EECS 598. Program Synthesis: Techniques and Applications

Lecture 1: Introduction

Xinyu Wang
Logistics: Course staff

• Instructor: Xinyu Wang
  • Assistant Professor in CSE
  • Research: Programming Languages, Formal Methods, Software Engineering
  • Email: xwangsd@umich.edu
  • Homepage: https://web.eecs.umich.edu/~xwangsd/courses/f20/eecs598.html

• No TA
• Canvas
Introduce yourself

- Name?
- What program? What year?
- What areas in CS are you interested in?
Logistics: Course mode

- Mode: Hybrid but to COVID (some components in-person, other components online)
  - In-person lectures: 1014 DOW
  - Online lectures: Zoom (check course webpage!)
    - 75-90mins with 3mins break
    - Ask questions during lecture: unmute yourself and ask
    - Non-urgent questions/comments: type in chat
  - Recordings available after class
- Mode will be available at least 1 week prior to class (check schedule online!)
- Office hours: T/TH 4:30-5:30pm eastern time (Zoom)
Logistics: Course structure

• Research-oriented seminar class
• Reading papers, presentations, discussions
• Lectures on basics and general landscape
Logistics: Course structure (cont’d)

• Content: Program synthesis techniques & applications

• Four modules
  • Module 1: Programming-by-Example Techniques
  • Module 2: More techniques
  • Module 3: More applications
  • Module 4: Final Project Presentations
Logistics: Course structure (cont’d)

• Module 1: Programming-by-Example Techniques
  • A set of fundamental ideas/techniques underlying many program synthesizers
  • After this module, you should be familiar with all these techniques

• Module 2: More techniques
  • A set of more advanced techniques
  • You should be able to solve many problems using these techniques

• Module 3: Applications
  • Interesting applications that combine different techniques
Logistics: What do you need to do?

• Paper presentation(s): 1-2 papers/student, depending on how many students enrolled

• Paper reviews: at most 2 reviews per week

• Participation: discuss, ask questions, brainstorm new ideas, ...

• Final project: team (1-2 people), proposal, checkpoints, final report, final presentation
Logistics: Paper presentation

- Identify 1-2 papers you want to present
- Send to instructor
- If not, you may get any paper

- Prepare (e.g., slides, demo, thoughts, ideas, discuss with others)

- Present (45m talk + 30m QA)
  - Thorough (45m is quite a long time)
  - Give high-level ideas as well as important lower-level technical details
  - Introduce necessary background
- Send to instructor
- If not, you may get any paper
Logistics: Paper reviews

- Write a review (template available on course page)
  - A short summary, pros, cons
- Questions
- Thoughts

- Send to instructor via email by midnight the day before class
Logistics: Participation

- Attend
- Ask questions (don’t be shy!)
- Express your opinions
- Connect to your research
- Your ideas
- …
Logistics: Final project

• Find a teammate (solo is also okay; but if more than 3, check with instructor)
  • Sooner than later!

• Generate ideas
  • Sooner than later!

• Write proposal

• Checkpoints: Progress report

• Final presentation

• Final report
Logistics: Final project (cont’d)

- Different kinds of final projects
  - Extend/improve a technique in a paper
  - Apply an existing synthesis framework to a new problem domain
  - Develop a new synthesis technique for an existing problem
  - Develop a new synthesis technique for a new problem
  - …

- Grading of final project is based on: originality, completeness, scope
Logistics: Final project (cont’d)

• Proposal

• 1-2 pages, like an introduction, also include a timeline and a sketch of solution

• need to convince me your problem is worth solving and is technically challenging

• also need to convince me you are able to solve it within 2 months (at least partially)
Logistics: Final project (cont’d)

- Checkpoints
- Nothing but a progress report
- A partial final report that is gradually more complete over time
• Final project report

• 6-8 pages, structured like a conference paper

• Include:
  • Introduction — why this project
  • Motivating example — illustrate how your technique works concretely
  • Technical details — make sure to first give high-level idea before showing details
  • Evaluation — how it works in practice
  • Related work — how your idea relates to existing work
• Paper presentation: 20%
• Paper reviews: 30% (2% x 15)
• Participation: 5%
• Final project: 45%
  • Proposal: 5%
  • Checkpoints: 16% (8% x 2)
  • Final project presentation: 12%
  • Final project report: 12%
What is this course about?

• This course is about program synthesis, including both techniques and applications

• Techniques: general synthesis algorithms not necessarily tied to a specific application

• Applications: novel application of program synthesis techniques

• Beyond acquiring knowledge about program synthesis, also:

  • PL thinking

  • Writing, presentation, ...
You should take this course

• if you are doing or plan to do research in program synthesis
• if you are interested in programming languages research
• if you plan to explore possibilities of applying PL in your own research
• if you just want to learn about the topic!
What is “program synthesis”? 

- What is “program”?  
  - C/C++/Java/Python… 
  - Haskell/ML/OCaml/Lisp/… 
  - SQL/Datalog/… 
  - …

- Synthesis from what?  
  - Input-output examples 
  - Natural language 
  - Demonstrations 
  - …
Example 1: FlashFill [Gulwani et al. 11]

- Synthesize Excel macros for string processing from input-output examples (video)
Example 2: SQLizer [Yaghmazadeh et al. 17]

- Synthesize SQL queries from natural language (given schema)

**NL:**

“Find the number of papers in OOPSLA 2010”

**Schema:**

**SQL query:**

```sql
SELECT count(Publication.pid) FROM Publication JOIN Conference ON Publication.cid = Conference.cid WHERE Conference.name = "OOPSLA" AND Publication.year = 2010
```
Example 3: Rousillon [Chasins et al. 18]

• Synthesize web scraping scripts from example demonstrations (video)
Example 3: Rousillon [Chasins et al. 18] (cont’d)

- Synthesize web scraping scripts from example demonstrations ([video](#))
What is “program synthesis”?

“Program Synthesis correspond to a class of techniques that are able to generate a program from a collection of artifacts that establish semantic and syntactic requirements for the generated code.”

1 [http://people.csail.mit.edu/asolar/SynthesisCourse/Lecture1.htm](http://people.csail.mit.edu/asolar/SynthesisCourse/Lecture1.htm)
Program Synthesis vs. Machine Learning/Deep Learning

• ML/DL is also program synthesis?

• ML/DL: data is spec, model is program, try to learn a model that matches data

• At a high-level, yes

• But in this class, no, at least not the focus

  • Definitions of “programs” are very different (e.g., grammar vs. neural nets)

  • Data is noisy whereas spec is less noisy (but there is a trend in program synthesis to tolerate noise in spec)

  • Typically continuous in ML/DL vs. discrete search space in program synthesis

  • The line is getting blurry
Program Synthesis vs. Compilers

- Program synthesizers are compilers? Compilers are synthesizers?
- Compilers also convert high-level intent (code) to lower-level code
- At a high-level, yes
- But in this class, no, at least not the focus
  - Compilers translate (well, not really nowadays) whereas synthesizers discover
  - Compilers apply predefined transformations (again, not really nowadays) whereas synthesizers perform search
  - The line is getting blurry
Working definition of program synthesis in this course

High-level intent
Specifications

I/O examples, demonstrations, natural language, reference implementation, etc.

Program synthesis
Typically involves search

Program

Lower-level code

In some programming language (grammar + semantics)
Why program synthesis?

• Many useful applications  
  • E.g., FlashFill in Excel  
  
• Technically challenging  
  • Exponential search space (or even undecidable)  
  
• Cool  
  • Intersection of many areas: PL, AI, FM, systems, logics, ...
Three pillars of program synthesis [Gottschlich et al. 18]

- Intention
  - How do users specify their goals?
  - Examples, demonstrations, NL, ..., or their combinations!
  - Challenges: under-specified, ambiguous, unstructured

- Invention
  - How to find the right solution?
  - Search-based, representation-based, learning-based, ..., and their combinations!
  - Challenges: scalability, ambiguity

- Adaptation
  - How to find the right solution, not starting from scratch?
  - Bug fixes, patches, extension to new hardwares, ...
  - Challenges: analyzing, learning, scalability
This course

• Module 1: Techniques for example-based specs
  • Representation-based techniques (both top-down and bottom-up)
  • Search-based techniques (both top-down and bottom-up)
  • Using deduction to guide search and prune search space

• Module 2: Techniques for specs beyond just examples
  • Specs: reference implementation, types, NL, multi-modal
  • Techniques: CEGIS, ML/DL-based, combinations, interactive

• Module 3: Applications
  • Super-optimizations, SE, web, DB, security, graphics, arch, …
Timeline (still tentative)

Module 1
- Sept 4. Paper assignment

Module 2
- Oct 2. FP team & ideas
- Oct 14. FP proposal

Module 3
- Oct 30. Checkpoint 1
- Nov 20. Checkpoint 2
- Dec 9. FP report
- Dec 1,3,8(?). FP presentations
Summary of this lecture

• Program synthesis is cool

• You should take this class and learn about it

• You will learn a lot from this class
Next lecture (Sept 3)

• Syntax-guided synthesis
  • Popular framework for program synthesis

• Representation-based techniques
  • Top-down: FlashFill [Gulwani11], Sept 8
  • Bottom-up: Dace [Wang17], Sept 10

• Search-based techniques
  • Top-down: L2 [Feser15], Sept 15
  • Bottom-up: Optional readings of Sept 15 — [Udupa13], [Albarghouthi13]
Survey (optional)

• Send me a brief email with:
  
  • Name
  
  • I am a [CS/__] [PhD/Masters/undergrad] in year [1/2/3/4/5/…]
  
  • Write one reason why you are taking this class or one thing you want to get out of it
  
  • One thing you would like the instructor to do in this class
  
  • One fun fact about you, or what you like to do in your spare time, or whatever