CS 515 Lecture 0

Course Overview and Logistics

He Zhu

What is this course about?

- This course is about program synthesis, including its basic techniques and general landscape, as well as its applications and impact on programming languages design and compiler implementation.
    - General program synthesis algorithm not necessarily tied to a specific application
    - Novel application of program synthesis techniques.

- Beyond acquiring knowledge about program synthesis, along the way, we will take about program analysis and verification, automated reasoning, theorem proving, formal methods, deep learning in PL …

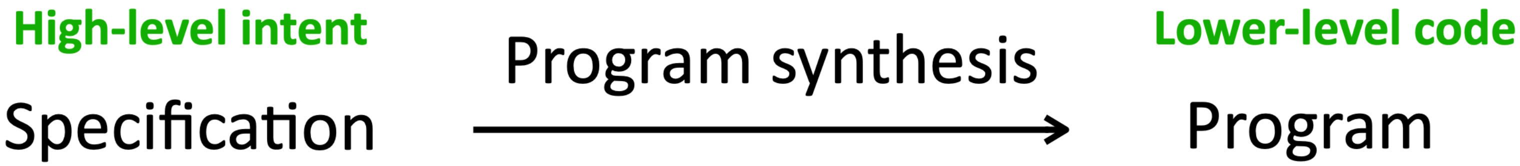
What is “program synthesis”?

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

- Synthesis from what?
  - Input-out examples
  - Natural languages
  - Demonstrations
  - …

What is “program synthesis”? (cont’d)

- “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]



High-level intent

Specification

Program synthesis

Lower-level code

Program

[1] http://people.csail.mit.edu/asolar/SynthesisCourse/Lecture1.htm

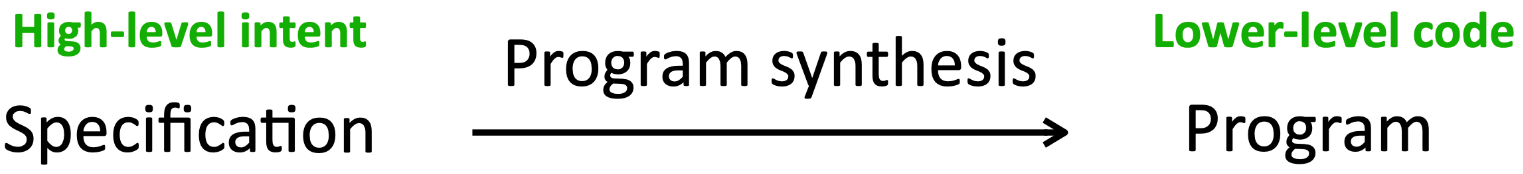
Program Synthesis vs. Machine Learning / Deep Learning

- 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

Specification

Program synthesis

Lower-level code

Program

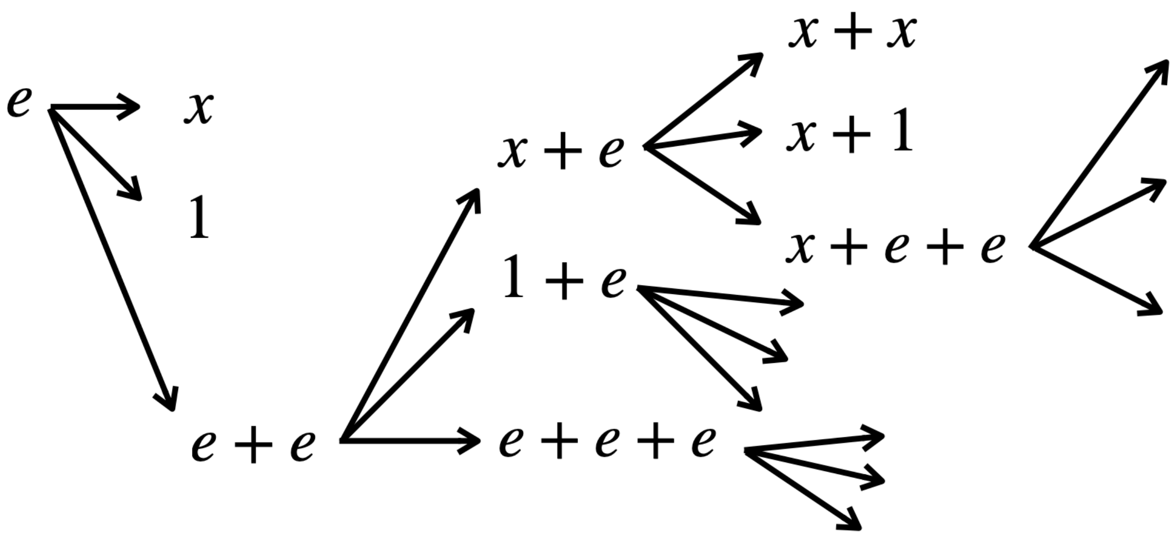
- I/O examples, demonstrations, natural languages, reference implementations, formal logical specifications

- In some programming language (grammar + semantics)

Example

- Program Grammar:
  - e := x | 1 | e + e

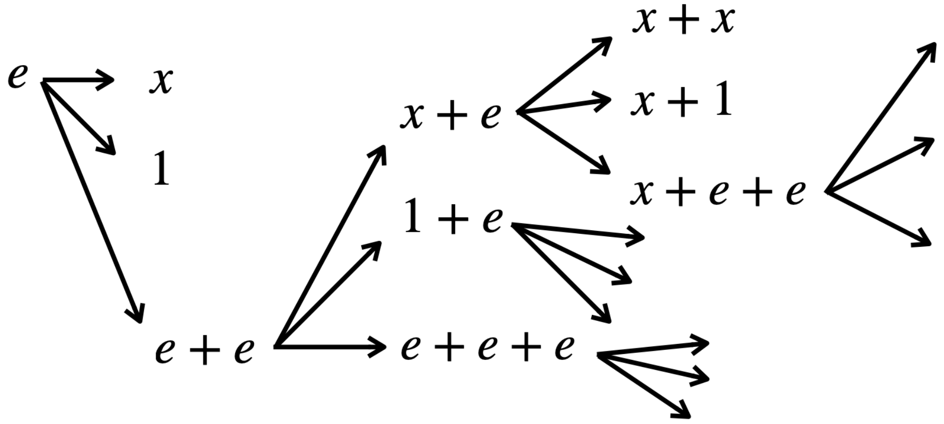
Meta-variable



- Step1 : begin with the start symbol
- Step 2: pick a meta-variable in current result and replace it with one of its productions
- Step 3: continue step 2 until no more non-meta var remains

Example (cont'd):

- Specification: 1 -> 2 (IO example)
- Our first synthesizer:



e

x

1

$e+e$

$x+e$

$1+e$

$e+e+e$

$x+x$

$x+1$

$x+e+e$

iter 0: e

iter 1: $x \quad 1 \quad e+e$

iter 2: $1 \quad e+e$

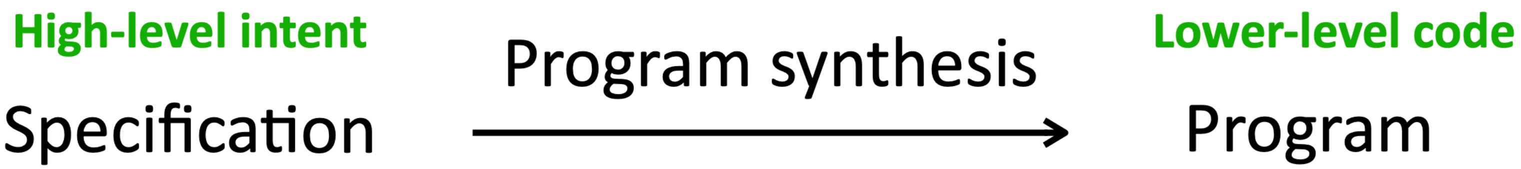
iter 3: $e+e$

iter 4: $x+e \quad 1+e \quad e+e+e$
$e+x \quad e+1 \quad e+e+e$

iter 5: $x+x \quad x+1 \quad x+e+e$
$1+e \quad e+e+e$
$e+x \quad e+1 \quad e+e+e$

iter 6: **return** $x+x$

Example (cont’d)



High-level intent

Specification

Program synthesis

Lower-level code

Program

- IO: 1 -> 2

- Result: x + x

Why studying program synthesis?

- Many useful applications
  - E.g. data transformation 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]

- Intension
  - 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 solutions, not starting from scratch?
  - Bug fixes, patches, extension to new hardware, …
  - Challenges: analyzing, learning, scalability

Example:

- Syntax

e := f | **concat**(f, e)

f := s | **substr**(x, p, p)

p := k | **position**(x, r, k)

r := t | **seq**(t, …, t)

t := <num> | <let> | <ws> | <any> | …

s is string constant, k is int constant,
x is input variable

- Semantics

- Specification

“Bill Gates” —> “BG”

Some sample programs in this language:

concat(“a”, “b”)

“12ab” —> ???

concat(“a”, **substr**(x, 0, 1))

“12ab” —> ???

concat(“a”, **substr**(x, 0, **position**(x, <num>, 1))

“12ab” —> ???

concat(**substr**(x, 0, 1),
          substr(x, **position**(x, <ws>, 1), **position**(x, <cap>, 2)))

What does this program do?

Example (cont’d):

- “Bill Gates” —> “BG”

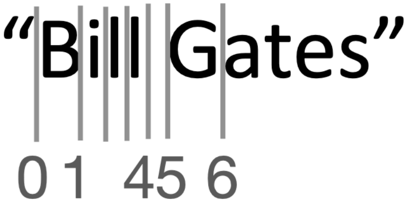
e := f | **concat**(f, e)

f := s | **substr**(x, p, p)

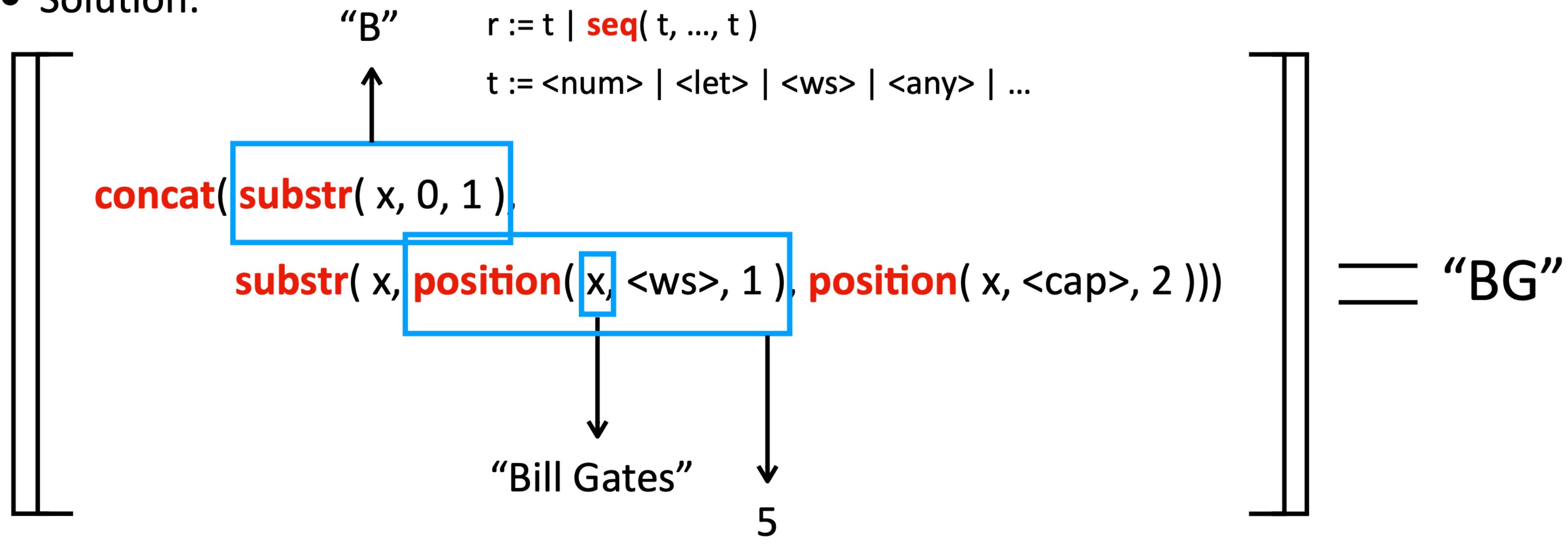
p := k | **position**(x, r, k)

r := t | **seq**(t, …, t)

t := <num> | <let> | <ws> | <any> | …



- Solution:



concat(**substr**(x, 0, 1),
substr(x, **position**(x, <ws>, 1), **position**(x, <cap>, 2)))

= “BG”

Example (cont’d):

- “Bill Gates” —> “BG”
  - Given solution, simple to check correctness
  - … but we do not have solution a priori (only spec!)
  - How to find the solution?

e := f | **concat**(f, e)

f := s | **substr**(x, p, p)

p := k | **position**(x, r, k)

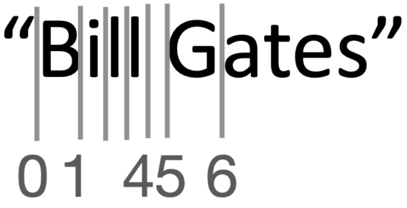
r := t | **seq**(t, …, t)

t := <num> | <let> | <ws> | <any> | …

Example (cont’d):

- Huge search space (easily >10^{20} in simplified FlashFill language!) — how to scale?

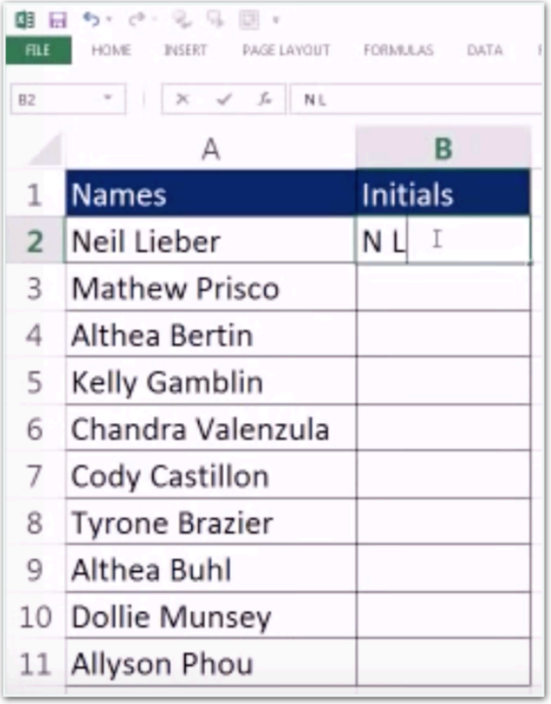
```
e := f | concat( f, e )
f := s | substr( x, p, p )
p := k | position( x, r, k )
r := t | seq( t, …, t )
t := <num> | <let> | <ws> | <any> | …
```



- Ambiguity — how to find the desired program w/o too many examples?

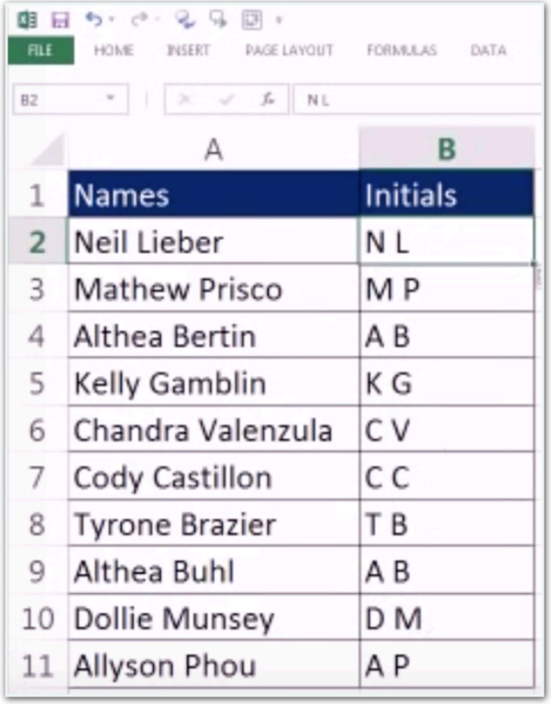
```
concat( substr( x, 0, 1 ), substr( x, position( x, <ws>, 1 ), position( x, <cap>, 2 )))
concat( substr( x, 0, 1 ), substr( x, 5, 6 )))
concat( “B”, “G” )
concat( “B”, substr( x, position( x, <ws>, 1 ), position( x, <cap>, 2 )))
…
```

Example (cont’d):



	A	B
1	Names	Initials
2	Neil Lieber	N L
3	Mathew Prisco	
4	Althea Bertin	
5	Kelly Gamblin	
6	Chandra Valenzula	
7	Cody Castillon	
8	Tyrone Brazier	
9	Althea Buhl	
10	Dollie Munsey	
11	Allyson Phou	

Excel Flash Fill →



	A	B
1	Names	Initials
2	Neil Lieber	N L
3	Mathew Prisco	M P
4	Althea Bertin	A B
5	Kelly Gamblin	K G
6	Chandra Valenzula	C V
7	Cody Castillon	C C
8	Tyrone Brazier	T B
9	Althea Buhl	A B
10	Dollie Munsey	D M
11	Allyson Phou	A P

Logistics: Course mode

- Lectures:
  - Online : Zoom (https://rutgers.zoom.us/j/97915693526)
  - 10 mins break after every 50 mins.
  - Ask questions during lecture: unmute yourself and ask.
- Office hours:
  - Right after each lecture or by appointment.
- Course website:
  - https://github.com/RU-Automated-Reasoning-Group/CS515/wiki
  - The website will be private; please send your GitHub id to instructor by the end of this week.

Logistics: What do you need to do?

- Paper reviews: 6-8 papers
- Paper presentation: 1 per student.
- Participation: discuss, ask questions, brainstorm new ideas, …
- Programming assignments: ~3 ***lightweight*** tasks
- Final project: team (1~2 people), proposal, checkpoints, final report, final presentation.

Logistics: Paper reviews

- Write a review:
  - A short summary
  - Pros
  - Cons
  - Answer questions provided by instructor
  - Your own thoughts (e.g. extensions, improvements, alternative solutions)
- Submit to Canvas (by deadlines)

Logistics: Paper presentation

- There is a reading list.
- Identify 2~3 paper you want to present
  - Send to instructor and you will be assigned one paper.
  - If not, you may get any paper.
- Prepare (e.g. slides, demo, thoughts, ideas, discuss with others)
- Present (45m talks + 30m QA)
  - Thorough (45m is quite a long time).
  - Give high-level ideas as well as important lower-level technical details.
  - Introduce necessary background.

Logistics: Participation

- Attend
- Ask questions (anytime)
- Express your opinions (anytime)
- Connect to your research
- Your ideas
- …

Logistics: Final project

- Projects are expected to be done in teams of **one or two**.
    - The scope of the project should be commensurate with the size of the team
    - Find your teammate by posting using the issue tracker on the course website.
- Generate ideas:
    - There will be a list of project ideas on the course website.
    - You are encouraged to propose your own idea.
    - Kinds of projects:
        - re-implement a technique from a paper
        - apply existing synthesis framework to a new domain
        - extend/improve existing synthesis algorithm or tool
        - develop a new synthesis algorithm or tool

Logistics: Final project (cont’d)

- Judged in terms of
  - quality of execution
  - originality
  - scope
- Write proposal (1 page)
- Checkpoints: progress report (1 page)
- Final representation (30-45m)
- Final report (3-8 pages)

Logistics: Final project (cont’d)

- Proposal:
  - 1 page, like an introduction, also include a timeline and a sketch of solution
  - Need to show your problem is worth solving and is technically challenging.
  - Also need to show you are able to solve it within 2 months.

Logistics: Final project (cont’d)

- Checkpoints
  - Nothing but a progress report (1 page)
  - A partial final report that is gradually more complete over time.

Logistics: Final project (cont’d)

- Final project report
  - 3-8 pages, structured like a conference paper
  - Include:
    - Introduction – why this project.
    - Motivating example – illustrate how your tool works concretely.
    - Technical details – make sure to first give high-level idea before details.
    - Evaluation – how it works in practice.
    - Related work – how your idea relates to existing work (***in the reading list***).

Logistics: Grading

- Paper reviews: 30%
  - 6 ~ 8 papers.
- Paper presentation: 10%
  - Lead the discussion of a paper (from the reading list).
- Programming assignments: 30%
  - 3 lightweight assignments
- Final Project: 30%
  - 1-page project proposal: 5%
  - Project presentation: 10%
  - Final code & report: 15%