Future of Software Development?

Program Synthesis and its Application

Paul Zhu

School of Software,
Tsinghua University

May 27, 2019

History & Trends in Software Development

In the past:

- Write programs that can work
- Write programs that runs fast

Issues:

- Write it once, debug it everywhere
- No silver bullet
- Many programmers don't welcome changes

Nowadays and in future:

- Write programs that are correct
- Write programs that generate programs

Program Synthesis!

> *Program Synthesis is the task of automatically finding programs from the underlying programming language that satisfy user intent expressed in some form of constraints. (Gulwani, Polozov, and Singh 2017)*

Features:

- automated programming
- correctness-by-construction

v.s.

- compilers
- code generator

Today

1. Program Synthesis in Action
2. Programming by Examples
   - Case Study: FlashFill
3. Sketch-based Synthesis
   - Case Study: Rosette

Contents

FlashFill

Automated String Processing in Spreadsheets by Examples (Gulwani 2011)

Name	Title	Email Alias	First Name	Last Name	Full Email Address
Johanna Lorenz	Senior Engineer	JohannL	Johanna	Lorenz	johannl@contoso.com
Irvin Sayers	Project Manager, Co-Project Lead	IrvinS	Irvin		
Lidia Holloway	Product Manager, Co-Project Lead	LidiaH	Lidia		
Henrietta Mueller	Developer	HenriettaM	Henrietta		
Ben Walters	Tester	BenW	Ben		
Megan Bowen	Marketing Manager	MeganB	Megan		
Enrico Cattaneo	Attorney	EnricoC	Enrico		
Pradeep Gupta	Accountant	PradeepG	Pradeep		
Adele Vance	Developer	AdeleV	Adele		
Alex Wilber	Tester	AlexW	Alex		
Allen Deyoung	Developer	AllenD	Allen		
Brian Johnson	Engineer	BrianJ	Brian		
Emily Braun	Engineer	EmilyB	Emily		
Lee Gu	Developer	LeeG	Lee		
Patti Fernandez	Tester	PattiF	Patti		
Miriam Graham	Developer	MiriamG	Miriam		

FlashExtract

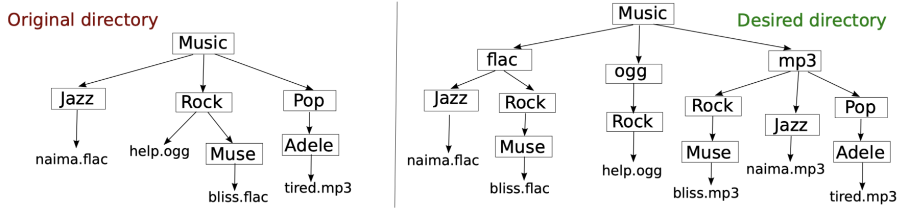
Data Extraction by Examples (Le and Gulwani 2014)

Show: 20 **1-20** Next >

Title / Author	Cited by	Year
Finding bugs with a constraint solver D Jackson, M Vaziri ACM SIGSOFT Software Engineering Notes 25 (5), 14-25	197	2000
Associating synchronization constraints with data in an object-oriented language M Vaziri, F Tip, J Dolby ACM SIGPLAN Notices 41 (1), 334-345	176	2006
Some Shortcomings of OCL, the Object Constraint Language of UML. M Vaziri, D Jackson TOOLS (34), 555-562	81	2000
Model checking software systems: A case study JM Wing, M Vaziri-Farahani ACM SIGSOFT Software Engineering Notes 20 (4), 128-139	68	1995

Hades

Transformations on Hierarchically Structured Data (Yaghmazadeh et al. 2016)



Sketch-N-Sketch

Programmatic and Direct Manipulation Together (Chugh et al. 2016)

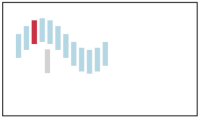
(A) Excerpt from `prelude.little`

```
(defrec range (λ(i j)
  (if (> i j) nil
      (cons i (range (+ 1^ℓ1 i) j)))))

(def zeroTo (λn (range 0^ℓ0 (- n 1))))
```

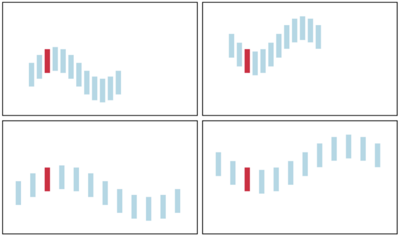
(B) `sineWaveOfBoxes.little`

```
(def [x0 y0 w h sep amp] [50 120 20 90 30 60])
(def n 12!{3-30})
(def boxi (λi
  (let xi (+ x0 (* i sep))
  (let yi (- y0 (* amp (sin (* i (/ twoPi n)))))
    (rect 'lightblue' xi yi w h)))))

(svg (map boxi (zeroTo n)))
```

(C) Suppose the user clicks on the third box from the left (colored darker in red for emphasis) and drags it to a new position down and to the right (colored lighter in gray):



(D) Sketch-n-Sketch synthesizes four candidate updates to the program, which have the following effects:



Natural Language Programming

Program synthesis using Natural Language (Desai et al. 2016)

An end user says: "I would like the time of your earliest flight in the morning from Philadelphia to Washington on American Airlines."

Then it is translated to a domain-specific language program:

```
ColSelect(DEP_TIME, RowMin(DEP_TIME,
    RowPred(EqDepart(PHILADELPHIA, Time(MORNING)),
            EqArrive(WASHINGTON, Time(ANY)),
            EqAirline(AMERICAN))))
```

Smart Education

Automated Feedback Generation for Introductory Programming Assignments (Singh, Gulwani, and Solar-Lezama 2013)

```python
def computeDeriv(poly):
    length = int(len(poly)-1)
    i = length
    deriv = range(1,length)
    if len(poly) == 1:
        deriv = [0]
    else:
        while i >= 0:
            new = poly[i] * i
            i -= 1
            deriv[i] = new
    return deriv
```

The program requires **2** changes:

- In the expression **range(1, length)** in **line 4**, increment **length** by **1**.
- In the comparison expression **(i >= 0)** in **line 8**, change operator **>=** to **!=**.

Scythe

Synthesizing Highly Expressive SQL Queries by Examples (Wang, Cheung, and Bodik 2017)

T_1

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

T_2

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

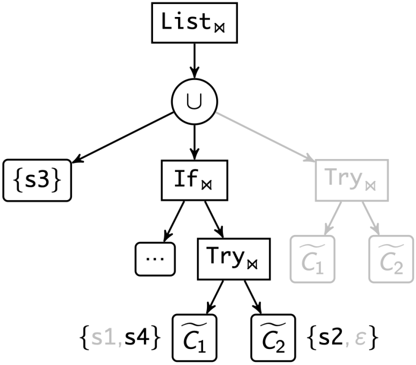
T_{out}

c_0	c_1	c_2	c_3	c_4
1	12/25	1	1	30
4	12/24	2	2	10

```
Select *
From   (Select *
        From   T1
        Where  T1.date = 12/24
               Or T1.date = 12/25) T3
Join   (Select   oid, Max(val)
        From     (Select *
                  From   T2
                  Where  T2.val < 50) T4
        Group By oid) T5
On     T3.uid = T5.oid
```

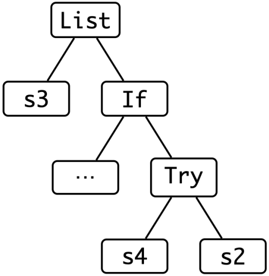
AutoMerge

Conflict Resolution for Structured Merge (Zhu and He 2018)



constructed VSA

includes



expected

S3

Syntax- and Semantic-guided Repair Synthesis (Le et al. 2017)

```
1  if (sourceExcerpt != null) {
2    ...
3    -if (excerpt.equals(LINE) && 0 <= charno
4    -    && charno < sourceExcerpt.length()) {
5    +if (excerpt.equals(LINE) && 0 <= charno
6    +    && charno <= sourceExcerpt.length()) {
7    ...
8  }
```

Figure 1: A bug in Closure compiler, revision *1e070472*. The bug is at lines 3–4. The developer fix is shown on lines 5–6; it turns a `<` to a `<=` in the second line of the if condition.

Test	Input			Desired Output
	charno	*(M1)* `excerpt.equals(LINE)`	*(M2)* `sourceExcerpt.length()`	
A	7	`true`	7	`true`
B	10	`true`	10	`true`

Figure 2: Input-output examples for both variables and conditions, extracted for the Closure compiler bug described in Figure 1. We use M1 and M2 to refer to the conditions in columns 3–4 in subsequent exposition. The last column represents the desired output of the overall branch decision.

Compilation Error Repair

Parsing Tree Transformation Synthesis (under review)

```
public class MethodeReturn : MonoBehaviour
{
    static void LongHypo(float a, float b)
    {
        float SommeCar = a * a + b * b;
        return SommeCar;
        ^
    }

    void Start()
    {
        float result = LongHypo(3, 4);
        Debug.Log(result);
    }
}
```

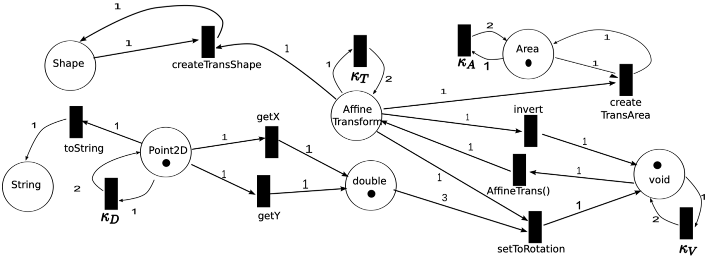
Synthesized repair: return type `void` is changed to `float` for `LongHypo`.

SyPet

Component-based Synthesis for Complex APIs (Feng et al. 2017)

```
public void test1() {
    Area a1 = new Area(new Rectangle(0, 0, 10, 2));
    Area a2 = new Area(new Rectangle(-2, 0, 2, 10));
    Point2D p = new Point2D.Double(0, 0);
    assertTrue(a2.equals(rotate(a1, p, Math.PI/2)));
}
```

```
Area rotate(Area obj, Point2D pt, double angle) {
    AffineTransform at = new AffineTransform();
    double x = pt.getX();
    double y = pt.getY();
    at.setToRotation(angle, x, y);
    Area obj2 = obj.createTransformedArea(at);
    return obj2;
}
```



Interactive Parser Synthesis by Example

(Leung, Sarracino, and Lerner 2015)

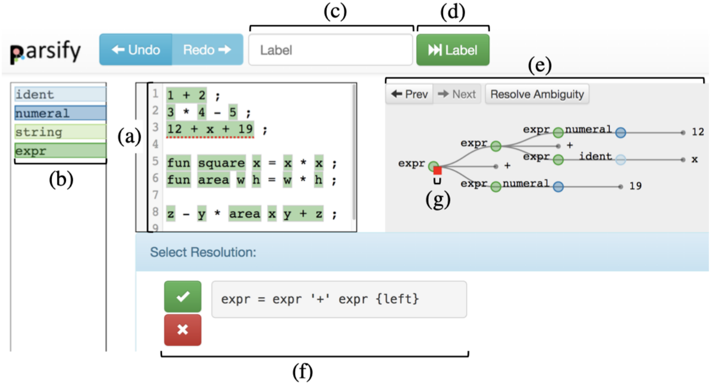
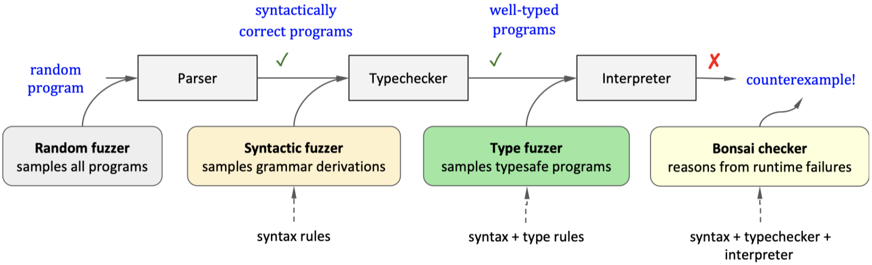


Figure 1. The Parsify user interface: (a) File View, (b) Legend, (c) Label Box, (d) Label Button, (e) Parse Tree Pane, (f) Resolution Pane, and (g) Negative Label

Bonsai

Synthesis-based Reasoning for Type Systems (Chandra and Bodik 2017)



Program Systhesis Helps!

Many people can benefit from it:

- end users
- developers
- educators & learners
- researchers

A lot of fields are studying it:

- programming languages
- software development
- program repair
- program analysis & verification
- machine learning

Contents

An Excel Task

Example (Month Extraction)

Input v_1	Output
01/21/2001	01
22.02.2002	02
2003-23-03	03
06/01/2019	?
⋮	⋮

Bob specifies what he wants to do by filling in the output cell for the first three rows. Now he wants Excel to finish the remaining rows.

Regular Expressions?

Bob asks Alice, and Alice says: “You can use regular expressions!”

Now, Bob has two problems...

Programming by Examples

Features:

- input-output examples as specification
- enable non-programmers to create programs for automating repetitive tasks

Challenges:

- description of program space
- representation of program space
- disambiguation

Contents

Domain-specific Language (DSL) of FlashFill

$$
\begin{array}{rcl}
\text{Program } P & ::= & \texttt{Switch}((b_1, e_1), \ldots, (b_n, e_n)) \\
\text{Bool } b & ::= & d_1 \vee d_2 \vee \cdots \vee d_n \\
\text{Conjunction } d & ::= & \pi_1 \wedge \pi_2 \wedge \cdots \wedge \pi_n \\
\text{Predicate } \pi & ::= & \texttt{Match}(v_i, r, k) \mid \neg\texttt{Match}(v_i, r, k) \\
\text{Trace expression } e & ::= & \texttt{Concat}(f, e) \mid f \\
\text{Atomic expression } f & ::= & \texttt{SubStr}(v_i, p_1, p_2) \mid \texttt{Const}(s) \mid \texttt{Loop}(\lambda w.e) \\
\text{Position } p & ::= & \texttt{CPos}(k) \mid \texttt{Pos}(r_1, r_2, c) \\
\text{Regular expression } r & ::= & t_1 t_2 \cdots t_n \\
\text{Token } t & ::= & C\texttt{+} \mid [\hat{}\,C]\texttt{+} \mid \texttt{Start} \mid \texttt{End} \mid \texttt{.} \mid \texttt{-} \mid \texttt{*} \mid \texttt{/} \mid \cdots \\
\text{Character set } C & ::= & \texttt{[0-9]} \mid \texttt{[A-Z]} \mid \texttt{[a-z]} \mid \texttt{[A-Za-z]} \mid \cdots \\
\text{Integer expression } c & ::= & k \mid k_1 w + k_2
\end{array}
$$

where k denotes an integer, s denotes a string, w binds to an integer, and v_i refers to the i-th column of input.

DSL is Dead?

Shriram Krishnamurthi once[1] said: "These are the only choices in life for a configuration language: it grows up to a programming language, or it dies."

- Gradle build scripts are written using a variant of Groovy/Kotlin.
- Sbt can execute a Scala script containing dependency declarations or other settings.

[1] `https://www.youtube.com/watch?v=3N__tvmZrzc`

Expressive? Yes

Example (Month Extraction)

Input v_1	Output
01/21/2001	01
22.02.2002	02
2003-23-03	03

Program: $\texttt{Switch}((b_1, e_1), (b_2, e_2), (b_3, e_3))$, where

- $b_1 = \texttt{Match}(v_1, \texttt{/})$
- $b_2 = \texttt{Match}(v_1, \texttt{.})$
- $b_3 = \texttt{Match}(v_1, \texttt{-})$
- $e_1 = \texttt{SubStr}(v_1, \texttt{Pos}(\texttt{StartToken}, \varepsilon, 1), \texttt{Pos}(\varepsilon, \texttt{/}, 1))$
- $e_2 = \texttt{SubStr}(v_1, \texttt{Pos}(\texttt{.}, \varepsilon, 1), \texttt{Pos}(\varepsilon, \texttt{.}, 2))$
- $e_3 = \texttt{SubStr}(v_1, \texttt{Pos}(\texttt{-}, \varepsilon, 2), \texttt{Pos}(\texttt{EndToken}, \varepsilon, 1))$

… and No

Example (To Lower Case)

Input v_1	Output
HELLO	hello
World	world

Learning Traces: Motivation

$$\boxed{e ::= \texttt{Concat}(f, e) \mid f}$$

Example

All trace expressions yielding “abc” could be any one of the following:

- all atomic expressions yielding “abc”
- concatenations of all atomic expressions yielding “ab” and all trace expressions yielding “c”
- concatenations of all atomic expressions yielding “a” and all trace expressions yielding “bc”

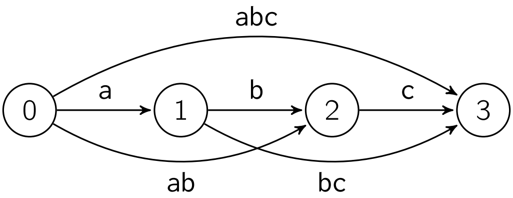
Learning Traces: Motivation

$$\boxed{e ::= \texttt{Concat}(f, e) \mid f}$$

Example

All trace expressions yielding "abc" could be any one of the following:

- all atomic expressions yielding "abc"
- all concatenations of a possible split of "abc"



Version Space Algebra (VSA)

- First defined by Mitchell 1982, and then expanded upon *program synthesis* by Gulwani 2011; Polozov and Gulwani 2015.
- Succint representation by its memory-sharing mechanism.

$$
\begin{array}{lcll}
\text{VSA } \widetilde{N} & ::= & \{P_1, P_2, \ldots, P_k\} & \text{(explicit)} \\
 & | & \widetilde{N_1} \cup \widetilde{N_2} \cup \cdots \cup \widetilde{N_k} & \text{(union)} \\
 & | & F_{\bowtie}(\widetilde{N_1}, \widetilde{N_2}, \ldots, \widetilde{N_k}) & \text{(join)}
\end{array}
$$

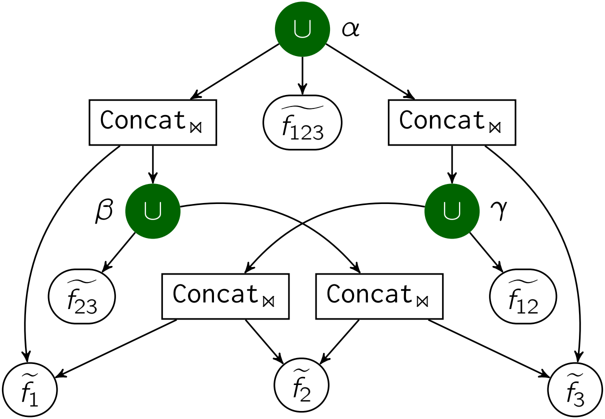
where each VSA node represents a set of concrete programs:

$$
\begin{aligned}
[\![\{P_1, \ldots, P_k\}]\!] &= \{P_1, \ldots, P_k\} \\
[\![\widetilde{N_1} \cup \cdots \cup \widetilde{N_k}]\!] &= [\![\widetilde{N_1}]\!] \cup \cdots \cup [\![\widetilde{N_k}]\!] \\
[\![F_{\bowtie}(\widetilde{N_1}, \ldots, \widetilde{N_k})]\!] &= \{F(P_1, \ldots, P_k) \mid P_1 \in [\![\widetilde{N_1}]\!], \ldots, P_k \in [\![\widetilde{N_k}]\!]\}
\end{aligned}
$$

Learning Traces: VSA Representation

$$\boxed{e ::= \texttt{Concat}(f, e) \mid f}$$

Let $\widetilde{f_{123}}$, $\widetilde{f_{12}}$, etc. be the VSA for atomic expressions yielding resp. "abc", "ab", etc. Then the VSA for trace expressions yielding "abc" is:



Deductive Synthesis

$$\boxed{\widetilde{P} \models \phi}$$

Programs are expressed by	Rule
a DSL terminal	$\dfrac{P_1 \models \phi, \ldots, P_k \models \phi}{\widetilde{\{}P_1, \ldots, P_k\widetilde{\}} \models \phi}$
a DSL nonterminal with k choices	$\dfrac{\widetilde{P_1} \models \phi, \ldots, \widetilde{P_k} \models \phi}{\widetilde{P_1} \cup \cdots \cup \widetilde{P_k} \models \phi}$
a k-ary DSL operator F	$\dfrac{\widetilde{P_1} \models \phi_1, \ldots, \widetilde{P_k} \models \phi_k}{F_{\bowtie}(\widetilde{P_1}, \ldots, \widetilde{P_k}) \models \phi}$

In the last case, we need a witness function for F to compute $\phi_1, \ldots, \phi_k$ from ϕ.

Witness Functions give Reverse Semantics

- Semantics: given $[\![f]\!]_\sigma = s_1$ and $[\![e]\!]_\sigma = s_2$, compute the output for $[\![\texttt{Concat}(f, e)]\!]_\sigma$, i.e. s_1 concatenates with s_2.
- Witness function: given $\texttt{Concat}_{\bowtie}(\widetilde{f}, \widetilde{e}) \models \phi$, find ϕ_1 and ϕ_2 such that $\widetilde{f} \models \phi_1$ and $\widetilde{e} \models \phi_2$.
- A specification ϕ is a set of pairs of an input state with an output string.
- Reverse semantics: given $[\![\texttt{Concat}(f, e)]\!]_\sigma = s$, find all possible $[\![f]\!]_\sigma$ and $[\![e]\!]_\sigma$, i.e. all concatenations which yields s.

Ranking

Based upon Occam's razor:

- A `Concat` constructor is simpler than another one if it contains smaller number of arguments or its arguments are pairwise simpler.
- `SubStr`-expressions are simpler than `Const`-expressions.
- `Pos`-expressions are simpler than `CPos`-expressions.
- `Start` and `End` are simpler than all other tokens.
- etc.

Alternatively, scoring functions (e.g. in PROSE) and even machine learning techniques (e.g. Ellis and Gulwani 2017) come to rescue here.

Artifacts

Microsoft PROgram Synthesis using Examples SDK

- Tutorial: `https://microsoft.github.io/prose`
- Samples: `https://github.com/microsoft/prose`
- Platform: .NET Framework, .NET Core
- Warning: the official document may be inconsistent with the implementation, and some SDK APIs may behave unexpectedly

My implementation: `https://github.com/paulzfm/StringProcessing`

Contents

Logical Programming

In Prolog[2], one may specify a knowledge base:

```
likes(sam,Food) :- indian(Food), mild(Food).
likes(sam,Food) :- chinese(Food).
likes(sam,chips).

indian(curry).
indian(tandoori).
mild(tandoori).
chinese(chow_mein).
chinese(jiao_zi).
```

and execute a query

```
?- likes(sam, X).
```

whose solutions are

```
X = tandoori ; X = chow_mein ; X = jiao_zi ; X = chips.
```

[2] http://www.swi-prolog.org

Make Logical Programming Great Again?

> *Next-paradigm programming languages could be based on any of several potential technologies – e.g., perhaps on machine learning and statistical techniques, or on SMT solvers and symbolic reasoning. Regardless of the technology, however ... "principles":*
>
> - *Productivity and Performance Tied Together*
> - *Need For Firm Mental Grounding*
> - *Workflows Will Change*
>
> – *Yannis Smaragdakis* [3]

Also, he invented a "theorem":

$$Programs \neq Algorithms + Data\ Structures$$

$$Compiler = Algorithms + Data\ Structures$$

[3] `https://yanniss.github.io/next-paradigm-apr30-2019.pdf`

We've Seen this

Given a DSL expressing bitwise operations:

$$
\begin{aligned}
\text{program } P &::= \mathsf{plus}(E_1, E_2) \mid \mathsf{mul}(E_1, E_2) \mid \mathsf{shl}(E, C) \mid \mathsf{shr}(E, C) \\
\text{expression } E &::= x \mid C \\
\text{8-bit constant } C &::= (00000000)_2 \mid (00000001)_2 \mid \cdots \mid (11111111)_2
\end{aligned}
$$

where the input variable x is bound to a 8-bit unsigned integer.

We hope to synthesize a program P satisfying the specification

$$\phi = \forall x. P(x) \geqslant 0.$$

In other words, find an instantiation for the sketch $P = ?$ wrt ϕ.

Contents

Demonstration

Rosette[4] is a solver-aided programming language based on Racket, which combines verification and sketch-based synthesis.

The example comes from Torlak and Bodik 2014.

[4] `https://docs.racket-lang.org/rosette-guide/index.html`

Will Program Synthesis Change the World?

Bad news:

- Scalability is a big issue
- Yet not very practical for industry

Good news:

- Many people are already benefit from PBE
- Solvers are becoming more and more efficient and powerful
- Program synthesis is an active and interdisciplinary field

References I

- Kartik Chandra and Rastislav Bodik. "Bonsai: Synthesis-based Reasoning for Type Systems". In: *Proc. ACM Program. Lang.* 2.POPL (Dec. 2017), 62:1–62:34.
- Ravi Chugh et al. "Programmatic and Direct Manipulation, Together at Last". In: *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI '16. Santa Barbara, CA, USA: ACM, 2016, pp. 341–354.
- Aditya Desai et al. "Program synthesis using natural language". In: *Proceedings of the 38th International Conference on Software Engineering*. ACM. 2016, pp. 345–356.
- Kevin Ellis and Sumit Gulwani. "Learning to Learn Programs from Examples: Going Beyond Program Structure". In: 2017.
- Yu Feng et al. "Component-based Synthesis for Complex APIs". In: *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*. POPL 2017. Paris, France: ACM, 2017, pp. 599–612.

References II

- Sumit Gulwani. "Automating String Processing in Spreadsheets Using Input-output Examples". In: *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL '11. Austin, Texas, USA: ACM, 2011, pp. 317–330.
- Sumit Gulwani, Alex Polozov, and Rishabh Singh. *Program Synthesis*. Vol. 4. NOW, 2017, pp. 1–119.
- Vu Le and Sumit Gulwani. "FlashExtract: A Framework for Data Extraction by Examples". In: *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI '14. Edinburgh, United Kingdom: ACM, 2014, pp. 542–553.
- Xuan-Bach D. Le et al. "S3: Syntax- and Semantic-guided Repair Synthesis via Programming by Examples". In: *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. ESEC/FSE 2017. Paderborn, Germany: ACM, 2017, pp. 593–604.

References III

Alan Leung, John Sarracino, and Sorin Lerner. "Interactive Parser Synthesis by Example". In: *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI '15. Portland, OR, USA: ACM, 2015, pp. 565–574.

Tom M. Mitchell. "Generalization as search". In: *Artificial Intelligence* 18.2 (1982), pp. 203 –226.

Oleksandr Polozov and Sumit Gulwani. "FlashMeta: A framework for inductive program synthesis". In: *ACM SIGPLAN Notices* 50.10 (2015), pp. 107–126.

Rishabh Singh, Sumit Gulwani, and Armando Solar-Lezama. "Automated Feedback Generation for Introductory Programming Assignments". In: *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI '13. Seattle, Washington, USA: ACM, 2013, pp. 15–26.

Emina Torlak and Rastislav Bodik. "A Lightweight Symbolic Virtual Machine for Solver-aided Host Languages". In: *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI '14. Edinburgh, United Kingdom: ACM, 2014, pp. 530–541.

References IV

Chenglong Wang, Alvin Cheung, and Rastislav Bodik. "Synthesizing Highly Expressive SQL Queries from Input-output Examples". In: *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI 2017. Barcelona, Spain: ACM, 2017, pp. 452–466.

Navid Yaghmazadeh et al. "Synthesizing Transformations on Hierarchically Structured Data". In: *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI '16. Santa Barbara, CA, USA: ACM, 2016, pp. 508–521.

Fengmin Zhu and Fei He. "Conflict Resolution for Structured Merge via Version Space Algebra". In: *Proc. ACM Program. Lang.* 2.OOPSLA (Oct. 2018), 166:1–166:25.