Can grandmas program?

• The development of programming languages is to raise the level of abstraction

What is the next?

- Haskell (1990), Prolog (1972)
- Java
- C
- Assembly

Level of Abstraction
Why cannot?

• Programming languages come with many guarantees
  • Well-typed programs are guaranteed to compile
  • Compiled programs have clear, well-defined semantics

• It is difficult to further raise the level of abstraction
Program Synthesis saves grandmas

• Generate a program from a specification
  • Specification can be fuzzy
  • Generation is not guaranteed

“One of the most central problems in the theory of programming.”
----Amir Pneuli
  Turing Award Recipient

“The fundamental way to improve software productivity.”
----Jiafu Xu
  Founder of Software Research in China
History of Program Synthesis

1957
• Start of program synthesis
• Circuit synthesis problem by Alonzo Church

Before 2000
• Deductive Synthesis

After 2000
• Inductive Synthesis
# Application – Data Wrangling

<table>
<thead>
<tr>
<th>Email</th>
<th>Column 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="mailto:Nancy.FreeHafer@fourthcoffee.com">Nancy.FreeHafer@fourthcoffee.com</a></td>
<td>nancy freehafer</td>
</tr>
<tr>
<td><a href="mailto:Jan.Kotas@litwareinc.com">Jan.Kotas@litwareinc.com</a></td>
<td>jan kotas</td>
</tr>
<tr>
<td><a href="mailto:Mariya.Sergienko@gradicdesigninstitute.com">Mariya.Sergienko@gradicdesigninstitute.com</a></td>
<td>mariya sergienko</td>
</tr>
<tr>
<td><a href="mailto:Steven.Thorpe@northwindtraders.com">Steven.Thorpe@northwindtraders.com</a></td>
<td>steven thorpe</td>
</tr>
<tr>
<td><a href="mailto:Michael.Nepper@northwindtraders.com">Michael.Nepper@northwindtraders.com</a></td>
<td>michael nepper</td>
</tr>
<tr>
<td><a href="mailto:Robert.Zare@northwindtraders.com">Robert.Zare@northwindtraders.com</a></td>
<td>robert zare</td>
</tr>
<tr>
<td><a href="mailto:Laura.Giussani@adventure-works.com">Laura.Giussani@adventure-works.com</a></td>
<td>laura giussani</td>
</tr>
<tr>
<td><a href="mailto:Anne.HL@northwindtraders.com">Anne.HL@northwindtraders.com</a></td>
<td>anne hl</td>
</tr>
<tr>
<td><a href="mailto:Alexander.David@contoso.com">Alexander.David@contoso.com</a></td>
<td>alexander david</td>
</tr>
<tr>
<td><a href="mailto:Kim.Shane@northwindtraders.com">Kim.Shane@northwindtraders.com</a></td>
<td>kim shane</td>
</tr>
<tr>
<td><a href="mailto:Manish.Chopra@northwindtraders.com">Manish.Chopra@northwindtraders.com</a></td>
<td>manish chopra</td>
</tr>
<tr>
<td><a href="mailto:Gerwald.Oberleitner@northwindtraders.com">Gerwald.Oberleitner@northwindtraders.com</a></td>
<td>gerwald oberleitner</td>
</tr>
<tr>
<td><a href="mailto:Amr.Zaki@northwindtraders.com">Amr.Zaki@northwindtraders.com</a></td>
<td>amr zaki</td>
</tr>
<tr>
<td><a href="mailto:Yvonne.McKay@northwindtraders.com">Yvonne.McKay@northwindtraders.com</a></td>
<td>yvonne mckay</td>
</tr>
<tr>
<td><a href="mailto:Amanda.Pinto@northwindtraders.com">Amanda.Pinto@northwindtraders.com</a></td>
<td>amanda pinto</td>
</tr>
</tbody>
</table>
Application – Superoptimization

```
i = round(i);
a = 6755399441055744.0;
i = (i + a) - a;
```
Application – Reducing Duplicated Programming

```python
class AcidicSwampOoze(MinionCard):
    def __init__(self):
        super().__init__("Acidic Swamp Ooze", 2,
                         CHARACTER_CLASS.ALL, CARD_RARITY.COMMON,
                         battlecry=Battlecry(Destroy(),
                                             WeaponSelector(EnemyPlayer())))

    def create_minion(self, player):
        return Minion(3, 2)
```
Application – Program Repair

/** Compute the maximum of two values
 * @param a first value
 * @param b second value
 * @return b if a is lesser or equal to b, a otherwise */

public static int max(final int a, final int b) {
    return (a <= b) ? a : b;
}

Synthesize an expression to replace the buggy one
Synthesize a unit test to cover a path
Synthesize a strategy for a class of problems

Application – Analysis

SMT Solver

Apply Tactic 1
If formula is long
Apply Tactic 2
Else
Apply Tactic 3

Strategies
## Defining Program Synthesis

### Classic Synthesis
- **Input:**
  - A specification
- **Output:** A program that
  - meets the specification

### Program Optimization
- **Input:**
  - A specification
  - A cost function
- **Output:** A program that
  - meets the specification,
  - and
  - maximizes the cost function

### Program Estimation
- **Input:**
  - A specification
  - A dataset for target distribution
- **Output:** A program that
  - meets the specification and
  - maximizes the probability represented by the dataset

### Test Generation

### Superoptimization

### Program Repair
This Lecture

<table>
<thead>
<tr>
<th>Classic Synthesis</th>
<th>Program Estimation</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Problem Definition</td>
<td>• Problem Definition</td>
</tr>
<tr>
<td>• Enumerative</td>
<td>• Estimating Probabilities</td>
</tr>
<tr>
<td>• Presentation-based</td>
<td>• Locating the most-likely one</td>
</tr>
<tr>
<td>• Constraint-based</td>
<td></td>
</tr>
</tbody>
</table>
SyGuS: Syntax-Guided Synthesis

• A standardization of classic program synthesis problem.

• Input:
  • grammar $G$
  • specification $S$

• Output:
  • program $P$
  • such that $P \in G \land P \rightarrow S$
Example: max

• Grammar:

\[
\text{Expr} ::= \begin{array}{c}
  x \\
  y \\
  \text{Expr} + \text{Expr} \\
  \text{ite} \ \text{BoolExpr} \ \text{Expr} \ \text{Expr} \\
\end{array}
\]

\[
\text{BoolExpr} ::= \begin{array}{c}
  \text{BoolExpr} \land \text{BoolExpr} \\
  \neg \text{BoolExpr} \\
  \text{Expr} \leq \text{Expr} \\
\end{array}
\]

• Specification:

\[
\forall x, y : \mathbb{Z}, \quad \max_2 (x, y) \geq x \land \max_2 (x, y) \geq y \\
\land (\max_2 (x, y) = x \lor \max_2 (x, y) = y)
\]

• Expected answer: \text{ite} (x \leq y) y x
SyGuS format: Synth-Lib

• Synth-Lib uses a format similar to SMT-Lib
  • http://sygus.seas.upenn.edu/files/SyGuS-IF.pdf

```
(set-logic LIA)
(synth-fun max2 ((x Int) (y Int)) Int
   ((Start Int (x y
        (+ Start Start)
        (ite StartBool Start Start)))......))

(declare-var x Int)
(declare-var y Int)
(constraint (>= (max2 x y) x))
......

(check-synth)
```
Program Synthesis as a Search Problem

Q1: How to generate the next program to be verified?

Q2: How to verify the correctness?
Q1: How to verify correctness?

• If the specification includes only tests,
  • test the program.  
  
  Fast

• If the specification is a logic constraint $S$,
  • verify $Program \rightarrow S$ by an SMT solver.
  • Synth-lib directly supports this

  Slow

Can we combine the two?
CEGIS: Counter-Example Guided Inductive Synthesis

- Constraint solvers give counter-examples
- Save counter-examples as tests
- First use tests to validate programs
Q2: How to generate the next program to be verified?

- Enumerative – exhaustive search
- Representation-based – manipulate sets of programs instead of single programs
- Constraint-based – convert to an SMT problem
Top-Down Enumeration

• Expand according to the grammar
  • Expr
  • x, y, Expr+Expr, if(BoolExpr, Expr, Expr)
  • y, Expr+Expr, if(BoolExpr, Expr, Expr)
  • Expr+Expr, if(BoolExpr, Expr, Expr)
  • x+Expr, y+Expr, Expr+Expr+Expr+Expr, if(BoolExpr, Expr, Expr)+Expr, if(BoolExpr, Expr, Expr)
  • ...

Bottom-Up Enumeration

• Combine expressions from small to big
  • size=1
    • x, y
  • size=2
  • size=3
    • x+y
  • size=4
  • size=5
    • x+(x+y), (x+y)+y
  • size=6
    • if(x<=y, x, y), ...
Optimization

• Discard a partial program early

• Pruning
  • None of the expansions could satisfy the specification
  • `Ite BoolExpr x x`

• Equivalence reduction
  • Equivalent to a previous program
  • `Expr+x, x+Expr`
Pruning

• Generate constraints from the partial program

\[
\text{Ite BoolExpr } x \ x \quad \rightarrow \quad \text{(declare-fun boolExpr () Int)}
\]
\[
\text{(declare-fun max2 ((x Int) (y Int)) Int)}
\]
\[
\text{(ite boolExpr } x \ x)\)
\]

• Generate constraints from each test

\[
\text{max2(1,2)=2} \quad \rightarrow \quad \text{(assert (= (max2 1 2) 2))}
\]

(\text{check-sat)}

Needs to balance between the benefit and the cost.
Equivalence reduction: How to determine equivalence?

• With an SMT solver
  • Check satisfiability of $f(x, y) \neq f'(x, y)$
  • The cost may not pay off

• With tests
  • Check if $f = f'$ on all tests
  • Not safe for logic specifications
  • Does not work on partial programs

• With predefined-rules
  • e.g Expr+x and x+Expr
  • Needs customization for each domain
How to generate the next program to be verified?

• Enumerative – exhaustive search
• Representation-based – manipulate sets of programs instead of single programs
• Constraint-based – convert to an SMT problem
Representation-based

• Enumerative approaches manipulates single programs
  • Inefficient: too many in number

• Can we manipulate sets of programs? e.g.
  • Find a set that satisfies a specification
  • Intersects sets for a conjunction of specifications
  • Combine sets with program constructs to satisfy more complex specifications

• Representation-based
  • Use data structures to represent such a set
  • E.g. Grammars, Automata, Logic Formulas
FlashMeta: Basic Idea

• Grammar is a representation of sets
  • Size of a grammar = \(O(\log(#\text{Represented Program}))\)

• The original grammar is too coarse-grained

• Idea: Annotate a non-terminal with a synthesis goal
  • \([2]\text{Expr} – \text{expressions that evaluates to 2}\)
FlashMeta: Single Test

• Pick a test
  • max2(1,2) = 2

• Refine the grammar
  • [1]Expr → x | ...
  • [true]BoolExpr → ¬[false]BoolExpr
    | [true]BoolExpr ∧ [true]BoolExpr
  • ...
  • Assume a user-provided operation to perform the refinement

• Any program represented by the grammar passes the test
Intersection of grammars

• Suppose
  - $N \rightarrow P_1 | \cdots | P_k$
  - $N' \rightarrow P'_1 | \cdots | P'_{k'}$

• $N \cap N' = P_1 \cap P'_1 | P_1 \cap P_2' | \cdots | P_1 \cap P'_{k'}$,
  - $| P_2 \cap P'_1 | P_2 \cap P'_2 | \cdots | P_2 \cap P'_{k'}$
  - $| \cdots$
  - $| P_k \cap P'_1 | P_k \cap P'_2 | \cdots | P_k \cap P'_{k'}$

• $P_1 \cap P_2 = \emptyset$ if $P_1$ and $P_2$ are of different types

• $f(N_1, \ldots, N_k) \cap f(N'_1, \ldots, N'_k) = f(N_1 \cap N'_1, \ldots, N_k \cap N'_k)$
FlashMeta: Multiple Tests

- Produce a grammar for each test
- Intersects the grammars
FlashMeta: Discussion

- Avoids duplicated computation
  - \([1]Expr + [1]Expr\)
  - \([1]Expr\) is explored only once in FlashMeta

- Pruning is naturally included
  - \([1]Expr → Expr + Expr\)

- Needs user-provided operation for refinement
  - \([65536]Expr\)

- Trivia: original paper uses version space algebra, which is essentially grammar
How to generate the next program to be verified?

• Enumerative – exhaustive search
• Representation-based – manipulate sets of programs instead of single programs
• Constraint-based – convert to an SMT problem
Component-Based Program Synthesis

Connection Points

Components

Label variables:
- \(l_{i11}, l_{i22}, \ldots\)
- \(l_{o1}, l_{o2}, \ldots\)
- \(l_{o}:\) program output

\[l_{o6} = l_{i31} = 4\]
Generate constraints

• Test
  • $o6 = 1 \land o7 = 2$
  • $o \geq 1 \land o \geq 2 \land (o = 1 \lor o = 2)$

• Component Semantics
  • $o1 = i11 + i12$

• Label Semantics
  • $l_{o1} = l_{i11} \rightarrow o_1 = i_{11}$

• Label Range
  • $l_{o1} \geq 1 \land l_{o1} \leq 9$

• Uniqueness of Output
  • $l_{o1} \neq l_{o2}$

• No Cycle
  • $l_{i11} < l_{o1}$

Why use connection points? What if we remove connection points and output label $l_{ox}$, and use $l_{ixx}$ to represent the index of the output?
This Lecture

Classic Synthesis

- Problem Definition
- Enumerative
- Presentation-based
- Constraint-based

Program Estimation

- Problem Definition
- Estimating Probabilities
- Locating the most-likely one
Program Estimation

• Input:
  • program space G
  • specification S
  • context C
  • a training set T of context-program pairs

• Output:
  • program P
  • such that \( P \in G \land P \rightarrow S \land \Pr(P \mid C) \)
  • where \( \Pr \) represents the probability learned from \( T \)
Program Estimation as an Search Problem

Q3: How to estimate the probability $\Pr(P \mid C)$?

Q4: How to find program $P$ such that $\Pr(\text{prog} \mid \text{context})$ is the largest?
Learning to synthesis (L2S)

• A general framework to address program estimation

• Combining four tools
  • **Rewriting rules**: defining a search problem
  • **Constraint solving**: pruning off invalid choices in each step
  • **Machine-learned models**: estimating the probabilities of choices in each step
  • **Search algorithms**: solving the search problem
Example: Condition Completion

• Given a program without a conditional expression, completing the condition

```java
public static long fibonacci(int n) {
    if ( ??? ) return n;
    else return fibonacci(n-1) + fibonacci(n-2);
}
```

• Useful in program repair
  • Many bugs are caused by incorrect conditions
  • Existing work could localize the faulty condition
  • Can we generate a correct condition to replace the incorrect one?

Space of Conditions

E → E “>12”
| E “>0”
| E “+” E
| “hours”
| “value”
| ...
Q3: Estimating the Probability

• Idea: Using machine learning
  • To train over a set of programs and their contexts

• Problem: machine learning usually works for classification problems
  • where the number of classes are usually small

• Idea: turn the generation problem into a set of classification problem along the grammar
Decomposing Generation

• In each step, we estimate the probabilities of the rules to expand the left-most non-terminal
  • A classification problem
Probability of the program

\[ P(prog | context) = \prod_i P( rule_i | context, prog_i, position_i) \]

- context: The context of the program
- prog_i: The AST generated at the ith step
- position_i: The non-terminal to be expanded at the ith step
- rule: the chosen rule at the ith step
- prog: the complete program

```java
...;if(E > 12)
) throw new ArgException();
```

context context

prog_i

E

E

position_i
Training models

• Train a model for each non-terminal
  • to classify rules expanding this non-terminal

• Training set preparation
  • The original training set:
    • A set of programs
    • Their contexts
  • Decomposing the training set:
    • Parse the programs
    • Extract the rules chosen for each non-terminal
Feature Engineering

- Extract features from
  - *context* : The context
  - *prog\_i* : The generated partial AST
  - *position\_i* : The position of the node to be expanded

```java
...; if(E > 12) throw new ArgException();
```
Can we use a different expansion order?

• Top-down

• Bottom-up

The order may greatly affect the performance of L2S.
Annotations

• Introduce annotations to symbols
  • $E^D$ indicates $E$ can be expanded downward
  • $E^U$ indicates $E$ can be expanded upward
  • $E^{UD}$ indicates $E$ can be expanded in both directions
From Grammar to Rewriting Rules

<table>
<thead>
<tr>
<th>Grammar</th>
<th>Top-down Rules</th>
<th>Bottom-up Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E \rightarrow E &quot;+&quot; E$</td>
<td>$E^D \Rightarrow E \rightarrow E^D &quot;+&quot; E^D$</td>
<td>$E^U \Rightarrow E^U \rightarrow E &quot;+&quot; E^D$</td>
</tr>
<tr>
<td>$E \rightarrow E &quot;&gt;12&quot;$</td>
<td>$E^D \Rightarrow E \rightarrow E^D &quot;&gt;12&quot;$</td>
<td>$E^U \Rightarrow E^U \rightarrow E &quot;&gt;12&quot;$</td>
</tr>
<tr>
<td>$E \rightarrow &quot;hours&quot;$</td>
<td>$E^D \Rightarrow E \rightarrow &quot;hours&quot;$</td>
<td>&quot;hours&quot;$^U \Rightarrow E^U \rightarrow &quot;hours&quot;$</td>
</tr>
</tbody>
</table>

Creation Rules

- $\Rightarrow E^D$ // starting from the root
- $\Rightarrow E^{DU}$ // starting from a middle node
- $\Rightarrow "hours"^U$ // starting from a leaf

Ending Rule

$E^U \Rightarrow E$
Example

• Top-down

⇒ $E^D$

$E^D$ ⇒ $E \rightarrow E^D \text{”} >12\text{”}$

$E \rightarrow >12$

$E \rightarrow \text{“hours”}$

$E \rightarrow >12$

• Bottom-up

⇒ “hours”$^U$

“hours”$^U$ ⇒ $E^U \rightarrow \text{“hours”}$

$E^U \rightarrow >12$

$E^U \rightarrow E \text{“} + \text{”} E^D$

$E \rightarrow >12$

$E \rightarrow \text{hours}$
Unambiguity

• A set of rewriting rules are unambiguous if
  • there is at most one unique set of rule applications to construct any program.

• When the rule set is unambiguous, we still have
  • $P(prog | context) = \prod_i P(rule_i | context, prog_i, position_i)$
Q4: How to find the most probable program?

- Local Optimal ≠ Global Optimal

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$E_0$</td>
<td>$E \rightarrow E \ “ &gt; 12”$</td>
<td>0.3</td>
<td>$E \rightarrow E \ “ &gt; 0”$</td>
<td>0.6</td>
</tr>
<tr>
<td>$E_1$</td>
<td>$E \rightarrow “hours”$</td>
<td>0.1</td>
<td>$E \rightarrow “value”$</td>
<td>0.2</td>
</tr>
<tr>
<td>$E_2$</td>
<td>$E \rightarrow “hours”$</td>
<td>0.8</td>
<td>$E \rightarrow “value”$</td>
<td>0.1</td>
</tr>
</tbody>
</table>

$0.6 \times 0.2 = 0.12$

$0.3 \times 0.8 = 0.24$
Use Metaheuristic Search

• Beam Search:
  • Keep n most probable partial programs
  • Expand the programs to get new programs

• Genetic Search:
  • Keep n most probably complete programs
  • Mutate the programs to get new programs
Applications

• Application 1:
  • Repairing Conditional Expressions

• Application 2:
  • Generating Code from Natural Language Expression
Repairing Conditional Expressions

- Condition bugs are common

```java
lcm = Math.abs(a+b);
+ if (lcm == Integer.MIN_VALUE)
+   throw new ArithmeticException();

- if (hours >= 24)
+ if (hours > 24)
   withinOneDay=true;
```

- Missing boundary checks
- Conditions too weak or too strong

**Steps:**

1. Localize a buggy if condition with SBFL and predicate switching
2. Synthesize an if condition to replace the buggy one
3. Validate the new program with tests
L2S Configuration

• Rewriting rules
  • Bottom-up
  • Estimate the leftmost variable first

• Machine learning
  • Xgboost
  • Manually designed features

• Constraints
  • Type constraints & size constraints

• Search algorithm
  • Beam search
Results

Benchmark: Defects4J

Number of Repaired Bugs

Precision

Also repaired 8 unique bugs that have never been repaired by any approach.
Generating Code from Natural Language Expression

• Can we generate code automatically to avoid repetitive coding?

• Existing approaches use RNN to translate natural language descriptions to programs
  • Long dependency problem: work poorly on long programs
L2S Configuration

• Rewriting rules
  • Top-down

• Machine learning
  • A CNN-based network

• Constraints
  • Size constraints

• Search algorithm
  • Beam search
A CNN-based Network Architecture
## Results

### Benchmark: HearthStone

<table>
<thead>
<tr>
<th>Model</th>
<th>StrAcc</th>
<th>Acc+</th>
<th>BLEU</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPN (Ling et al. 2016)</td>
<td>6.1</td>
<td>–</td>
<td>67.1</td>
</tr>
<tr>
<td>SEQ2TREE (Dong and Lapata 2016)</td>
<td>1.5</td>
<td>–</td>
<td>53.4</td>
</tr>
<tr>
<td>SNM (Yin and Neubig 2017)</td>
<td>16.2</td>
<td>~18.2</td>
<td>75.8</td>
</tr>
<tr>
<td>ASN (Rabinovich, Stern, and Klein 2017)</td>
<td>18.2</td>
<td>–</td>
<td>77.6</td>
</tr>
<tr>
<td>ASN+SUPATT (Rabinovich, Stern, and Klein 2017)</td>
<td>22.7</td>
<td>–</td>
<td>79.2</td>
</tr>
<tr>
<td><strong>Our system</strong></td>
<td><strong>27.3</strong></td>
<td><strong>30.3</strong></td>
<td><strong>79.6</strong></td>
</tr>
</tbody>
</table>
Newest Results

• Replacing CNN with Transformer
  • Transformer: a new neural architecture at 2017
  • The flexibility of L2S allows to easily utilize new models

<table>
<thead>
<tr>
<th>Model</th>
<th>StrAcc</th>
<th>Acc+</th>
<th>BLEU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plain</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LPN (Ling et al., 2016)</td>
<td>6.1</td>
<td>–</td>
<td>67.1</td>
</tr>
<tr>
<td>SEQ2TREE (Dong and Lapata, 2016)</td>
<td>1.5</td>
<td>–</td>
<td>53.4</td>
</tr>
<tr>
<td>YN17 (Yin and Neubig, 2017)</td>
<td>16.2</td>
<td>~18.2</td>
<td>75.8</td>
</tr>
<tr>
<td>ASN (Rabinovich et al., 2017)</td>
<td>18.2</td>
<td>–</td>
<td>77.6</td>
</tr>
<tr>
<td>ReCode (Hayati et al., 2018)</td>
<td>19.6</td>
<td>–</td>
<td>78.4</td>
</tr>
<tr>
<td>CodeTrans-A</td>
<td>25.8</td>
<td>25.8</td>
<td>79.3</td>
</tr>
<tr>
<td>Structured</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ASN+SUPATT (Rabinovich et al., 2017)</td>
<td>22.7</td>
<td>–</td>
<td>79.2</td>
</tr>
<tr>
<td>SZM19 (Sun et al., 2019)</td>
<td>27.3</td>
<td>30.3</td>
<td>79.6</td>
</tr>
<tr>
<td>CodeTrans-B</td>
<td>31.8</td>
<td>33.3</td>
<td>80.8</td>
</tr>
</tbody>
</table>
Future Learning

• Surveys:

• Tools:
  • sygus.org – the SyGuS competition, a good place to look at
  • Some tools we recently used
    • EUSolver
    • CVC4
    • Second-Order Solver

• Course:
  • Program Synthesis by Nadia Polikarpova@UCSD
  • https://github.com/nadia-polikarpova/cse291-program-synthesis/
Reference

• Enumerative

• FlashMeta
  • Oleksandr Polozov, Sumit Gulwani: FlashMeta: a framework for inductive program synthesis. OOPSLA 2015: 107-126

• Componen-Based Program Synthesis
  • Susmit Jha, Sumit Gulwani, Sanjit A. Seshia, Ashish Tiwari: Oracle-guided component-based program synthesis. ICSE (1) 2010: 215-224

• L2S
  • Yingfei Xiong, Bo Wang, et al.: Learning to Synthesize. GI'18: Genetic Improvement Workshop, May 2018