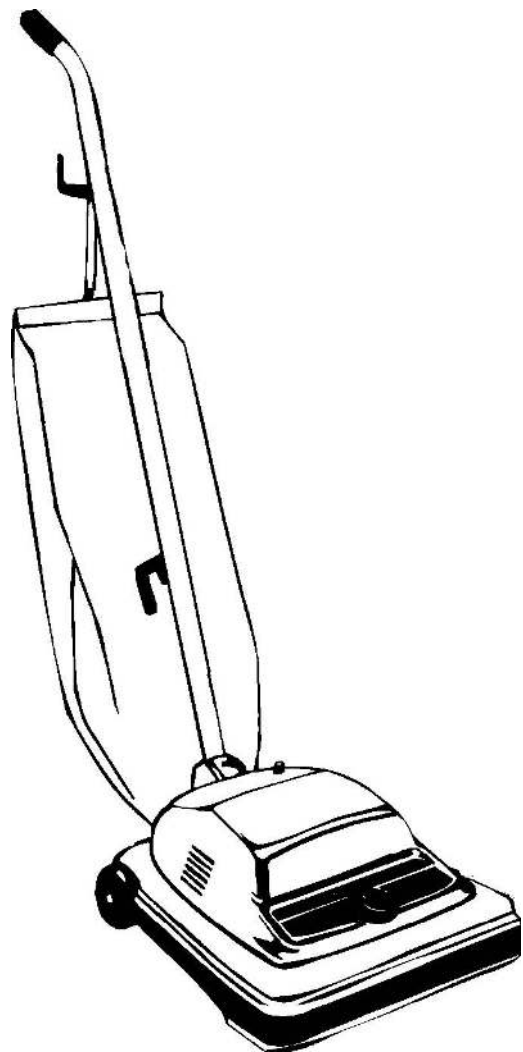# Valid programming with pragmatic program synthesis
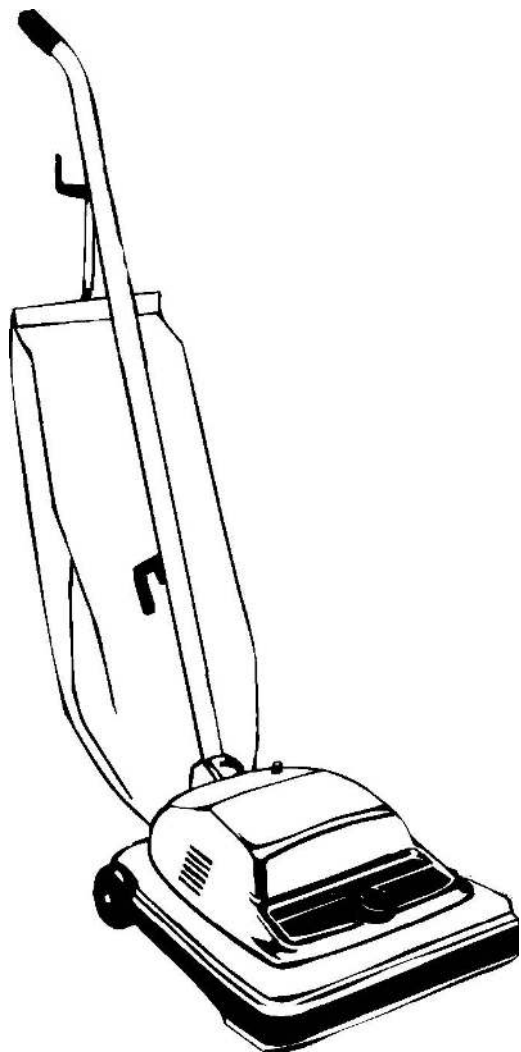
Long Ouyang

Validity: how to ensure that a system that meets its formal requirements does not have unwanted behaviors and consequences (*"Did I build the right system?"*)

Validity: how to ensure that a system that meets its formal requirements does not have unwanted behaviors and consequences ("*Did I build the right system?*")

Validity: how to ensure that a system that meets its formal requirements does not have unwanted behaviors and consequences ("*Did I build the right system?*")

"*clean up as much dirt as possible*"

Validity: how to ensure that a system that meets its formal requirements does not have unwanted behaviors and consequences ("*Did I build the right system?*")
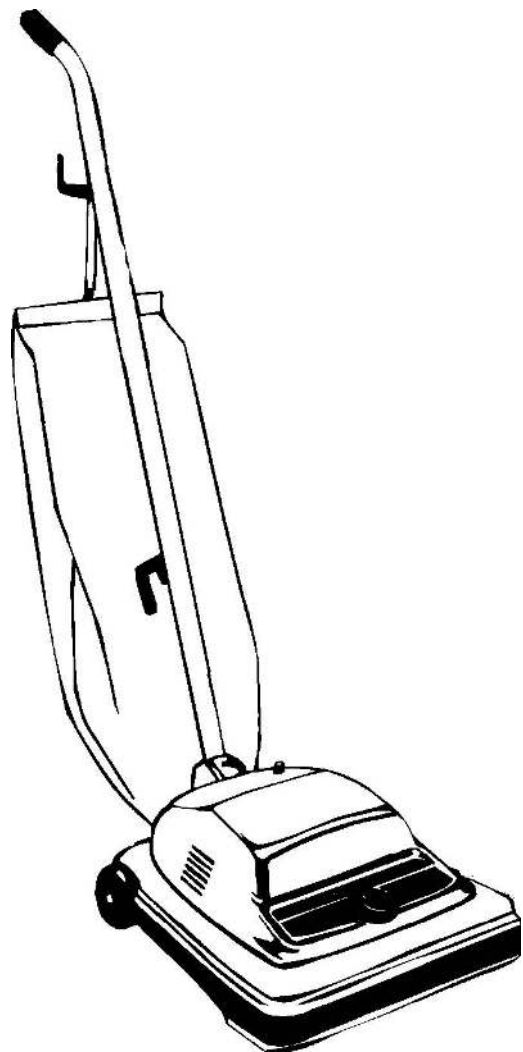
"clean up as much dirt as possible"

finds one patch of dirt, repeatedly picks it up and puts it down

Bad: Imperative specification   *"how"*

Bad: Imperative specification *"how"*

```
# calculate a 15% tip
subtotal = 0
for i in items:
 subtotal += price[i]
tip = 0.15 * subtotal
```

Bad: Imperative specification   *"how"*

```
# calculate a 15% tip
subtotal = 0
for i in items:
 subtotal += price[i]
tip = 0.15 * subtotal
```

Better: declarative specification   *"what"*

Bad: Imperative specification   *"how"*

```
# calculate a 15% tip
subtotal = 0
for i in items:
 subtotal += price[i]
tip = 0.15 * subtotal
```

Better: declarative specification   *"what"*

```
tip([90,10])      = 15,
tip([50,50,100]) = 30,
…
```

Program synthesis
(programming by example)

```
tip([90,10])     = 15,
tip([50,50,100]) = 30,
…
```
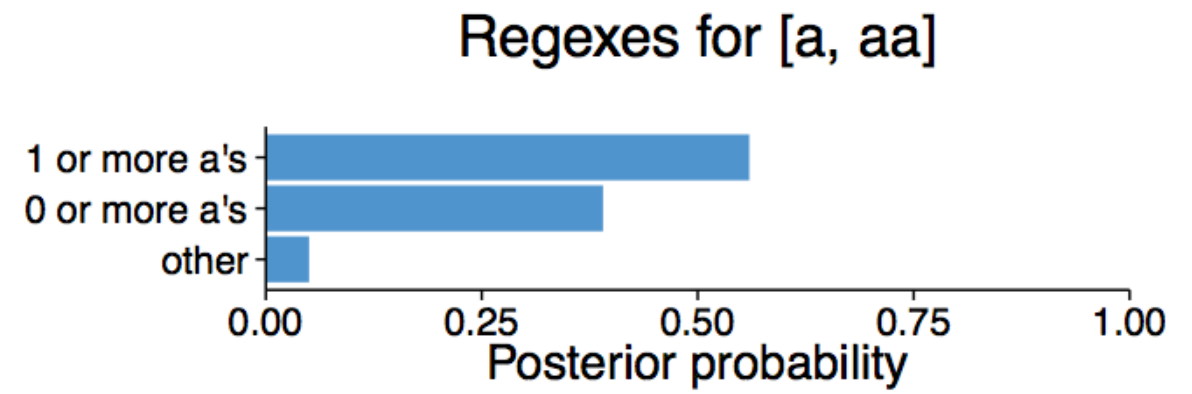
Program synthesis
(programming by example)

"a"   ✔

"aa"   ✔

Program synthesis
(programming by example)

"a" ✔

"aa" ✔

⟶

Regexes for [a, aa]
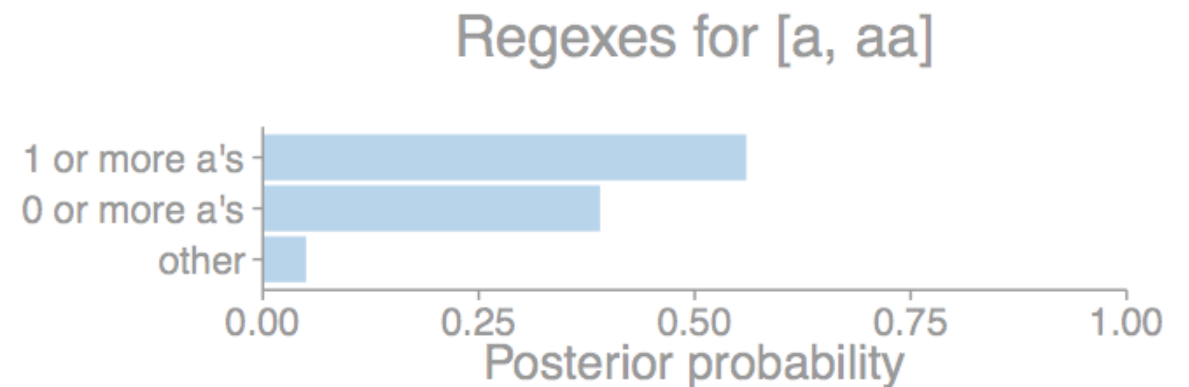
Programming by example is good for validity

- Write tests, get code for free (ish)
- Reduce surface area for errors (e.g., syntax, type errors, mis-specification)
- Enables thinking at high (domain-specific) level of abstraction
- Empowers non-programmers to produce code

But.. PBE can be invalid

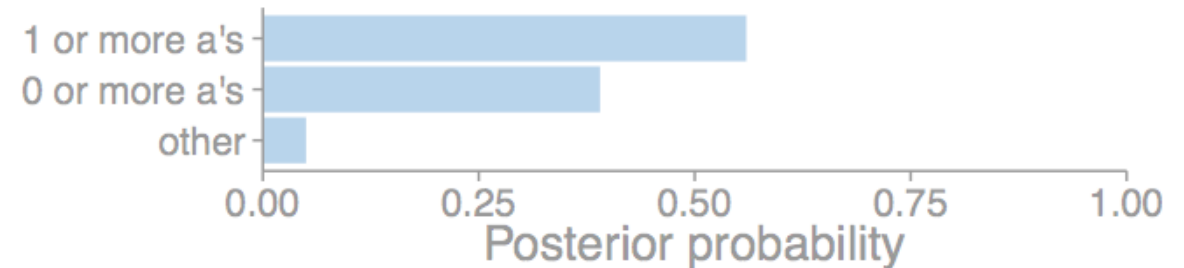# Program synthesis
(programming by example)

"a" ✔
"aa" ✔ →

Regexes for [a, aa]

1 or more a's
0 or more a's
other

0.00   0.25   0.50   0.75   1.00
Posterior probability

"aa" ✔
"aaa" ✔

Program synthesis
(programming by example)

"a" ✔
"aa" ✔ →

Regexes for [a, aa]

1 or more a's
0 or more a's
other

0.00 0.25 0.50 0.75 1.00
Posterior probability

Regexes for [aa, aaa]

"aa" ✔
"aaa" ✔ →

2 or more a's
1 or more a's
0 or more a's
other

0.00 0.25 0.50 0.75 1.00
Posterior probability

Program synthesis
(programming by example)

"a" ✔
"aa" ✔ →

Regexes for [a, aa]

1 or more a's	
0 or more a's	
other	

0.00   0.25   0.50   0.75   1.00
Posterior probability

"aa" ✔
"aaa" ✔ →

Regexes for [aa, aaa]

2 or more a's	
1 or more a's	
0 or more a's	
other	

0.00   0.25   0.50   0.75   1.00
Posterior probability
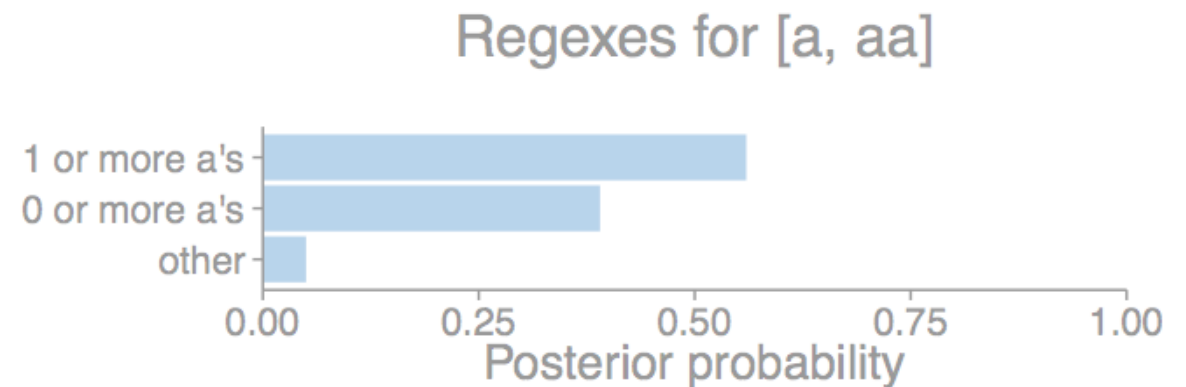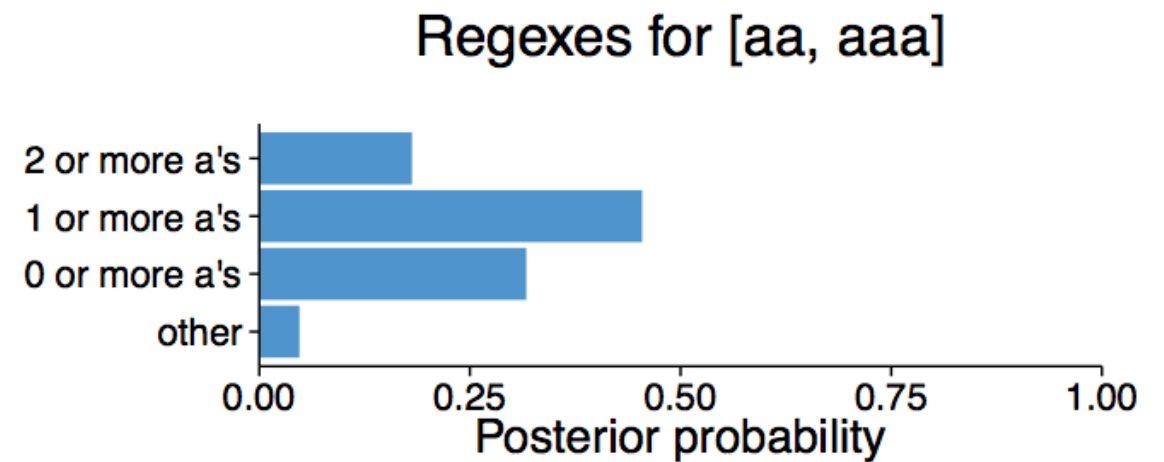
Current synthesis systems interpret examples *literally*

Program synthesis
(programming by example)
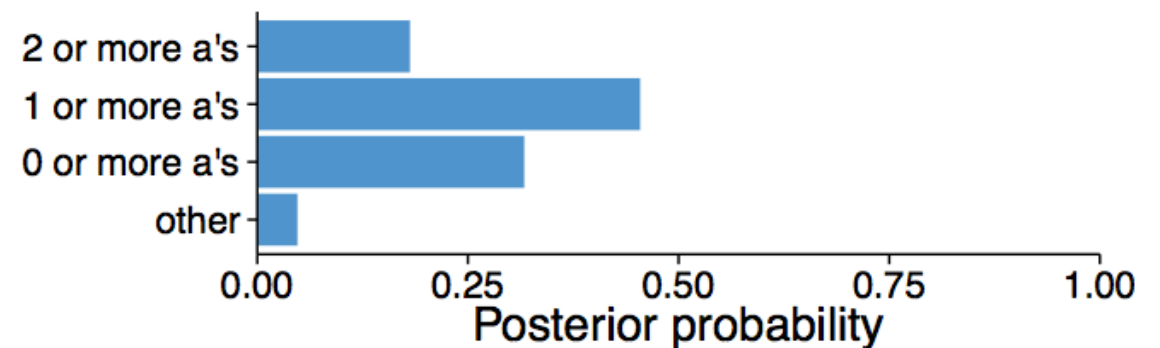

"a" ✔
"aa" ✔   →   **Regexes for [a, aa]**
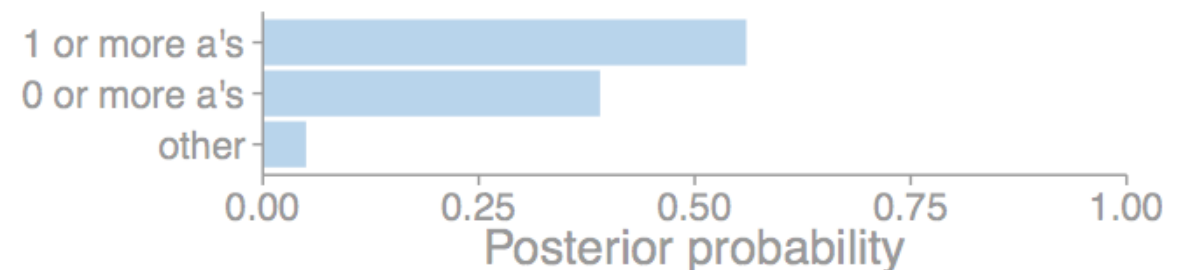
"aa" ✔
"aaa" ✔   →   **Regexes for [aa, aaa]**

Current synthesis systems interpret examples *literally*

Goal: more sophisticated (*pragmatic*) interpretation

Literal vs. pragmatic

Literal vs. pragmatic

"The one with glasses"

Literal vs. pragmatic

"The one with glasses"

Literal: 0   0.5   0.5

Literal vs. pragmatic

"The one with glasses"

Literal:	0	0.5	0.5
Pragmatic:	0	0.9	0.1

Pragmatic program synthesis

"aa" ✔
"aaa" ✔

Literal:
search for programs that
<u>satisfy these examples</u>

Pragmatic:
search for programs that
<u>would make a person</u>
<u>produce these examples</u>

Generative models

$$P(r \mid x) \propto P(r) \times P(x \mid r)$$

Generative models

Literal:

interpret regexes as
PCFGs, do Earley parsing

$$P(r \mid x) \propto P(r) \times P(x \mid r)$$

Generative models

$$P(r \mid x) \propto P(r) \times P(x \mid r)$$

Literal:

interpret regexes as PCFGs, do Earley parsing

Pragmatic:

need a model how people produce examples for particular regexes

So far

Collected data on how people generate examples

Work in progress on regex induction $P(r \mid x)$
  Collaboration: cognitive science research on language acquisition

Work on tooling: `webppl`
  Automated posterior visualization w/ static analysis (POPL '17 PPS workshop)
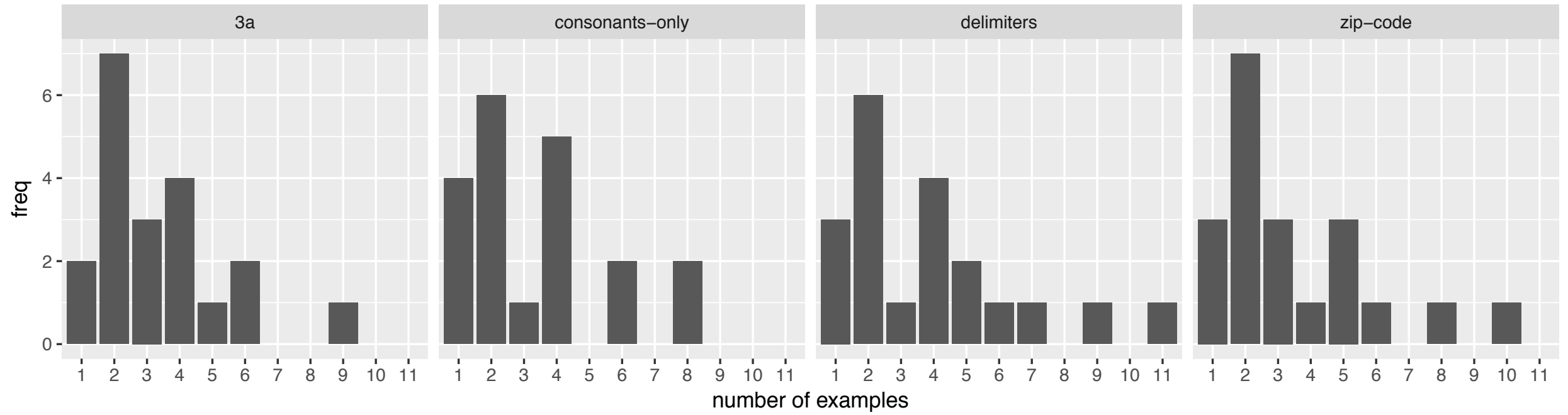 Automated inference?

Initial experimental data
(plan to submit to CogSci '17 but suggestions welcome)

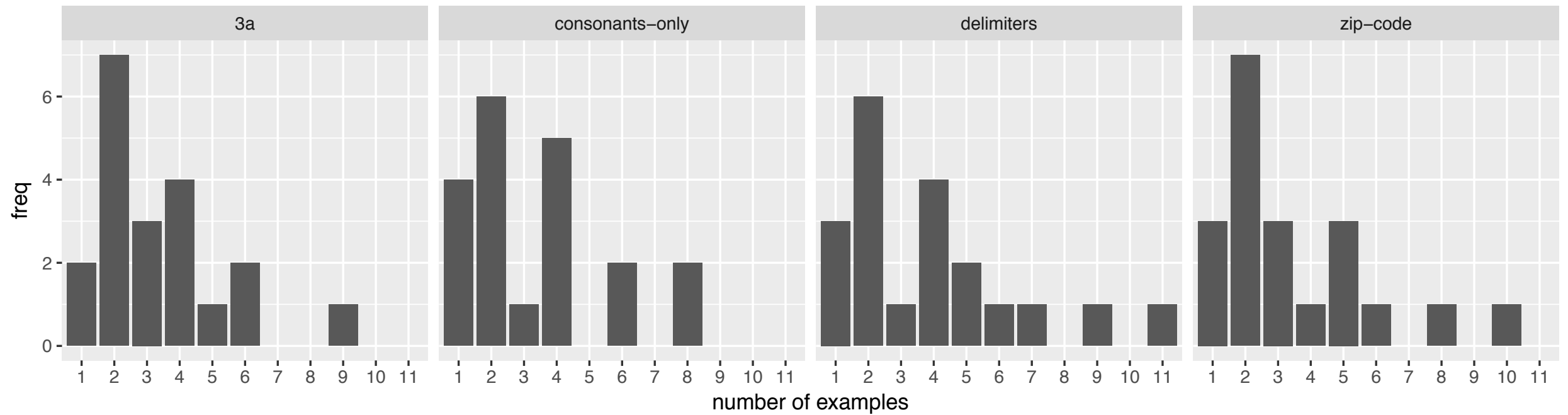Mechanical Turk subjects: mean age ~40, little to no programming experience

Demo

People give between 1 and 11 examples:

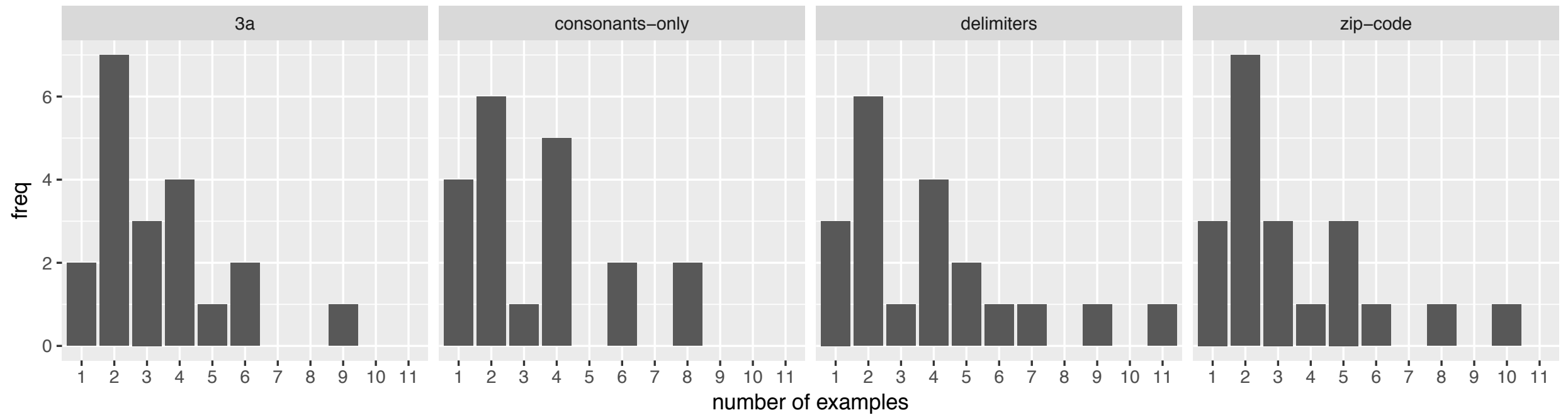# People give between 1 and 11 examples:

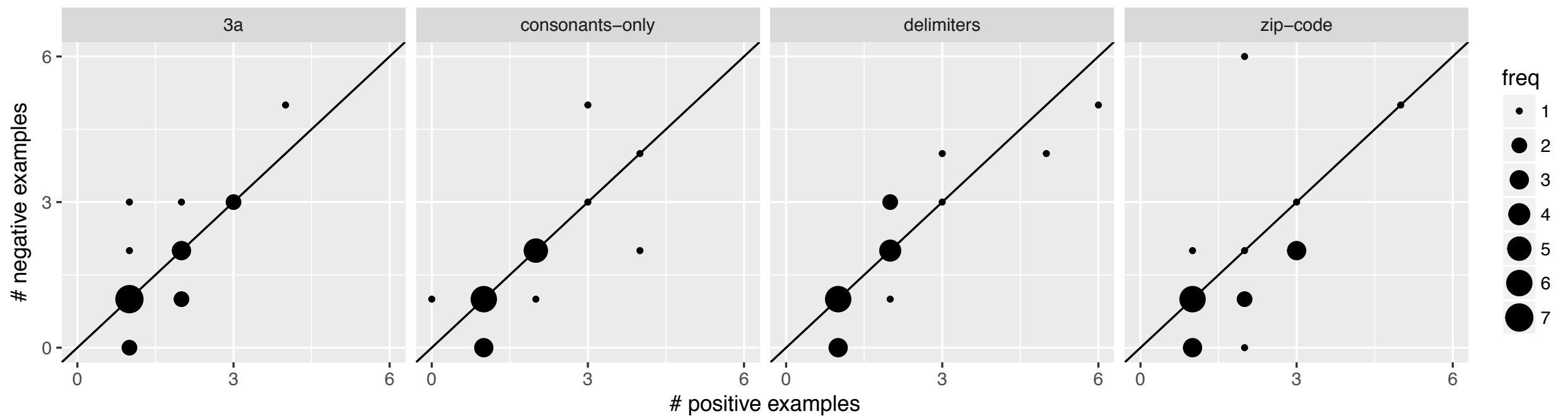# People give between 1 and 11 examples:

Examples are fairly balanced in polarity:

People give between 1 and 11 examples:

Examples are fairly balanced in polarity:

Examples tend to be related
e.g., **[qwerty]** and **qwerty]**, **12521** and **125219**

Examples tend to be related
e.g., **[qwerty]** and **qwerty]**, **12521** and **125219**
(near miss)

Examples tend to be related
e.g., **`[qwerty]`** and **`qwerty]`**, **12521** and **125219**
(near miss)

p < 0.001 by permutation test

Examples tend to be related
e.g., **`[qwerty]`** and **`qwerty]`**, **`12521`** and **`125219`**
                                          (near miss)

p < 0.001 by permutation test

Rich sequencing structure

Examples tend to be related
e.g., **[qwerty]** and **qwerty]**, **12521** and **125219**
(near miss)

p < 0.001 by permutation test

Rich sequencing structure

Ahead

Collect more data, experiment with different stimuli, subjects, prompts, interfaces for example generation

Build pragmatic synthesis system for regular expressions, string transformations
  Other domains: data transformation, data extraction, gesture, planning

Work on efficient inference (PPLs? deep learning?)

Analyze benefits of pragmatic versus literal synthesis