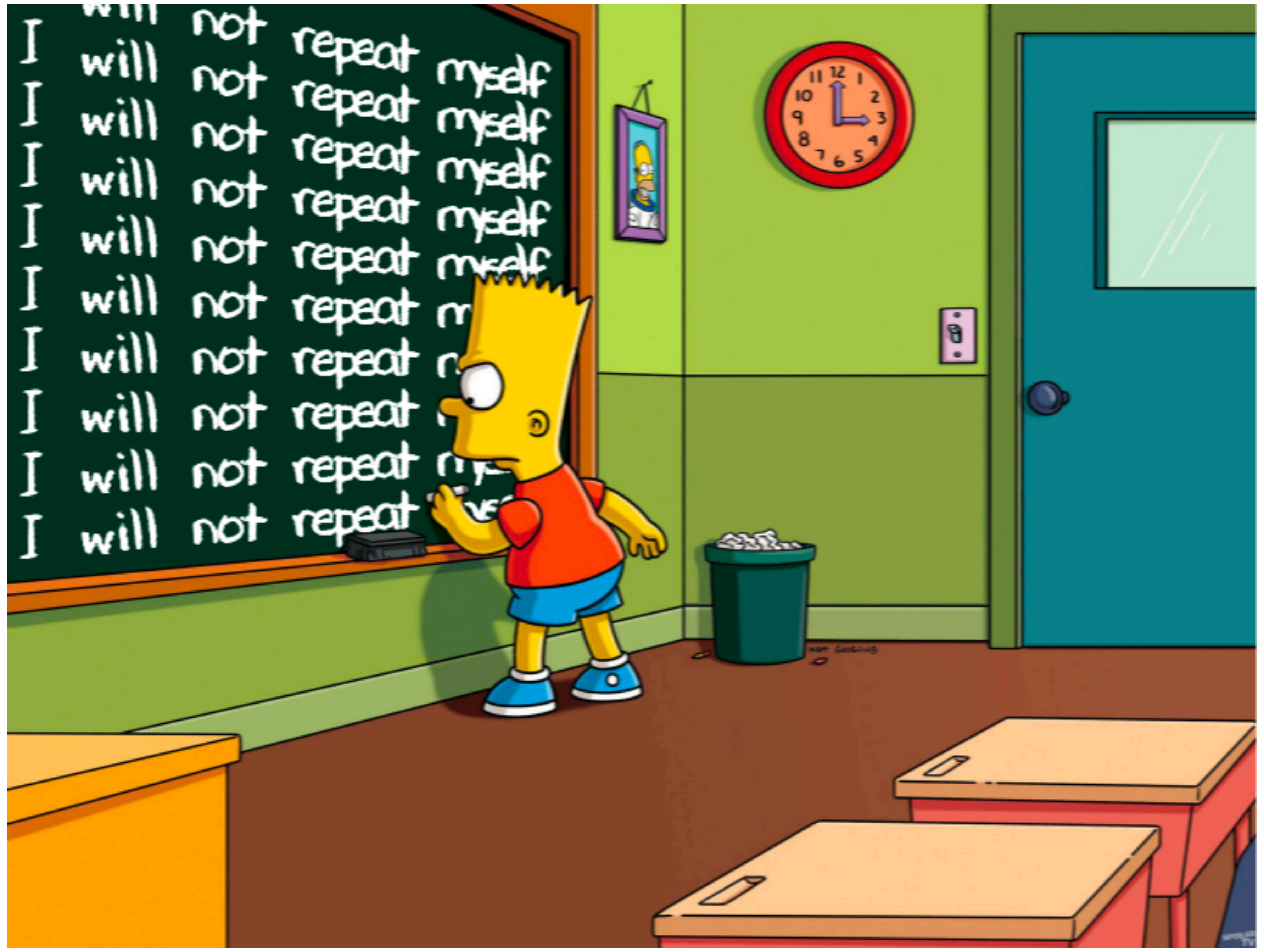
UC San Diego

# Digging for Fold: Synthesis-Aided API Discovery for Haskell

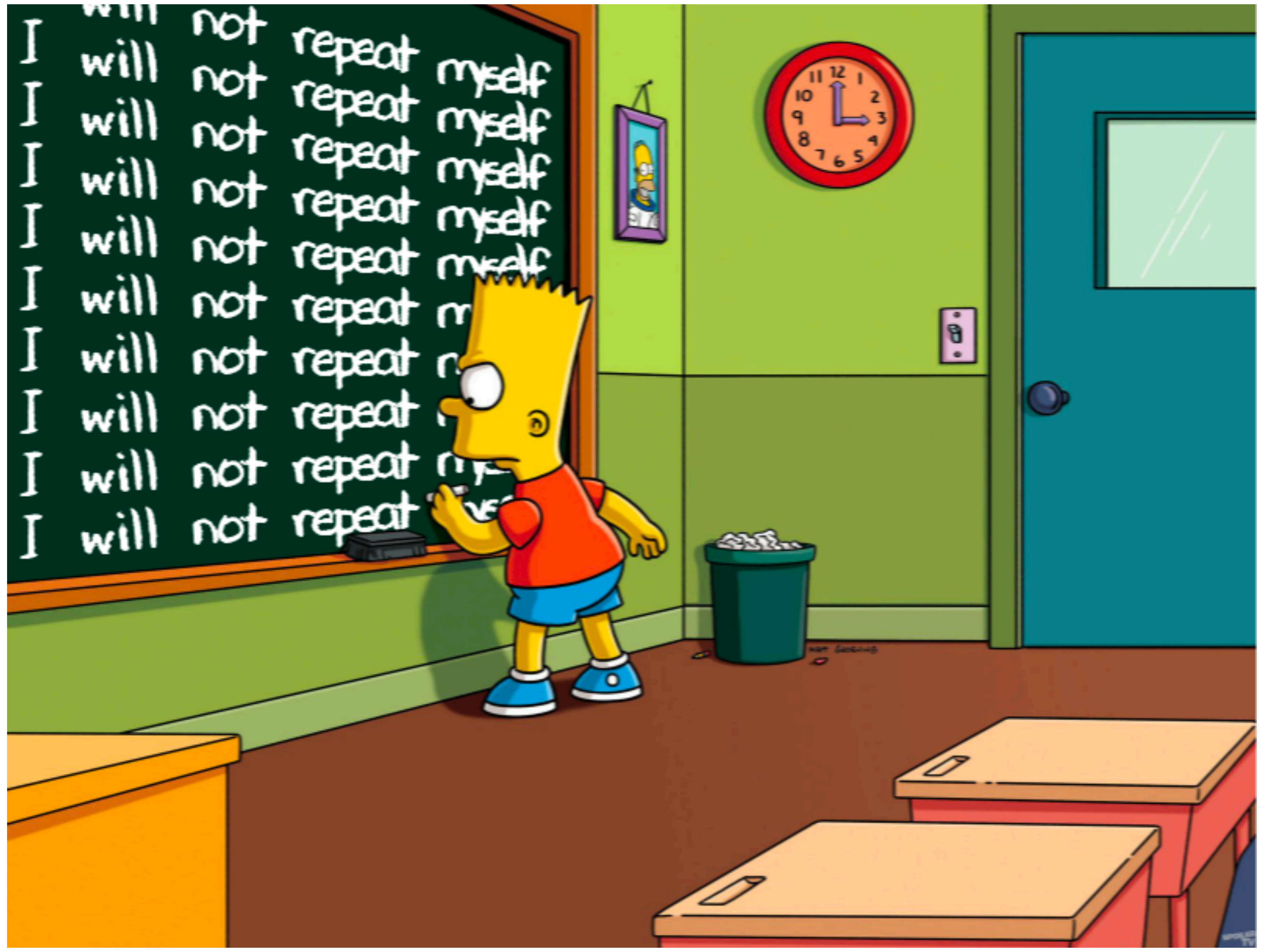
**OOPSLA 2021 / 2020**

**Michael B. James**, Zheng Guo, Ziteng Wang, Shivani Doshi, Hila Peleg,
Ranjit Jhala, Nadia Polikarpova

# Programmers don’t want to repeat code themselves



# Programmers don’t want to repeat code themselves



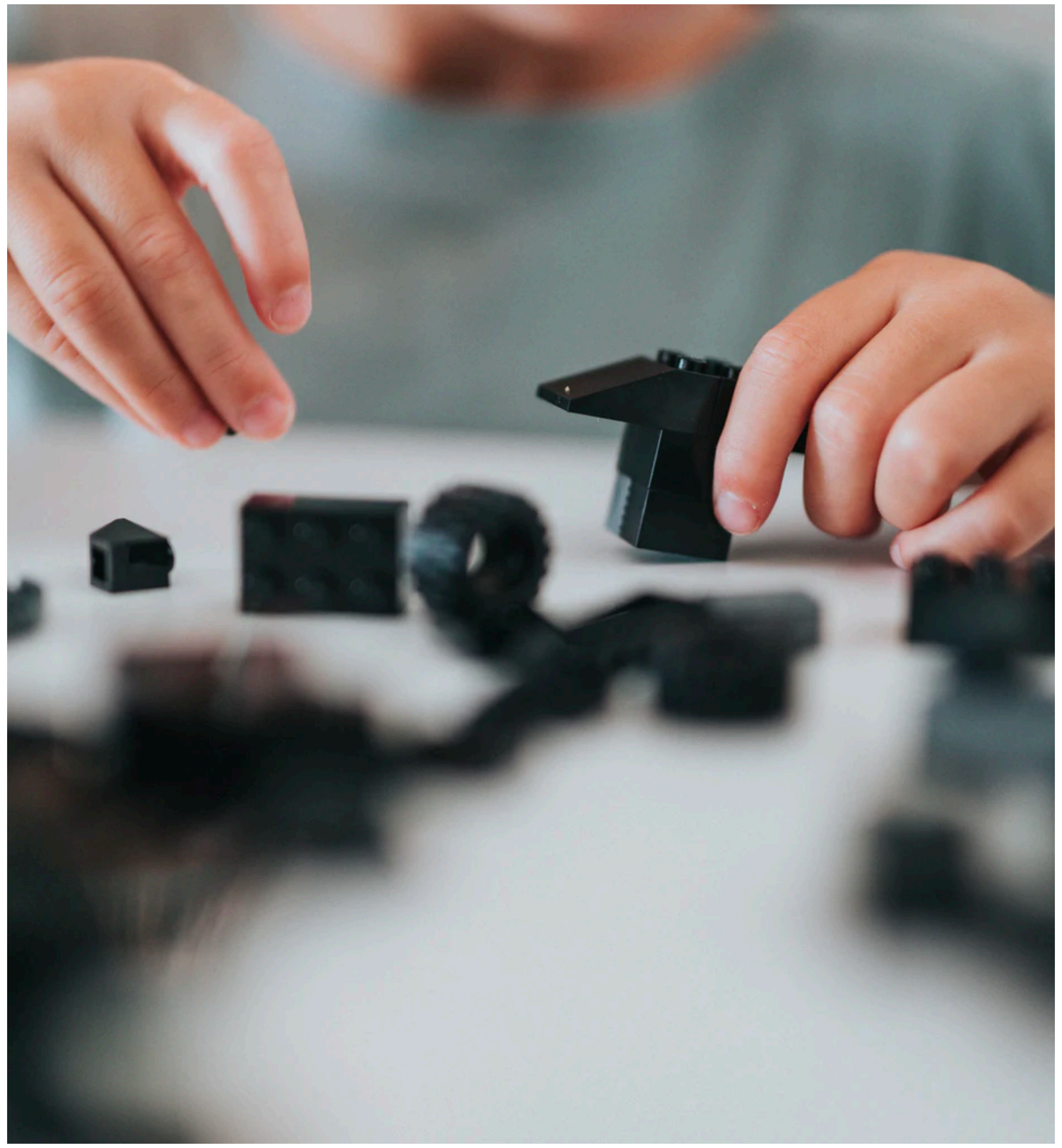
APIs reduce code repetition

# API Discovery Problem

# API Discovery Problem

# API Discovery Problem

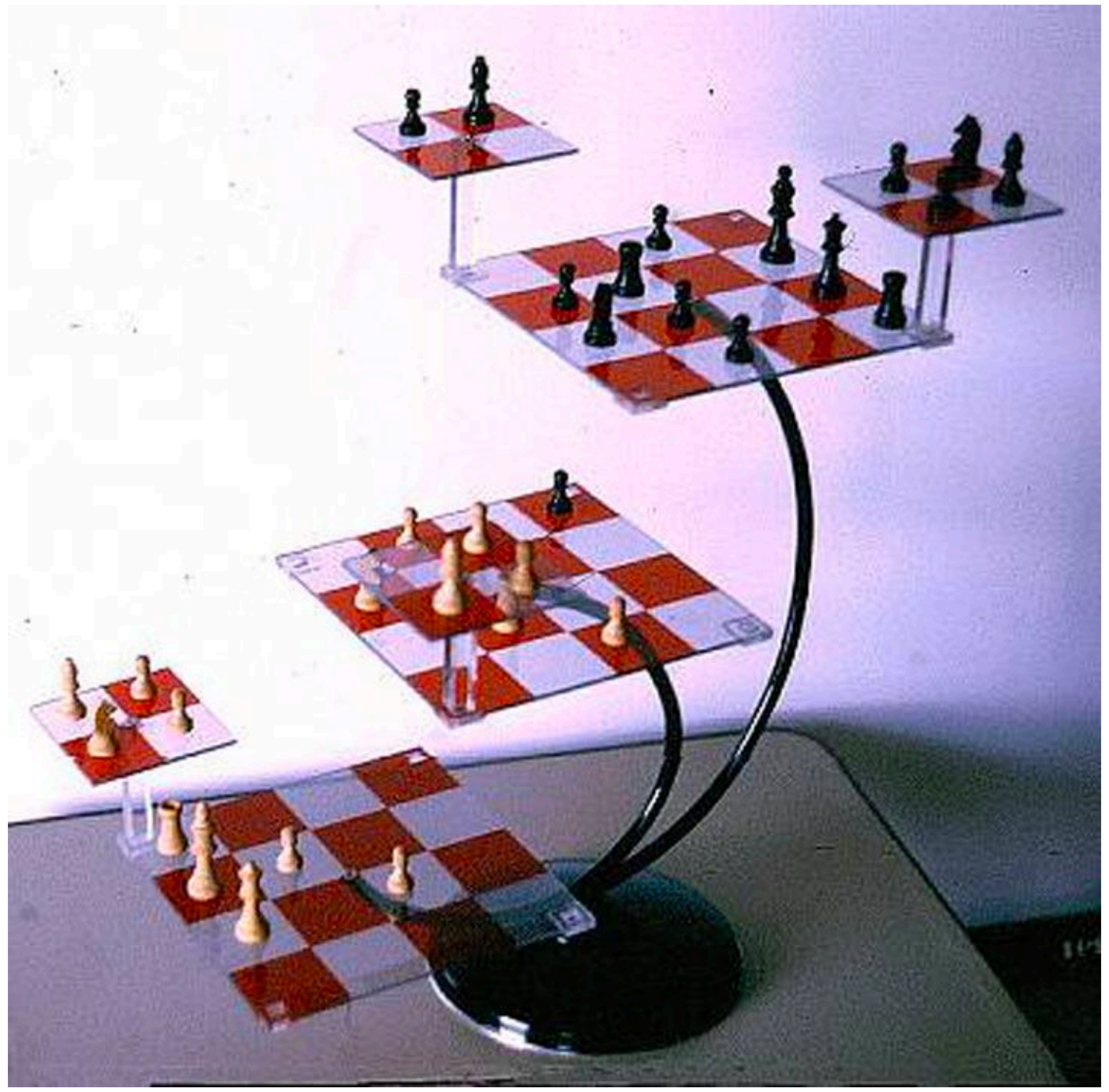
# API Discovery Problem

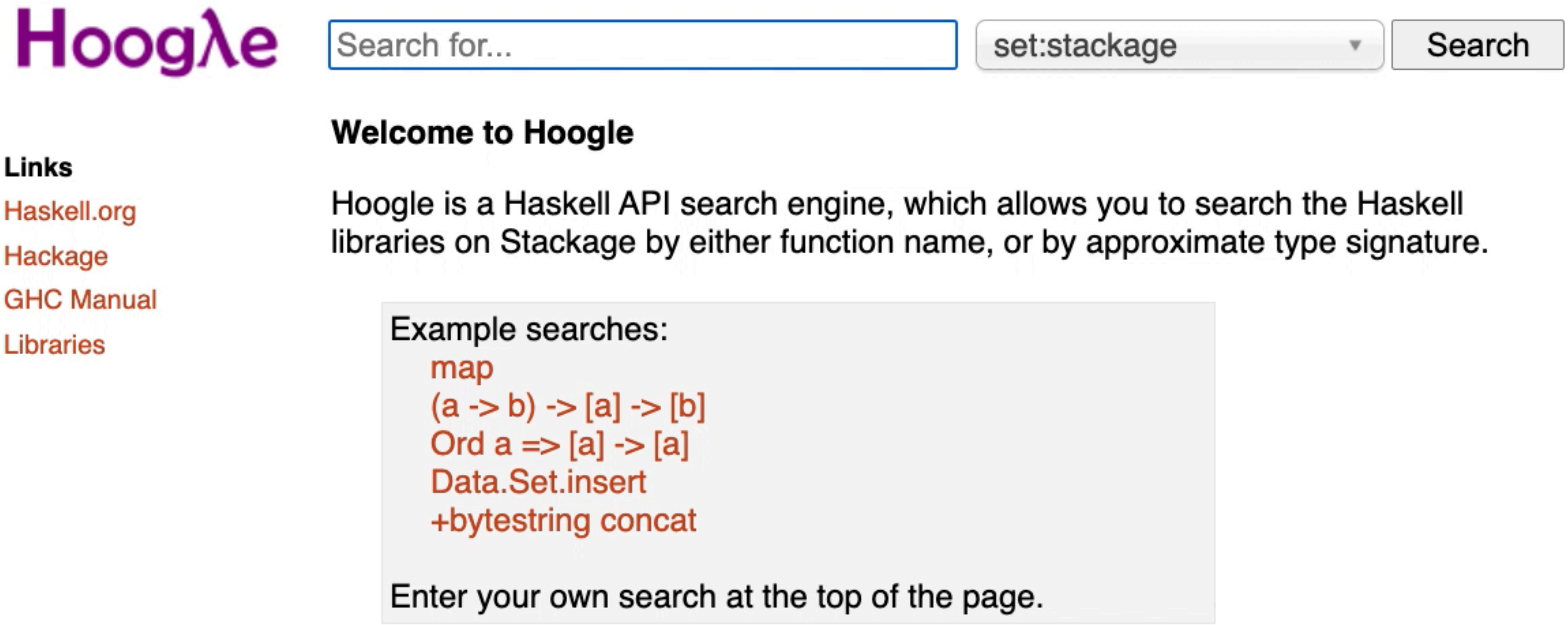


# Haskell makes this harder

# Haskell makes this harder

≈





## Welcome to Hoogle

Hoogle is a Haskell API search engine, which allows you to search the Haskell libraries on Stackage by either function name, or by approximate type signature.

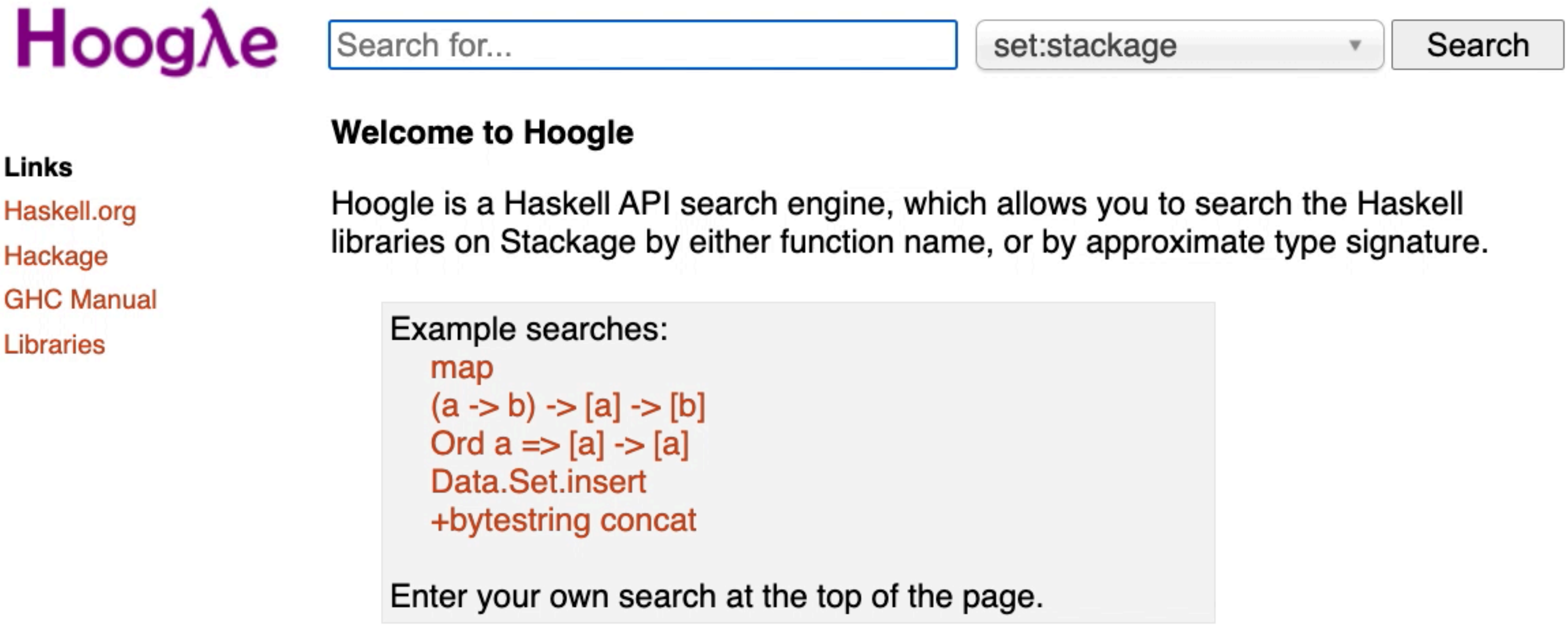
**Links**

- Haskell.org
- Hackage
- GHC Manual
- Libraries

Example searches:

```
map
(a -> b) -> [a] -> [b]
Ord a => [a] -> [a]
Data.Set.insert
+bytestring concat
```

Enter your own search at the top of the page.



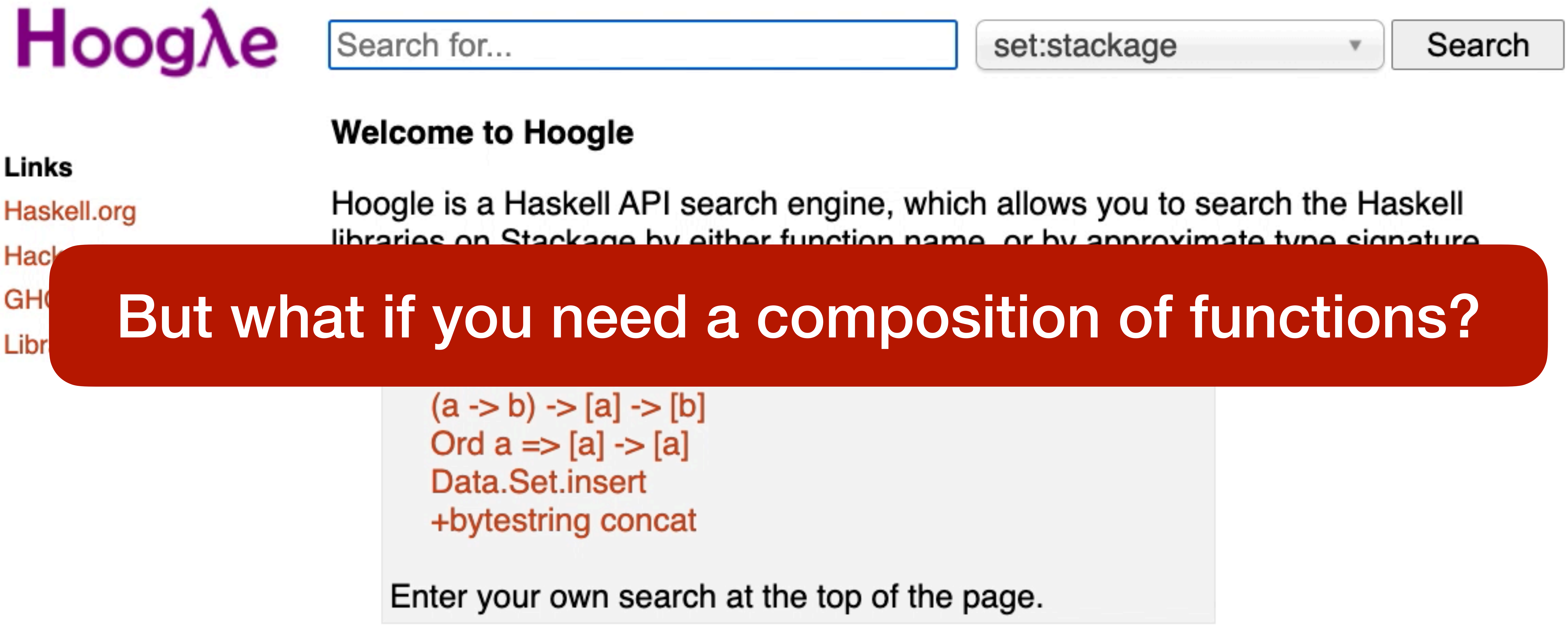
## Welcome to Hoogle

Hoogle is a Haskell API search engine, which allows you to search the Haskell libraries on Stackage by either function name, or by approximate type signature.

Example searches:

```
map
(a -> b) -> [a] -> [b]
Ord a => [a] -> [a]
Data.Set.insert
+bytestring concat
```

Enter your own search at the top of the page.



**Welcome to Hoogle**

**Links**

Haskell.org

Hack

GH

Libr

Hoogle is a Haskell API search engine, which allows you to search the Haskell libraries on Stackage by either function name, or by approximate type signature

(a -> b) -> [a] -> [b]
Ord a => [a] -> [a]
Data.Set.insert
+bytestring concat

Enter your own search at the top of the page.

## But what if you need a composition of functions?

# Running Example

Task: Remove adjacent duplicates

# Running Example

Task: Remove adjacent duplicates

```
dedup [1,2,1,1] = [1,2,1]
```

```
dedup xs = map head (group xs)
```

# Running Example

Task: Remove adjacent duplicates

```
dedup [1,2,1,1] = [1,2,1]


dedup xs = map head (group xs)
         = map head [[1,1], [2], [1]]
         = [1,2,1]
```

# Running Example

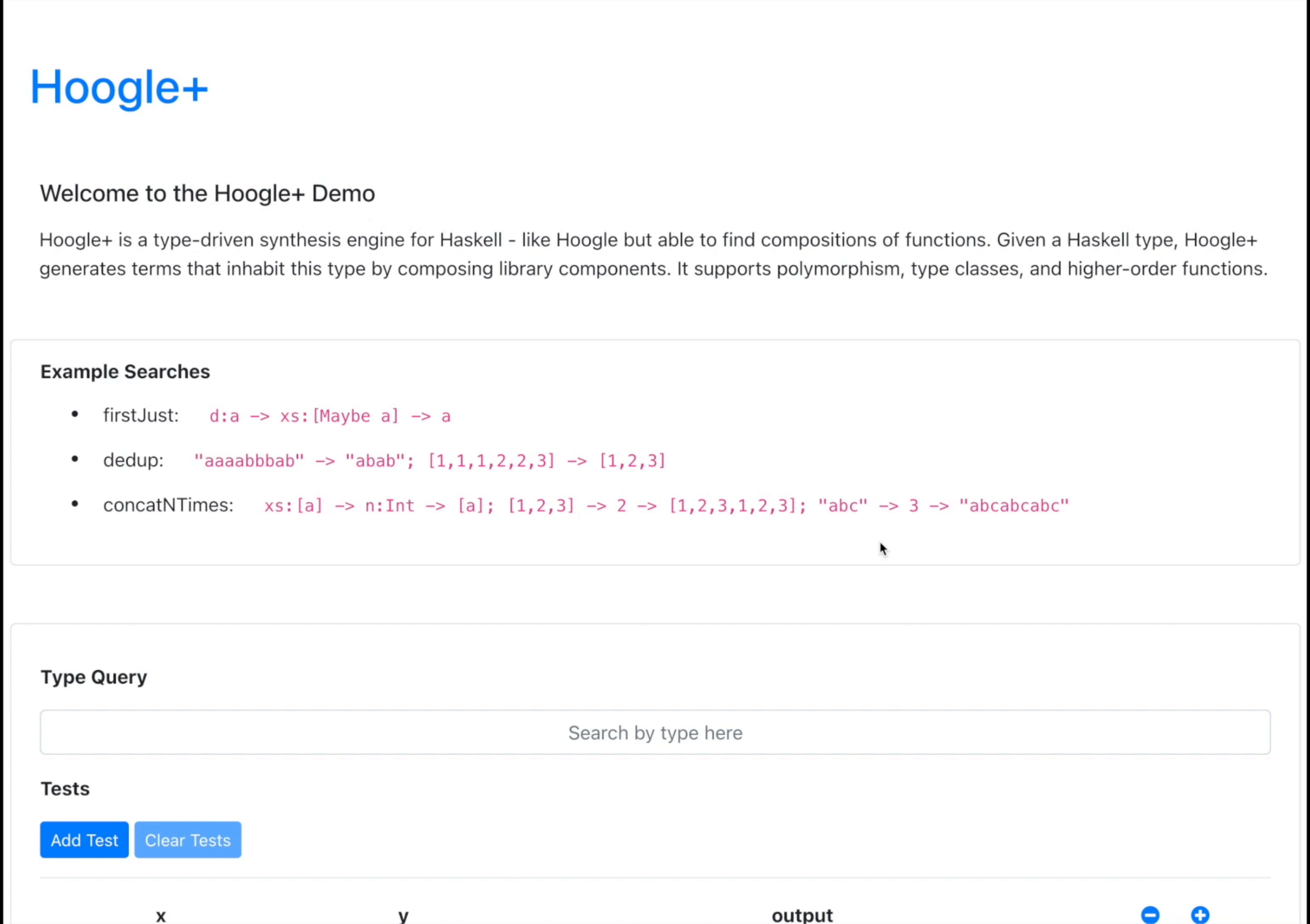
Task: Remove adjacent duplicates

```
dedup [1,2,1,1] = [1,2,1]

dedup :: Eq a => [a] -> [a]
dedup xs = map head (group xs)

         = map head [[1,1], [2], [1]]

         = [1,2,1]
```



# Hoogle+

## Welcome to the Hoogle+ Demo

Hoogle+ is a type-driven synthesis engine for Haskell - like Hoogle but able to find compositions of functions. Given a Haskell type, Hoogle+ generates terms that inhabit this type by composing library components. It supports polymorphism, type classes, and higher-order functions.

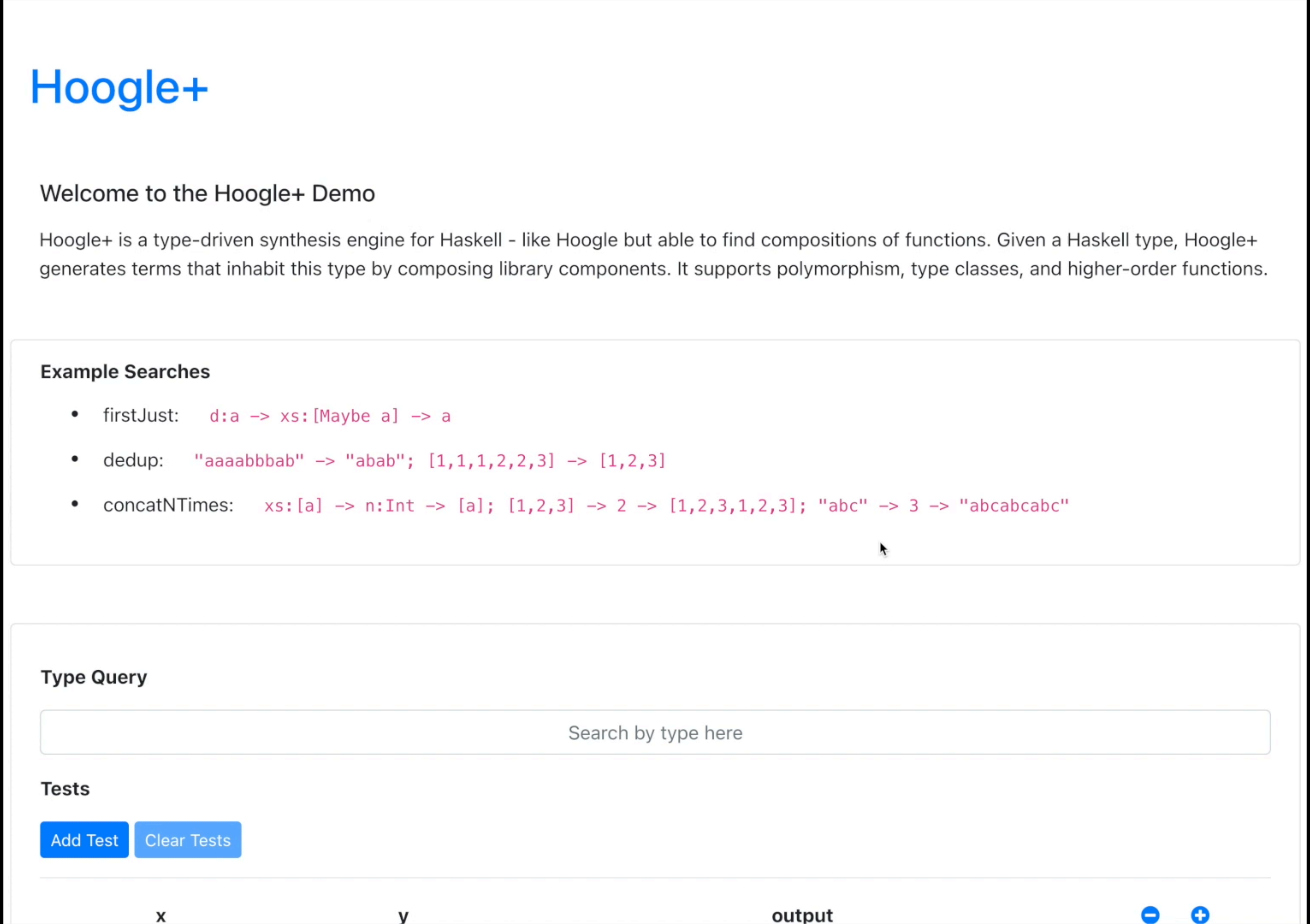
### Example Searches

- firstJust: `d:a -> xs:[Maybe a] -> a`
- dedup: `"aaaabbbab" -> "abab"; [1,1,1,2,2,3] -> [1,2,3]`
- concatNTimes: `xs:[a] -> n:Int -> [a]; [1,2,3] -> 2 -> [1,2,3,1,2,3]; "abc" -> 3 -> "abcabcabc"`

### Type Query

### Tests

| x | y | output | ⊖ ⊕ |
|---|---|---|---|



# Hoogle+

## Welcome to the Hoogle+ Demo

Hoogle+ is a type-driven synthesis engine for Haskell - like Hoogle but able to find compositions of functions. Given a Haskell type, Hoogle+ generates terms that inhabit this type by composing library components. It supports polymorphism, type classes, and higher-order functions.

### Example Searches

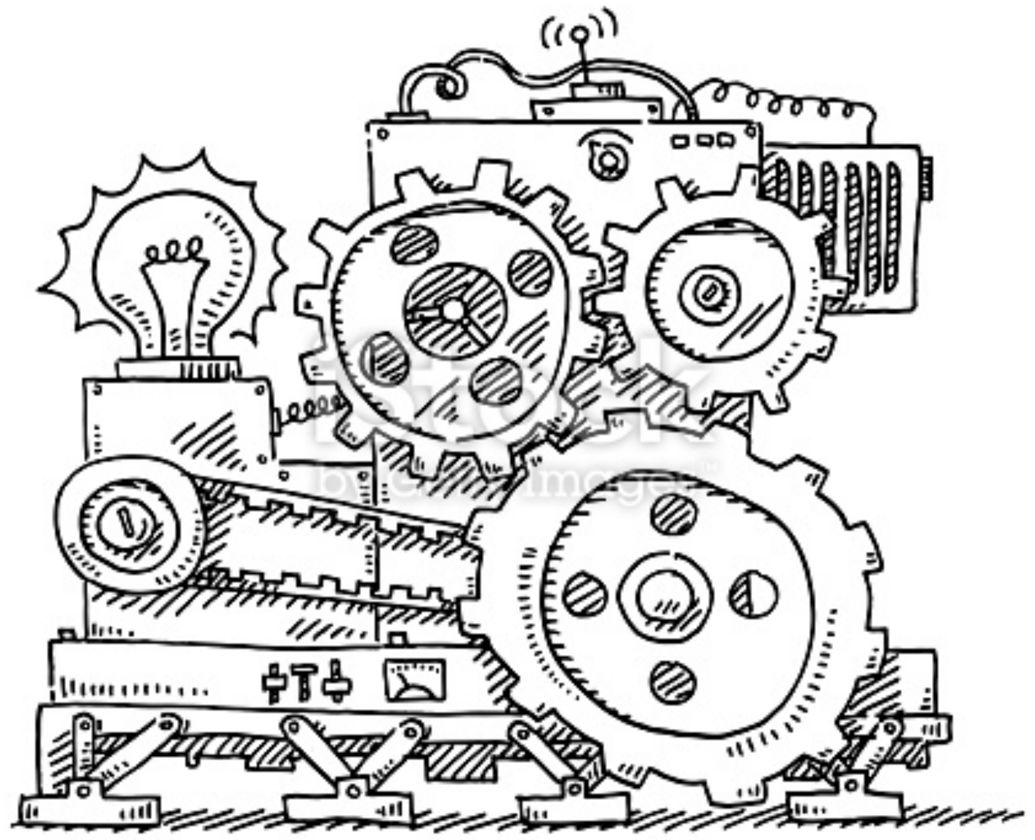
- firstJust: `d:a -> xs:[Maybe a] -> a`
- dedup: `"aaaabbbab" -> "abab"; [1,1,1,2,2,3] -> [1,2,3]`
- concatNTimes: `xs:[a] -> n:Int -> [a]; [1,2,3] -> 2 -> [1,2,3,1,2,3]; "abc" -> 3 -> "abcabcabc"`

### Type Query

### Tests

| x | y | output |
|---|---|---|

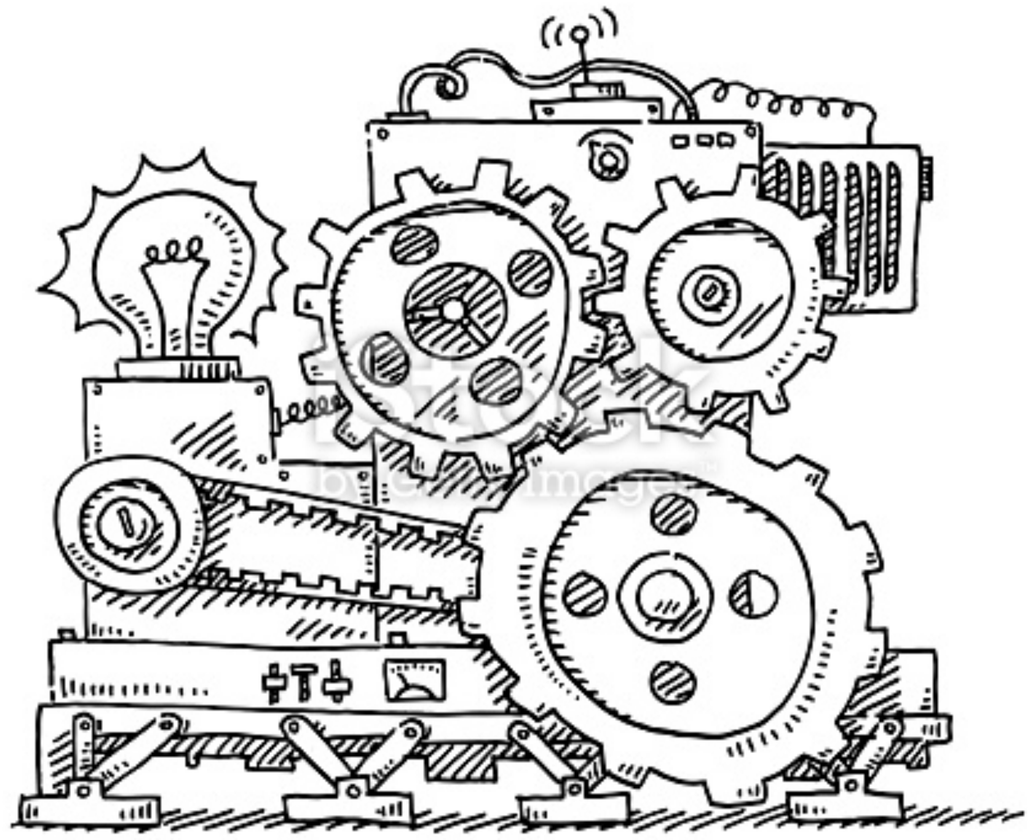
# Core Engine



**Program Synthesis by**
**Type-Guided Abstraction Refinement**
**[Guo et al. 2020]**

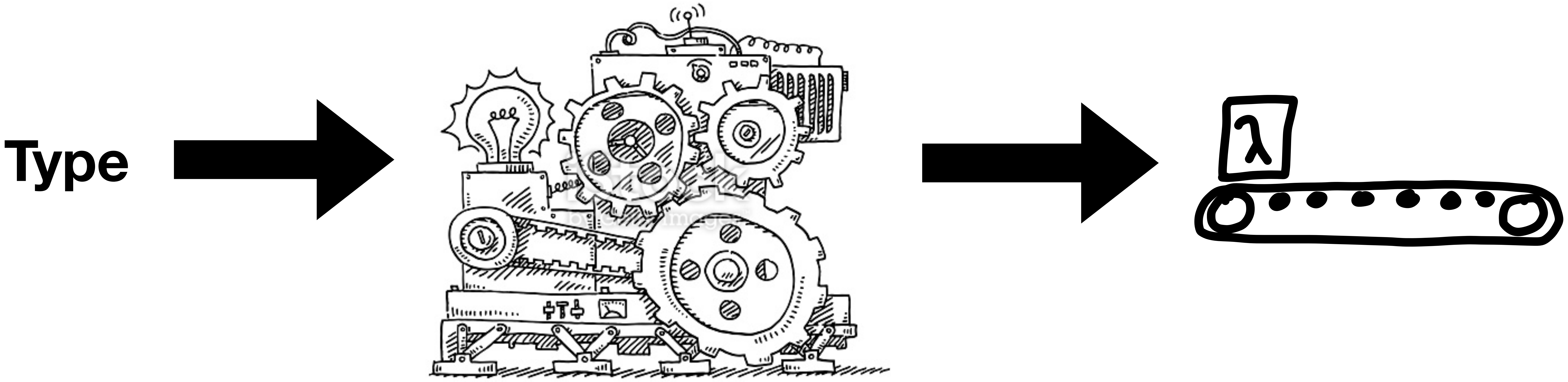
# Core Engine

Type



**Program Synthesis by**
**Type-Guided Abstraction Refinement**
**[Guo et al. 2020]**

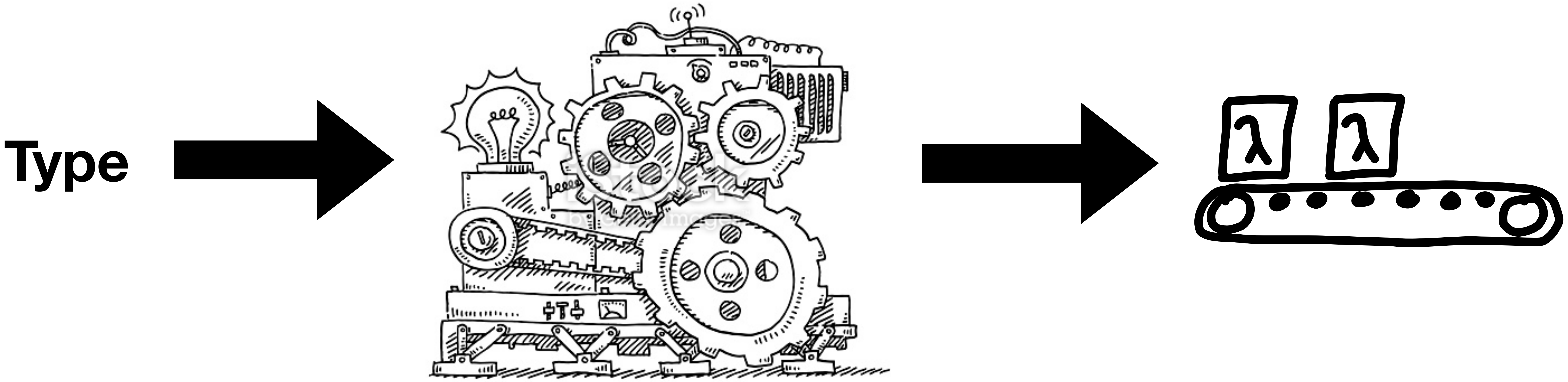
# Core Engine



Type

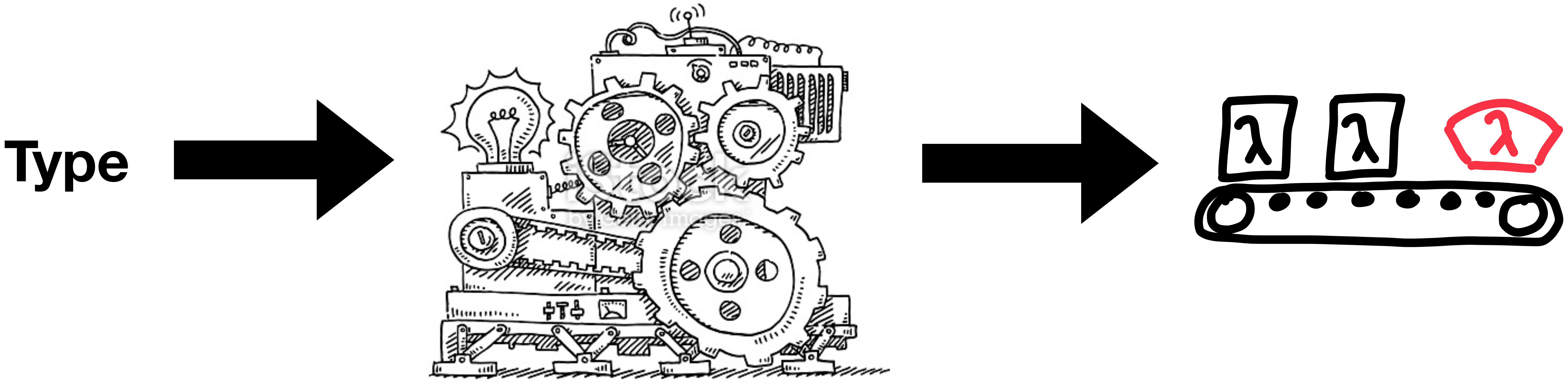
**Program Synthesis by**
**Type-Guided Abstraction Refinement**
**[Guo et al. 2020]**

# Core Engine



**Program Synthesis by**
**Type-Guided Abstraction Refinement**
**[Guo et al. 2020]**

# Core Engine



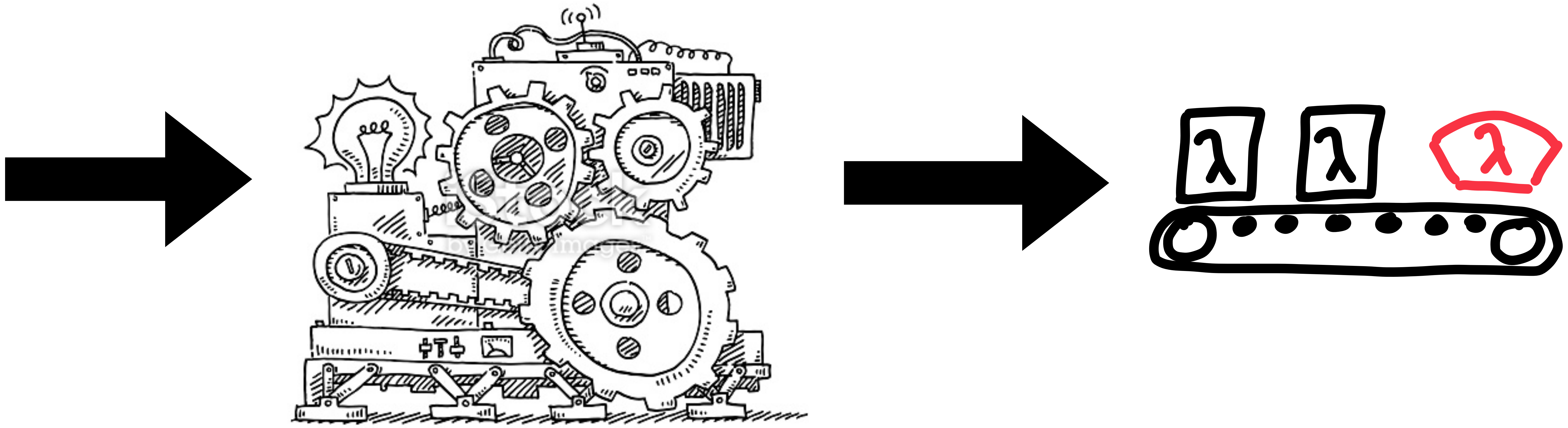
Type

Program Synthesis by
Type-Guided Abstraction Refinement
[Guo et al. 2020]

# Core Engine

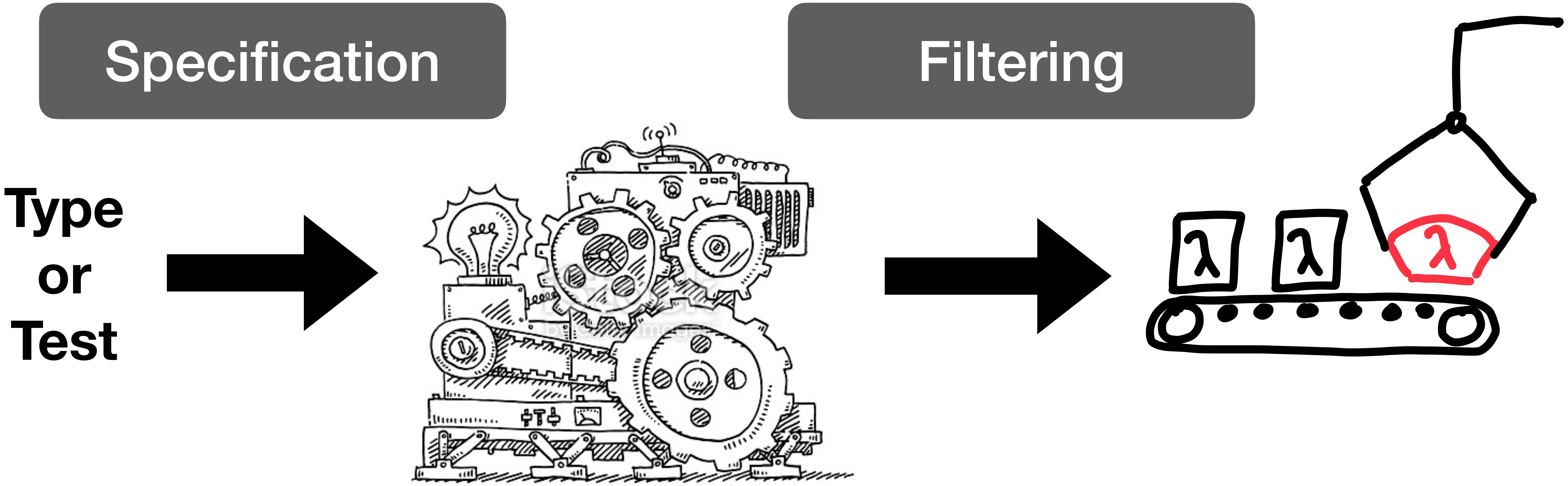
Specification

Type
or
Test



Program Synthesis by
Type-Guided Abstraction Refinement
[Guo et al. 2020]

# Core Engine

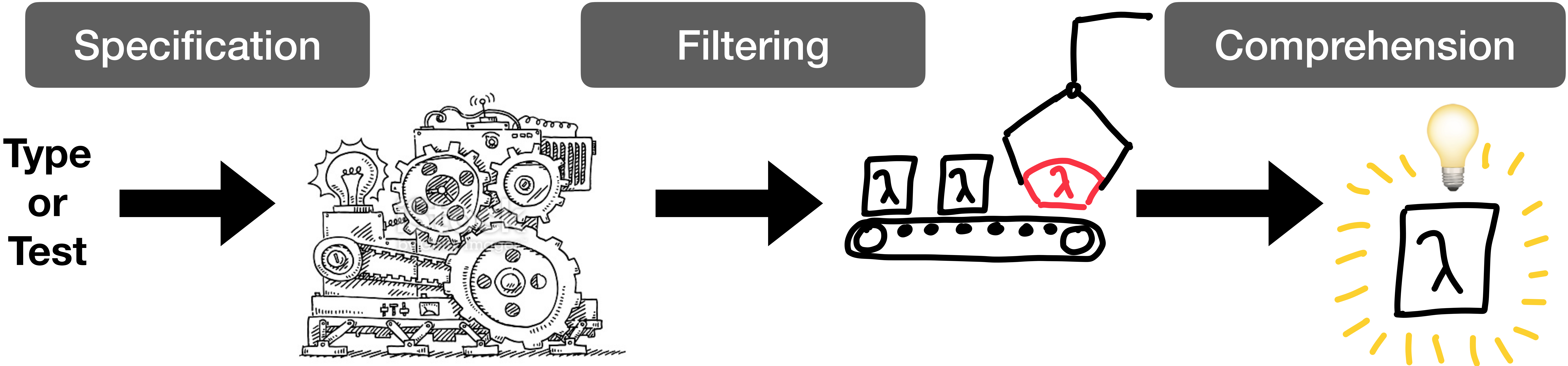


Specification

Type
or
Test

Filtering

**Program Synthesis by**
**Type-Guided Abstraction Refinement**
**[Guo et al. 2020]**



Specification

Filtering

Comprehension

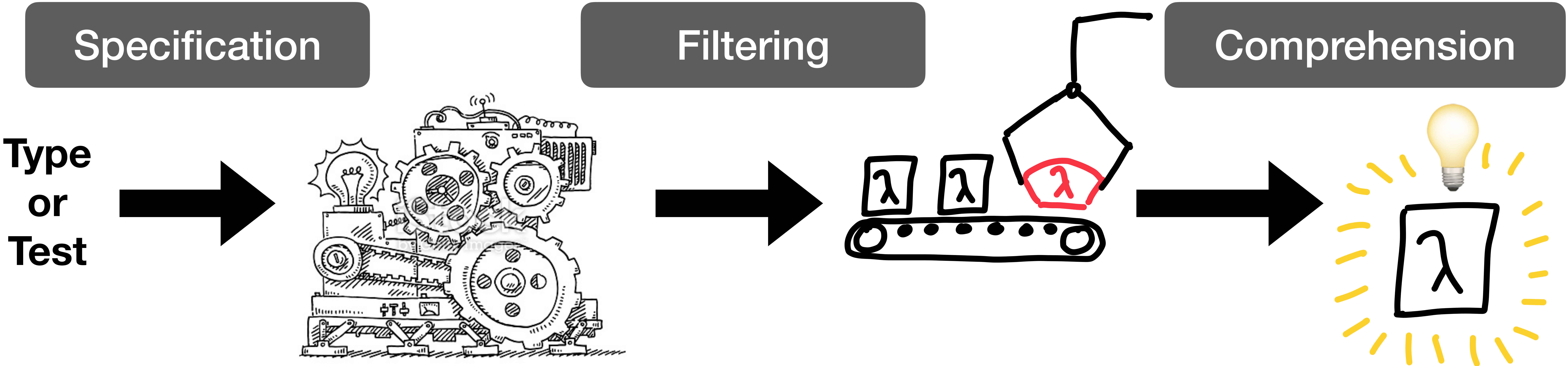
Type

or

Test

Program Synthesis by
Type-Guided Abstraction Refinement
[Guo et al. 2020]

# Hoogle+



Specification

Filtering

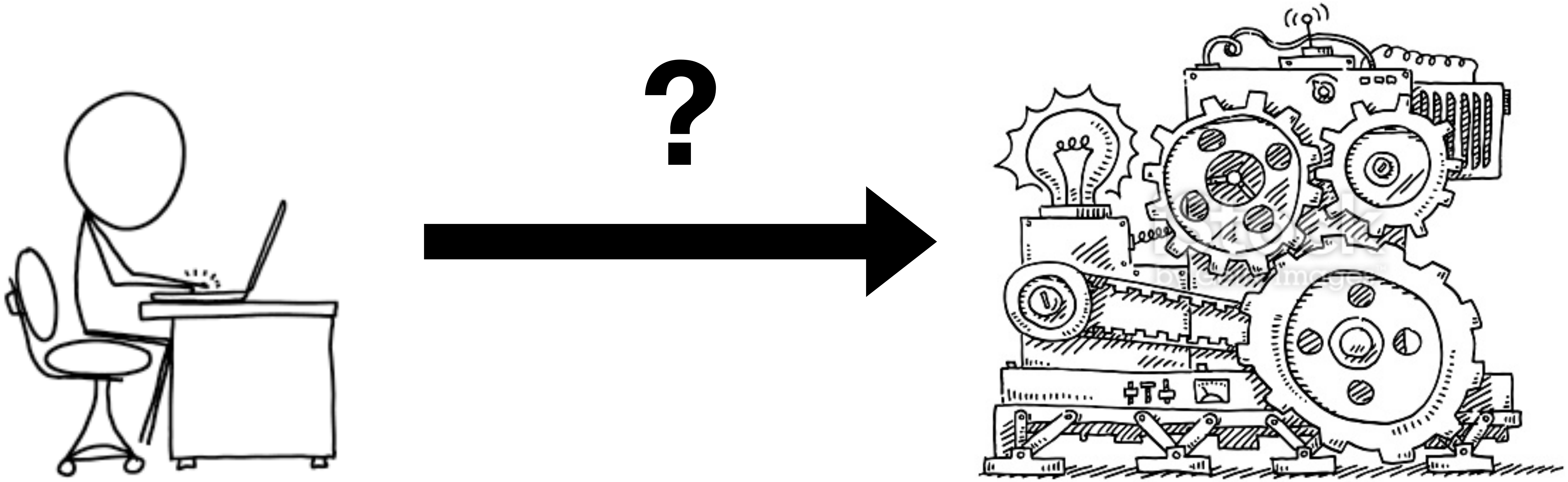
Comprehension

Type
or
Test

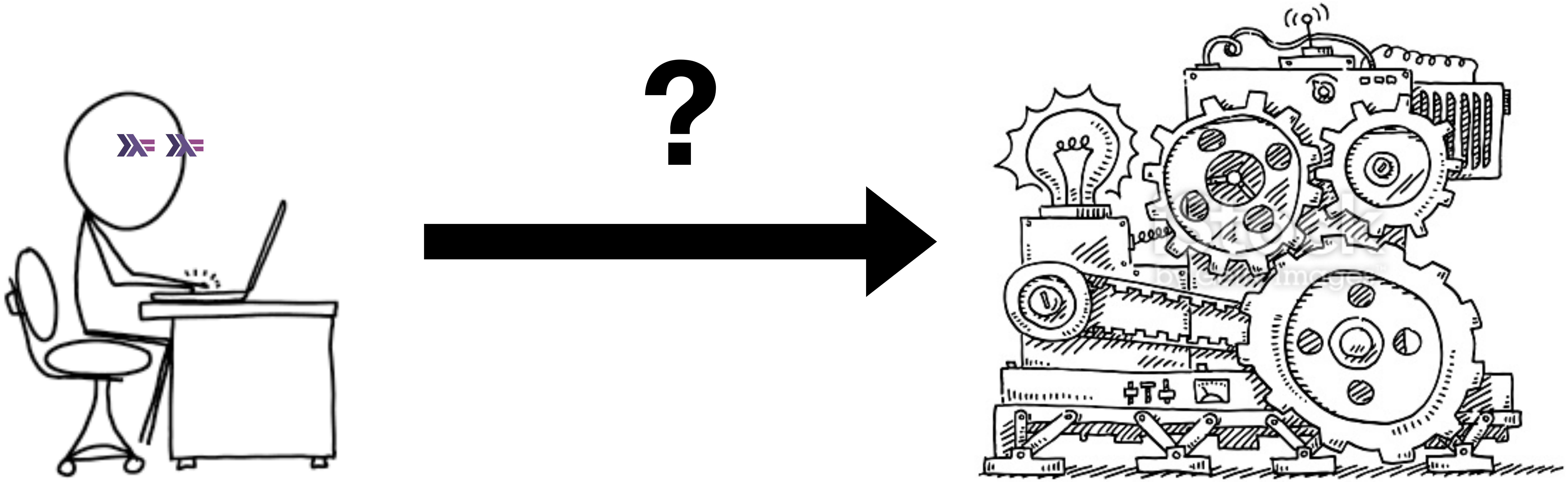
Program Synthesis by
Type-Guided Abstraction Refinement
[Guo et al. 2020]

Hoogle+ User Study

# Specification

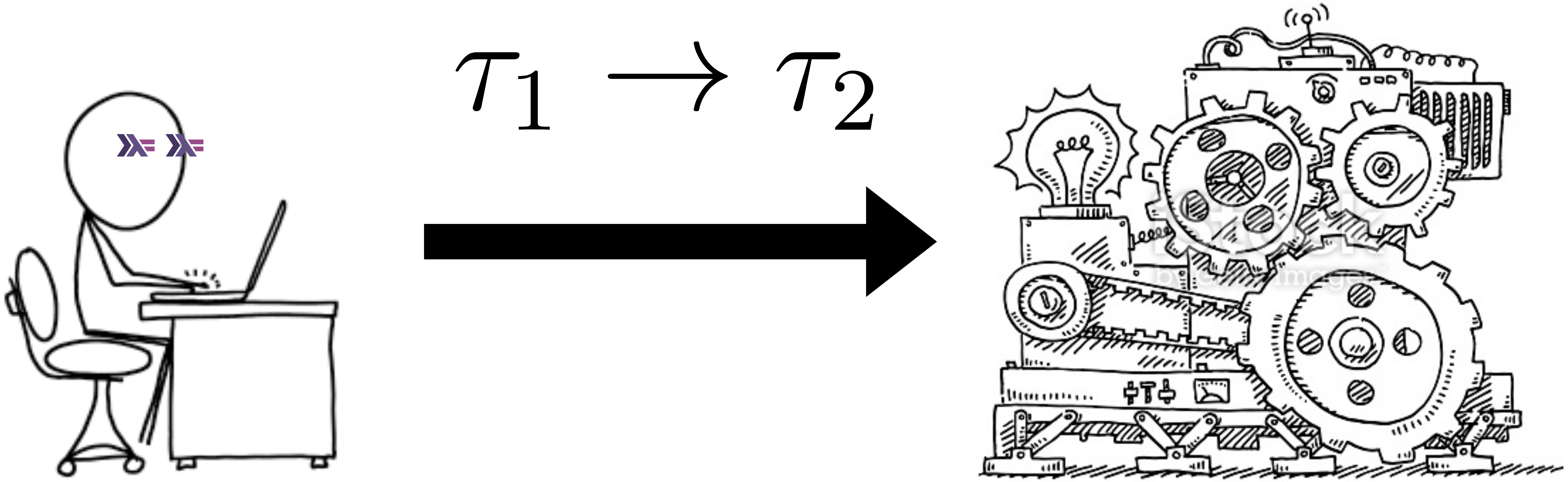


# Specification



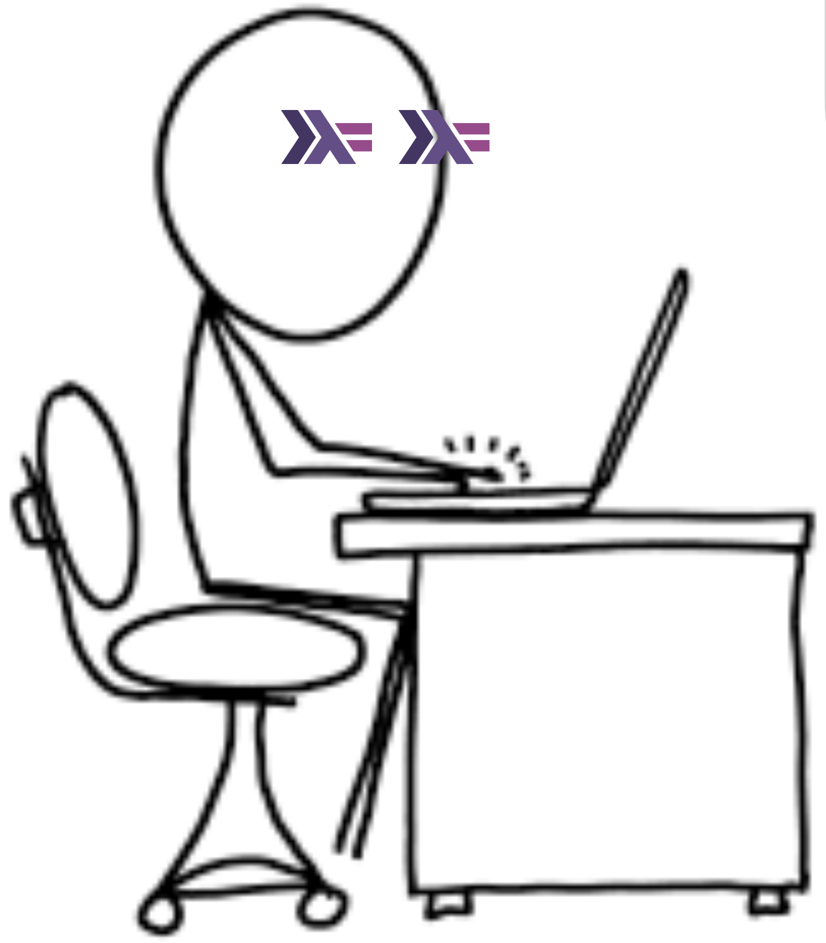
?

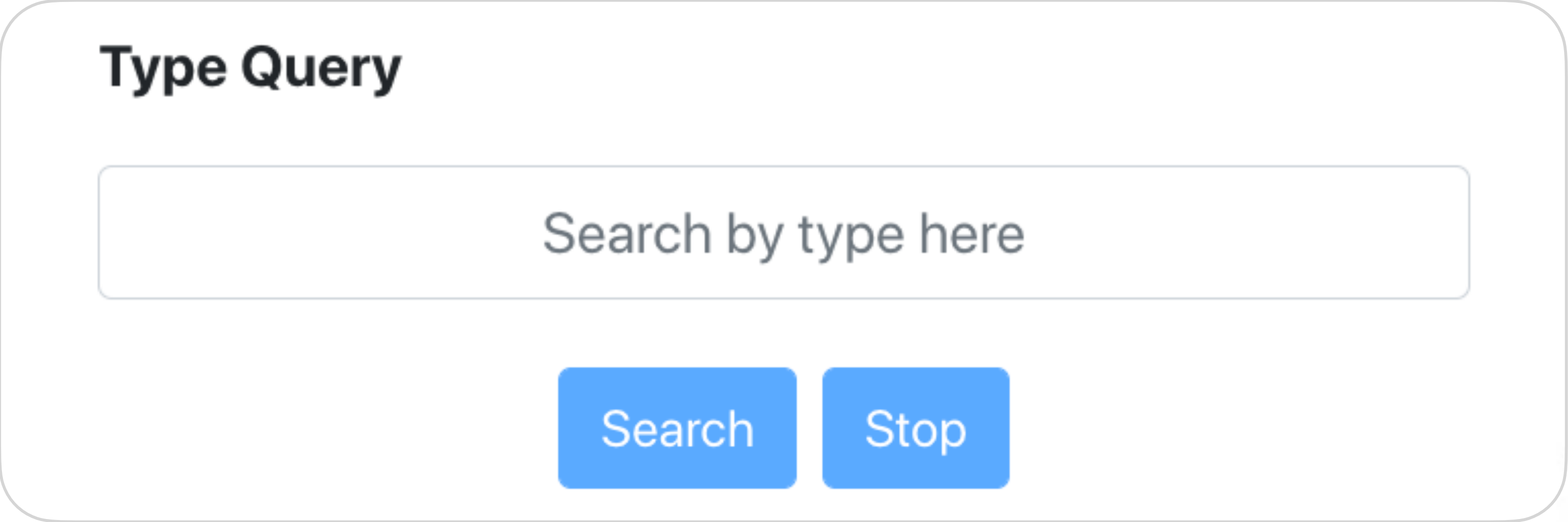
# Specification



$\tau_1 \to \tau_2$

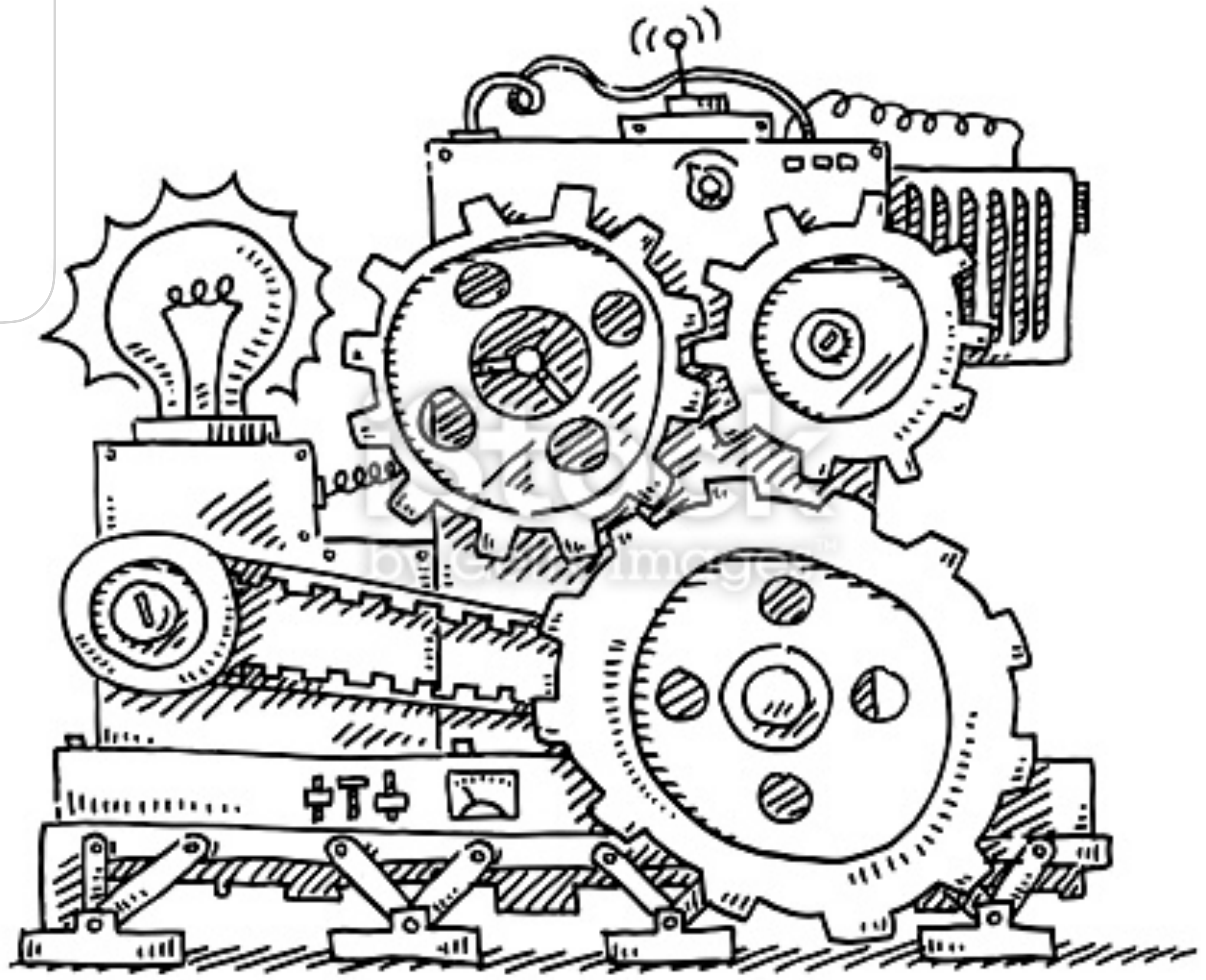
# Specification



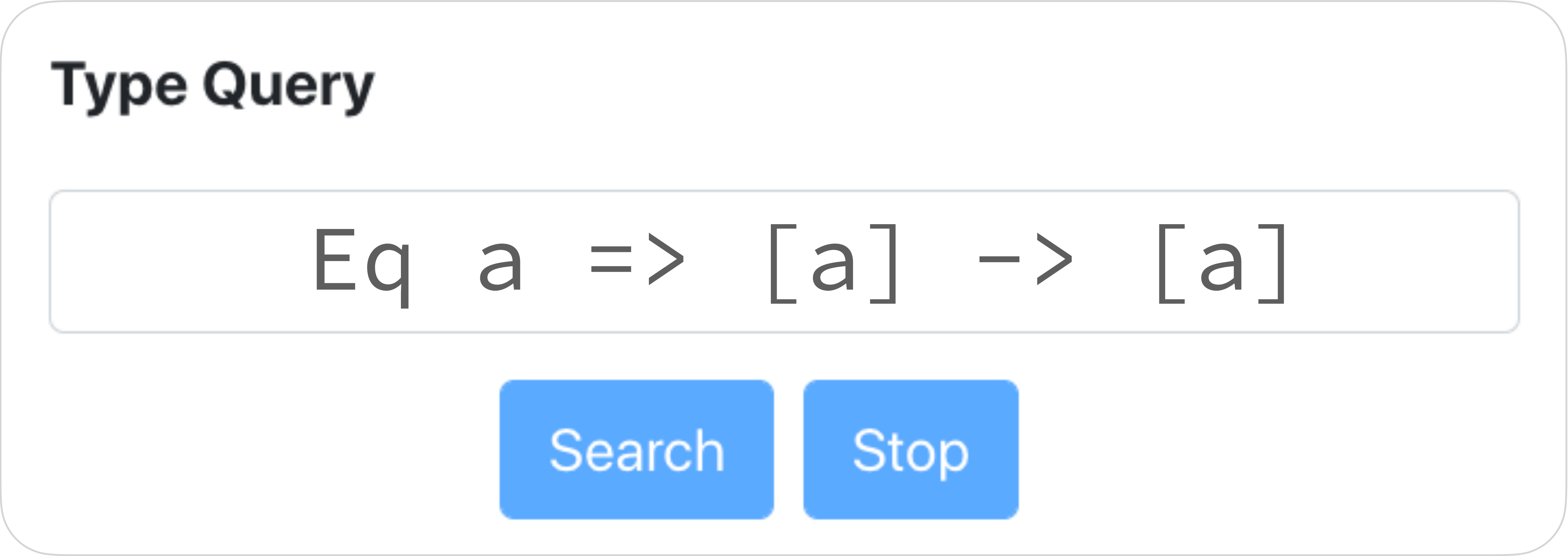


Type Query

Search by type here



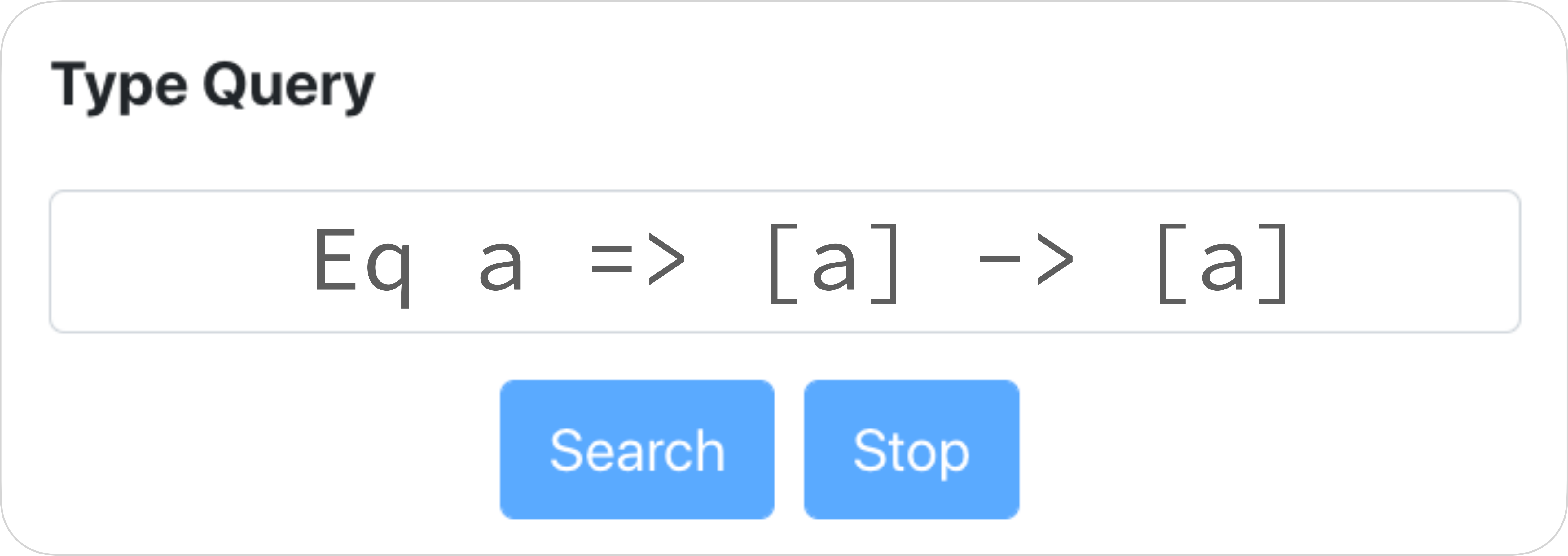
# Specifying dedup



```
Eq a => [a] -> [a]
```

```
dedup xs = map head (group xs)
```

# Specifying dedup

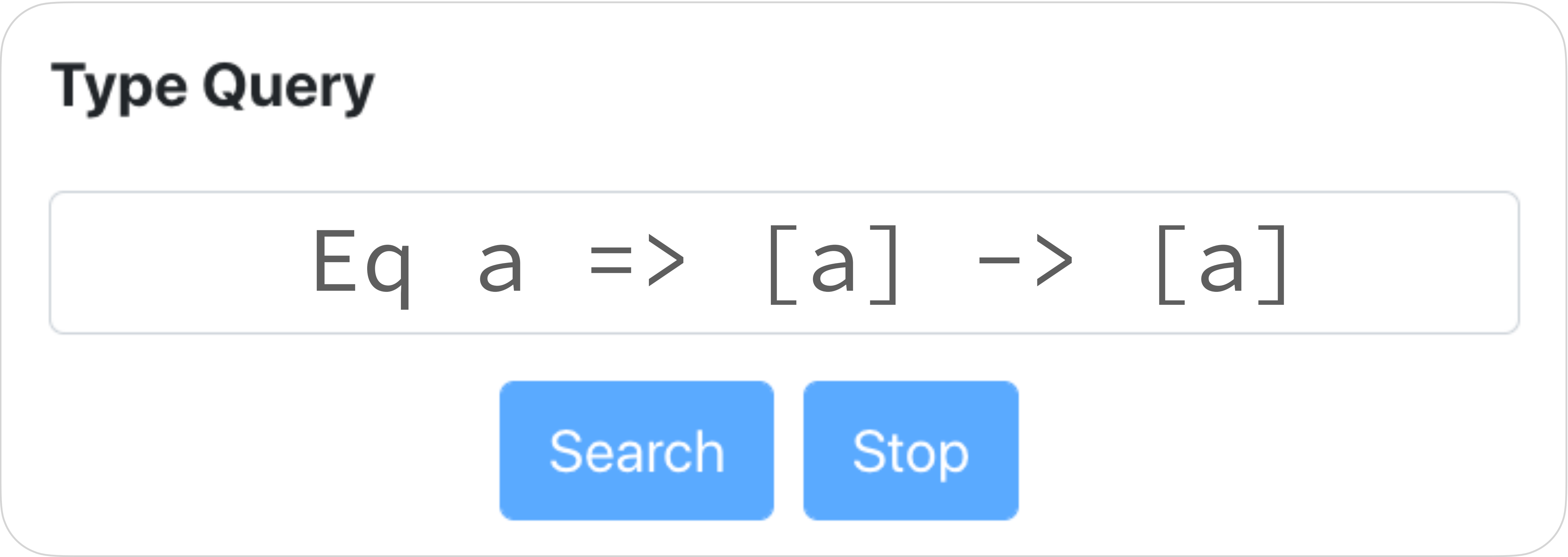

**Type Query**

```
Eq a => [a] -> [a]
```

```
dedup xs = map head (group xs)
```

```
dedup [1,2,1,1] = [1,2,1]
```

# Specifying dedup



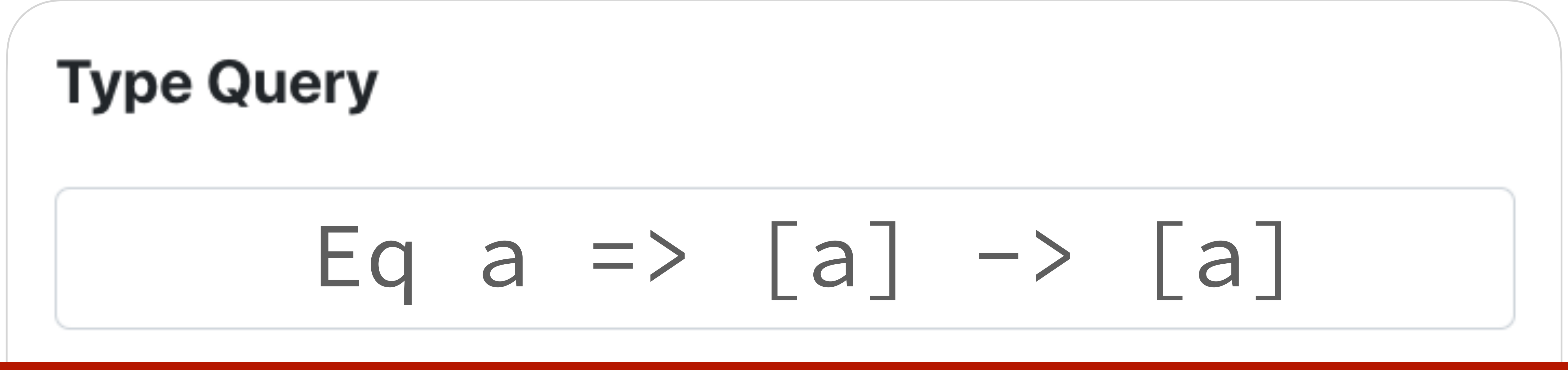
Type Query

```
Eq a => [a] -> [a]
```

```
dedup xs = map head (group xs)

dedup [1,2,1,1] = [1,2,1]
dedup “OOPSLA2020” = “OPSLA2020”
```

# Specifying dedup



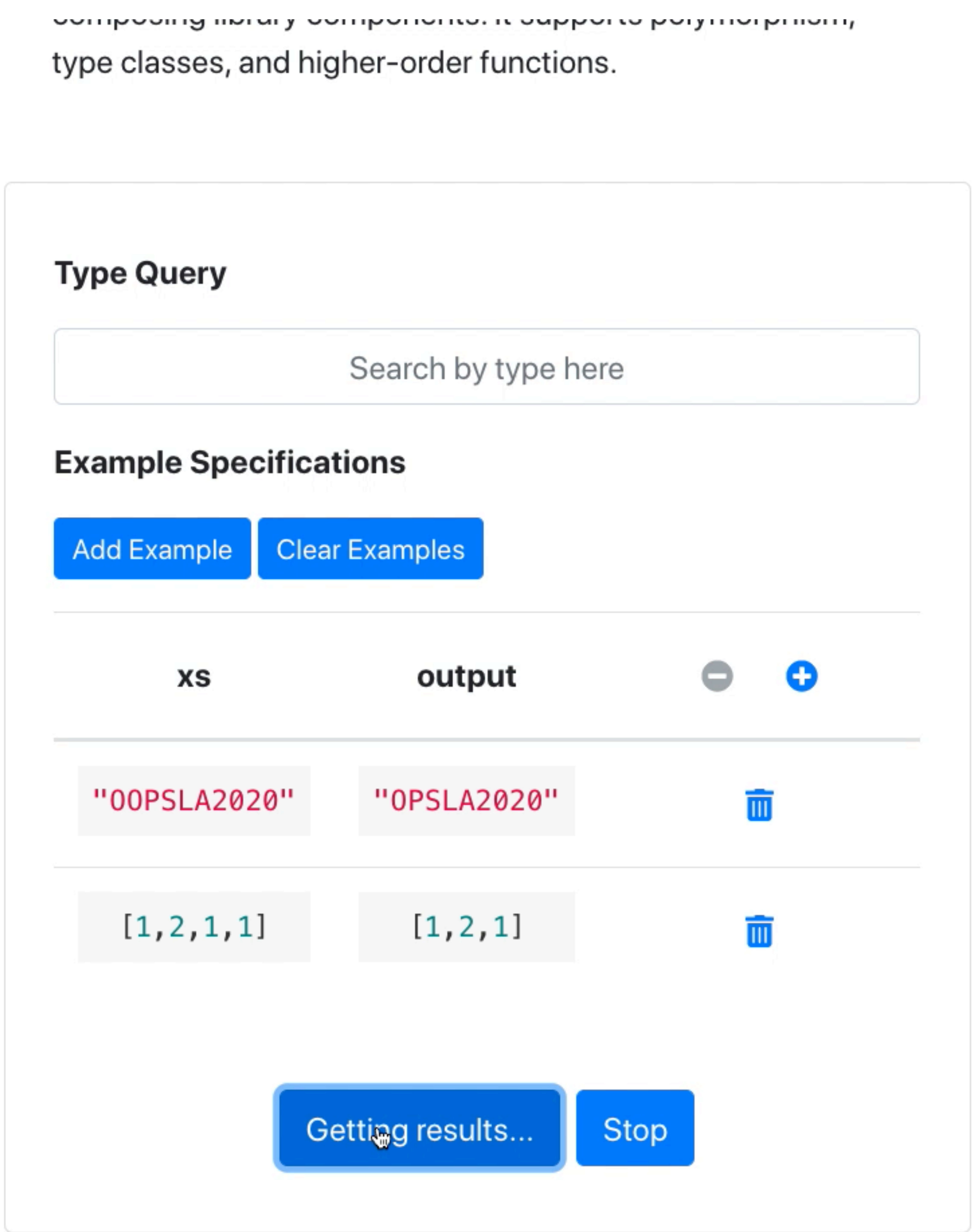
Type Query

```
Eq a => [a] -> [a]
```

Challenge: How to infer likely type specifications from tests?

```
dedup [1,2,1,1] = [1,2,1]
dedup “OOPSLA2020” = “OPSLA2020”
```

# Challenge: How to infer likely type specifications



composing library components. It supports polymorphism, type classes, and higher-order functions.

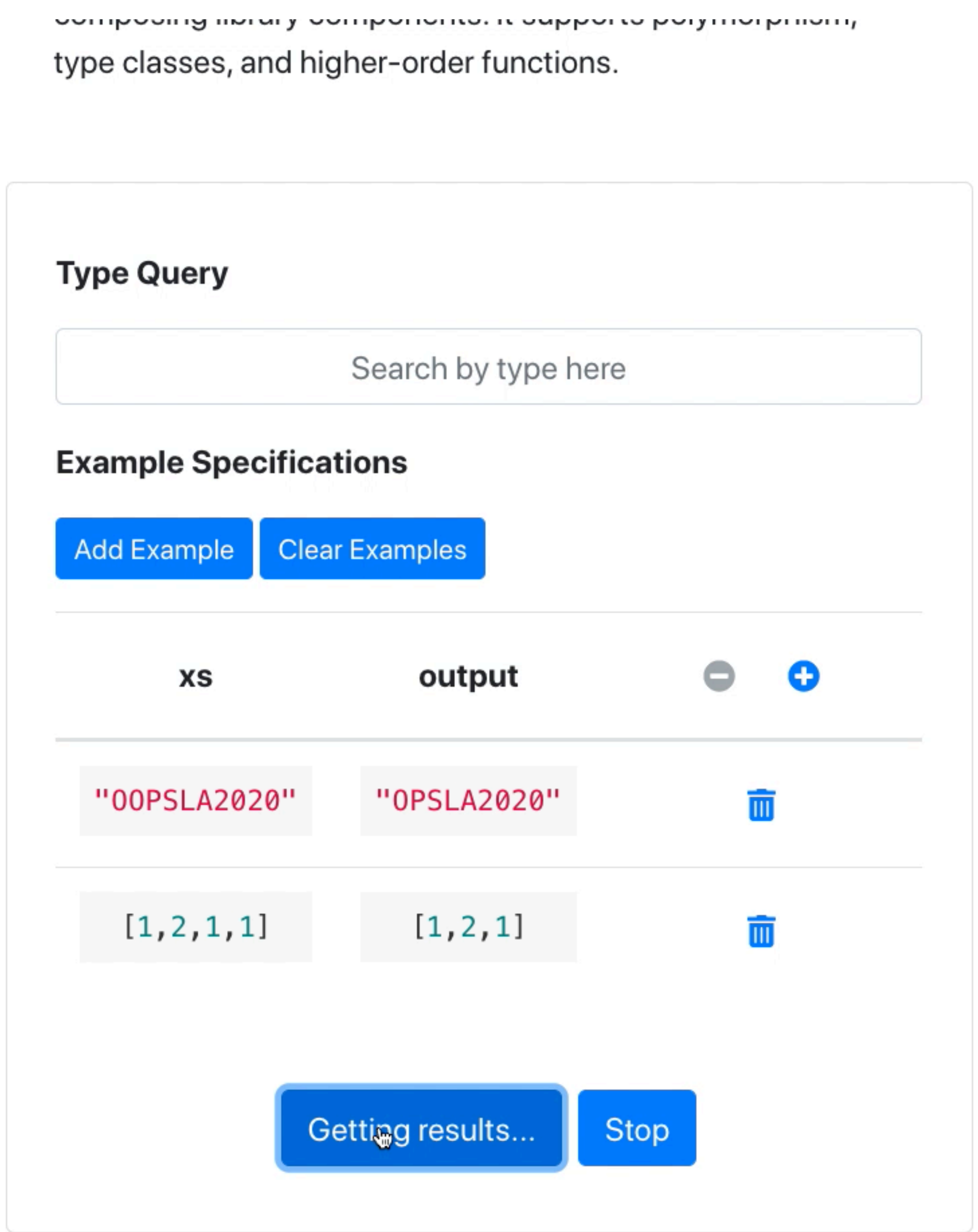
**Type Query**

Search by type here

**Example Specifications**

| xs | output |
|---|---|
| "OOPSLA2020" | "OPSLA2020" |
| [1,2,1,1] | [1,2,1] |

# Challenge: How to infer likely type specifications



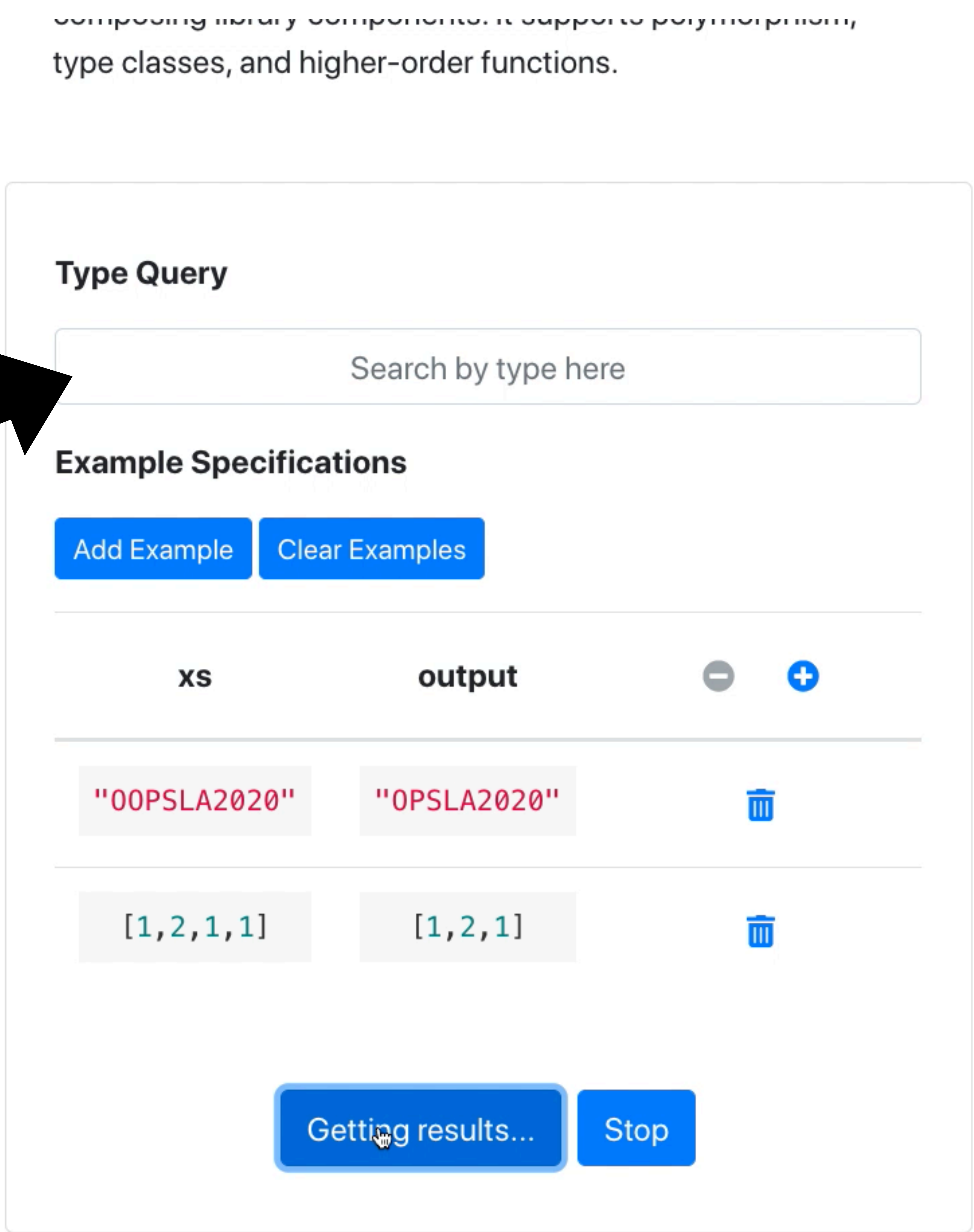
composing library components. It supports polymorphism, type classes, and higher-order functions.

**Type Query**

**Example Specifications**

| xs | output |
|---|---|
| "OOPSLA2020" | "OPSLA2020" |
| [1,2,1,1] | [1,2,1] |

Challenge: How to infer likely type specifications



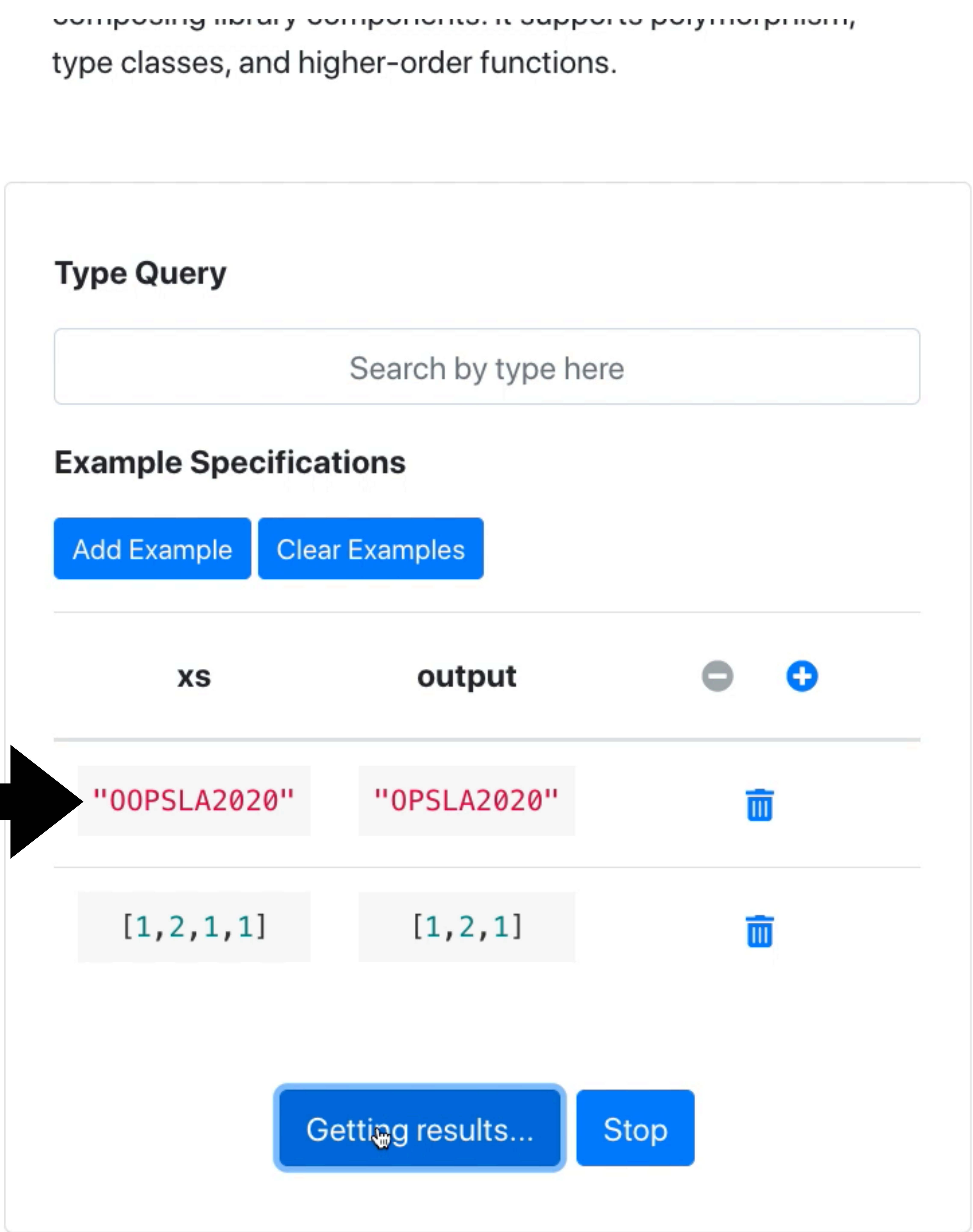
composing library components. It supports polymorphism, type classes, and higher-order functions.

Type Query

Example Specifications

| xs | output |
|---|---|
| "OOPSLA2020" | "OPSLA2020" |
| [1,2,1,1] | [1,2,1] |

Challenge: How to infer likely type specifications



composing library components. It supports polymorphism,
type classes, and higher-order functions.

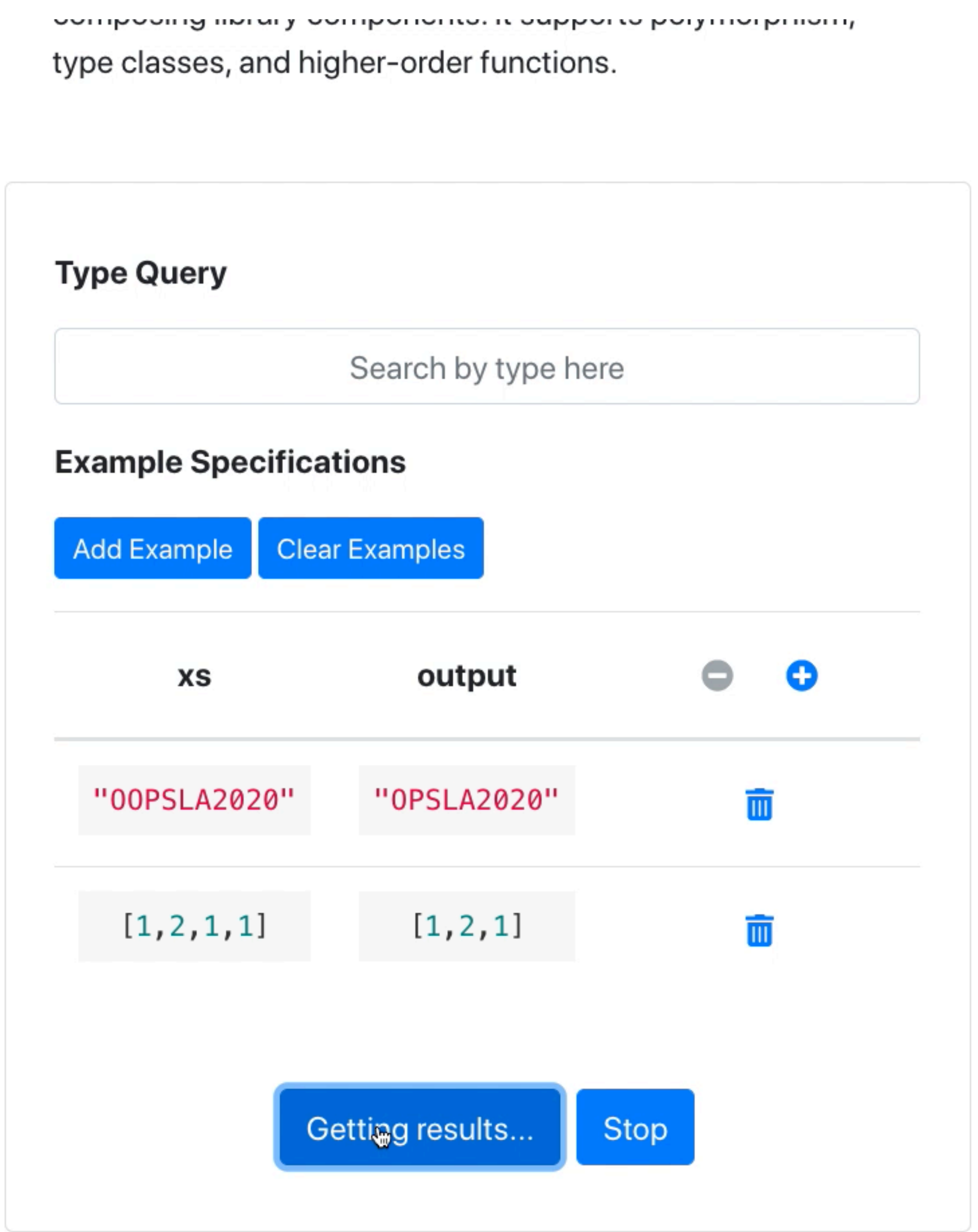
**Type Query**

Search by type here

**Example Specifications**

| xs | output |
|---|---|
| "OOPSLA2020" | "OPSLA2020" |
| [1,2,1,1] | [1,2,1] |

# Challenge: How to infer likely type specifications



composing library components. It supports polymorphism, type classes, and higher-order functions.

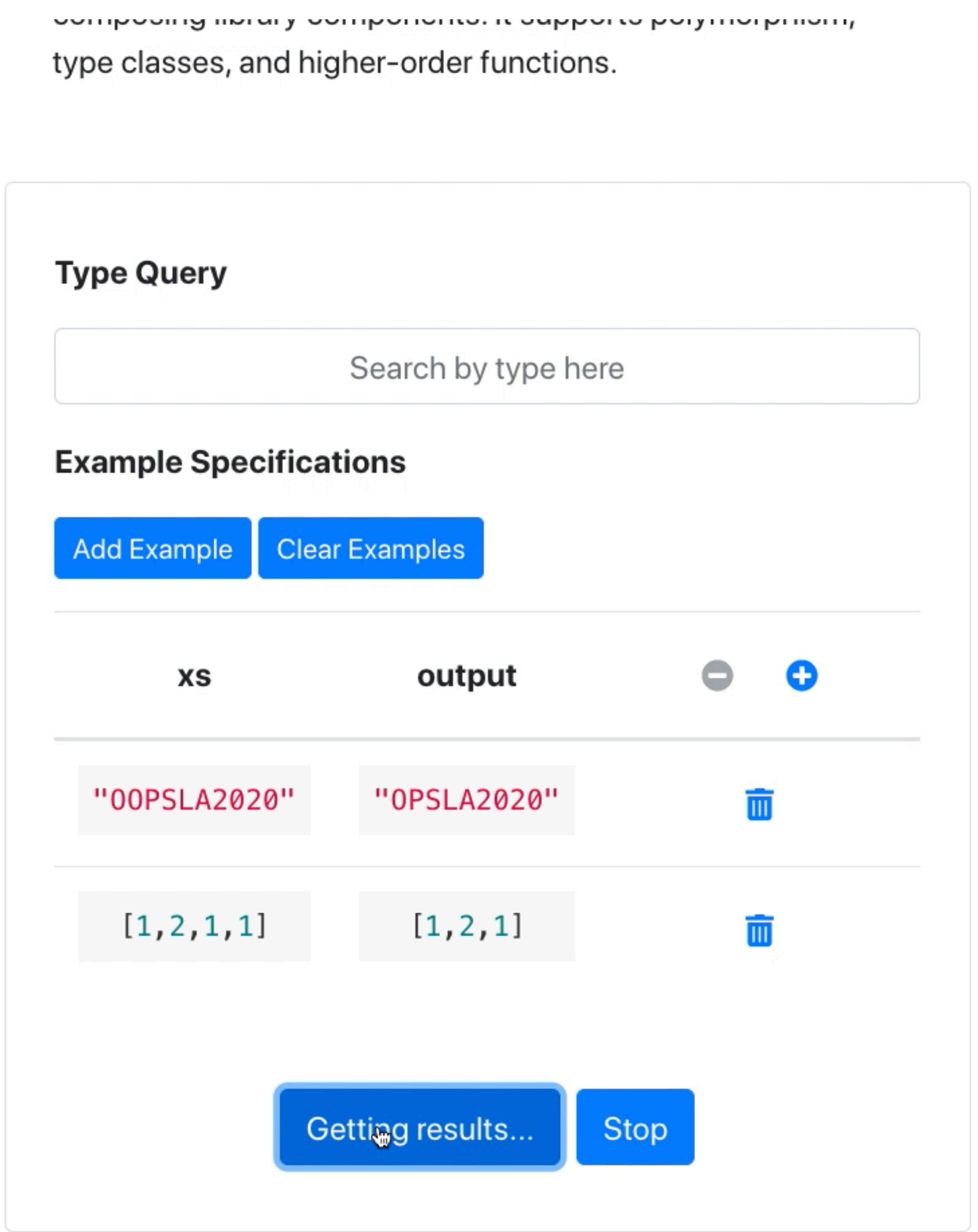
**Type Query**

Search by type here

**Example Specifications**

| xs | output |
|---|---|
| "OOPSLA2020" | "OPSLA2020" |
| [1,2,1,1] | [1,2,1] |

# Challenge: How to infer likely type specifications



composing library components. It supports polymorphism, type classes, and higher-order functions.

**Type Query**

**Example Specifications**

| xs | output |
|---|---|
| "OOPSLA2020" | "OPSLA2020" |
| [1,2,1,1] | [1,2,1] |

# Searching for likely types

```
[1,2,1,1] -> [1,2,1]
```

```
“OOPSLA2020” -> “OPSLA2020”
```

# Searching for likely types

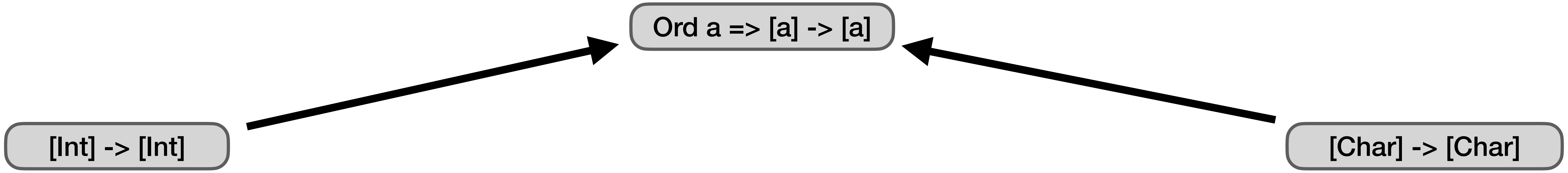
[Int] -> [Int]

[1,2,1,1] -> [1,2,1]

[Char] -> [Char]

“OOPSLA2020” -> “OPSLA2020”

# Searching for likely types



Ord a => [a] -> [a]

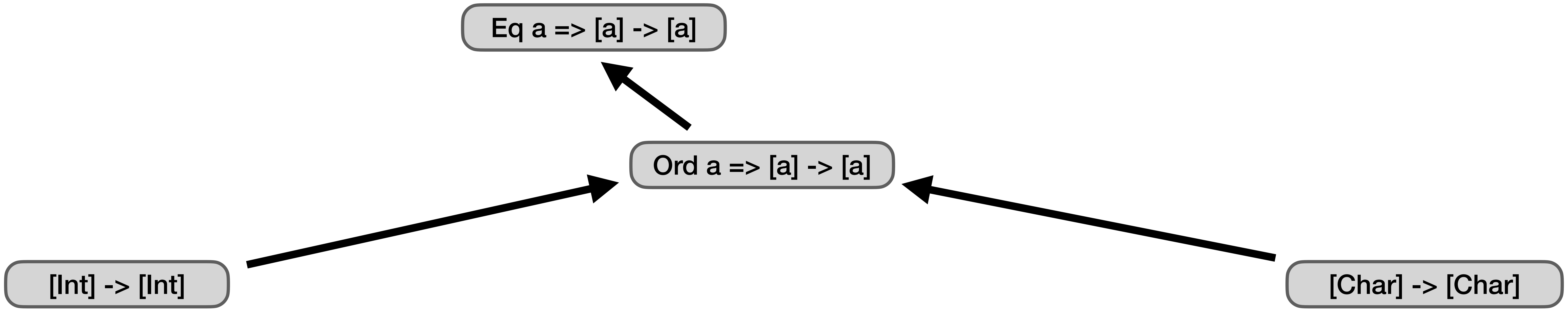
[Int] -> [Int]

[Char] -> [Char]

`[1,2,1,1] -> [1,2,1]`

`“OOPSLA2020” -> “OPSLA2020”`

# Searching for likely types



Eq a => [a] -> [a]

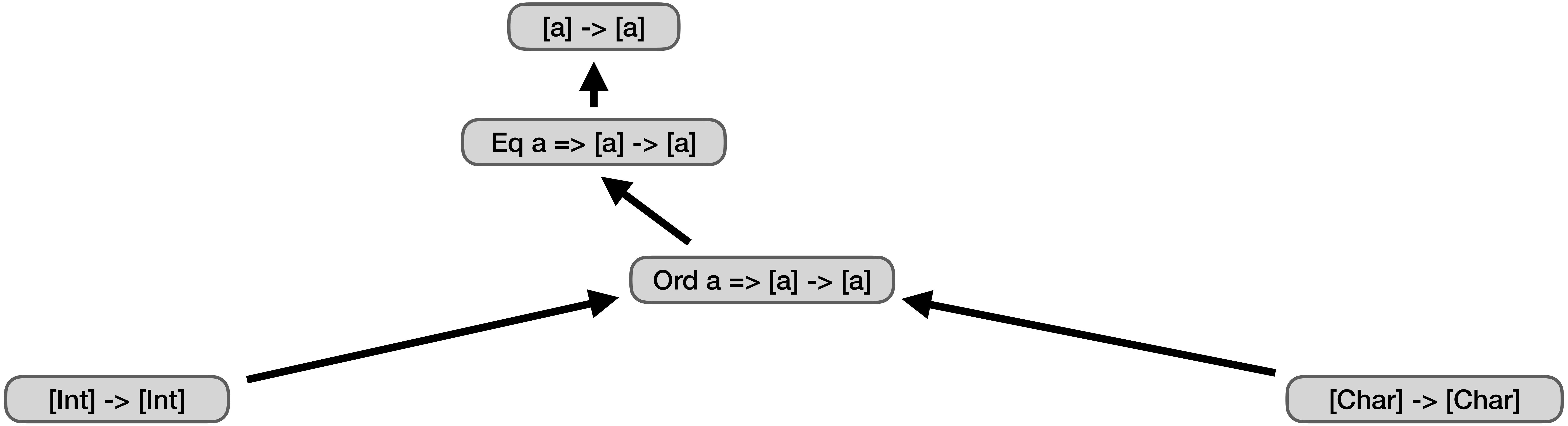
Ord a => [a] -> [a]

[Int] -> [Int]

[Char] -> [Char]

```
[1,2,1,1] -> [1,2,1]                    “OOPSLA2020” -> “OPSLA2020”
```

# Searching for likely types



[a] -> [a]

Eq a => [a] -> [a]

Ord a => [a] -> [a]

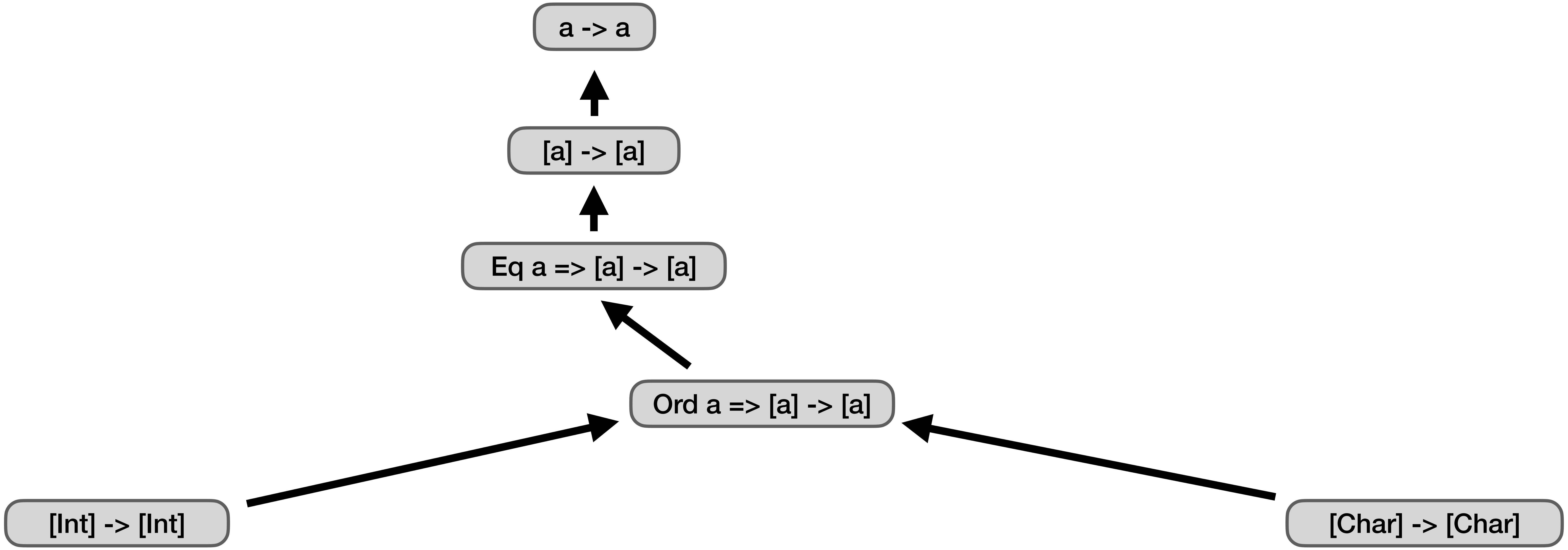
[Int] -> [Int]

[Char] -> [Char]

```
[1,2,1,1] -> [1,2,1]
```

```
“OOPSLA2020” -> “OPSLA2020”
```

# Searching for likely types



a -> a

[a] -> [a]

Eq a => [a] -> [a]

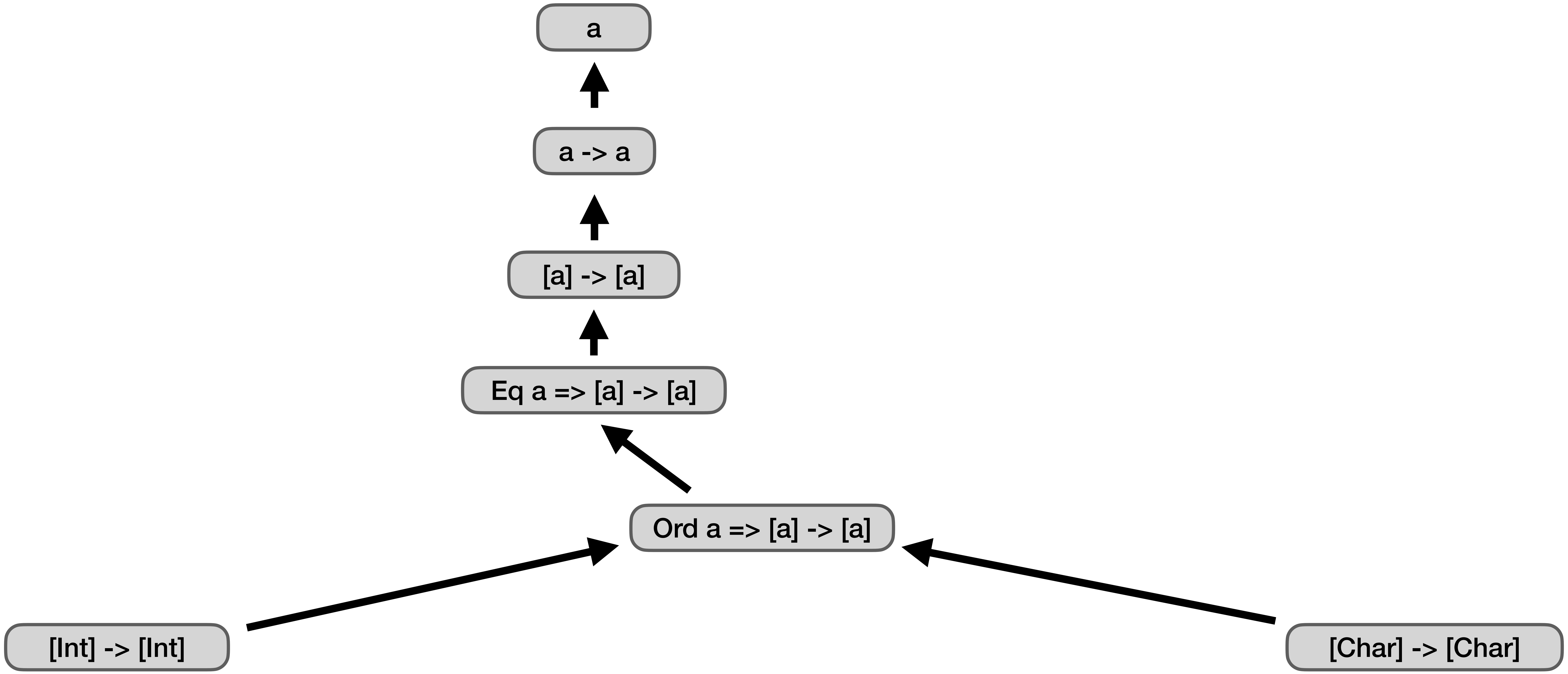
Ord a => [a] -> [a]

[Int] -> [Int]

[Char] -> [Char]

```
[1,2,1,1] -> [1,2,1]                    “OOPSLA2020” -> “OPSLA2020”
```

# Searching for likely types



a

a -> a

[a] -> [a]

Eq a => [a] -> [a]

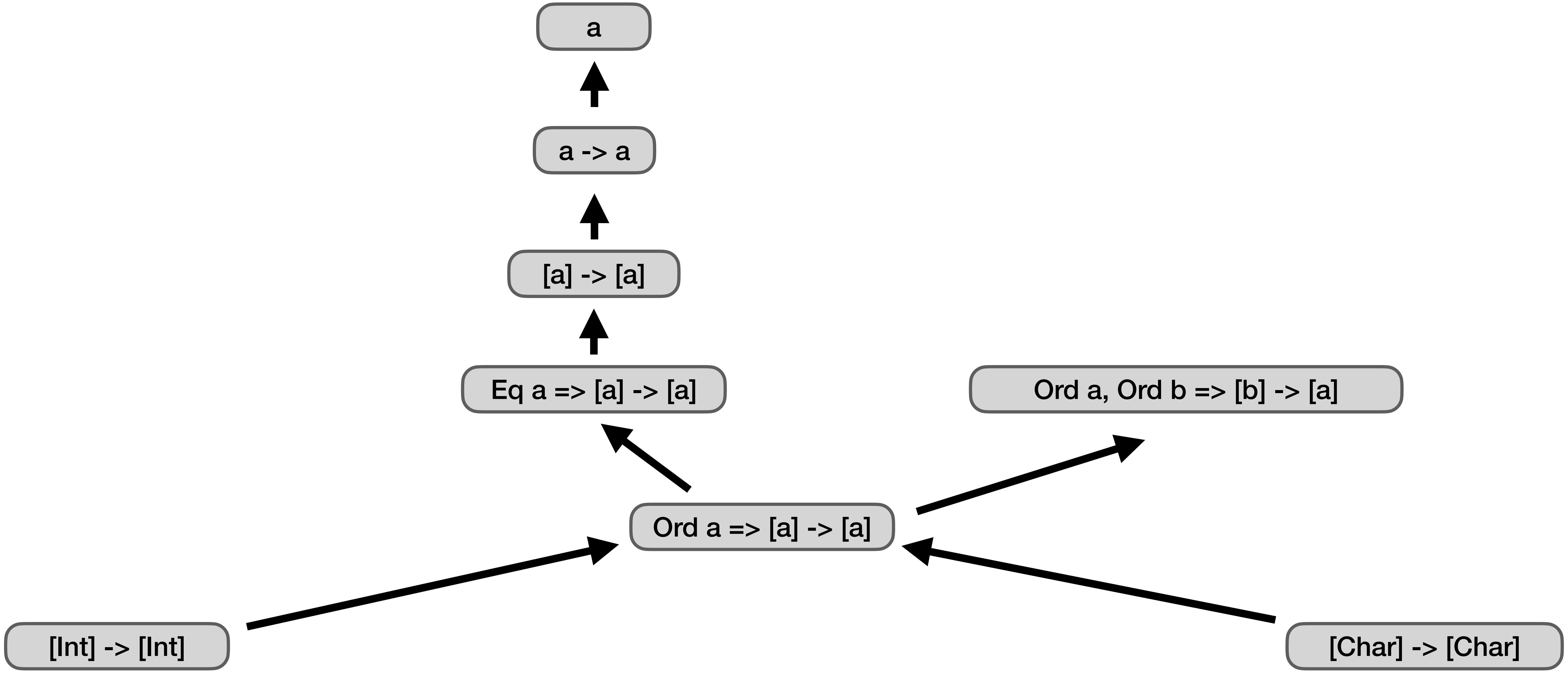
Ord a => [a] -> [a]

[Int] -> [Int]

[Char] -> [Char]

```
[1,2,1,1] -> [1,2,1]                    “OOPSLA2020” -> “OPSLA2020”
```

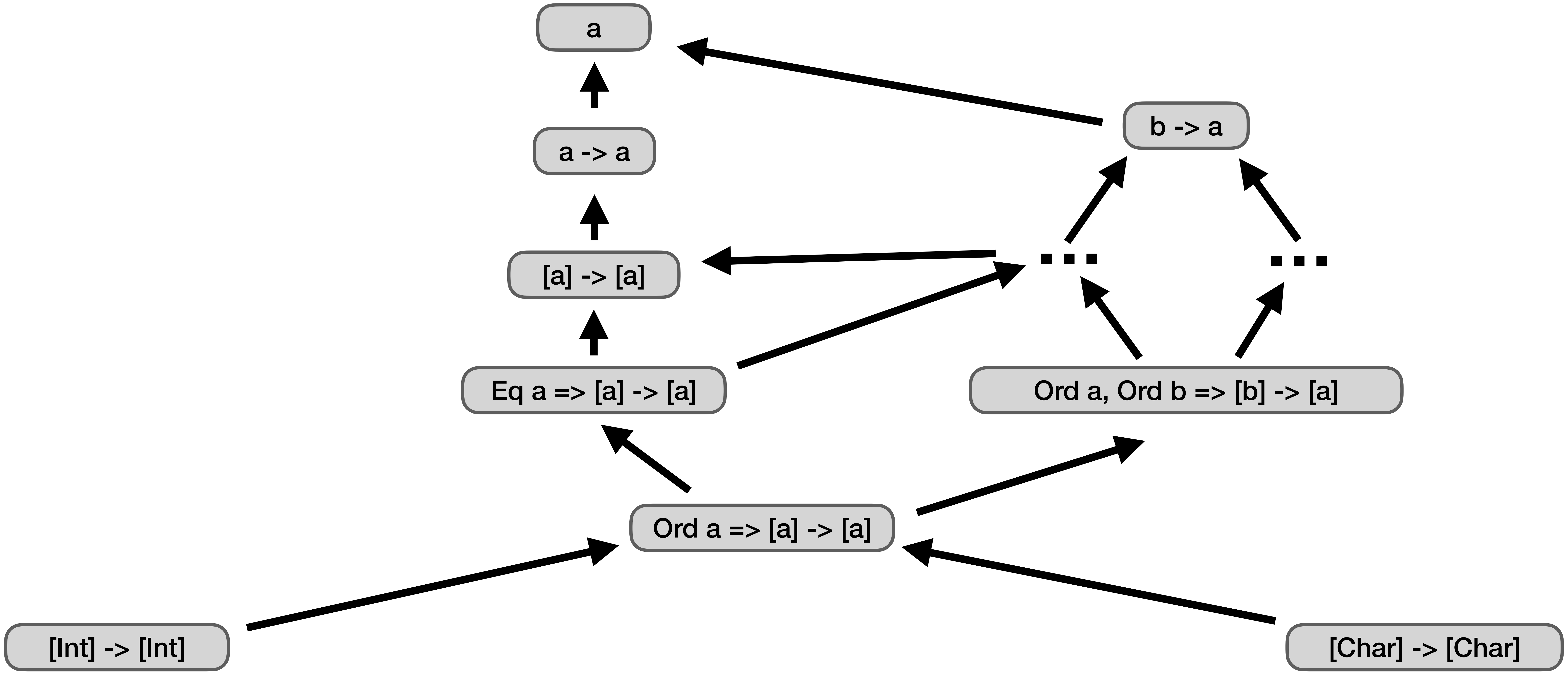
# Searching for likely types



`[1,2,1,1] -> [1,2,1]`

`“OOPSLA2020” -> “OPSLA2020”`

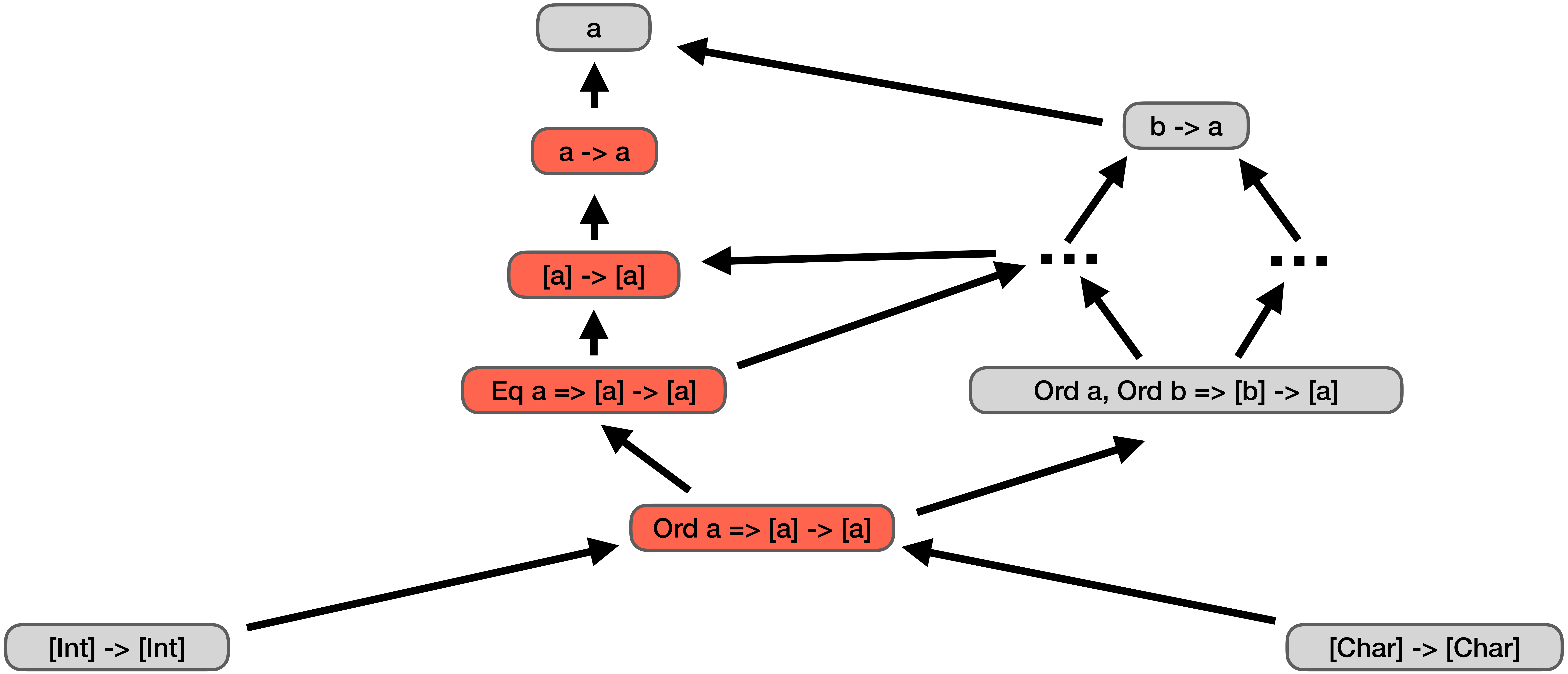
# Searching for likely types



`[1,2,1,1] -> [1,2,1]`

`“OOPSLA2020” -> “OPSLA2020”`

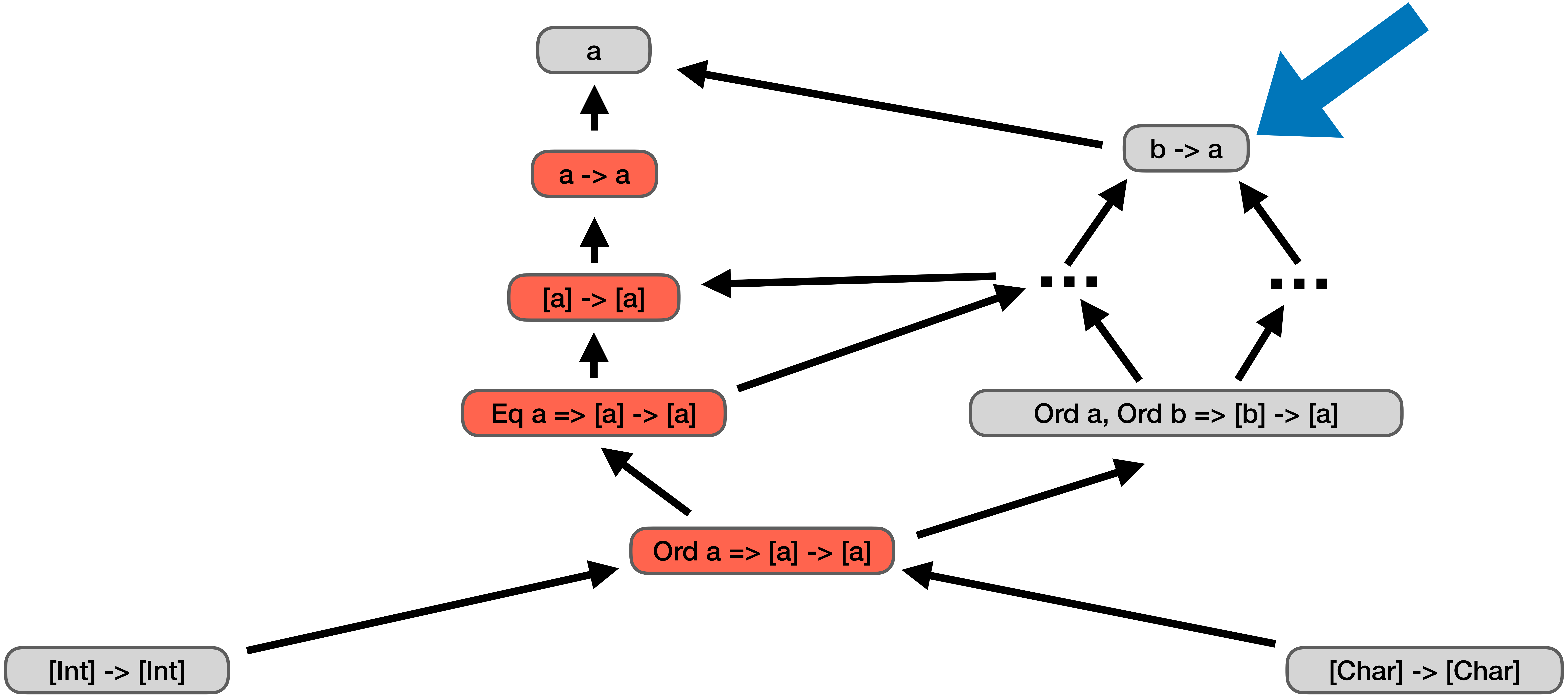
# Searching for likely types



```
[1,2,1,1] -> [1,2,1]
```

```
“OOPSLA2020” -> “OPSLA2020”
```

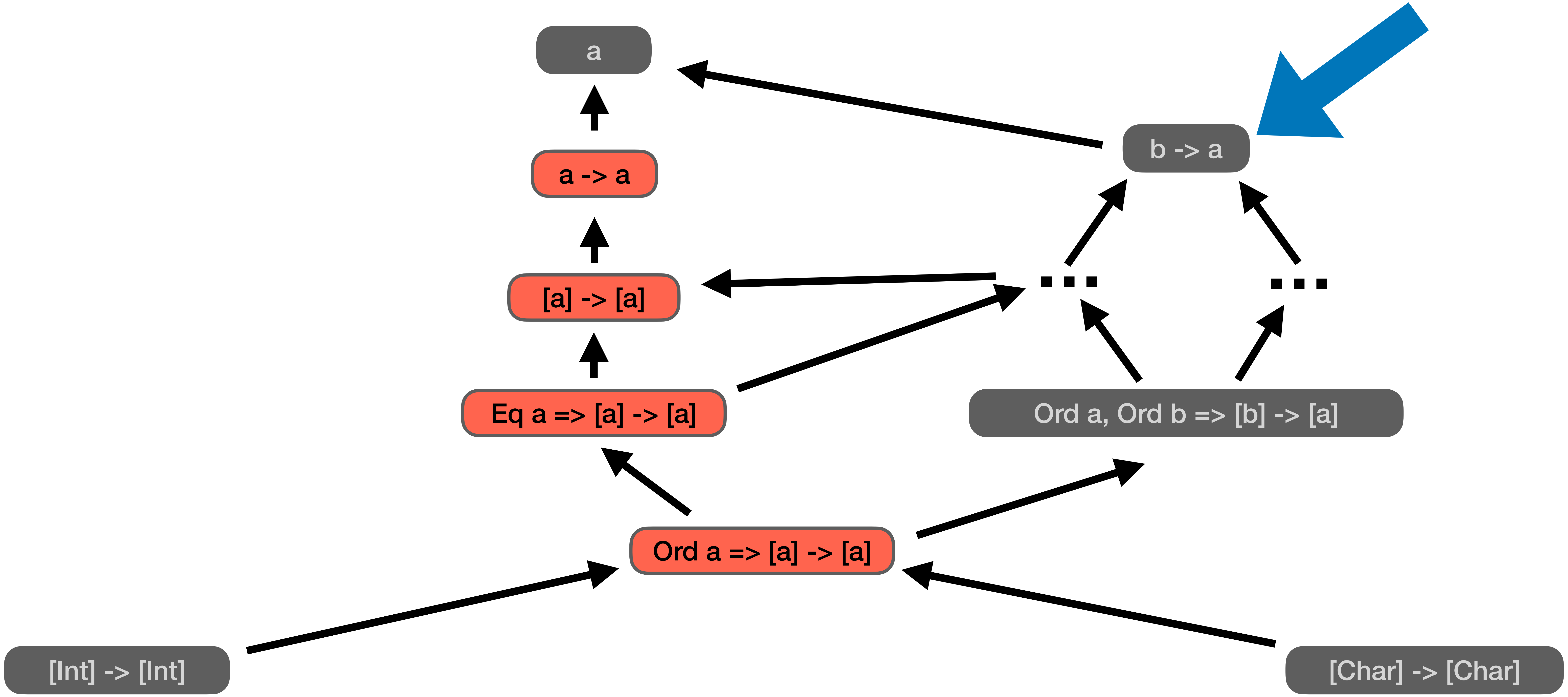
# Filtering for likely types



`[1,2,1,1] -> [1,2,1]`

`“OOPSLA2020” -> “OPSLA2020”`

# Filtering for likely types



```
[1,2,1,1] -> [1,2,1]
```

```
“OOPSLA2020” -> “OPSLA2020”
```

# Ranking types

| | |
|---|---|
| [a] -> [a] | a -> a |
| Eq a => [a] -> [a] | Eq a => a -> a |
| Ord a => [a] -> [a] | Ord a => a -> a |

```
[1,2,1,1] -> [1,2,1]
```

```
“OOPSLA2020” -> “OPSLA2020”
```

# Ranking types

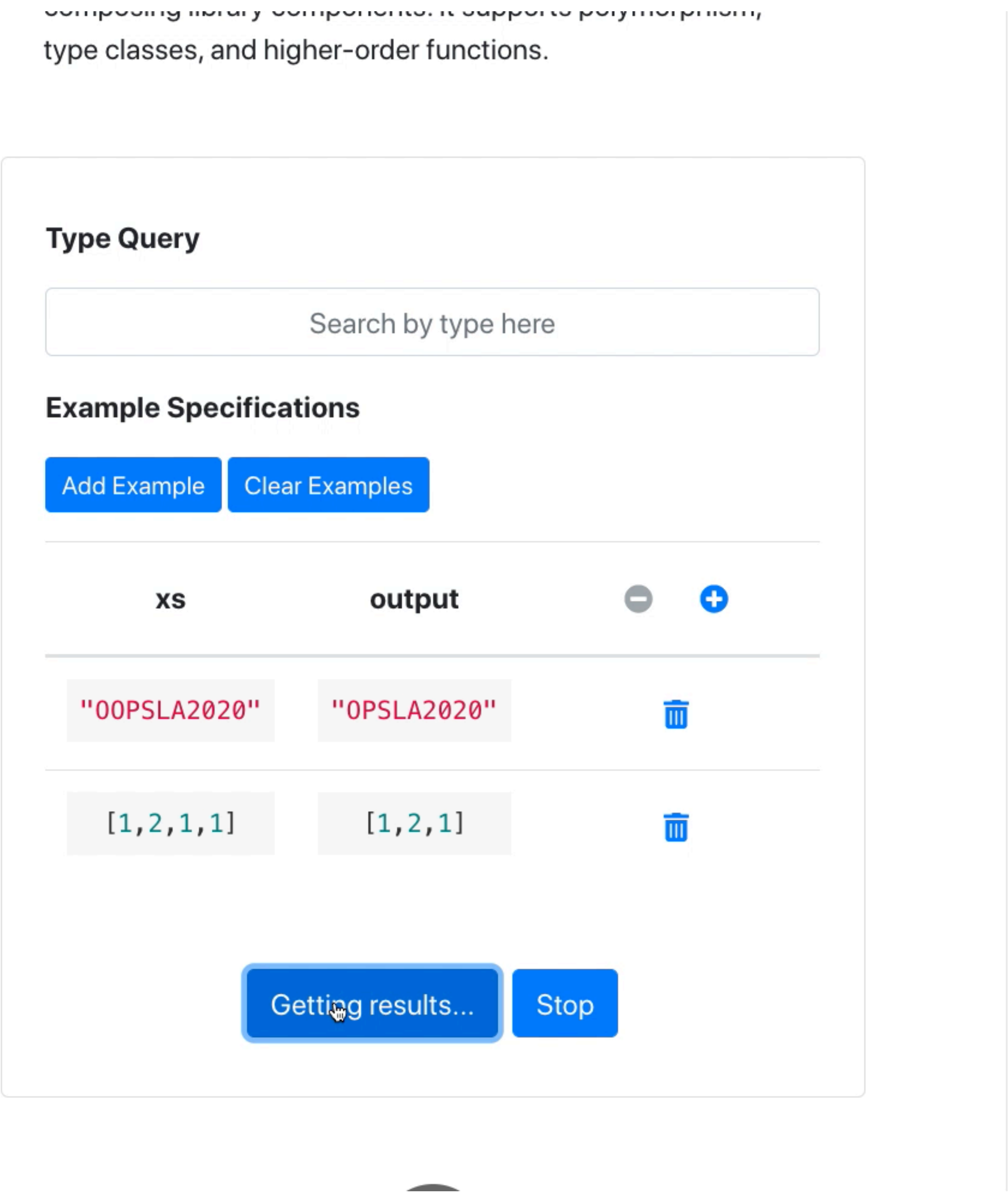
1. [a] -> [a]
2. Eq a => [a] -> [a]
3. Ord a => [a] -> [a]
4. a -> a
5. Eq a => a -> a

```
[1,2,1,1] -> [1,2,1]                    “OOPSLA2020” -> “OPSLA2020”
```

# Types from Tests

Challenge: How to infer likely type specifications from tests?

1. Generalized types
2. Filter types
3. Rank types



type classes, and higher-order functions.

**Type Query**

Search by type here

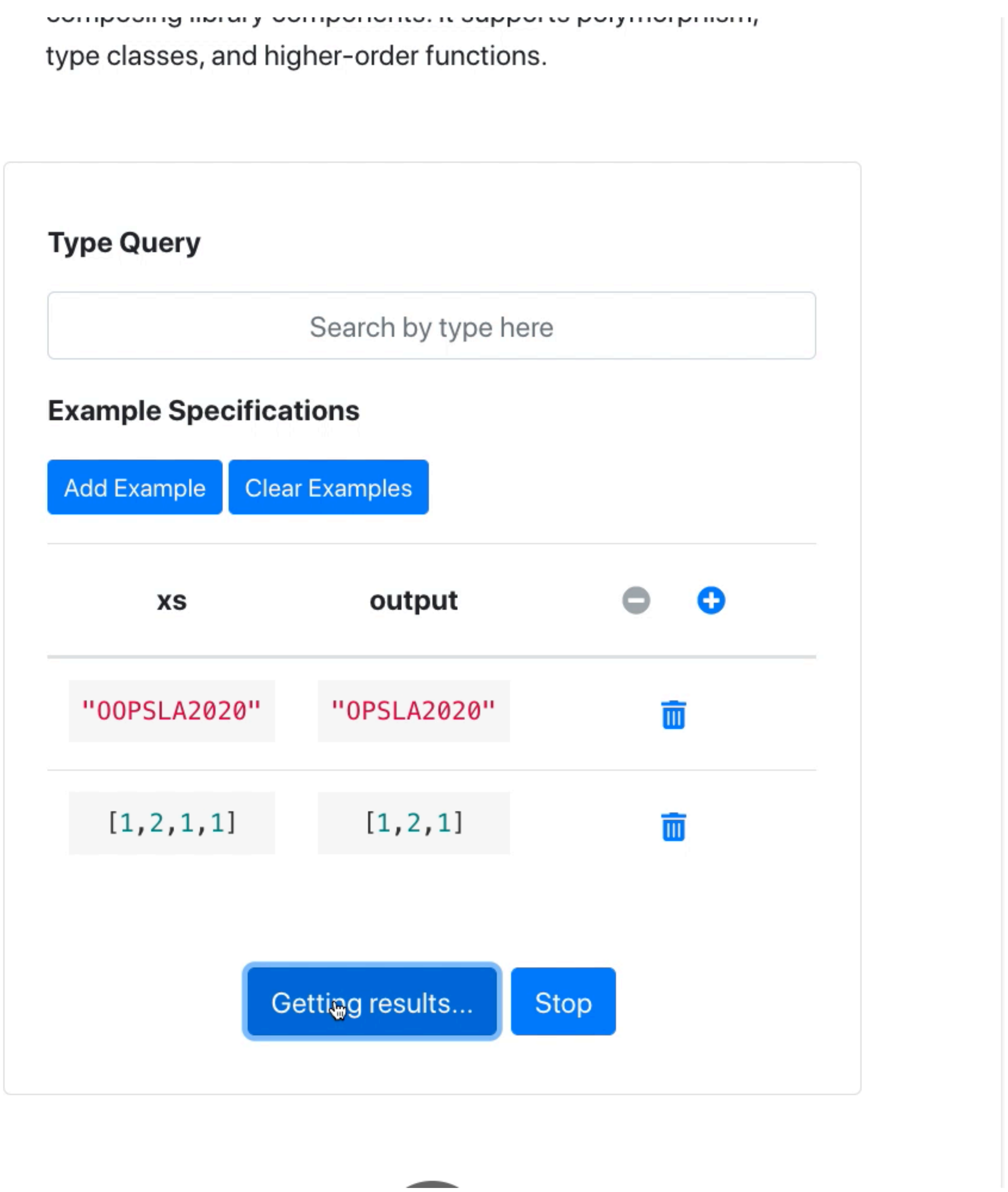
**Example Specifications**

| xs | output |
|---|---|
| "OOPSLA2020" | "OPSLA2020" |
| [1,2,1,1] | [1,2,1] |

# Types from Tests

**Challenge: How to infer likely type specifications from tests?**

1. Generalized types
2. Filter types
3. Rank types



composing library components. It supports polymorphism, type classes, and higher-order functions.

**Type Query**

Search by type here

**Example Specifications**

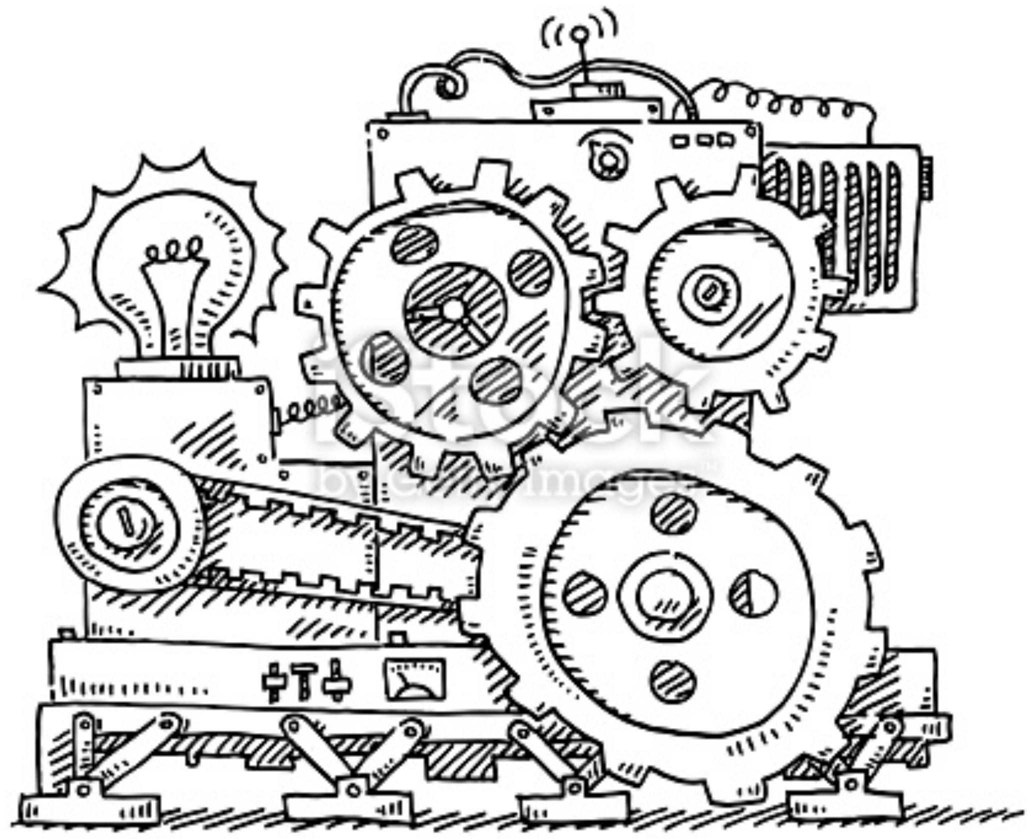
| xs | output |
|---|---|
| "OOPSLA2020" | "OPSLA2020" |
| [1,2,1,1] | [1,2,1] |

Specification

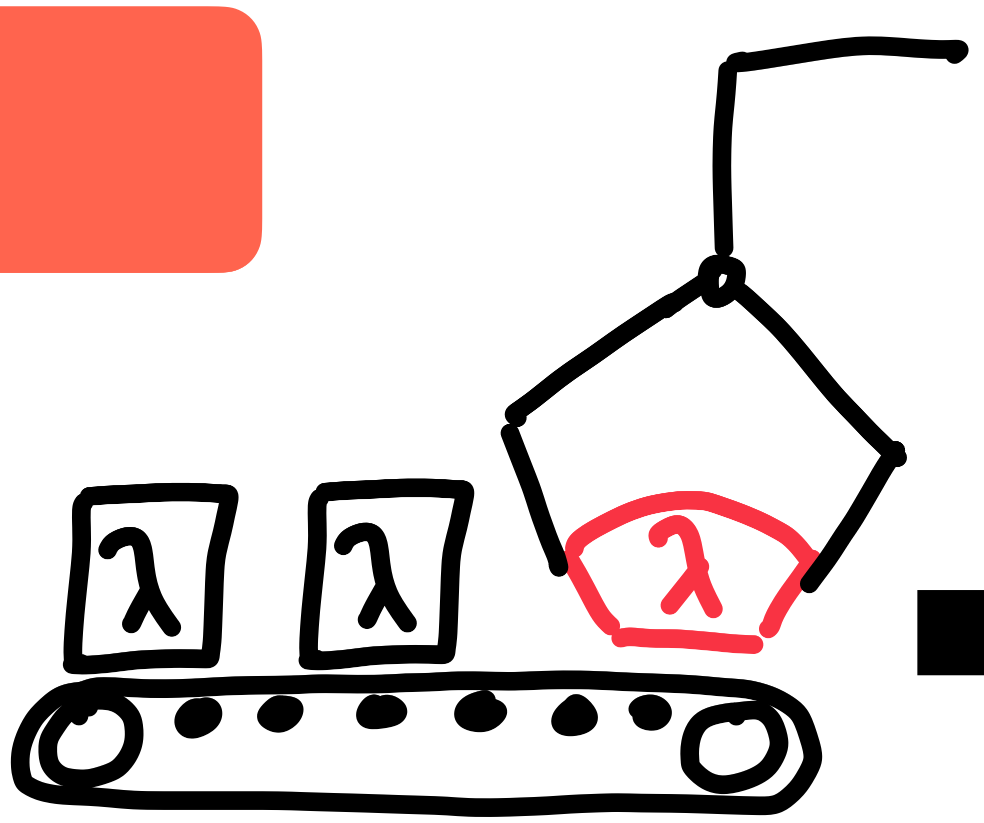
Filtering

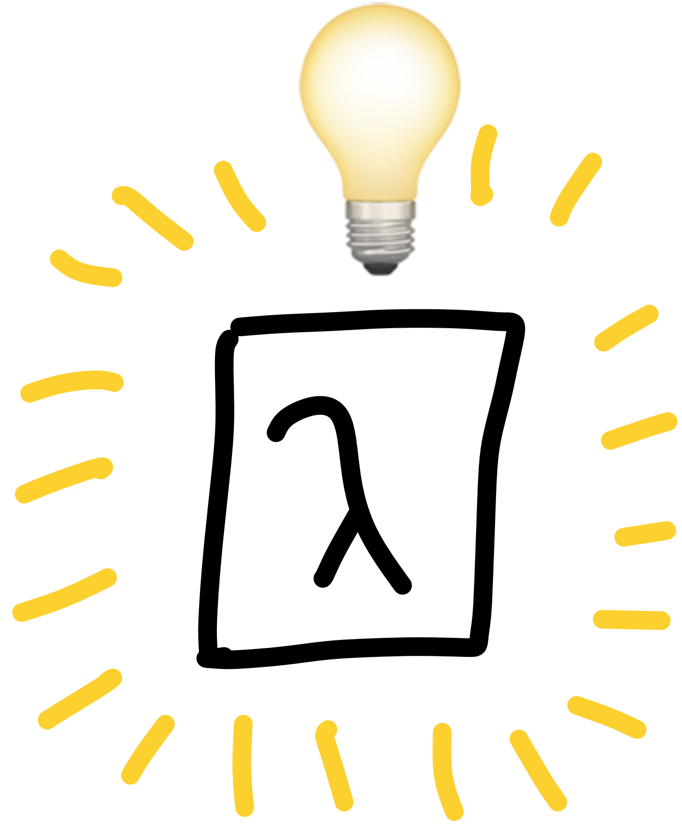
Comprehension

**Type**
**or**
**Test**



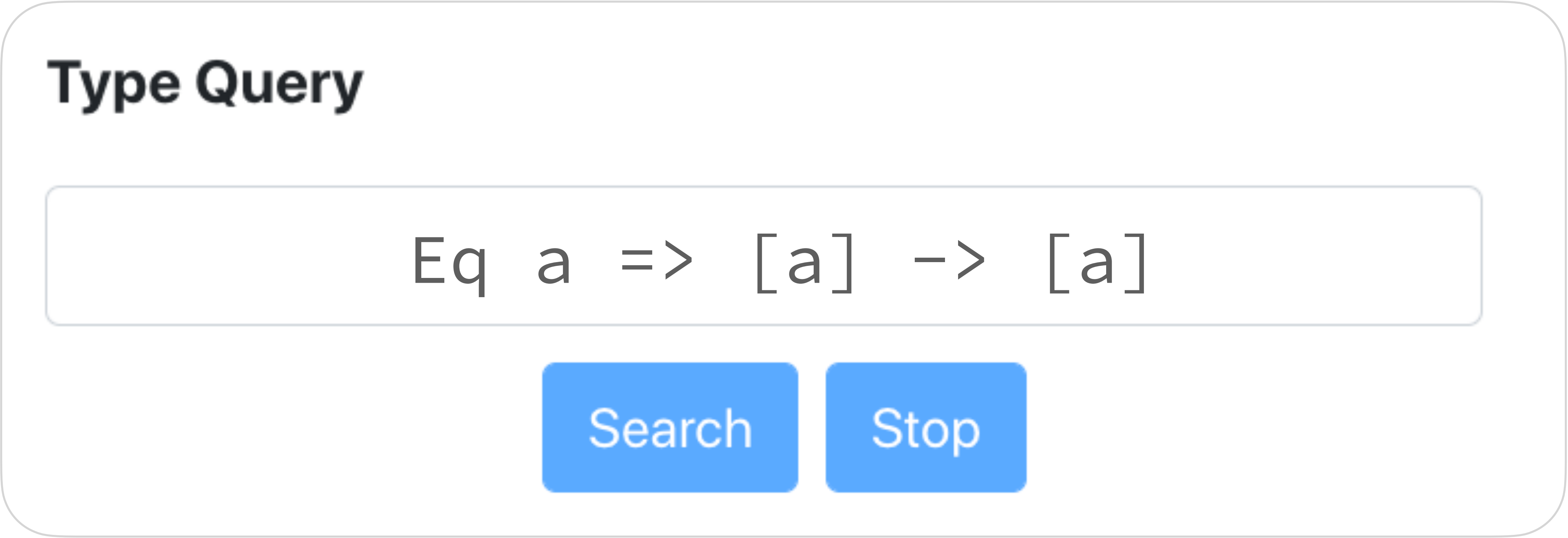
**Program Synthesis by**
**Type-Guided Abstraction Refinement**
**[Guo et al. 2020]**





Hoogle+ User Study

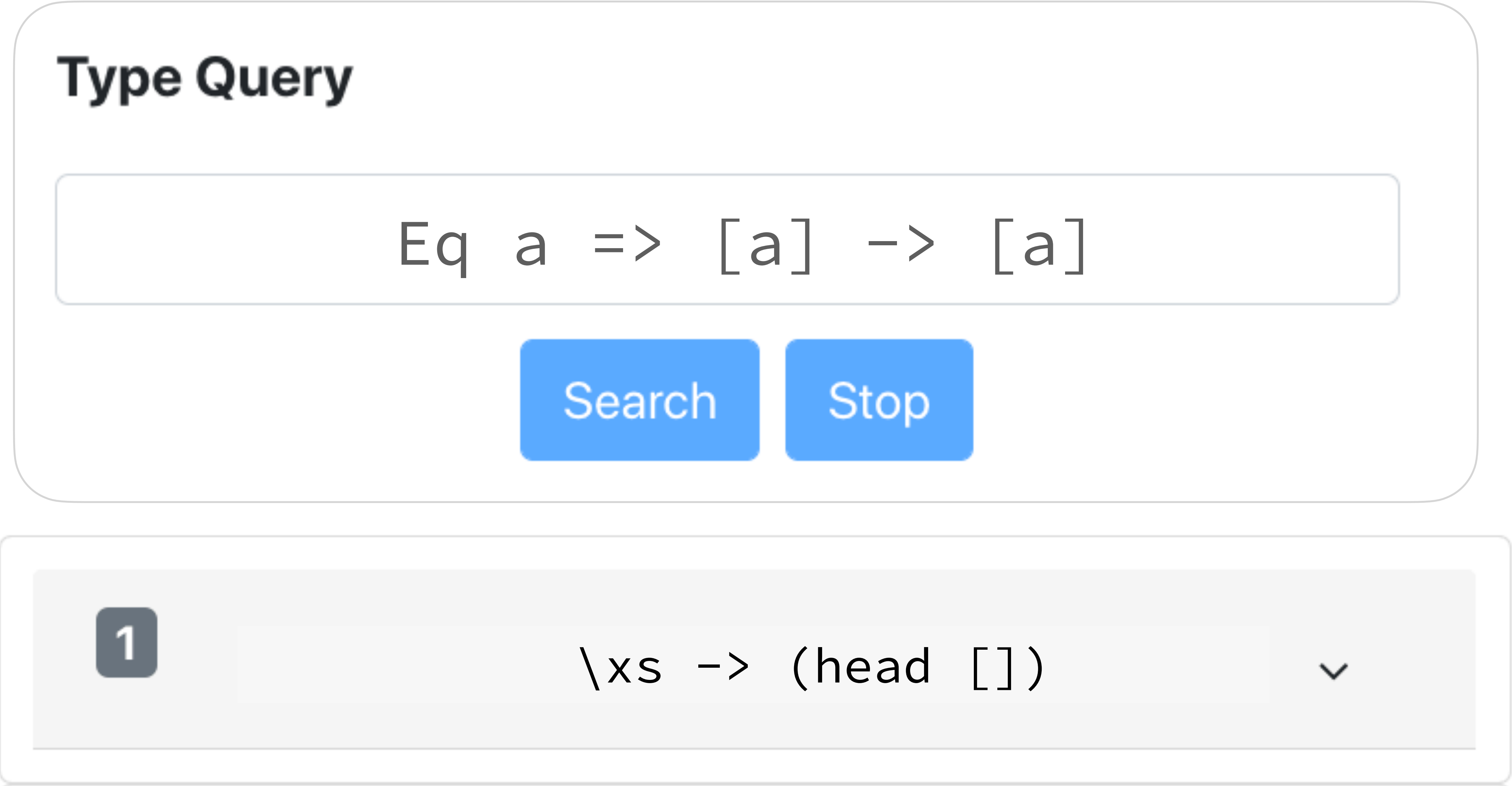
# Filtering Programs



Type Query

```
Eq a => [a] -> [a]
```

# Filtering Programs



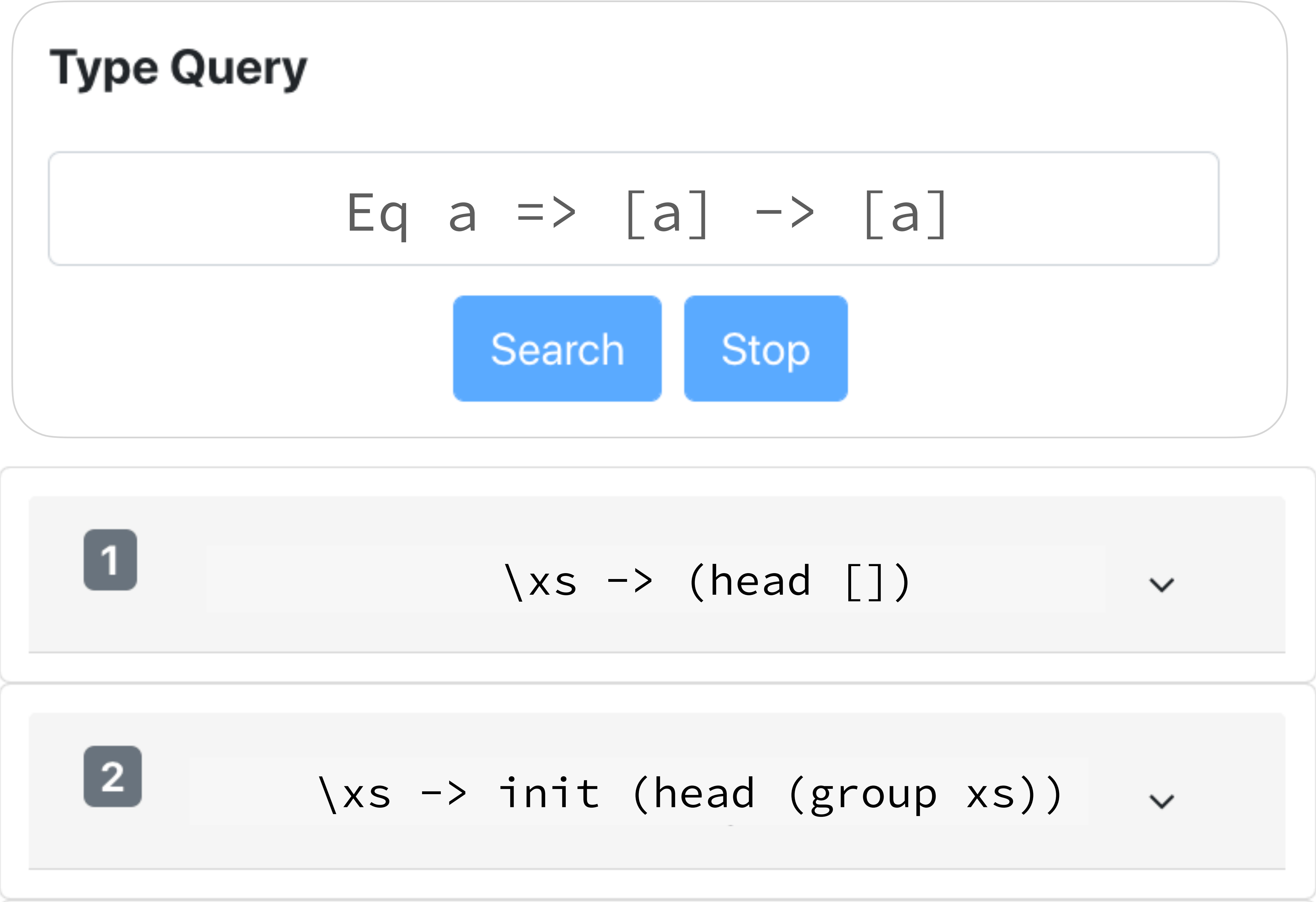
Type Query

```
Eq a => [a] -> [a]
```

1

```
\xs -> (head [])
```

# Filtering Programs



Type Query

```
Eq a => [a] -> [a]
```

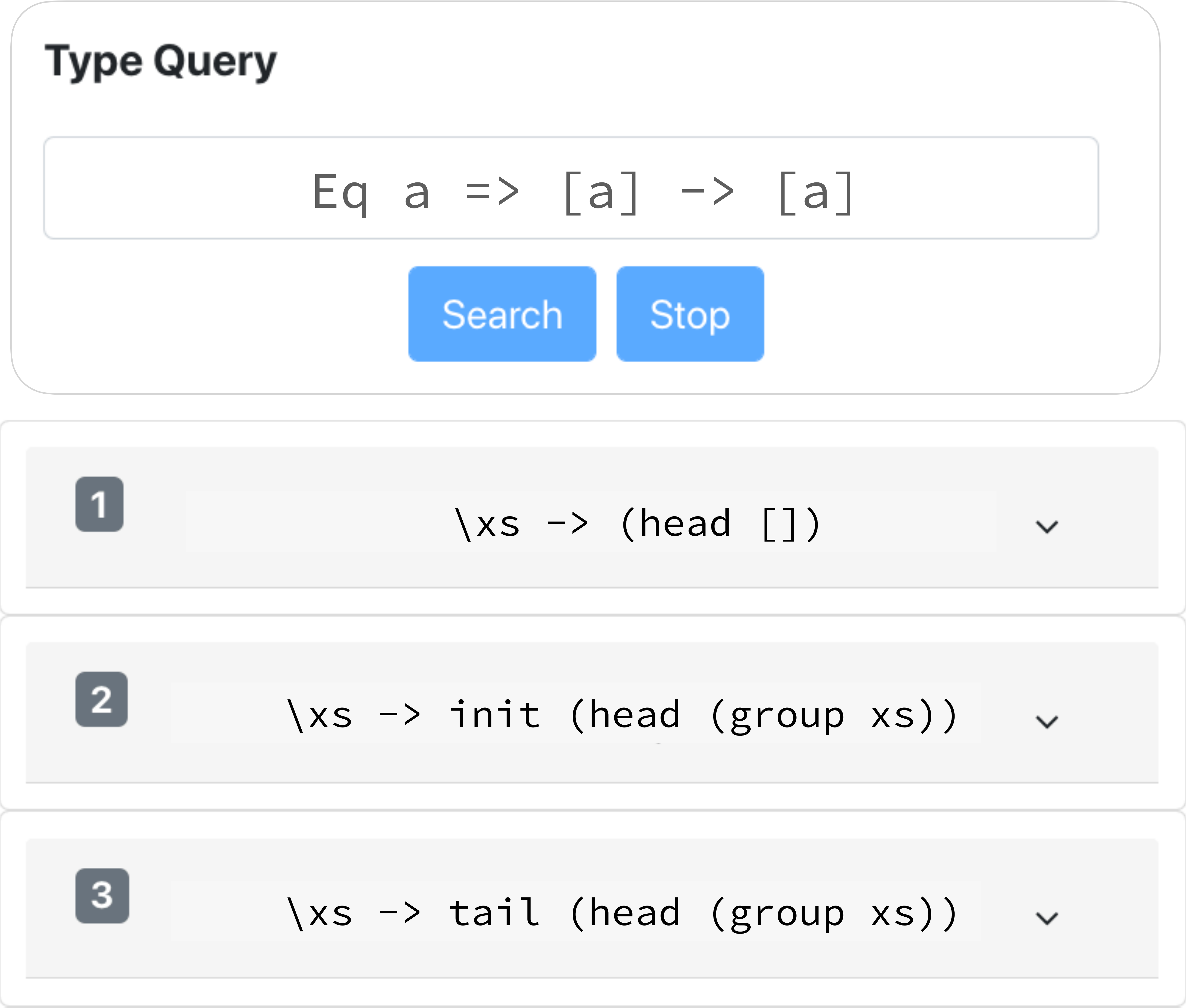
1

```
\xs -> (head [])
```

2

```
\xs -> init (head (group xs))
```

# Filtering Programs



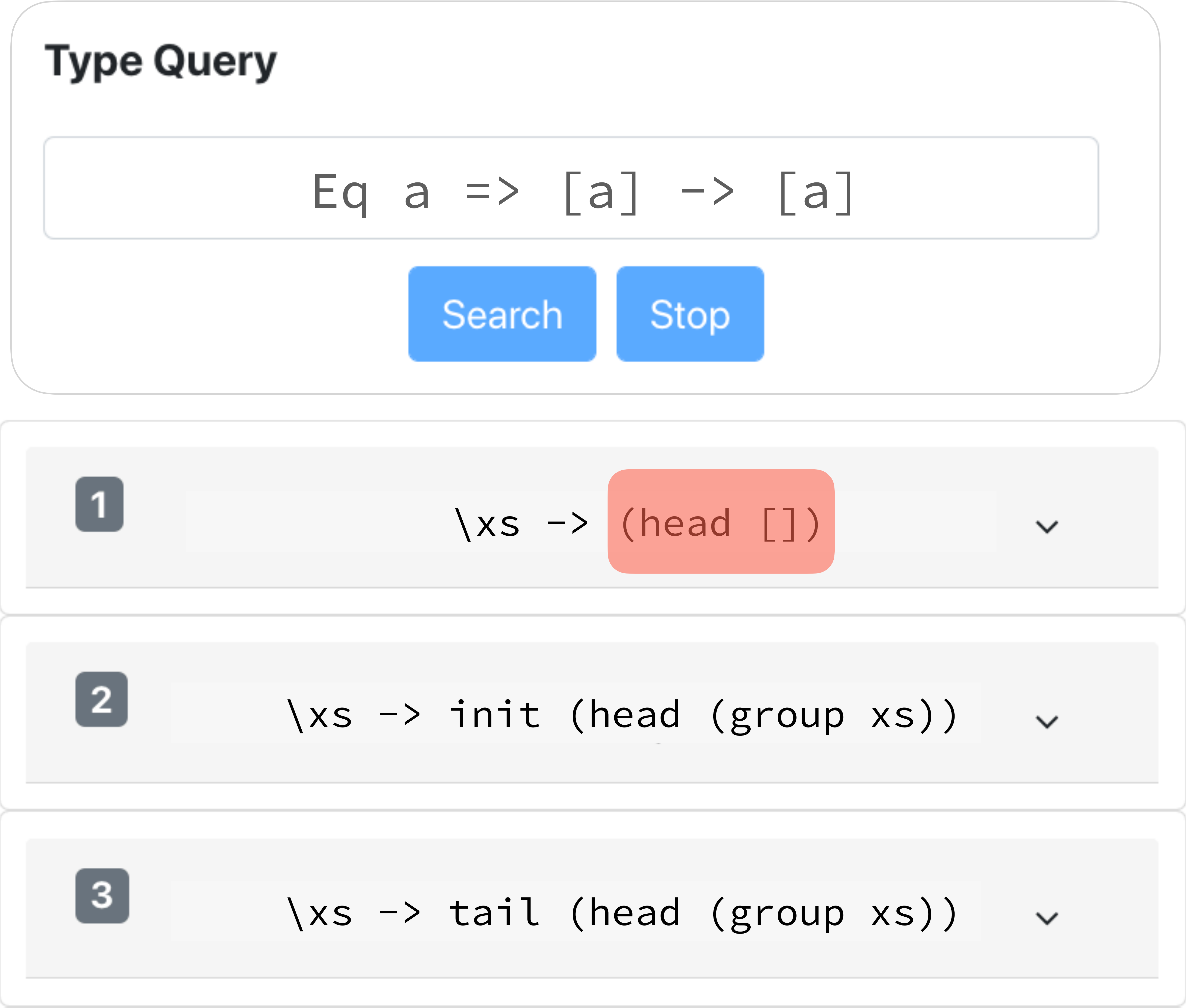
```
Eq a => [a] -> [a]
```

1
```
\xs -> (head [])
```

2
```
\xs -> init (head (group xs))
```

3
```
\xs -> tail (head (group xs))
```

# Filtering Programs

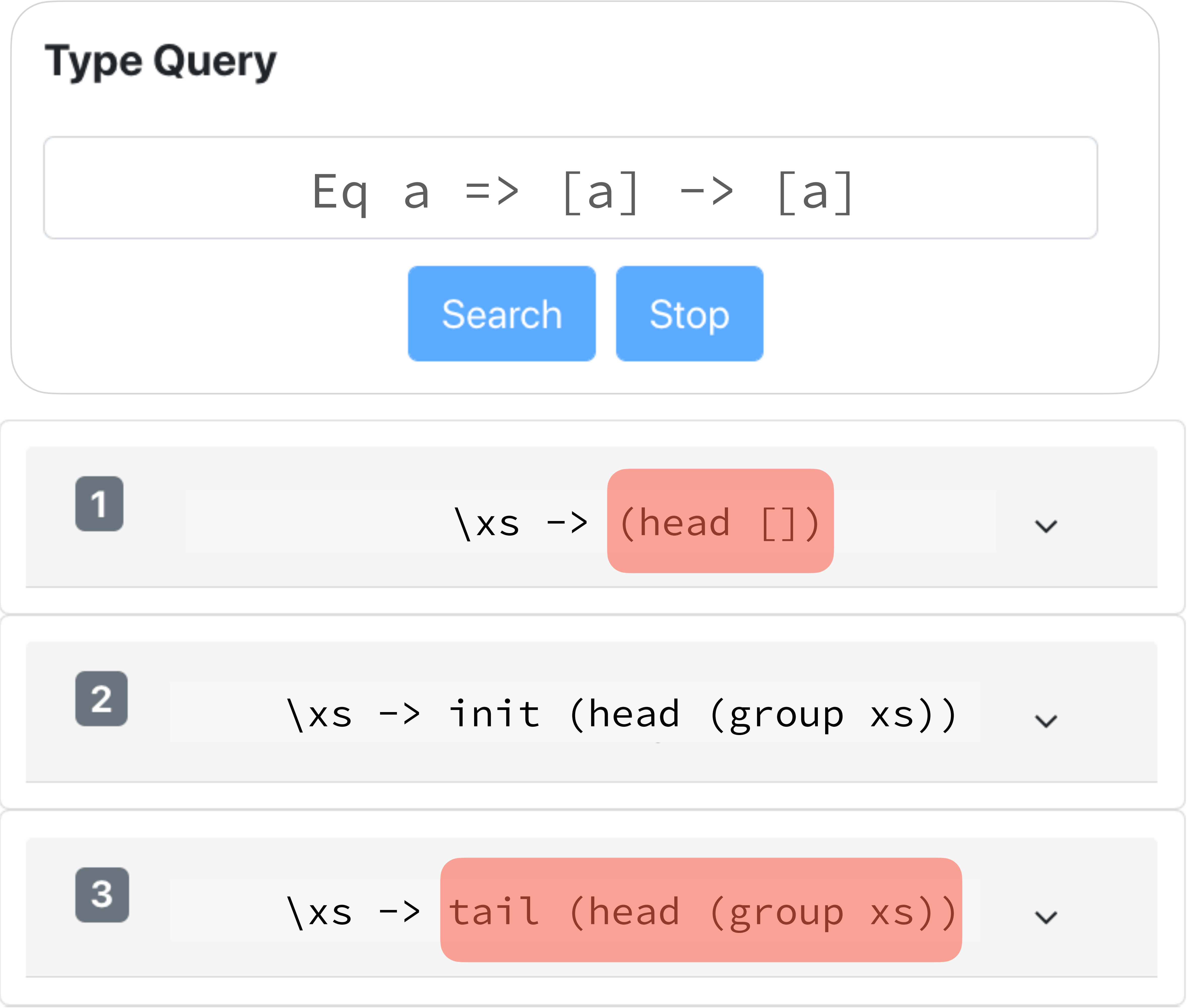


Type Query

```
Eq a => [a] -> [a]
```

```
1  \xs -> (head [])
2  \xs -> init (head (group xs))
3  \xs -> tail (head (group xs))
```

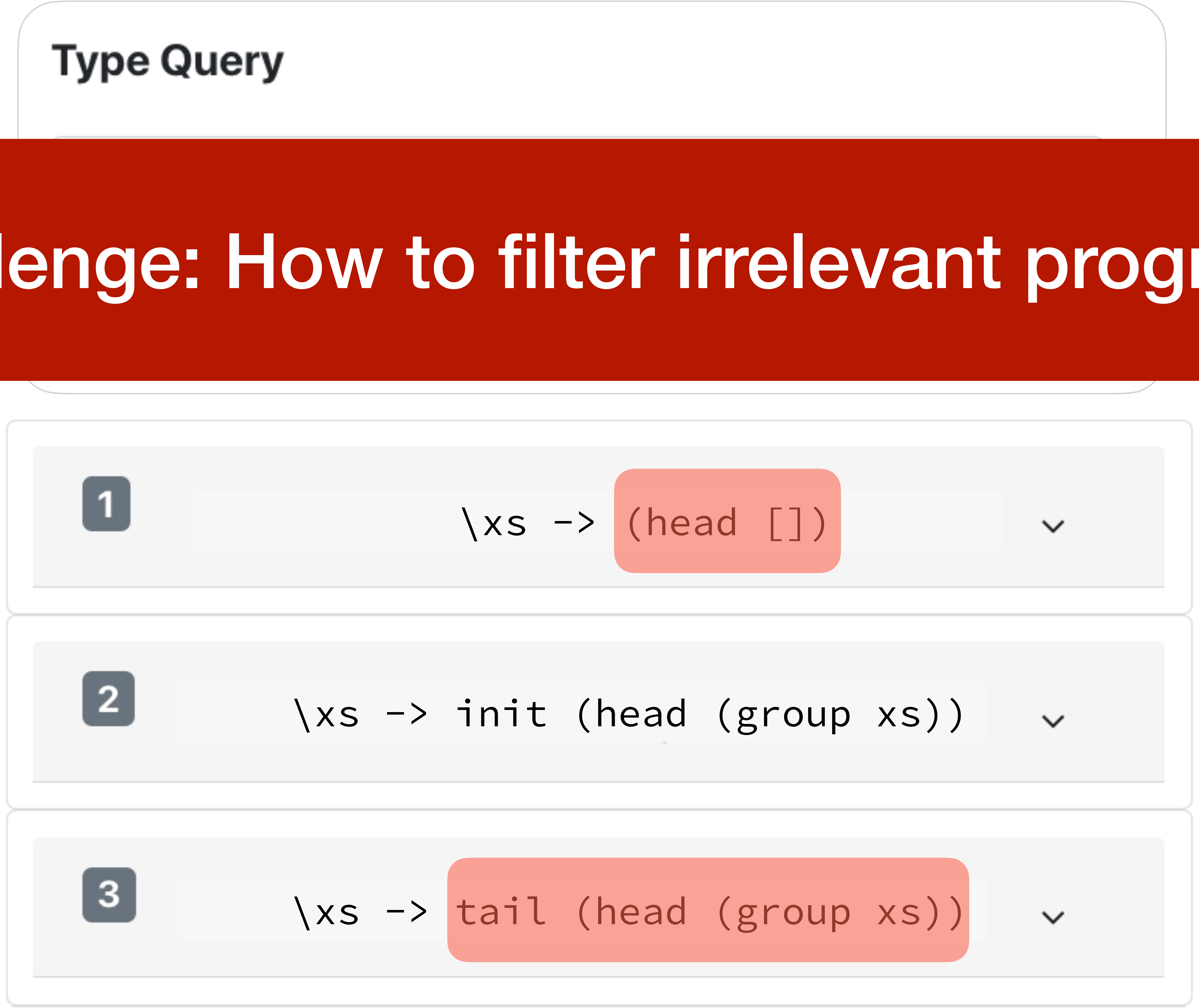
# Filtering Programs



```
Eq a => [a] -> [a]
```

```
1  \xs -> (head [])
2  \xs -> init (head (group xs))
3  \xs -> tail (head (group xs))
```

# Filtering Programs



Type Query

Challenge: How to filter irrelevant programs?

1
```
\xs -> (head [])
```

2
```
\xs -> init (head (group xs))
```

3
```
\xs -> tail (head (group xs))
```

# Filtering Programs - Smallcheck

**Test ALL the values!**

Smallcheck[†]

[†]:[Runciman, Naylor, Lindblad. 2008]

# Filtering Programs - Smallcheck

$\nexists x, y. (\ldots)$ →

Property

**Test ALL the values!**

Smallcheck[†]

†:[Runciman, Naylor, Lindblad. 2008]

# Filtering Programs - Smallcheck

**Test ALL the values!**

$\nexists x, y. (\ldots)$ →

Property

Smallcheck[†]

Holds (up to k) → ✅

†:[Runciman, Naylor, Lindblad. 2008]

# Filtering Programs - Smallcheck

$\nexists x, y. (\ldots)$ →

Property

**Test ALL the values!**

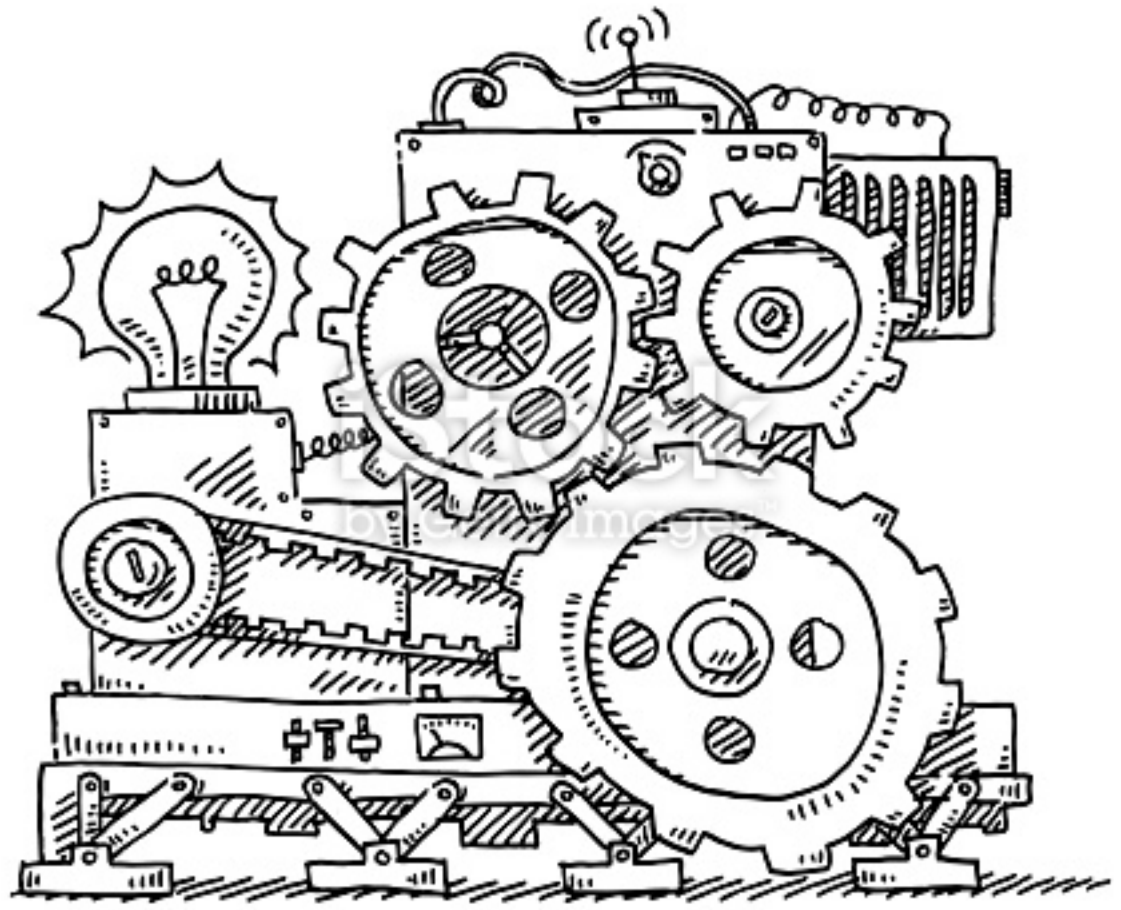
Smallcheck[†]

Holds (up to k) → ✅

Does not hold → ❌💁‍♀️

†:[Runciman, Naylor, Lindblad. 2008]

# Filtering Programs - Hoogle+



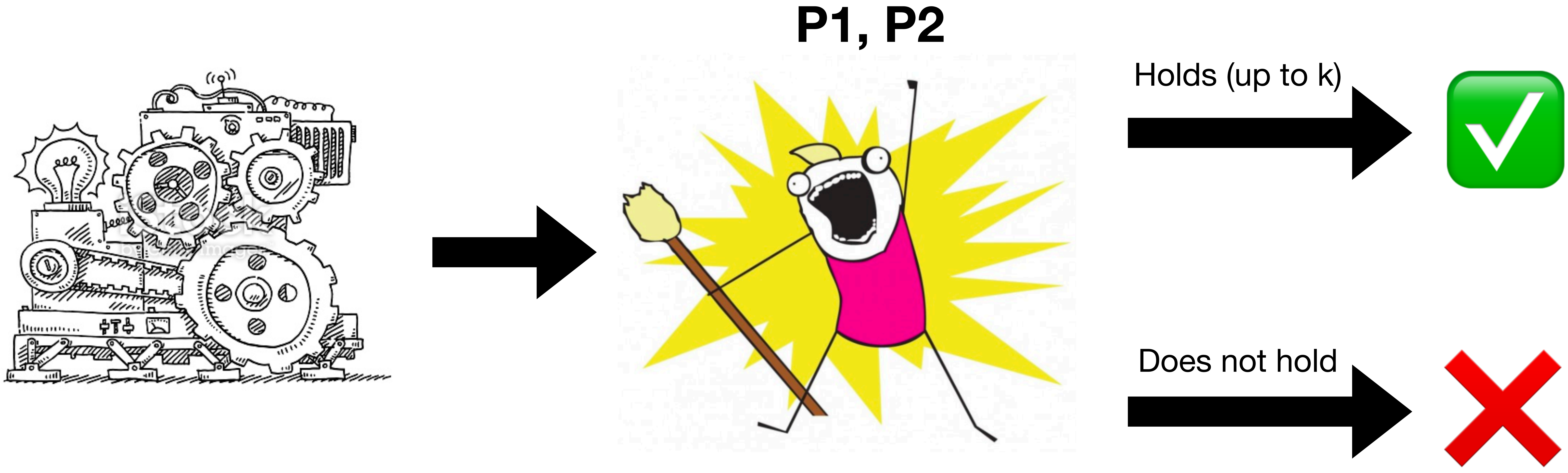
Smallcheck†

Holds (up to k) ✅

Does not hold ❌

†:[Runciman, Naylor, Lindblad. 2008]

# Filtering Programs - Hoogle+



**P1, P2**

Holds (up to k) ✅

Does not hold ❌

Smallcheck†

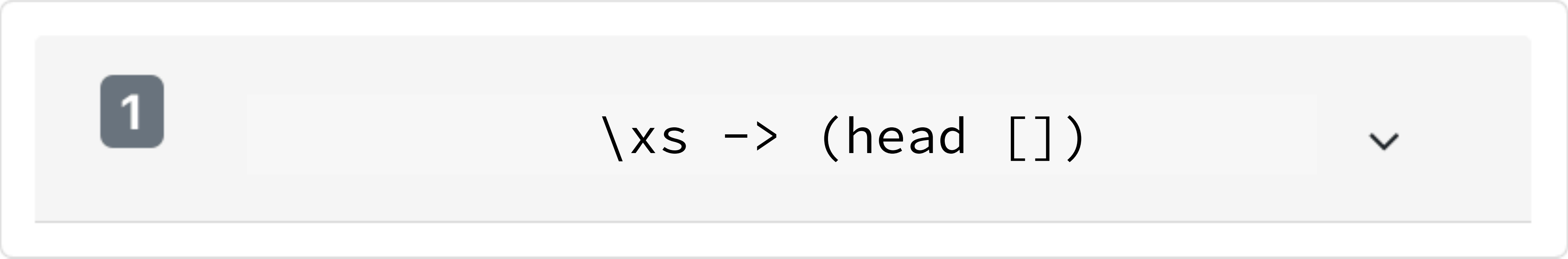
†:[Runciman, Naylor, Lindblad. 2008]

# Filtering Properties

P1. SOME input produces ANY output

# Filtering Properties

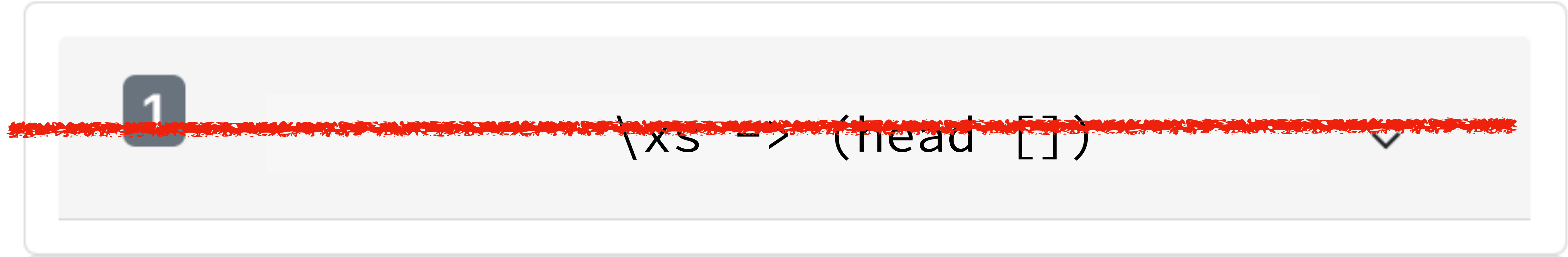
P1. SOME input produces ANY output



```
\xs -> (head [])
```

# Filtering Properties

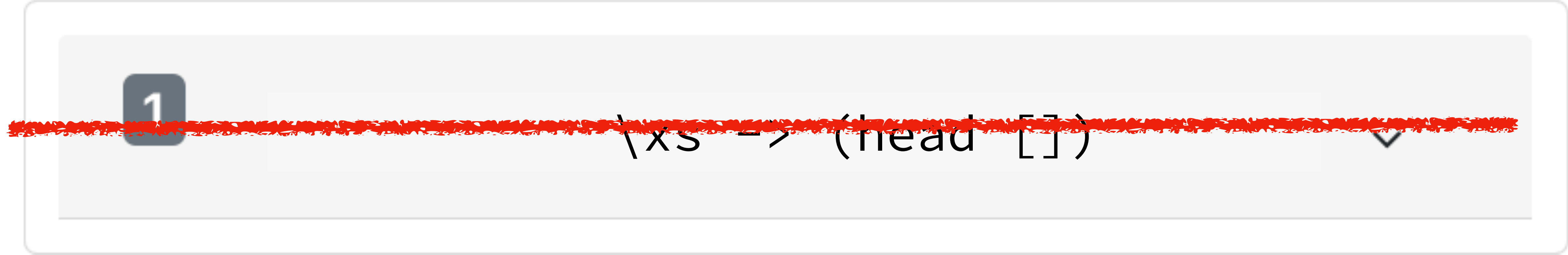
P1. SOME input produces ANY output



1 ~~`\xs -> (head [])`~~

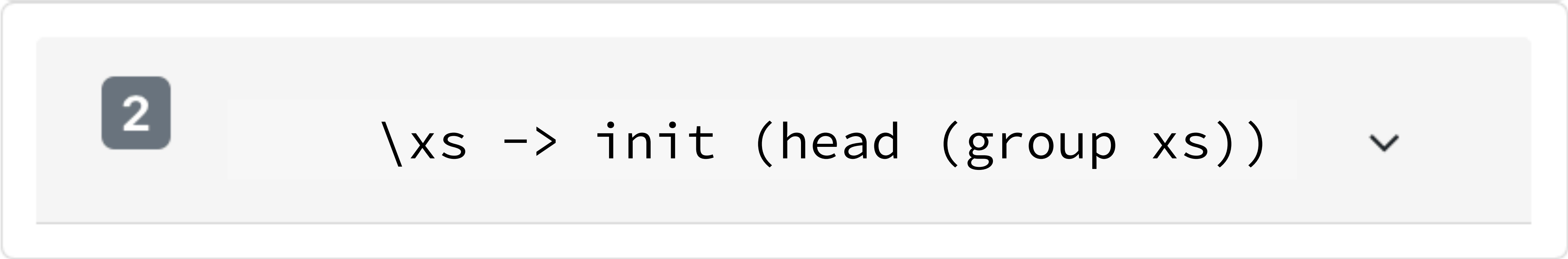
# Filtering Properties

P1. SOME input produces ANY output

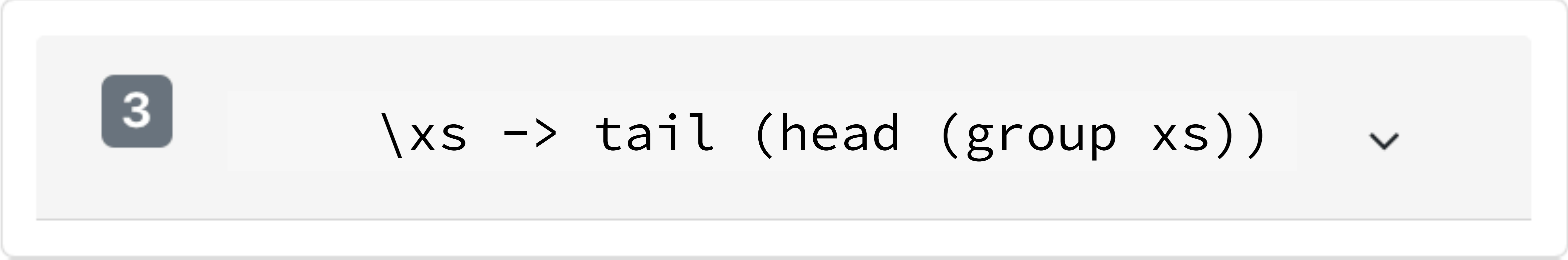


1 ~~\xs -> (head [])~~

P2. SOME input produces different outputs



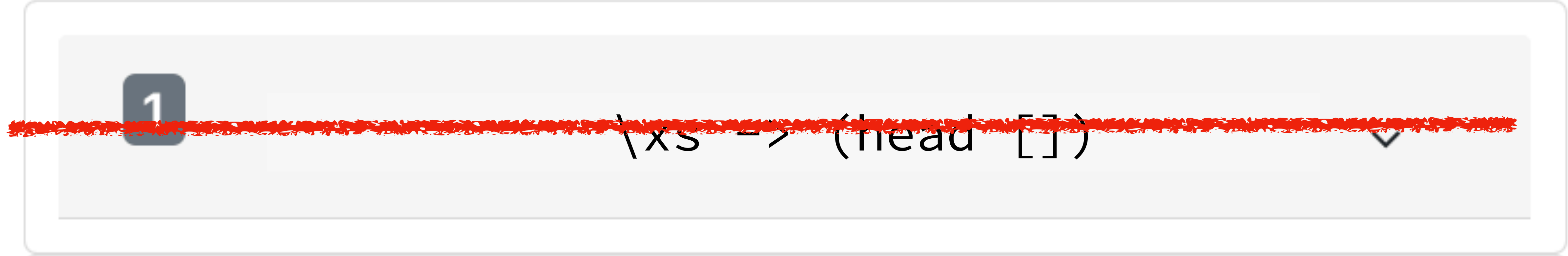
2 `\xs -> init (head (group xs))`



3 `\xs -> tail (head (group xs))`

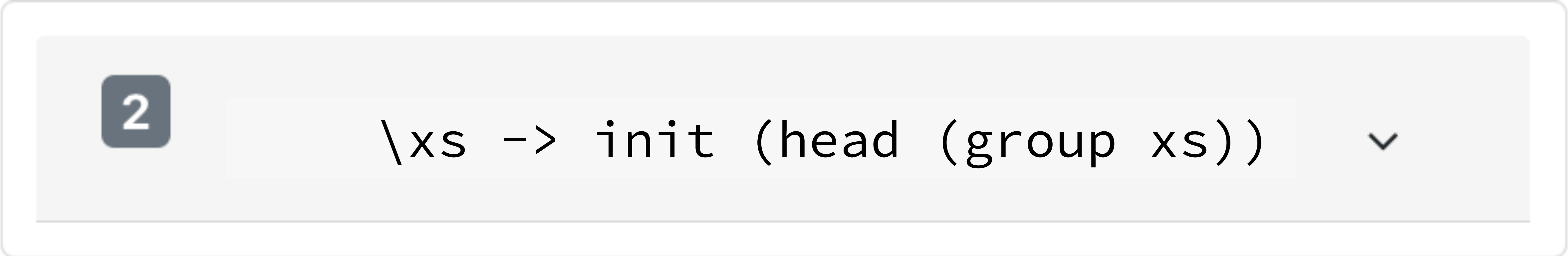
# Filtering Properties

P1. SOME input produces ANY output

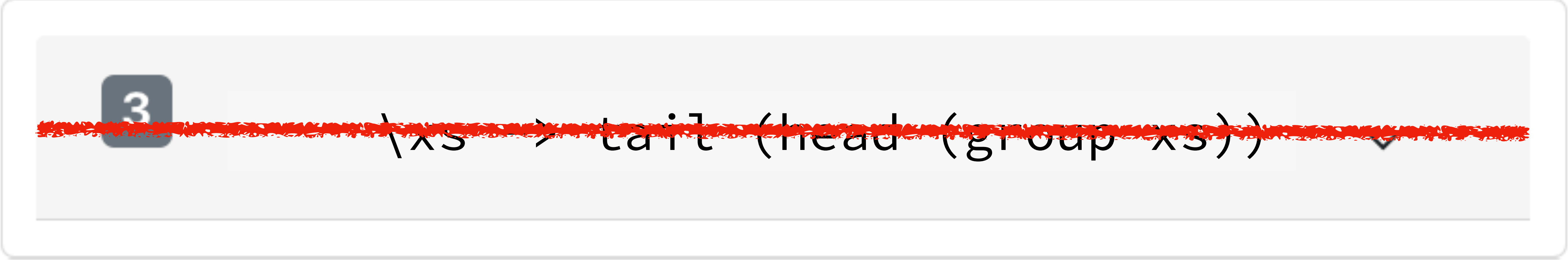


1 ~~\xs -> (head [])~~

P2. SOME input produces different outputs



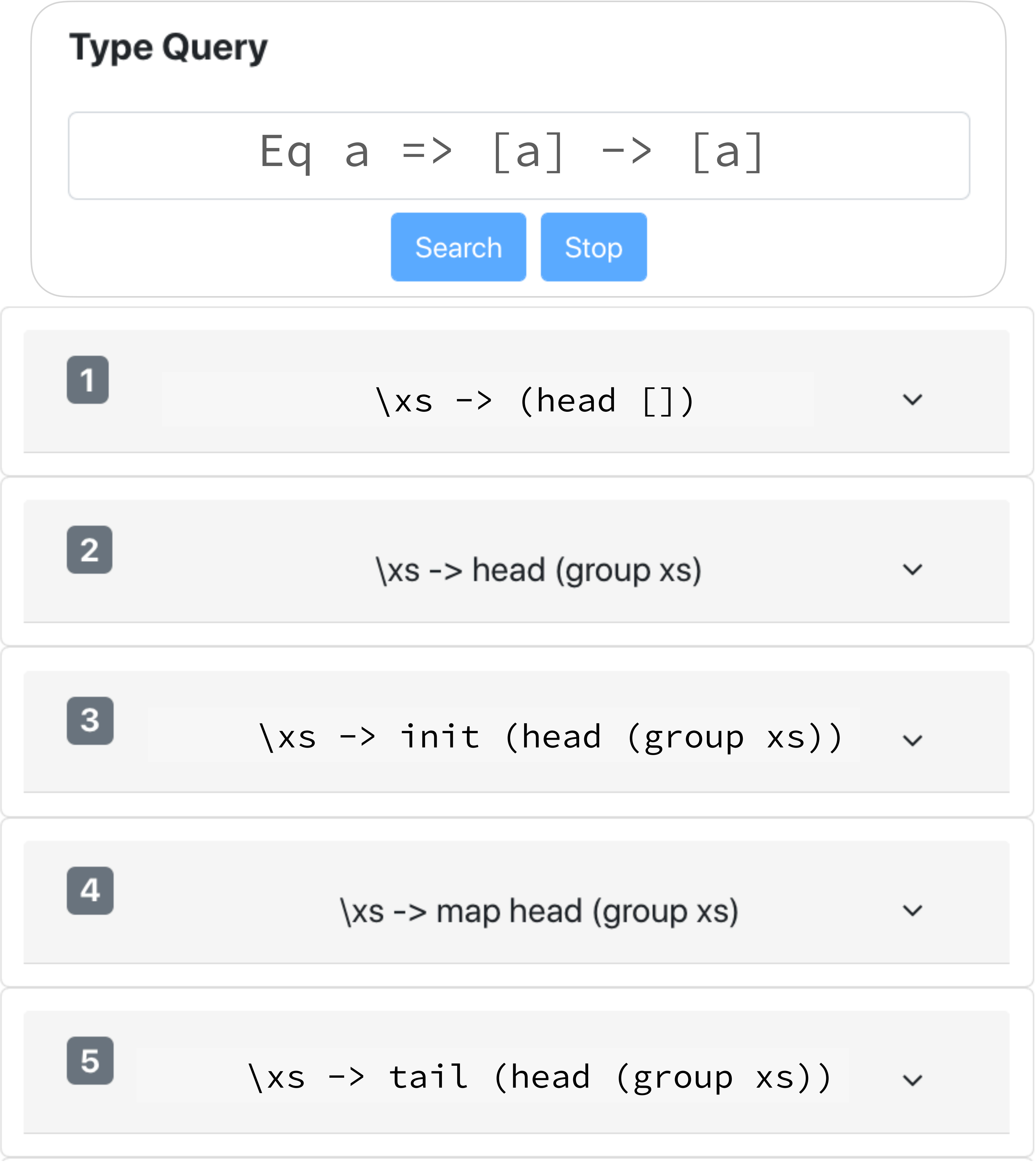
2 \xs -> init (head (group xs))



3 ~~\xs -> tail (head (group xs))~~

# Filtered Search

Challenge: How to filter irrelevant programs?



**Type Query**

```
Eq a => [a] -> [a]
```

1 `\xs -> (head [])`

2 `\xs -> head (group xs)`

3 `\xs -> init (head (group xs))`

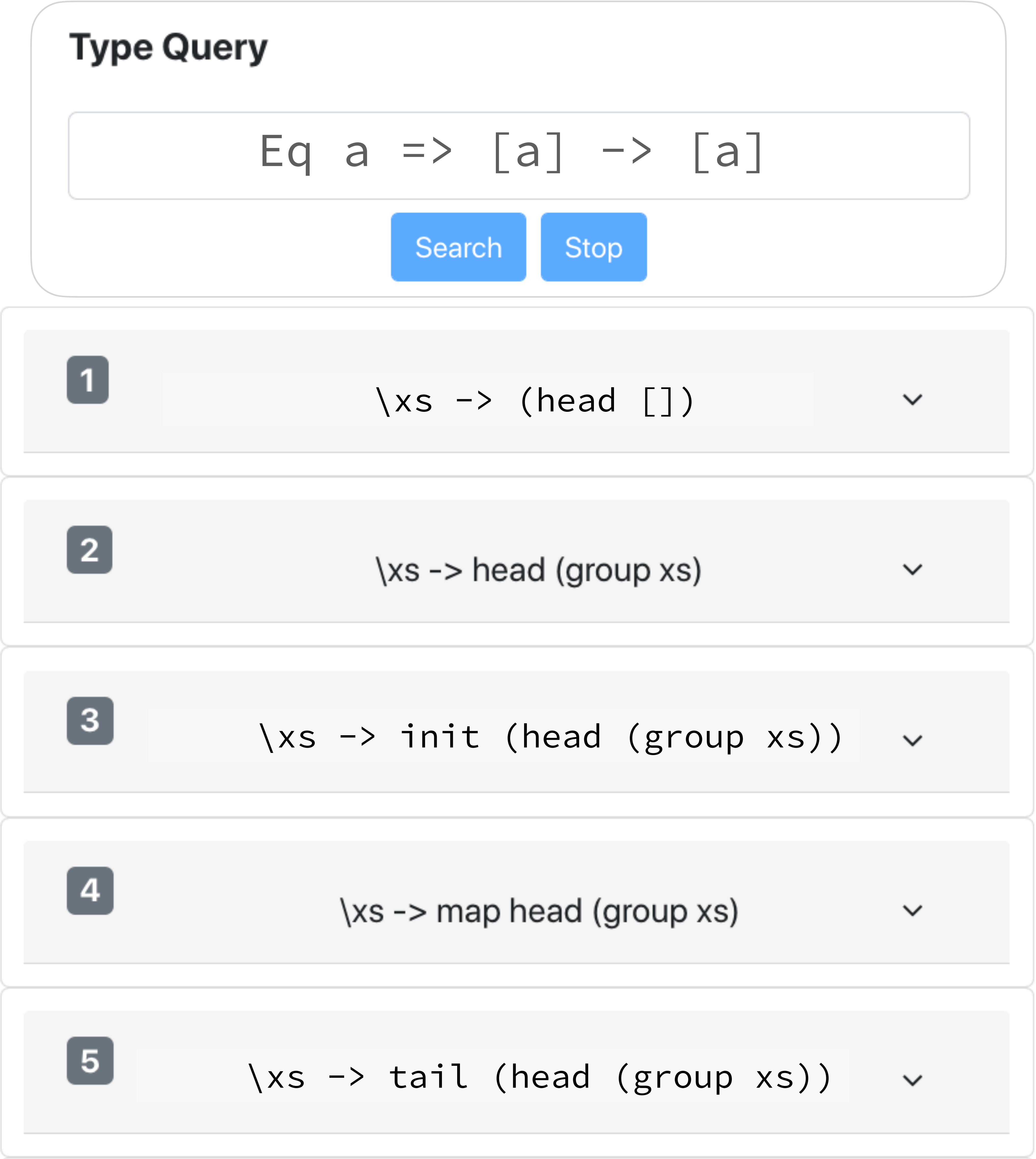
4 `\xs -> map head (group xs)`

5 `\xs -> tail (head (group xs))`

# Filtered Search

Challenge: How to filter irrelevant programs?

1. Test to produce output



Type Query

```
Eq a => [a] -> [a]
```

1 `\xs -> (head [])`

2 `\xs -> head (group xs)`

3 `\xs -> init (head (group xs))`

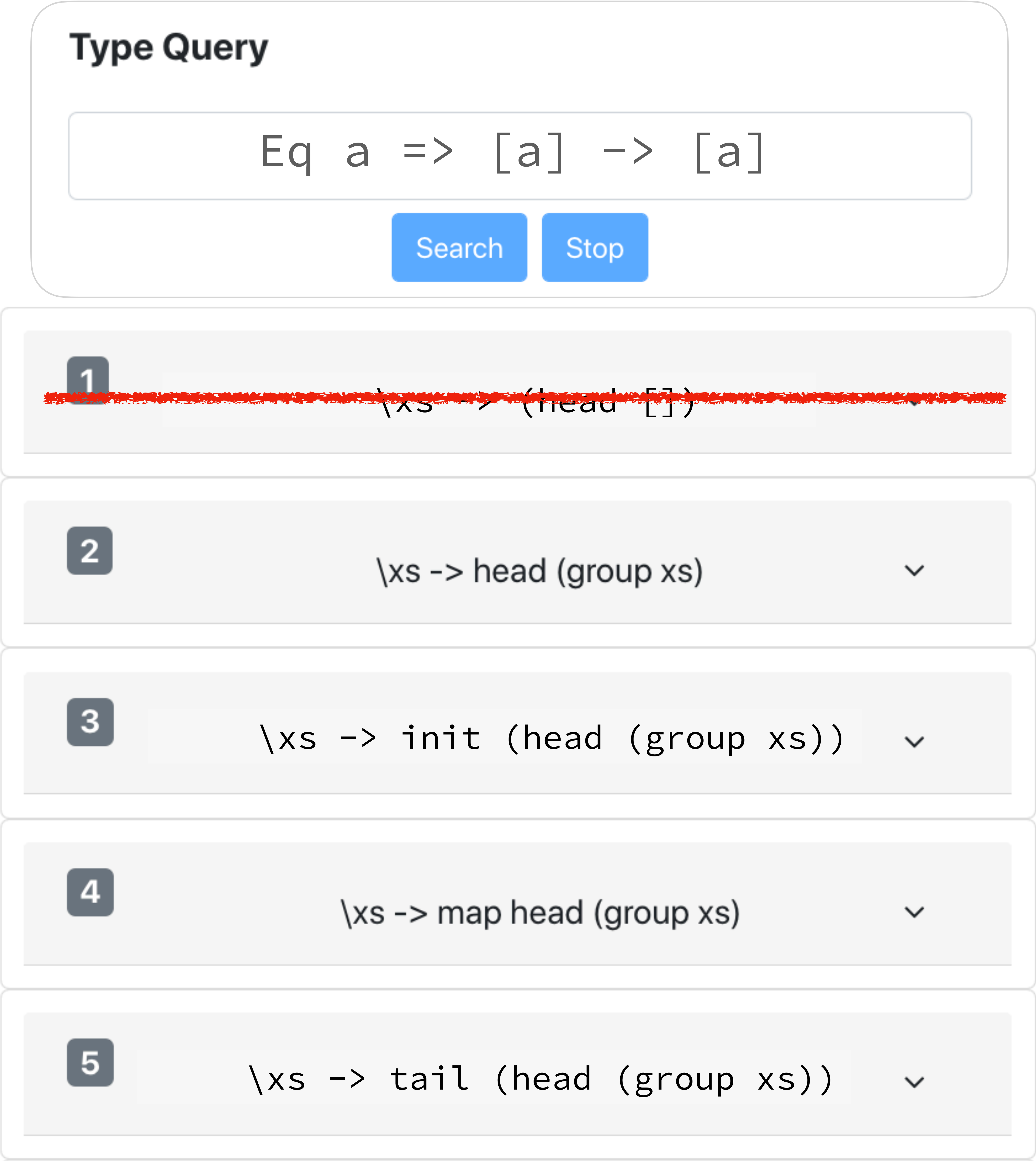
4 `\xs -> map head (group xs)`

5 `\xs -> tail (head (group xs))`

# Filtered Search

**Challenge: How to filter irrelevant programs?**

1. Test to produce output



Type Query

`Eq a => [a] -> [a]`

1 ~~`\xs -> (head [])`~~

2 `\xs -> head (group xs)`

3 `\xs -> init (head (group xs))`

4 `\xs -> map head (group xs)`

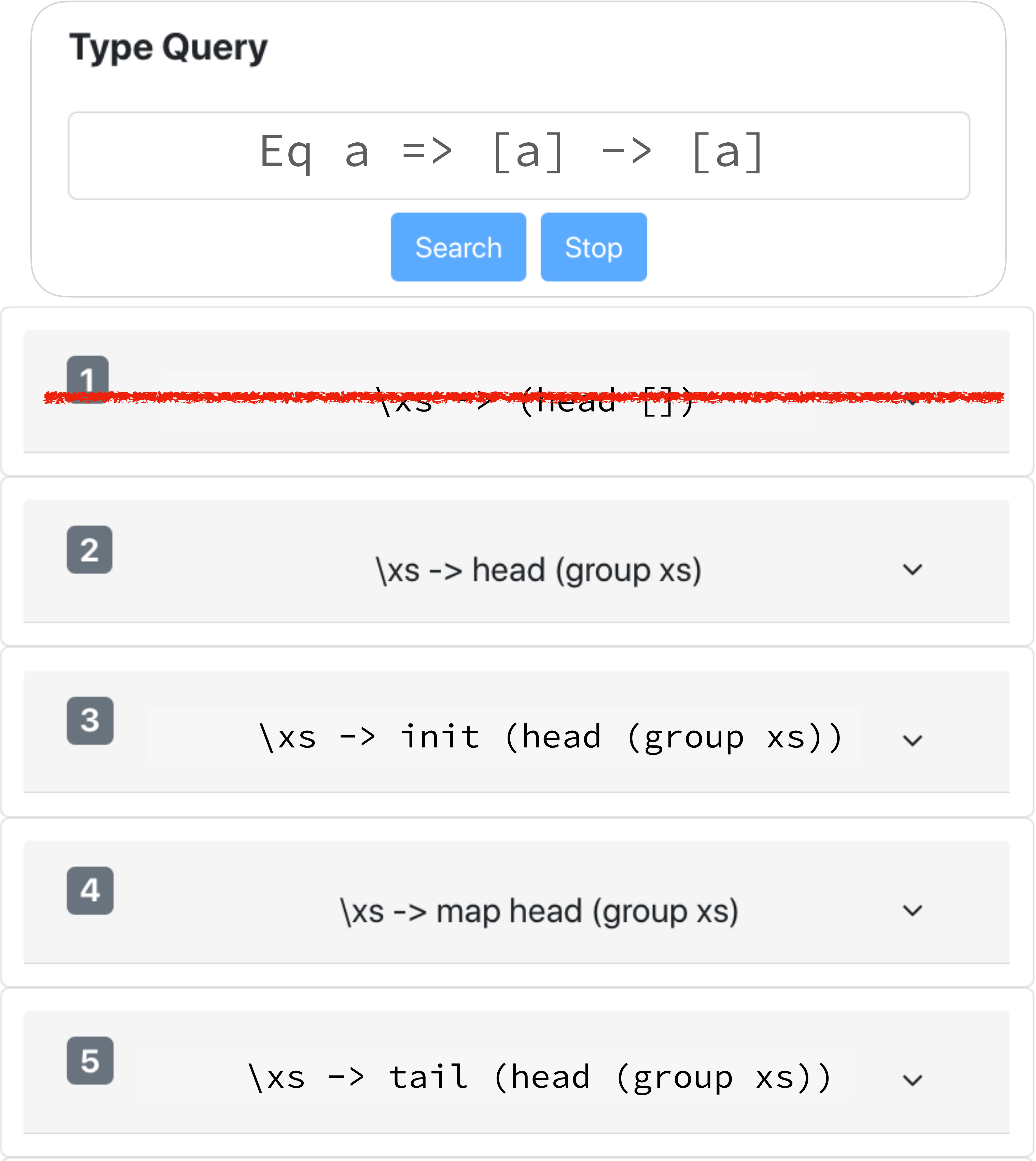
5 `\xs -> tail (head (group xs))`

# Filtered Search

**Challenge: How to filter irrelevant programs?**

1. Test to produce output

2. Test to distinguish



```
Eq a => [a] -> [a]
```

1 ~~`\xs -> (head [])`~~

2 `\xs -> head (group xs)`

3 `\xs -> init (head (group xs))`

4 `\xs -> map head (group xs)`

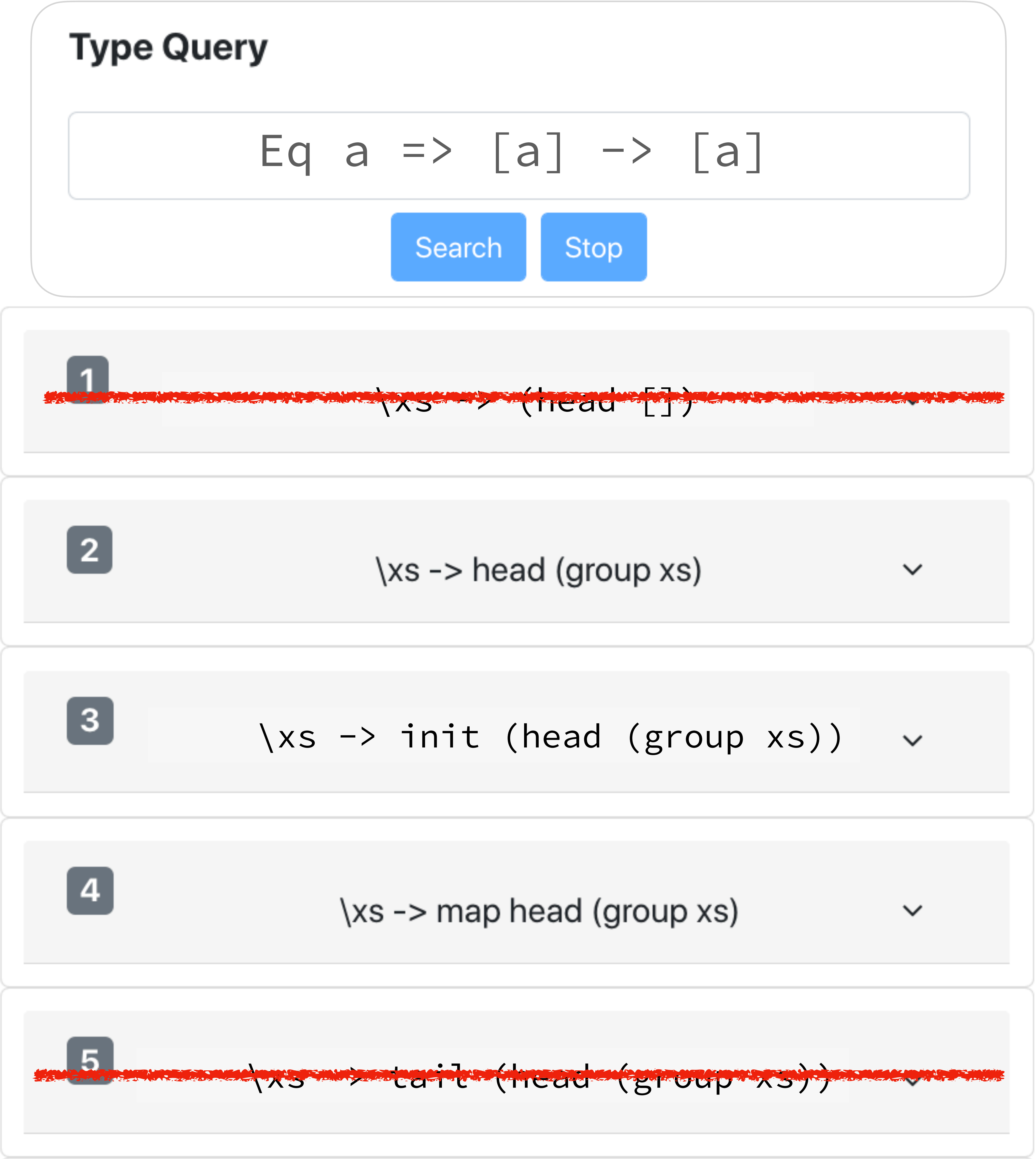
5 `\xs -> tail (head (group xs))`

# Filtered Search

**Challenge: How to filter irrelevant programs?**

1. Test to produce output

2. Test to distinguish



```
Eq a => [a] -> [a]
```

1 ~~\xs -> (head [])~~

2 \xs -> head (group xs)

3 \xs -> init (head (group xs))

4 \xs -> map head (group xs)

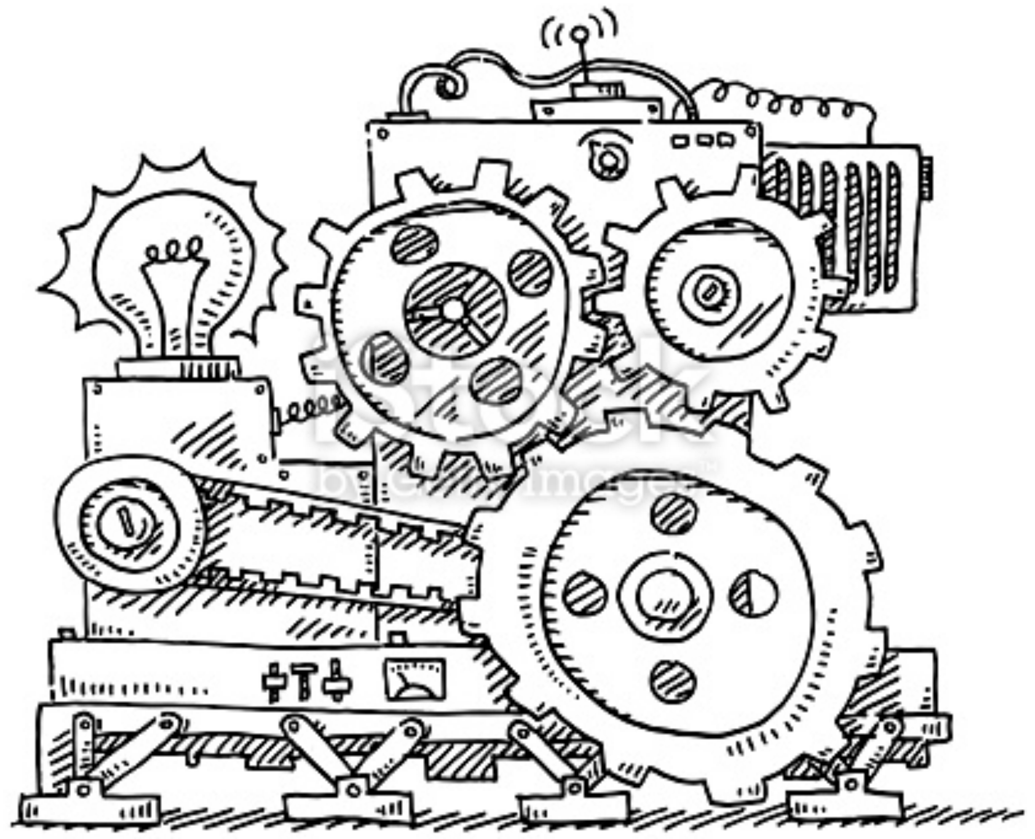
5 ~~\xs -> tail (head (group xs))~~

Specification

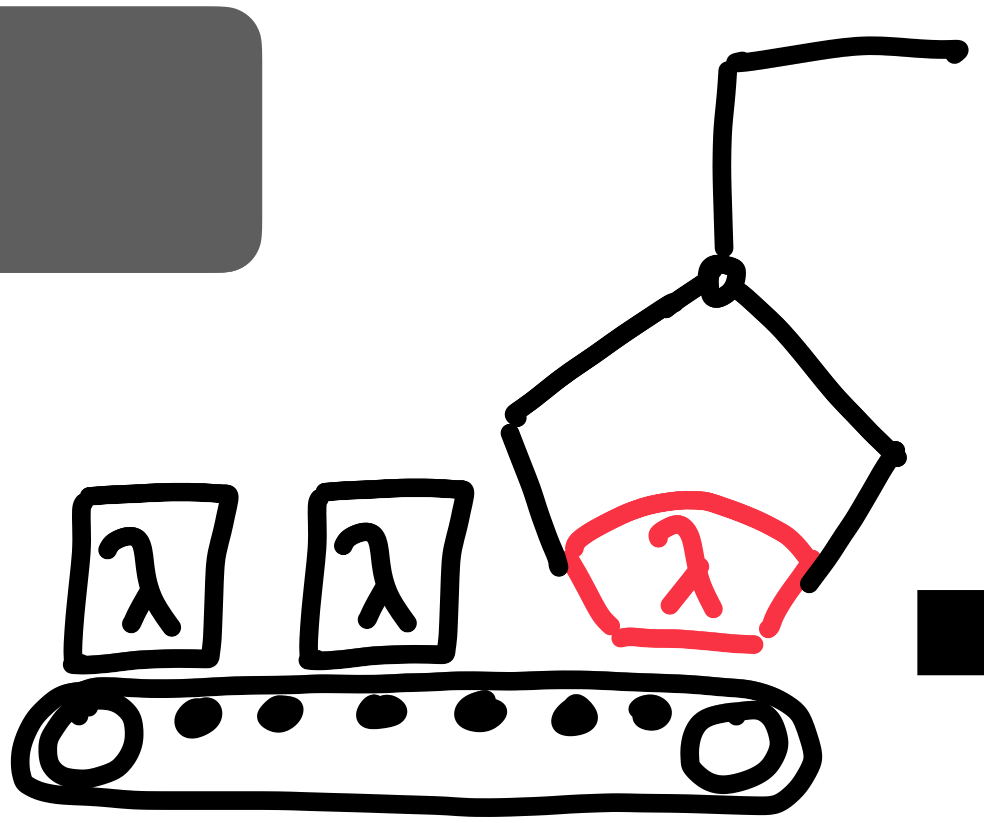
Filtering

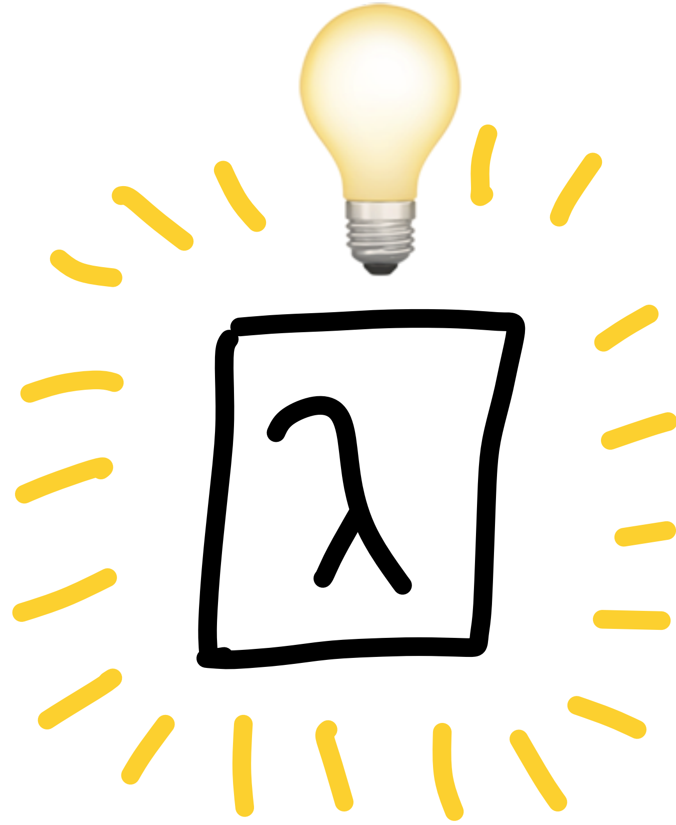
Comprehension

Type
or
Test



Program Synthesis by
Type-Guided Abstraction Refinement
[Guo et al. 2020]





Hoogle+ User Study

# Without aid

Challenge: How to help users pick their program?

Type Query: `dedup :: Eq a => [a] -> [a]`

Results:

```
1. \xs -> concat (group xs)
2. \xs -> head (group xs)
3. \xs -> last (group xs)
4. \xs -> map head (group xs)
```

# Without aid

Challenge: How to help users pick their program?

Type Query: `dedup :: Eq a => [a] -> [a]`

Are any right?

Results:

```
1. \xs -> concat (group xs)
2. \xs -> head (group xs)
3. \xs -> last (group xs)
4. \xs -> map head (group xs)
```

# Without aid

Challenge: How to help users pick their program?

Type Query:

```
dedup :: Eq a => [a] -> [a]
```

Are any right?

Results:

```
1. \xs -> concat (group xs)
2. \xs -> head (group xs)
3. \xs -> last (group xs)
4. \xs -> map head (group xs)
```

How are they different?

# Without aid

Challenge: How to help users pick their program?

Type Query: `dedup :: Eq a => [a] -> [a]`

Are any right?

Results:

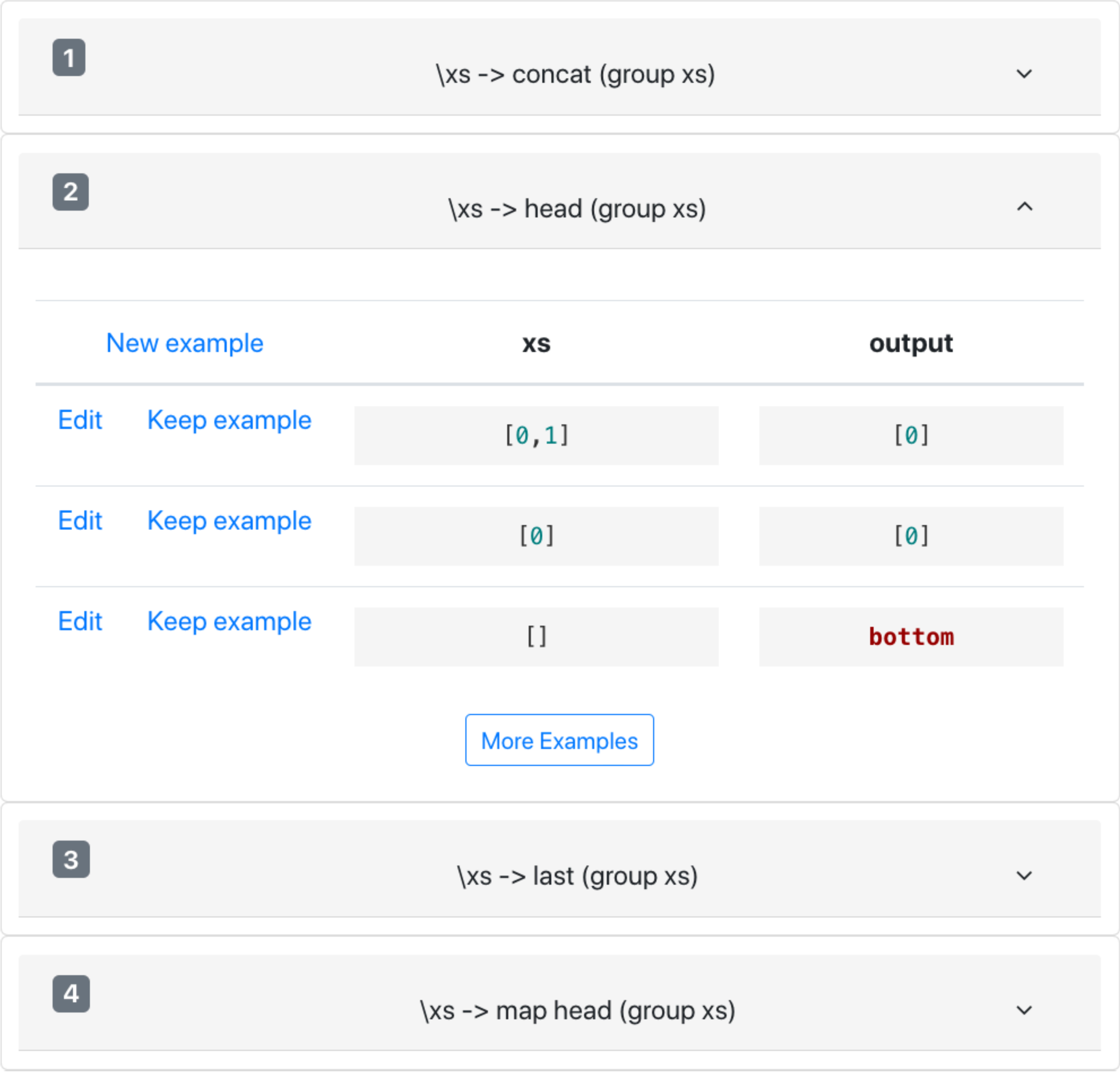
```
1. \xs -> concat (group xs)
2. \xs -> head (group xs)
3. \xs -> last (group xs)
4. \xs -> map head (group xs)
```

How are they different?

What about edge cases?

# Hoogle+’s UI

Type Query: `Eq a => [a] -> [a]`



1 \xs -> concat (group xs)

2 \xs -> head (group xs)

| New example | | xs | output |
|---|---|---|---|
| Edit | Keep example | [0,1] | [0] |
| Edit | Keep example | [0] | [0] |
| Edit | Keep example | [] | bottom |

More Examples

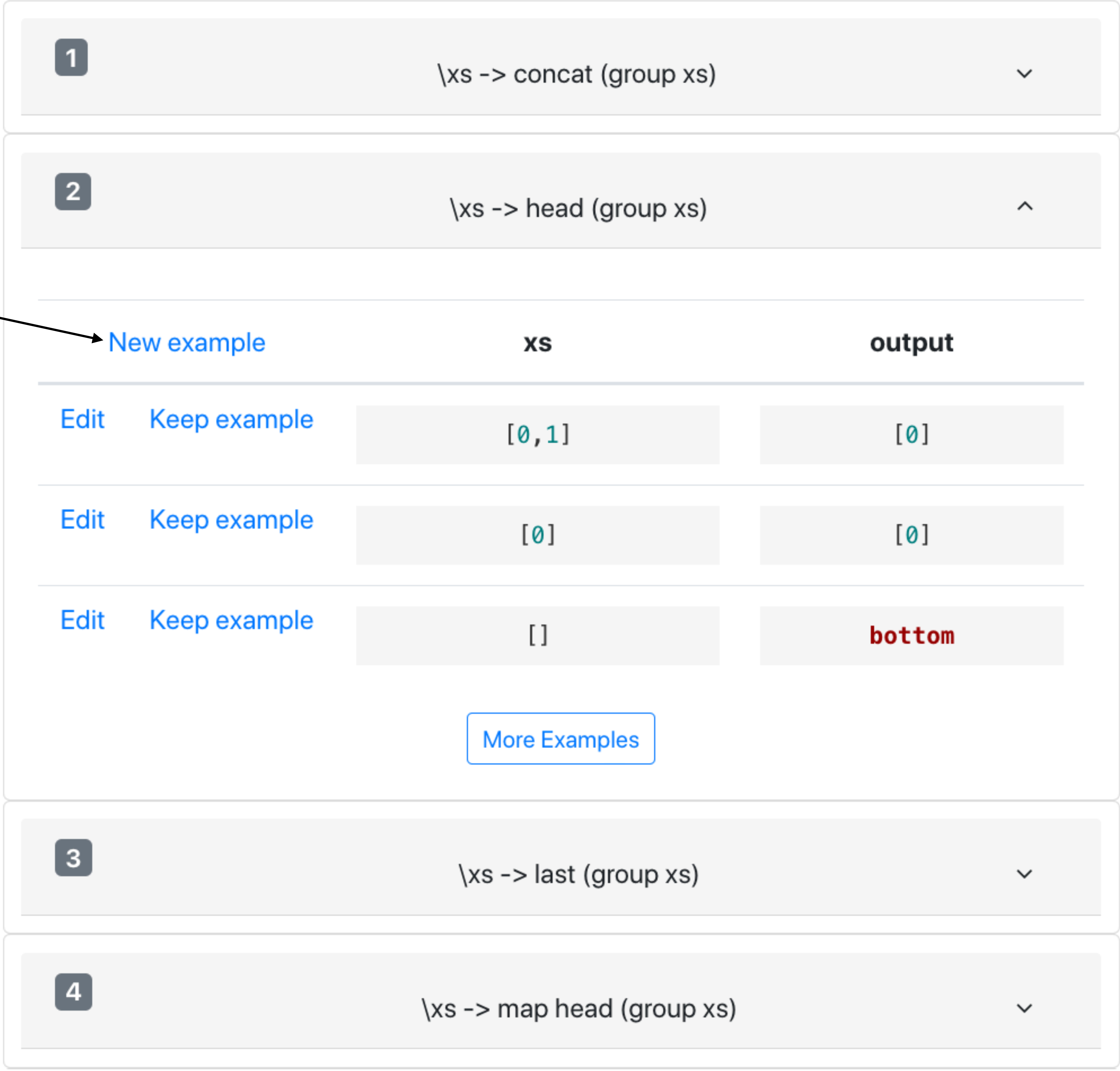
3 \xs -> last (group xs)

4 \xs -> map head (group xs)

# Hoogle+’s UI

User-Provided Example

Type Query: `Eq a => [a] -> [a]`



1 \xs -> concat (group xs)

2 \xs -> head (group xs)

| New example | xs | output |
|---|---|---|
| Edit Keep example | `[0,1]` | `[0]` |
| Edit Keep example | `[0]` | `[0]` |
| Edit Keep example | `[]` | **bottom** |

More Examples

3 \xs -> last (group xs)

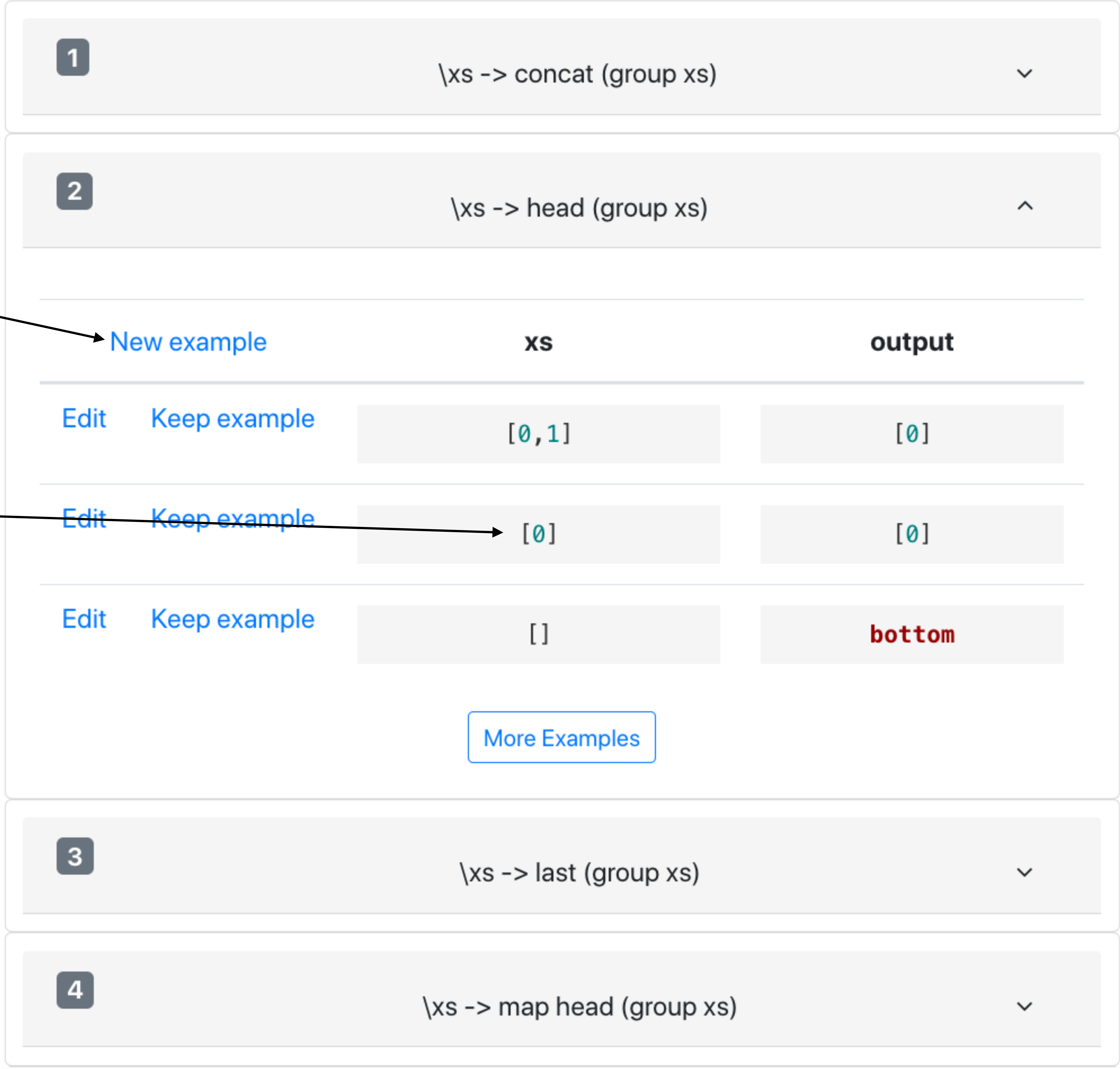
4 \xs -> map head (group xs)

# Hoogle+'s UI

Type Query: `Eq a => [a] -> [a]`

User-Provided Example

Generated Examples



1 \xs -> concat (group xs)

2 \xs -> head (group xs)

| New example | xs | output |
|---|---|---|
| Edit Keep example | `[0,1]` | `[0]` |
| Edit Keep example | `[0]` | `[0]` |
| Edit Keep example | `[]` | `bottom` |

3 \xs -> last (group xs)

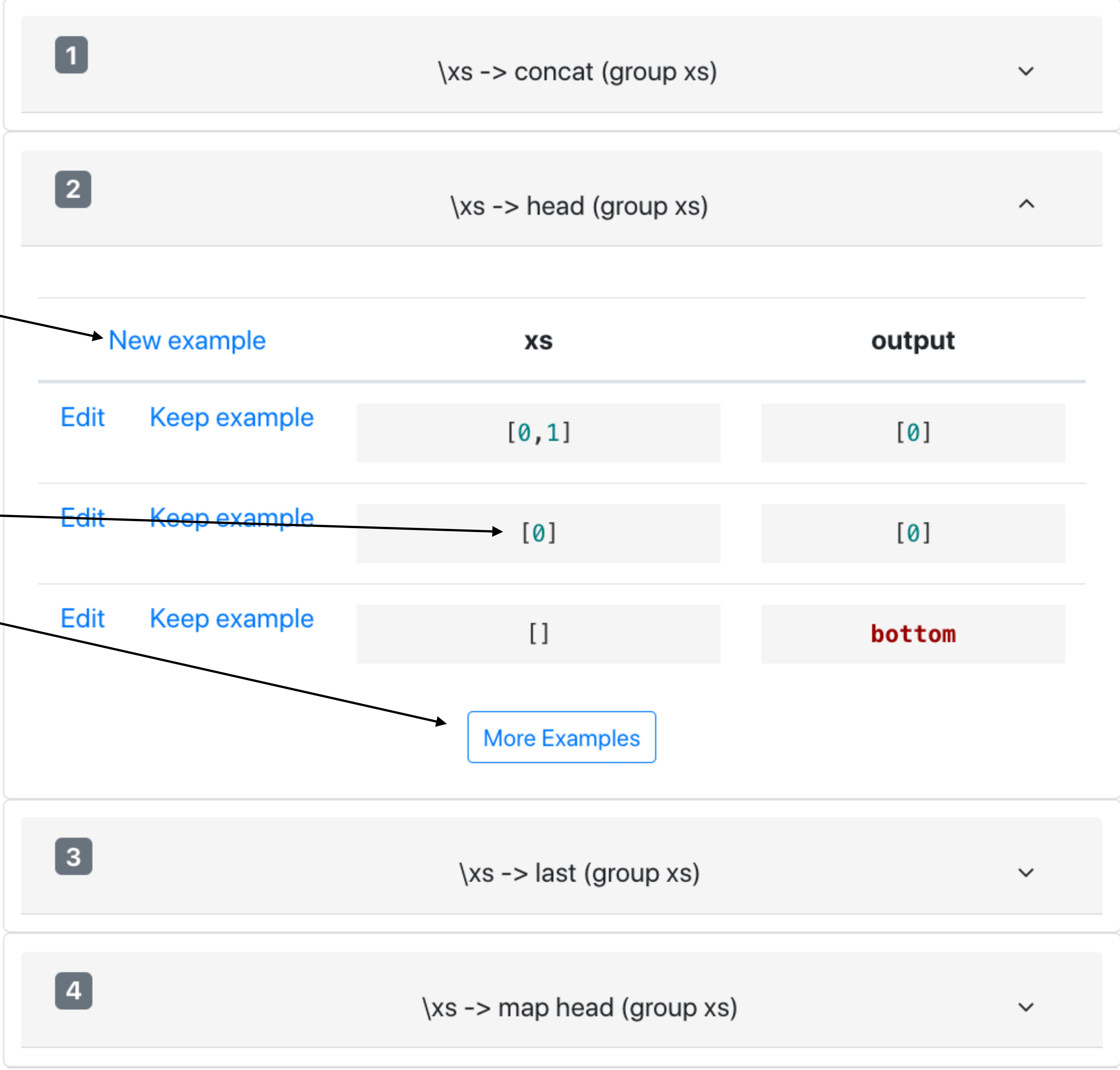
4 \xs -> map head (group xs)

# Hoogle+'s UI

Type Query: `Eq a => [a] -> [a]`

User-Provided Example

Generated Examples



1 \xs -> concat (group xs)

2 \xs -> head (group xs)

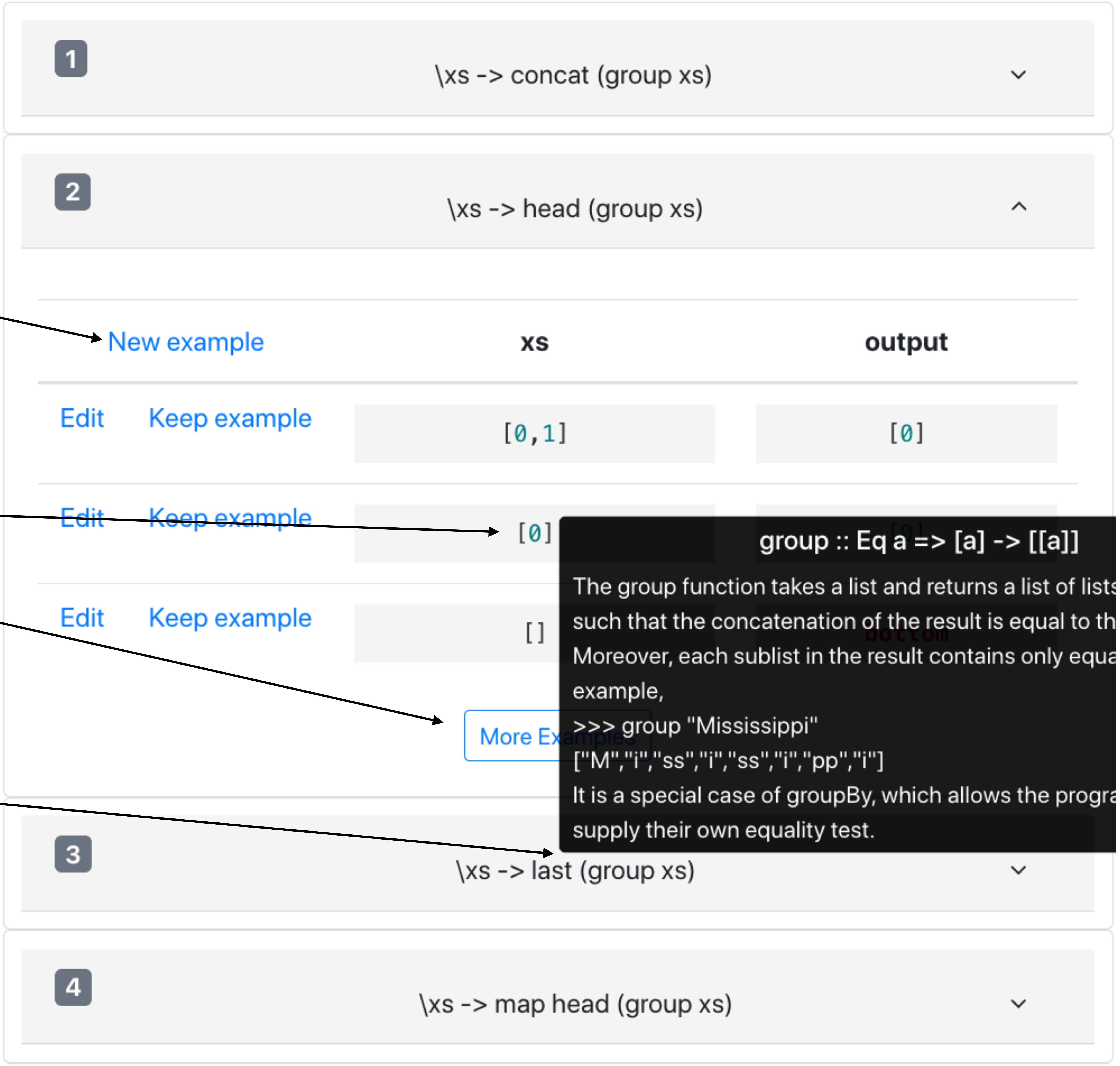
| New example | xs | output |
|---|---|---|
| Edit Keep example | `[0,1]` | `[0]` |
| Edit Keep example | `[0]` | `[0]` |
| Edit Keep example | `[]` | `bottom` |

3 \xs -> last (group xs)

4 \xs -> map head (group xs)

# Hoogle+’s UI

Type Query: `Eq a => [a] -> [a]`



1 \xs -> concat (group xs)

2 \xs -> head (group xs)

| New example | xs | output |
|---|---|---|
| Edit Keep example | `[0,1]` | `[0]` |
| Edit Keep example | `[0]` | `[0]` |
| Edit Keep example | `[]` | `bottom` |

More Examples

group :: Eq a => [a] -> [[a]]

The group function takes a list and returns a list of lists
such that the concatenation of the result is equal to th
Moreover, each sublist in the result contains only equa
example,

```
>>> group "Mississippi"
["M","i","ss","i","ss","i","pp","i"]
```

It is a special case of groupBy, which allows the progra
supply their own equality test.

3 \xs -> last (group xs)

4 \xs -> map head (group xs)

User-Provided Example

Generated Examples

Documentation

Specification

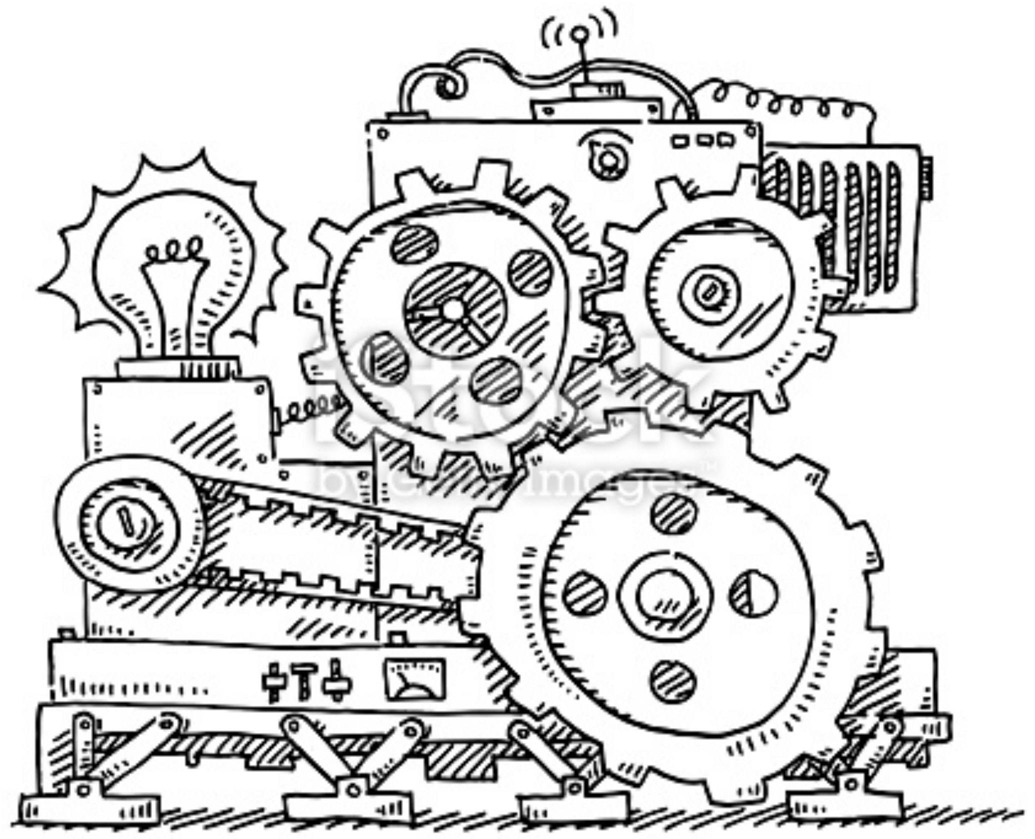
Filtering

Comprehension

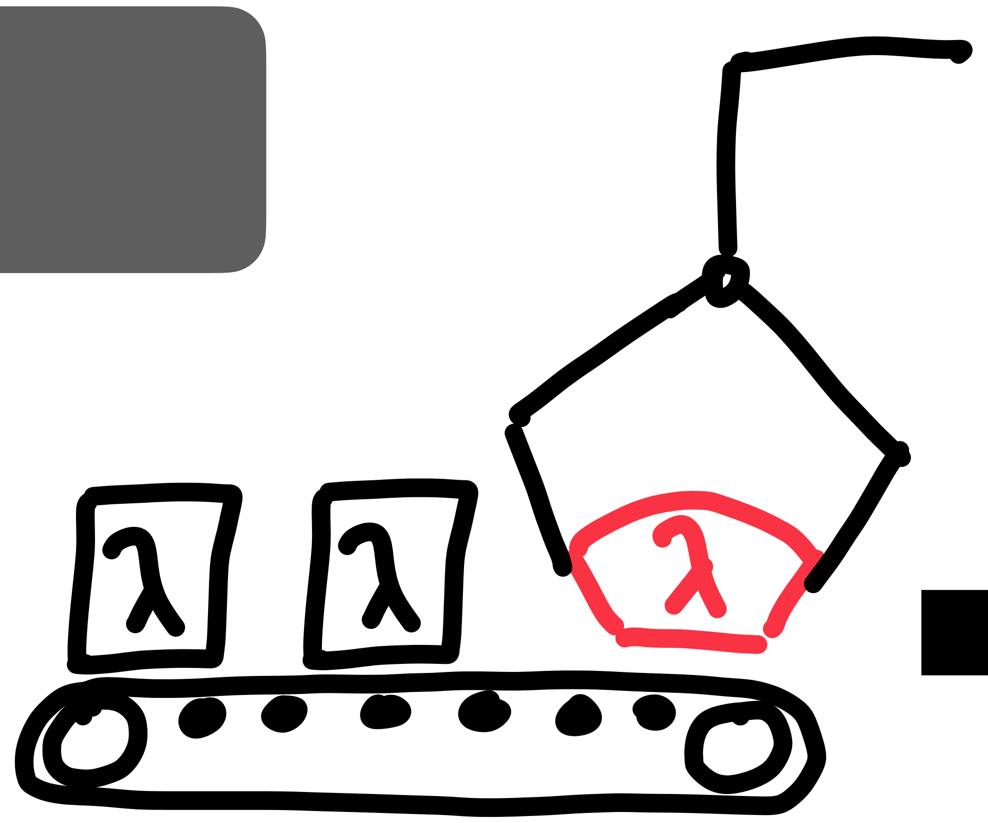
Type

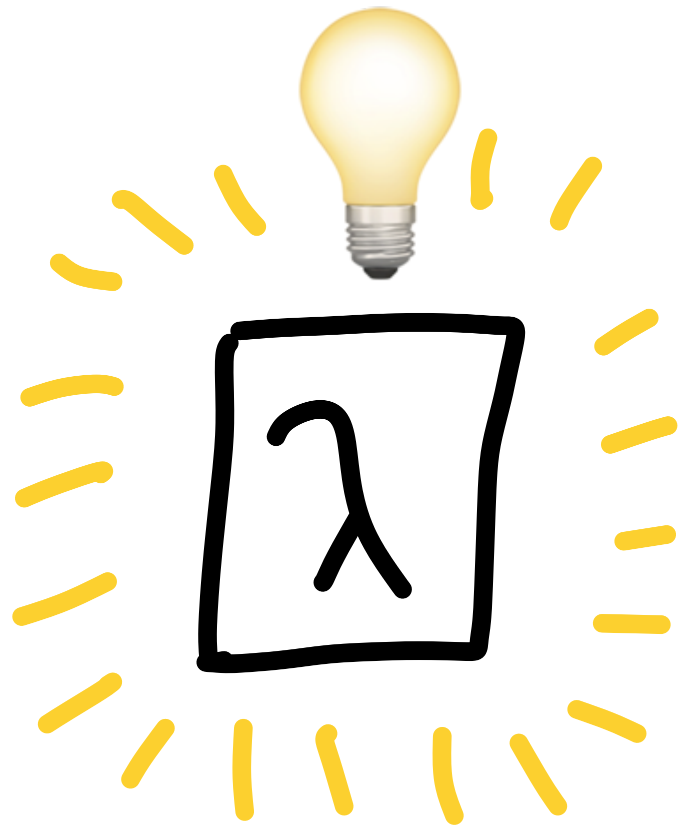
or

Test



Program Synthesis by
Type-Guided Abstraction Refinement
[Guo et al. 2020]





Hoogle+ User Study

# User Study

# User Study

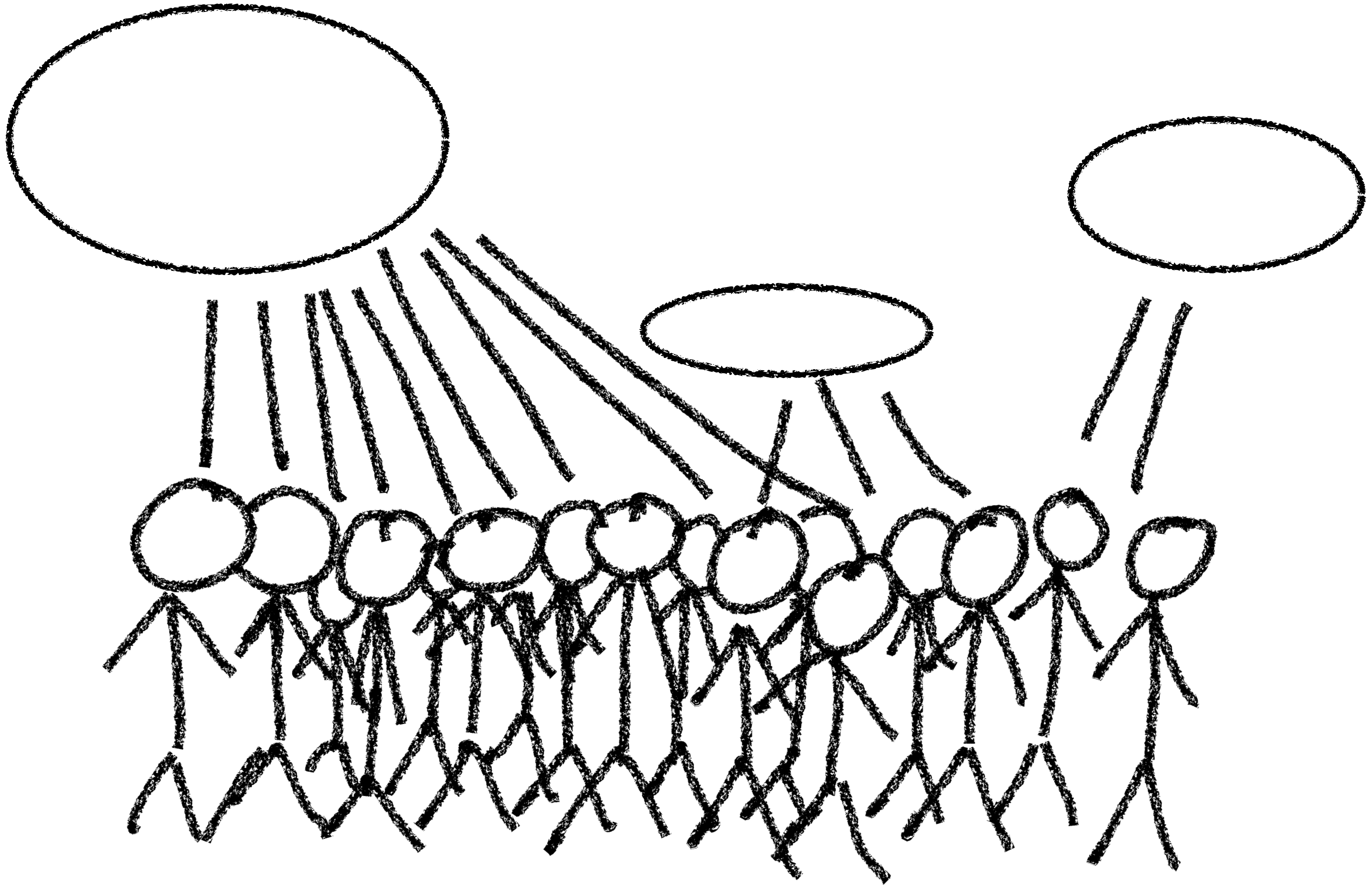
RQ 1 Does our synthesizer help functional programmers solve their program search tasks, compared to traditional methods?

# User Study

| | |
|---|---|
| RQ 1 | Does our synthesizer help functional programmers solve their program search tasks, compared to traditional methods? |
| RQ 2 | How do Hoogle+ users specify their search intent? |

# User Study

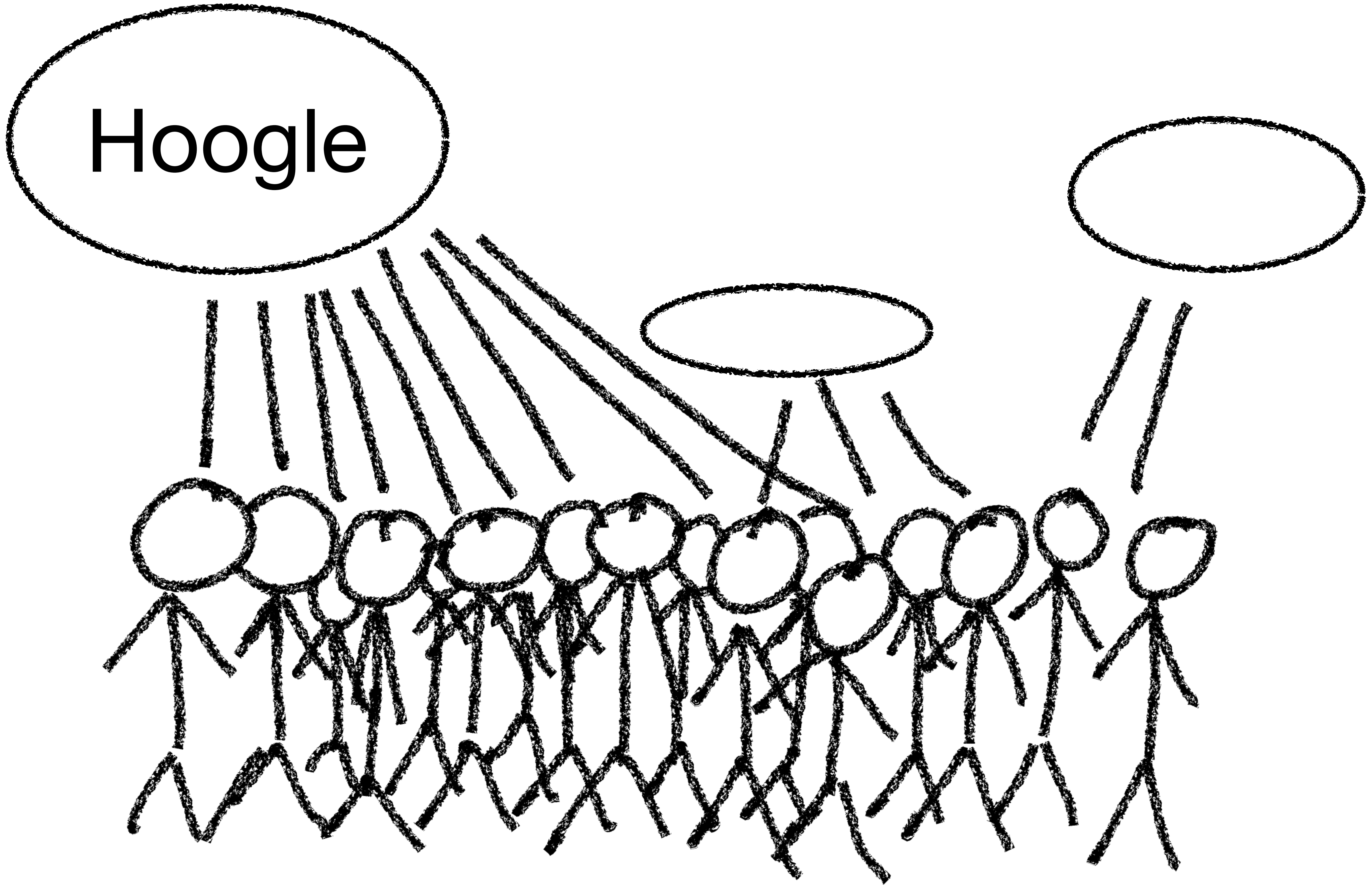
What are your traditional methods for code snippet searches?



150 Haskellers

# User Study

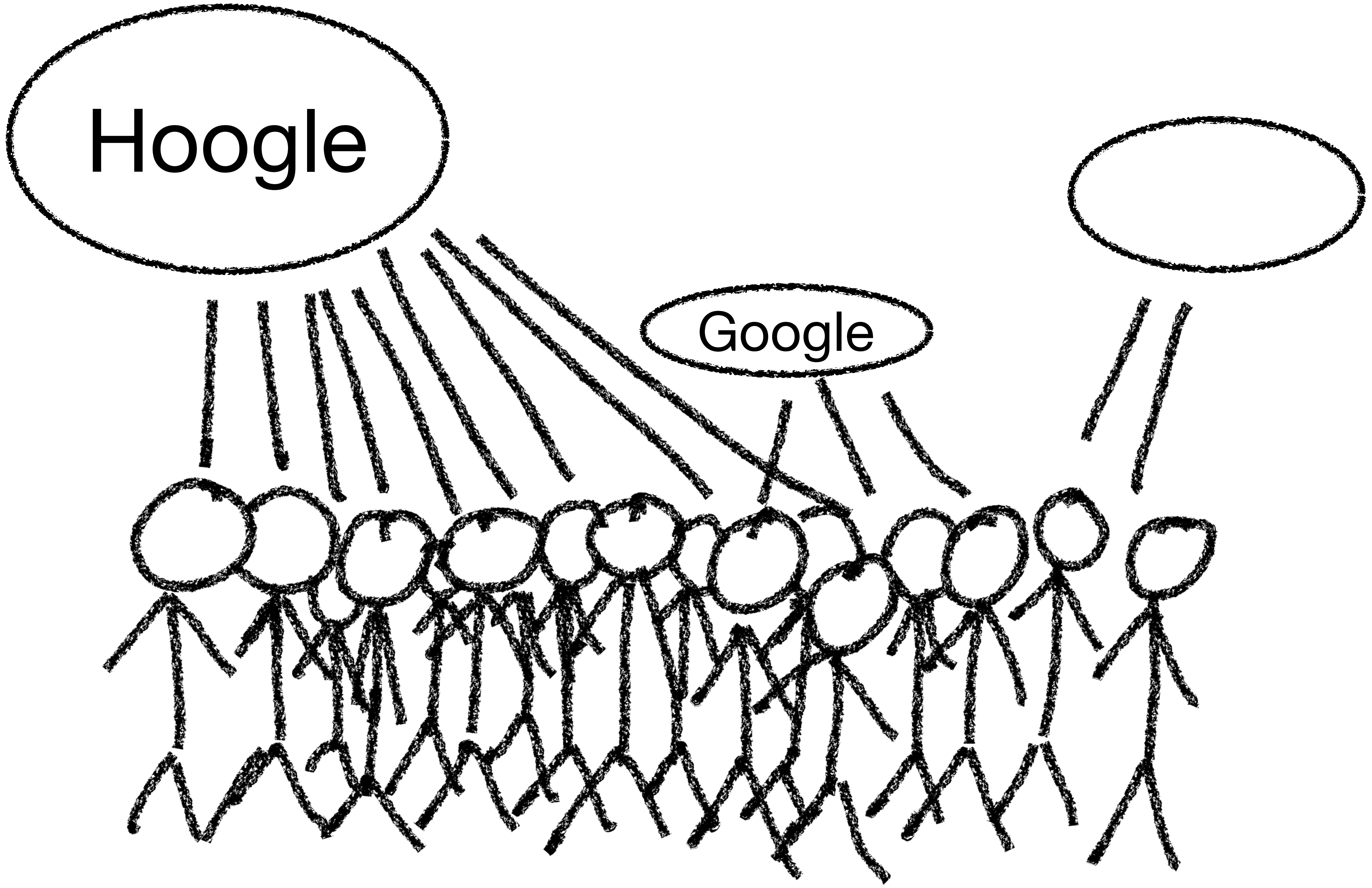
What are your traditional methods for code snippet searches?



150 Haskellers

# User Study

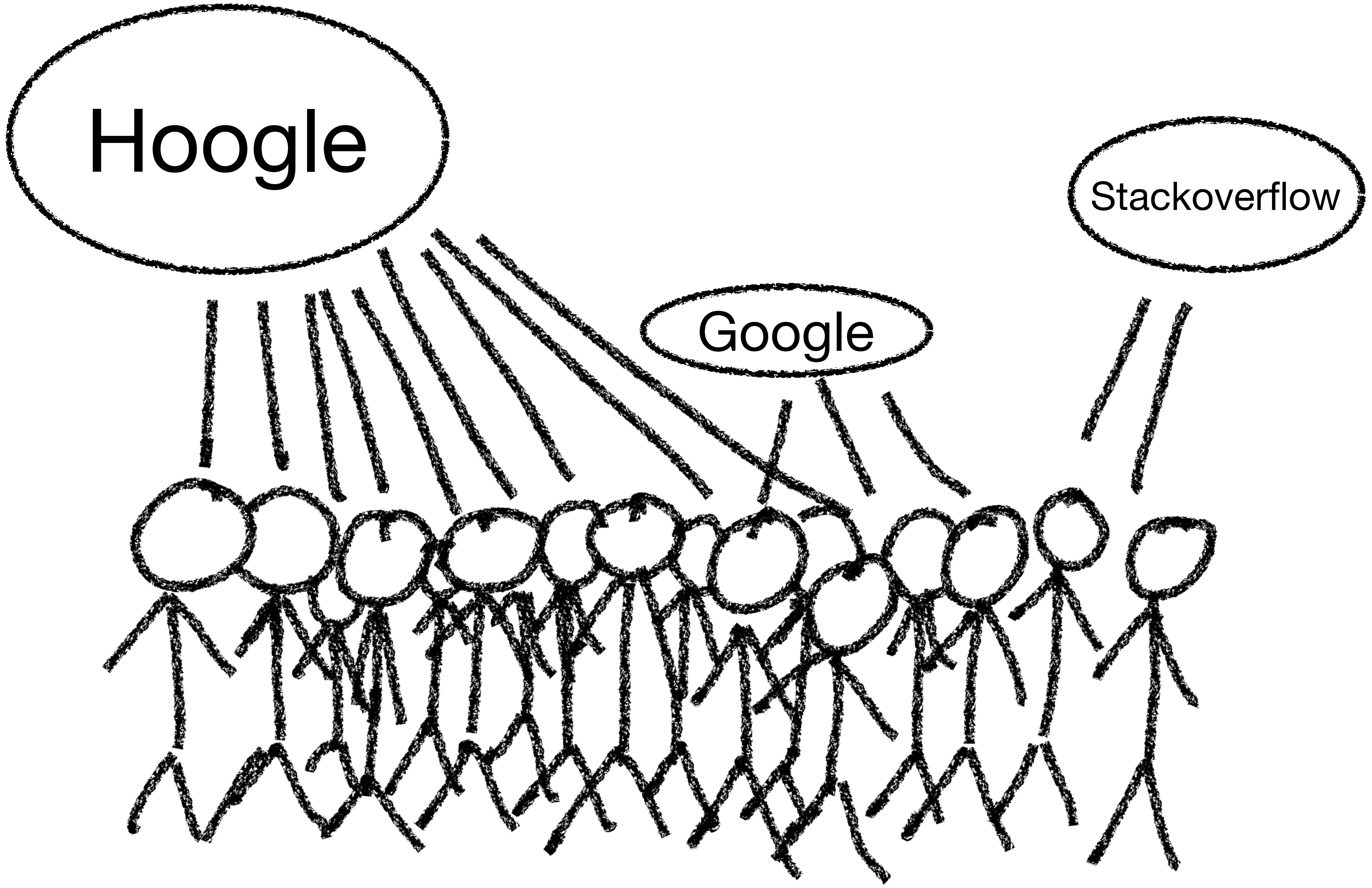
What are your traditional methods for code snippet searches?



150 Haskellers

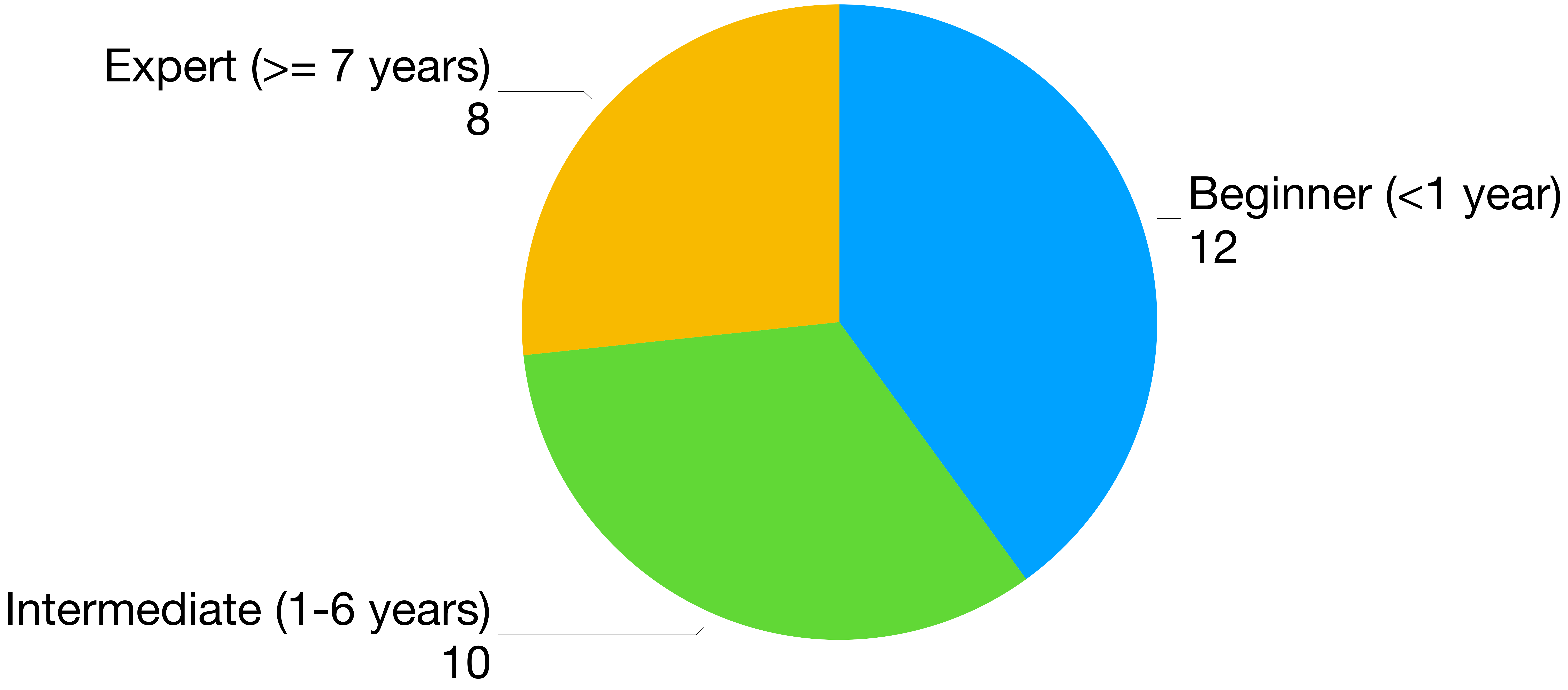
# User Study

What are your
traditional methods
for code snippet searches?



150 Haskellers

# 30 Participants



# What is a task?

# What is a task?

Description:

```
Function dedup takes ...
```

# What is a task?

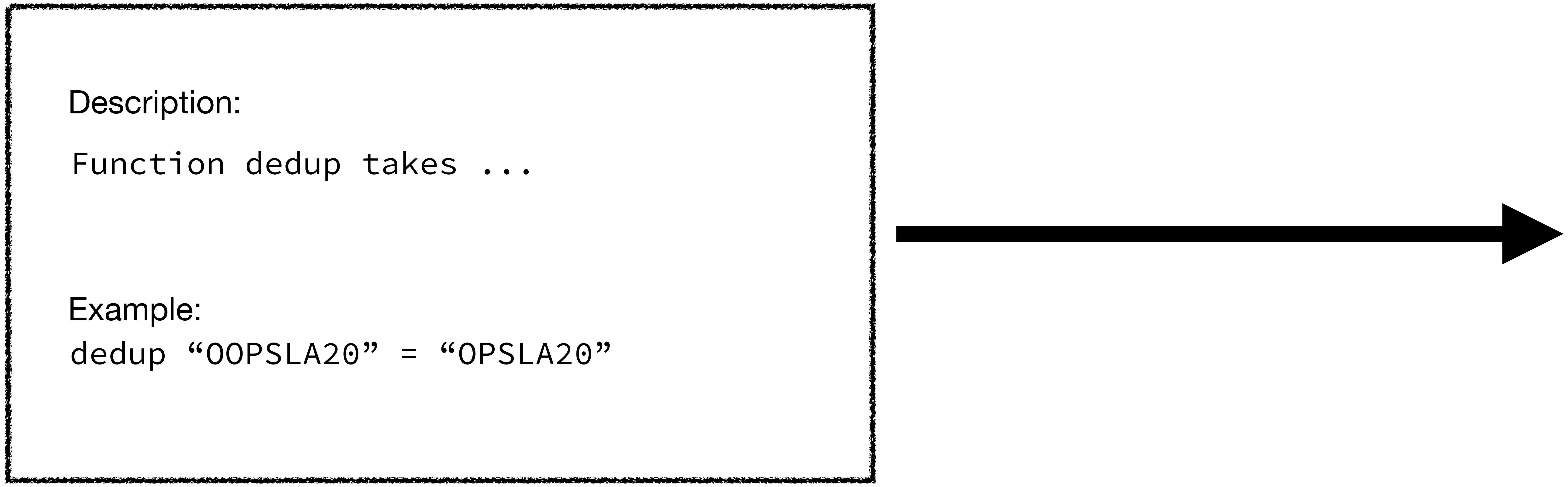
Description:

```
Function dedup takes ...
```

Example:

```
dedup “OOPSLA20” = “OPSLA20”
```

# What is a task?



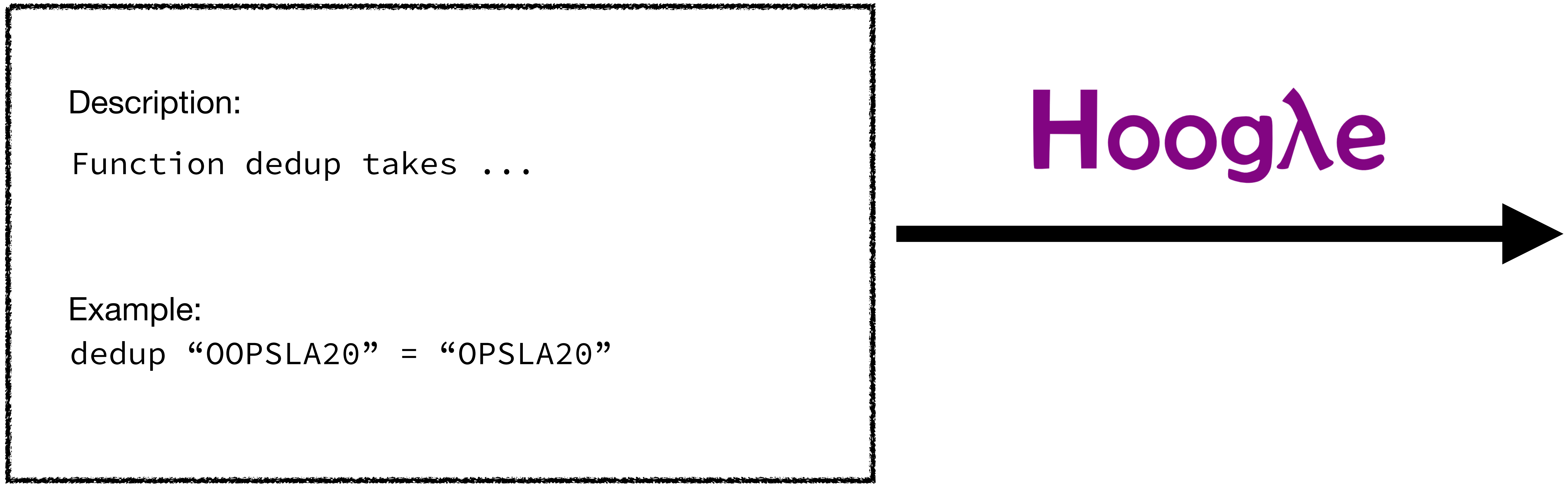
Description:

```
Function dedup takes ...
```

Example:

```
dedup “OOPSLA20” = “OPSLA20”
```

# What is a task?



Description:

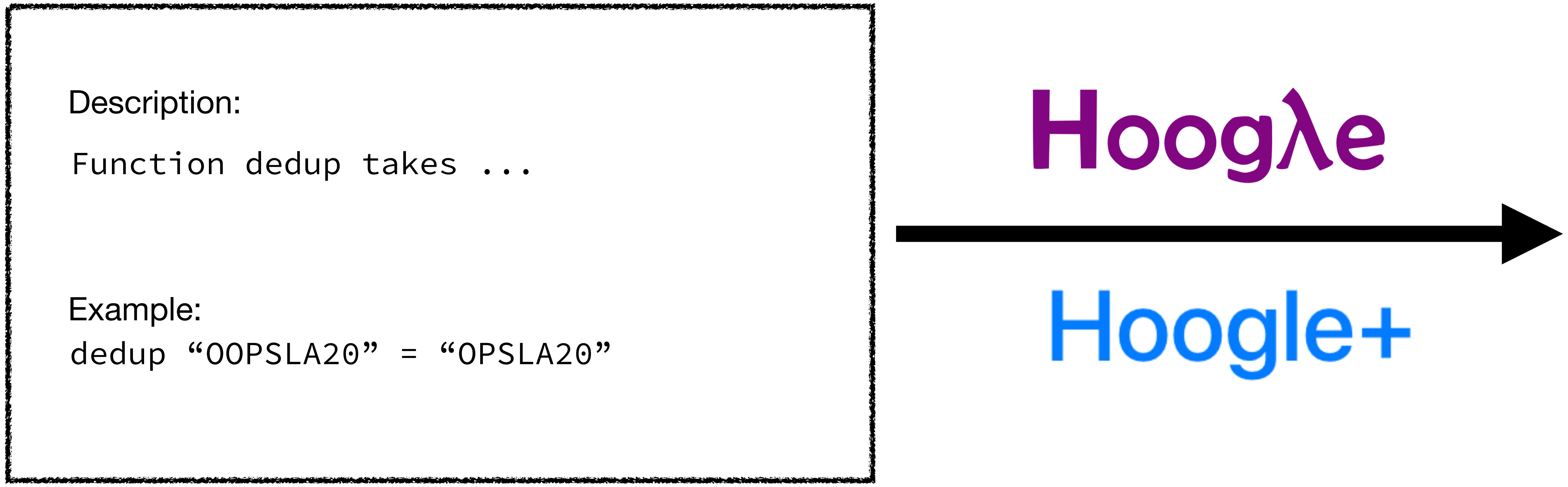
```
Function dedup takes ...
```

Example:

```
dedup “OOPSLA20” = “OPSLA20”
```

Hoogλe

# What is a task?



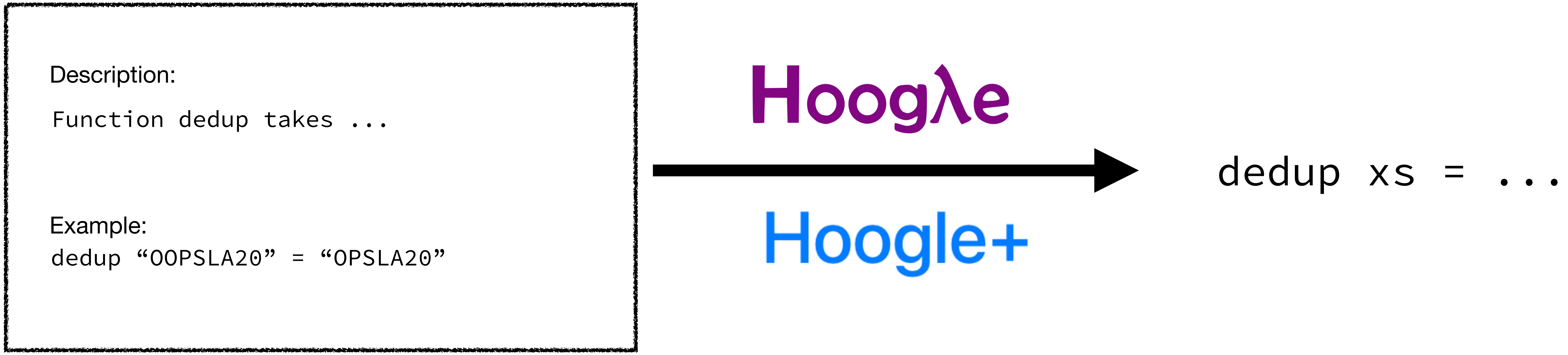
Description:

```
Function dedup takes ...
```

Example:

```
dedup “OOPSLA20” = “OPSLA20”
```

# What is a task?



Description:

```
Function dedup takes ...
```

Example:

```
dedup “OOPSLA20” = “OPSLA20”
```

Hoogλe

Hoogle+

```
dedup xs = ...
```

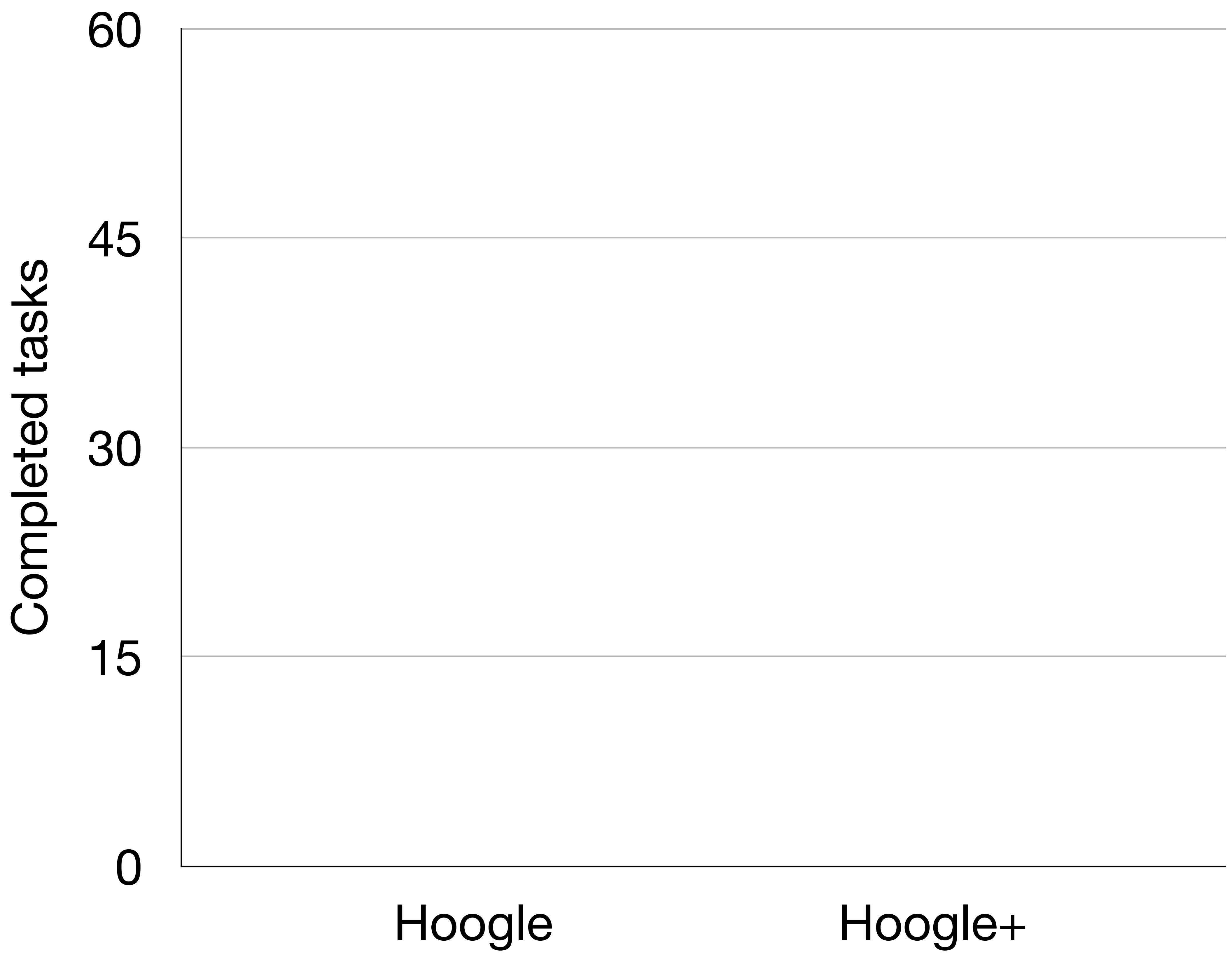
# Results

Completion Rate

Time-to-complete

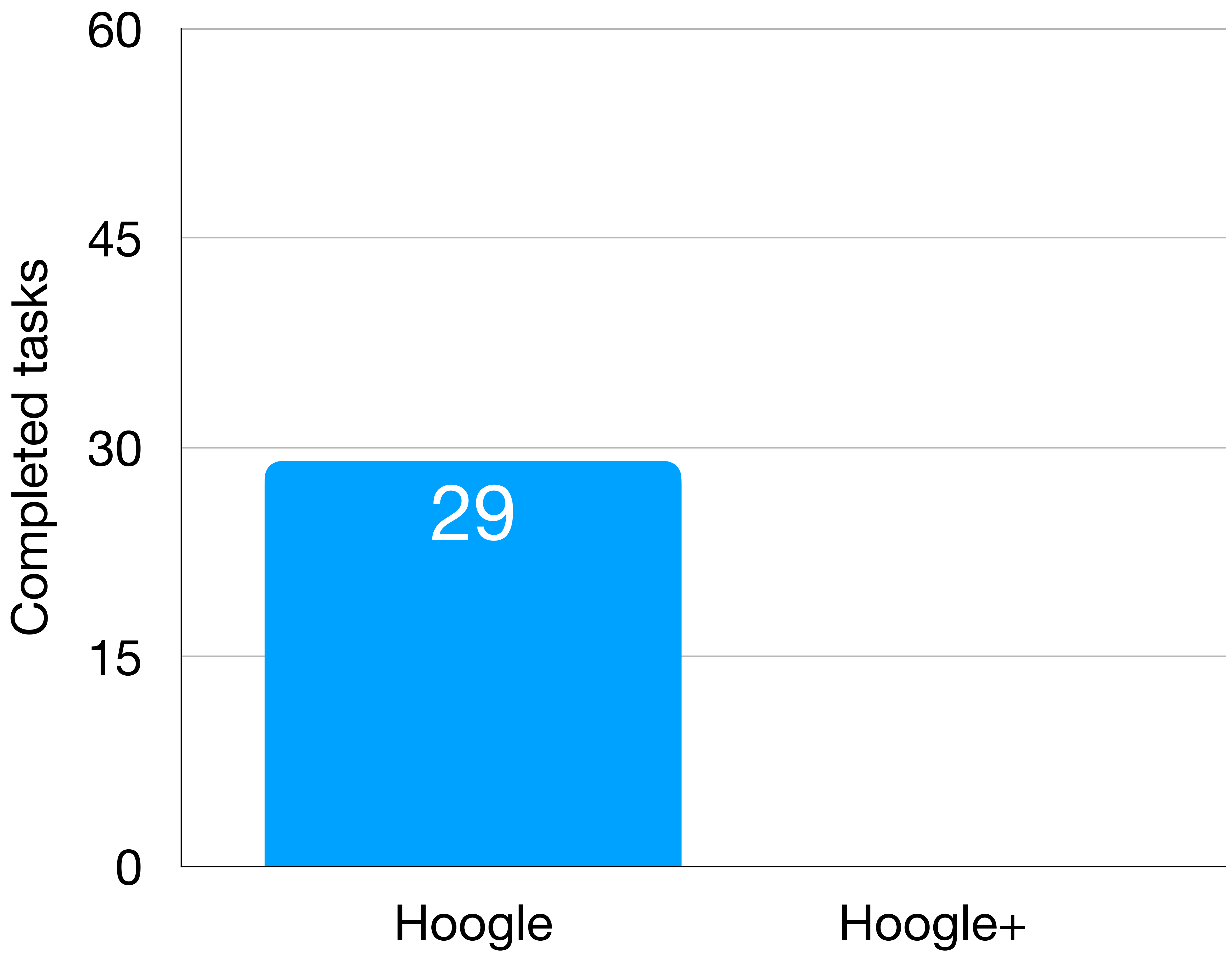
# Results

## Completion Rate



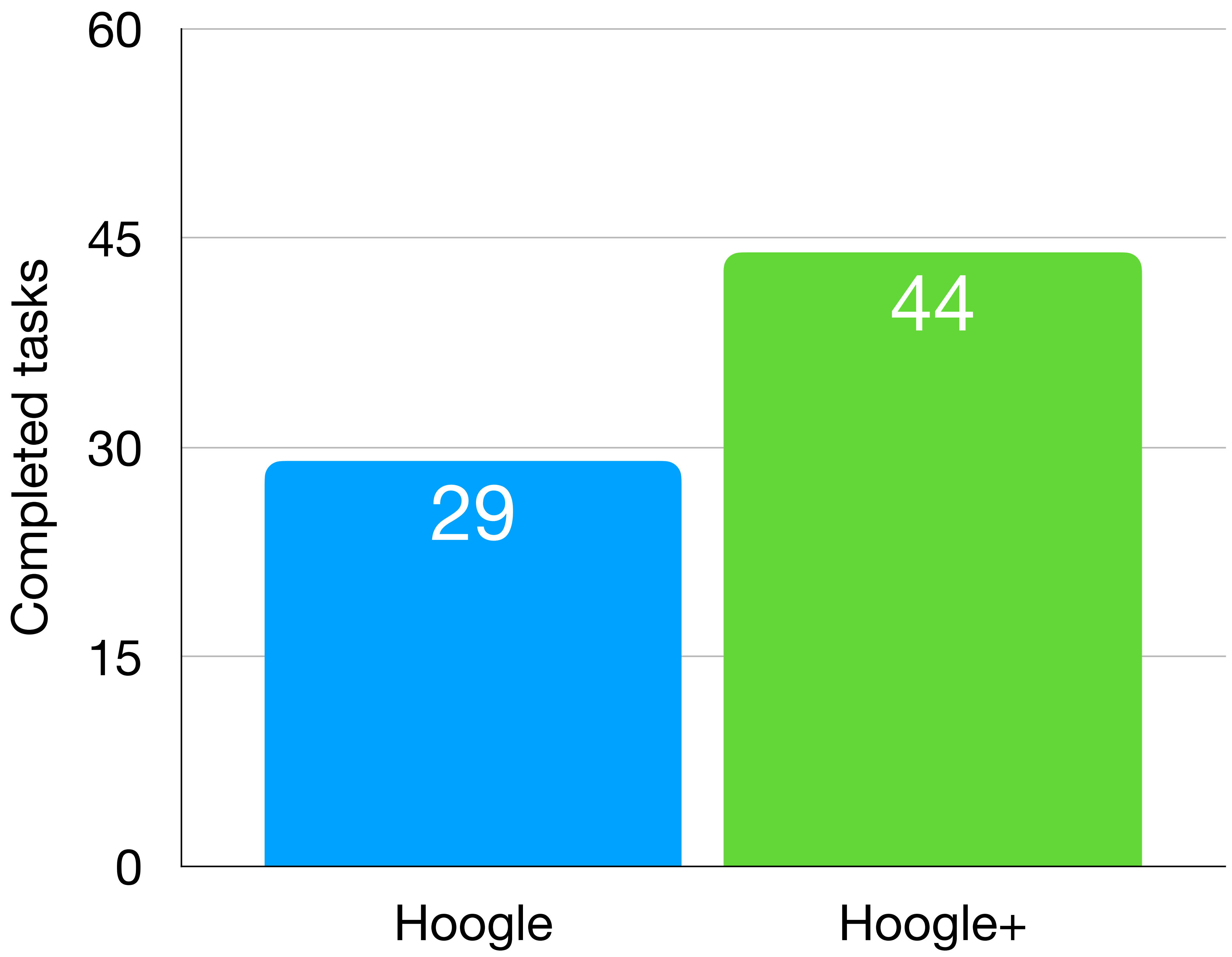
# Results

## Completion Rate



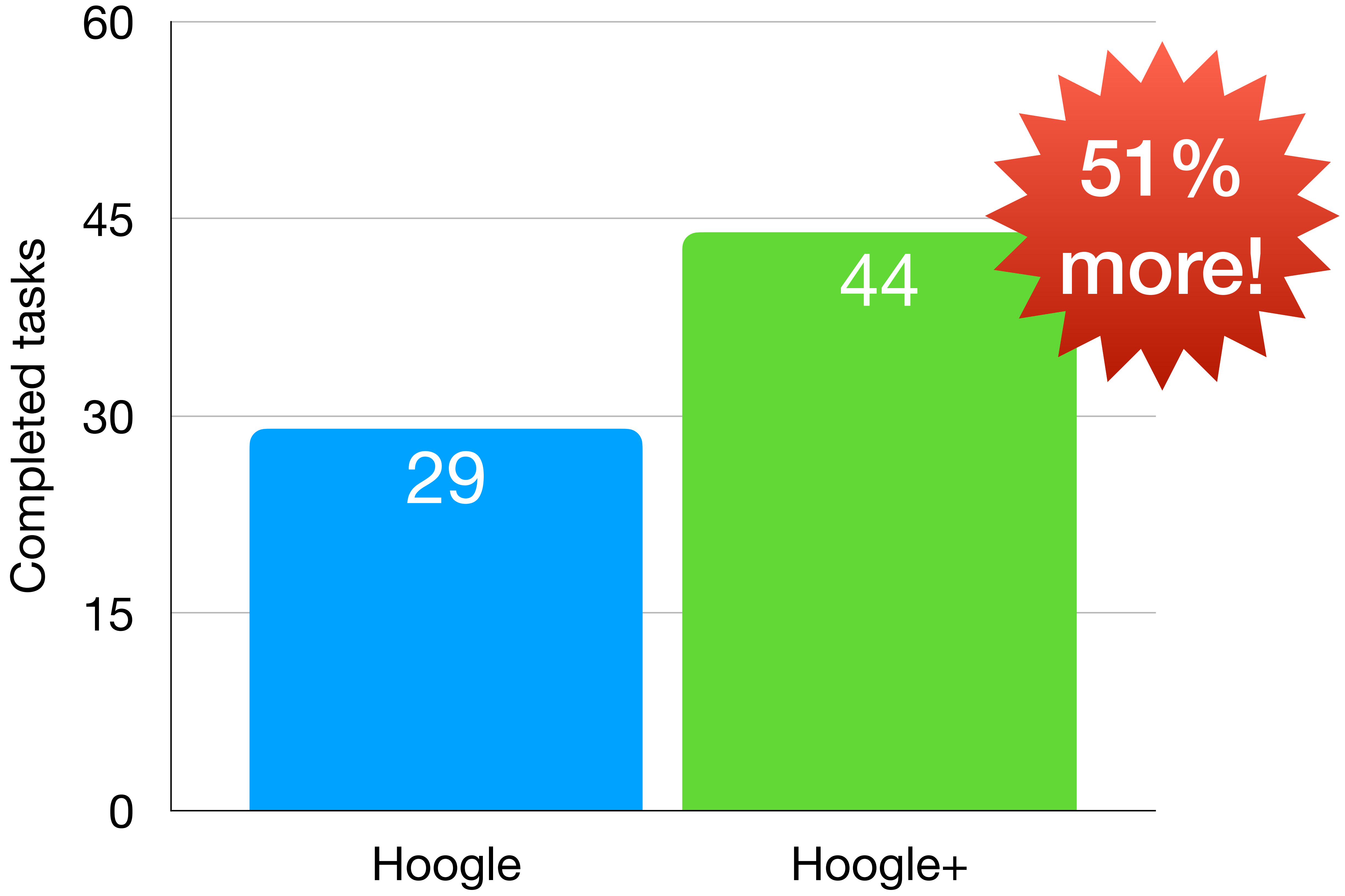
# Results

## Completion Rate



# Results

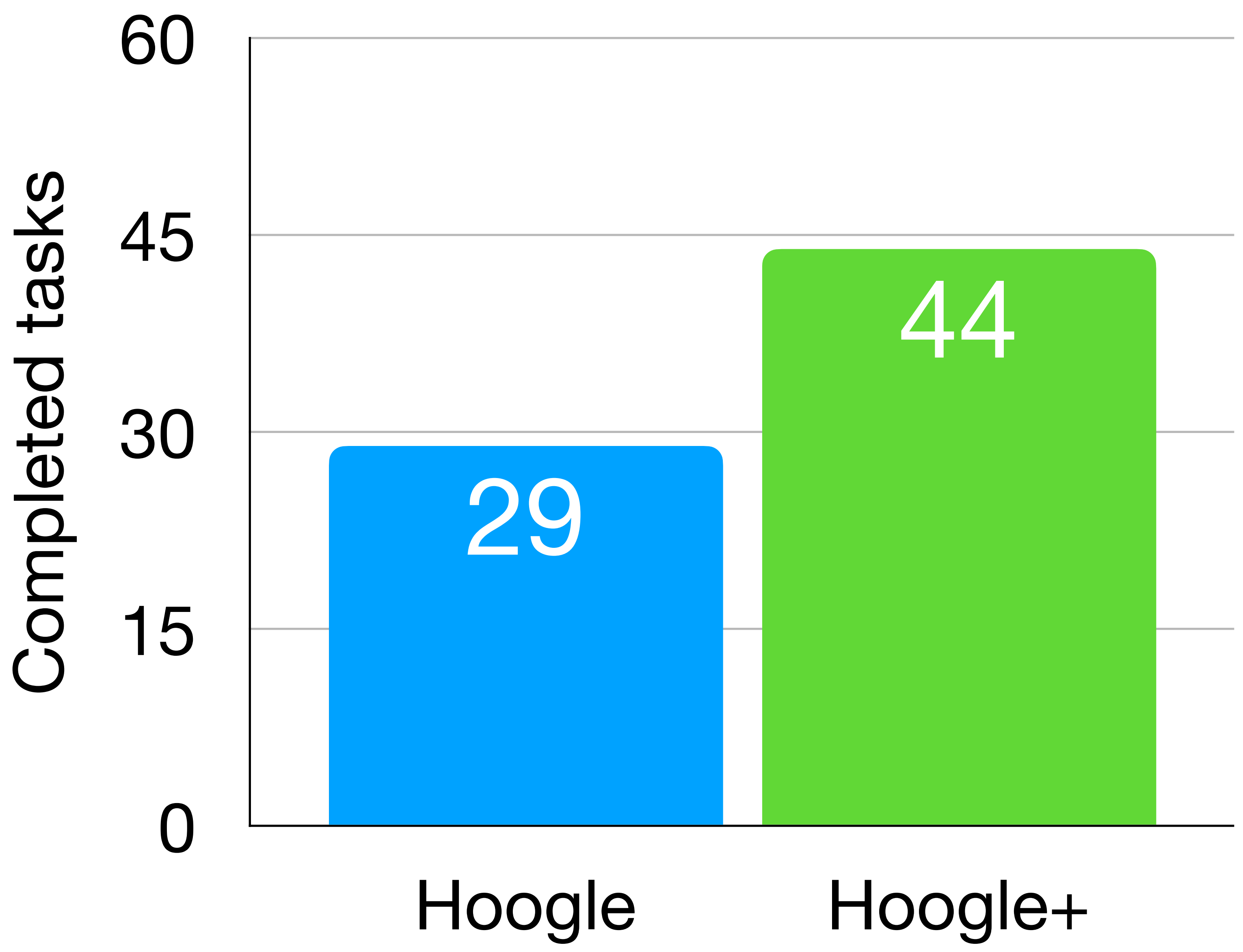
## Completion Rate



51% more!

# Results

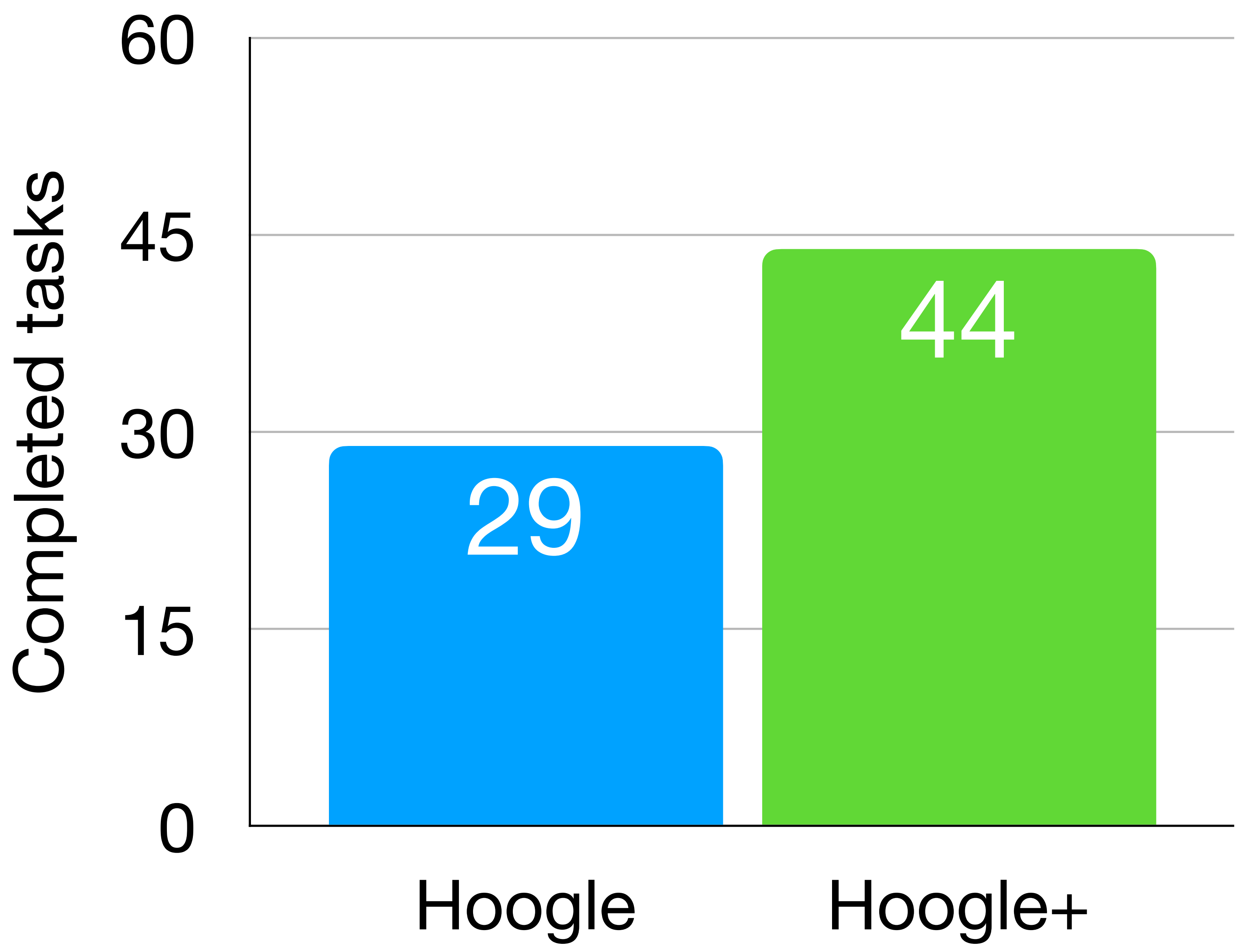
## Completion Rate



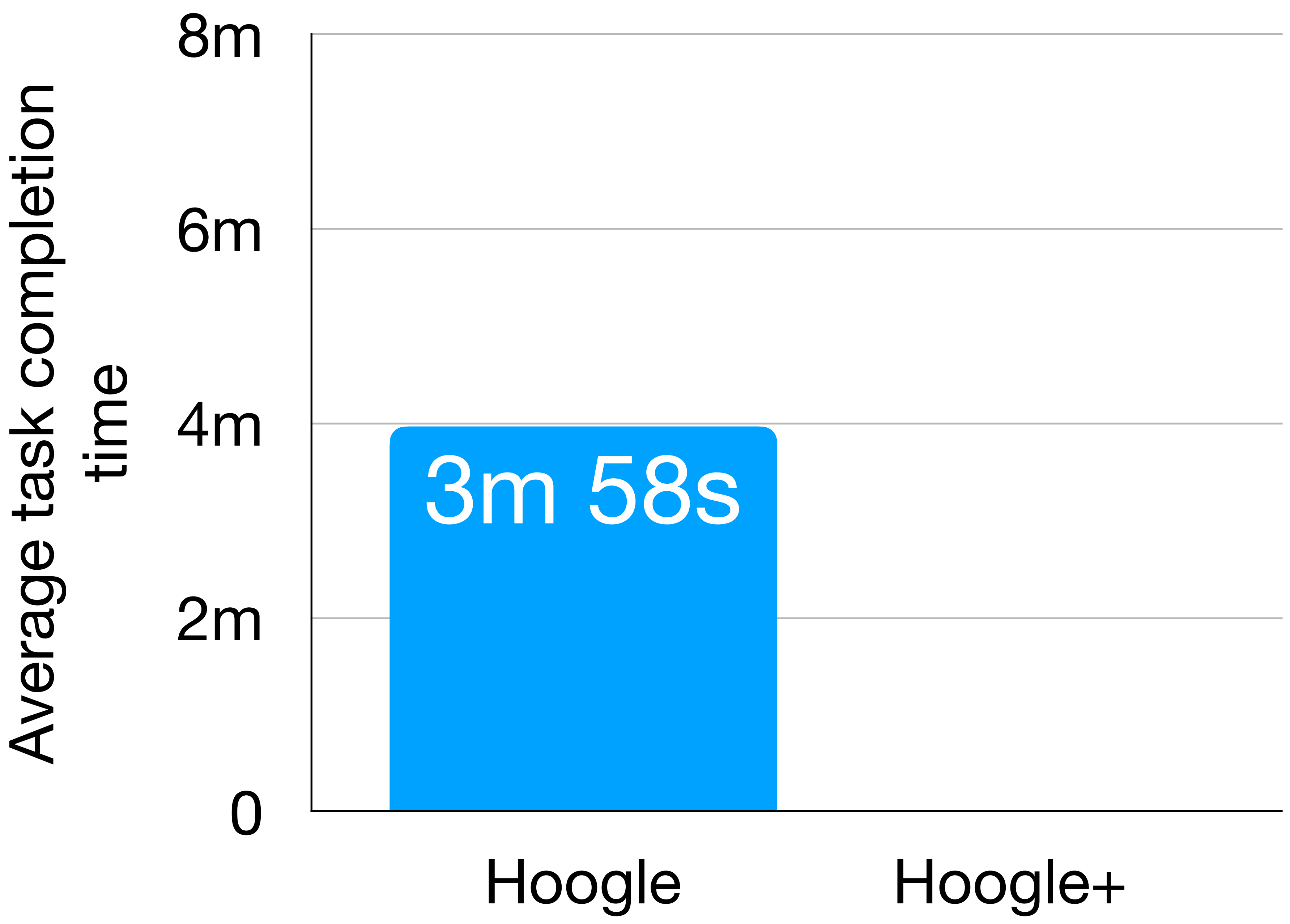
## Time-to-complete

# Results

## Completion Rate

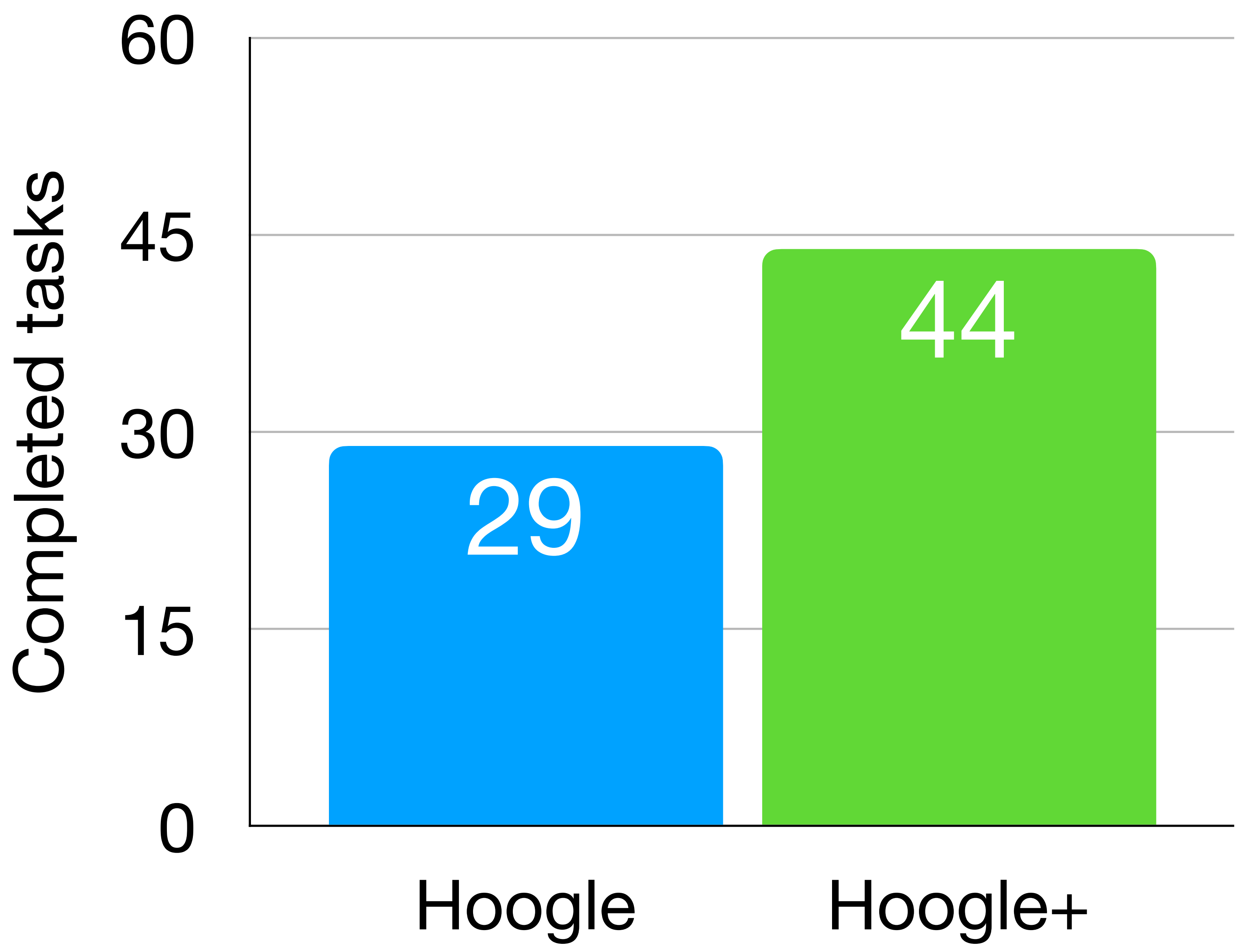


## Time-to-complete

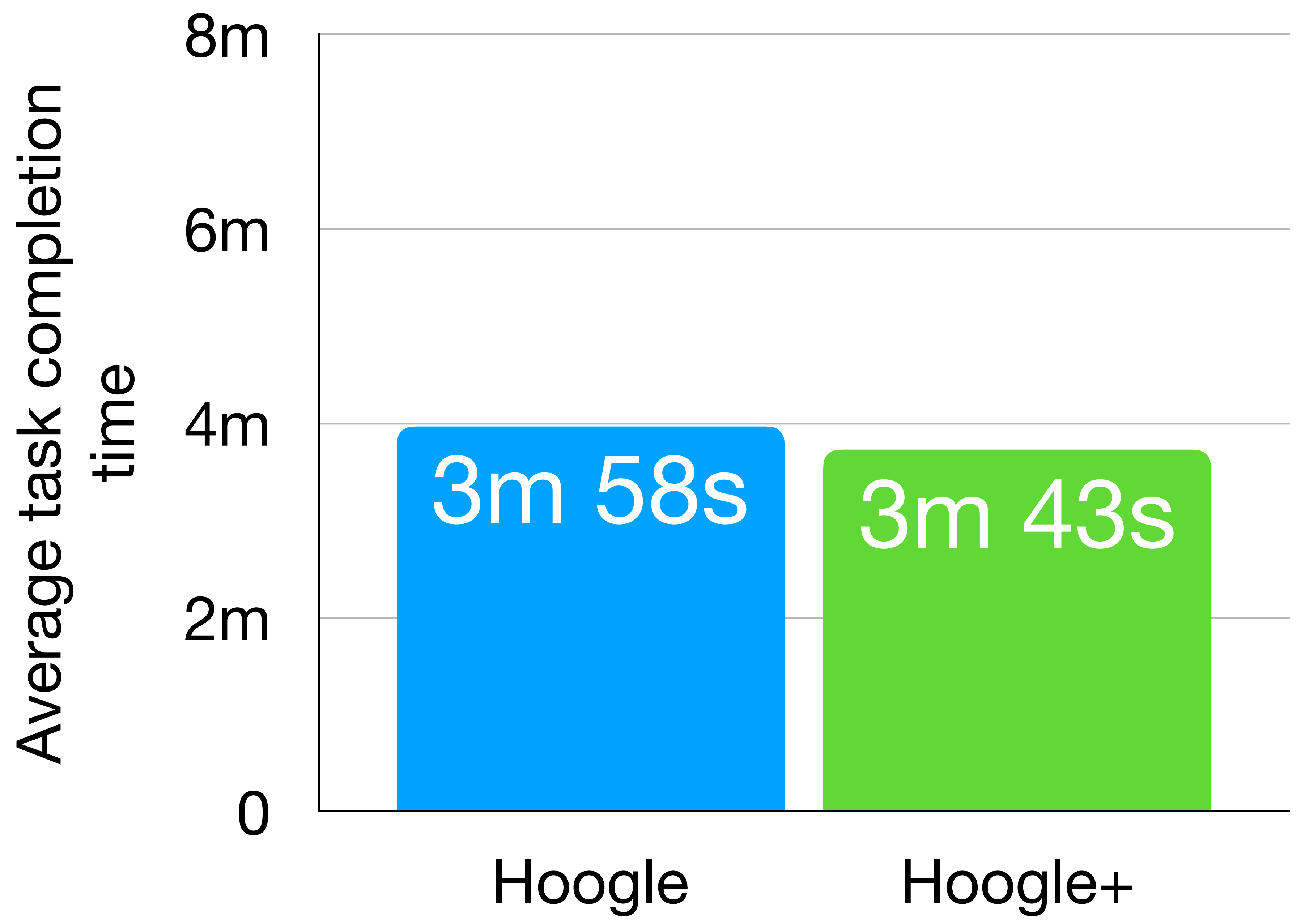


# Results

## Completion Rate

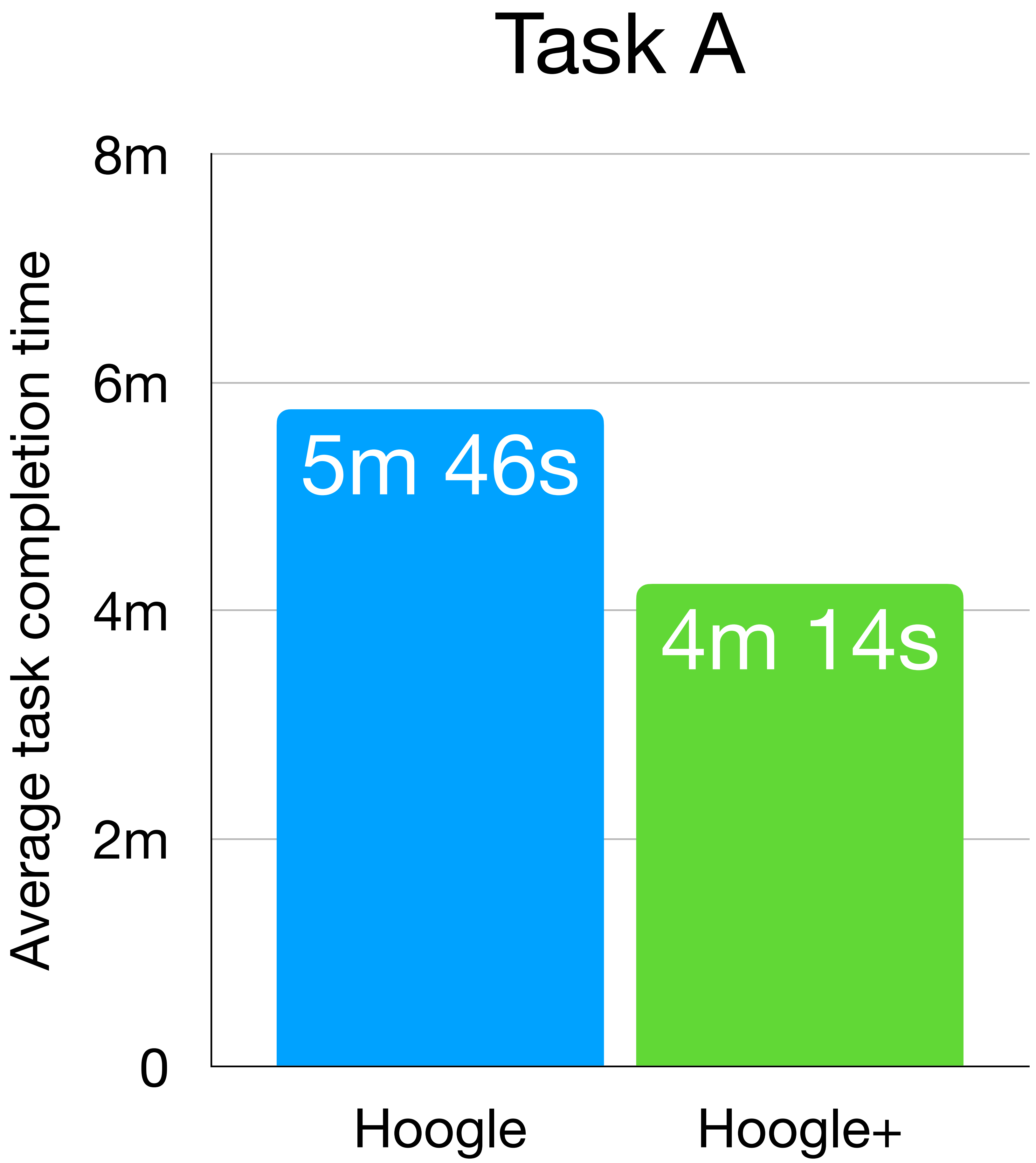


## Time-to-complete



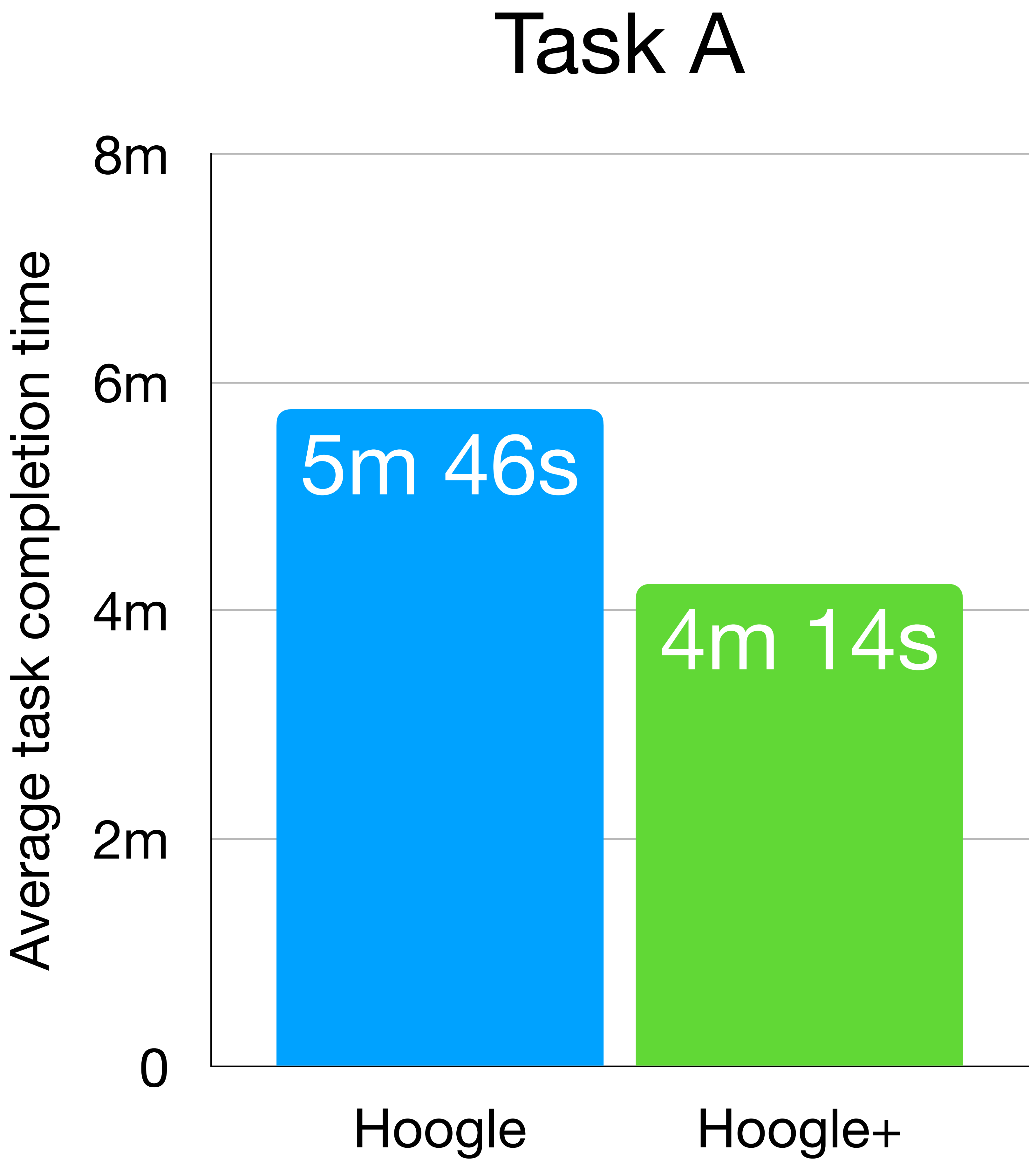
# Results

## Time-to-complete



# Results

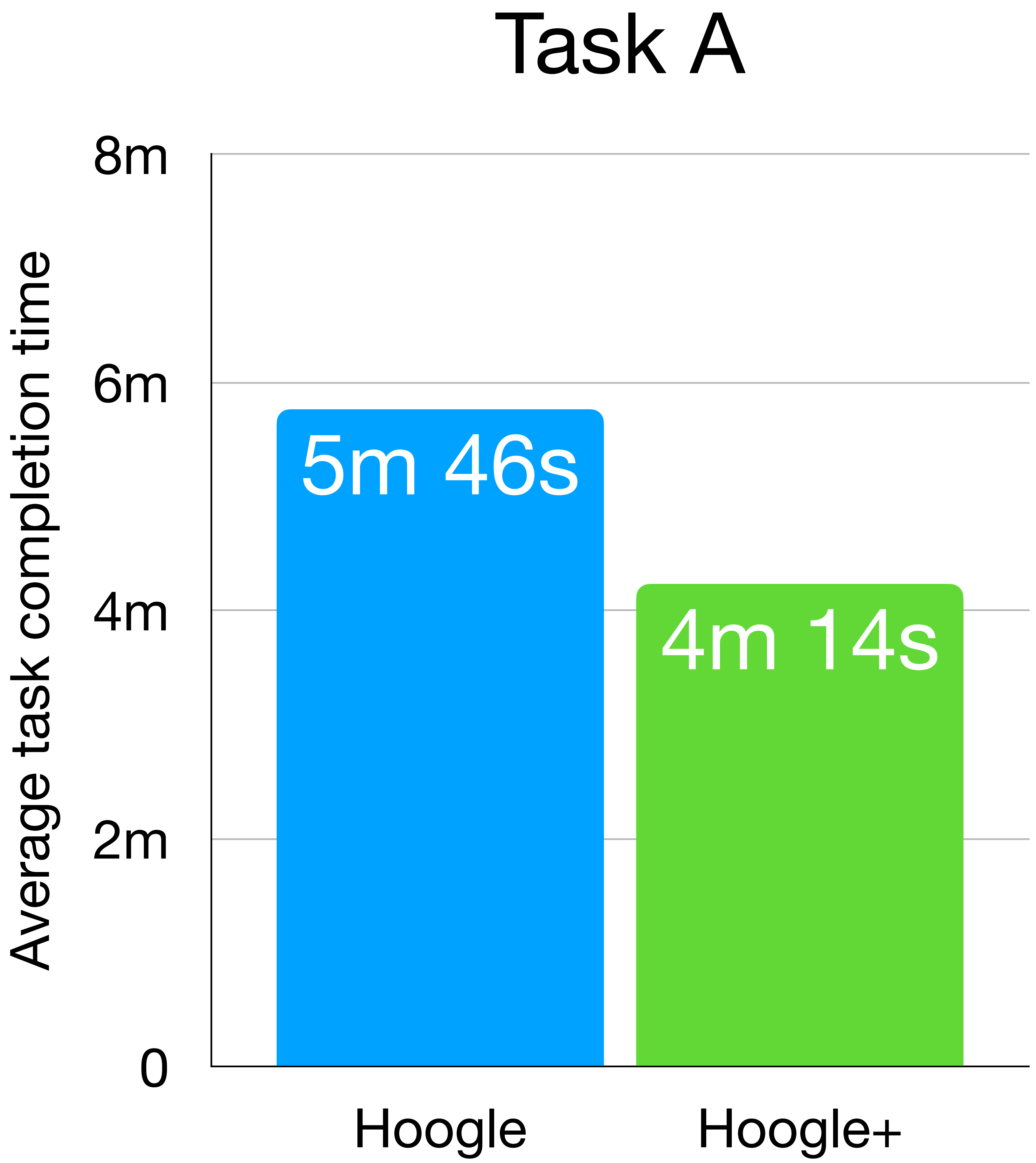
## Time-to-complete



90 second improvement

# Results

## Time-to-complete



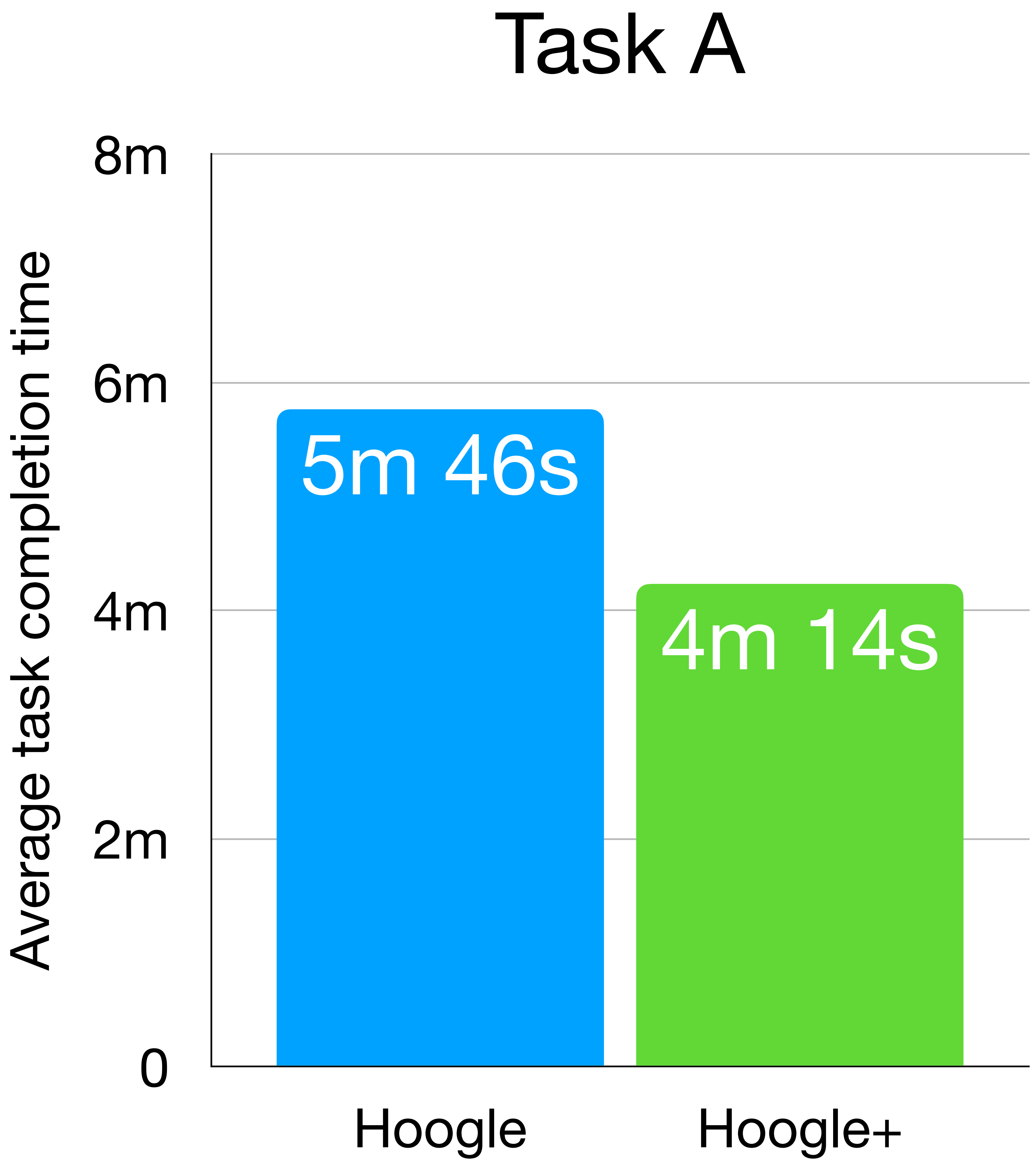
90 second improvement

**Input Types**

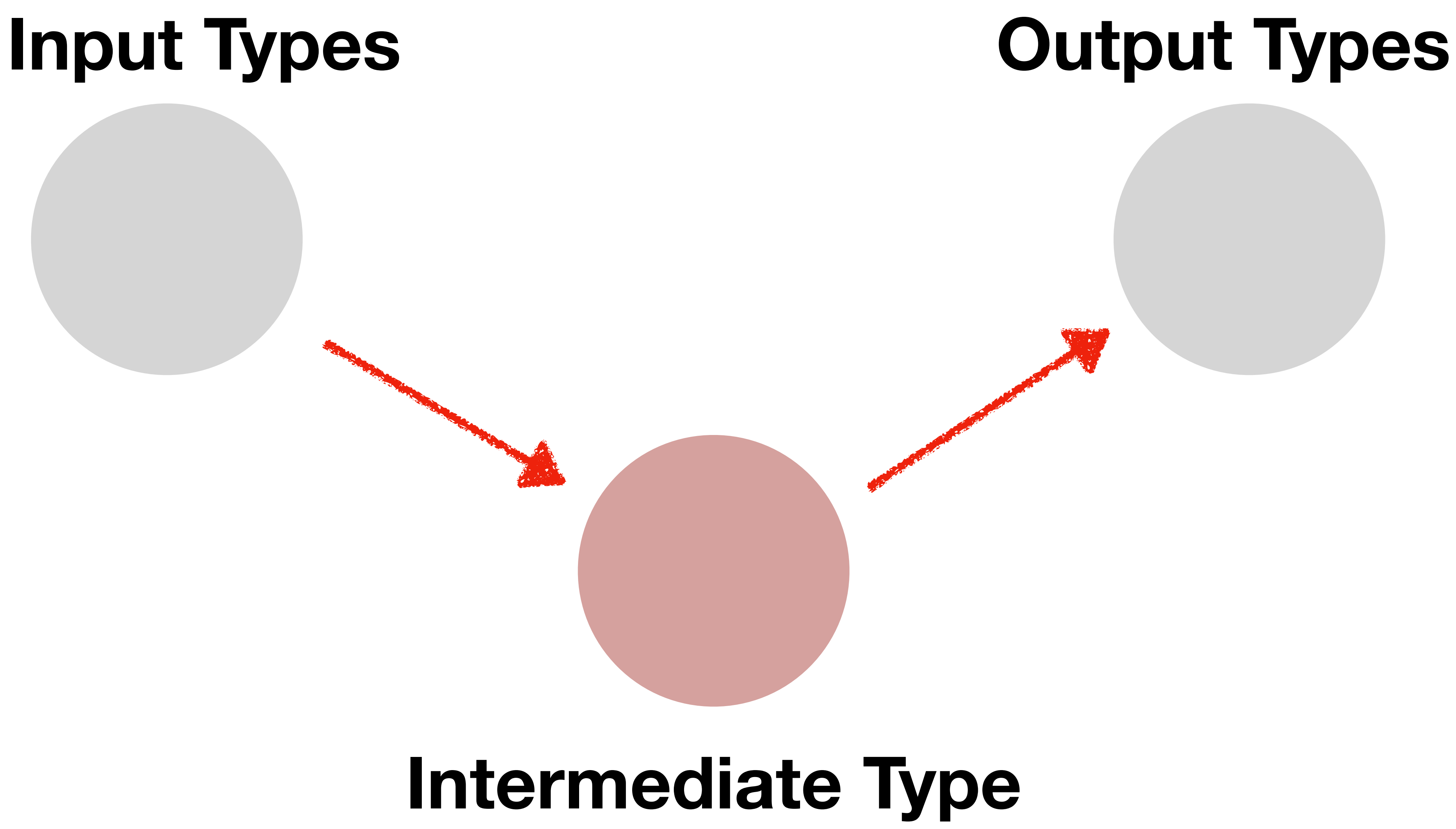
**Output Types**

# Results

Time-to-complete



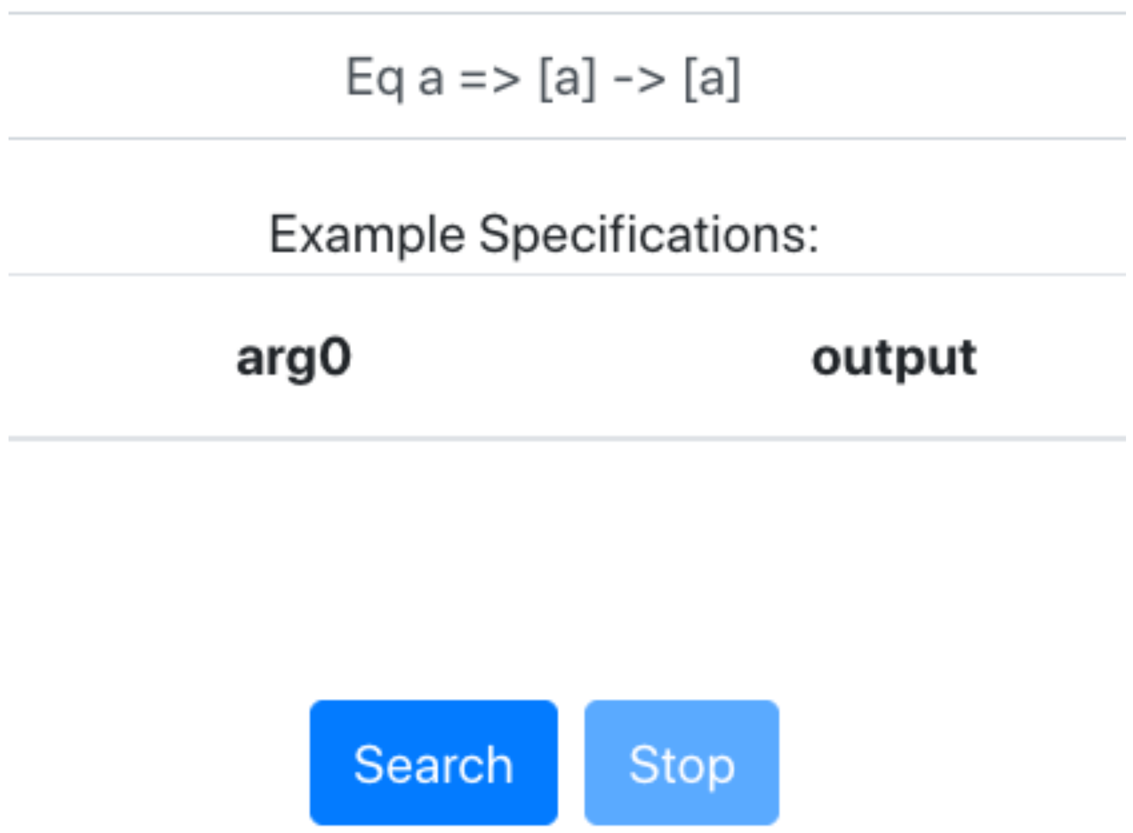
90 second improvement



# How did users give their specification?

# How did users give their specification?

Type Only



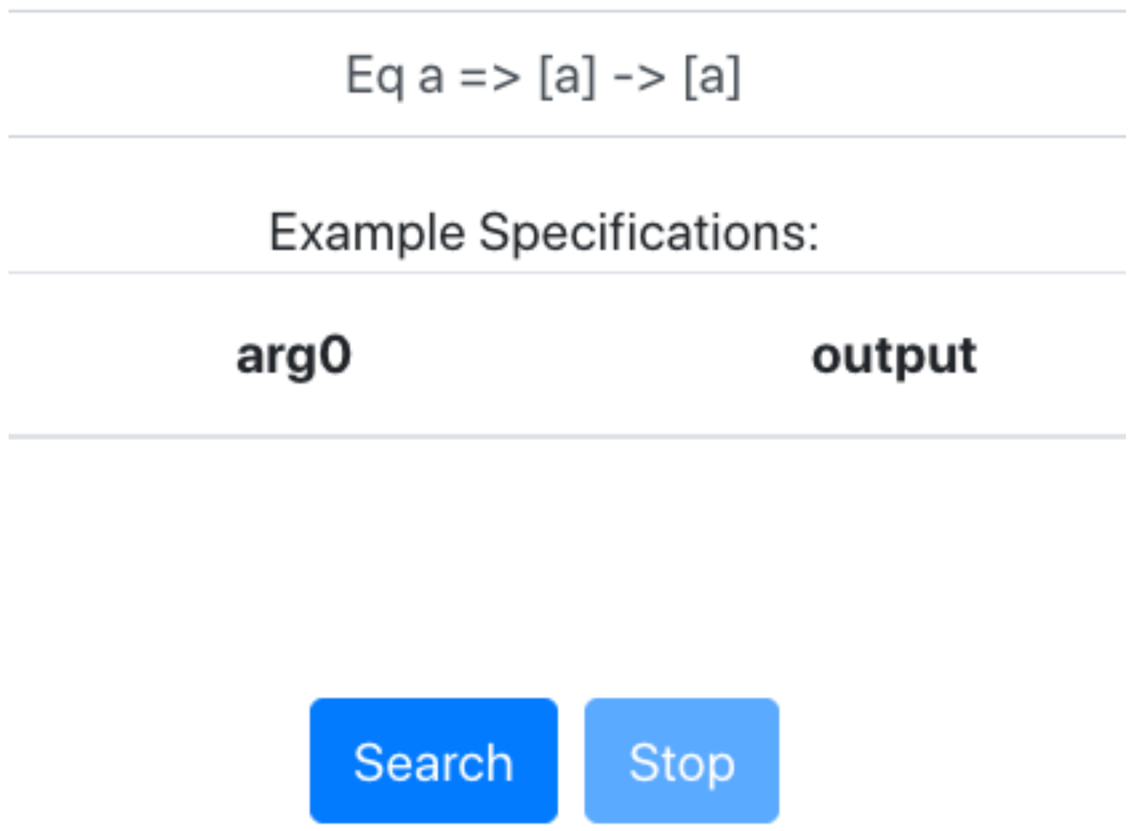
Eq a => [a] -> [a]

Example Specifications:

| arg0 | output |
|---|---|

# How did users give their specification?

Type Only

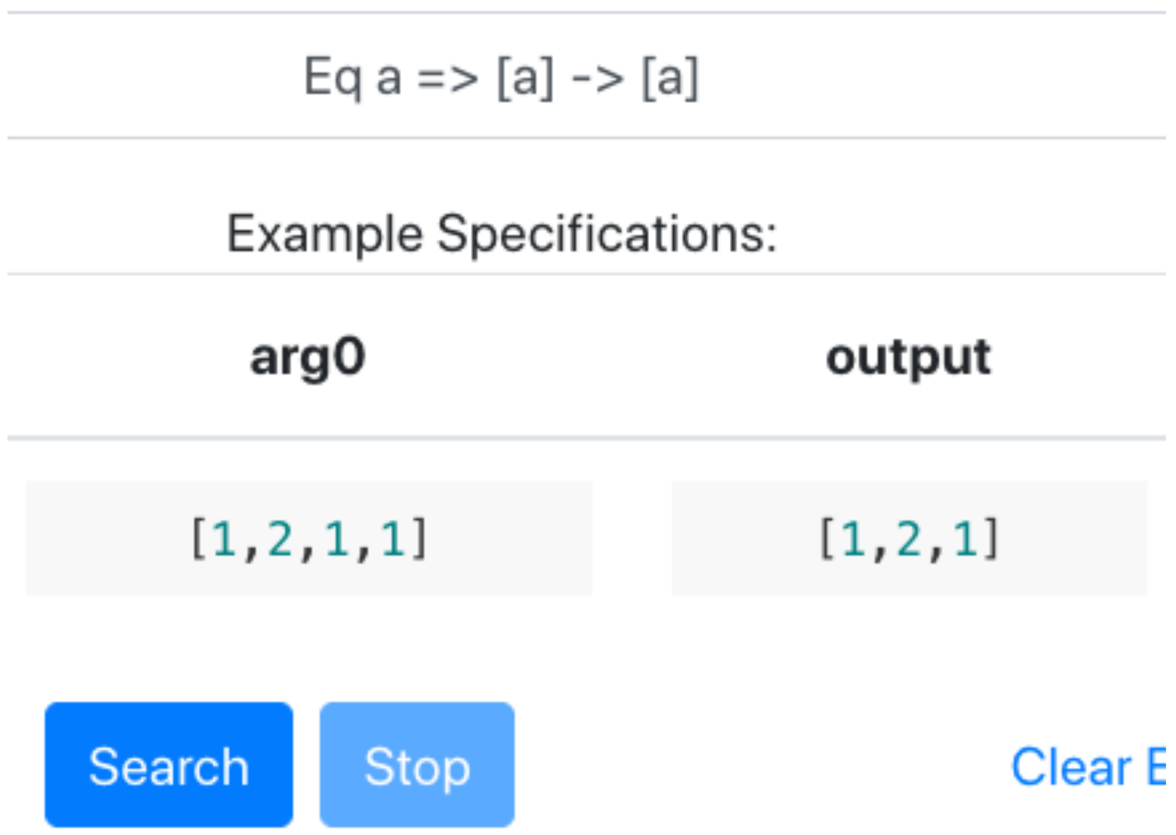


Eq a => [a] -> [a]

Example Specifications:

| arg0 | output |
|---|---|

Test + Type

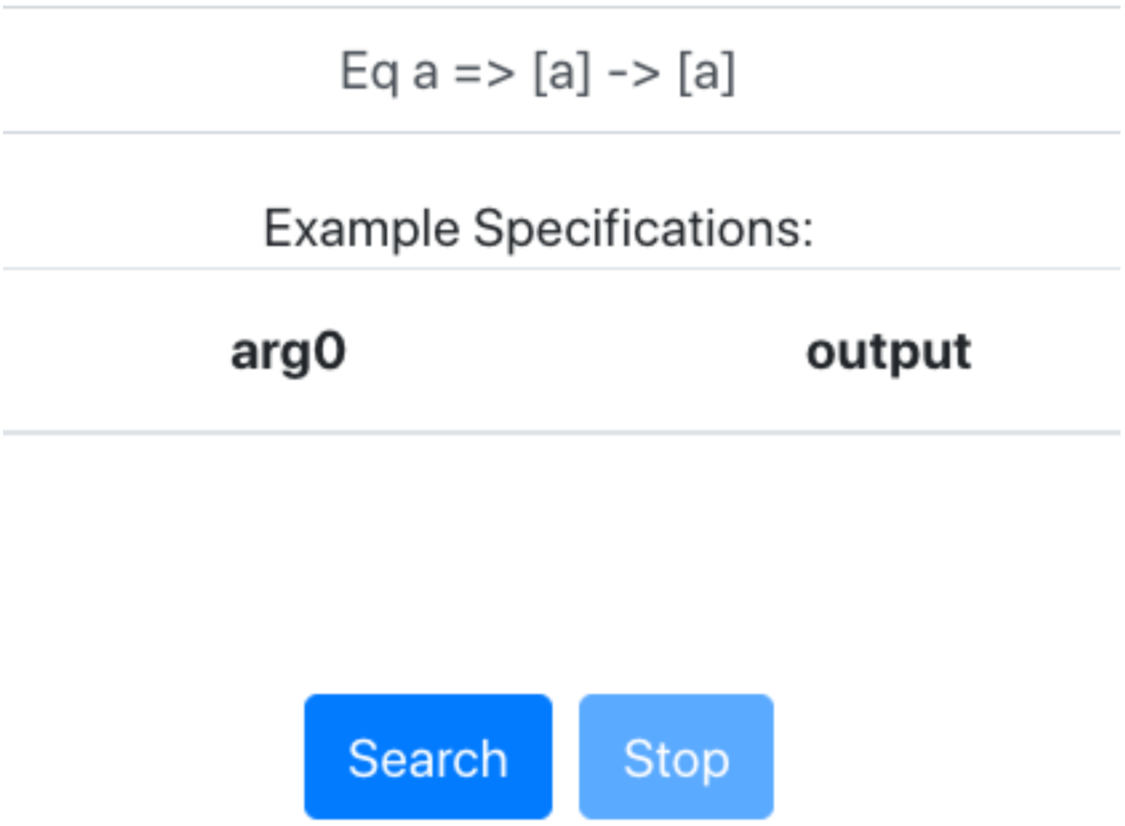


Eq a => [a] -> [a]

Example Specifications:

| arg0 | output |
|---|---|
| [1,2,1,1] | [1,2,1] |

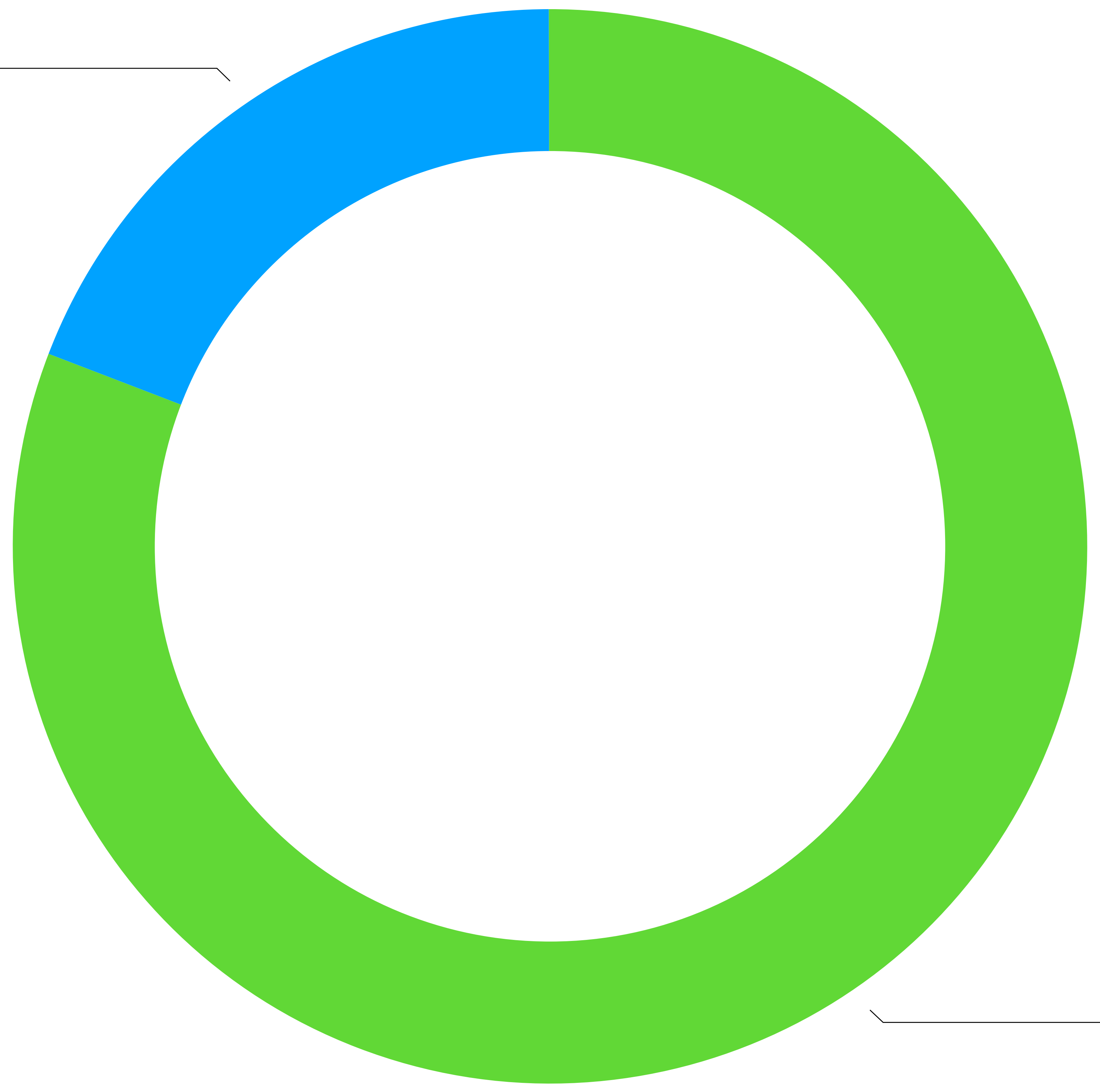
# How did users give their specification?

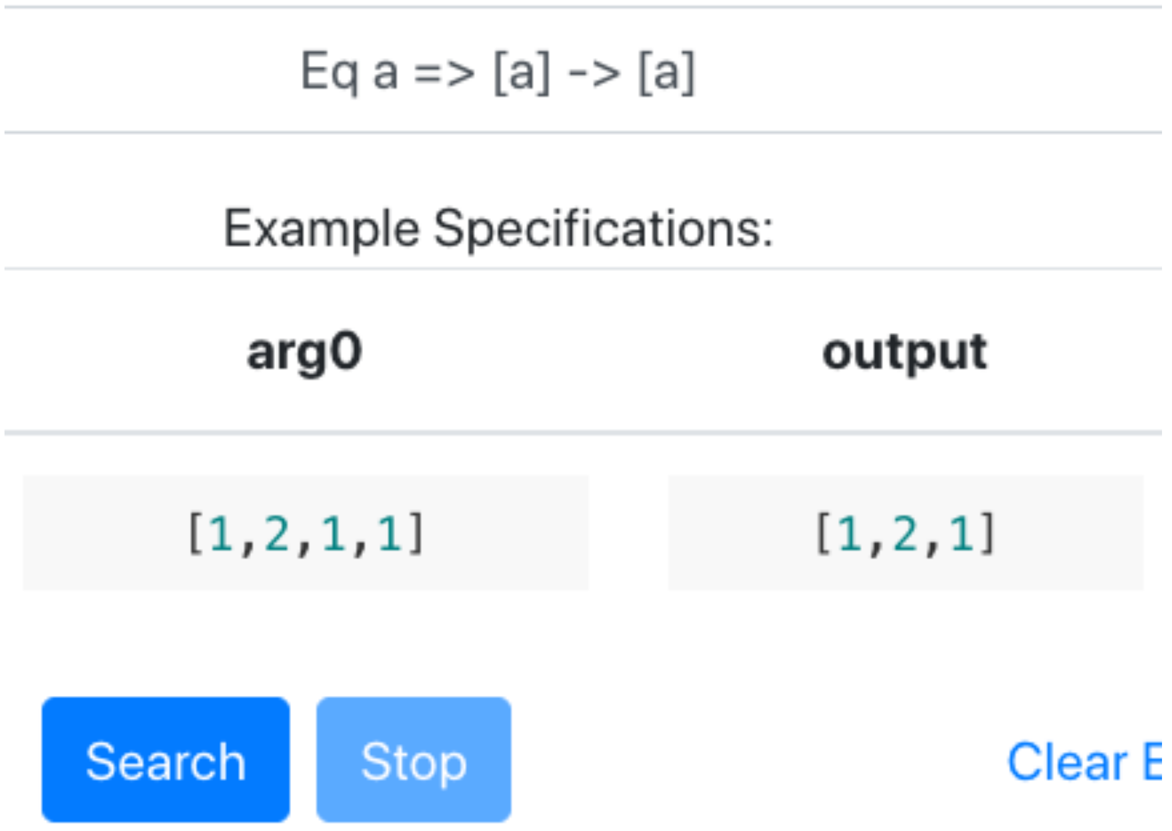


Eq a => [a] -> [a]

Example Specifications:

| arg0 | output |
|---|---|



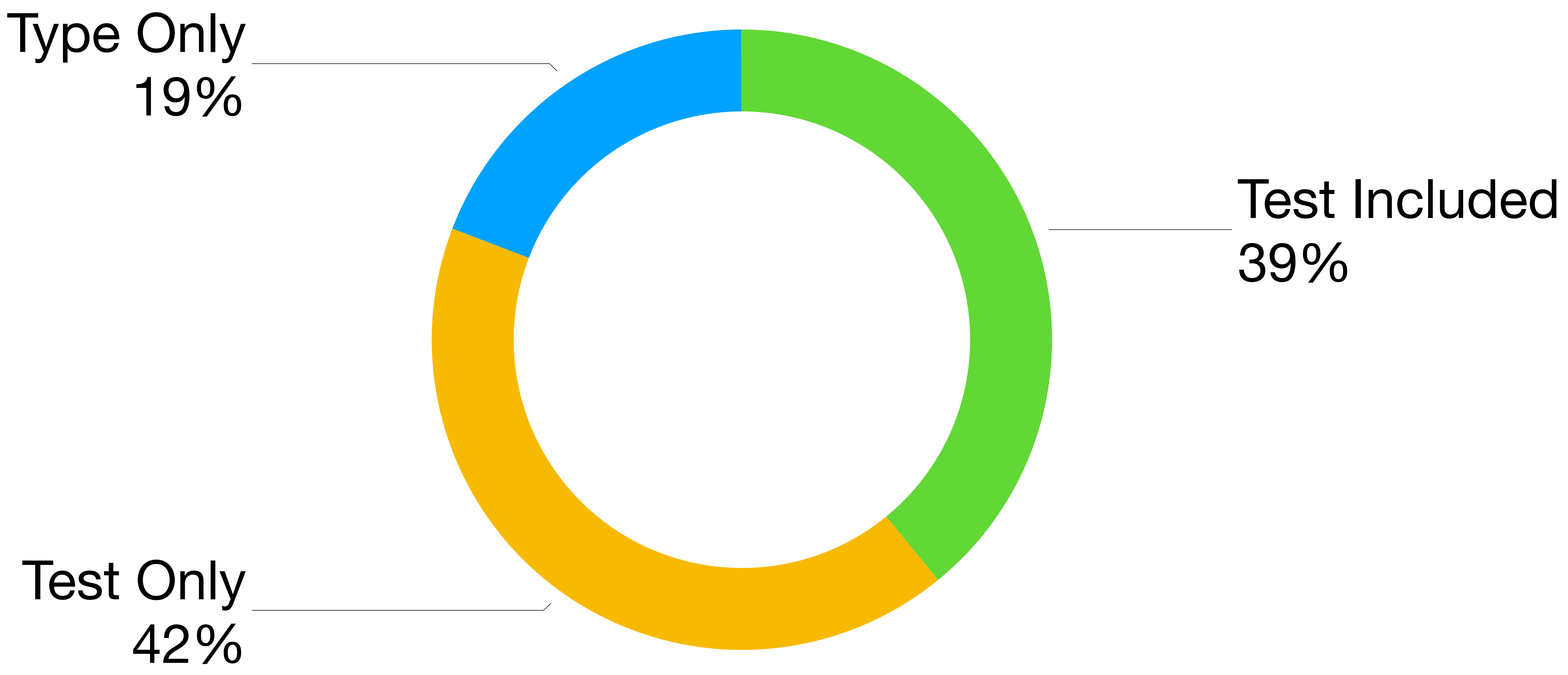


Eq a => [a] -> [a]

Example Specifications:

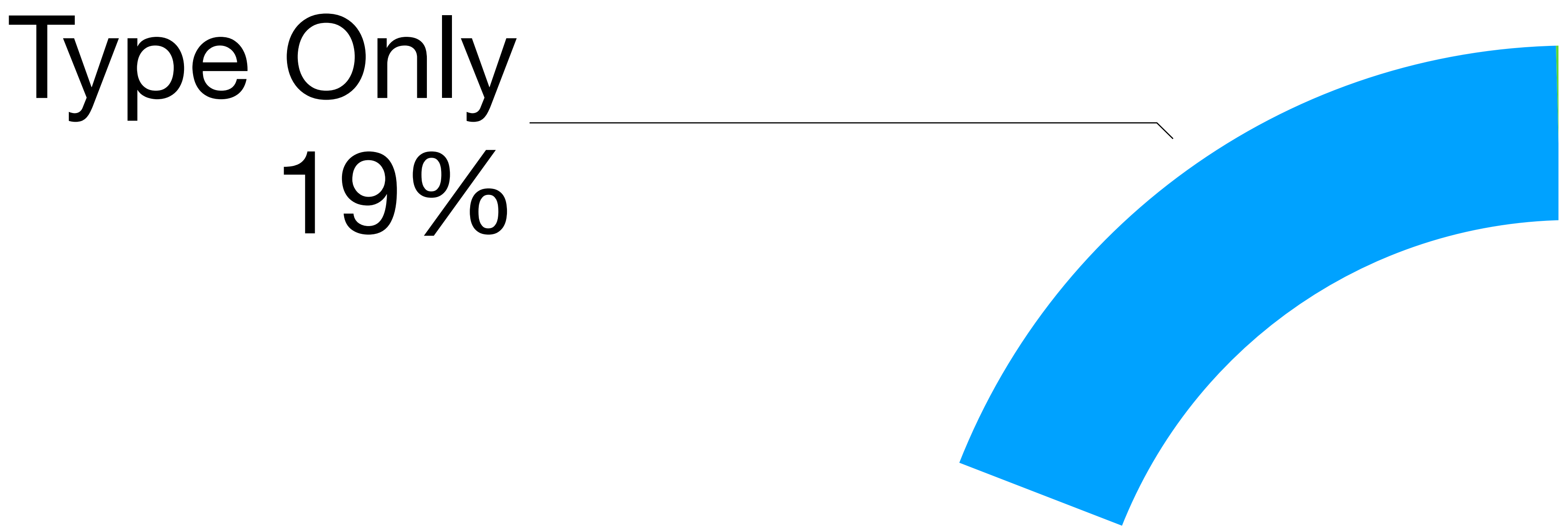
| arg0 | output |
|---|---|
| [1,2,1,1] | [1,2,1] |

# How did users give their specification?

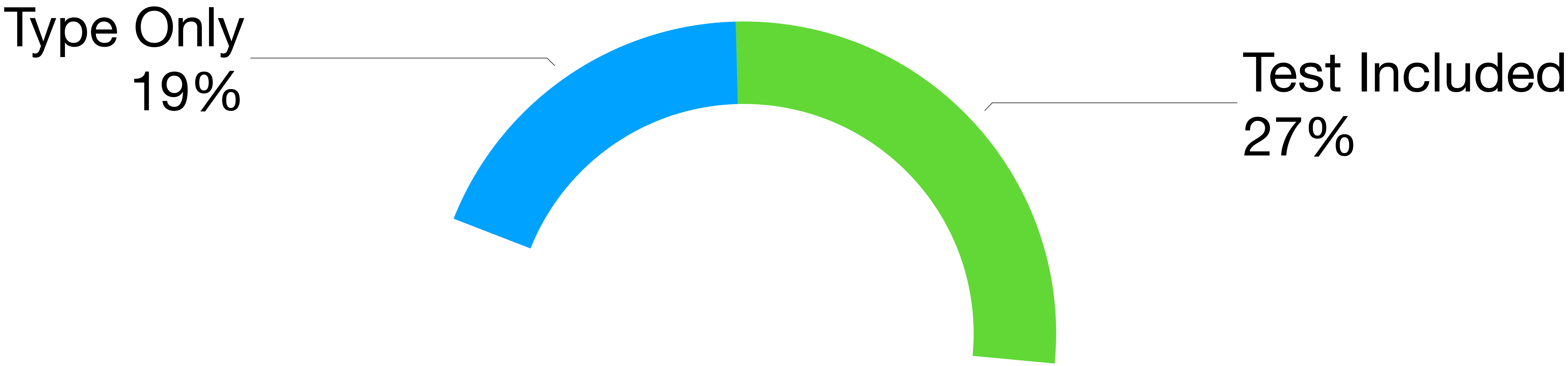


# Specification among novices

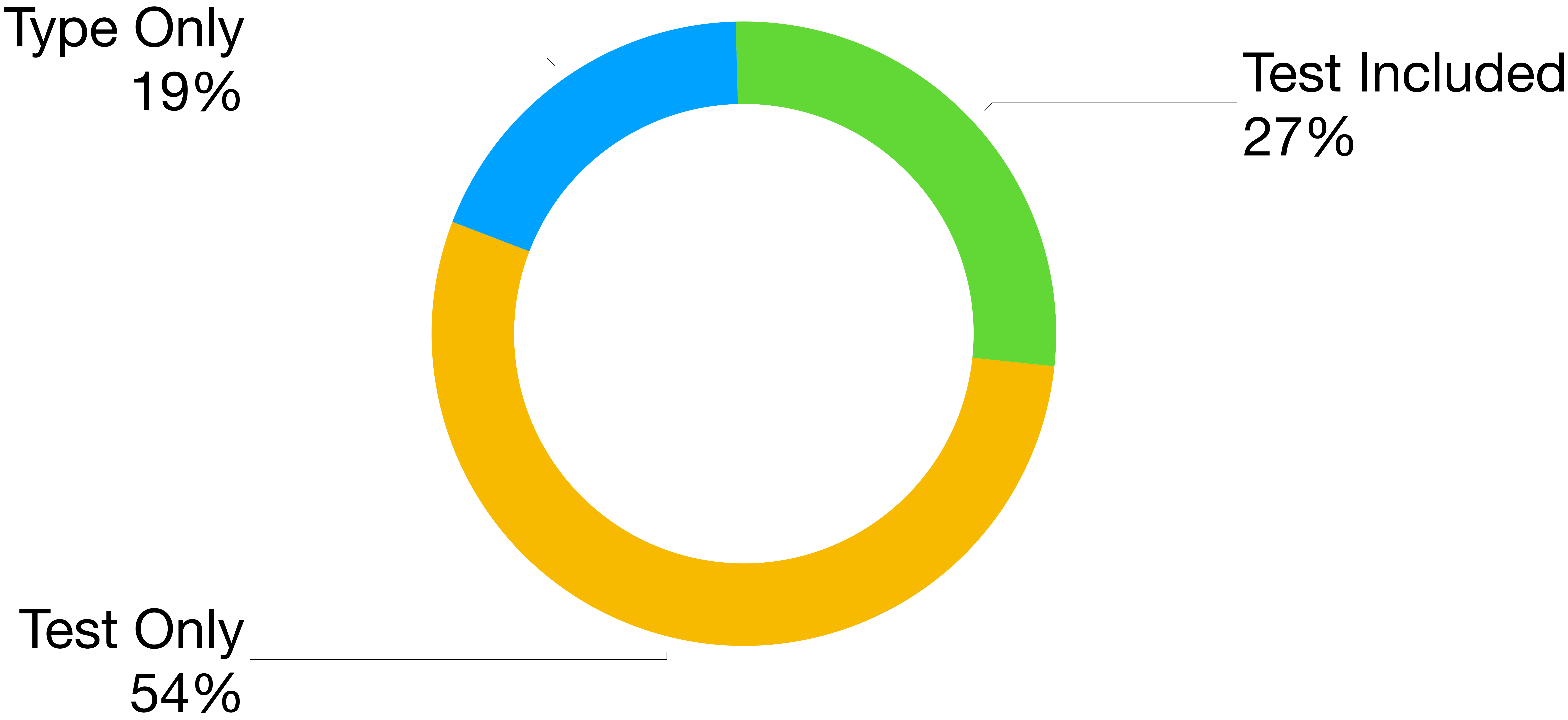
# Specification among novices



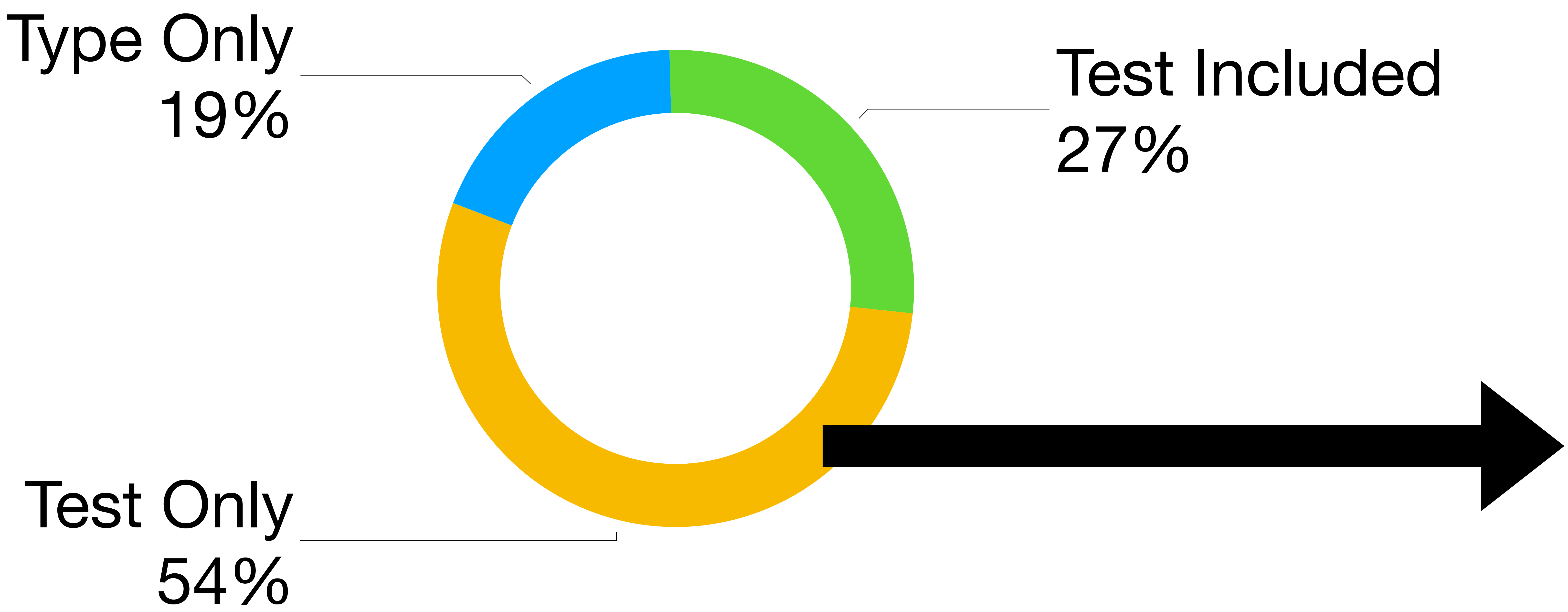
# Specification among novices

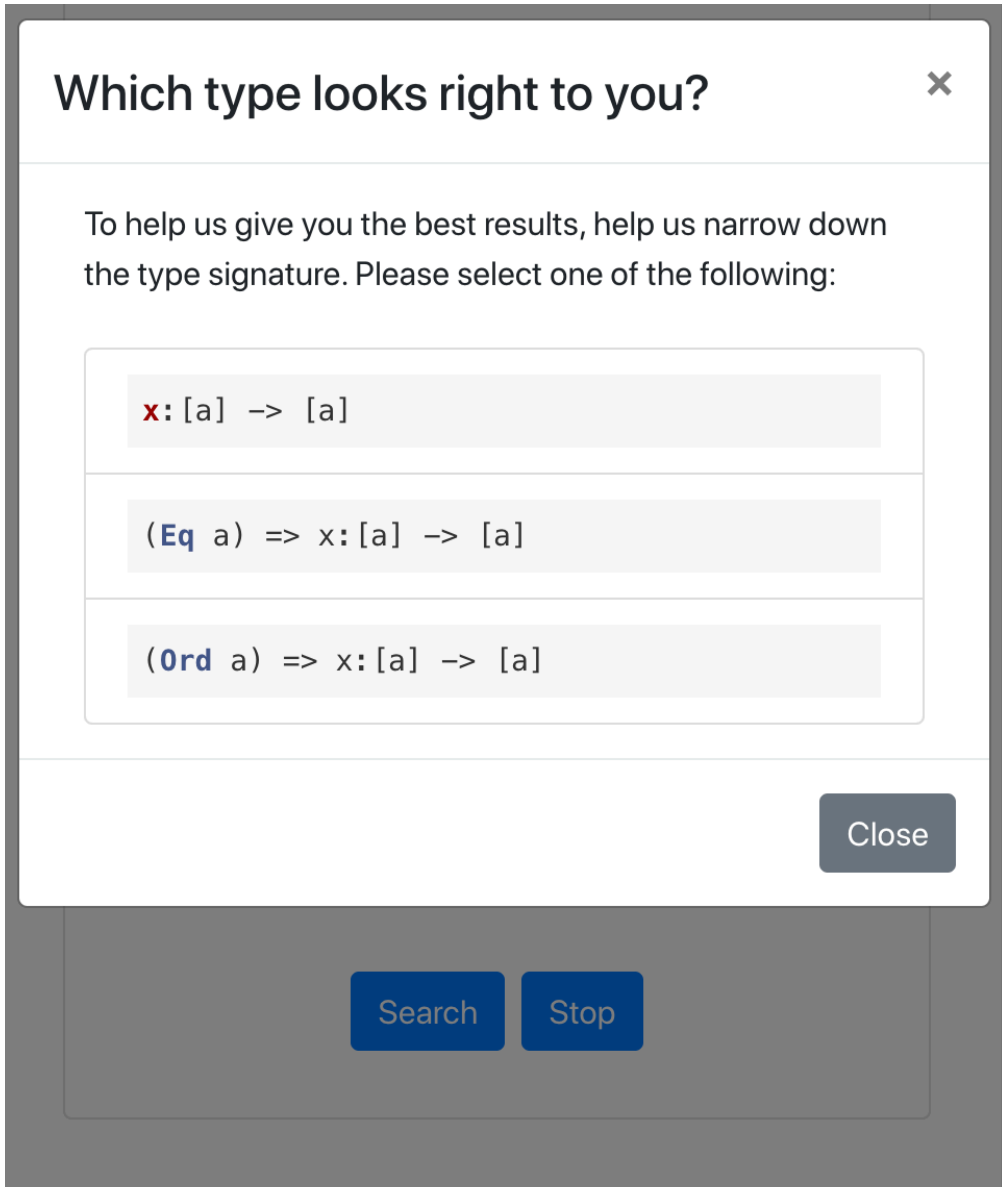


# Specification among novices



# Specification among novices



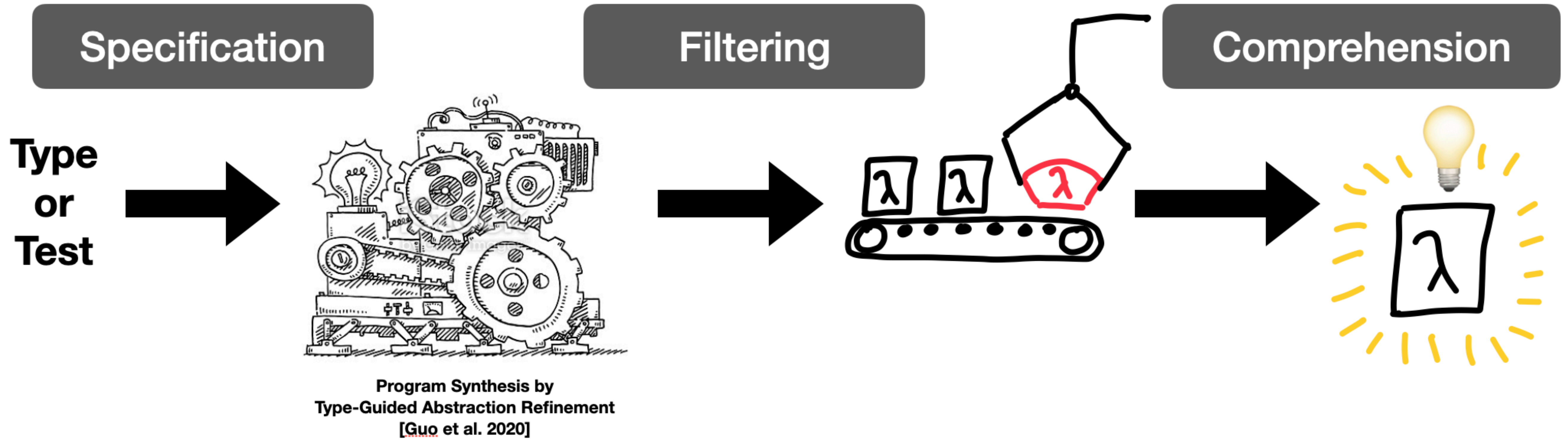


### Which type looks right to you?

To help us give you the best results, help us narrow down the type signature. Please select one of the following:

```
x:[a] -> [a]
(Eq a) => x:[a] -> [a]
(Ord a) => x:[a] -> [a]
```

# Conclusion

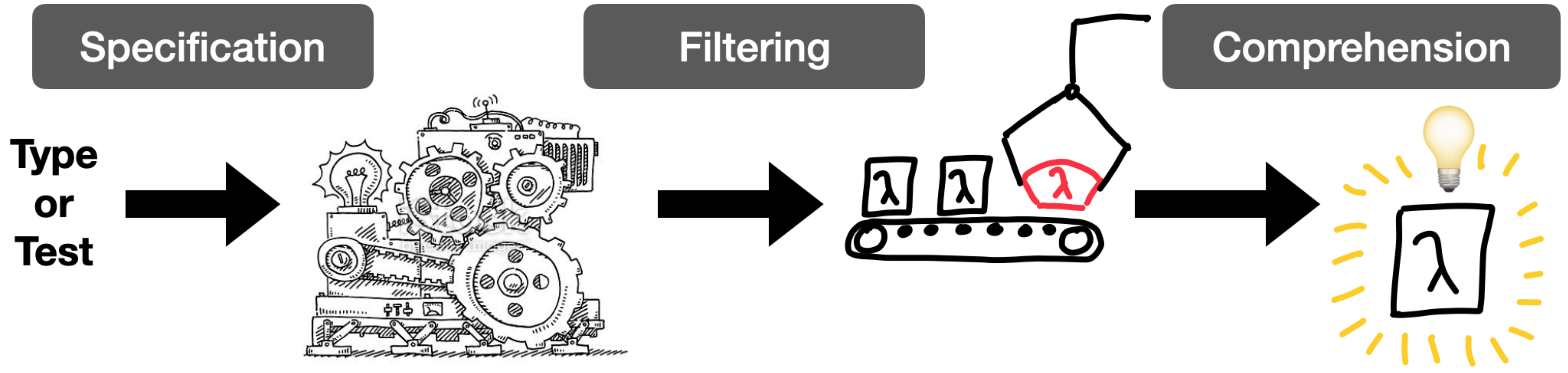


Program Synthesis by
Type-Guided Abstraction Refinement
[Guo et al. 2020]

Hoogle+ User Study

http://hplus.programming.systems

# Conclusion



Specification

Type or Test

Filtering

Comprehension

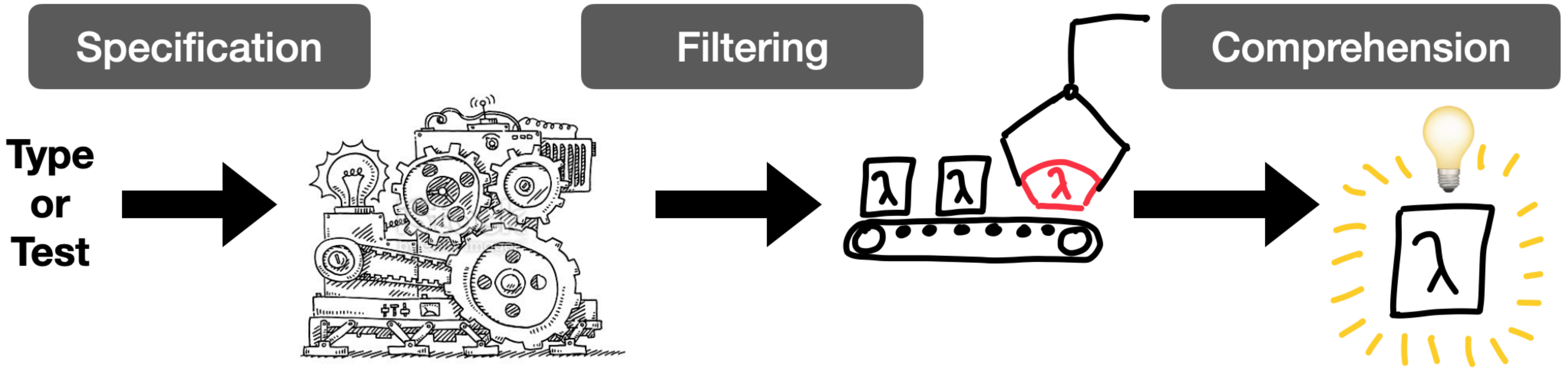
Program Synthesis by Type-Guided Abstraction Refinement [Guo et al. 2020]

Hoogle+ User Study

- Hoogle+ empowers users to complete more API-search focused tasks, faster

http://hplus.programming.systems

# Conclusion



Program Synthesis by
Type-Guided Abstraction Refinement
[Guo et al. 2020]

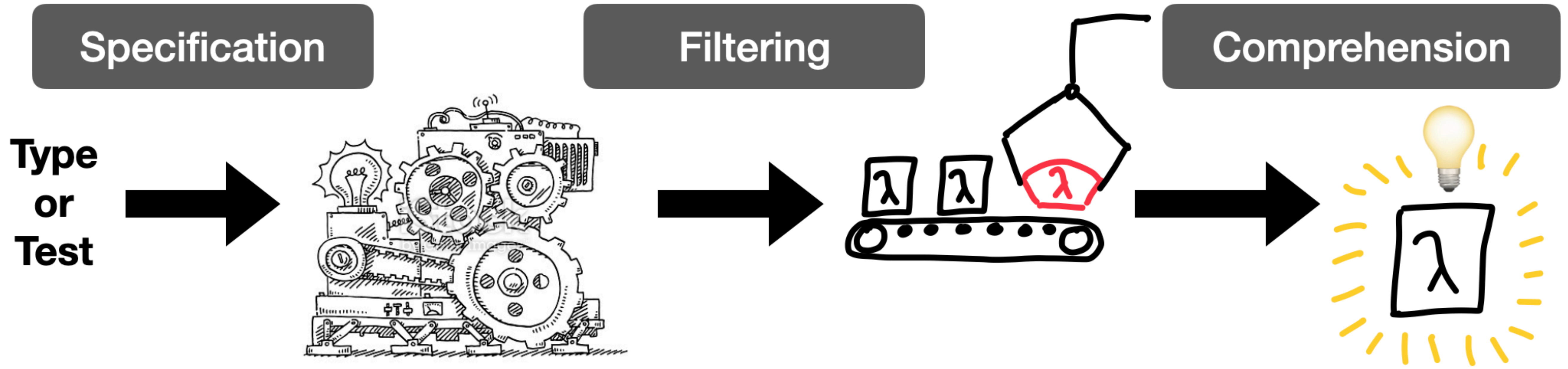
Hoogle+ User Study

- Hoogle+ empowers users to complete more API-search focused tasks, faster

- Infer likely types from tests

http://hplus.programming.systems

# Conclusion



Specification

Filtering

Comprehension

Type or Test

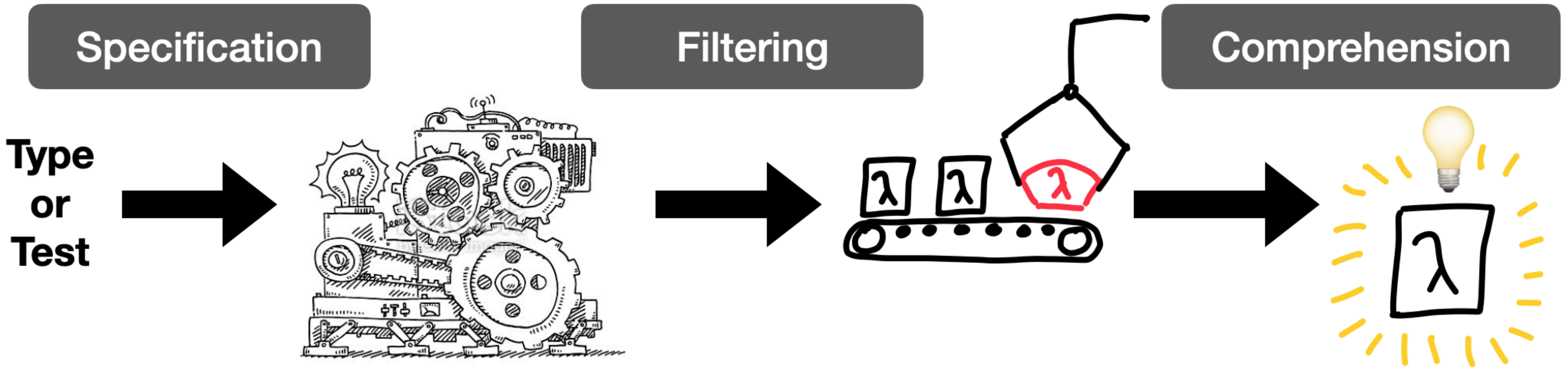
Program Synthesis by Type-Guided Abstraction Refinement [Guo et al. 2020]

Hoogle+ User Study

- Hoogle+ empowers users to complete more API-search focused tasks, faster
- Infer likely types from tests
- Filter away irrelevant programs

http://hplus.programming.systems

# Conclusion



Program Synthesis by
Type-Guided Abstraction Refinement
[Guo et al. 2020]

Hoogle+ User Study

- Hoogle+ empowers users to complete more API-search focused tasks, faster

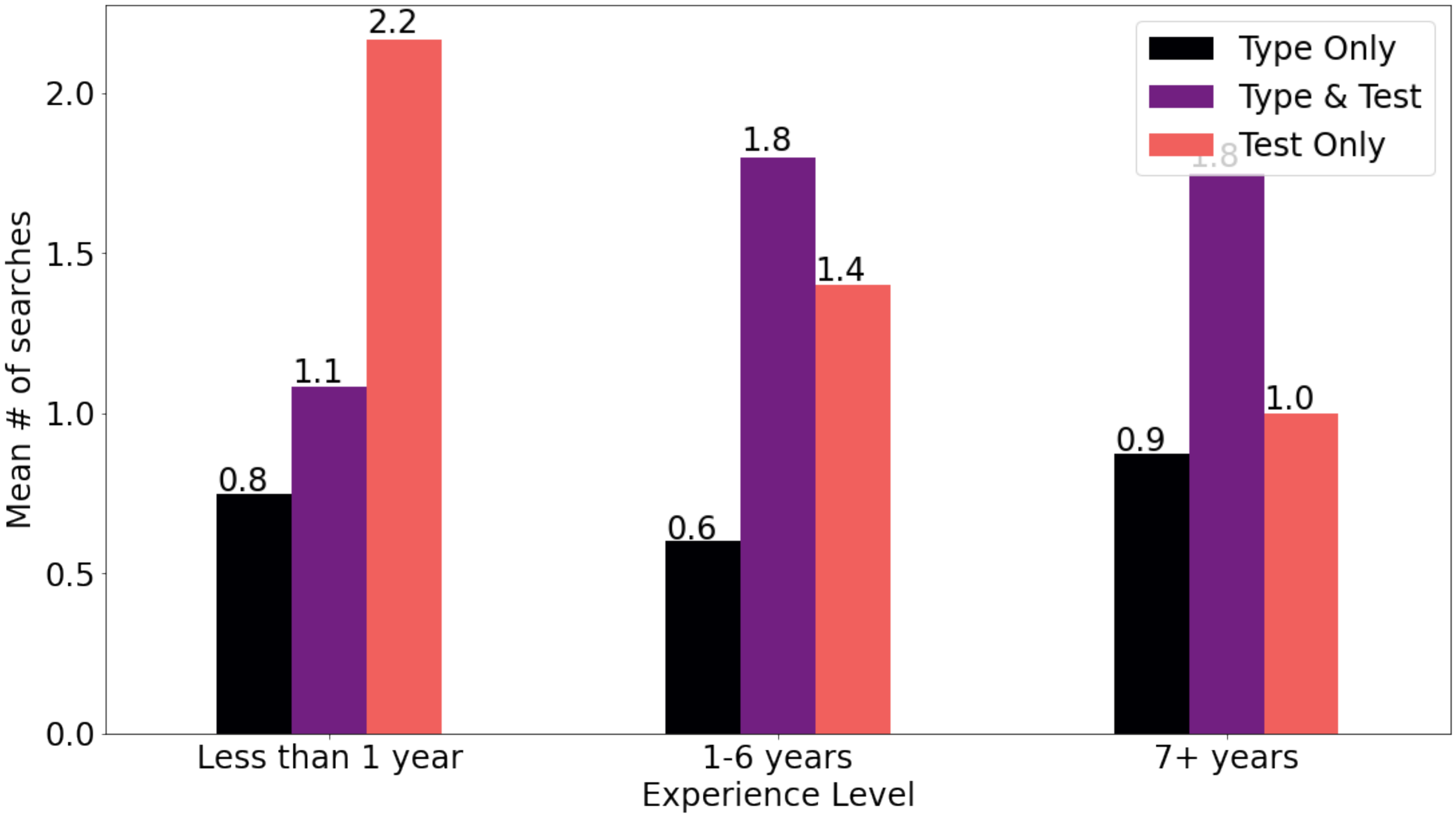
- Infer likely types from tests

- Filter away irrelevant programs

- Autogenerated comprehension examples

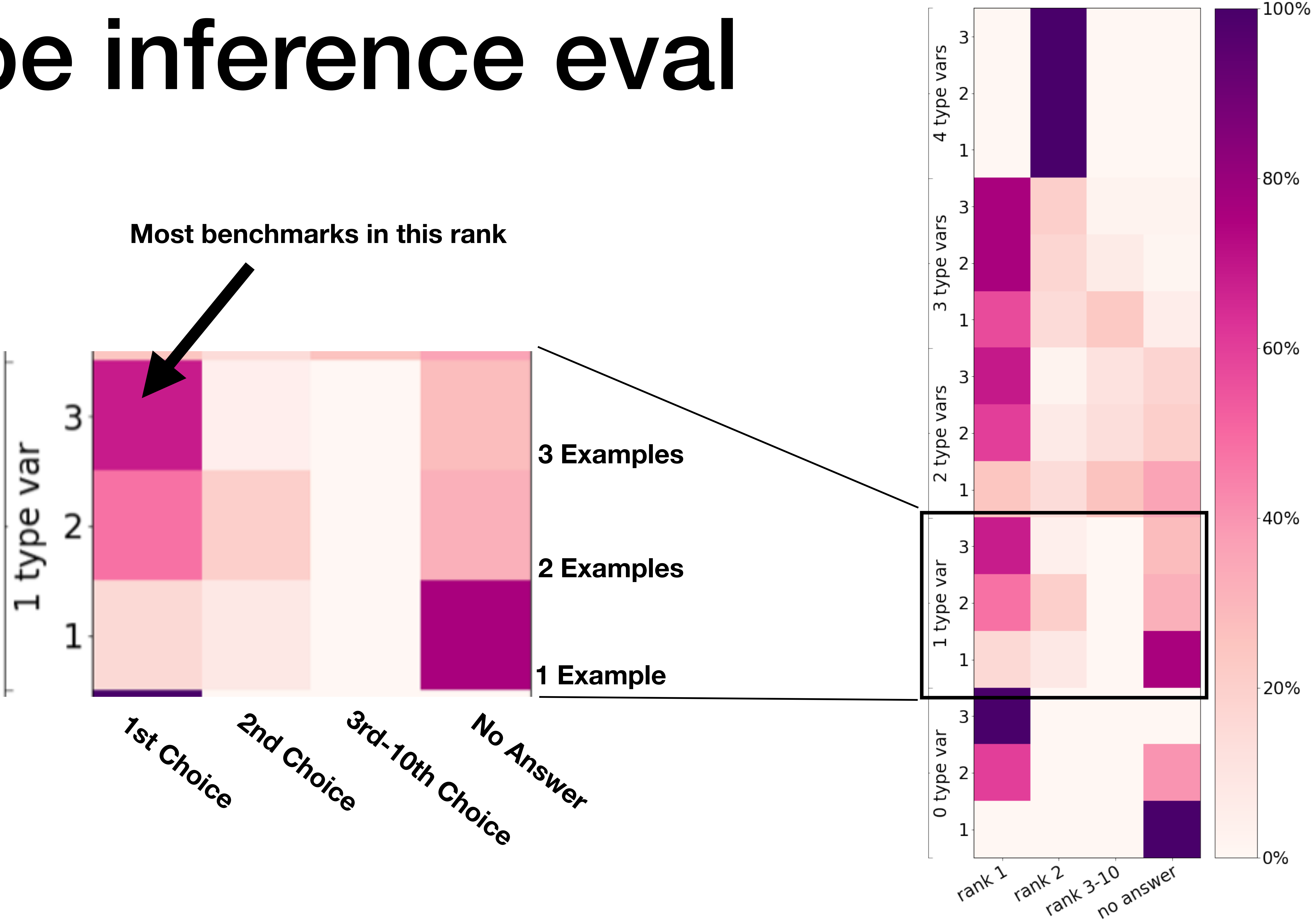
http://hplus.programming.systems

# Types of searches by experience



**Mean**

# Type inference eval



Most benchmarks in this rank

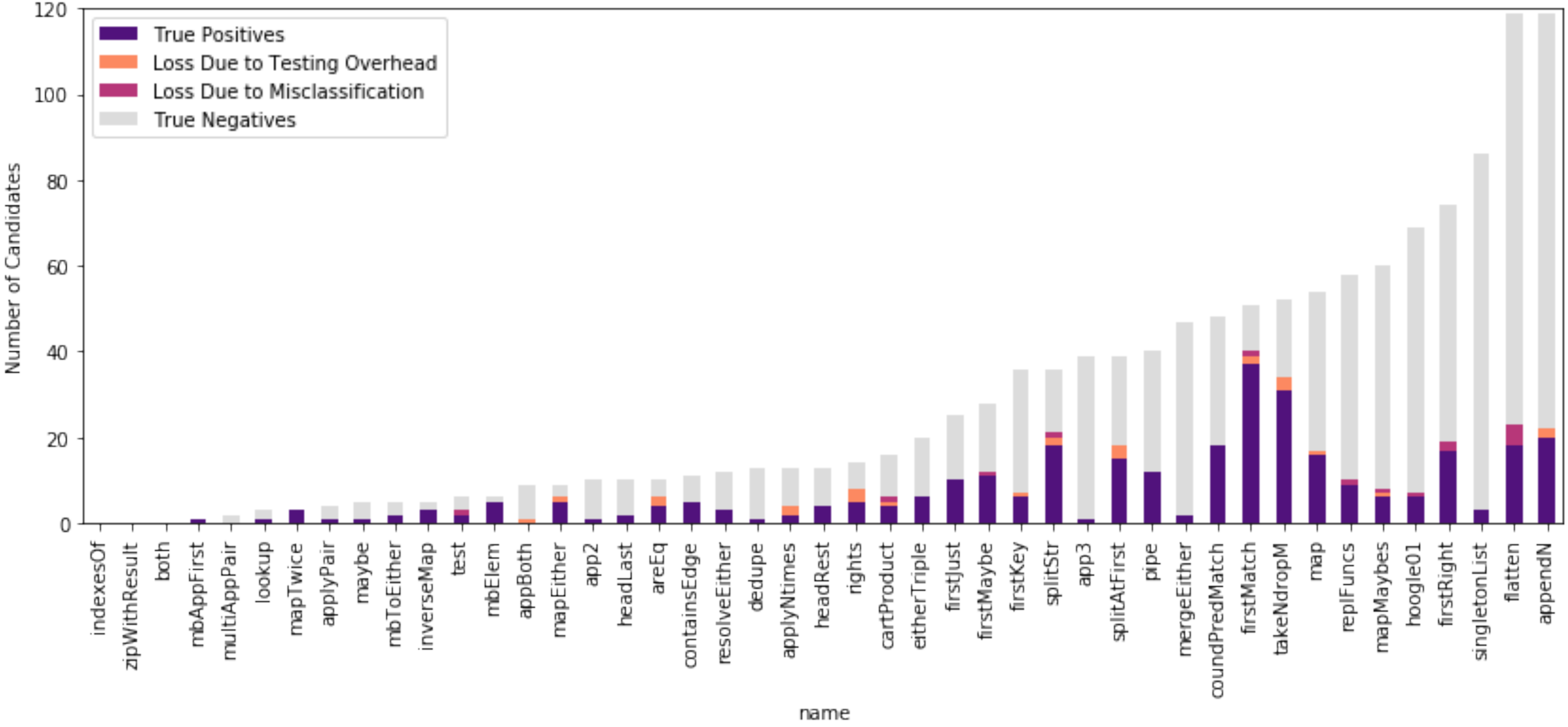
Expected
1 type variable

3 Examples

2 Examples

1 Example

# Filtering Eval


True Positives
Loss Due to Testing Overhead
Loss Due to Misclassification
True Negatives
Number of Candidates
name
indexesOf
zipWithResult
both
mbAppFirst
multiAppPair
lookup
mapTwice
applyPair
maybe
mbToEither
inverseMap
test
mbElem
appBoth
mapEither
app2
headLast
areEq
containsEdge
resolveEither
dedupe
applyNtimes
headRest
rights
cartProduct
eitherTriple
firstJust
firstMaybe
firstKey
splitStr
app3
splitAtFirst
pipe
mergeEither
coundPredMatch
firstMatch
takeNdropM
map
replFuncs
mapMaybes
hoogle01
firstRight
singletonList
flatten
appendN