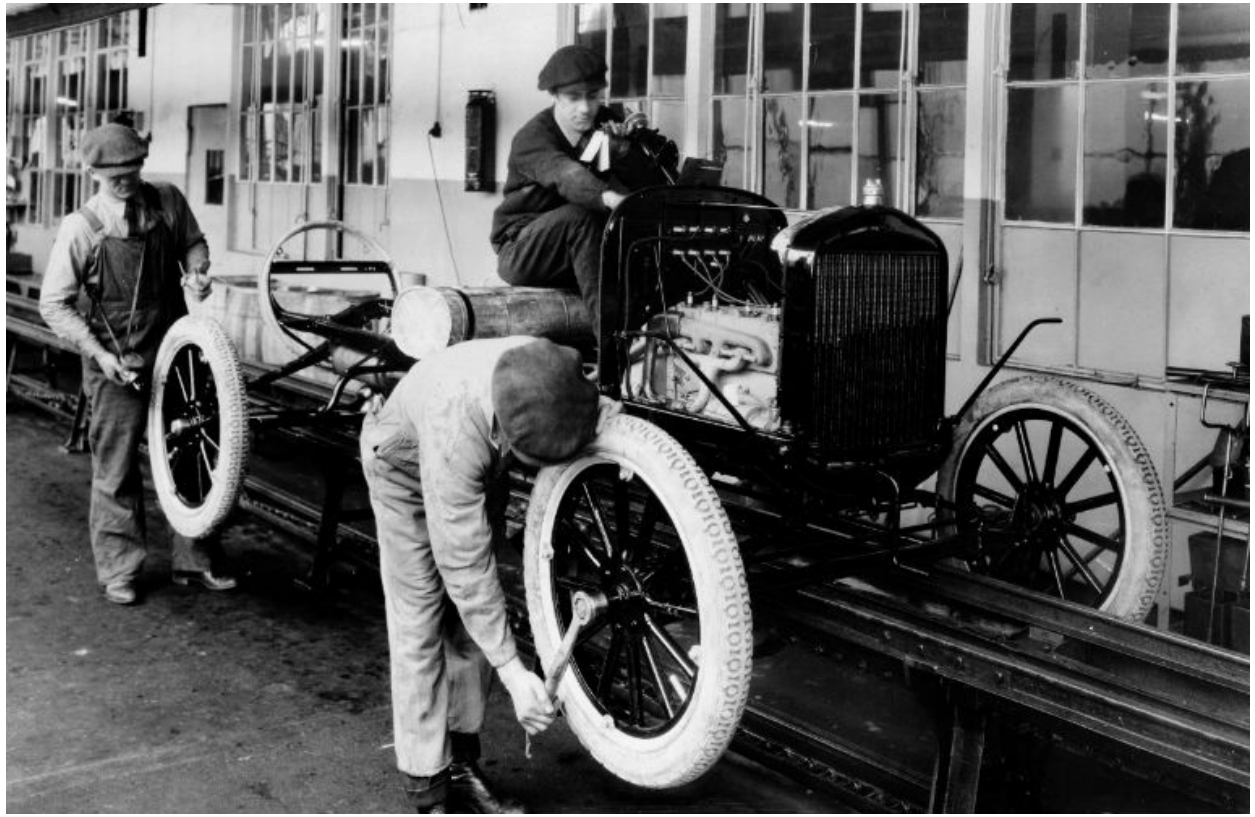
big ideas in program synthesis

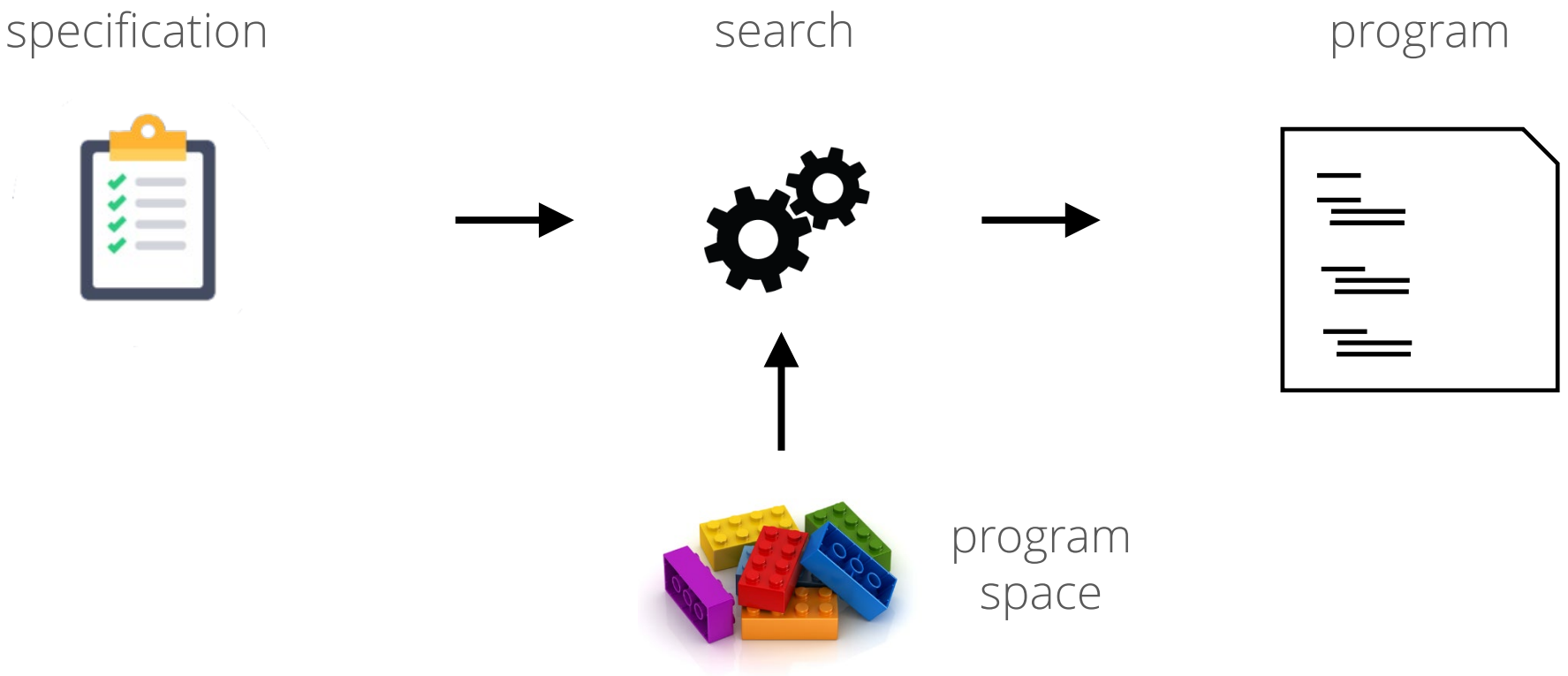
Nadia Polikarpova
PLMW@POPL'23

UC San Diego
COMPUTER SCIENCE & ENGINEERING

goal: automate programming



program synthesis



program synthesis: specs

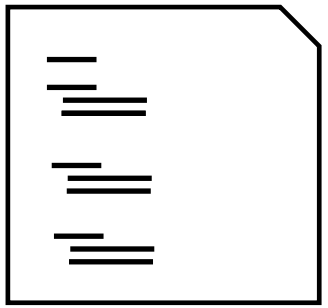
specification

examples
programs
logic
types
natural language
...

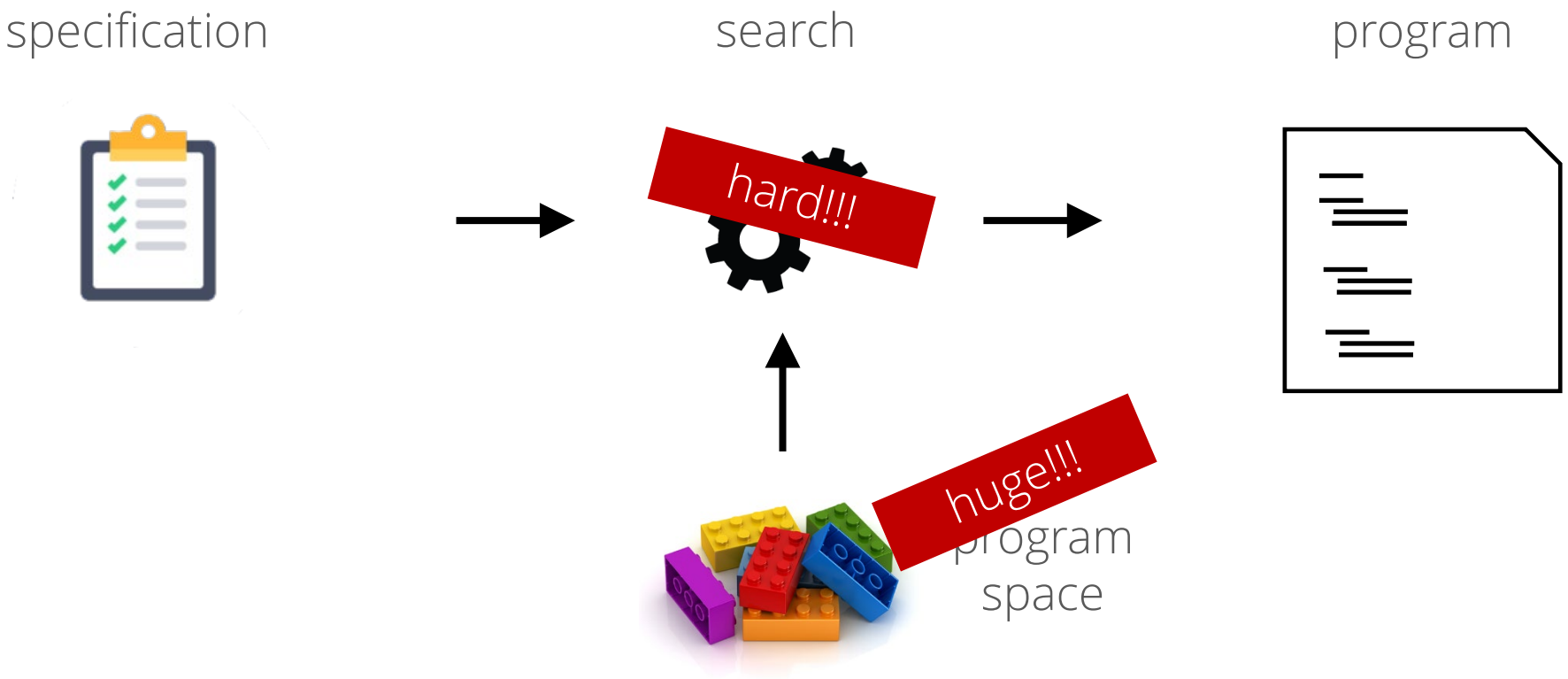
search

program
space

program



program synthesis: challenge



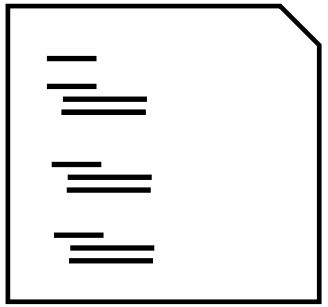
program synthesis: search strategies

specification

search

enumerative
stochastic
constraint-based
neural
...

program



big ideas*

*caution: a non-exhaustive and very subjective list

1. observational equivalence
2. CEGIS
3. deductive synthesis
4. learn while searching

setup

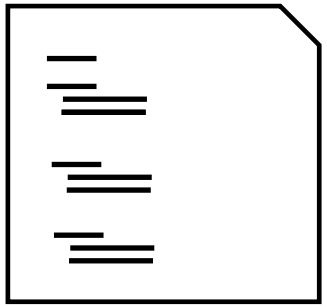
specification

examples
programs
logic
types
natural language
...

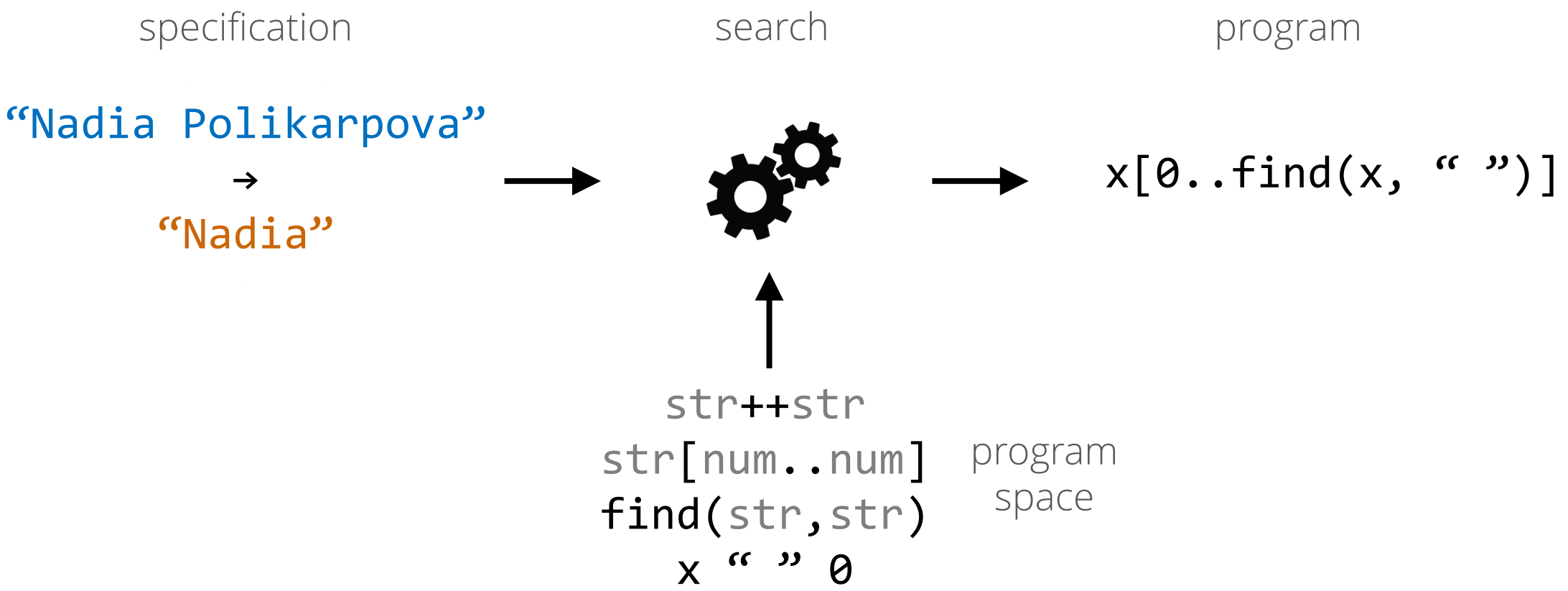
search

enumerative
stochastic
constraint-based
neural
...

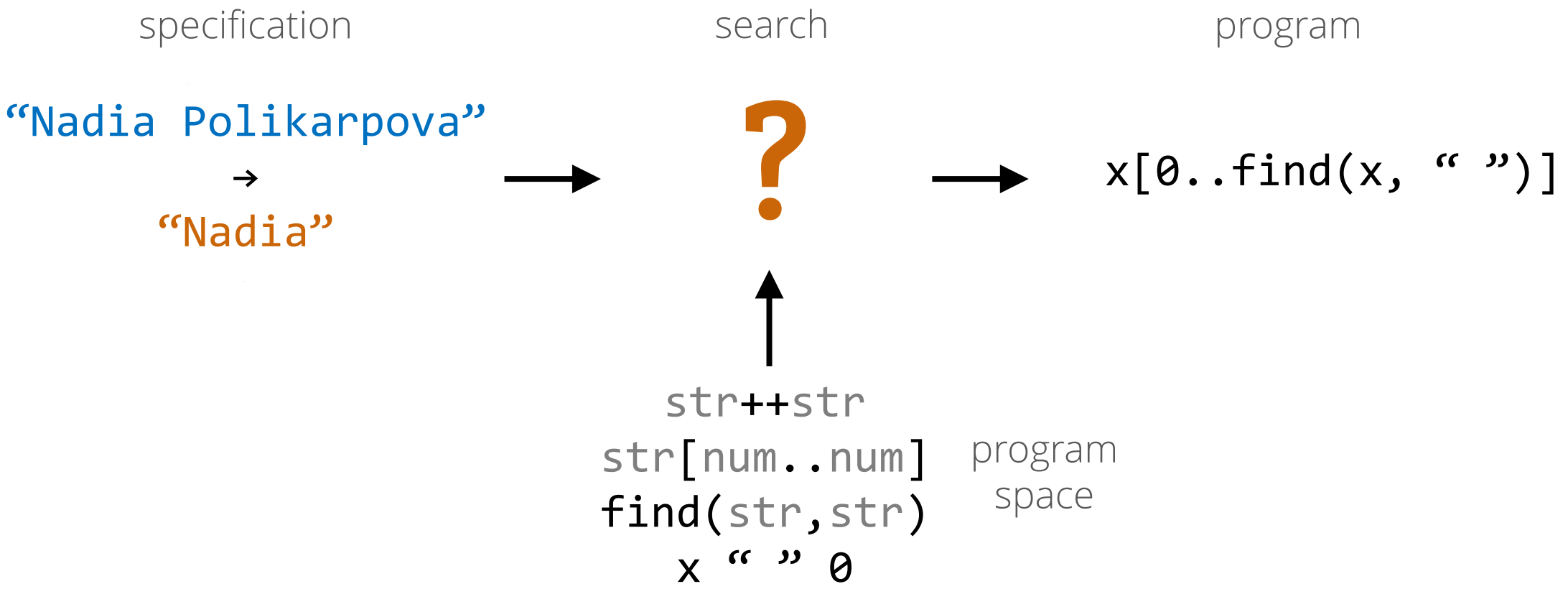
program



example: extract first name



example: extract first name



bottom-up enumeration

program bank

size 1 | x | “ ” | 0

specification

“Nadia Polikarpova” → “Nadia”

search space

```
str++str
str[num..num]
find(str,str)
```

x “ ” 0

bottom-up enumeration

program bank

size 1 `x` `“ ”` `0`

size 3

specification

`“Nadia Polikarpova” → “Nadia”`

search space

`str++str`
`str[num..num]`
`find(str,str)`
`x “ ” 0`

bottom-up enumeration

program bank

size 1 `x` `“ ”` `0`

size 3

specification

`“Nadia Polikarpova” → “Nadia”`

search space

`str++str`

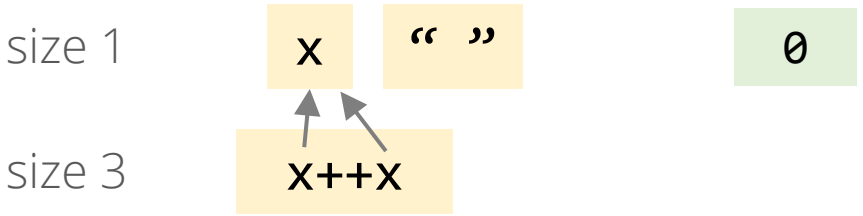
`str[num..num]`

`find(str,str)`

`x “ ” 0`

bottom-up enumeration

program bank



specification

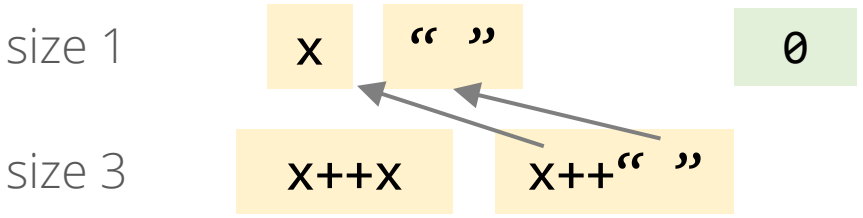
“Nadia Polikarpova” → “Nadia”

search space

```
str++str
str[num..num]
find(str,str)
x “ ” 0
```

bottom-up enumeration

program bank



specification

“Nadia Polikarpova” → “Nadia”

search space

```
str++str
str[num..num]
find(str,str)
x “ ” 0
```

bottom-up enumeration

program bank

size 1 `x` `" "` `0`

size 3 `x++x` `x++" "` `" "++x` `" "++" "`

specification

`"Nadia Polikarpova" → "Nadia"`

search space

`str++str` ←

`str[num..num]`

`find(str,str)`

`x " " 0`

bottom-up enumeration

program bank

size 1 `x` `" "` `0`

size 3 `x++x` `x++" "` `" "++x` `" "++" "`

specification

`"Nadia Polikarpova" → "Nadia"`

search space

```
str++str
str[num..num]
find(str,str)
```

`x " " 0`

bottom-up enumeration

program bank

size 1 `x` `" "` `0`

size 3 `x++x` `x++" "` `" "++x` `" "++" "`

specification

`"Nadia Polikarpova" → "Nadia"`

search space

```
str++str
str[num..num]
find(str,str)
x " " 0
```

bottom-up enumeration

program bank

size 1 `x` `" "` `0`

size 3 `x++x` `x++" "` `" "++x` `" "++" "`

specification

`"Nadia Polikarpova" → "Nadia"`

search space

```
str++str
str[num..num]
find(str,str)
```

`x " " 0`

bottom-up enumeration

program bank

size 1 `x` `" "` `0`

size 3 `x++x` `x++" "` `" "++x` `" "++" "`

specification

`"Nadia Polikarpova" → "Nadia"`

search space

```
str++str
str[num..num]
find(str,str)
x " " 0
```

bottom-up enumeration

program bank

size 1 `x` `" "` `0`

size 3 `x++x` `x++" "` `" "++x` `" "++" "`

`find(x,x)` `find(x," ")` `find(" ",x)` `find(" "," ")`

specification

`"Nadia Polikarpova" → "Nadia"`

search space

`str++str`
`str[num..num]`
`find(str,str)` ←
`x " " 0`

bottom-up enumeration

program bank

size 1 `x` `" "` `0`

size 3 `x++x` `x++" "` `" "++x` `" "++" "`

`find(x,x)` `find(x," ")` `find(" ",x)` `find(" "," ")`

specification

`"Nadia Polikarpova" → "Nadia"`

search space

```
str++str
str[num..num]
find(str,str)
x " " 0
```

bottom-up enumeration

program bank

size 1 `x` `" "` `0`

size 3 `x++x` `x++" "` `" "++x` `" "++" "`

`find(x,x)` `find(x," ")` `find(" ",x)` `find(" "," ")`

size 4

specification

`"Nadia Polikarpova" → "Nadia"`

search space

```
str++str
str[num..num]
find(str,str)
x " " 0
```

bottom-up enumeration

program bank

size 1 `x` `" "` `0`

size 3 `x++x` `x++" "` `" "++x` `" "++" "` `find(x,x)` `find(x," ")` `find(" ",x)` `find(" "," ")`

size 4 `x[0..0]` `" "[0..0]`

specification

`"Nadia Polikarpova" → "Nadia"`

search space

```
str++str
str[num..num]
find(str,str)
x " " 0
```

bottom-up enumeration

program bank

size 1 `x` `" "` `0`

size 3 `x++x` `x++" "` `" "++x` `" "++" "` `find(x,x)` `find(x," ")` `find(" ",x)` `find(" "," ")`

size 4 `x[0..0]` `" "[0..0]`

size 5

specification

`"Nadia Polikarpova" → "Nadia"`

search space

```
str++str
str[num..num]
find(str,str)
x " " 0
```

bottom-up enumeration

program bank

size 1: `x` `" "` `0`

size 3: `x++x` `x++" "` `" "++x` `" "++" "` `find(x,x)` `find(x," ")` `find(" ",x)` `find(" "," ")`

size 4: `x[0..0]` `" "[0..0]`

size 5: `x++(x++x)` `x++(x++" ")` `x++(" "++x)` `x++(" "++" ")` `(x++x)++x` `(x++" ")++x` `(x++x)+" "` `(x++" ")++" "` `" "++(x++x)` `(" "++x)++x` … `find(x,x++x)` `find(x,x++" ")` `find(x, " "++x)` `find(x, " "++" ")` …

specification

`"Nadia Polikarpova" → "Nadia"`

search space

```
str++str
str[num..num]
find(str,str)
x " " 0
```

bottom-up enumeration

program bank

size 1 `x` `" "` `0`

size 3 `x++x` `x++" "` `" "++x` `" "++" "`

`find(x,x)` `find(x," ")` `find(" ",x)` `find(" "," ")`

size 4 `x[0..0]` `" "[0..0]`

size 5 `x++(x++x)` `x++(x++" ")` `x++(" "++x)` `x++(" "++" ")` `(x++x)++x` `(x++" ")++x`

`(x++x)+" "` `(x++" ")++" "` `" "++(x++x)` `(" "++x)++x` …

`find(x,x++x)` `find(x,x++" ")` `find(x, " "++x)` `find(x, " "++" ")` …

size 6

specification

`"Nadia Polikarpova" → "Nadia"`

search space

```
str++str
str[num..num]
find(str,str)
x " " 0
```

bottom-up enumeration

program bank

size 1 `x` `" "` `0`

size 3 `x++x` `x++" "` `" "++x` `" "++" "`

`find(x,x)` `find(x," ")` `find(" ",x)` `find(" "," ")`

size 4 `x[0..0]` `" "[0..0]`

size 5 `x++(x++x)` `x++(x++" ")` `x++(" "++x)` `x++(" "++" ")` `(x++x)++x` `(x++" ")++x`

`(x++x)+" "` `(x++" ")++" "` `" "++(x++x)` `(" "++x)++x` …

`find(x,x++x)` `find(x,x++" ")` `find(x, " "++x)` `find(x, " "++" ")` …

size 6 … `x[0..find(x, " ")]`

specification

`"Nadia Polikarpova" → "Nadia"`

search space

```
str++str
str[num..num]
find(str,str)
x " " 0
```

bottom-up enumeration

program bank

size 1 `x` `" "` `0`

size 3 `x++x` `x++" "` `" "++x` `" "++" "`

`find(x,x)` `find(x," ")` `find(" ",x)` `find(" "," ")`

size 4 `x[0..0]` `" "[0..0]`

size 5 `x++(x++x)` `x++(x++" ")` `x++(" "++x)` `x++(" "++" ")` `(x++x)++x` `(x++" ")++x`

`(x++x)+" "` `(x++" ")++" "` `" "++(x++x)` `(" "++x)++x` …

`find(x,x++x)` `find(x,x++" ")` `find(x, " "++x)` `find(x, " "++" ")` …

size 6 … `x[0..find(x, " ")]` 

specification

`"Nadia Polikarpova" → "Nadia"`

search space

```
str++str
str[num..num]
find(str,str)
x " " 0
```

bottom-up enumeration

program bank

size 1: `x` `" "` `0`

size 3: `x++x` `x++" "` `" "++x` `" "++" "`

`find(x,x)` `find(" ",x)` `find(" "," ")`

size 4: `x[0..0]` `" "[0..0]`

size 5: `x++(x++x)` `x++(x++" ")` `x++(" "++x)` `x++(" "++" ")` `(x++x)++x` `(x++" ")++x`

`(x++x)+" "` `(x++" ")++" "` `" "++(x++x)` `(" "++x)++x` …

`find(x,x++x)` `find(x,x++" ")` `find(x, " "++x)` `find(x, " "++" ")` …

size 6: … `x[0..find(x, " ")]`

explodes quickly!!!

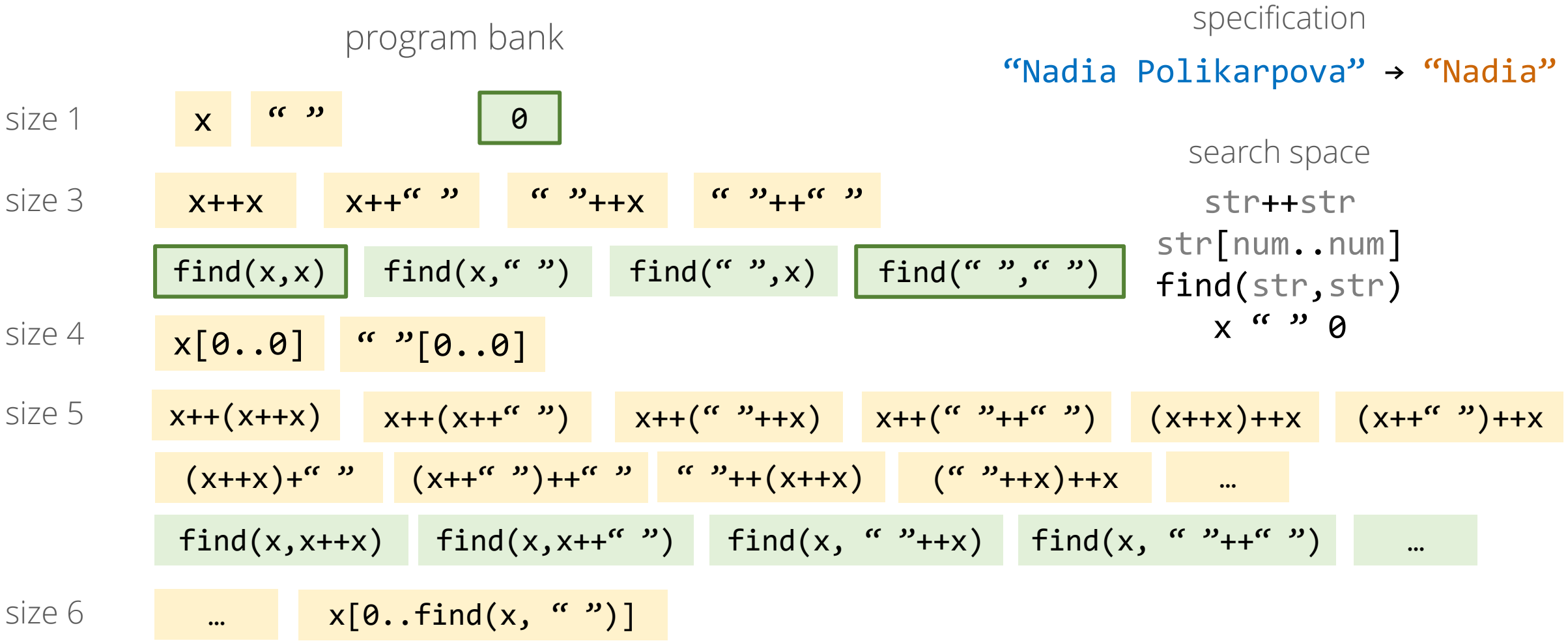
specification

`"Nadia Polikarpova" → "Nadia"`

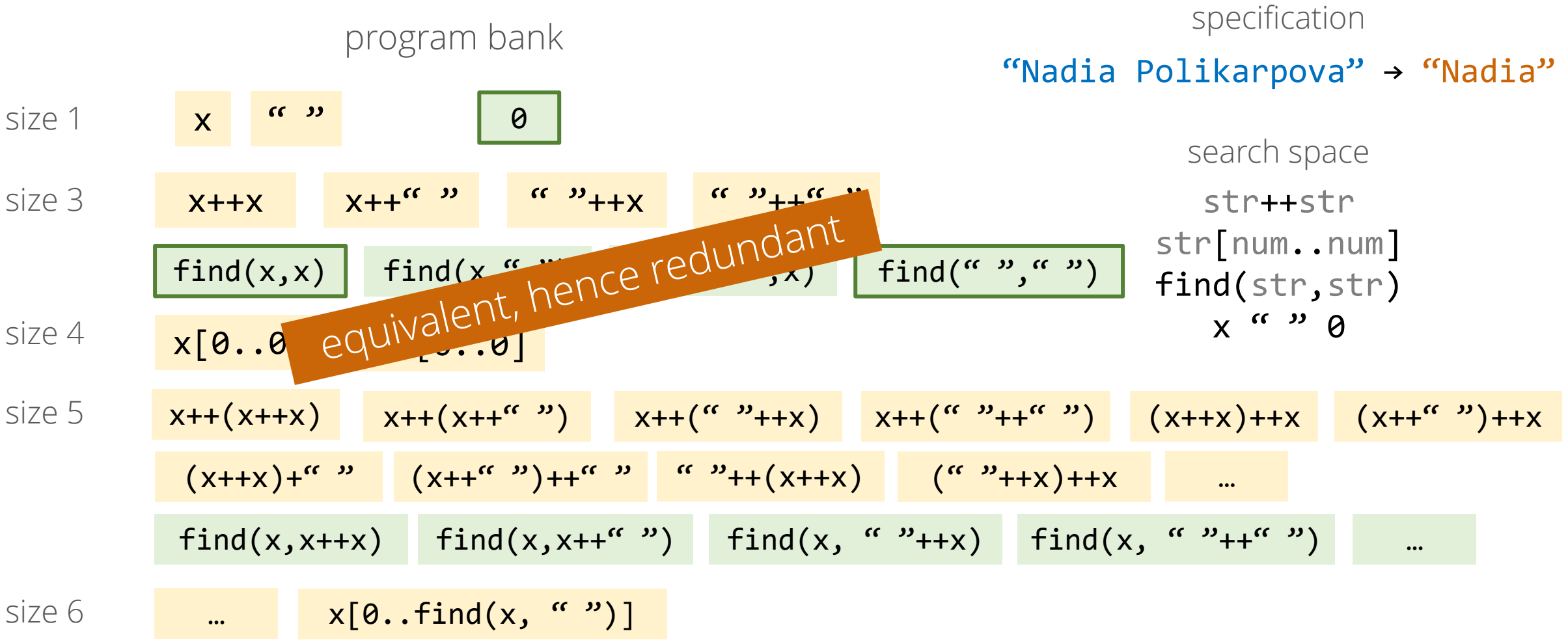
search space

```
str++str
str[num..num]
find(str,str)
  x " " 0
```

pruning redundant programs



pruning redundant programs



pruning redundant programs

program bank

specification

“Nadia Polikarpova” → “Nadia”

search space

str++str
str[num..num]
find(str,str)
x “ ” 0

size 1: `x` `“ ”` `0`

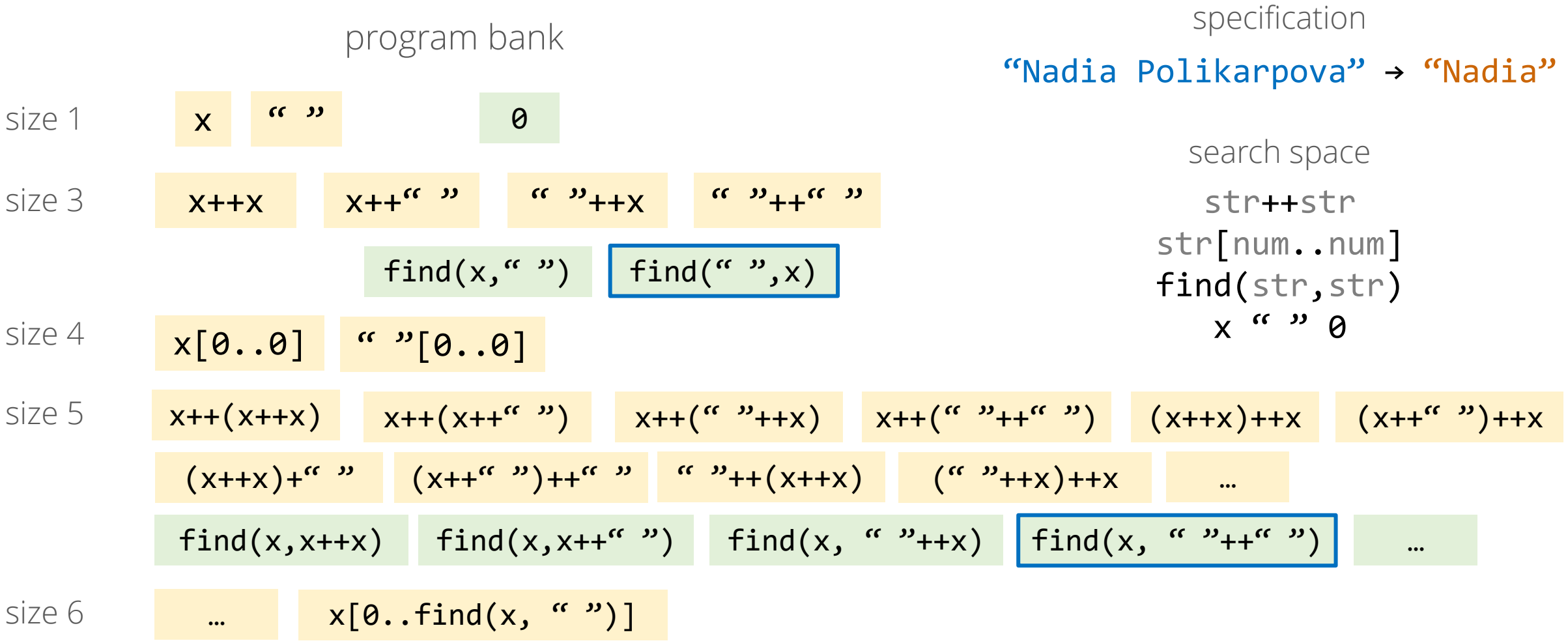
size 3: `x++x` `x++“ ”` `“ ”++x` `“ ”++“ ”` `find(x “ ”` `,x)`

size 4: `x[0..0` `[0..0]` — equivalent, hence redundant

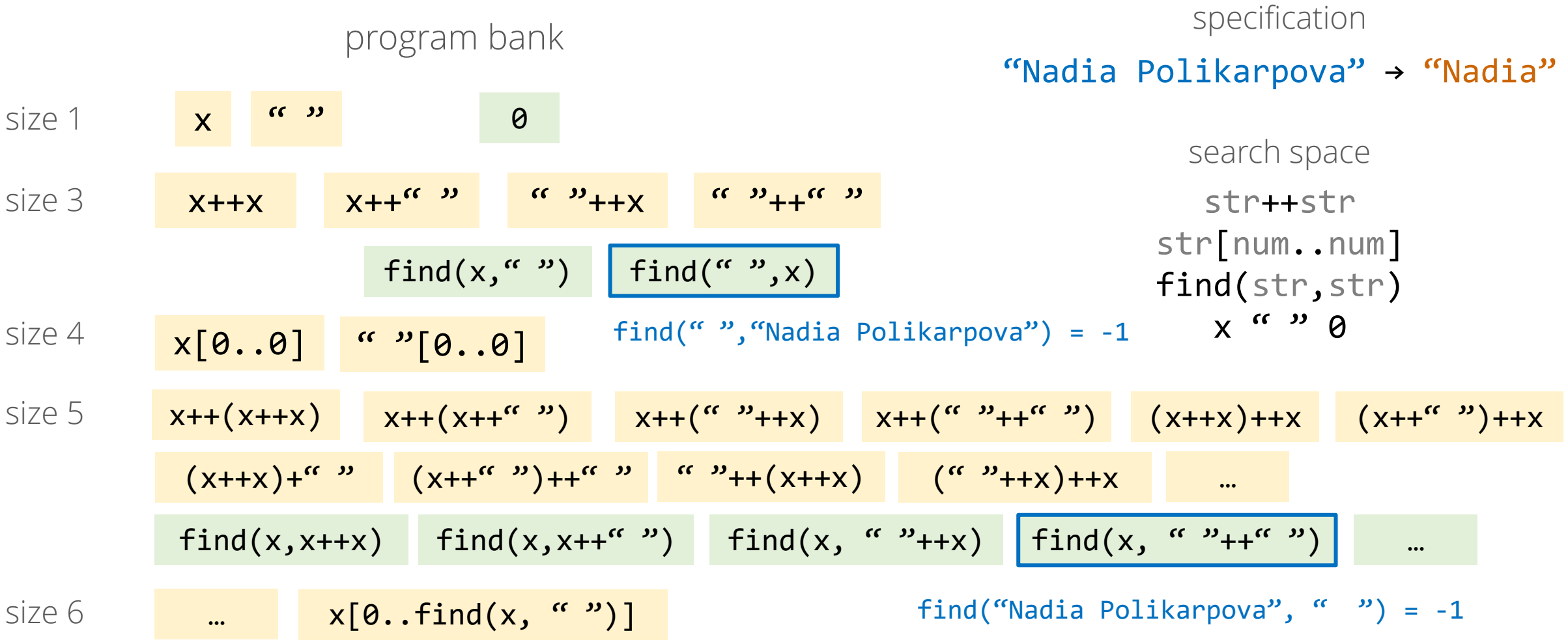
size 5: `x++(x++x)` `x++(x++“ ”)` `x++(“ ”++x)` `x++(“ ”++“ ”)` `(x++x)++x` `(x++“ ”)++x`
`(x++x)+“ ”` `(x++“ ”)++“ ”` `“ ”++(x++x)` `(“ ”++x)++x` `…`
`find(x,x++x)` `find(x,x++“ ”)` `find(x, “ ”++x)` `find(x, “ ”++“ ”)` `…`

size 6: `…` `x[0..find(x, “ ”)]`

pruning redundant programs



pruning redundant programs



pruning redundant programs

program bank

specification

“Nadia Polikarpova” → “Nadia”

search space

str++str
str[num..num]
find(str,str)
x “ ” 0

size 1 `x` `“ ”` `0`

size 3 `x++x` `x++“ ”` `“ ”++x` `“ ”++“ ”`

`find(x,“ ”)` `find(“ ”,x)`

size 4 `x[0..0]` `“ ”[0..0]`

`find(“ ”,“Nadia Polikarpova”) = -1`

size 5 `x++(x++x)` `x++(x++“ ”)` `(x++x)++x` `(x++“ ”)++x`

`(x++x)+“ ”` `(“ ”++x)++x` …

`find(x,x+` `(x,x++“ ”)` `find(x, “ ”++x)` `find(x, “ ”++“ ”)` …

equivalent on our spec, also redundant!

size 6 … `x[0..find(x, “ ”)]`

`find(“Nadia Polikarpova”, “ ”) = -1`

pruning redundant programs

program bank

specification

“Nadia Polikarpova” → “Nadia”

search space

str++str
str[num..num]
find(str,str)
x “ ” 0

size 1 `x` `“ ”` `0`

size 3 `x++x` `x++“ ”` `“ ”++x` `“ ”++“ ”` `find(x,“ ”)` `find(“ ”,x)`

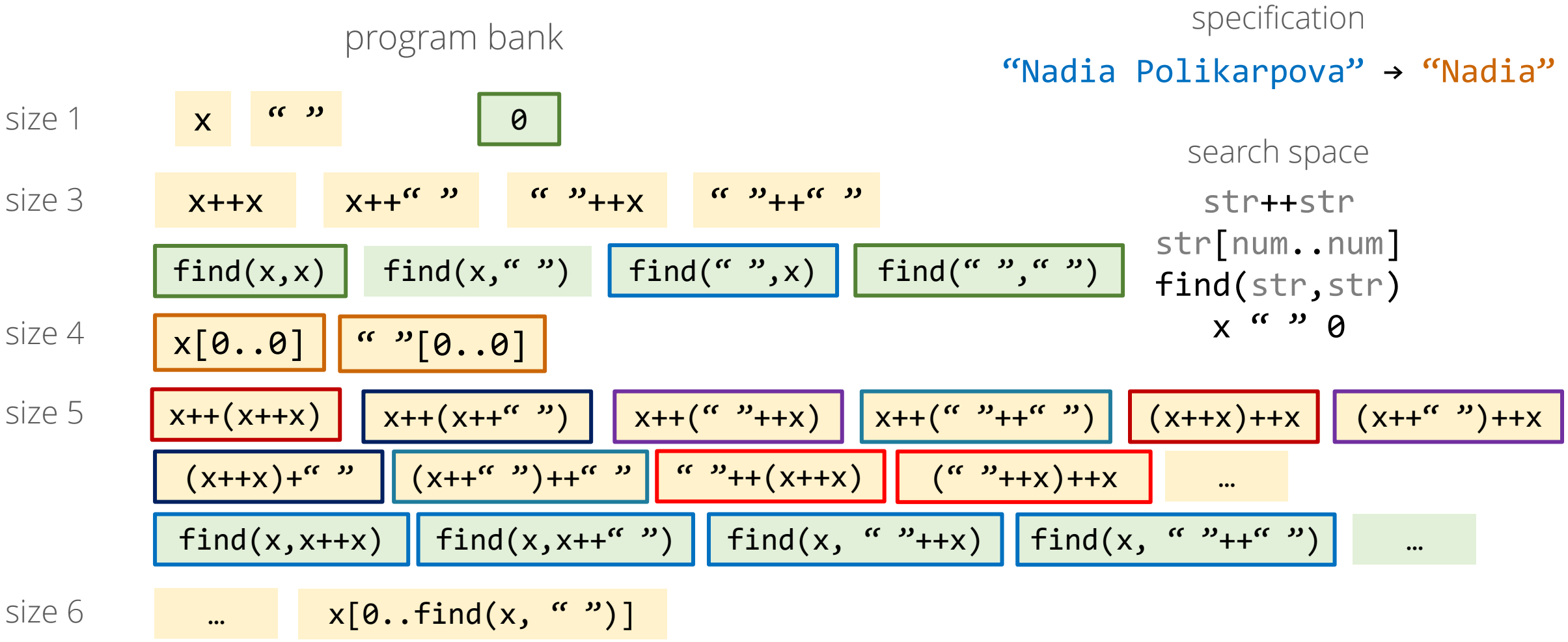
size 4 `x[0..0]` `“ ”[0..0]`

size 5 `x++(x++x)` `x++(x++“ ”)` `(x++x)++x` `(x++“ ”)++x` `(x++x)+“ ”` `(“ ”++x)++x` … `find(x,x+` `find(x,x++“ ”)` `find(x, “ ”++x)` …

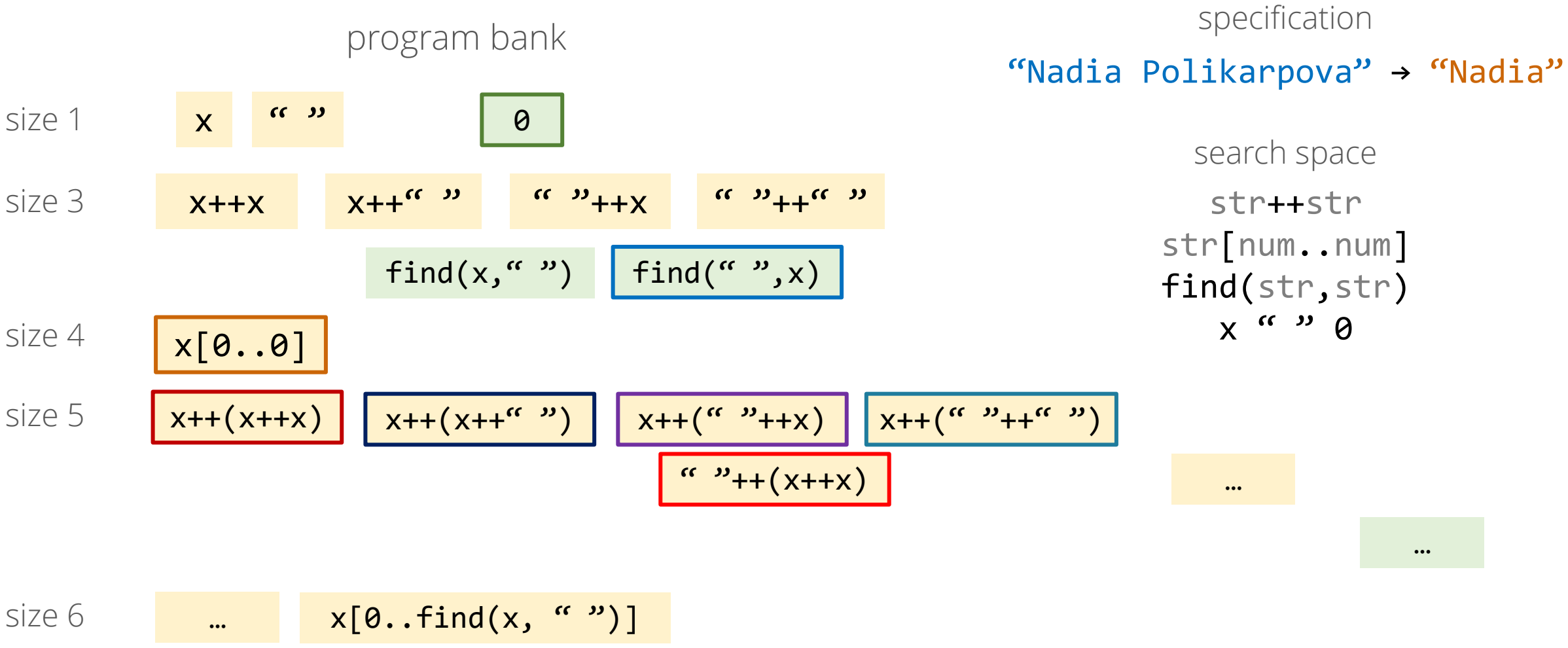
equivalent on our spec, also redundant!

size 6 … `x[0..find(x, “ ”)]`

pruning redundant programs



pruning redundant programs



observational equivalence

big idea:
programs that behave the same on spec are redundant

- only need to keep one
- savings compound!

demo

snippy

references

Aws Albarghouthi, Sumit Gulwani, Zachary Kincaid
Recursive Program Synthesis
CAV 2013

Abhishek Udupa et al.
TRANSIT: specifying protocols with concolic snippets
PLDI 2013

Shraddha Barke, Hila Peleg, Nadia Polikarpova
Just-in-Time Learning for Bottom-Up Enumerative Synthesis
OOPSLA 2020

1. observational equivalence
2. **CEGIS**
3. deductive synthesis
4. learn while searching

1. observational equivalence
2. **Counter-Example Guided Inductive Synthesis**
3. deductive synthesis
4. learn while searching

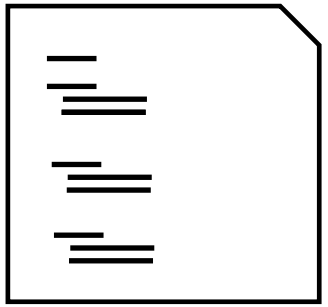
setup

specification

examples
programs
logic
types
natural language
...

search

program

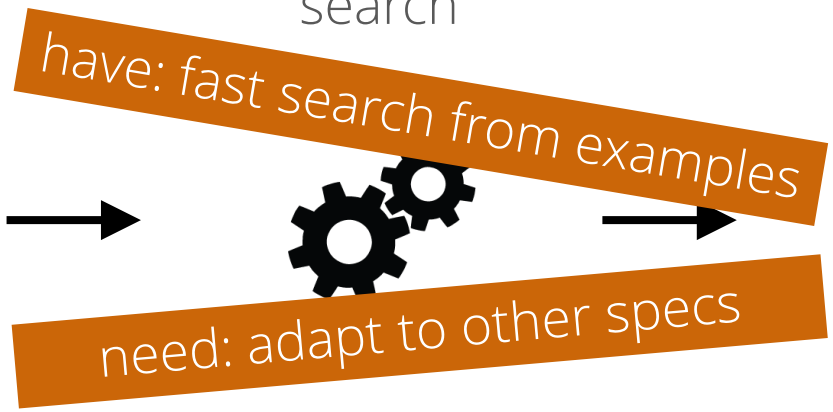


setup

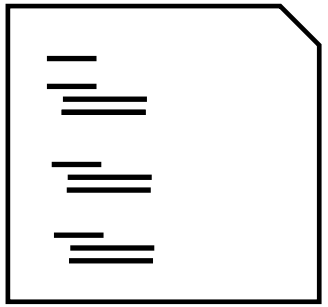
specification

examples
programs
logic
types
natural language
...

search



program



example: isolate zero

problem: isolate least significant zero bit in a bitvector

```
0010 0101  →  0000 0010
```

easy to implement as a loop:

```
bit[W] isolate0 (bit[W] x) {  // W: bitvector size
    bit[W] res = 0;
    for (int i = 0; i < W; i++)
        if (!x[i]) { res[i] = 1; return res; }
}
```

can we do this more efficiently with bitwise operations?

example: isolate zero

specification

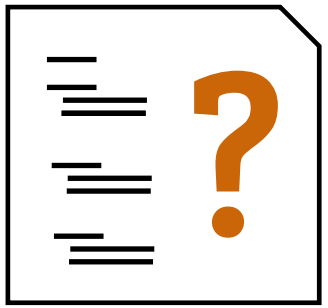
```
bit[W] isolate0 (bit[W] x) {
    bit[W] res = 0;
    for (int i = 0; i < W; i++)
        if (!x[i]) { res[i] = 1;
                     return res;
                   }
}
```

search

~bv
bv+bv bv|bv
bv&bv bv^bv
x 0 1 …

program space

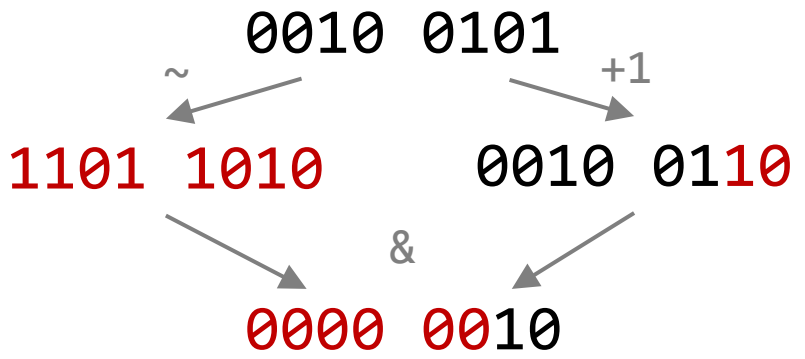
program



demo

sketch

demo



sketch

```
~x & (x + 1)
```

example: isolate zero

specification

```
bit[W] isolate0 (bit[W] x) {
    bit[W] res = 0;
    for (int i = 0; i < W; i++)
        if (!x[i]) { res[i] = 1;
                     return res;
                   }
}
```

search

~bv

bv+bv bv|bv

bv&bv bv^bv

x 0 1 …

program space

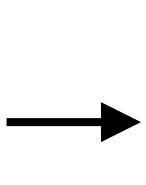
program

```
~x & (x + 1)
```

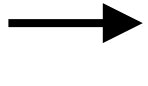
example: isolate zero

specification

```
bit[W] isolate0 (bit[W] x) {
    bit[W] res = 0;
    for (int i = 0; i < W; i++)
        if (!x[i]) { res[i] = 1;
                     return res;
                   }
}
```

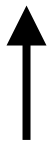


search



program

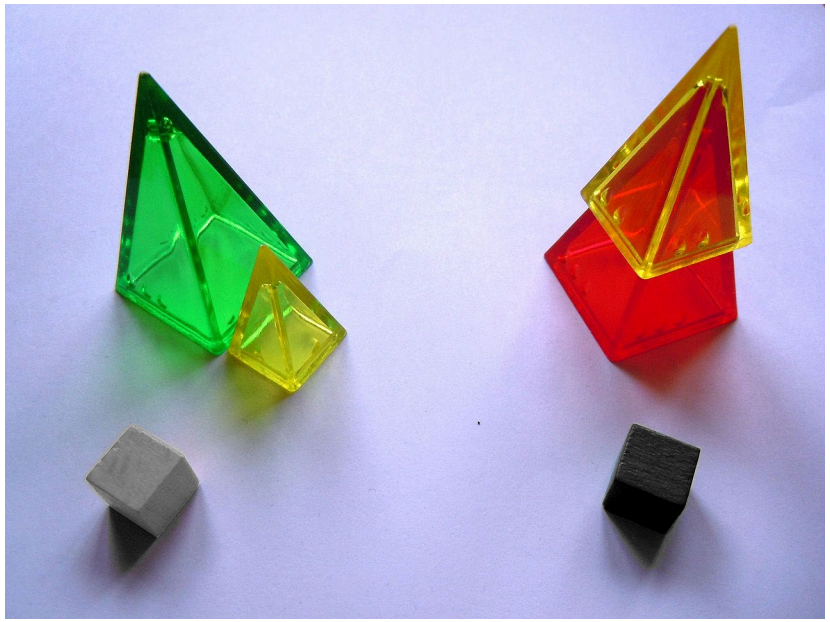
```
~x & (x + 1)
```



~bv
bv+bv bv|bv
bv&bv bv^bv
x 0 1 …

program space

the zendo game

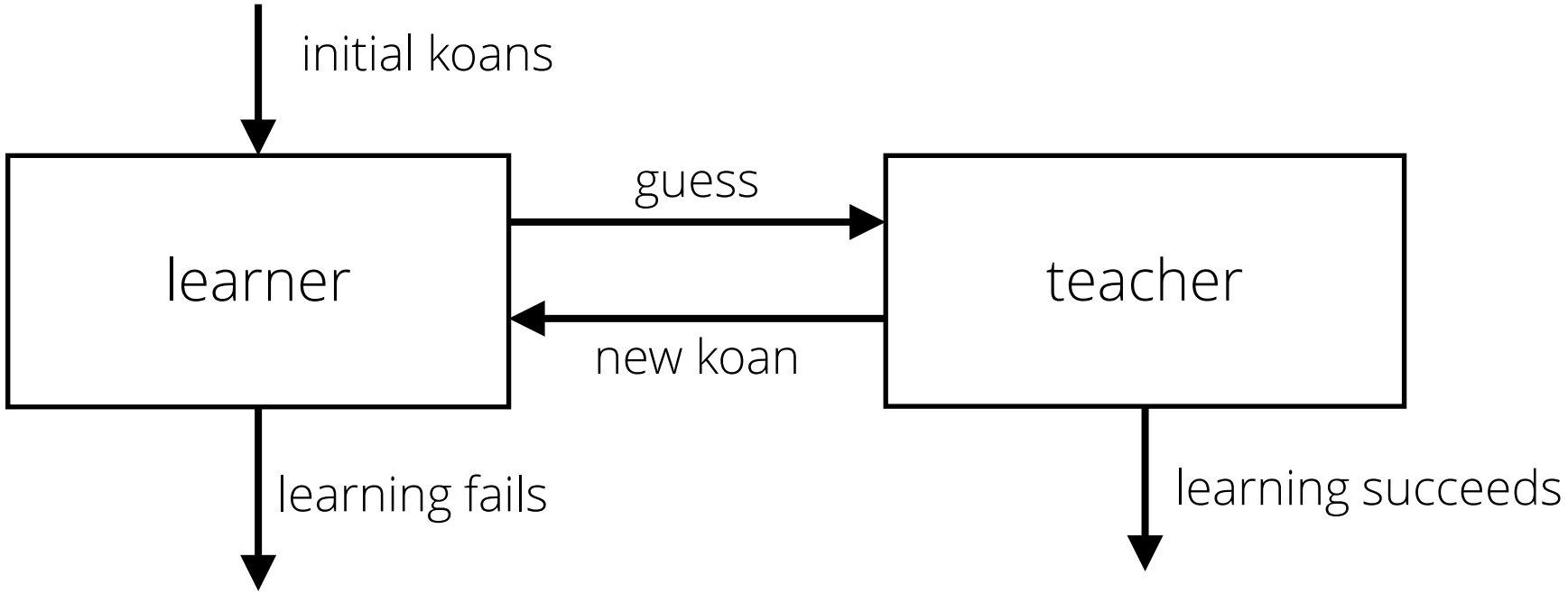


i made up a secret rule

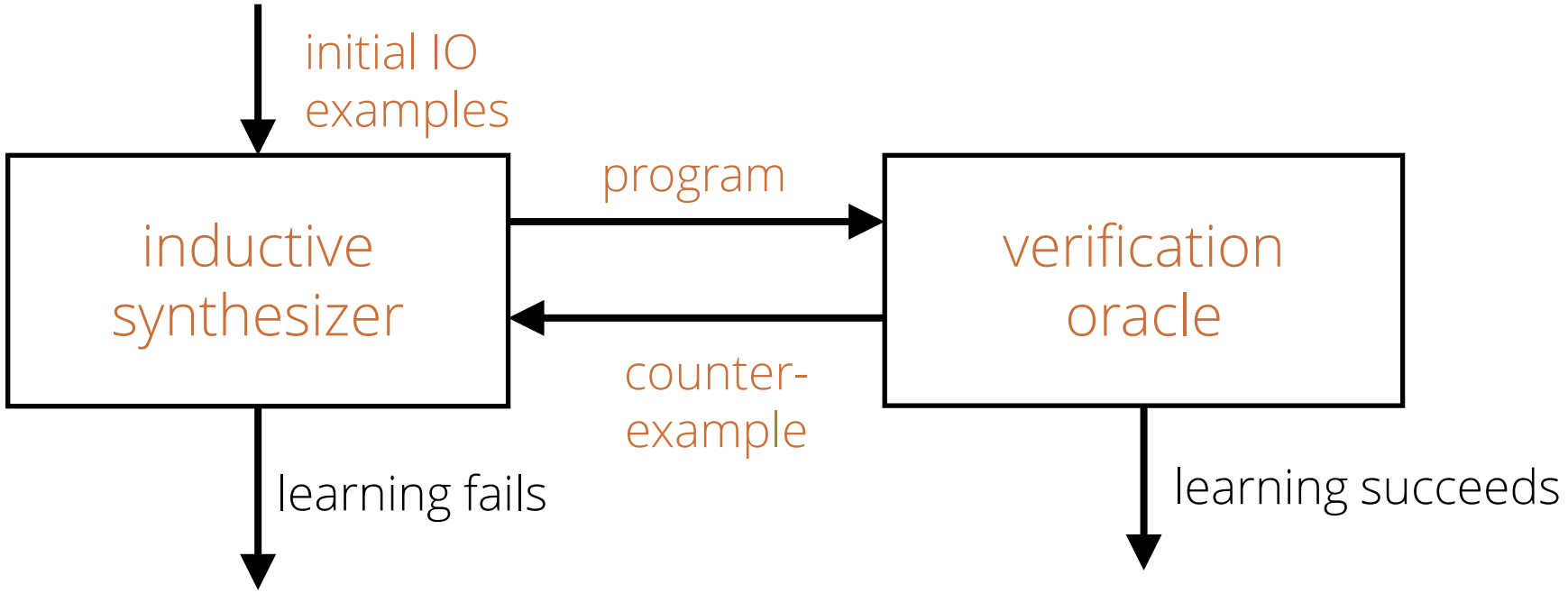
left “koan” satisfies the rule
and right does not

can you guess the rule?

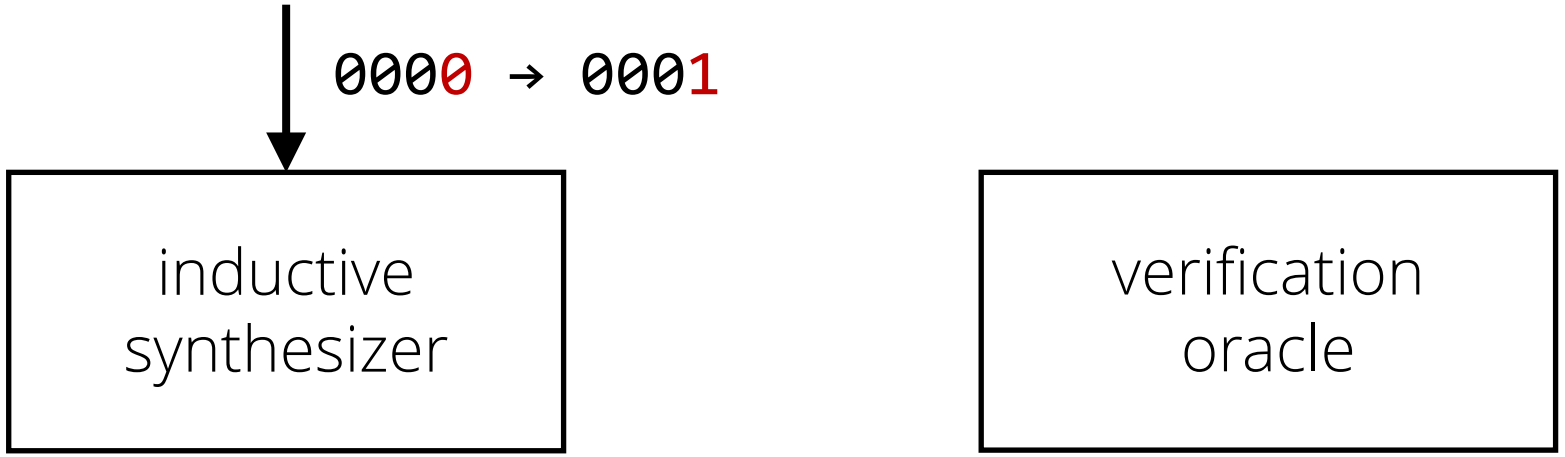
the zendo game



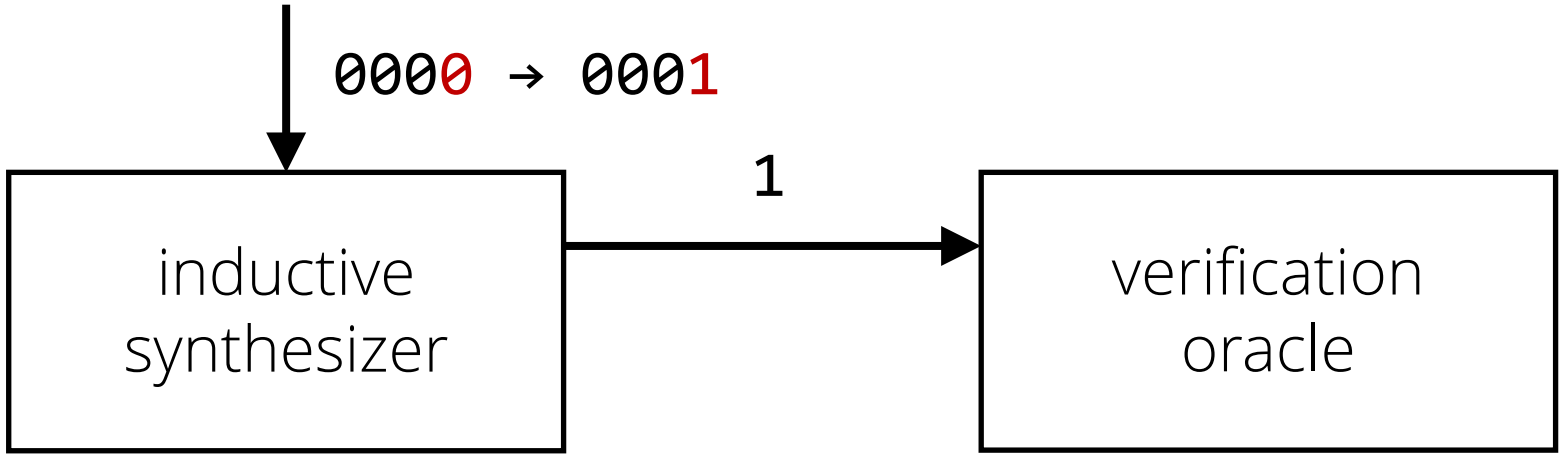
CEGIS = the zendo of program synthesis



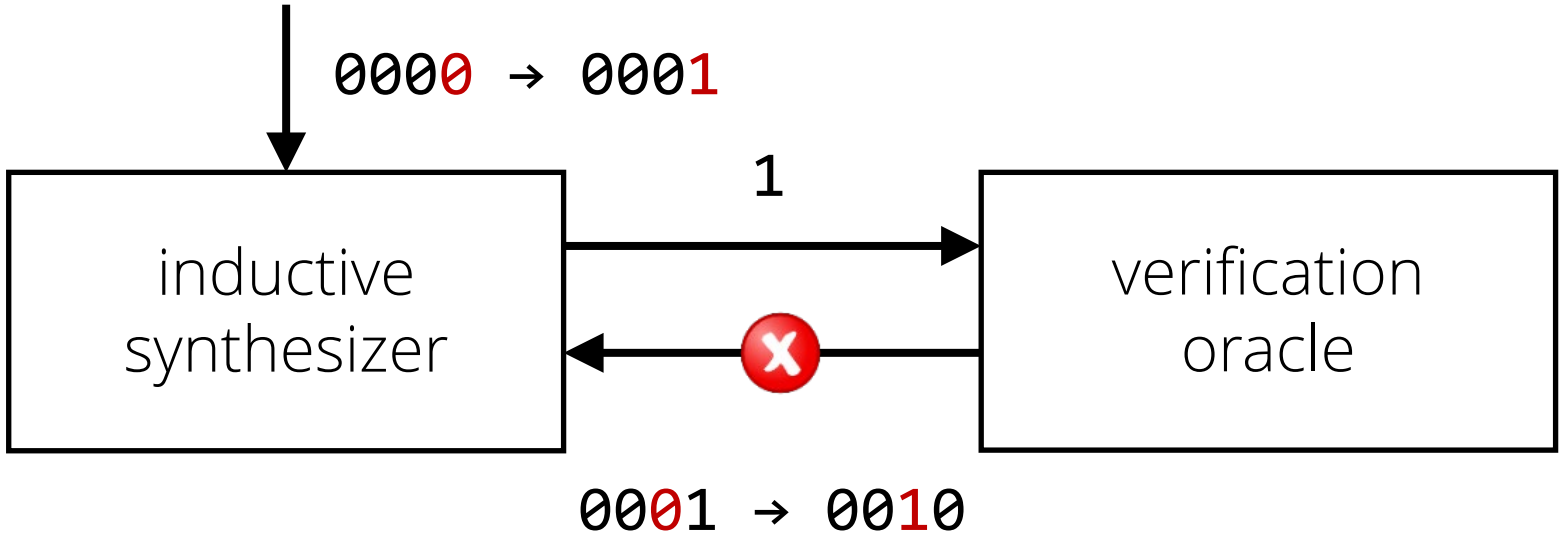
example: isolate zero



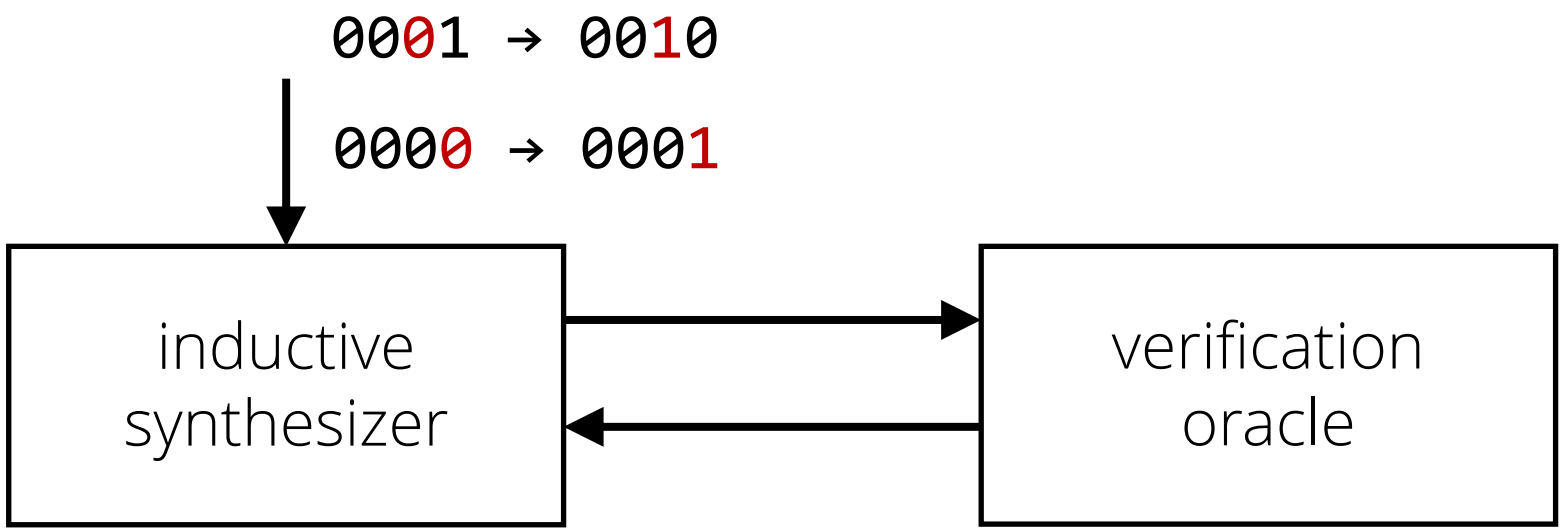
example: isolate zero



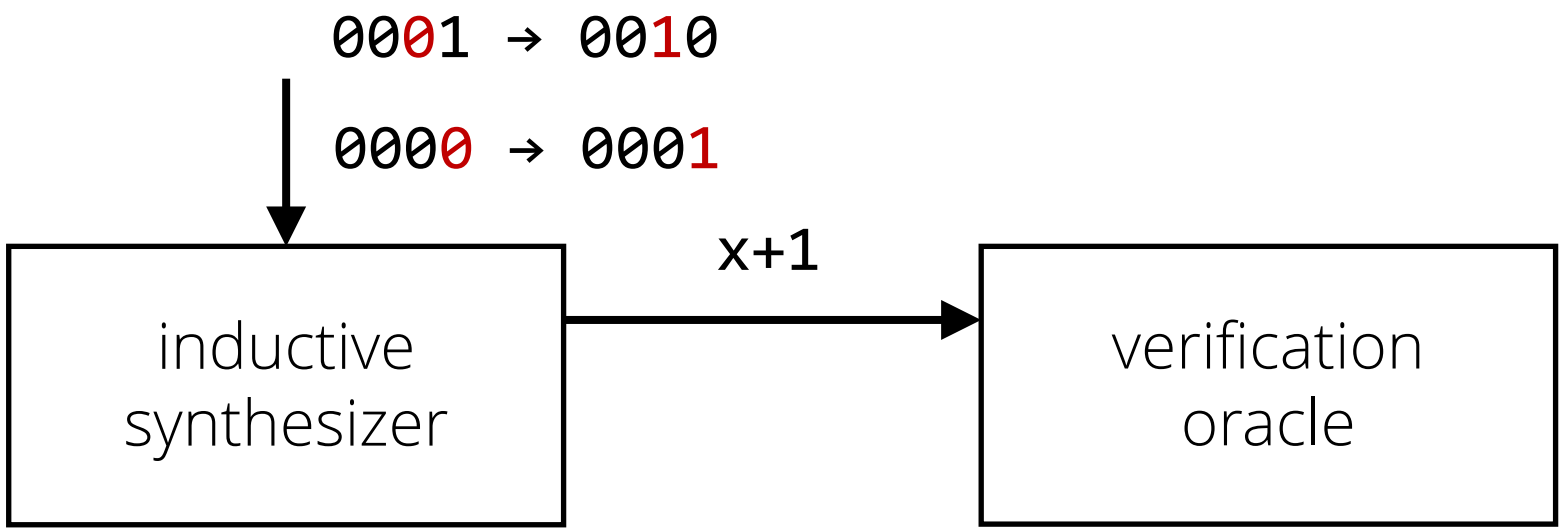
example: isolate zero



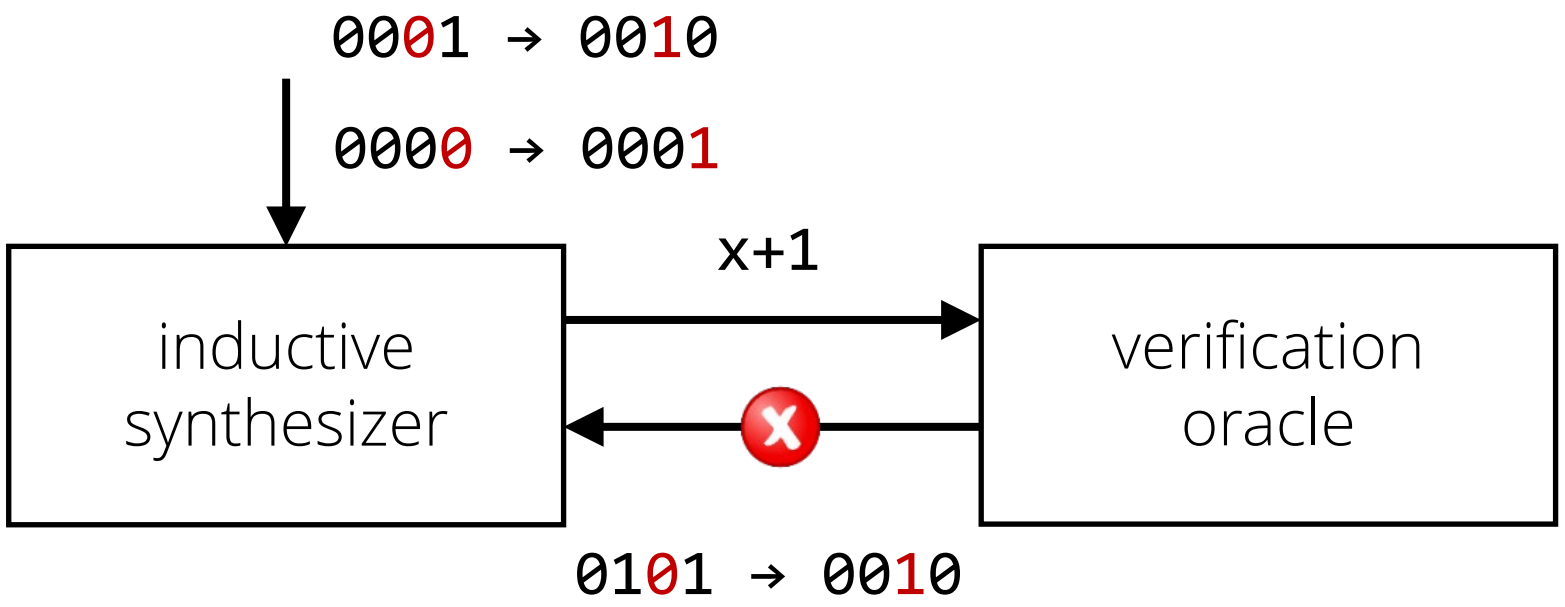
example: isolate zero



example: isolate zero

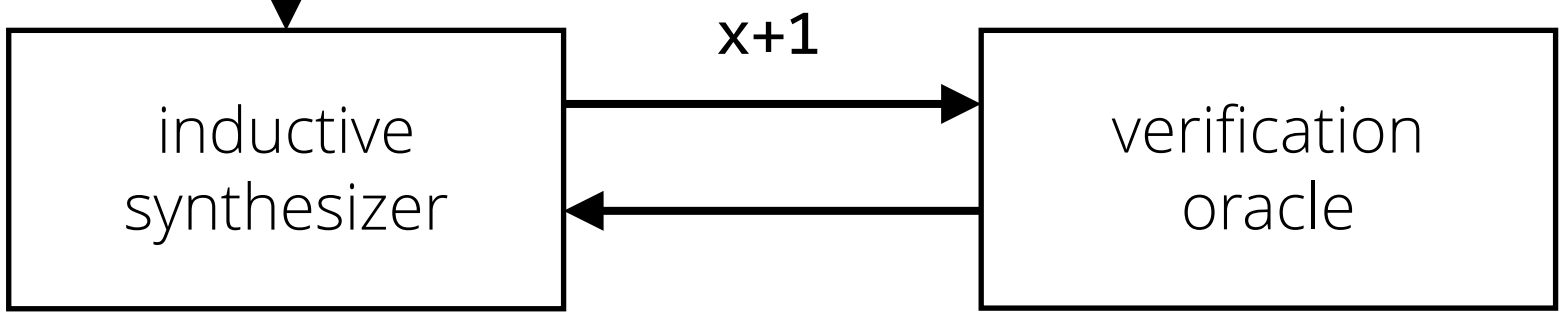


example: isolate zero

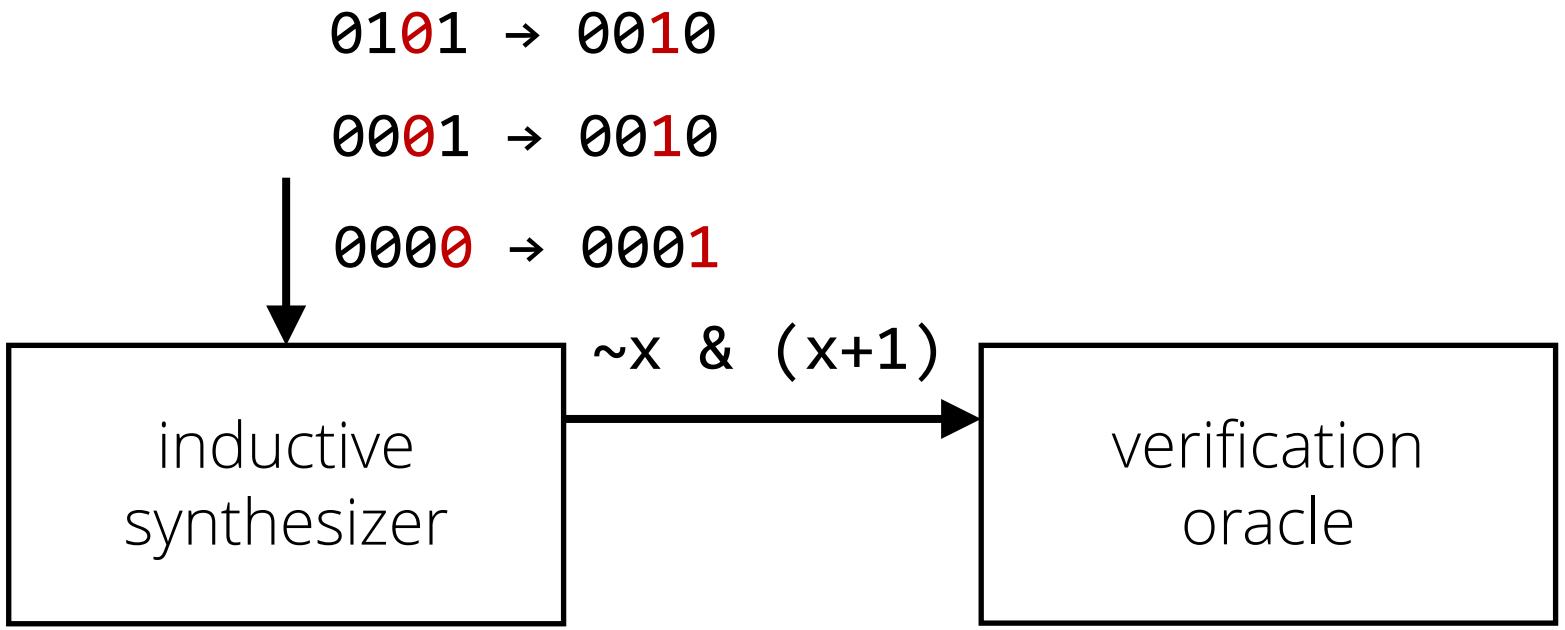


example: isolate zero

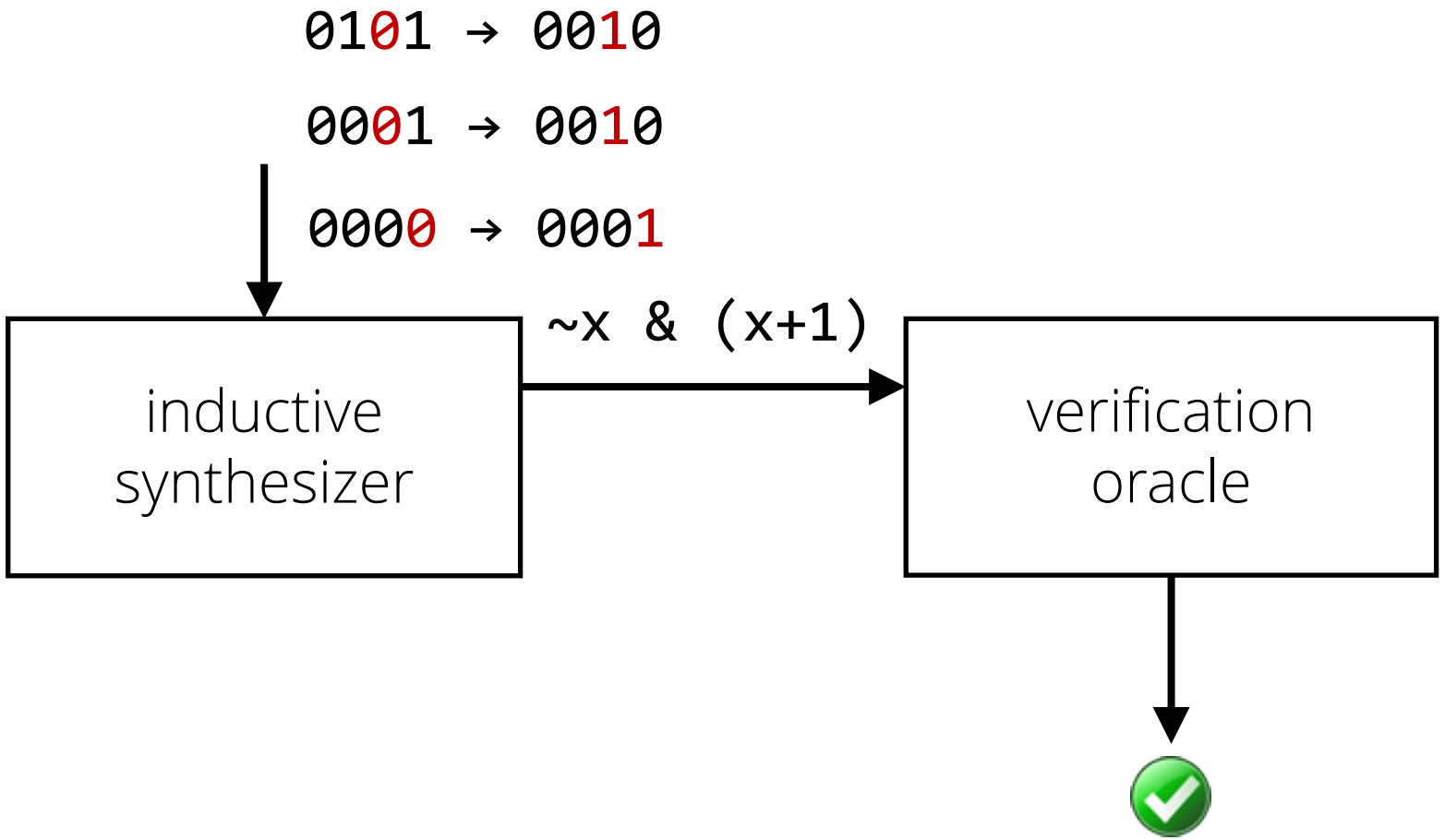
```
0101 → 0010
0001 → 0010
0000 → 0001
```



example: isolate zero



example: isolate zero



CEGIS

big idea:
iteratively learn from counter-examples

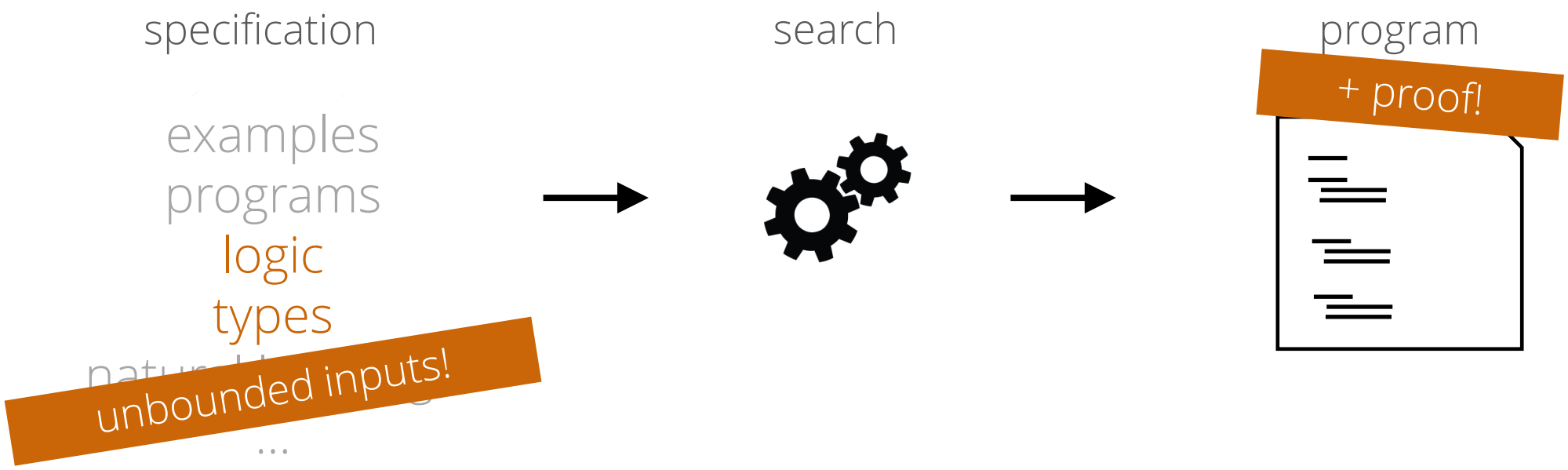
- inductive synthesizer + verification oracle
- can use constraint solver to implement both!

references

Armando Solar-Lezama:
Program sketching
STTT'13

1. observational equivalence
2. CEGIS
3. **deductive synthesis**
4. learn while searching

setup


specification
examples
programs
logic
types
unbounded inputs!
...
search
program
+ proof!

challenge 1: too many programs

challenge 1: too many programs

challenge 2: checking each program is hard

deductive synthesis

big idea: look for the proof to find the program

use spec to guide search!

example: swap

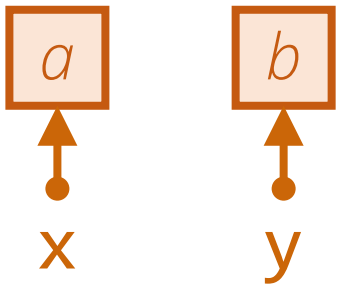
swap values of two distinct pointers in C*

```
void swap(int* x, int* y)
```

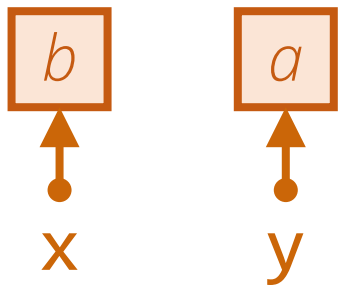
*simple C-like language with pointers

specifying swap in separation logic

start state:



end state:



in separation logic:

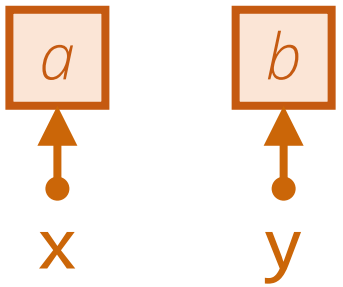
$\{ x \mapsto a * y \mapsto b \}$

```
void swap(int* x, int* y)
```

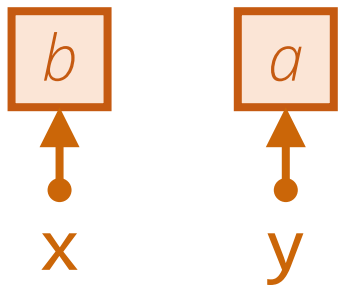
$\{ x \mapsto b * y \mapsto a \}$

specifying swap in separation logic

start state:



end state:

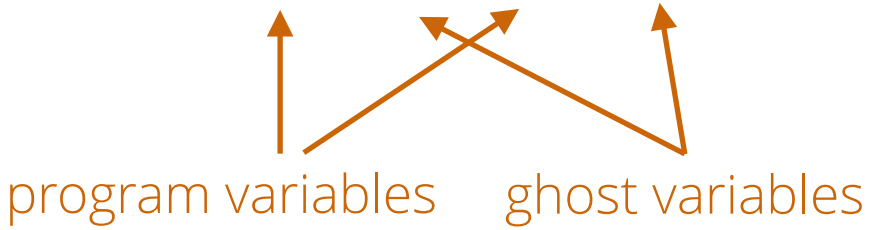


in separation logic:

$\{ x \mapsto a * y \mapsto b \}$

```
void swap(int* x, int* y)
```

$\{ x \mapsto b * y \mapsto a \}$



example: swap

specification

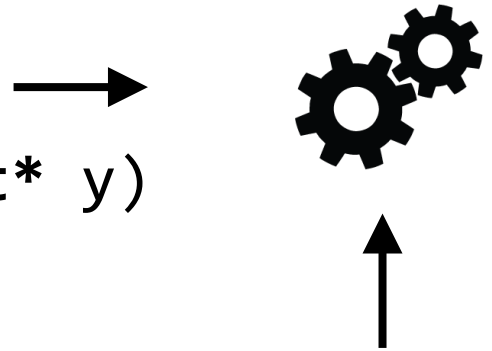
search

program

$\{ x \mapsto a * y \mapsto b \}$

```
void swap(int* x, int* y)
```

$\{ x \mapsto b * y \mapsto a \}$



```
void swap(int* x, int* y) {
  int a1 = *x;
  int b1 = *y;
  *x = b1;
  *y = a1;
}
```

C programs

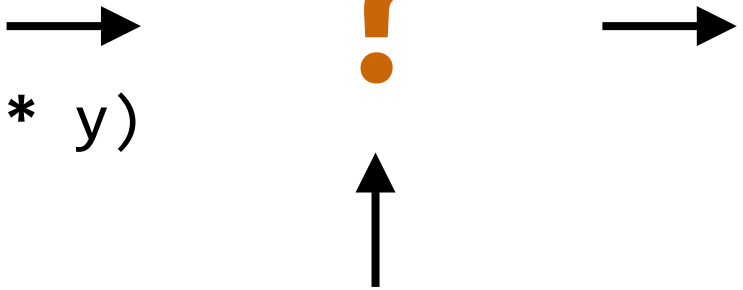
example: swap

specification

$\{ x \mapsto a * y \mapsto b \}$

```
void swap(int* x, int* y)
```

$\{ x \mapsto b * y \mapsto a \}$



search

program

```
void swap(int* x, int* y) {
  int a1 = *x;
  int b1 = *y;
  *x = b1;
  *y = a1;
}
```

C programs

unguided search

```
void swap(int* x, int* y) {

    ??

}
```

unguided search

```
void swap(int* x, int* y) {
    int n = *x;
    *y = x;
    int* z = malloc(2023);
}
```

```
int n = *(x+100);
```

??

unguided search

```
void swap(int* x, int* y) {
int n = *x;
                    int n = *(x+100);
          ??
    *y = x;
               int* z = malloc(2023);
}
```

C does not stop us from doing any of that…

… but separation logic does!

spec-guided search

$$\{ x \mapsto a * y \mapsto b \}$$

??

$$\{ x \mapsto b * y \mapsto a \}$$

spec-guided search

$\{ x \mapsto a * y \mapsto b \}$

??

$\{ x \mapsto b * y \mapsto a \}$

spec-guided search

```
let a1 = *x;
```

$\{ x \mapsto a1 * y \mapsto b \}$

??

$\{ x \mapsto b * y \mapsto a1 \}$

spec-guided search

```
let a1 = *x;
```

$\{ x \mapsto a1 * y \mapsto b \}$

??

$\{ x \mapsto b * y \mapsto a1 \}$

spec-guided search

```
let a1 = *x;
let b1 = *y;
```

{ x ↦ a1 * y ↦ b1 }

??

{ x ↦ b1 * y ↦ a1 }

spec-guided search

```
let a1 = *x;
let b1 = *y;
```

{ x ↦ a1 * y ↦ b1 }

??

{ x ↦ b1 * y ↦ a1 }

spec-guided search

```
let a1 = *x;
let b1 = *y;
*x = b1;
```

{ x ↦ b1 * y ↦ b1 }

??

{ x ↦ b1 * y ↦ a1 }

spec-guided search

```
let a1 = *x;
let b1 = *y;
*x = b1;
```

{ x ↦ b1 * y ↦ b1 }

??

{ x ↦ b1 * y ↦ a1 }

spec-guided search

```
let a1 = *x;
let b1 = *y;
*x = b1;
*y = a1;
```

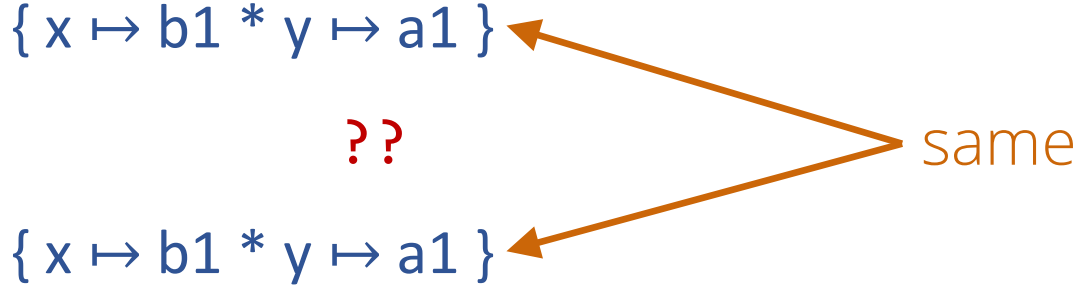
{ x ↦ b1 * y ↦ a1 }

??

{ x ↦ b1 * y ↦ a1 }

spec-guided search

```
let a1 = *x;
let b1 = *y;
*x = b1;
*y = a1;
```



spec-guided search

```
let a1 = *x;
let b1 = *y;
*x = b1;
*y = a1;
```

spec-guided search

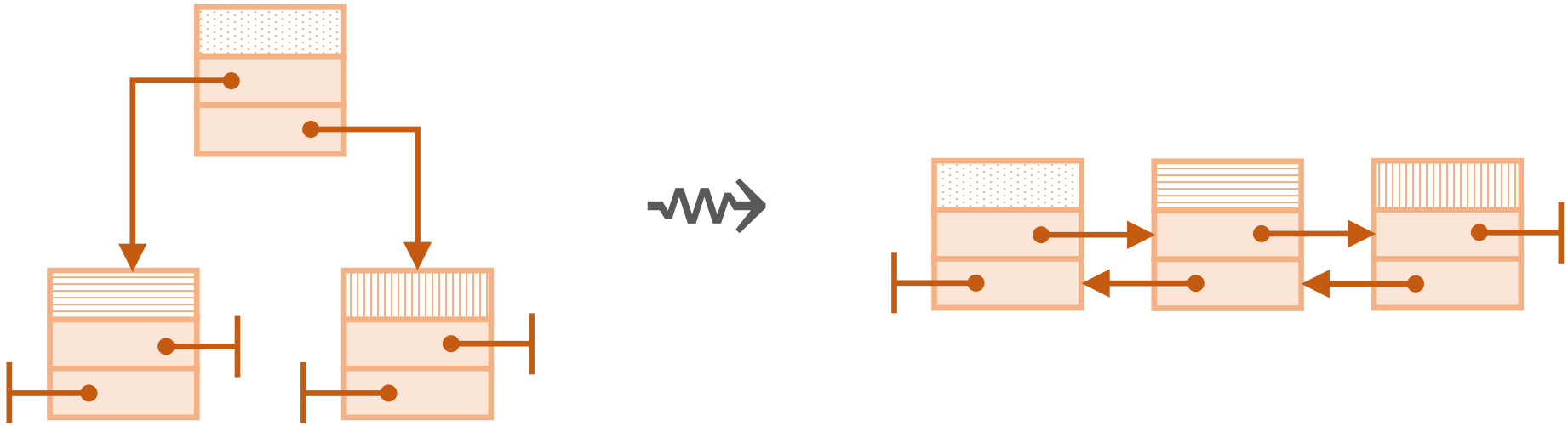
```
void swap(int* x, int* y) {
        int a1 = *x;
        int b1 = *y;
        *x = b1;
        *y = a1;
}
```

deductive synthesis

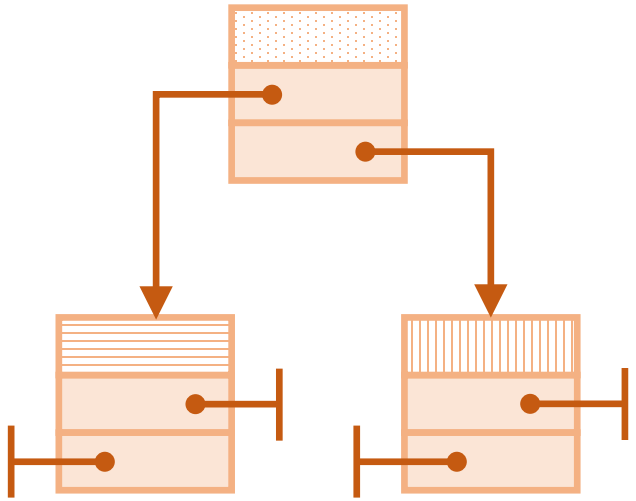
big idea:
look for the proof the find the program

- space of proofs is more restricted!
- bonus: provably correct result!

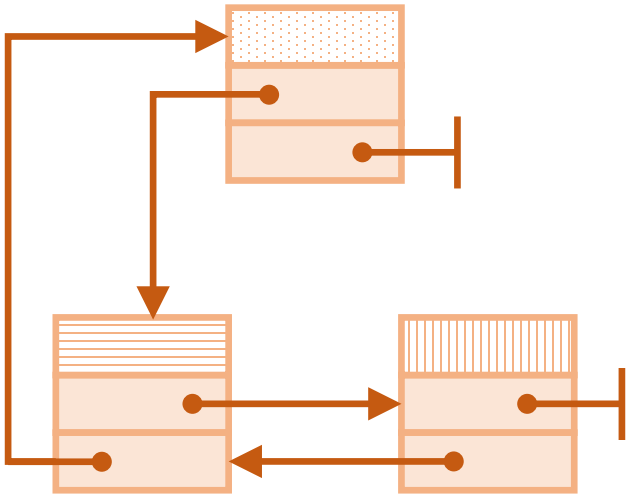
demo: flatten a tree into a list



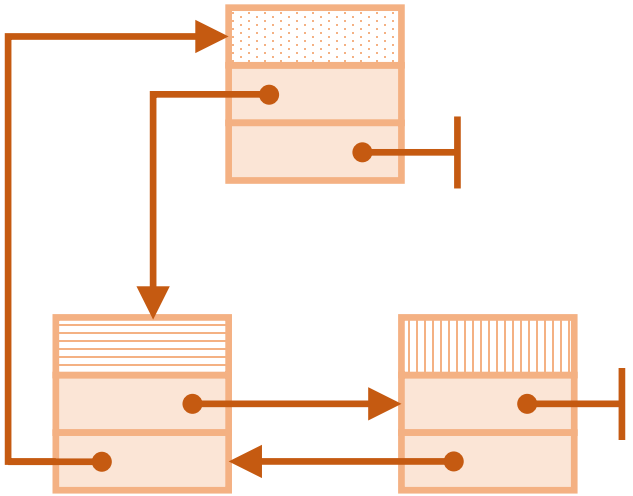
demo: flatten a tree into a list (in place)



demo: flatten a tree into a list (in place)



demo: flatten a tree into a list (in place)



specification

references

Etienne Kneuss, Viktor Kuncak, Ivan Kuraj, Philippe Suter
Synthesis modulo recursive functions
OOPSLA 2013

Nadia Polikarpova, Ilya Sergey
Structuring the Synthesis of Heap-Manipulating Programs
POPL 2019

Peter-Michael Osera, Steve Zdancewic
Type-and-example-directed program synthesis
PLDI 2015

1. observational equivalence
2. CEGIS
3. deductive synthesis
4. learn while searching

challenge: too many programs

how do humans do it?

1. learn from others

how do humans do it?

1. learn fr[illegible]ers

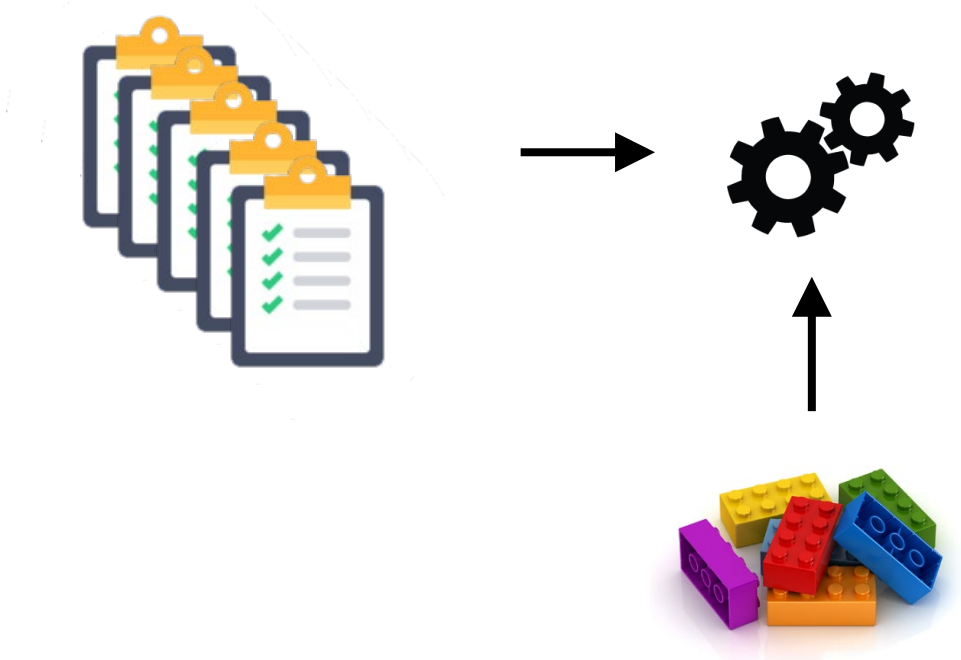
requires data

2. learn by doing!

library learning

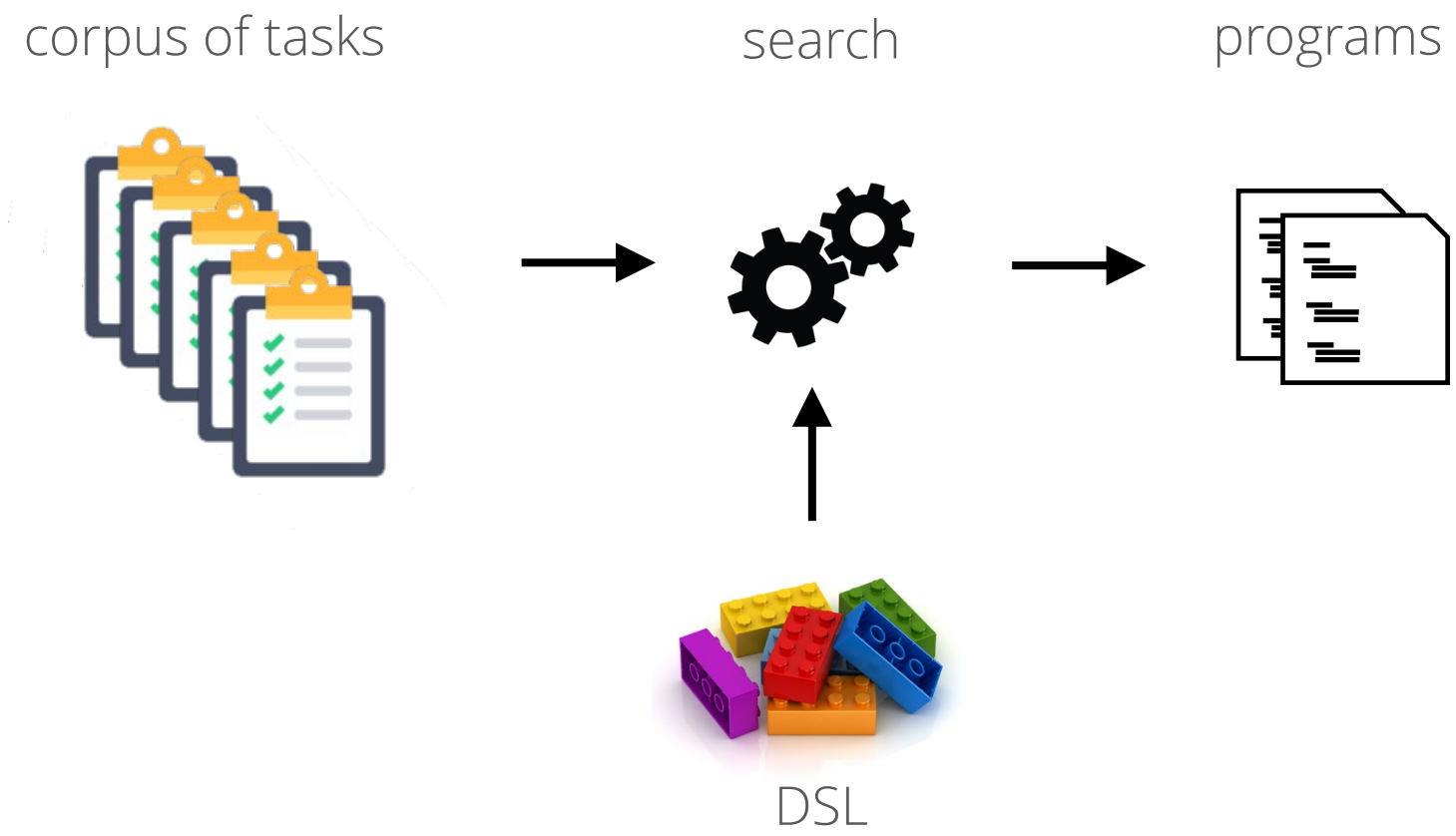
corpus of tasks

search

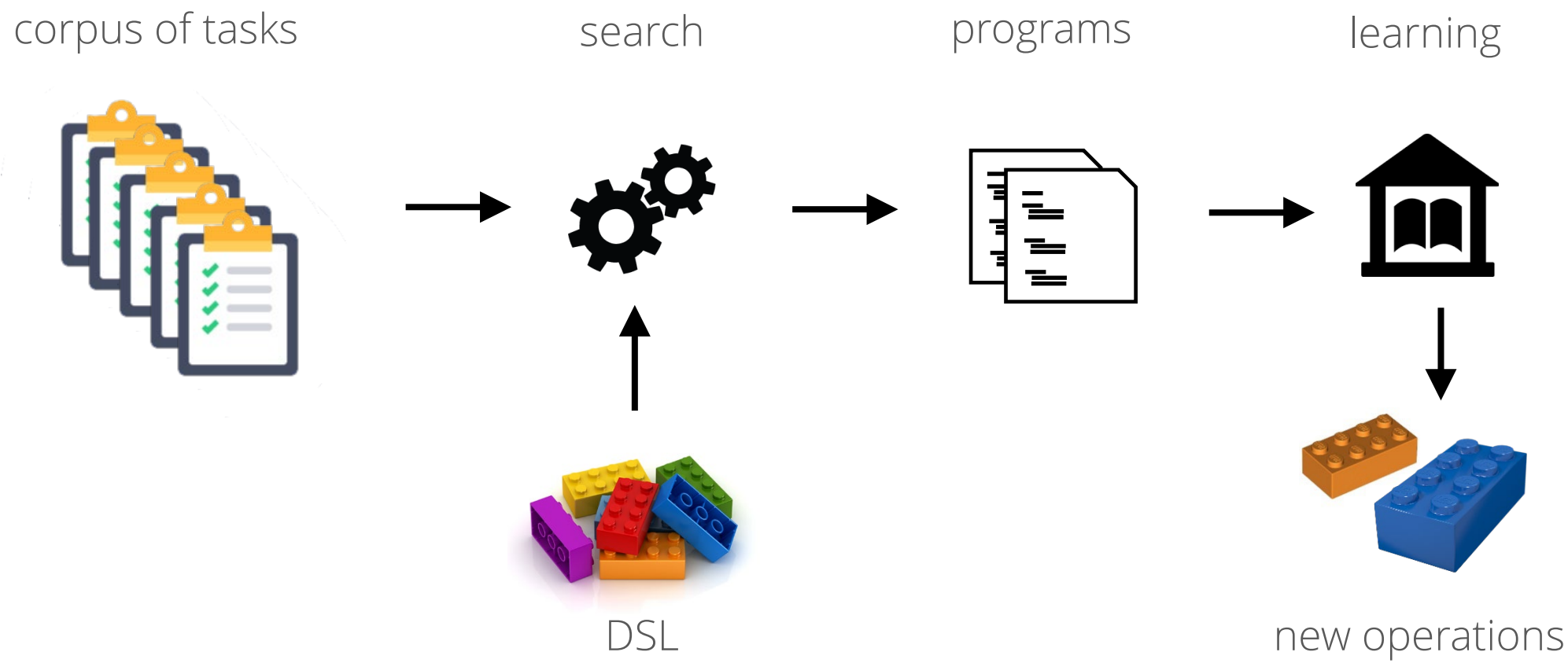


DSL

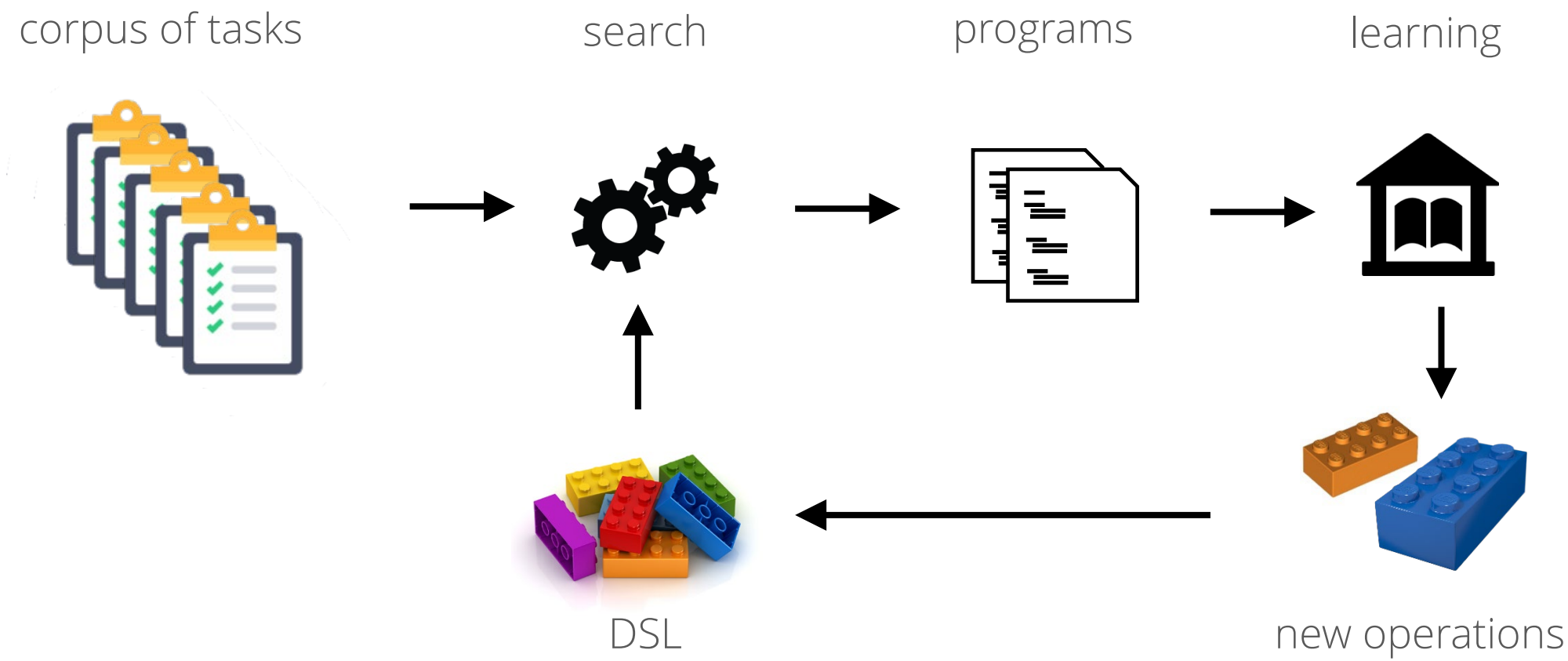
library learning



library learning



library learning



example: list manipulations

corpus of tasks

```
positives
   [0] → []
 [4 0 8] → [4 8]

evens
   [3] → []
 [7 6 3 2] → [6 2]

…

sort
   [9 2 7 1] → [1 2 7 9]
 [3 8 9 4 2] → [2 3 4 8 9]
[6 2 2 3 8 5] → [2 2 3 5 6 8]
```

example: list manipulations

corpus of tasks

```
positives
   [0] → []
 [4 0 8] → [4 8]

evens
   [3] → []
 [7 6 3 2] → [6 2]

…

sort
   [9 2 7 1] → [1 2 7 9]
 [3 8 9 4 2] → [2 3 4 8 9]
[6 2 2 3 8 5] → [2 2 3 5 6 8]
```

```
map fold
if cons >
length range
```

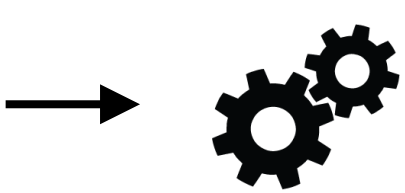
DSL

example: list manipulations

corpus of tasks

```
positives
   [0] → []
 [4 0 8] → [4 8]

evens
   [3] → []
 [7 6 3 2] → [6 2]

…

sort
   [9 2 7 1] → [1 2 7 9]
 [3 8 9 4 2] → [2 3 4 8 9]
[6 2 2 3 8 5] → [2 2 3 5 6 8]
```

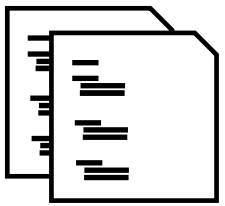
search



```
map fold
if cons >
length range
```

DSL

programs



example: list manipulations

corpus of tasks

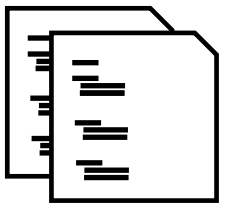
```
positives
[0] → []
[4 0 8] → [4 8]

evens
[3] → []
[7 6 3 2] → [6 2]

…

sort
[9 2 7 1] → [1 2 7 9]
[3 8 9 4 2] → [2 3 4 8 9]
[6 2 2 3 8 5] → [2 2 3 5 6 8]
```

search

```
map fold
if cons >
length range
```

DSL

programs



learning

filter

new operations

example: list manipulations

corpus of tasks

positives

```
[0] → []
[4 0 8] → [4 8]
```

evens

```
[3] → []
[7 6 3 2] → [6 2]
```

…

sort

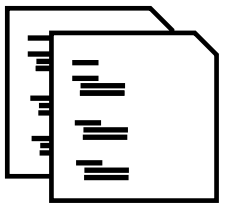
```
[9 2 7 1] → [1 2 7 9]
[3 8 9 4 2] → [2 3 4 8 9]
[6 2 2 3 8 5] → [2 2 3 5 6 8]
```

search

```
map fold
if cons >
length range
```

DSL

programs

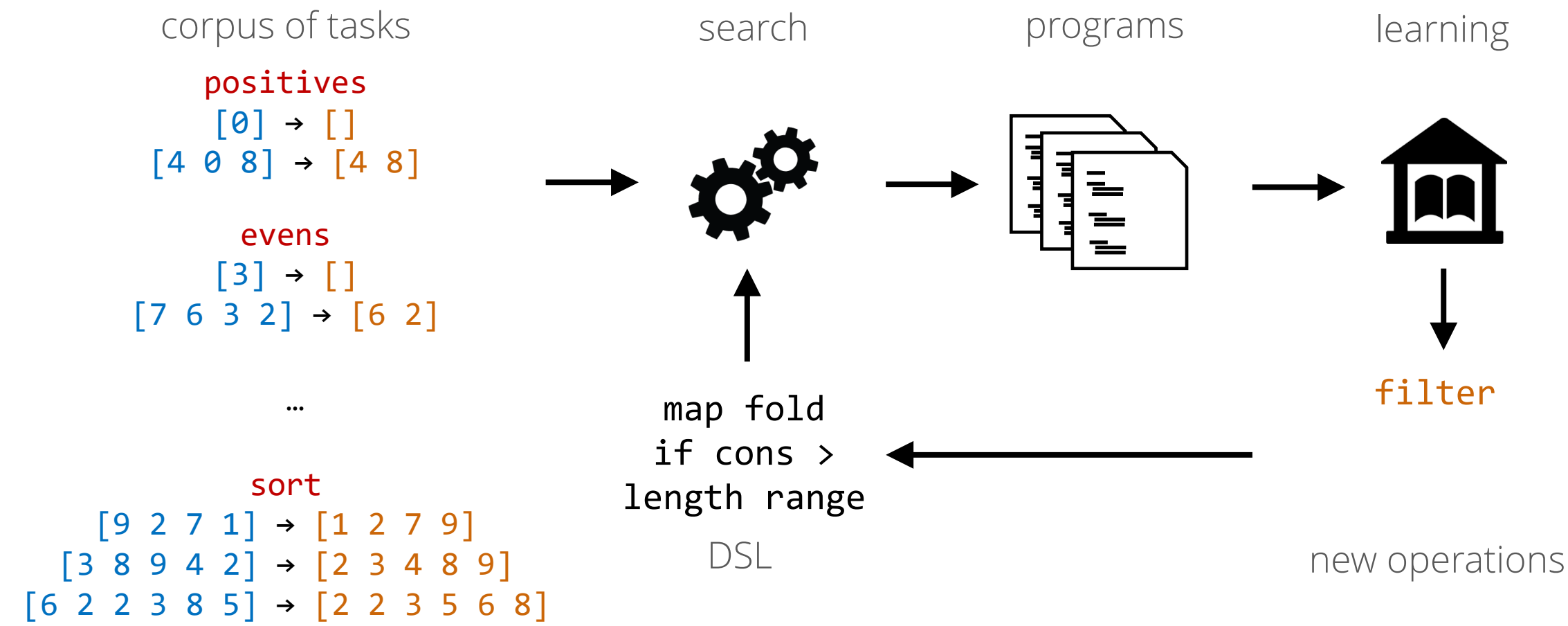


learning

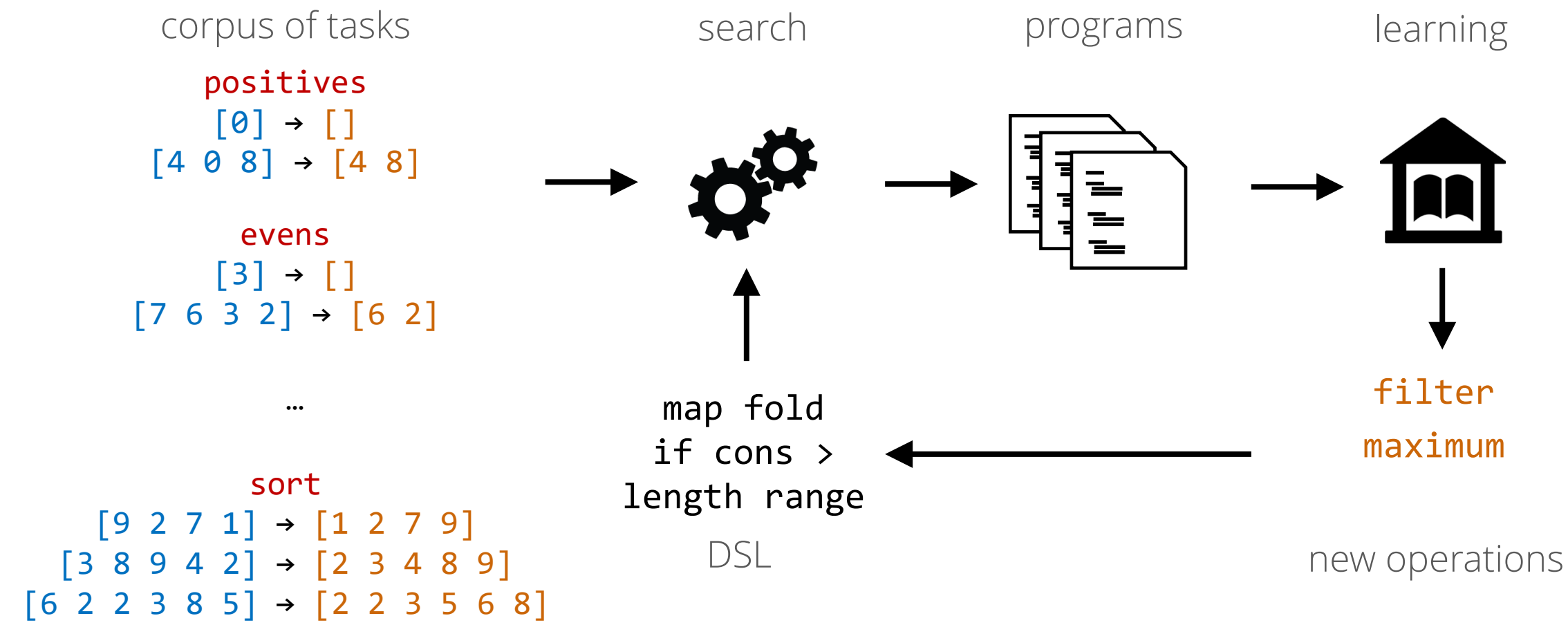
filter

new operations

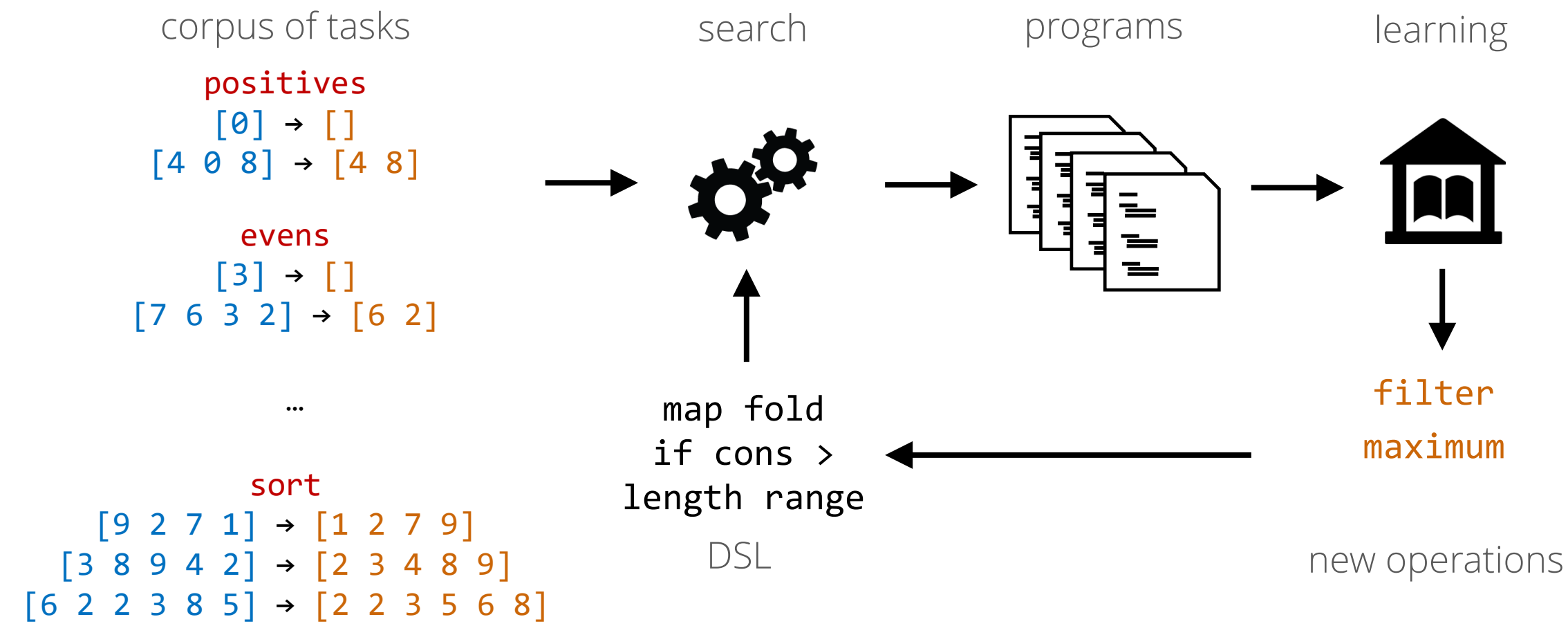
example: list manipulations



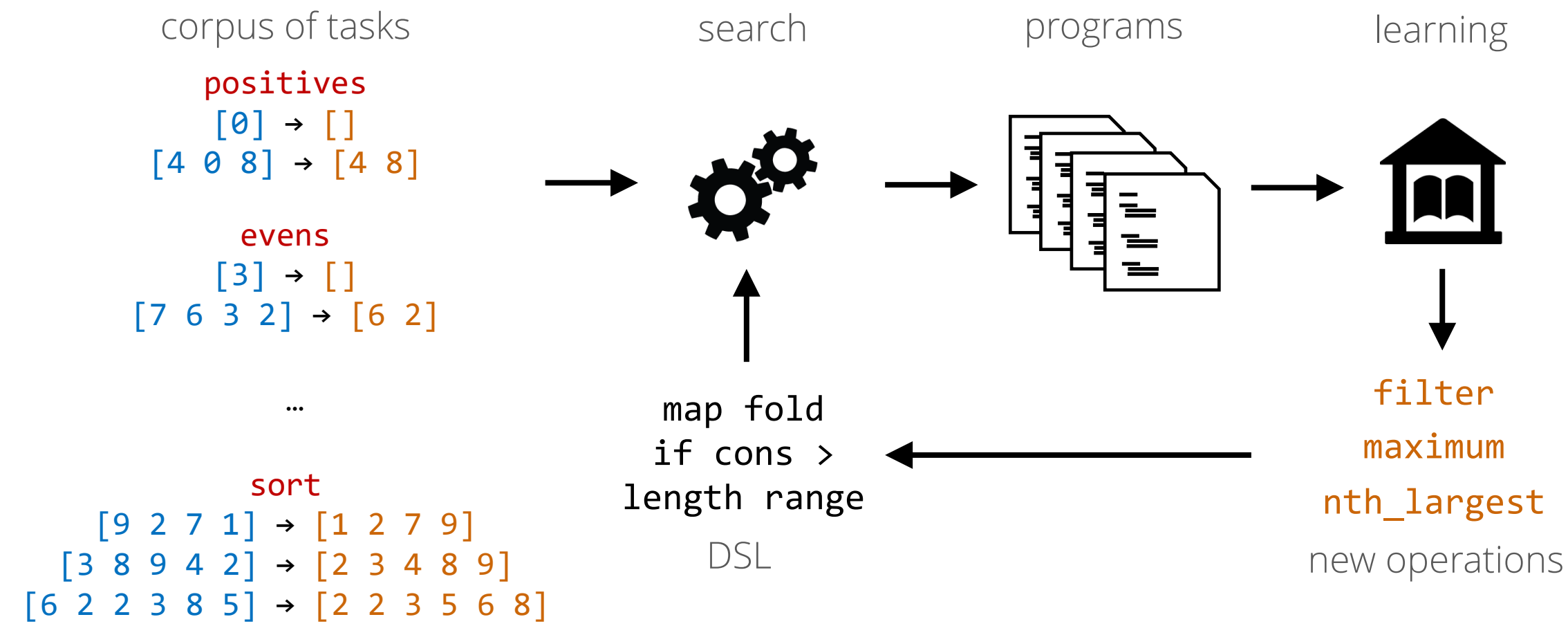
example: list manipulations



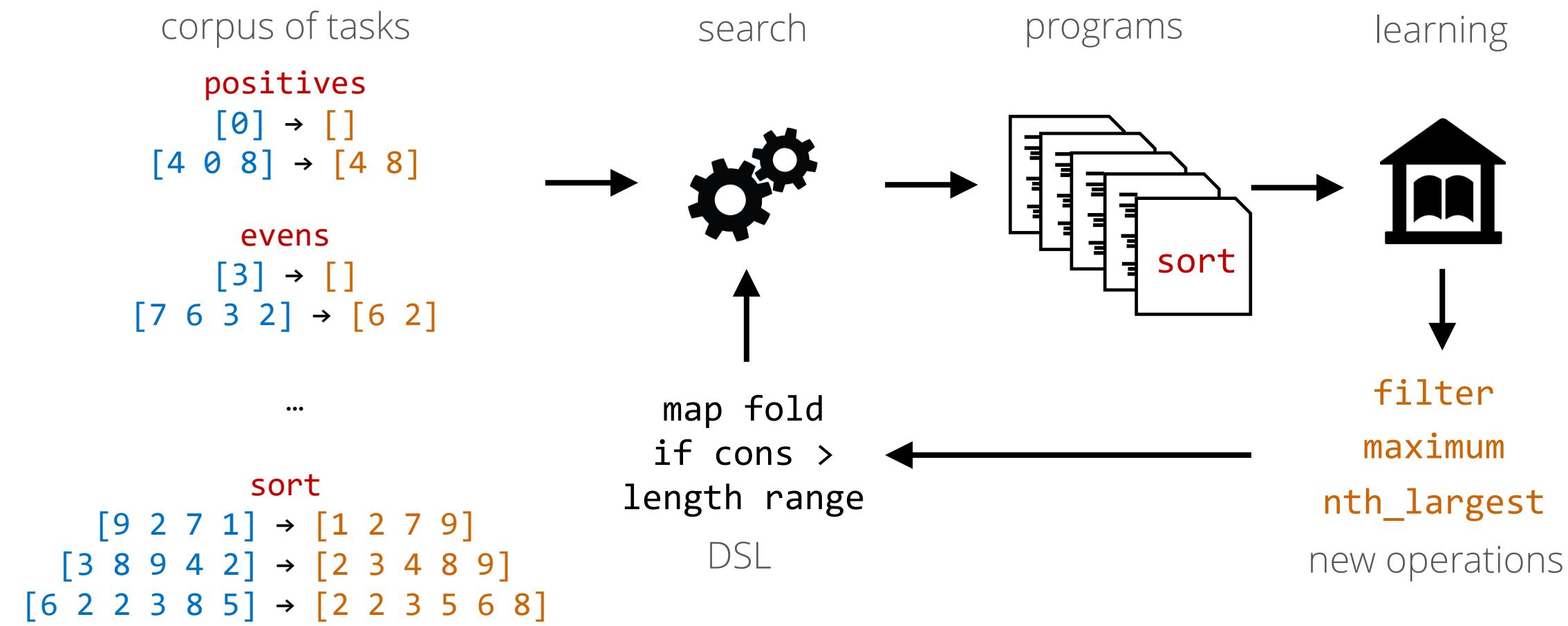
example: list manipulations



example: list manipulations



example: list manipulations



example: list manipulations

sort
in learned DSL

```
\xs → map
        (\n → nth_largest xs (n + 1))
        (range (length xs))
```

sort
in original DSL

```
\xs -> map
       (\n -> car (fold (fold xs nil (\z u -> if n +
         1 > length (fold xs nil (\v w -> if z > v
         then cons v w else w)) then cons z u else
         u)) nil (\a b -> if nil? (fold (fold xs nil
         (\c d -> if n + 1 > length (fold xs nil (\e
         f -> if c > e then cons e f else f)) then
         cons c d else d)) nil (\g h -> if g > a then
         cons g h else h)) then cons a b else b)))
      (range (length xs))
```

library learning

big idea:
learn useful abstractions by solving similar tasks

references

Kevin Ellis et al.
DreamCoder: Bootstrapping Inductive Program Synthesis with Wake-Sleep Library Learning
PLDI'21

Kensen Shi, Jacob Steinhardt, Percy Liang
FrAngel: Component-Based Synthesis with Control Structures
POPL 2019

Shraddha Barke, Hila Peleg, Nadia Polikarpova
Just-in-Time Learning for Bottom-Up Enumerative Synthesis
OOPSLA 2020

synthesis II @POPL (thu 10:20)

two new library learning papers!

Matt Bowers, Theo X. Olausson, Lionel Wong, Gabriel Grand, Joshua B. Tenenbaum, Kevin Ellis, Armando Solar-Lezama
Top-Down Synthesis for Library Learning

David Cao, Rose Kunkel, Chandrakana Nandi, Max Willsey, Zachary Tatlock, Nadia Polikarpova
babble: Learning Better Abstractions with E-Graphs and Anti-unification

big ideas in program synthesis

1. observational equivalence
2. CEGIS
3. deductive synthesis
4. learn while searching