Motivation for Syntax-Guided Synthesis
- Recent trends in program synthesis (the problem)
- The big picture, SAT & SMT success stories (the inspiration)

Formalization of Syntax-Guided Synthesis

Solution Strategies

SyGuS Competition Results
Program Synthesis

Synthesizer

Specification S
High Level
"WHAT"

Program P
Low Level
"HOW"

Turn off rightmost continuous 1 bits

Graph connectivity

Sorting
New Trends in Synthesis

Synthesizer

Specification S
High Level
"WHAT"

Syntactic restrictions on the solutions domain

Use at most two of each of the following operators: &&, <<, ...

Program P
Low Level
"HOW"

Turn off rightmost continuous 1 bits:
1010110 -> 1010000
New Trends in Synthesis

Motivation:

- Tractability
- Combine human expert insights with computers exhaustiveness & rapidness
- Benefit progress SAT & SMT Solvers
Ex 1. Parallel Parking By Sketching

The challenge is finding the parameters

- When to start turning?
- How much to turn?

```c
Err = 0.0;
for(t = 0; t<T; t+=dT){
    if(stage==STRAIGHT){ // (1) Backup straight
        if(t > ??) stage= INTURN;
    }
    if(stage==INTURN){ // (2) Turn
        car.ang = car.ang - ??;
        if(t > ??) stage= OUTTURN;
    }
    if(stage==OUTTURN){ // (3) Straighten
        car.ang = car.ang + ??;
        if(t > ??) break;
    }
    simulate_car(car);
    Err += check_collision(car);
}
Err += check_destination(car);
```

Structure of the program is known

[Chaudhuri & Solar-Lezama PLDI 2010]
Given a program $P$, find a “better” 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.
Ex 3. Automatic Invariant Generation

Given a program $P$ and a post condition $S$, find invariants $I_1, I_2$ with which we can prove program is correct.

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

```
SelecionSort(int A[], n) {
    i1 := 0;
    while (i1 < n - 1) {
        v1 := i1;
        i2 := i1 + 1;
        while (i2 < n) {
            if (A[i2] < A[v1])
                v1 := i2;
            i2++;
        }
        swap(A[i1], A[v1]);
        i1++;
    }
    return A;
}
```

Invariant: ???

Invariant: ????

Post: $\forall k : 0 \leq k < n \Rightarrow A[k] \leq A[k+1]$
Given a program $P$ and a post condition $S$

Find invariants $I_1, I_2, \ldots, I_k$ with which we can prove program is correct

Ex 3. Template-Based Invariant Generation

```java
SelecionSort(int A[], n) {
    i1 := 0;
    while (i1 < n-1) {
        v1 := i1;
        i2 := i1 + 1;
        while (i2 < n) {
            if (A[i2] < A[v1])
                v1 := i2;
            i2++;
        }
        swap(A[i1], A[v1]);
        i1++;
    }
    return A;
}
```

Post: $\forall k: 0 \leq k < n \Rightarrow A[k] \leq A[k+1]$
Syntax-Guided Program Synthesis

- Common theme to many recent efforts
  - Sketch (Bodik, Solar-Lezama et al)
  - FlashFill (Gulwani et al)
  - Super-optimization (Schkufza et al)
  - Invariant generation (Many recent efforts...)
  - TRANSIT for protocol synthesis (Udupa et al)
  - Oracle-guided program synthesis (Jha et al)
  - Implicit programming: Scala^Z3 (Kuncak et al)
  - Auto-grader (Singh et al)

But no way to have a generic solver for all 😞
Talk Outline

- **Motivation for Syntax-Guided Synthesis**
  - Recent trends in program synthesis (the problem)
  - The big picture, SAT & SMT success stories (the inspiration)
- **Formalization of Syntax-Guided Synthesis**
- **Solution Strategies**
- **SyGuS Competition Results**
The Big Picture

- **Assertion Checking:**
  \[ P(i) |\leq S(i) ? \]
- **Program Verification:**
  \[ \forall i: P(i) |\leq S(i) ? \]
- **Constraint Programming:**
  \[ \text{Find } o: o |\leq S(i) \]
- **Program Synthesis:**
  \[ \text{Find } P: \forall i: P(i) |\leq S(i) \]

- **Given list } i \text{ is } P(i) \text{ sorted?}**
- **Given } P \text{ only Spec } S \text{ Spec } S**
- **Return a sorting program } P \text{**}

Slide adapted from a presentation of Viktor Kuncak
### The Big Picture

<table>
<thead>
<tr>
<th>Given</th>
<th>Assertion Checking: $P(i) \models S(i)$?</th>
<th>Program Verification: $\forall i: P(i) \models S(i)$?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prog $P$ Spec $S$</td>
<td>Return a program $P$ implementing \textit{turnoff} rightmost 1's</td>
<td>Program Synthesis: $\exists P: \forall i: P(i) \models S(i)$</td>
</tr>
<tr>
<td>Given only Spec $S$</td>
<td>Return a program $P$ implementing \textit{turnoff} rightmost 1's using only so and so operators</td>
<td>Syntax-Guided Synthesis: $\exists P \in [R]: \forall i: P(i) \models S(i)$</td>
</tr>
</tbody>
</table>
From Satisfiability to Synthesis

Recent trends in program synthesis:

Problem
(verify/synthesize nature)

\[ P(i) \models S(i) \]
\[ \forall i: P(i) \models S(i) \]
\[ \exists o: o \models S(i) \]
\[ \exists P: \forall i: P(i) \models S(i) \]
\[ \exists P \in [R]: \forall i: P(i) \models S(i) \]

Syntactic Restrictions on solution domain

SAT/SMT Solver
Satisfiability

Is formula $\varphi$, satisfiable?

- **SAT**
  Formulas use Boolean variables + Logical connectives

- **SMT** *(Satisfiability Modulo Theories)*
  Formulas are interpreted in some theory, e.g. BitVectors, Integers, Uninterpreted functions.
SAT & SMT Solvers and Verification Tools

- Fundamental Thm of CS:
  SAT is NP-complete (Cook, 1971)
- Nonetheless, today's SAT solvers are capable of handling instances with 20K variables.
- Continuous progress in SMT solvers achieved among other things via
  - SMT-LIB standardization
  - currently over 100,000 benchmarks
  - annual competition
- Many verification tools build on SMT solvers

- CBMC
- VCC
- SAGE
- Spec#

- Z3
- Yices
- CVC4
- MathSat5

The Glue
SyGuS - The Vision

Program Optimization

Program Sketching

Programming by examples

Invariant Generation

???
Talk Outline

- Motivation for Syntax-Guided Synthesis
  - Recent trends in program synthesis (the problem)
  - The big picture, SAT & SMT success stories (the inspiration)

- Formalization of Syntax-Guided Synthesis

- Solution Strategies

- SyGuS Competition Results
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 $T + \text{symbol } f$
  - Context-free grammar $G$
    - Characterizing the set of allowed expressions $[G]$ (in theory $T$)
- Computational problem:
  - Find expression $e$ in $[G]$ such that $\varphi[f/e]$ is valid (in theory $T$)
SyGuS - formalization example

- Theory QF-LIA
  - Types: Integers and Booleans
  - Logical connectives, Conditionals, and Linear arithmetic
  - Quantifier-free formulas

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

- Specification:
  \[
  f(x,y) \geq x \&
  f(x,y) \geq y \&
  ( f(x,y)=x \mid f(x,y)=y )
  \]

- 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 - formalization example

- **Theory QF-LIA**
  
  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**
  
  \( f(x,y) \geq x \ & \ f(x,y) \geq y \ & \ (f(x,y) = x \ | \ f(x,y) = y) \)

- **Candidate Implementations: Conditional expressions with comparisons**

  - **Term**:  \( x \ | \ y \ | \ Const \ | \ If\text{-}Then\text{-}Else (\text{Cond}, \text{Term}, \text{Term}) \)
  - **Cond**:  \( \text{Term} \ \leq \ \text{Term} \ | \ \text{Cond} \ & \ \text{Cond} \ | \ \sim \text{Cond} \ | \ (\text{Cond}) \)

- **Possible solution**

  \( If\text{-}Then\text{-}Else (x \ \leq \ y, \ y, \ x) \)
Let Expressions

- Synthesized expression maps directly to a straight-line program

- Grammar derivations correspond to expression parse-trees

**Problem:**
How to capture common sub-expressions (which map to aux vars)?

**Solution:**
Allow "let" expressions

**Candidate-expressions for a function f(int x, int y) : int**

\[ U := x \mid y \mid \text{Const} \mid (U) \mid U + U \mid U*U \mid (\text{let} ((z \text{ int } U)) \text{ in } U) \]
Talk Outline

- **Motivation for Syntax-Guided Synthesis**
  - Recent trends in program synthesis (the problem)
  - SAT & SMT success stories (the inspiration)

- **Formalization of Syntax-Guided Synthesis**

- **Solution Strategies**

- **SyGuS Competition Results**
Solving SyGuS

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

\[ \exists P \in [G] \colon \forall i \colon P(i) \models S(i) \]

- SyGuS can sometimes be reduced to Quantified-SMT, but not always
  - Set \([G]\) is all linear expressions over input vars \(x, y\)
    SyGuS reduces to \(\exists a, b, c. \forall x, y. \varphi [ f / ax + by + c ]\)
  - Set \([G]\) is all conditional expressions
    SyGuS cannot be reduced to deciding a formula in LIA

- Syntactic structure of the set \([G]\) of candidate implementations can be used effectively by a solver

- Existing work on solving Quantified-SMT formulas suggests solution strategies for SyGuS
Concept class: \( \text{Set } [G] \) of expressions

Examples: Concrete input values
Running Example

- **Specification:**
  
  \[(x \leq f(x,y)) \&
  (y \leq f(x,y)) \&
  (f(x,y) = x \mid f(x,y) = y)\]

- **Syntactic Restrictions:**
  
  all expressions built from \(x,y,0,1,\)
  
  \(<=, =, \Rightarrow, +,\)
  
  If-Then-Else
SyGuS as Active Learning (CEGIS)

- Concrete inputs $I$ for learning
  
  $f(x,y) = \{ (x=a_0,y=b_0), (x=a_1,y=b_1), \ldots \}$

- Learning algorithm proposes candidate expression $e$ such that $\varphi[f/e]$ holds for all values in $I$

- Check if $\varphi[f/e]$ is valid for all values using SMT solver

- If valid, then stop and return $e$

- If not, let $(x=a, y=b, \ldots)$ be a counter-example (satisfies $\sim \varphi[f/e]$)

- Add $(x=a, y=b)$ to tests $I$ for next iteration
SyGuS Solutions

- **CEGIS** - *Counter-Example Guided Inductive Synthesis* approach (Solar-Lezama, Seshia et al)

- Related work: 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): Library components - Gulwani et al (PLDI'11);
  - **Stochastic** (probabilistic walk): Schkufza et al (ASPLOS'13)
  - **Geometric** (learns guarded linear expressions): Garg et al.
Enumerative

- 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
  - E.g. If-Then-Else $(0 \leq x, e_1, e_2)$ considered equivalent to $e_1$ if in current set of Examples $x$ has only non-negative values
  - Only one representative among equivalent sub-expressions needs to be considered for building larger expressions
Symbolic CEGIS

- Recall, in general we cannot use SMT solvers to solve SyGuS

\[ \exists P \in [G]: \forall i: P(i) \models S(i) \]

- Idea:
  We can use the constraint solver also in the learning algorithm by restricting the size/depth of the expressions

- We need a way to encode the restricted set of expressions, and if no such expression exists, increase the size

- Two versions:
  - Symbolic [Jha et al.]
  - Sketch-Based [Solar-Lezama et al.]
Each production in the grammar is thought of as a library component.

A well-typed loop-free program comprising these component corresponds to an expression DAG from the grammar.
Symbolic [Jha et al.]

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

- Synthesis Constraints:
  - Program composed of library’s component, Shape is a DAG, Types are consistent
  - Spec $\varphi[f/e]$ is satisfied on every concrete input values in Examples

- Use an SMT solver to find a satisfying solution.
  - If synthesis fails, try increasing the number of occurrences of components in the library in an outer loop.
Stochastic [adaptation of Schufza et al.]

Idea:
Find desired expression $e$ by probabilistic walk on graph where nodes are expressions and edges capture single-edits (for a fixed expression size $n$)
Metropolis-Hastings Algorithm: Given a probability distribution $P$ over domain $X$, and an ergodic Markov chain over $X$, samples from $X$.

Because the graph is strongly connected, we can reach each node with some probability.

Let $\text{Score}(e)$ be the “Extent to which $e$ meets the spec $\varphi$”

Having $P(e) \propto \text{Score}(e)$ we increase the chances of getting to expressions with better score.

To escape “local minima” we allow with some probability moving to expressions with lower score.

Specific choice of score:
For a given set $I$ of concrete inputs, $\text{Score}(e) = \exp(-\frac{1}{2} \text{Wrong}(e))$
where $\text{Wrong}(e) = \text{No of examples in } I \text{ for which } \sim \varphi [f/e]$

$\text{Score}(e)$ is large when $\text{Wrong}(e)$ is small

$\Rightarrow$ Expressions $e$ with $\text{Wrong}(e) = 0$ more likely to be chosen in the limit than any other expr
Stochastic

- 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 $v$ with subtree of same size, sampled uniformly

- With probability $\min\{1, \text{Score}(e')/\text{Score}(e)\}$, replace $e$ with $e'$

- Outer loop responsible for updating expression size $n$
Expr size = 6
All expressions of this size have the following form
One chosen at random, with probability \( \frac{1}{(4 \times 4 \times 4 \times 4 \times 3)} = \frac{1}{768} \)
Suppose it is \( e = \text{ITE}(x \leq 0, y, x) \)
Next a sub-expression (node) to mutate is chosen at random
Probability to mutate node 2 is \( \frac{1}{6} \)
There are \( \frac{1}{(4 \times 4 \times 3)} = \frac{1}{48} \) conditional expression node 2 can be replaced with
Choice \( e' = \text{ITE}(0 \leq y, y, x) \) is considered with probability \( \frac{1}{(48 \times 6)} = \frac{1}{288} \)
If concrete examples are \((-1, 4), (-3, -1), (-1, -2), (1, 2), (3, 1), (6, 2)\)

With |
Geometric

- **Idea:** Learns only guarded linear expression
  - Agnostic to the SyGuS constraints
  - Learns only from the examples
  - Examples are assumed to be given as \((x, \text{val})\) where \(x\) is a vector \((x_1, x_2, \ldots, x_k)\) of variables

- **Process:**
  
  **Phase 1 [Geometry]:**
  - Synthesizes linear expressions \(LE_1, LE_2, \ldots, LE_k\), such that each example is evaluated correctly by at least one.
  - Essentially finds a small set of planes that includes all given examples

  **Phase 2 [Classifier]:**
  - Synthesizes the guards using a fast and scalable machine learning algorithm for decision trees

*Does not work for partially-specified problems, or problems which can not be implemented by guarded linear expressions or the grammar disallows it.*
Motivation for Syntax-Guided Synthesis
- Recent trends in program synthesis (the problem)
- SAT & SMT success stories (the inspiration)

Formalization of Syntax-Guided Synthesis

Solution Strategies

SyGuS Competition Results
SyGuS Competition

- **Competition** of SyGuS solvers at **FLoC Olympic Games**, 30 June 2014

- Competition was conducted **offline**, using **StarExec**

- **Single track**, solvers should support **full SyGuS-IF** format for the theories of **bit-vector** and **integer linear arithmetic**.

- **Benchmarks**:
  - Bit-manipulation programs from Hacker’s delight
  - Integer arithmetic: Find max, search in sorted array
  - Challenge problems such as computing Morton’s number
  - Invariant generation problem
  - ...

Percentage of Benchmarks Solved

Total:
254 benchmarks

Total solved:
148 benchmarks

Number of Benchmark Solved out of the Total per Category

Benchmarks arranged in 10 Categories
Number of Benchmarks Solved

5 solvers participated in the competition
For each of the solvers there were some benchmarks only it solved.
Vehicle Control

Solving times arranged in buckets

Relative Performance on the Vehicle Control Benchmarks

- Enumerative
- Stochastic
- Sketch
- Alchemist
- Symbolic
## Interesting Variations

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>array_sum_2.5.sl</td>
<td>-</td>
<td>-</td>
<td>31</td>
<td>139</td>
<td>-</td>
<td>Alchemist</td>
<td>9.012</td>
</tr>
<tr>
<td>hd-17-d5-prog.sl</td>
<td>8</td>
<td>73</td>
<td>-</td>
<td>-</td>
<td>9</td>
<td>Enumerative</td>
<td>5.653</td>
</tr>
<tr>
<td>LinExpr_eq1.sl</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1056</td>
<td>Alchemist</td>
<td>1128.870</td>
</tr>
<tr>
<td>LinExpr_eq2.sl</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>180</td>
<td>Alchemist</td>
<td>303.780</td>
</tr>
<tr>
<td>max4.sl</td>
<td>-</td>
<td>39</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Stochastic</td>
<td>95.683</td>
</tr>
<tr>
<td>VC22_f2.sl</td>
<td>14</td>
<td>118</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Enumerative</td>
<td>5.475</td>
</tr>
<tr>
<td>VC22_c2.sl</td>
<td>-</td>
<td>425</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Stochastic</td>
<td>128.635</td>
</tr>
<tr>
<td>VC22_a.g.sl</td>
<td>6</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Enumerative</td>
<td>0.016</td>
</tr>
<tr>
<td>fivefuncs.sl</td>
<td>-</td>
<td>-</td>
<td>35</td>
<td>-</td>
<td>43</td>
<td>Symbolic</td>
<td>2.540</td>
</tr>
<tr>
<td>sixfuncs.sl</td>
<td>-</td>
<td>-</td>
<td>122</td>
<td>-</td>
<td>-</td>
<td>Sketch-based</td>
<td>233.447</td>
</tr>
</tbody>
</table>
Results Summary

- **Enumerative** has better performance and solves many benchmarks within seconds (and always produces a minimal expr)
  
  Potential problem: Synthesis of complex constants

- All solvers succeeded to solve some benchmarks uniquely.

- Choice of grammar has impact on synthesis time
  
  When E is set of much larger than necessary solvers struggle

- None of the solvers succeed on some benchmarks
  
  Morton constants, ICFP, Search in integer arrays of size > 4

**Bottomline:** Improving solvers is a great opportunity for research!
Future Plans

- Competition on July 2015
- Easier tracks,
  - maybe restricted syntax, only guarded linear expressions
  - only complete specifications
- Suggestions?
  - think of something later?
    write us to sygus-organizers@seas.upenn.edu.
The End

Thank you!

Comments or questions?