Introduction to Program Synthesis

Wei Li

Department of Computer Science
Shanghai Jiao Tong University

Oct. 10th, 2012

References

Dimensions in Program Synthesis
Invited Talk on PPDP 10' Sumit Gulwani

Synthesizing Programs with Constraint Solvers
Invited Talk on CAV 12' Ras Bodik and Emina Torlak

Tutorial on Sketch Programming
Invited Talk on PLDI 12' Armando Solar-Lezama

Outline

1. Why program synthesis
2. What is program synthesis
3. How program synthesis works
4. Conclusions and future work

Why program synthesis

Why program synthesis

Programmers

- General purpose programming assistance
  - **Automatically** discover tricky/menial details
  - Automated debugging
    (Identified **suspected region** *and a description of* **desired behaviors***)*
- Discovery of new algorithms

Why program synthesis

End-users

- Develop **utility programs** with no programming background
- Teaching
  - Automated problem solving

What is program synthesis

What is program synthesis

Synthesize an **executable program** from user intent expressed in form of some **constraints**

What is program synthesis

Find a program P that meets a spec ϕ(input,output):

$$\exists P \, \forall x \, . \, \phi(x, P(x))$$

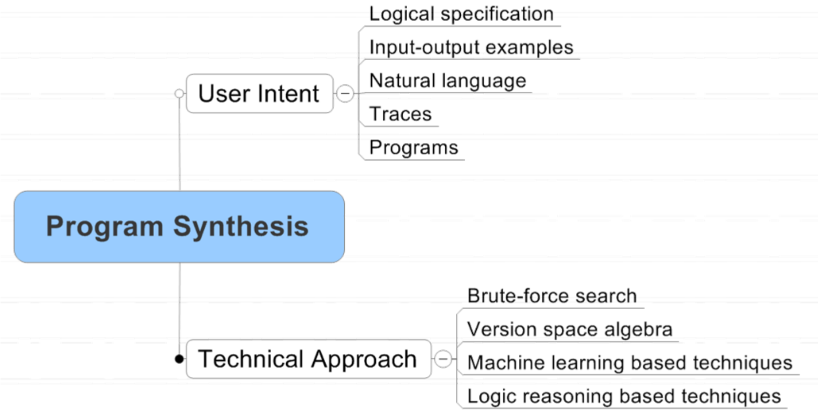
When to use synthesis:

productivity: when writing ϕ is faster than writing P

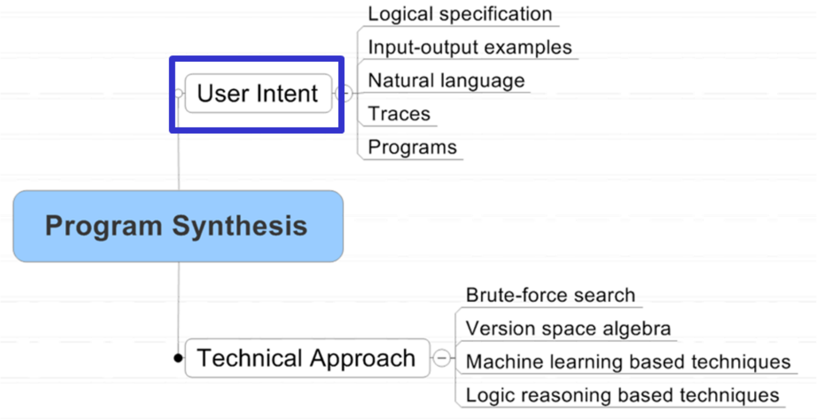
correctness: when proving ϕ is easier than proving P

How program synthesis works

How program synthesis works



User intent


Program Synthesis
User Intent
Logical specification
Input-output examples
Natural language
Traces
Programs
Technical Approach
Brute-force search
Version space algebra
Machine learning based techniques
Logic reasoning based techniques

Logical specification

Logical **relation** between *inputs and outputs* of a program

Example: sorting algorithm

$$\begin{aligned} & \forall k.\ (0 \leq k < n-1) \implies (B[k] \leq B[k+1]) && (1) \\ \wedge \quad & \forall k\ \exists j.\ (0 \leq k < n) \implies (0 \leq j < n \wedge B[j] = A[k]) && (2) \end{aligned}$$

Logical specification

- Require **additional knowledge** of logic
- Harder to get it **right**
- Not be preferred by **end-users**

Natural language

Map natural language sentences into **logical representations**

Input-output examples

User driven interaction

- User **inspects**
- User add new input-output example

Synthesizer driven interaction

- Synthesizer finds distinguishing input

Input-output examples

Example: **Bitvector program**

- Masks off the rightmost contiguous sequence of 1s in the input bitvector
- Synthesizer driven interaction

User	Oracle		
Input → Output	**Program 1**	**Program 2**	**Distinguishing Input ?**
01011 → 01000	$(x+1)\,\&\,(x-1)$	$(x+1)\,\&\,x$	00000 ?
00000 → 00000	$-(\neg x)\,\&\,x$	$(((x\& -x)\mid -(x-1))\&x)\oplus x$	00101 ?
00101 → 00100	$(x+1)\&x$	…	01111 ?
01111 → 00000	…	…	00110 ?
00110 → 00000	…	…	01100 ?
01100 → 00000	…	…	01010 ?
01010 → 01000	$(((x-1)\mid x)+1)\&x$	None	Program is $(((x-1)\mid x)+1)\&x$

Traces

A **trace** is a detailed **step-by-step** description of how the program should behave on a **given input**

Example: **Compute factorial(n)**

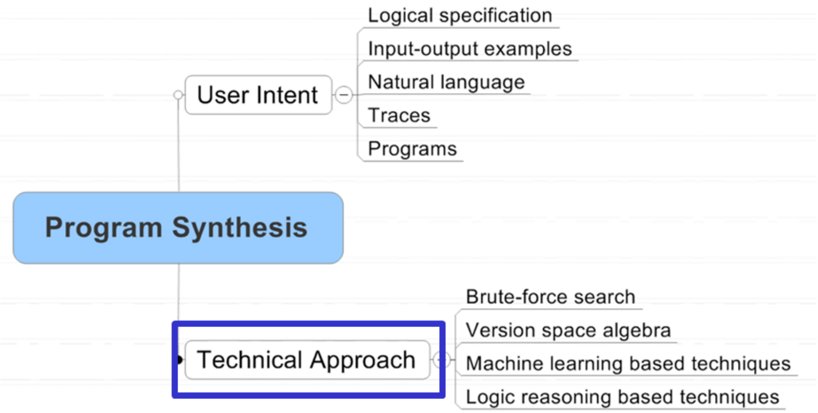
- A demonstration for input 7
- The string $7 \times 6 \times 5 \times 4 \times 3 \times 2$ or the recursive trace $7 \times \mathit{factorial}(6)$
- Easier to describe than providing the **final simplified output 5040**

Programs

Partial programs

- Sketch – interger holes
- Some tricky or mundane details in the programs

Technical approach



Existing programs

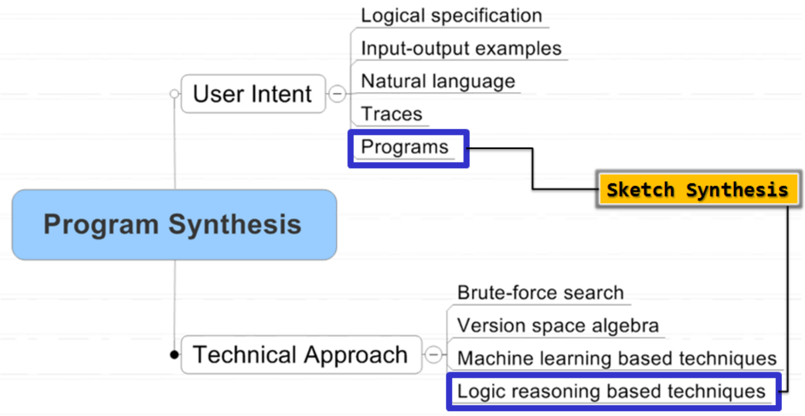
Sketch Synthesis

- *PLDI 07'*
- *Armando Solar-Lezama* - MIT

String transformation

- *POPL 12'*
- *Sumit Gulwani* - MSR

Sketch synthesis



Sketch synthesis

- **Specification** of the desired **functionality**, usually to be *unoptimized and inefficient* but *easy* to implement
- Partial implementation – **sketch** of an optimized program
- Synthesizer completes the sketch to behave like the **specification**

Sketch synthesis

Demo

Logic reasoning technique

- Generalized **boolean satisfiability** problem
- Replace with *Control variables*
- Translate to **boolean functions**

Logic reasoning technique

1. 2QBF function

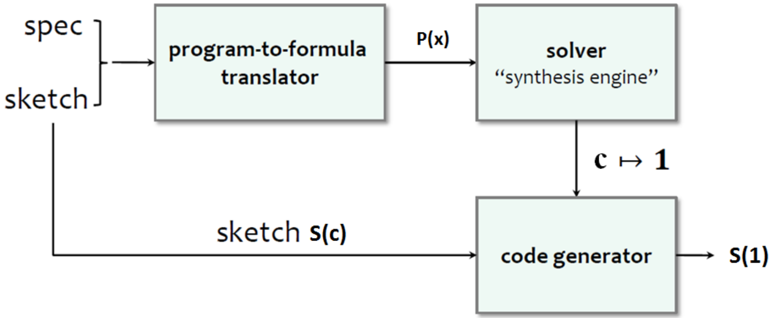
$$\exists c.\forall x.P(x) = S(x, c)$$

2. Counterexample-driven search procedure

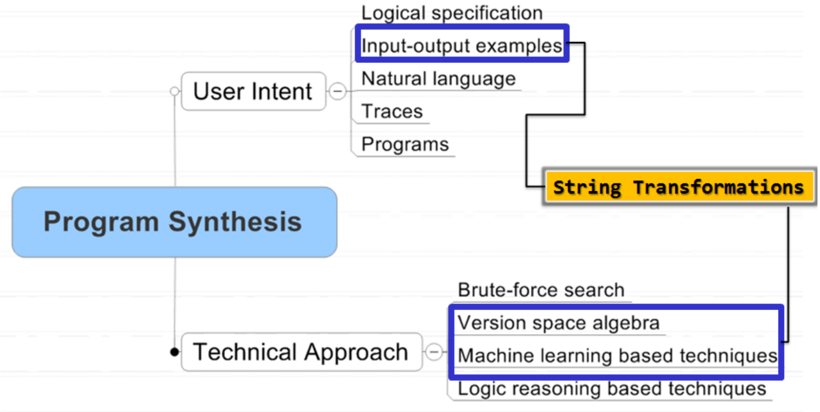
$$\exists c.\forall x \in E.P(x) = S(x, c)$$

3. SAT solver

Logic reasoning technique



String transformation



String transformation

Synthesizing a wide range of **string processing programs** in spreadsheets from **input-output examples**

String transformation

Example: **Extracting directory name**

Input v_1	*Output*
Company\Code\index.html	*Company\Code*
Company\Docs\Spec\specs.doc	*Company\Docs\Spec*

Example: **Generate abbreviation**

Input v_1	*Output*
International Business Machines	*IBM*
Principles Of Programming Languages	*POPL*
International Conference on Software Engineering	*ICSE*

Language of string programs

More expressive language to reduce search complexity

- More expressive language
  (*eg: concatenate, switch, substring, etc*)
- Tradeoff between the **expressiveness** of a language and the **complexity** of the search technique

Language of string programs

Example: **Extracting directory name**

Input v_1	*Output*
Company\Code\index.html	*Company\Code*
Company\Docs\Spec\specs.doc	*Company\Docs\Spec*

String Program:

$\mathtt{SubStr}(v_1, \mathtt{CPos}(0), \mathtt{Pos}(SlashTok, \epsilon, -1))$

Ranking strategies

- **Shorter** programs
  eg: fewer conditions, shorter string expressions
- Less number of **constants**

Limitations and follow-up work

This program: syntactic manipulations of strings

- **Semantic** manipulations
- Manipulations of **tables**

Future work

Conclusions

- Revolution in computing
- How to what

Future work

- Combine various forms of user intent in a **unified programming interface**
- Combine the power of various **search techniques**

Q & A

Thank you!