Syntax-Guided Synthesis

Rajeev Alur

University of Pennsylvania
Program Verification

Verifier

Specification S
Program P

Proof of correctness or Witness of a bug
Classical Program Synthesis

Specification S
High Level
“WHAT”

Synthesizer

Program P
Low Level
“HOW”
Syntax-Guided Synthesis

Specification $S$ given by logical constraints

Syntactic restrictions $R$ on the space of programs

Synthesizer

Program $P$
Outline

- Motivating Examples
  - Formalization of SyGuS
  - Solving SyGuS
  - SyGuS Competition and Conclusions
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 $\varphi$ (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
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:

\[ x \& (1 + (x \mid (x-1))) \]
### FlashFill: Programming by Examples

Ref: Gulwani (POPL 2011)

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>(425)-706-7709</td>
<td>425-706-7709</td>
</tr>
<tr>
<td>510.220.5586</td>
<td>510-220-5586</td>
</tr>
<tr>
<td>1425 235 7654</td>
<td>425-235-7654</td>
</tr>
<tr>
<td>425 745-8139</td>
<td>425-745-8139</td>
</tr>
</tbody>
</table>

**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 $P'$

```c
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?
Given a ckt $C$, automatically synthesize a ckt $C'$ such that
1. $C'$ is functionally equivalent to $C$ [semantic constraint]
2. All input-to-output paths in $C'$ have same length [syntactic constraint]

Existing EDA tools cannot handle this synthesis problem.
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.`
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: \( \forall k : 0 \leq k < n \Rightarrow A[k] \leq A[k + 1] \)
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: ∀k : 0 ≤ k < n ⇒ A[k] ≤ 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: ∀k : 0 ≤k<n ⇒ A[k]≤A[k + 1]

Invariant:
∀k1,k2. 0≤k1<k2<n ∧
          k1<i ⇒ A[k1]<A[k2]

Invariant:
∀k1,k2. 0≤k1<k2<n ∧
          k1<i ⇒ A[k1]<A[k2] ∧
          (∀k1,k2. 0≤k1<k2<n ∧
           k1<i ⇒ A[k1]<A[k2]) ∧
          (∀k. i1≤k<j ∧
           k≥0 ⇒ A[v]≤A[k])
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 $\varphi$ (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

SMT-LIB Standardized Interchange Format (smt-lib.org)
Problem classification + Benchmark repositories
LIA, LIA_UF, LRA, QF_LIA, ...

+ Annual Competition (smt-competition.org)
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 \(\varphi\)
    - 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: \( f(\text{int } x, \text{int } y): \text{int} \)

- Specification: \( (x \leq f(x,y)) \land (y \leq f(x,y)) \)

- Candidate Implementations: Linear expressions
  \( \text{LinExp} := x \mid y \mid \text{Const} \mid \text{LinExp} + \text{LinExp} \mid \text{LinExp} - \text{LinExp} \)

- No solution exists
SyGuS Example

- Theory QF-LIA

- Function to be synthesized: \( f (\text{int } x, \text{int } y) : \text{int} \)

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

- Candidate Implementations: Conditional expressions without +

  \[
  \text{Term} := x \mid y \mid \text{Const} \mid \text{If-Then-Else} (\text{Cond}, \text{Term}, \text{Term}) \\
  \text{Cond} := \text{Term} \leq \text{Term} \mid \text{Cond} \& \text{Cond} \mid \neg \text{Cond} \mid (\text{Cond})
  \]

- Possible solution:
  \[
  \text{If-Then-Else} (x \leq y, y, x)
  \]
(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 \( f(\text{int } x, \text{int } y) : \text{int} \)

\[
T := (\text{let } [z = U] \text{ in } z + z) \\
U := x \mid y \mid \text{Const} \mid (U) \mid U + U \mid U - U
\]
Invariant Generation as SyGuS

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**: \( f(\text{matrix } A, B) : \text{matrix} \)

- **Specification**: \( f(A,B) \) is matrix product
  \[
  \]
  ...

- **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 $f(\text{int } x): \text{int}$
  
  \[ x \leq f(x) \land -x \leq f(x) \]

- 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 up to 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:

\[(a_1 + a_3 < 1) \& (a_2 <= 1) \& (a_4 = 0) \& (a_5 = 0)\]

\[a_1, a_2, a_3, a_4, a_5: \text{water quantities starting at some bucket}\]

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:

\[(a_1 + a_3 < 1) \& (a_2 \leq 1) \& (a_4 = 0) \& (a_5 = 0)\]

\[b_1, b_2, b_3, b_4, b_5: \text{water quantities after stepmother's turn}\]

Claim: \[b_1 + b_3 + b_4 + b_5 < 2\]

Either \[(b_1 + b_4 < 1)\] or \[(b_3 + b_5 < 1)\]

Suppose \[(b_1 + b_4 < 1)\]. Other case similar.

Cinderella strategy: empty buckets 2 and 3.

We have: \[(b_4 + b_1 < 1) \& (b_5 \leq 1) \& (b_2 = 0) \& (b_3 = 0)\]
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 $\varphi$
    - 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

Initial examples I

Learning Algorithm

Verification Oracle

Candidate Expression

Counterexample

Fail

Success

Concept class: Set E of expressions

Examples: Concrete input values
Counterexample-Guided Inductive Synthesis

- Specification: \((x \leq f(x, y)) \land (y \leq f(x, y))\)

- Set E: All expressions built from \(x, y, 0, 1, \text{Comparison}, +, \text{If-Then-Else}\)

Examples = {}  

Candidate  
f(x, y) = x  

Verification Oracle  

Example  
\((x=0, y=1)\)  

Learning Algorithm
CEGIS Example

- Specification: \((x \leq f(x,y)) \land (y \leq f(x,y))\)

- Set E: All expressions built from \(x, y, 0, 1, \text{Comparison}, +, \text{If-Then-Else}\)

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

Candidate \(f(x,y) = y\)

Example \((x=1, y=0)\)
CEGIS Example

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

- **Set E:** All expressions built from \(x, y, 0, 1, \text{Comparison, } +, \text{If-Then-Else}

Examples = 
\(((x=0, y=1) \\
(x=1, y=0) \\
(x=0, y=0) \\
(x=1, y=1))\)

Candidate
ITE \((x \leq y, y, x)\)

**Learning Algorithm**

**Verification Oracle**

Success
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 $\neg \varphi(f, x)$ 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: \( E := x \mid y \mid 0 \mid 1 \mid E + E \) 
- 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 $n$. $P(e) \propto \text{Score}(e)$ (“Extent to which $e$ meets the spec $\varphi$”)

- For a given set Examples, $\text{Score}(e) = \exp(-0.5 \text{Wrong}(e))$, where $\text{Wrong}(e) = \text{No of inputs in Examples for which } \sim \varphi [f/e]$

- $\text{Score}(e)$ is large when $\text{Wrong}(e)$ is small. Expressions $e$ with $\text{Wrong}(e) = 0$ more likely to be chosen in the limit than any other expression
Stochastic Learning

- Initial candidate expression $e$ sampled uniformly from $E_n$
- When $\text{Score}(e) = 1$, 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 $\min\{1, \frac{\text{Score}(e')}{\text{Score}(e)}\}$, replace $e$ with $e'$
- Outer loop responsible for updating expression size $n$
SyGuS Solvers ↔ Synthesis Tools

- Program optimization
- Program repair
- Programming by examples
- Invariant generation

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: Fisman, Singh, Solar-Lezama
SyGuS Progress

- 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

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 \((f(x,y) \geq x) \& (f(x,y) \geq y)\), 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 \(x \geq y\) and \(f(x,y) = y\) is correct when \(x \leq y\)

- 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

Partial Solutions

0
1
\(x\)
\(y\)

Examples

(1, 1)
(1, 2)
(2, 1)
...

Predicates

0 \(\geq\) 1
1 \(\geq\) 1
\(x \geq 1\)
\(x \geq 2\)
\(x \geq y\)

Step 2: Generate predicates

Step 3: Combine!

if \((x \geq y)\) then \(x\) else \(y\)
Conditional Expression Grammars

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

Term Grammar

Term Enumerator

Terms

Predicate Grammar

Predicate Enumerator

Predicates

Decision Tree Learning

Expression
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. $x \leq y$, ...)
  - 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. $f(x, y) \geq x \& f(x, y) \geq y$)

- 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 $p$,
  \[
  \text{Gain}(p) = \frac{|X_p|}{|X|} \times H(X_p) + \frac{|X_{\sim p}|}{|X|} \times H(X_{\sim p})
  \]
- Split $X$ using the predicate $p$ in $P$ for which $\text{Gain}(p)$ 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

- 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

Model Checking
Searching for bugs in state-space

Constraint Solving
Searching for a satisfying assignment

Syntax-Guided Synthesis
Searching for a correct expression

www.sygus.org