Synthesis

Reading Reflection

Discuss in groups

- If you could express your intent to the computer in any way at all, how would you want to write programs?
  - What input would you have the computer take?
  - How would the interaction between you and the computer work?
- What was confusing about synthesis from the first reading/your understanding of synthesis so far?
  - It's ok if this is lots of things! We'll be getting hands-on soon, which should clear up a lot of confusions. :)
- Are there applications that you'd expect are amenable to synthesis but that haven't made it into the literature yet? (Weren't mentioned in Chapter 2.)

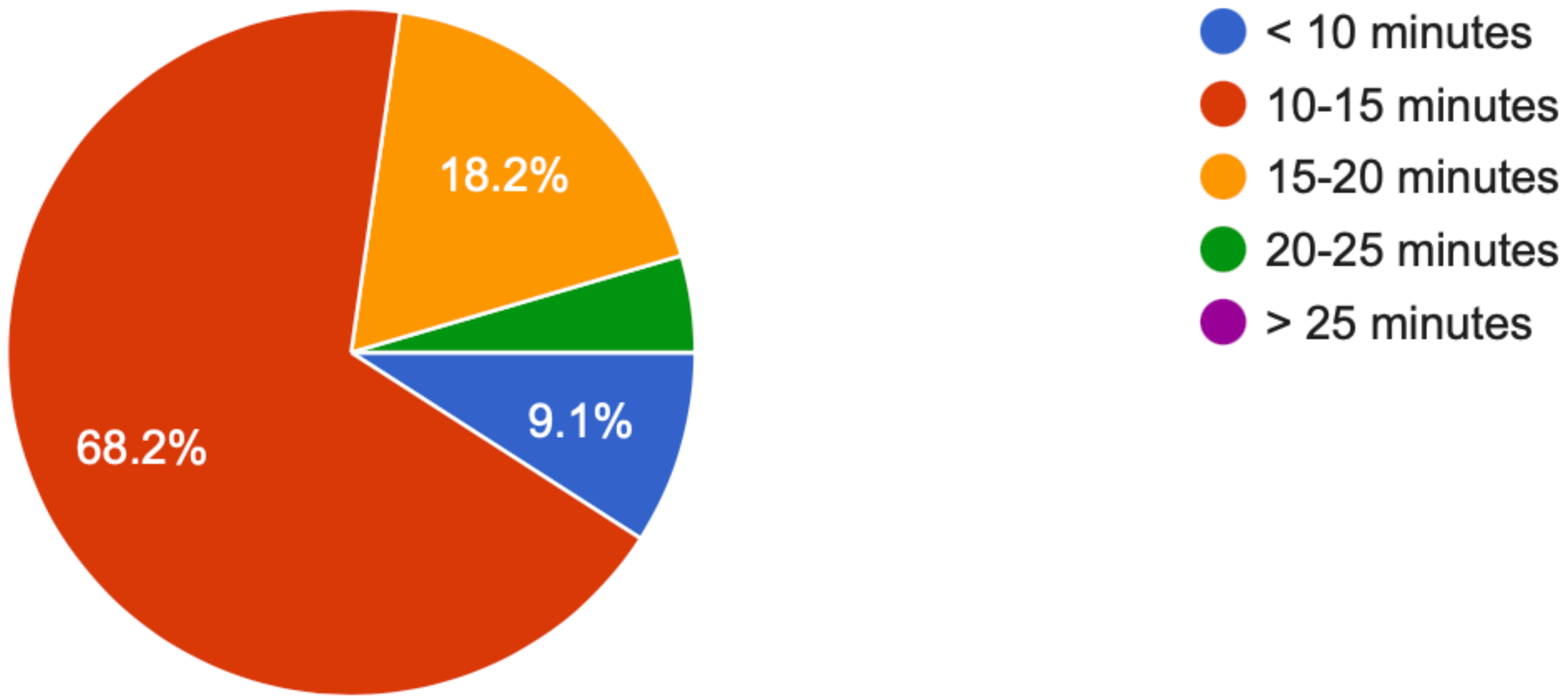
Reading Key Takeaways

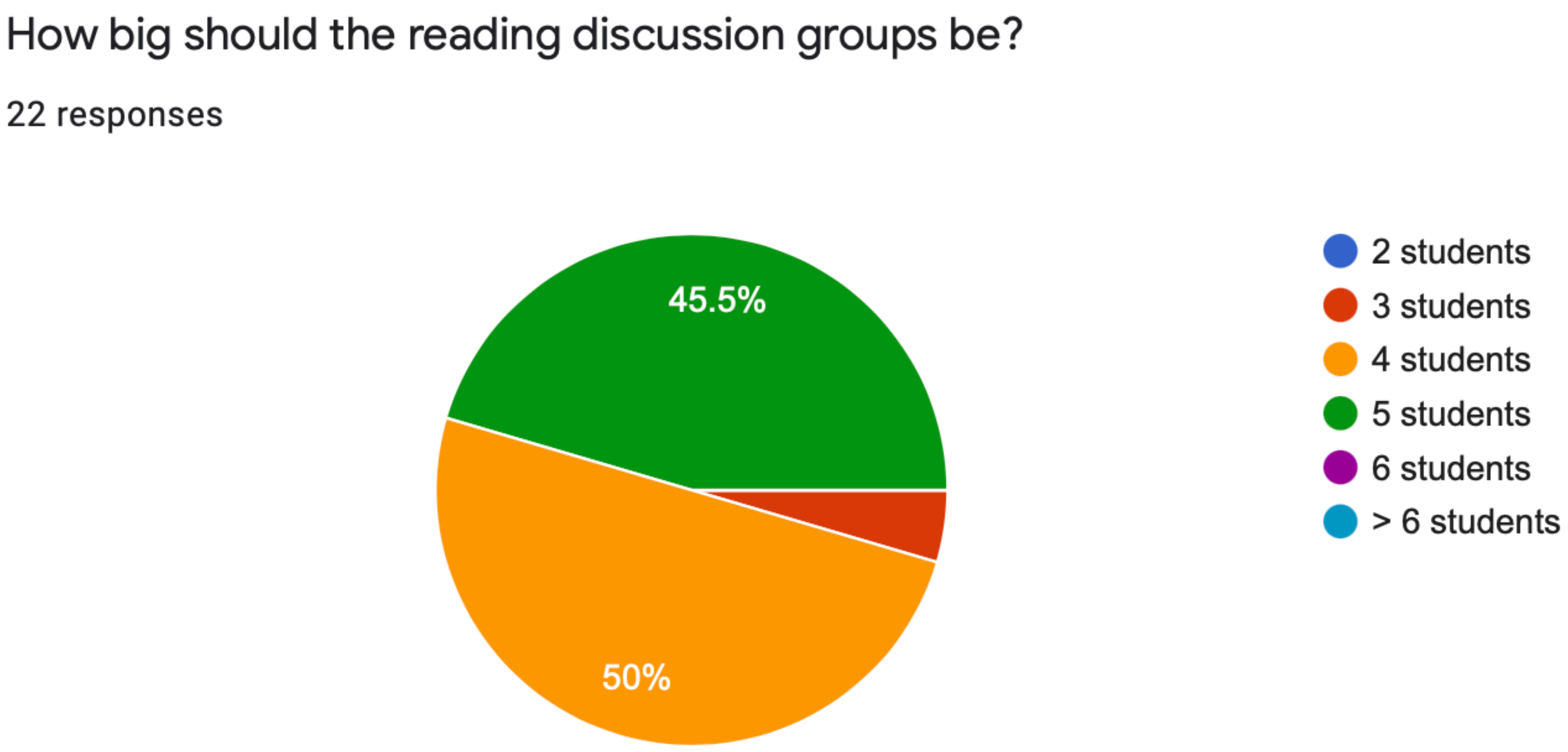
- The core challenges in synthesis:
  - Scalability/size of the program space
  - Capturing user intent—What's a good spec? How do we get it?
- The variety of plausible specs we can get from users
  - I/O examples, demonstrations, logical specs, natural language, programs with holes, equivalent programs (!)
- The variety of search techniques
  - Enumerative, constraint-based, deductive, statistical
  - And at a higher level, the fact that synthesis is not just **one** technique
- A general sense of the problems to which synthesis has been applied so far

Thank you for your survey answers!

How much time should we spend in the reading discussion groups?

22 responses

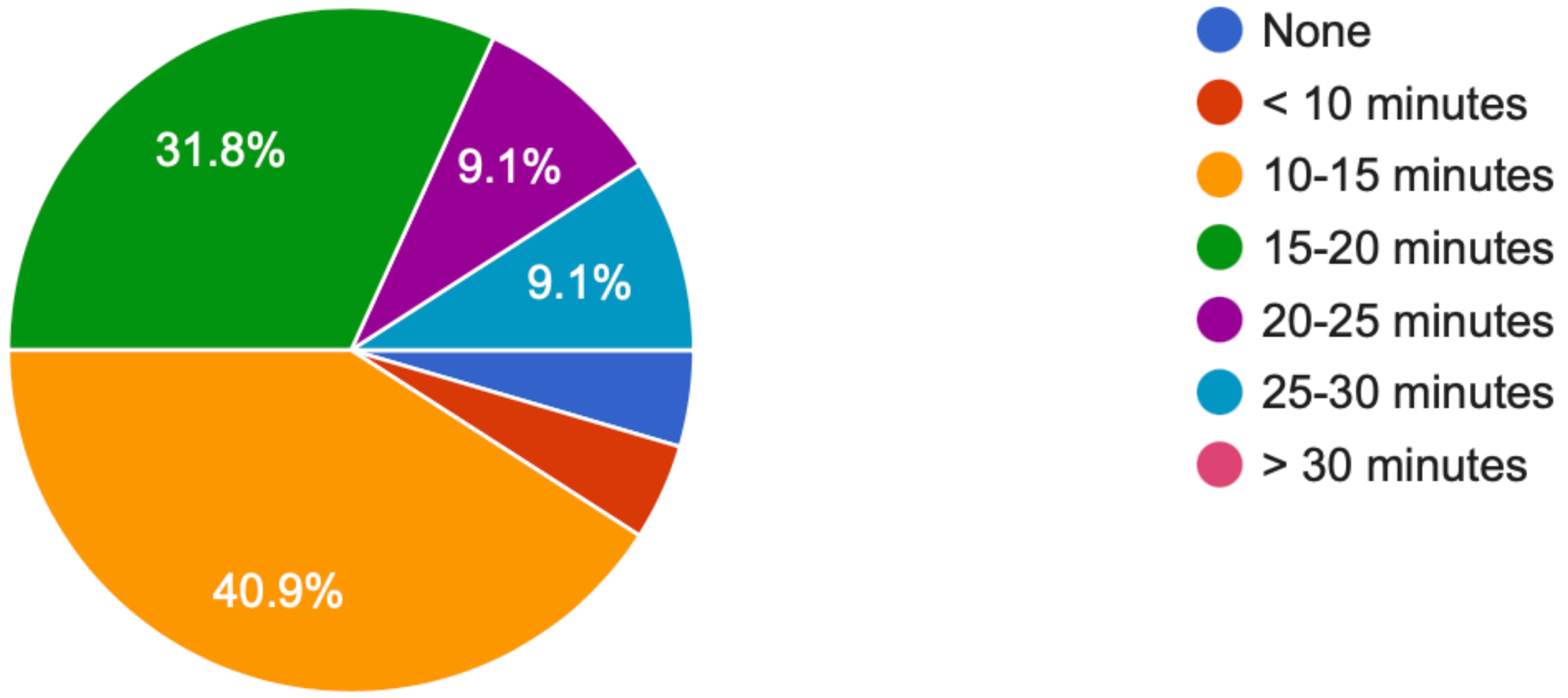




I forgot to ask if it's ok that I walk around during group discussion time. Feel free to let me know if you have strong feelings about this.

How much time should we spend in the assignment work groups at the end of class?

22 responses



How do you feel about the mini breaks in the middle?

22 responses

Other changes

- Some comments that readings can be a bit long; related, that it’d be nice to mix in non-reading resources
  - I’ll be extra clear about which readings are fine to skim vs. require quite close reading!
    - (Sorry, other people said length/amount of content was a KEEP, so we’re not going to completely remove long readings.)
  - I’ll start mixing in some non-reading resources for topics that have good non-paper sources
- Discussions on recent PL+HCI works or work that folks in class are doing
  - This is coming up! (Basically as soon as we’ve built up a foundation.)

Why synthesis?

There are a few PL techniques that just keep coming up in HCI tasks!

- Program synthesis
- Projection/Structure editors
- Program slicing

Others come up, but these seem to come up all the time.

Demo time

FlashFill

Do you have Excel installed? You can probably run this demo on your own laptop while I run it on mine!

B2 Prof. Cheung

	A	B
1	Whole Name	
2	Alvin Cheung	Prof. Cheung
3	Armando Fox	
4	Jonathan Ragan-Kelley	
5	Koushik Sen	
6	Sanjit A. Seshia	
7	Katherine A. Yelick	

B3

	A	B	C
1	Whole Name	Prof. Name	
2	Alvin Cheung	Prof. Cheung	
3	Armando Fox	Prof. Fox	
4	Jonathan Ragan-Kelley	Prof. Kelley	
5	Koushik Sen	Prof. Sen	
6	Sanjit A. Seshia	Prof. Seshia	
7	Katherine A. Yelick	Prof. Yelick	
8			

CTRL + E

Scythe

To run this one, head to: https://scythe.cs.washington.edu/demo

Scythe
Home
Demo
Click the Synthesis button to synthesize Queries from the example!
Empty Panel
Remove Panel
Load An Example Panel
Connect to Database
Offline (No backend DB connected)
Example Task: Find the span of career peak (the year when the first paper and most cited papers are published) of computer scientists given the list of their pushlished papers.
papers
author | title | year | citation
H. Simon | Understanding wi | 1974 | 277
H. Simon | Organization | 1997 | 20057
H. Simon | The sciences of tl | 1996 | 17561
R. Tibshirani | Developmental re | 2004 | 50
R. Tibshirani | Flexible discrimin | 1995 | 51
R. Tibshirani | IRF9 and STAT1 a | 2008 | 47
P. Bork | Automated pair-w | 1998 | 51
P. Bork | UN targets top kil | 2011 | 19
Add Row
Add Col
Del Col
Constants
None
Aggregators
(Optional)
output
author | min_year | peak_year
H. Simon | 1974 | 1997
R. Tibshirar | 1995 | 1995
P. Bork | 1998 | 1998
Add Row
Add Col
Del Col
Select t6.author,t6.min_year,t6.year
From
(Select t5.author, t5.min_year, t4.author As author1, t4.year
From ((Select
t3.author, Min(t3.year) As min_year
From
papers As t3
Group By
t3.author) As t5 Join
(Select t7.author,t7.year
From
(Select t1.author, t1.max_citation, t8.author As author1, t8.title, t8.year, t8.citation
From ((Select
t2.author, Max(t2.citation) As max_citation
From
papers As t2
Group By
t2.author) As t1 Join
papers As t8)) As t7
Where t7.max_citation = t7.citation
And t7.author = t7.author1) As t4)) As t6
Where t6.author = t6.author1;
Add Table
Remove Table
Synthesize
Synthesized Query 1
Run on DB
Visualize

Helena

If you want to run this one, you have to install an extension:

http://helena-lang.org/install

Rousillon: Scraping Distributed Hierarchical Web Data, me & my collaborators :)

Extensions × Profiles × +

https://scholar.google.com/citations?hl=en&view_op=search_authors&mauthors=label%3Acomputer_science&btnG=

Apps Extensions Other Bookmarks

Google Scholar

label:computer_science

Profiles My profile My library

Geoffrey Hinton

Emeritus Prof. Comp Sci, U.Toronto & Engineering Fellow, Google
Verified email at cs.toronto.edu

machine learning neural networks artificial intelligence cognitive science computer science

Cited by 246012

DEYWIS MORENO

High Energy Physicist, Universidad Antonio Narino
Verified email at uan.edu.co

High Energy Physics Computer science

Cited by 206009

David S. Johnson

Visiting Professor, Columbia University Computer Science Department
Verified email at research.att.com

Algorithms computer science optimization traveling salesman problem bin packing

Cited by 176731

David Haussler

Scientific Director, UC Santa Cruz Genomics Institute, University of California, Santa Cruz
Verified email at soe.ucsc.edu

genomics computer science molecular biology evolution cancer

Cited by 174202

vapnik

Cited by 170728

LENS

before	after

```
cmp    r1, #0
mov    r3, r1, asr #31
add    r2, r1, #7
mov    r3, r3, lsr #29
movge  r2, r1
ldrb   r0, [r0, r2, asr #3]
bic    r1, r2, #248
sub    r3, r1, r3
asr    r1, r0, r3
and    r0, r1, #1
```

(b)

```
asr    r3, r1, #2
add    r2, r1, r3, lsr #29
ldrb   r0, [r0, r2, asr #3]
and    r3, r2, #248
sub    r3, r1, r3
asr    r1, r0, r3
and    r0, r1, #1
```

(c)

I know, I know, not as photogenic, but it makes programs much faster!!

Falx

Coming soon to https://falx.cs.washington.edu/

5 min break

Back up. What's program synthesis?

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

Correctness Condition

$$\exists P.\overbrace{\forall x.\phi(x,P(x))}$$

Find *P*

- When to use synthesis:
  - **Ease-of-use/productivity:** When writing ϕ is faster or easier than writing P
  - **Correctness:** when proving ϕ is easier than proving P

Hey, I’ve seen this before

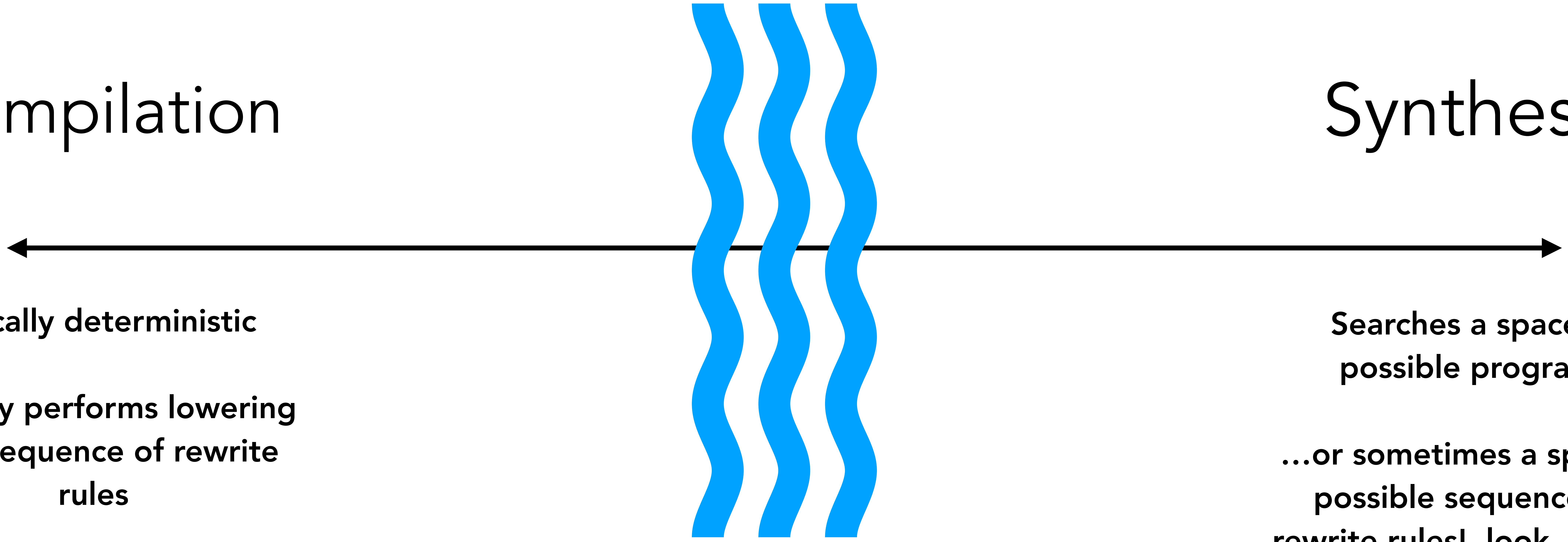
I give computer a high-level description of what I want it to do

Computer gives me back a low-level program for doing it

Synthesis vs. compilation

Compilation

Synthesis



Typically deterministic

Typically performs lowering via a sequence of rewrite rules

Searches a space of possible programs

...or sometimes a space of possible sequences of rewrite rules! look, the line is blurry ¯_(ツ)_/¯

If it involves search, we usually call it synthesis

Even if you don’t take away anything else from today’s lecture, take away that you can write a synthesizer!

Even if you don't take away anything else from today's lecture, take away that you can write a synthesizer!

What do we need to decide to make a synthesizer?

Hint: 3 things

How does the user express what they want the program to do?

What space of programs is the synthesizer allowed to use?

What algorithm will the synthesizer use to search that space?

What do we need to decide to make a synthesizer?

Hint: 3 things For today's sample synthesizer, let's pick…

How does the user express what they want the program to do?

Input-Output examples

What space of programs is the synthesizer allowed to use?

Anything in a Domain-Specific Language (DSL) of our choice

What algorithm will the synthesizer use to search that space?

Which is to say…generating programs until we find one that works

Input-Output Examples

- Any work here?
- Nah, this is going to be pretty straightforward.
- Example:

```
({"x" → 3, "y" → 7},     23)

({"x" → 4, "y" → 4},     19)

({"x" → 2, "y" → 12},    31)
```

Can you guess it?? Did you already synthesize this in your head?

Domain-Specific Language

- This one's a classic, but for another domain we might design something more customized

$$
\begin{aligned}
\mathit{expr} := \quad & N \\
& \mid v \\
& \mid (\mathit{expr} + \mathit{expr}) \\
& \mid (\mathit{expr} - \mathit{expr}) \\
& \mid (\mathit{expr} * \mathit{expr})
\end{aligned}
$$

Enumeration

Spec:

```
({"x" → 3, "y" → 7},  23)
({"x" → 4, "y" → 4},  19)
({"x" → 2, "y" → 12}, 31)
```

Space of programs:

$$
\begin{aligned}
expr := \;& \mathcal{N} \\
|\;& v \\
|\;& (expr + expr) \\
|\;& (expr - expr) \\
|\;& (expr * expr)
\end{aligned}
$$

```
level 0:
[0, 1, 2, 3, 4, y, x]
count: 7
```

Ok, no luck so far. Let's just mash these together! In every possible combination!

```
level 1 :
[0, 1, 2, 3, 4, y, x, (0+0), (0*0), (0-0), (0+1), (0*1), (0-1), (0+2), (0*2), (0-2),
(0+3), (0*3), (0-3), (0+4), (0*4), (0-4), (0+y), (0*y), (0-y), (0+x), (0*x), (0-x),
(1+0), (1*0), (1-0), (1+1), (1*1), (1-1), (1+2), (1*2), (1-2), (1+3), (1*3), (1-3),
(1+4), (1*4), (1-4), (1+y), (1*y), (1-y), (1+x), (1*x), (1-x), (2+0), (2*0), (2-0),
(2+1), (2*1), (2-1), (2+2), (2*2), (2-2), (2+3), (2*3), (2-3), (2+4), (2*4), (2-4),
(2+y), (2*y), (2-y), (2+x), (2*x), (2-x), (3+0), (3*0), (3-0), (3+1), (3*1), (3-1),
(3+2), (3*2), (3-2), (3+3), (3*3), (3-3), (3+4), (3*4), (3-4), (3+y), (3*y), (3-y),
(3+x), (3*x), (3-x), (4+0), (4*0), (4-0), (4+1), (4*1), (4-1), (4+2), (4*2), (4-2),
(4+3), (4*3), (4-3), (4+4), (4*4), (4-4), (4+y), (4*y), (4-y), (4+x), (4*x), (4-x),
(y+0), (y*0), (y-0), (y+1), (y*1), (y-1), (y+2), (y*2), (y-2), (y+3), (y*3), (y-3),
(y+4), (y*4), (y-4), (y+y), (y*y), (y-y), (y+x), (y*x), (y-x), (x+0), (x*0), (x-0),
(x+1), (x*1), (x-1), (x+2), (x*2), (x-2), (x+3), (x*3), (x-3), (x+4), (x*4), (x-4),
(x+y), (x*y), (x-y), (x+x), (x*x), (x-x)]
count: 154
```

Hm, still no luck. Keep mashing.

```
level 2 :
[0, 1, 2, 3, 4, y, x, (0+0), (0*0), (0-0), (0+1), (0*1), (0-1), (0+2), (0*2),
(0-2), (0+3), (0*3), (0-3), (0+4), (0*4), (0-4), (0+y), (0*y), (0-y), (0+x), (0*x),
(0-x), (1+0), (1*0), (1-0), (1+1), (1*1), (1-1), (1+2), (1*2), (1-2), (1+3), (1*3),
(1-3), (1+4), (1*4), (1-4), (1+y), (1*y), (1-y), (1+x), (1*x), (1-x), (2+0), (2*0),
(2-0), (2+1), (2*1), (2-1), (2+2), (2*2), (2-2), (2+3), (2*3), (2-3), (2+4), (2*4),
(2-4), (2+y), (2*y), (2-y), (2+x), (2*x), (2-x), (3+0), (3*0), (3-0), (3+1), (3*1),
(3-1), (3+2), (3*2), (3-2), (3+3), (3*3), (3-3), (3+4), (3*4), (3-4), (3+y), (3*y),
(3-y), (3+x), (3*x), (3-x), (4+0), (4*0), (4-0), (4+1), (4*1), (4-1), (4+2), (4*2),
(4-2), (4+3), (4*3), (4-3), (4+4), (4*4), (4-4), (4+y), (4*y), (4-y), (4+x), (4*x),
(4-x), (y+0), (y*0), (y-0), (y+1), (y*1), (y-1), (y+2), (y*2), (y-2), (y+3), (y*3),
(y-3), (y+4), (y*4), (y-4), (y+y), (y*y), (y-y), (y+x), (y*x), (y-x), (x+0), (x*0),
(x-0), (x+1), (x*1), (x-1), (x+2), (x*2), (x-2), (x+3), (x*3), (x-3), (x+4), (x*4),
(x-4), (x+y), (x*y), (x-y), (x+x), (x*x), (x-x), (0+0), (0*0), (0-0), (0+1), (0*1),
count: 71,302
```

Enumeration…pruned with Operational Equivalence

←Which is the fancy program synthesis way of saying "they do the same thing on the inputs we care about."

Spec:

({"x" → 3, "y" → 7}, 23)
({"x" → 4, "y" → 4}, 19)
({"x" → 2, "y" → 12}, 31)

Space of programs:

$$
\begin{aligned}
\mathit{expr} :=\ & \mathcal{N} \\
|\ & v \\
|\ & (\mathit{expr} + \mathit{expr}) \\
|\ & (\mathit{expr} - \mathit{expr}) \\
|\ & (\mathit{expr} * \mathit{expr})
\end{aligned}
$$

Ok, these are all just 0…which we already have. Why'd you give me these???

```
level 0:
[0, 1, 2, 3, 4, y, x]
count: 7

level 1 :
[0, 1, 2, 3, 4, y, x, (0+0), (0*0), (0-0), (0+1), (0*1), (0-1), (0+2), (0*2), (0-2),
(0+3), (0*3), (0-3), (0+4), (0*4), (0-4), (0+y), (0*y), (0-y), (0+x), (0*x), (0-x),
(1+0), (1*0), (1-0), (1+1), (1*1), (1-1), (1+2), (1*2), (1-2), (1+3), (1*3), (1-3),
(1+4), (1*4), (1-4), (1+y), (1*y), (1-y), (1+x), (1*x), (1-x), (2+0), (2*0), (2-0),
(2+1), (2*1), (2-1), (2+2), (2*2), (2-2), (2+3), (2*3), (2-3), (2+4), (2*4), (2-4),
(2+y), (2*y), (2-y), (2+x), (2*x), (2-x), (3+0), (3*0), (3-0), (3+1), (3*1), (3-1),
(3+2), (3*2), (3-2), (3+3), (3*3), (3-3), (3+4), (3*4), (3-4), (3+y), (3*y), (3-y),
(3+x), (3*x), (3-x), (4+0), (4*0), (4-0), (4+1), (4*1), (4-1), (4+2), (4*2), (4-2),
(4+3), (4*3), (4-3), (4+4), (4*4), (4-4), (4+y), (4*y), (4-y), (4+x), (4*x), (4-x),
(y+0), (y*0), (y-0), (y+1), (y*1), (y-1), (y+2), (y*2), (y-2), (y+3), (y*3), (y-3),
(y+4), (y*4), (y-4), (y+y), (y*y), (y-y), (y+x), (y*x), (y-x), (x+0), (x*0), (x-0),
(x+1), (x*1), (x-1), (x+2), (x*2), (x-2), (x+3), (x*3), (x-3), (x+4), (x*4), (x-4),
(x+y), (x*y), (x-y), (x+x), (x*x), (x-x)]
count: 154
```

And these are the same on all inputs.

And eventually we'll find some that aren't the same on *all* inputs, but *are* the same on {"x" → 3, "y" → 7}, {"x" → 4, "y" → 4}, and {"x" → 2, "y" → 12}

This is exactly as simple as it looks. Seriously, you can write this synthesizer in vanilla Python in one page. Let’s see it!

```python
import itertools
class Op:
    ops = {"+": lambda a,b: a+b, "-": lambda a,b: a-b, "*": lambda a,b: a*b}
    def __init__(self, a, op, b):
        self.a = a; self.op = op; self.b = b
    def __repr__(self):
        return "(" + str(self.a) + self.op + str(self.b) + ")"
    def interpret(self, argDict):
        return Op.ops[self.op](self.a.interpret(argDict), self.b.interpret(argDict))
class Val:
    def __init__(self, v):
        self.v = v
    def __repr__(self):
        return str(self.v)
    def interpret(self, argDict):
        return self.v
class Var:
    def __init__(self, n):
        self.n = n
    def __repr__(self):
        return self.n
    def interpret(self, argDict):
        return argDict[self.n]

spec = [({"x": 3, "y": 7}, 23), # our input-output pairs
        ({"x": 4, "y": 4}, 19),
        ({"x": 2, "y": 12}, 31)]
expected_outputs = [output for inputDict, output in spec] # for convenience, the outputs we expect for our i-o pairs
def test_against_spec(expr):
    outputs = [expr.interpret(inputDict) for inputDict, output in spec] # run the expression on our target inputs
    if (outputs == expected_outputs):
        print "found it!", expr
        exit()

exprs = [Val(x) for x in range(5)] + [Var(x) for x in spec[0][0].keys()] # the starting set is 0 through 5 and our args
print "level 0:\n", exprs, "\ncount:", len(exprs)
for expr in exprs:
    test_against_spec(expr) # let's just see if any of our starting exprs do the trick...

ops = Op.ops.keys() # what operators are we allowed to use?
level = 0
while(True):
    level += 1
    print "level", level, ":"
    for pair in itertools.product(exprs, exprs): #let's make bigger expressions!
        for op in ops:
            new_expr = Op(pair[0], op, pair[1])
            test_against_spec(new_expr)
            exprs.append(new_expr)
    print exprs, "\ncount:", len(exprs)
```

This one isn’t pruning at all.
What do we do to prune with OE?

Just an extra 6 lines!

Pruning based on Operational Equivalence can cut down our search space dramatically!

And this is just at level 2!

```
[Sarahs-MBP:othermaterials schasins$ python onePageSynthesizer.py
level 0:
[0, 1, 2, 3, 4, y, x]
count: 7
level 1 :
count: 154
level 2 :
count: 71302
level 3 :
found it! (3+(2*(y+x)))
[Sarahs-MBP:othermaterials schasins$ python onePageSynthesizerOE.py
level 0:
[0, 1, 2, 3, 4, y, x]
count: 7
level 1 :
count: 63
level 2 :
count: 2051
level 3 :
found it! (3+(2*(y+x)))
```

So if you're ever watching a synthesis talk and get confused…just remember enumeration. At a sufficiently high level of abstraction, it's just going through programs until it finds one that works.

We can make enumeration smarter

- Doesn't have to be just start with the smallest program, then list all the programs in order of size until you find one that works
- We can have heuristics or language models that let us explore better/likelier programs first instead of smaller programs first
- There are other ways of pruning (other than Operational Equivalence) that let us cut out much more of the space
- We can make smart choices about what constants to include
- This was the easy-to-write version, but there are many ways to make it more effective
  - For a long time, the winner of the SyGuS competition (the primary competition for people who write synthesizers) was an enumerative solver!
    - This is a real technique!

Quick brainstorm. What would you like to synthesize?

Synthesis is like a buffet

- This is not one technique that either applies or doesn't apply to your problem
- It's a whole family of techniques
  - Tackling a new problem, you'll probably be looking through a host of existing approaches and tools…
    - If you read synth literature, you'll see very different domains formalized in very different ways. This isn't accidental!
  - …and maybe inventing your own. Custom synthesizers are still common

To think about for Thursday's reading

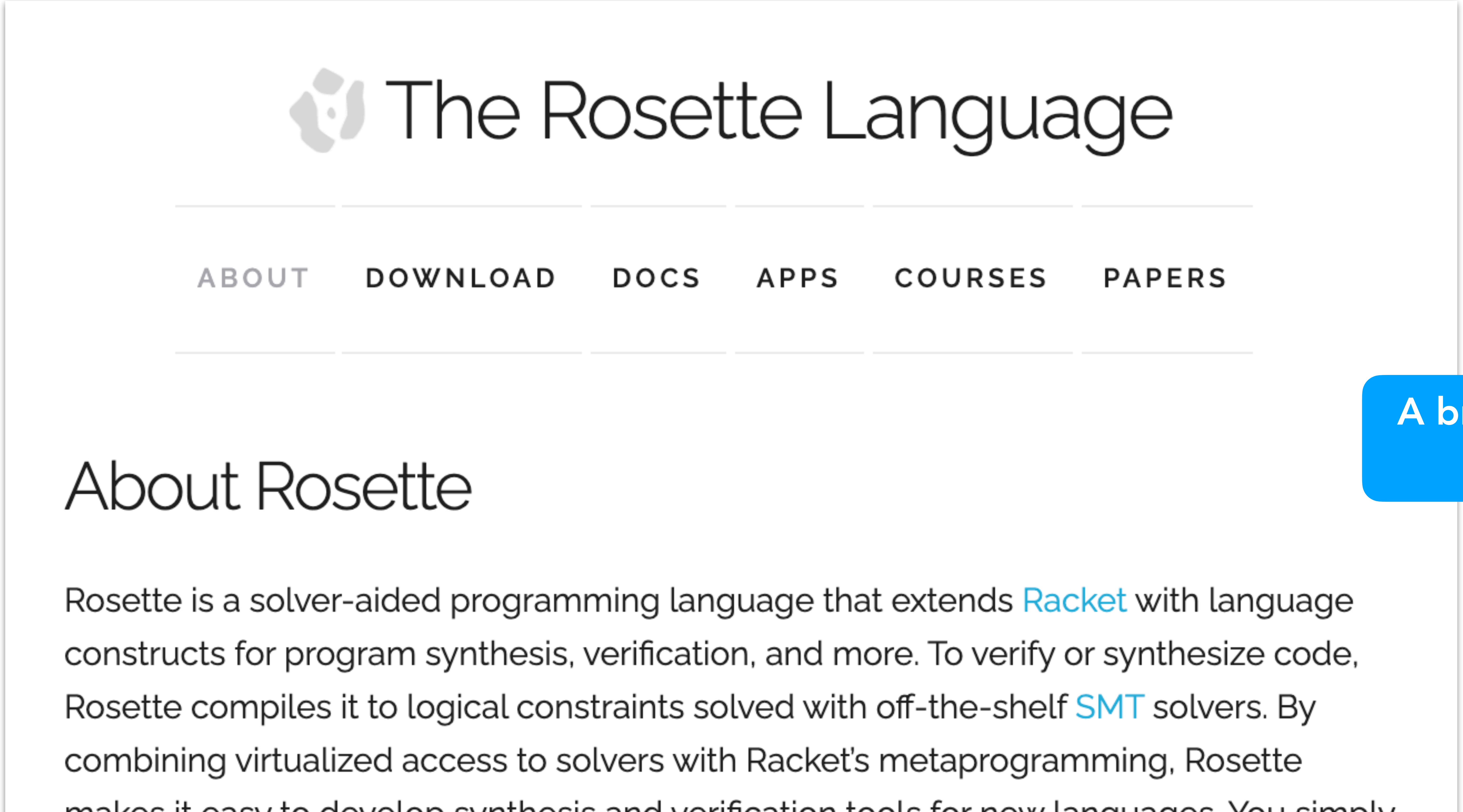
- The issue of ambiguous specs. As designers of usable tools, do we want to prevent ambiguous specs? If yes, how? Do we want to allow them? If yes, how does this affect our synthesizer?

- What constrains the design of a our target languages for synthesis?

  - What's the tradeoff between designing for making the synthesizer's task easier vs. designing for the user of the tool?

Please install before next class

https://docs.racket-lang.org/rosette-guide/ch_getting-started.html#%28part._sec~3aget%29



The Rosette Language

ABOUT DOWNLOAD DOCS APPS COURSES PAPERS

About Rosette

Rosette is a solver-aided programming language that extends Racket with language constructs for program synthesis, verification, and more. To verify or synthesize code, Rosette compiles it to logical constraints solved with off-the-shelf SMT solvers. By combining virtualized access to solvers with Racket's metaprogramming, Rosette makes it easy to develop synthesis and verification tools for new languages. You simply

A brilliant language from Emina Torlak