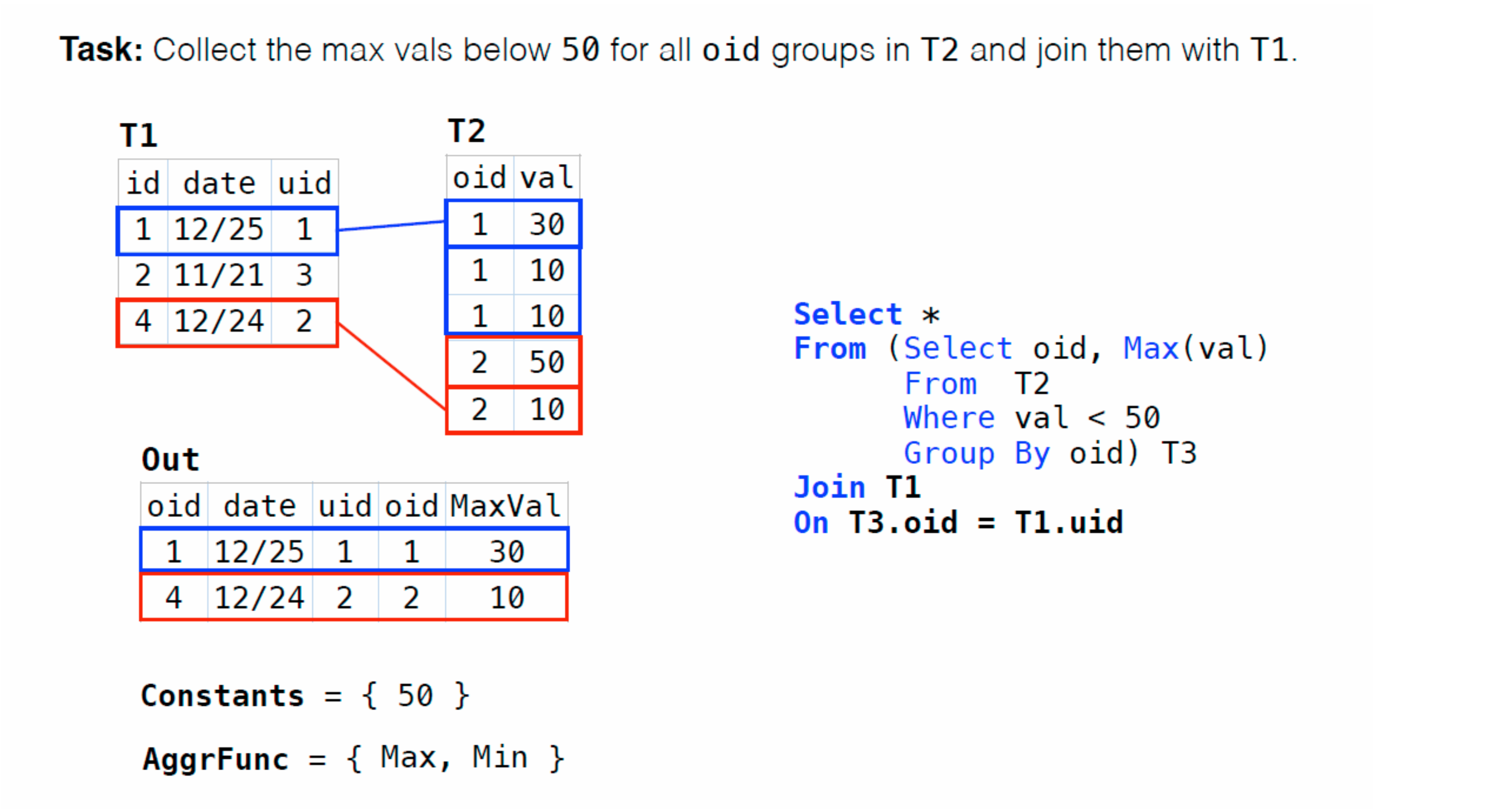
Ref: Gulwani (POPL 2011)

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.

Scythe: SQL queries from input-output examples

Wang, Cheung, Bodik; scythe.cs.washington.edu



Task: Collect the max vals below 50 for all oid groups in T2 and join them with T1.

T1

id	date	uid
1	12/25	1
2	11/21	3
4	12/24	2

T2

oid	val
1	30
1	10
1	10
2	50
2	10

Out

oid	date	uid	oid	MaxVal
1	12/25	1	1	30
4	12/24	2	2	10

```
Select *
From (Select oid, Max(val)
      From  T2
      Where val < 50
      Group By oid) T3
Join T1
On T3.oid = T1.uid
```

Constants = { 50 }

AggrFunc = { Max, Min }

2. 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’

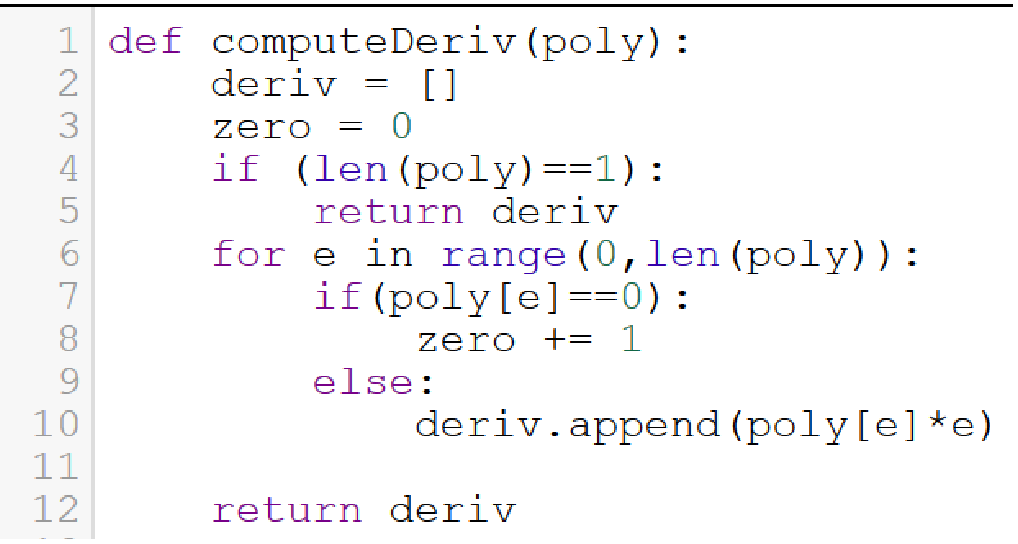
```
 average (bitvec[32] x, y) {
   bitvec[64] x1 = x;
   bitvec[64] y1 = y;
   bitvec[64] z1 = (x1+y1)/2;
   bitvec[32] z = z1;
   return z
}
```

Find equivalent code without extension to 64 bit vectors

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

3. Repair/Feedback for 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

4. Automatic Invariant Generation

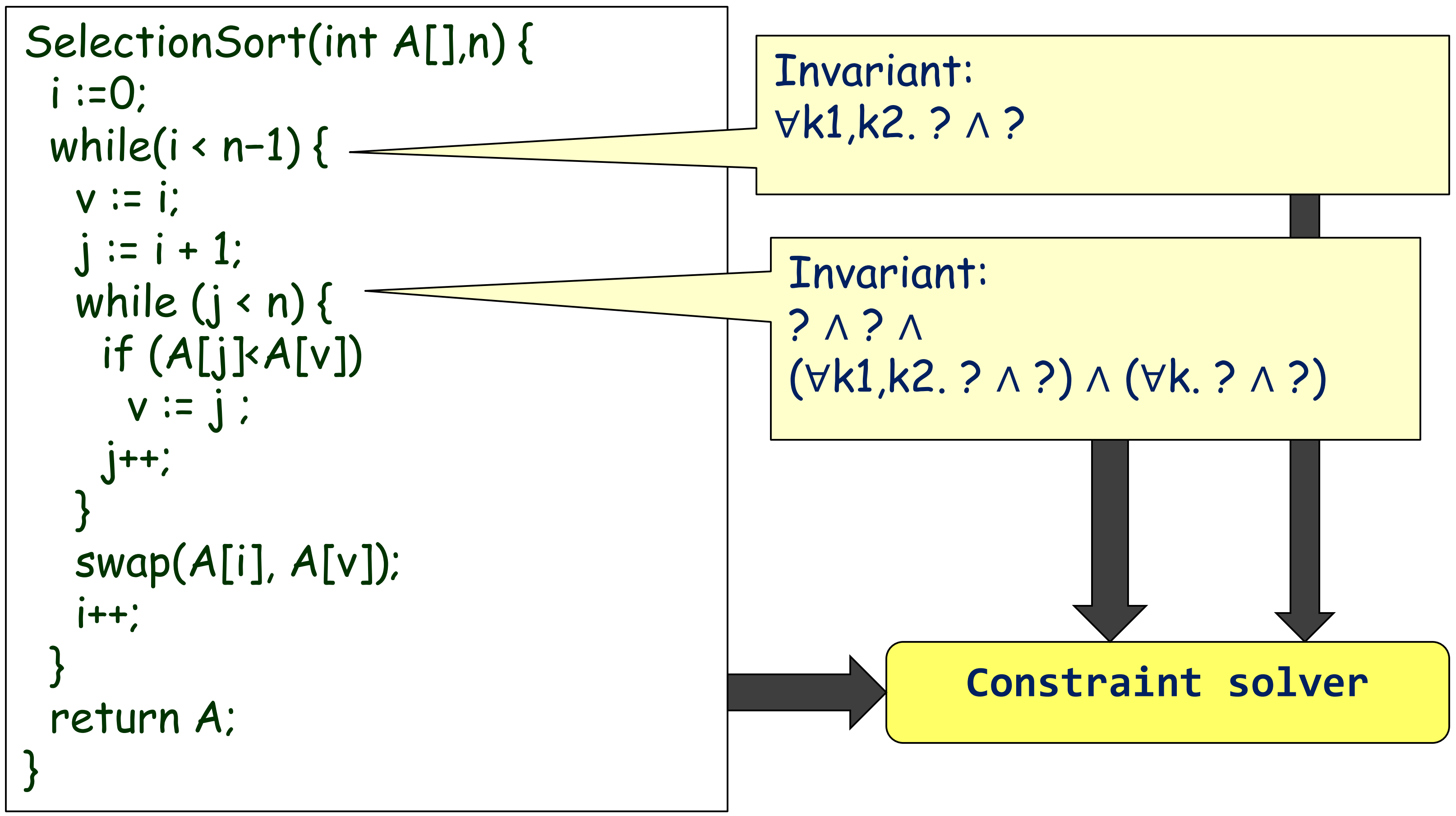
```
SelectionSort(int A[],n) {
  i := 0;
  while(i < n-1) {
    v := i;
    j := i + 1;
    while (j < n) {
      if (A[j]<A[v])
        v := j ;
      j++;
    }
    swap(A[i], A[v]);
    i++;
  }
  return A;
}
```

Invariant: ?

Invariant: ?

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

Template-based Automatic Invariant Generation



```
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:
$\forall k1,k2.\ ? \wedge ?$
Invariant:
$? \wedge ? \wedge$
$(\forall k1,k2.\ ? \wedge ?) \wedge (\forall k.\ ? \wedge ?)$
Constraint solver

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

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;
}
```

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

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

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

Syntax-Guided Program Synthesis

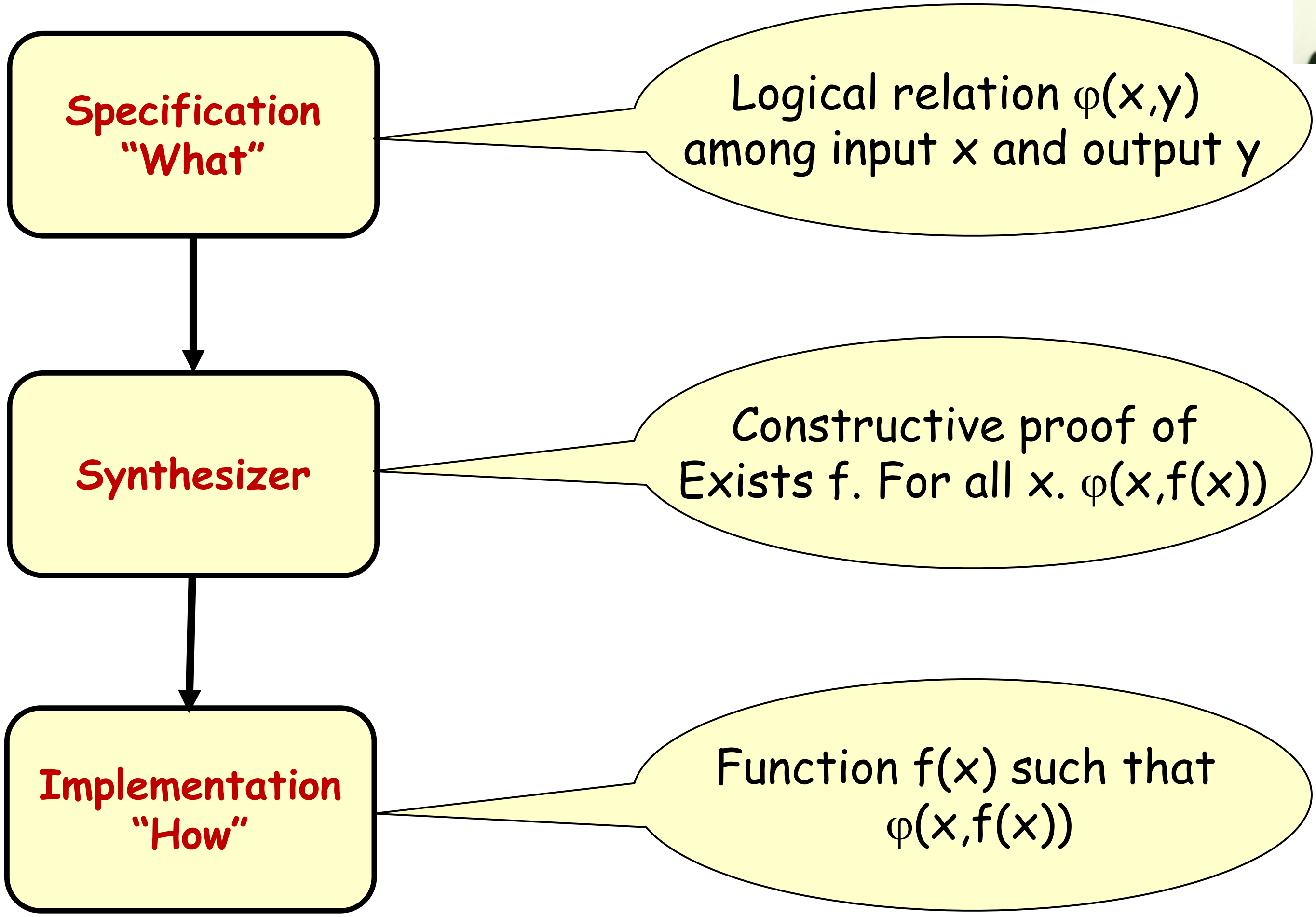
Rich variety of projects in programming systems and software engineering

1. Programming by examples
2. Program superoptimization
3. Automatic program repair
4. Template-guided invariant generation

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

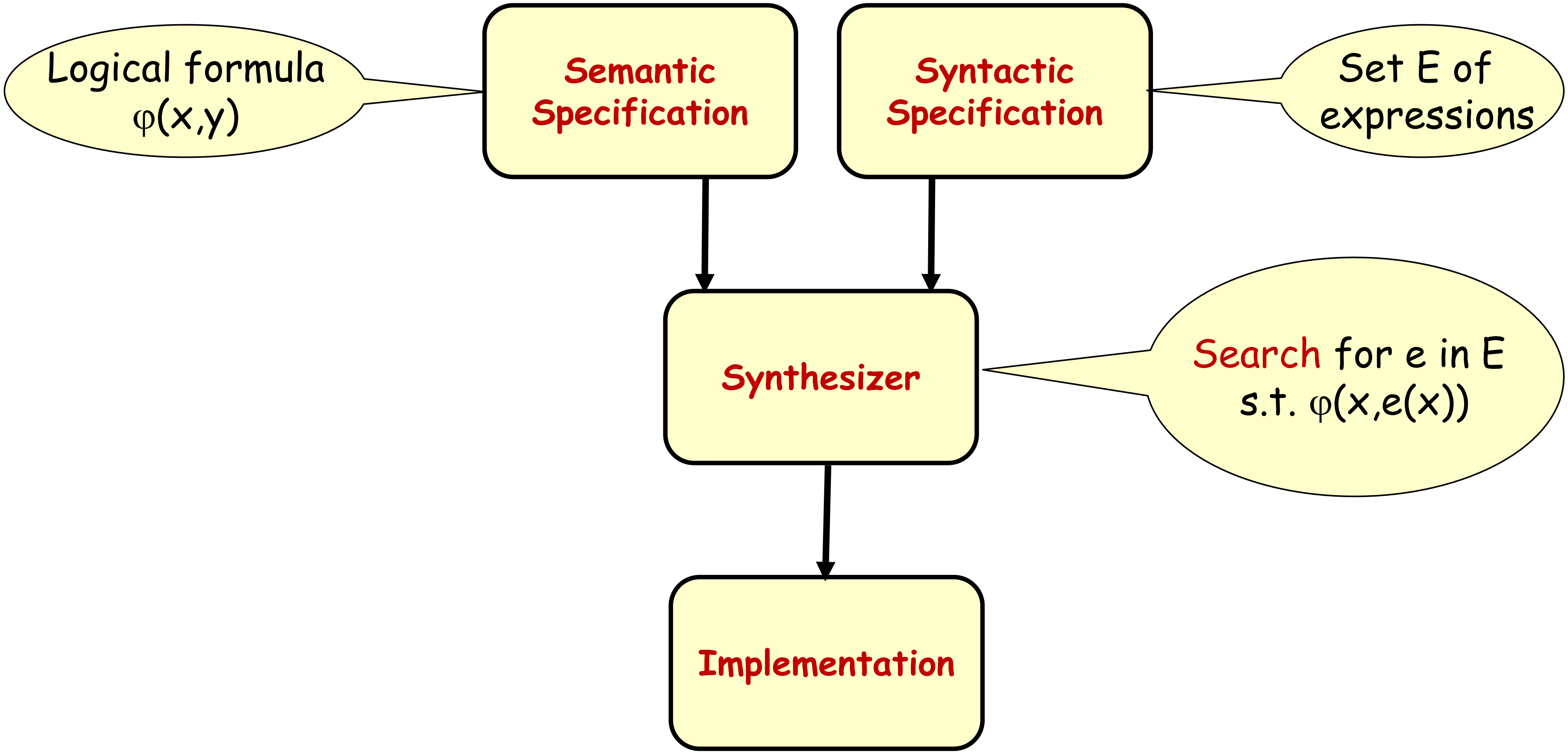
Classical Program Synthesis

Church (1957)


Specification
"What"
Logical relation φ(x,y) among input x and output y
Synthesizer
Constructive proof of Exists f. For all x. φ(x,f(x))
Implementation
"How"
Function f(x) such that φ(x,f(x))

Syntax-Guided Program Synthesis

www.sygus.org



Part II

Syntax-guided Synthesis: Formalization

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)

Little Engines of Proof

SAT; Linear arithmetic; Congruence closure

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 $\leq$ Term | Cond & Cond | ~ Cond | (Cond)

- Possible solution:

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

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 (≤  x (max2 x y)))
 (constraint (≤  y (max2 x y)))
 (constraint (or (= x (max2 x y)) (= y (max2 x y))))
 (check-synth)
```

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
  Update(x,y,z,a,b,c, x',y',z',a',b',c')
- Specification:
  (Inv & Update & Test') $\Rightarrow$ Inv'
  & Pre $\Rightarrow$ Inv & (Inv & ~Test $\Rightarrow$ 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)

Part III

Solving SyGuS

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

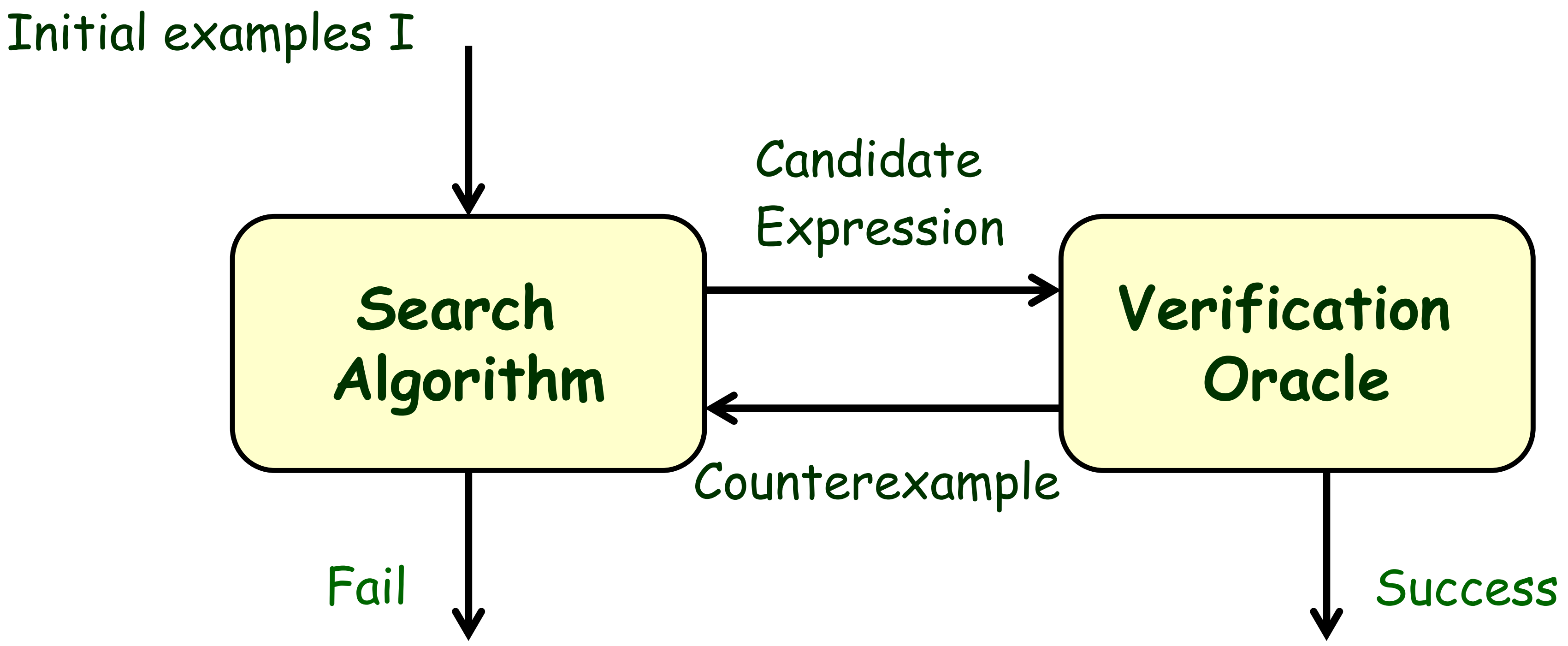
    SyGuS reduces to Exists a,b,c. Forall X. φ [f/ ax+by+c]
  - Set E is all conditional expressions

    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



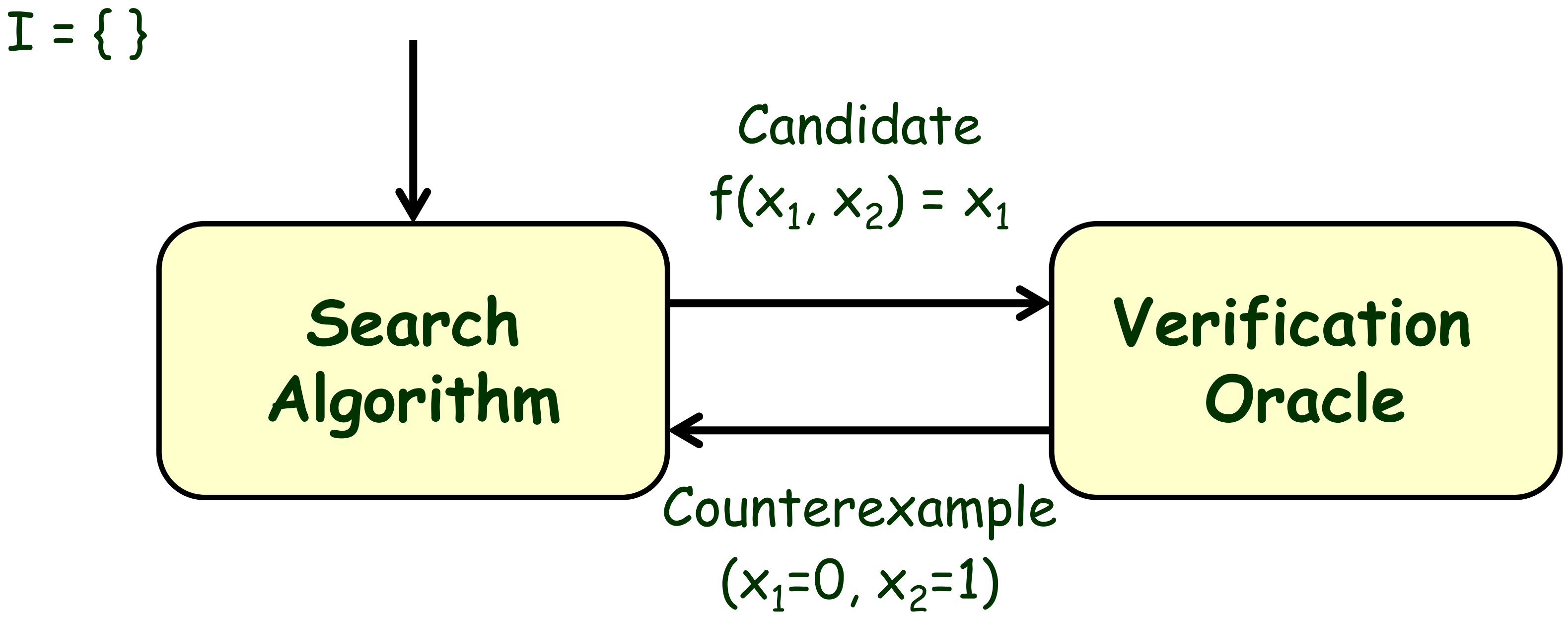
Concept class: Set E of expressions

Examples: Concrete input values

Counterexample-Guided Inductive Synthesis

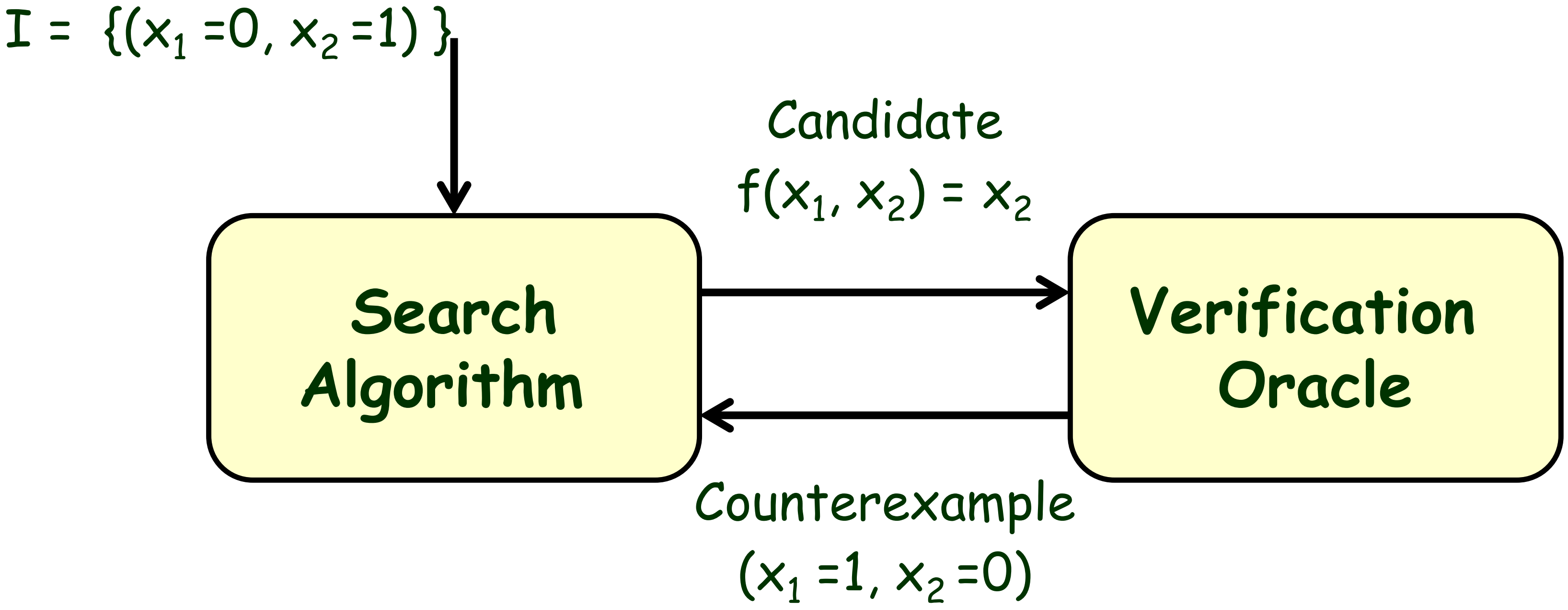
Solar-Lezama et al (ASPLOS'06)

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



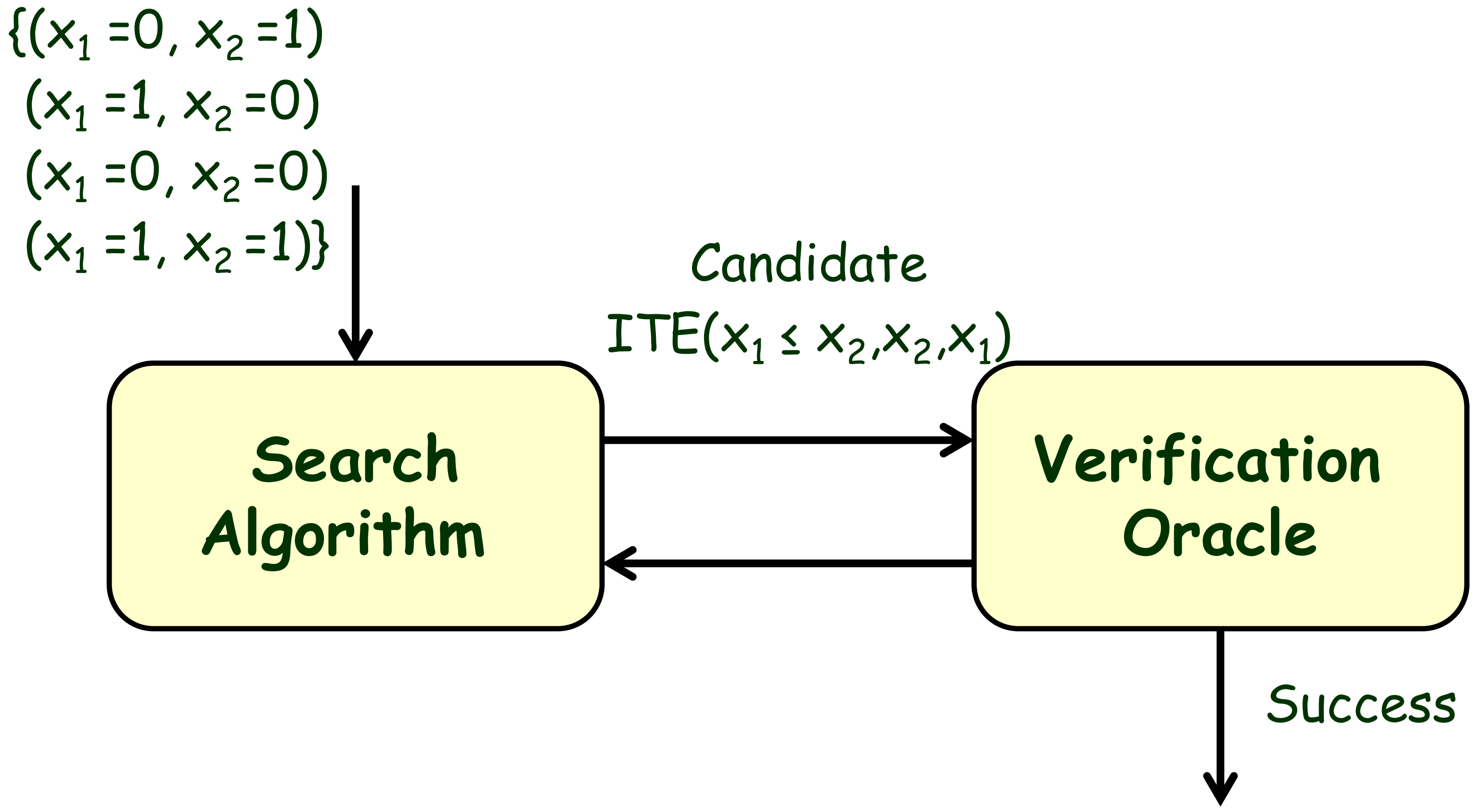
CEGIS Example

- Specification: $(x_1 \leq f(x_1, x_2))$ & $(x_2 \leq f(x_1, x_2))$
- 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



Counterexample-guided Inductive Synthesis (CEGIS)

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

```
I = { }; /* Interesting set of inputs */
Repeat
    Learn: Find f in E such that for all x in I, $\varphi(f, x)$ holds
    Verify: Find x in D such that $\varphi(f, x)$ does not hol
            If so, add x to I
            Else, return f
```

SyGuS Solutions

- ❑ CEGIS approach (Solar-Lezama et al, ASPLOS'08)

- ❑ Similar strategies for solving quantified formulas and invariant generation

- ❑ Initial learning strategies based on:
  1. Enumerative (search with pruning): Udupa et al (PLDI'13)
  2. Symbolic (solving constraints): Gulwani et al (PLDI'11)
  3. Stochastic (probabilistic walk): Schkufza et al (ASPLOS'13)

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 in 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: $E ::= x_1 \mid x_2 \mid 0 \mid 1 \mid E + E$
- $I = \{ (x_1=0, x_2=1) \}$
- Find an expression f such that $(f(0,1) > 0)$ & $(f(0,1) > 1)$

x_1 x_2

~~0~~ ~~1~~

~~$x_1 + x_1$~~ ~~$x_1 + x_2$~~ $x_2 + x_2$

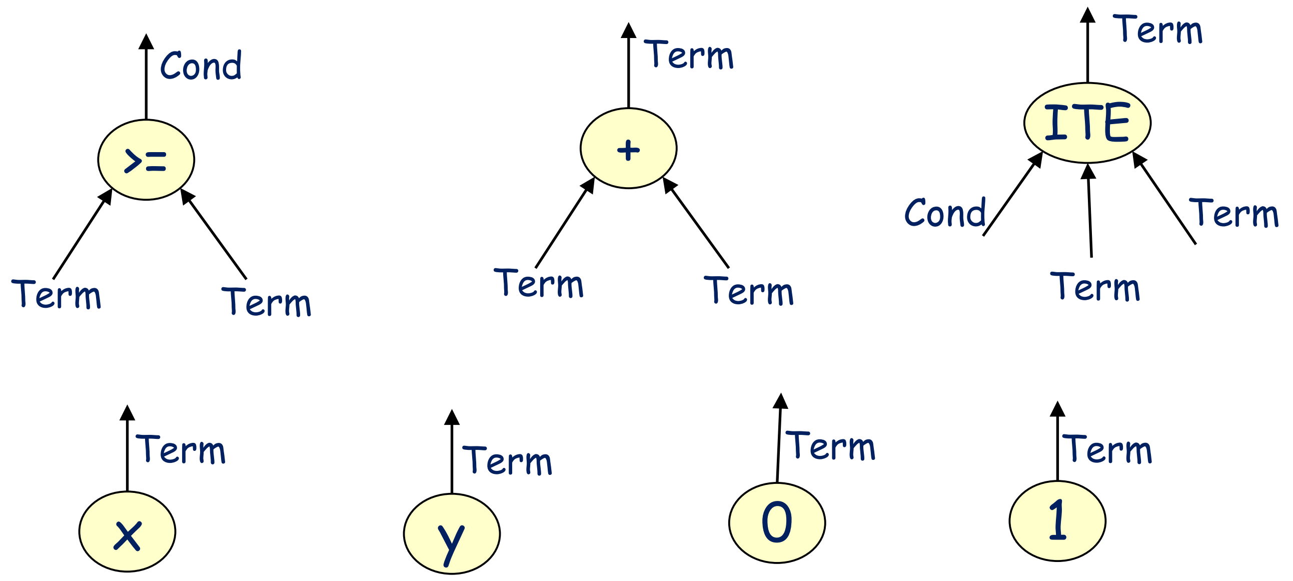
~~$x_2 + x_1$~~

2. Symbolic Search

- Use a constraint solver for both synthesis and verification steps

- Each production in the grammar is thought of as a component.

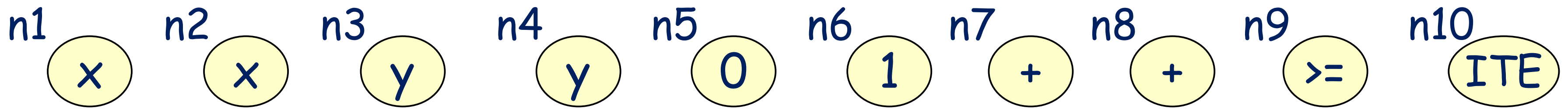
  Input and Output ports of every component are typed.



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

Symbolic Encoding

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



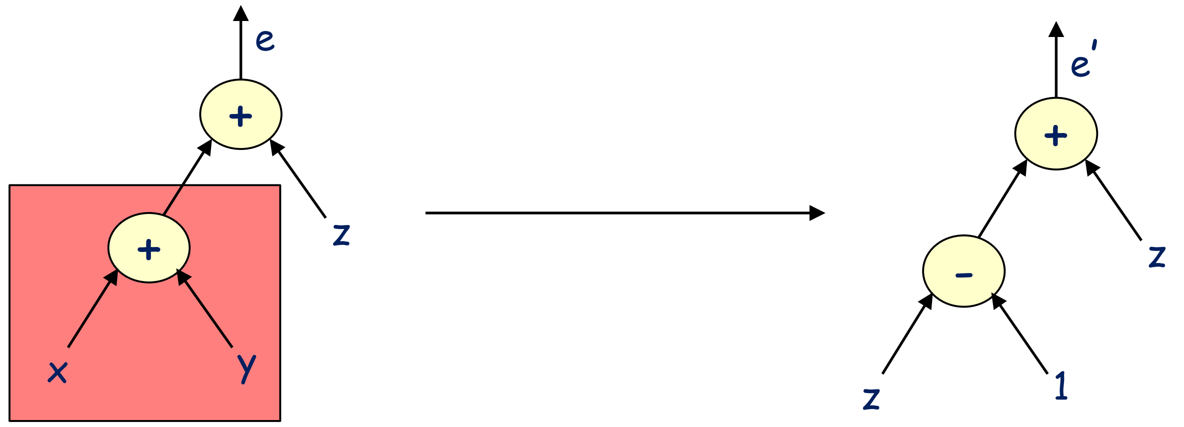
- Synthesis Constraints:
  - Shape is a DAG, Types are consistent
  - Spec $\varphi[f/e]$ is satisfied on every concrete input in I
- Use an SMT solver (Z3) to find a satisfying solution.
- If synthesis fails, try increasing the number of occurrences of components in the library in an outer loop

3. Stochastic Search

- 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 $\sim \varphi\,[f/e]$
- 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 Search

- 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

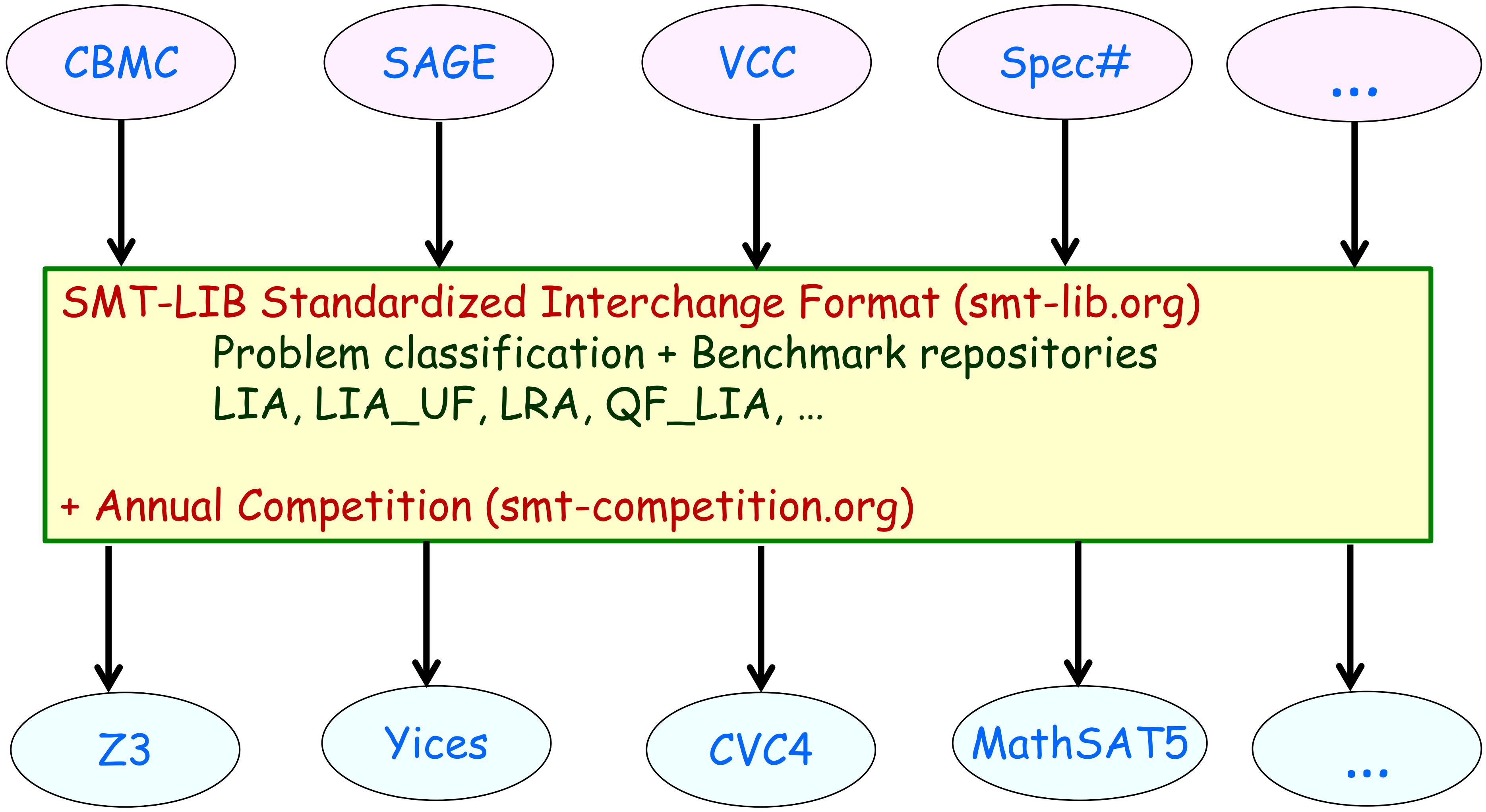


- With probability min{ 1, Score(e′)/Score(e) }, replace e with e′
- Outer loop responsible for updating expression size n

Part IV

SyGuS Competition and Evolution

SMT Success Story



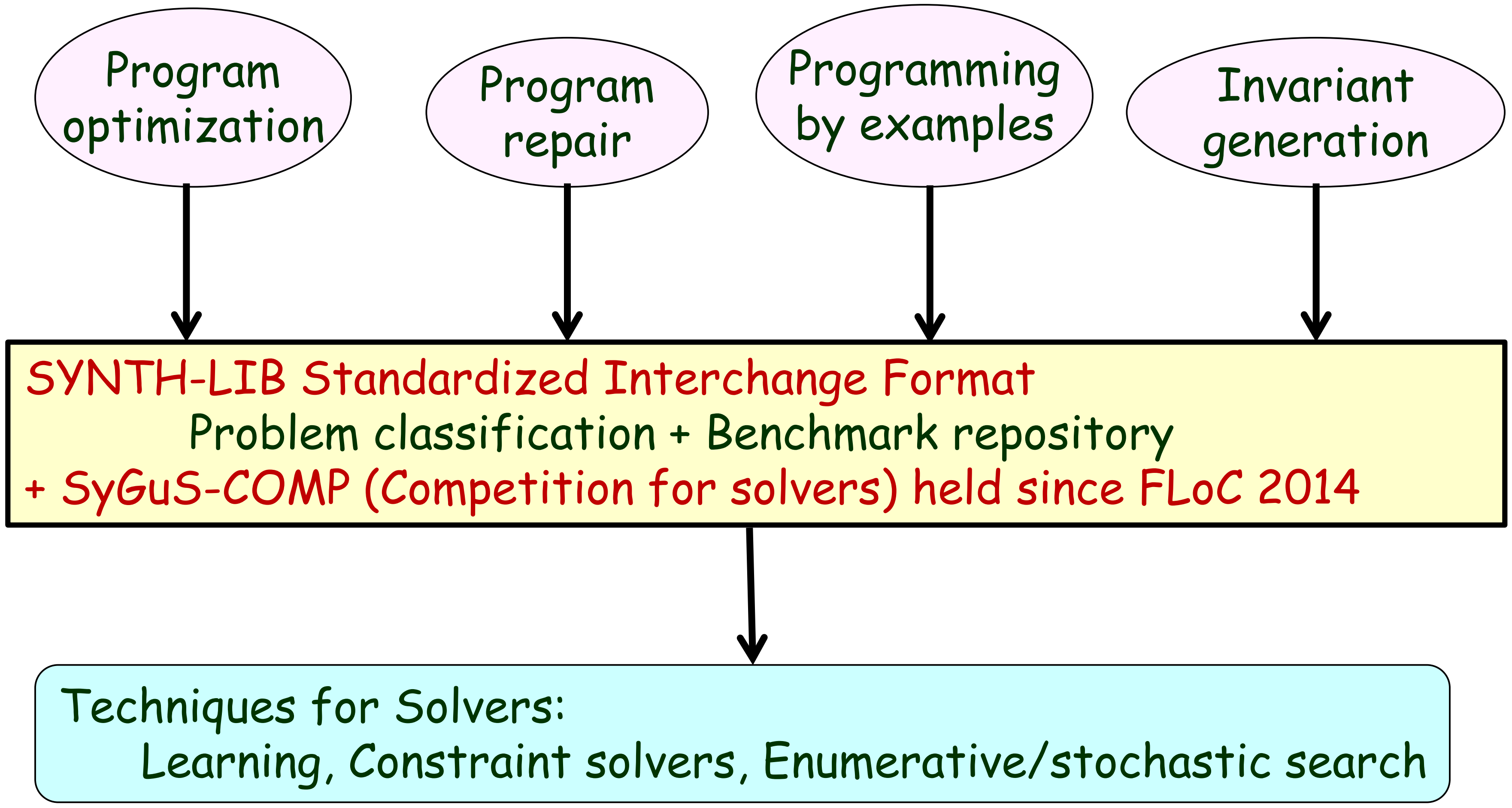
SMT-LIB Standardized Interchange Format (smt-lib.org)

Problem classification + Benchmark repositories

LIA, LIA_UF, LRA, QF_LIA, …

+ Annual Competition (smt-competition.org)

SyGuS Competition



SYNTH-LIB Standardized Interchange Format
Problem classification + Benchmark repository
+ SyGuS-COMP (Competition for solvers) held since FLoC 2014

Techniques for Solvers:
Learning, Constraint solvers, Enumerative/stochastic search

Collaborators: D. Fisman, S. Padhi, A. Reynolds, R. Singh, A. Solar-Lezama, A. Udupa

SyGuS Progress

www.sygus.org

- Over 2000 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 $(x_1 \leq x_2, x_2, x_1)$

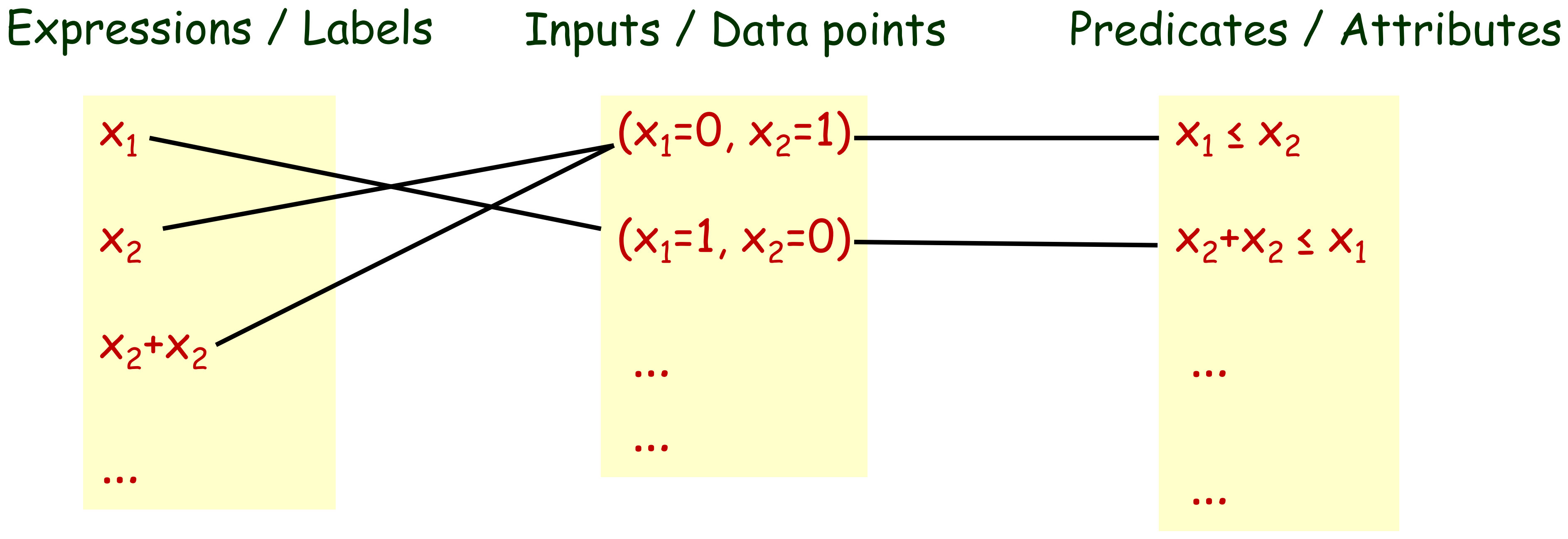
- Size of expressions in conditionals and terms can be much smaller than the size of the entire expression!

- $f(x_1, x_2)= x_2$ 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



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

2019 Winner : CVC4 (Reynolds et al):
Integration of enumerative search with constraint solving !!

Emerging Applications of SyGuS

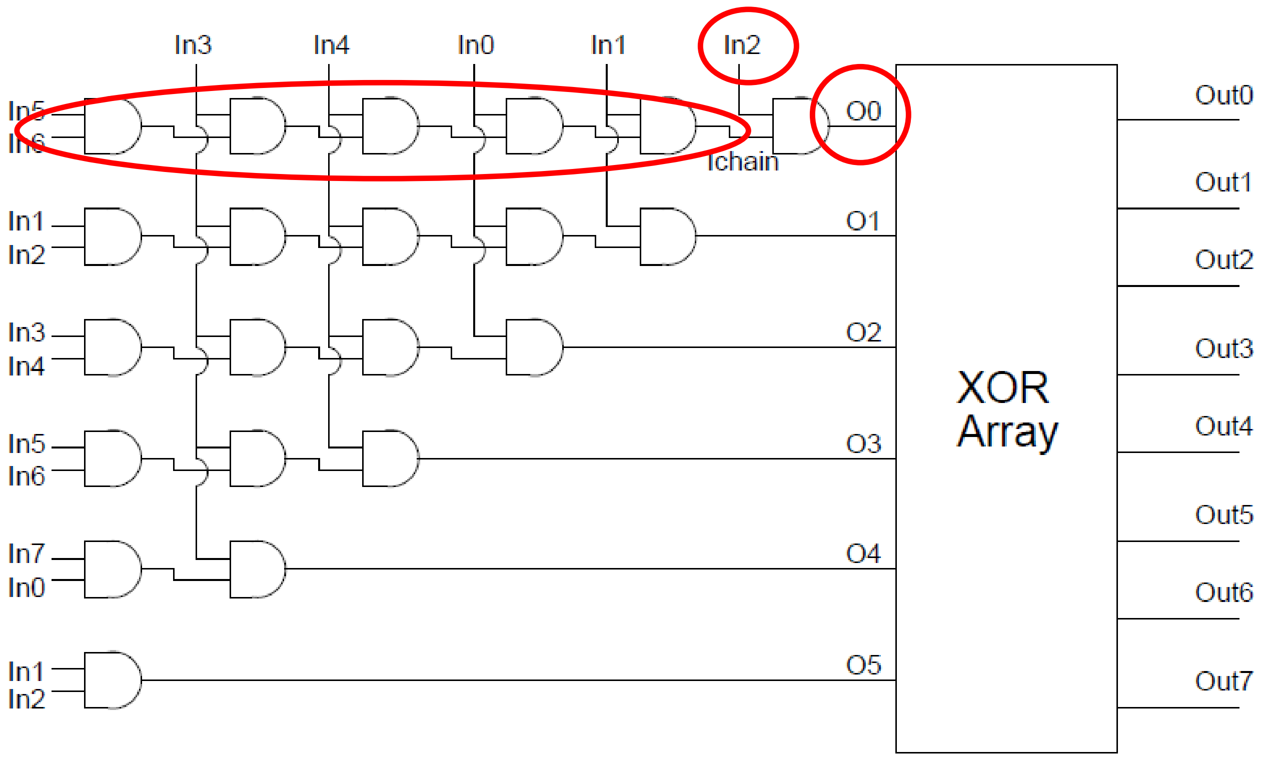
- Synthesis of crypto-circuits resilient to timing attack
  (Wang et al, CAV 2016)

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

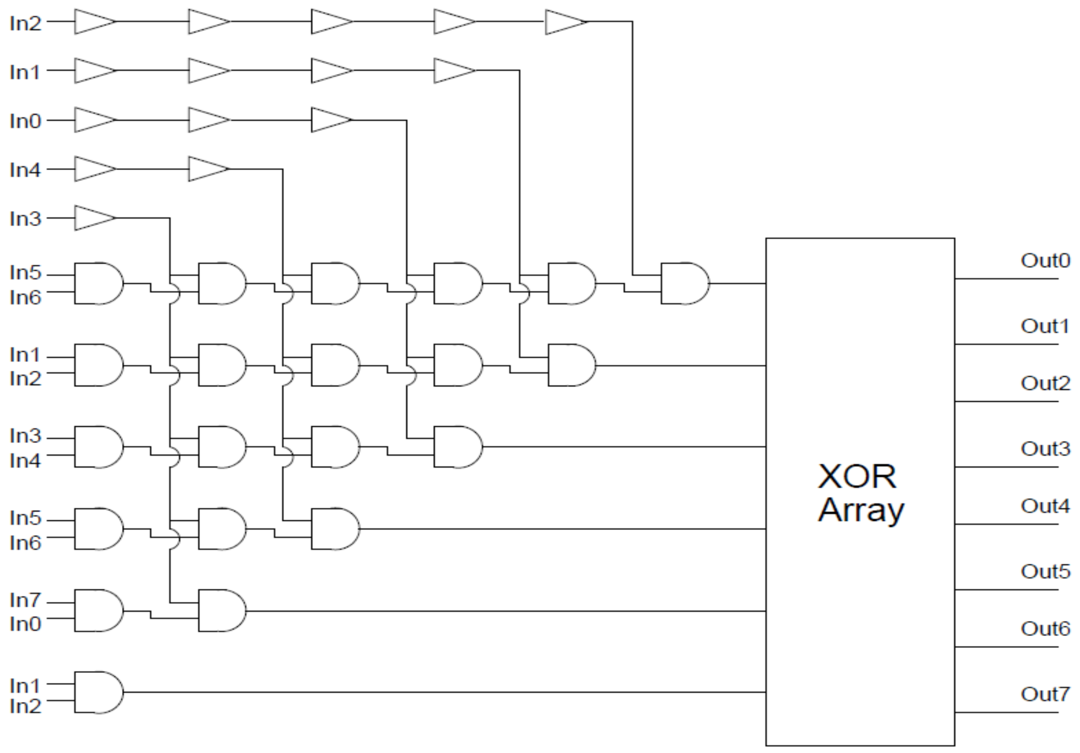
Side Channel Attacks on Cryptographic Circuits



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

Vulnerability: Timing-based attack can reveal secret input In2

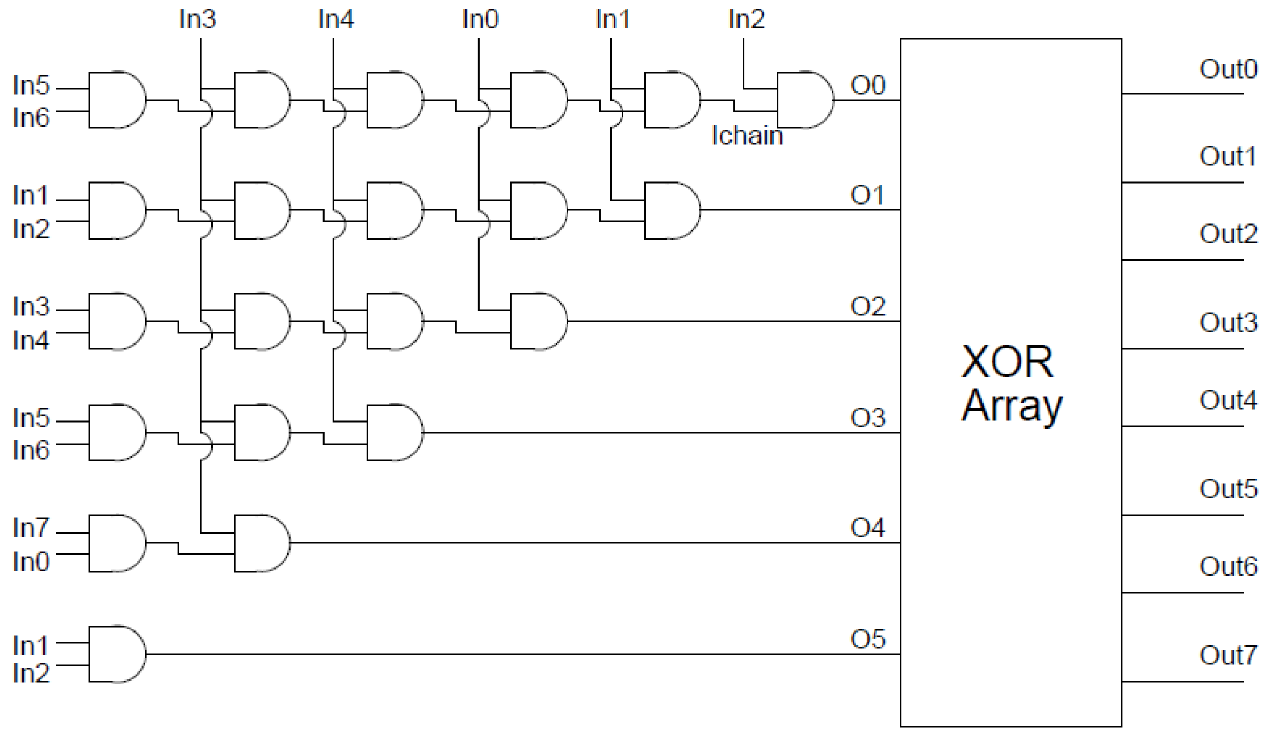
Countermeasure to Attack



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

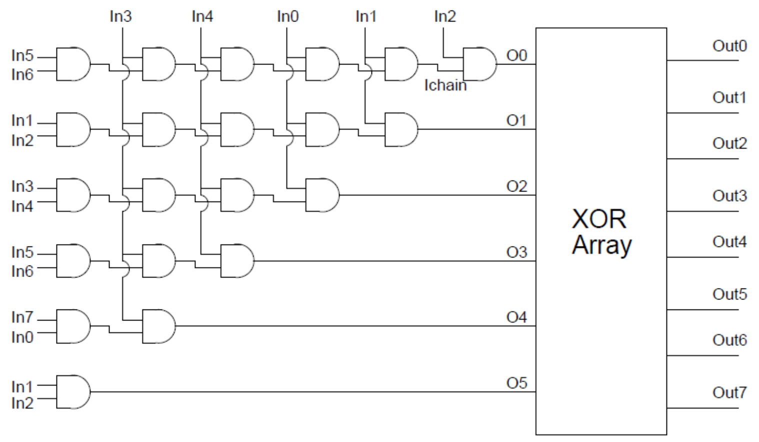


Given a circuit C, automatically synthesize a circuit C′ such that

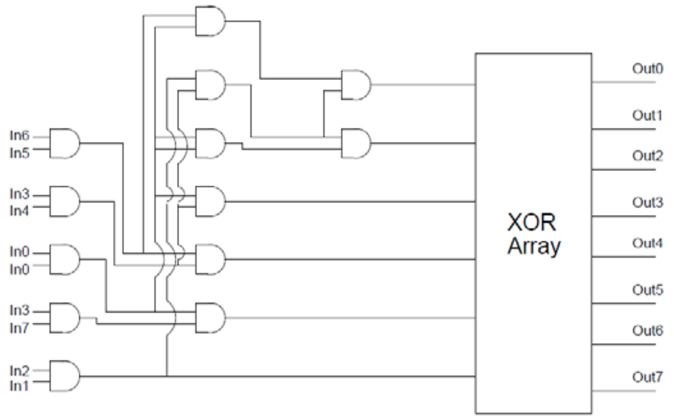
1. C′ is functionally equivalent to C [sematic constraint]
2. All input-to-output paths in C′ have same length [syntactic constraint]

Existing EDA tools cannot handle this synthesis problem

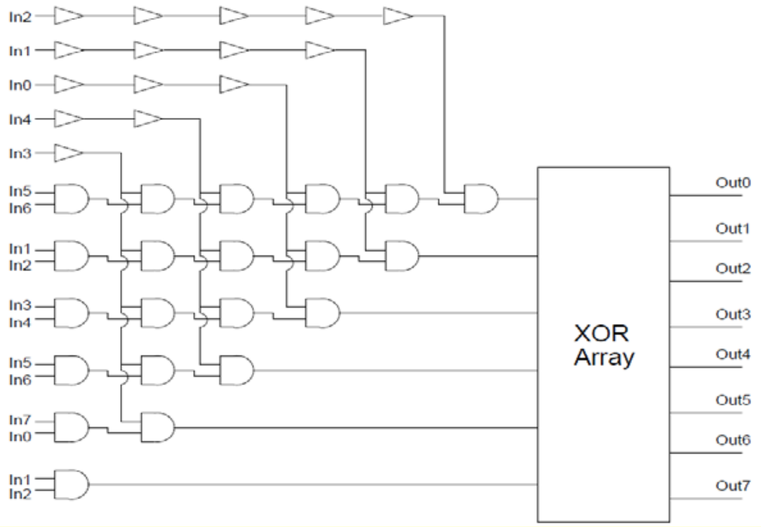
SyGuS Result



Original ckt prone to attack



SyGuS-generated Attack resilient ckt



Hand-crafted attack resilient ckt

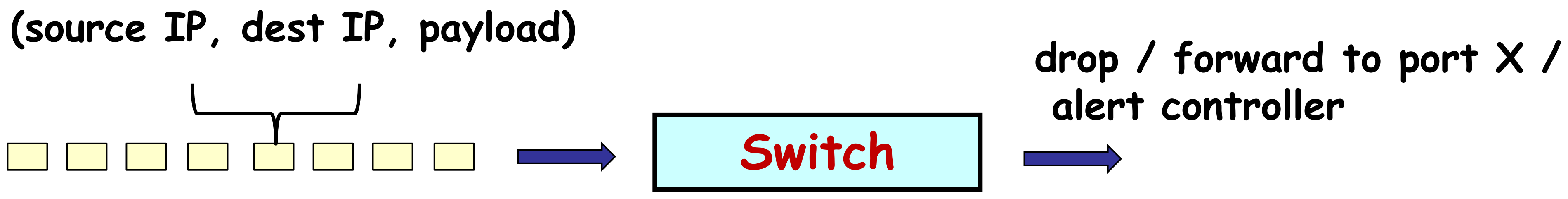
Fully automatic
Smaller size
Shorter delays

Part V

Application: Network Traffic Classification

Sharingan: Network traffic classification by program synthesis;
Collaborators: Shi, Loo; TACAS 2021

Network Traffic Engineering



Dynamic network management for traffic engineering requires writing classification rules to identify anomalous traffic

Example attack: DDoS (flood attack to exhaust available resources)

Too many open connections within a short time interval from SourceIPs in a close range

Synthesis of Network Traffic Classifiers

Can we learn a traffic classifier from positive/negative examples?

Challenges:

- Example: (very large) sequences of raw network packets
- Very few examples, particularly of anomalous traffic
- Operators need to manually examine potential false reports
- Synthesized classifier should be interpretable (easy to decipher / edit)
- Synthesized classifier should be efficiently implementable
- Should be expressive to capture application-layer protocols …

Active research area:
automatic generation of classifiers using machine learning and data mining

Kitsune: An ensemble of Autoencoders for online network intrusion detection; Mirsky et al; NSDI 2018

Syntax-Guided Synthesis Based Solution

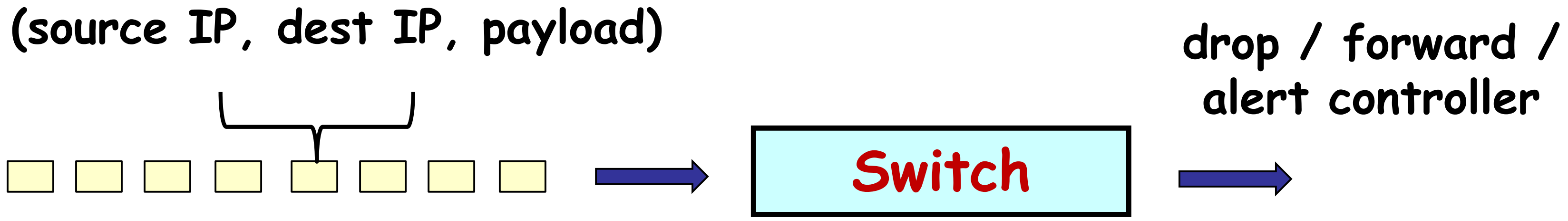
1. Choose a suitable DSL for traffic classifiers
   - Expressive enough to capture a variety of attacks
   - Built-in abstractions for succinct descriptions

2. Design a synthesis tool that, given positive and negative examples, finds a succinct expression in DSL consistent with examples
   - Scalable enough to generate interesting classifiers
   - Capable of handling traffic traces of thousands of packets

Program synthesis based approach, in principle, can meet challenges of network traffic classification better than ML-based solutions

Our solution: Sharingan

Choosing a DSL for Classifiers


(source IP, dest IP, payload)
Switch
drop / forward / alert controller

Example classifier:

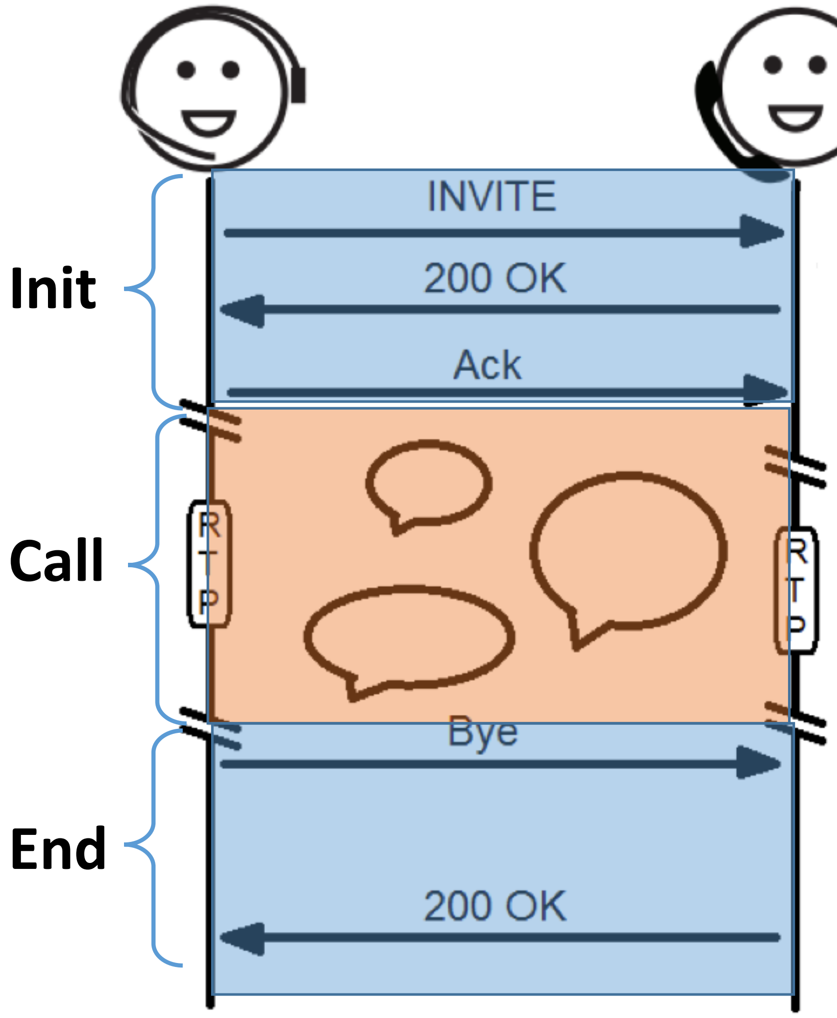
if number of packets in current VoIP session exceeds the average over past VoIP sessions by a standard deviation then drop the packet

Low-level programming:

What state to maintain? How to update it?

Desired high-level abstraction: Beyond packet sequence

High-level Specification of VoIP Session Monitor



Session Initiation Protocol

1. Focus on traffic between a specific source and destination
2. View data stream as a sequence of VoIP sessions
3. View a VoIP session as a sequence of three phases
4. Aggregate cost over call phase during a session, and aggregate cost across sessions

NetQRE Language for Network Traffic Classifiers

SIGCOMM 2017 (with Y. Yuan and B.-T. Loo)

- Core language: QRE (Quantitative Regular Expressions)
  - Regular expressions
  - Aggregate operators such as max, min, sum, average, …
- Domain-specific features
  - Ports, IP addresses, Range types
  - Group-by construct on IP-address keys
  - Tests on packet fields
  - References to time windows (e.g. packets in last 5 sec)
- Semantics of an expression: maps packet stream to a numerical value
- Efficient compiler and runtime system: each expression can be compiled into optimized code (with theoretical guarantees of how much state is stored)

NetQRE Examples

- Flow-level traffic measurements
  - e.g. detection of heavy hitters, super spreaders
- TCP state monitoring
  - e.g. aggregate statistics of TCP connections
  - detect SYN flood attack
- Application level monitoring
  - e.g. collect statistics about VoIP sessions
  - 18 lines of NetQRE code (vs 100s lines of C++)
- DDoS attack classifier (synthesized by Sharingan)

```
( ( /_* A _* B _*/ )*sum /_* C _*/ )sum > 4
where   A = [ip.src_ip ->[0%,50%]]
        B = [tcp.rst == 1]
        C = [time_since_last_pkt <= 50%]
```

Synthesis of NetQRE Expressions from Examples

- Given sets P and N of positive and negative packet traces, find a NetQRE expression that separates them
  - Recall: An expression maps a packet trace to a numerical value

- Key challenges for synthesis
  - Search space is large due to a rich set of constructs
  - Expressions involve numerical constants
  - Packet traces are long (thousands of packets)

- Ideas for optimized search from SyGuS solvers are relevant, yet can't use an off-the-shelf solver
  - No SMT theory of quantitative regular expressions

- Search in Sharingan
  - Optimized enumerative search (as in former SyGuS solvers)
  - Merge search: Divide-and-conquer solution to handle long traces
  - Partial evaluation for early pruning

Partial Evaluation

- Partial NetQRE expression e
  - The expression e still contains some non-terminals
  - Generated by the NetQRE grammar

- Partial evaluation of e on a packet trace t: interval [l, u]
  - Requirement: for every completion f of e, f(t) should belong to [l,u]

- Partial evaluation of e on all positive and negative examples may allow us to conclude that no completion of e can separate them
  - This early pruning critical for performance of synthesis tool

- Key technical challenge: how to do partial evaluation efficiently

Experimental Evaluation

- Benchmarks from CICIDS2017 database
  - 8 types of attacks: Slowloris, Slowhttps, DoS Hulk, SSH Patator, HTP Patator, DDoS, Botnet ARES, Portscan
- Accuracy of Sharingan:
  - In 6 out of 8 attacks, 100% true positive rate at 1% false positive rate
  - In 7 out of 8 attacks, above 0.994 of AUC-ROC
  - Comparable to existing approaches such as Kitsune
- Synthesizes short NetQRE programs
  - Can be interpreted
  - Editing possible: threshold can be adjusted manually
- Synthesized program can be deployed directly (or translated to rules)
- Synthesis time
  - For 7 out of 8: within 50 minutes; BotNET ARES: 300 minutes
  - Size of synthesized expressions: 20 – 30 terms
  - Optimizations (merge search and partial evaluation) critical for performance

Part VI

Conclusions and Research Directions

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 repair/optimization with respect to syntactic constraints
- ❑ Annual competition (SyGuS-comp)
  - Standardized interchange format + benchmarks repository

What next: Infrastructure

- Synthesis is a hot topic as measured by number of publications in CAV/POPL/PLDI and even database / machine learning / software engineering conferences.
  - Many use SyGuS benchmarks for evaluation
  - Some of these papers do use SyGuS or Sketch (Solar-Lezama et al) or Rosette (Torlak et al), yet many are stand-alone efforts
- Future direction: integrate constraint solving and synthesis in mainstream compilers such as LLVM
- Future direction: SyGuS-like back-end focused on efficient search, but decoupled from SMT solvers so as to allow interface with alternative testing / verification tools

What next: Applications

- PBE-based tools for automating repetitive code transformations
  - Sustained effort at Microsoft (see Bluepencil)
  - Key challenge: programmer interaction
- Low-level code optimizations for heterogeneous platforms
  - New effort on Machine Programming at Intel
- From data to logic (interpretable machine learning)
  - Synthesis of logic programs from data

What next: Solution Techniques

- How to make synthesis scalable? No magic bullet here, but slow and steady progress will continue…

- Exciting opportunity: Machine learning for program synthesis
  - Challenges in ML research: how to learn structured objects, how to integrate symbolic constraints in training of neural networks
  - Challenges for program synthesis: what's a suitable representation of a program
  - Lots of papers in ML conferences in last 2-3 years (e.g. Hoppity: Learning graph transformations to detect and fix bugs in programs; Dinella et al; ICLR 2020)

IDEs of Future

Program Synthesis

Machine Learning

Human-Computer Interaction