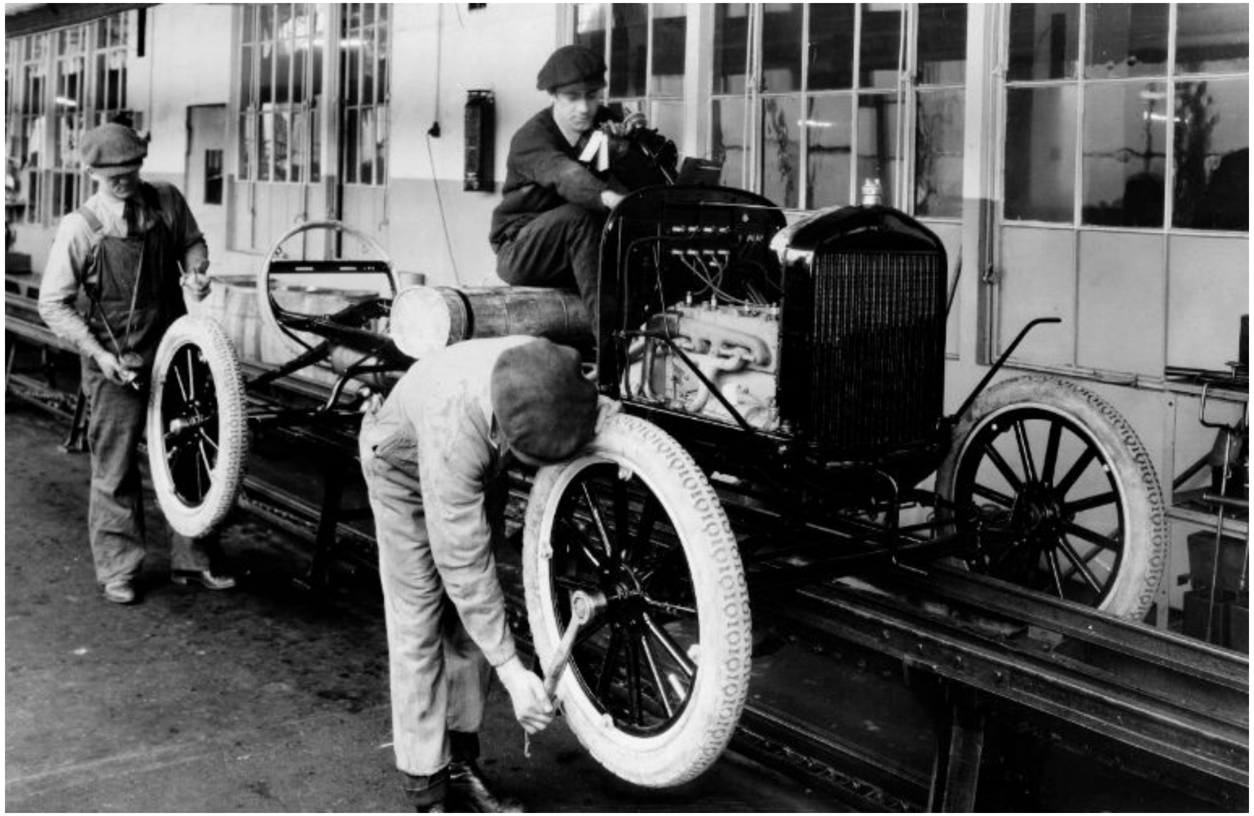
Program Synthesis

Ilya Sergey

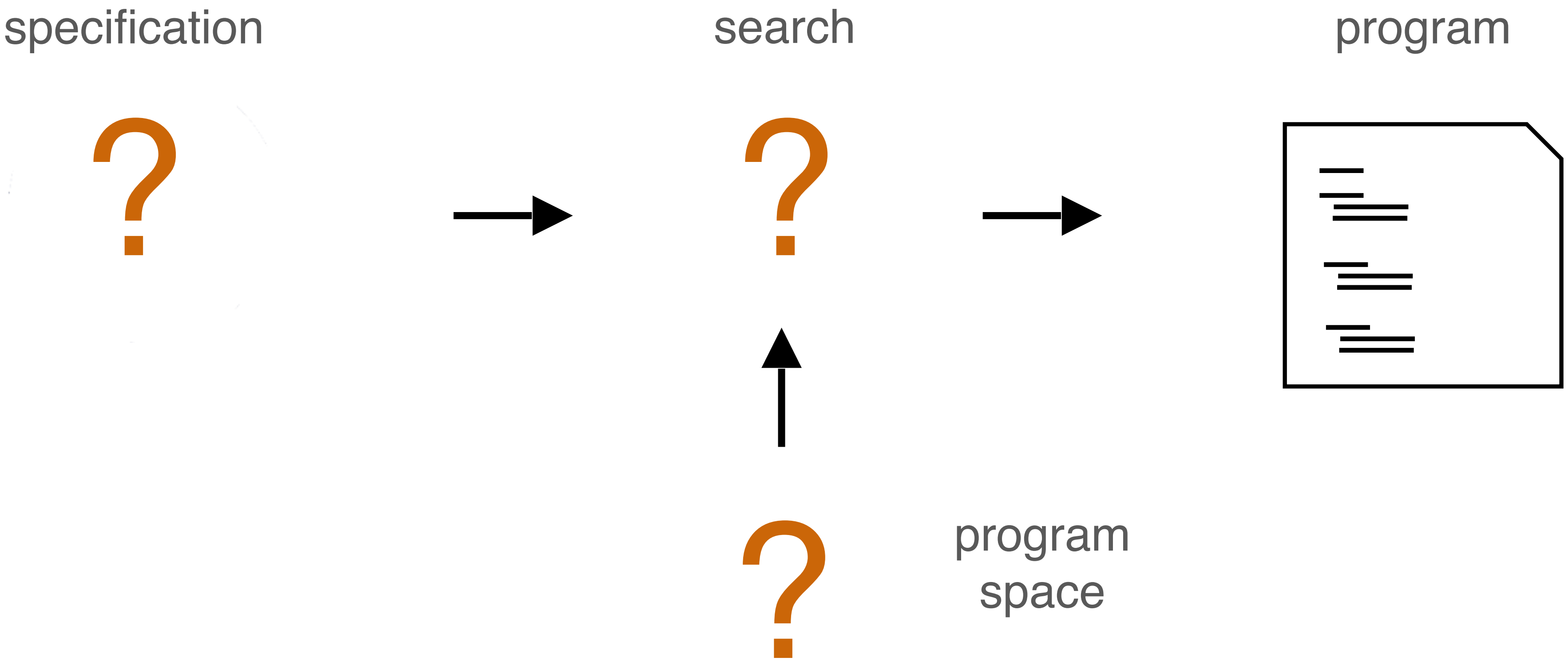
ilyasergey.net

NUS
National University
of Singapore

goal: automate programming



program synthesis



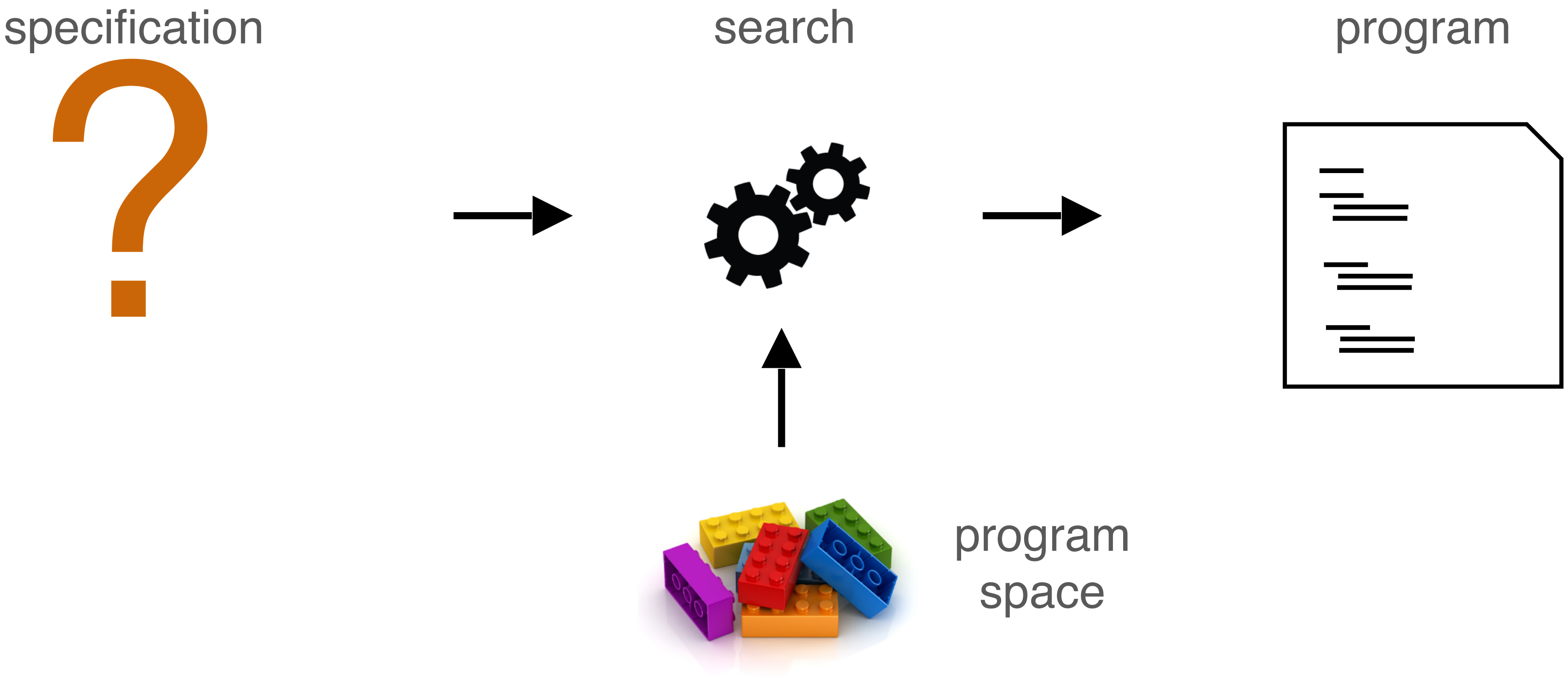
specification

search

program

program space

program synthesis



specification

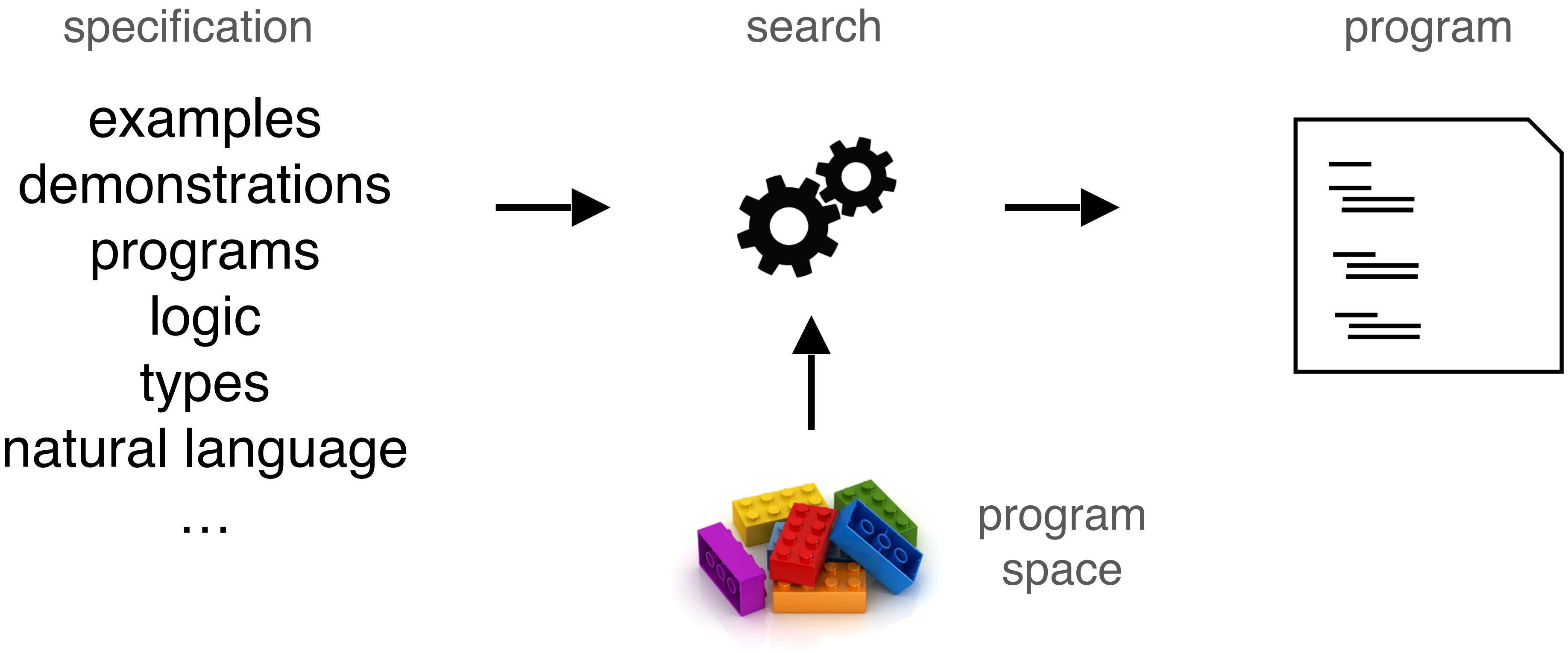
?

search

program

program space

program synthesis



specification

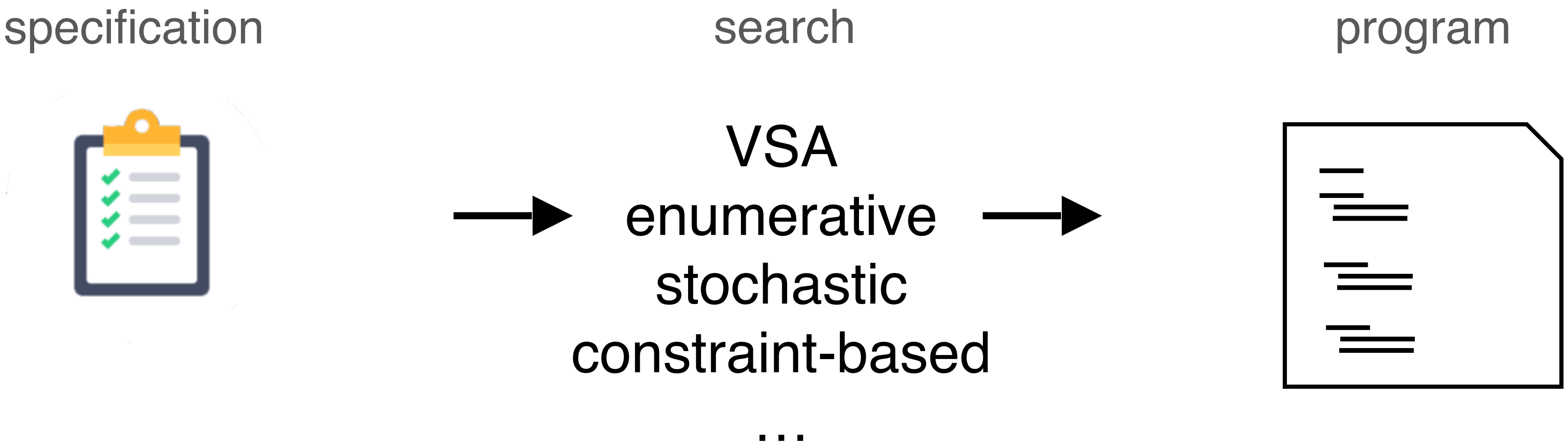
examples
demonstrations
programs
logic
types
natural language
…

search

program

program
space

program synthesis



specification

search

program

VSA
enumerative
stochastic
constraint-based
…

this talk

specifications

search strategies

what makes a good spec?

specification

1. human-friendly

   easier to write than the program

2. informative

   minimal ambiguity

3. synthesizer-friendly

   easy to check, guides the search

this talk

specifications

1. examples (PBE)
2. programs
3. natural language
4. types and logic

search strategies

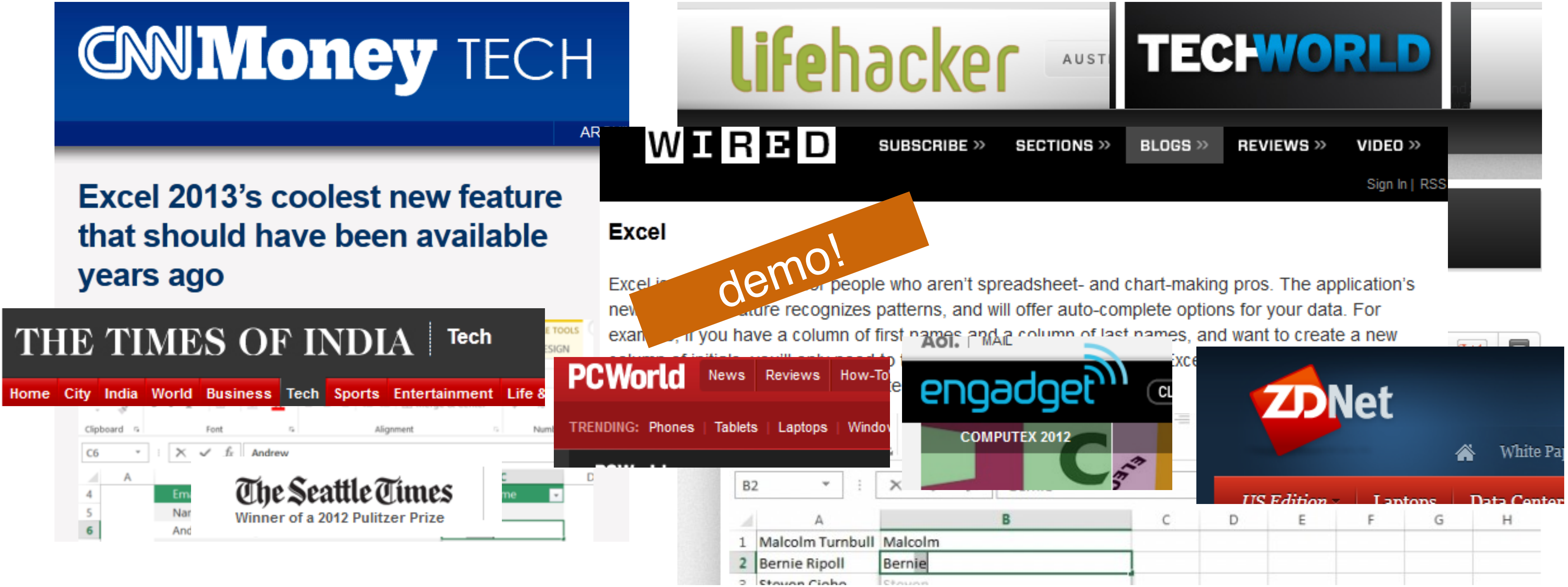
this talk

specifications

1. examples (PBE)
2. programs
3. natural language
4. types and logic

FlashFill

[Gulwani 2011]



demo!

Excel 2013's coolest new feature that should have been available years ago

Excel

Excel i… people who aren't spreadsheet- and chart-making pros. The application's new …ature recognizes patterns, and will offer auto-complete options for your data. For exam…, if you have a column of first names and a column of last names, and want to create a new

B2

	A	B	C	D	E	F	G	H
1	Malcolm Turnbull	Malcolm						
2	Bernie Ripoll	Bernie						
3	Steven Ciobo	Steven						

pbe / pbd: discussion

Domains:

- text transformations: FlashFill, FlashExtract
- web scraping: WebRelate, Rousillon
- data science: Morpheus, Wrex
- programmer’s assistant: FrAngel, Snippy

+ beginner-friendly
+ easy to check correctness

- ambiguous
- hard to write for complex programs / data structures
- cannot express non-functional properties

this talk

specifications

1. examples (PBE)
2. programs
3. natural language
4. types and logic

Sketch

[Solar-Lezama et al, 2005]

Problem: isolate the least significant zero bit in a word

0010 0101 → 0000 0010

Sketch

Problem: isolate the least significant zero bit in a word

Easy to implement with a loop

```
bit[32] isolate0 (bit[32] x) {
    bit[32] ret = 0;
    for (int i = 0; i < 32; i++)
        if (!x[i]) { ret[i] = 1; return ret; }
}
```

Can this be done more efficiently with bit manipulation?

- Trick: adding 1 to a string of ones turns the next zero to a 1
- i.e. 000111 + 1 = 001000

Sketch: synthesis goal

```
bit[32] isolate0fast (bit[32] x) implements isolate0 {
     return expr(x, 3);
}

// Sketch for bit-vector expressions with
// +, &, xor and bitwise negation (~)
generator bit[32] expr(bit[32] x, int depth){
    assert depth > 0;
    if(??) return x;
    if(??) return ??;
    if(??) return ~expr(x, depth-1);
    if(??){
        return {| expr(x, depth-1) (+ | & | ^) expr(x, depth-1) |};
    }
}
```

Sketch: output

```
bit[W] isolate0fast (bit[W] x) {
  return (~x) & (x + 1);
}
```

reference programs: discussion

Domains:

- superoptimization: Stoke, Lens
- verified lifting: QBS, STNG, all from Alvin Cheung's work

+ programmer-friendly
+ precise

- simple program does not always exist
- hard to check correctness

this talk

specifications

1. examples (PBE)
2. programs
3. natural language
4. types and logic

Regel

[Chen et al, 2020]

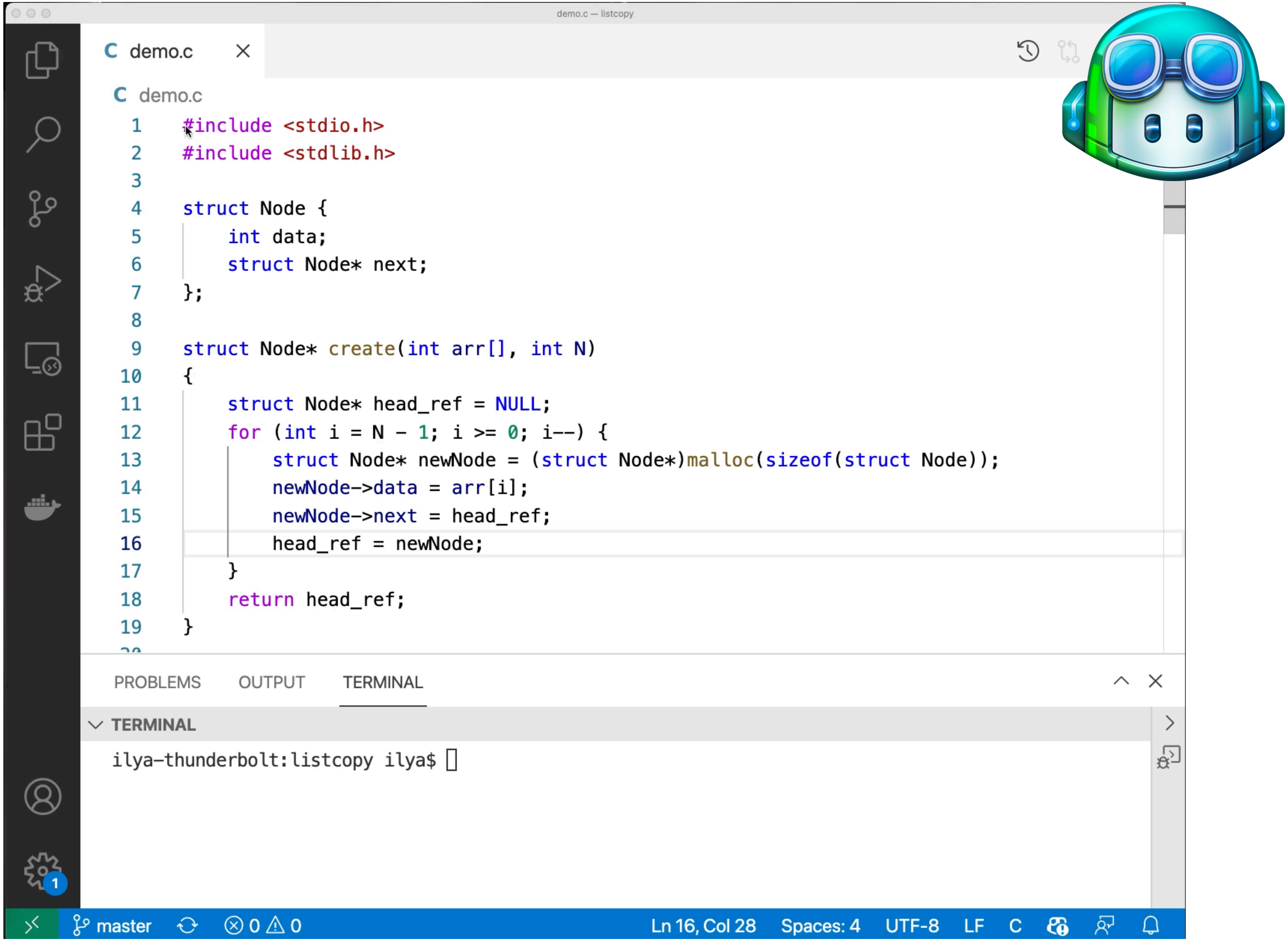
“I need a regular expression that validates `Decimal(18, 3)`, which means the max number of digits before comma is 15 then accept at max 3 numbers after the comma.”

```
Concat(
  RepeatRange(<num>,1,15),
  Optional(Concat(
            <.>,

RepeatRange(<num>,1,3))
```

Positive examples:	Negative examples:
123456789.123	1234567891234567
12345.1	123.1234

AI-Powered Program Synthesis at Scale

demo!



```c
#include <stdio.h>
#include <stdlib.h>

struct Node {
    int data;
    struct Node* next;
};

struct Node* create(int arr[], int N)
{
    struct Node* head_ref = NULL;
    for (int i = N - 1; i >= 0; i--) {
        struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
        newNode->data = arr[i];
        newNode->next = head_ref;
        head_ref = newNode;
    }
    return head_ref;
}
```

```
ilya-thunderbolt:listcopy ilya$ 
```

natural language: discussion

Domains:

- queries (SQL, regex): SQLizer, Regel
- programmer’s assistant: SWIM, GPT-3?
- GitHub CoPilot

+ beginner-friendly
+ expressive

- ambiguous
- verbose

this talk

specifications

1. examples (PBE)
2. programs
3. natural language
4. types and logic

Synquid

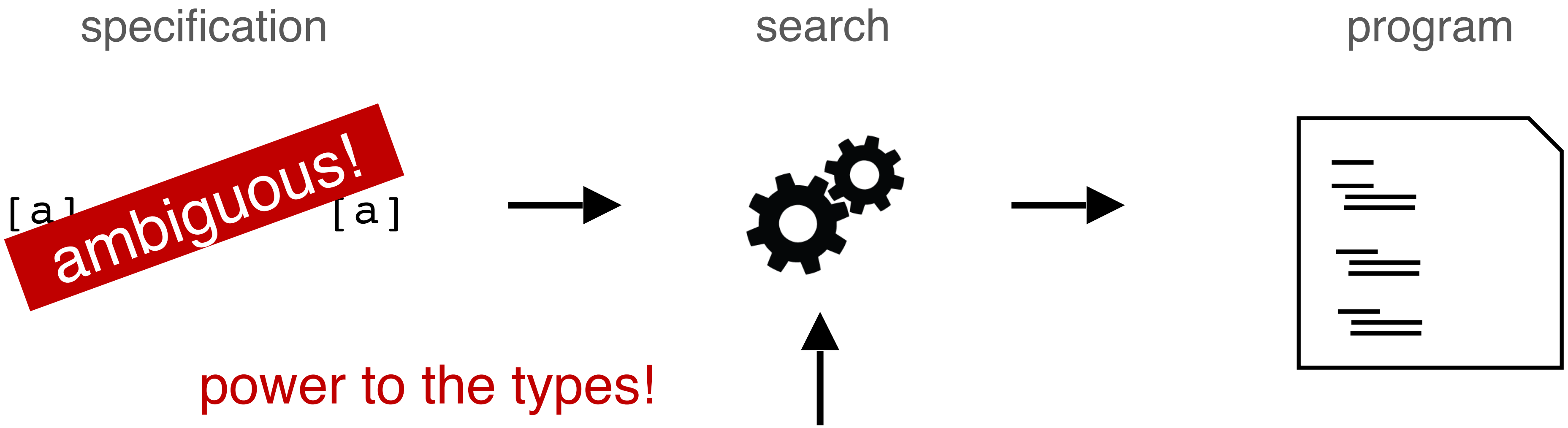
[Polikarpova et al, 2016]

Problem: replicate an element N times

using recursion

3 → 6 → [3, 3, 3, 3, 3, 3]

type-driven synthesis



specification

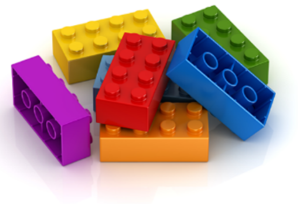
search

program

[a] [a]

ambiguous!

power to the types!



program space

refinement types

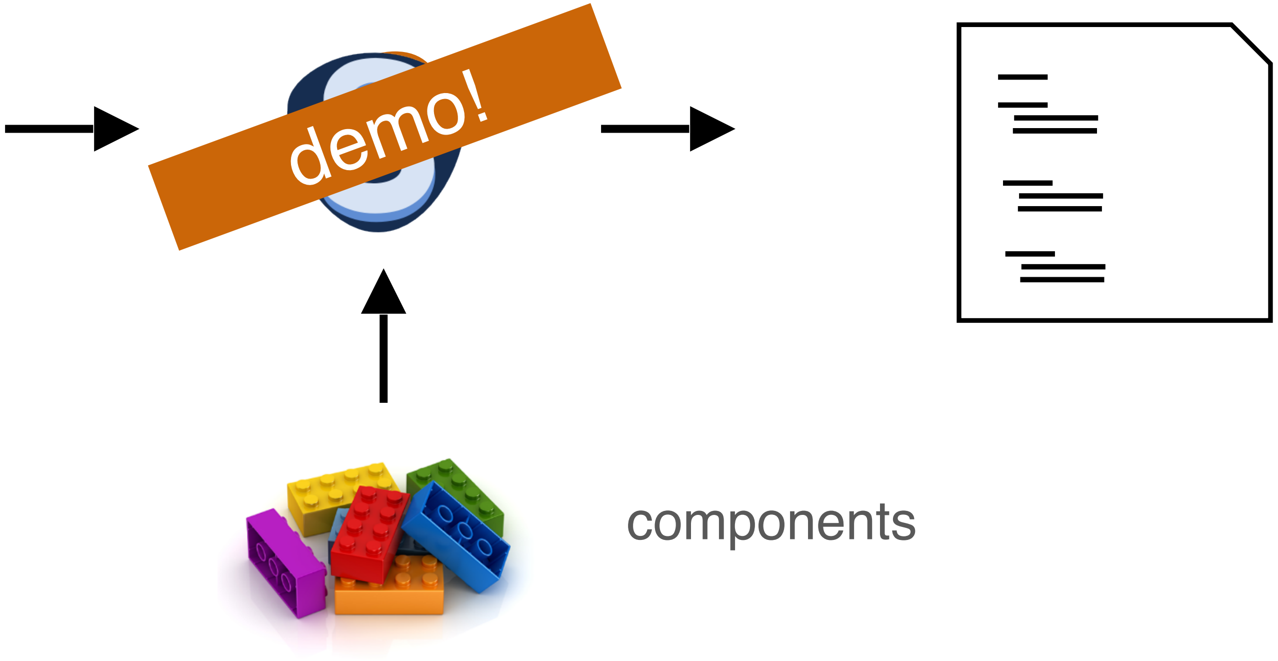
```
Int
```

Synquid

specification

program

refinement types



demo!

components

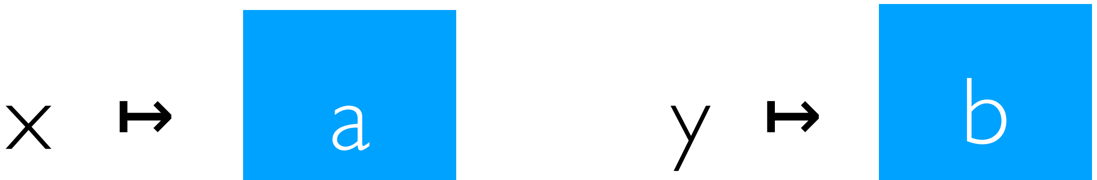
http://comcom.csail.mit.edu/comcom/#Synquid

What about imperative programs?

Program Synthesis from Logical Specifications

Let’s *swap* values of two *distinct* pointers

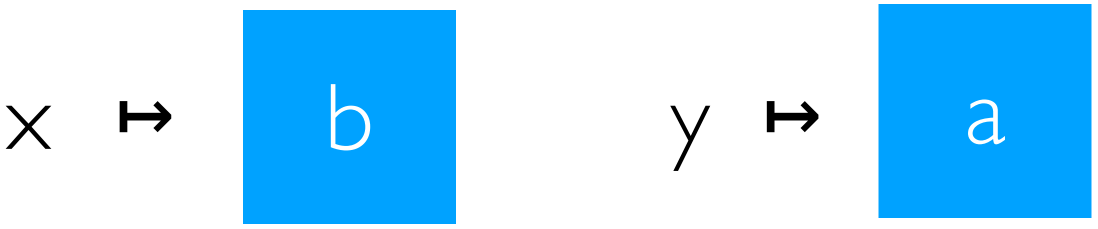
Let’s *swap* values of two *distinct* pointers



x ↦ a

y ↦ b

Let’s *swap* values of two *distinct* pointers



x ↦ b y ↦ a

swap

```
void swap(loc x, loc y)
```

$\{ \ \mathsf{x} \mapsto a \land \mathsf{y} \mapsto b \ \}$

```
void swap(loc x, loc y)
```

$$\{\ \mathsf{x} \mapsto a \land \mathsf{y} \mapsto b\ \}$$

```
void swap(loc x, loc y)
```

$$\{\ \mathsf{x} \mapsto b \land \mathsf{y} \mapsto a\ \}$$

“separately”

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

```
void swap(loc x, loc y)
```

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

Peter W. O'Hearn, John C. Reynolds, Hongseok Yang:
Local Reasoning about Programs that Alter Data Structures. CSL 2001

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

$$\texttt{void swap(loc}\ \boxed{\texttt{x}}\texttt{, loc}\ \boxed{\texttt{y}}\texttt{)}$$

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

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

```
void swap(loc x, loc y)
```

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

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

??

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

```
let a2 = *x;
```

$$\{\ x \mapsto a2 * y \mapsto \boxed{b}\ \}$$

??

$$\{\ x \mapsto \boxed{b} * y \mapsto a2\ \}$$

```
let a2 = *x;
let b2 = *y;
```

$$\{ x \mapsto a2 * y \mapsto b2 \}$$

??

$$\{ x \mapsto b2 * y \mapsto a2 \}$$

```
let a2 = *x;
let b2 = *y;
*x = b2;
```

$$\{ x \mapsto b2 \ast y \mapsto b2 \}$$

??

$$\{ x \mapsto b2 \ast y \mapsto a2 \}$$

```
let a2 = *x;
let b2 = *y;
*x = b2;
*y = a2;
```

$$\{\ x \mapsto b2 * y \mapsto a2\ \}$$

??

$$\{\ x \mapsto b2 * y \mapsto a2\ \}$$

```
let a2 = *x;
let b2 = *y;
*x = b2;
*y = a2;
```

$$\{\ x \mapsto b2 \ast y \mapsto a2\ \}$$

??

$$\{\ x \mapsto b2 \ast y \mapsto a2\ \}$$

$$x \mapsto b2 \ast y \mapsto a2 \ \vdash\ x \mapsto b2 \ast y \mapsto a2$$

```
let a2 = *x;
let b2 = *y;
*x = b2;
*y = a2;
```

$$\{ x \mapsto b2 \ast y \mapsto a2 \}$$

??

$$\{ x \mapsto b2 \ast y \mapsto a2 \}$$

$x \mapsto b2 \ast y \mapsto a2 \vdash x \mapsto b2 \ast y \mapsto a2$ ✓

```
void swap(loc x, loc y) {
       let a2 = *x;
       let b2 = *y;
       *x = b2;
       *y = a2;
}
```

Deductive Program Synthesis: An Overview

precondition *postcondition*

$$\{P\} \rightsquigarrow \{Q\} \mid ?$$

unknown implementation

Goal

Find a program that transforms P into Q

Method

Enumerative *proof search* on inference rules of a program logic

Result

A program that is ***correct by construction***

Synthetic Separation Logic

$$\Gamma\ ;\ P \rightsquigarrow Q \mid \mathbf{c}$$

$$\Gamma \; ; \; P \rightsquigarrow Q \mid \mathbf{c}$$

There *exists* a program **c**, using variables from Γ, such that

any initial state satisfying **P**,

c, after it terminates,

will transform to a state satisfying **Q**.

$$\Gamma; \{\mathsf{emp}\} \rightsquigarrow \{\mathsf{emp}\} \mid ??$$

$$\Gamma; \{\mathsf{emp}\} \rightsquigarrow \{\mathsf{emp}\} \mid \mathtt{\textbf{skip}} \qquad (\mathrm{Emp})$$

$$a \in \mathrm{GV}(\Gamma, \mathrm{P}, \mathrm{Q})$$

$$\Gamma; \{\, x \mapsto a * \mathrm{P} \,\} \rightsquigarrow \{\, \mathrm{Q} \,\} \mid ??$$

$$\frac{a \in \mathrm{GV}(\Gamma, P, Q) \qquad y \text{ is fresh} \qquad \Gamma, y\, ;\, [y/a]\{\, x \mapsto y * P \,\} \rightsquigarrow [y/a]\{\, Q \,\} \mid \mathtt{c}}{\Gamma;\, \{\, x \mapsto a * P \,\} \rightsquigarrow \{\, Q \,\} \mid \mathtt{let\ y\ =\ *x;\ c}} \text{(Read)}$$

$$\Gamma; \{ x \mapsto - * P \} \rightsquigarrow \{ x \mapsto e * Q \} \mid ??$$

$$\dfrac{\begin{array}{c} \mathit{Vars}(\mathsf{e}) \subseteq \Gamma \\ \Gamma \,;\{\, \mathsf{x} \mapsto \mathsf{e} * \mathsf{P} \,\} \rightsquigarrow \{\, \mathsf{x} \mapsto \mathsf{e} * \mathsf{Q} \,\} \mid \mathtt{c} \end{array}}{\Gamma;\{\, \mathsf{x} \mapsto - * \mathsf{P} \,\} \rightsquigarrow \{\, \mathsf{x} \mapsto \mathsf{e} * \mathsf{Q} \,\} \mid \mathtt{*x\ =\ e;\ c}} \;(\text{Write})$$

$$\Gamma; \{\, P * R \,\} \rightsquigarrow \{\, Q * R \,\} \mid ??$$

$$\frac{\begin{array}{c}\mathrm{EV}(\Gamma, P, Q) \cap \mathit{Vars}(R) = \varnothing \\ \Gamma \,;\, \{\, P \,\} \rightsquigarrow \{\, Q \,\} \mid \mathbf{c}\end{array}}{\Gamma ; \{\, P * R \,\} \rightsquigarrow \{\, Q * R \,\} \mid \mathbf{c}}\ (\mathrm{Frame})$$

$$\Gamma; \{\mathrm{emp}\} \rightsquigarrow \{\mathrm{emp}\} \mid \texttt{skip} \qquad \text{(Emp)}$$

$$\frac{\begin{array}{c} a \in GV(\Gamma, P, Q) \qquad y \text{ is fresh} \\ \Gamma, y\, ; [y/a]\{\, x \mapsto y * P \,\} \rightsquigarrow [y/a]\{\, Q \,\} \mid \texttt{c} \end{array}}{\Gamma; \{\, x \mapsto a * P \,\} \rightsquigarrow \{\, Q \,\} \mid \texttt{let y = *x; c}} \;\text{(Read)}$$

$$\frac{\begin{array}{c} EV(\Gamma, P, Q) \cap \mathit{Vars}(R) = \varnothing \\ \Gamma\, ; \{\, P \,\} \rightsquigarrow \{\, Q \,\} \mid \texttt{c} \end{array}}{\Gamma; \{\, P * R \,\} \rightsquigarrow \{\, Q * R \,\} \mid \texttt{c}} \;\text{(Frame)}$$

$$\frac{\begin{array}{c} \mathit{Vars}(e) \subseteq \Gamma \\ \Gamma\, ; \{\, x \mapsto e * P \,\} \rightsquigarrow \{\, x \mapsto e * Q \,\} \mid \texttt{c} \end{array}}{\Gamma; \{\, x \mapsto - * P \,\} \rightsquigarrow \{\, x \mapsto e * Q \,\} \mid \texttt{*x = e; c}} \;\text{(Write)}$$

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

```
void swap(loc x, loc y)
```

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

$$\{x, y\}\ ;\ \{x \mapsto a * y \mapsto b\} \rightsquigarrow \{x \mapsto b * y \mapsto a\} \quad \Big| \quad ??$$

$$\frac{\{\, x, y, a2 \,\}\,;\ \{\, x \mapsto a2 \ast y \mapsto b \,\} \rightsquigarrow \{\, x \mapsto b \ast y \mapsto a2 \,\} \mid \texttt{??}}{\{\, x, y \,\}\,;\ \{\, x \mapsto a \ast y \mapsto b \,\} \rightsquigarrow \{\, x \mapsto b \ast y \mapsto a \,\} \mid \texttt{let a2 = *x; ??}}\ (\text{Read})$$

$$\dfrac{\dfrac{\{\, \mathrm{x}, \mathrm{y}, \mathrm{a2}, \mathrm{b2} \,\} \,;\, \{\, \mathrm{x} \mapsto \mathrm{a2} * \mathrm{y} \mapsto \mathrm{b2} \,\} \rightsquigarrow \{\, \mathrm{x} \mapsto \mathrm{b2} * \mathrm{y} \mapsto \mathrm{a2} \,\} \mid \texttt{??}}{\{\, \mathrm{x}, \mathrm{y}, \mathrm{a2} \,\} \,;\, \{\, \mathrm{x} \mapsto \mathrm{a2} * \mathrm{y} \mapsto b \,\} \rightsquigarrow \{\, \mathrm{x} \mapsto b * \mathrm{y} \mapsto \mathrm{a2} \,\} \mid \texttt{let b2 = *y; ??}}\;\text{(Read)}}{\{\, \mathrm{x}, \mathrm{y} \,\} \,;\, \{\, \mathrm{x} \mapsto a * \mathrm{y} \mapsto b \,\} \rightsquigarrow \{\, \mathrm{x} \mapsto b * \mathrm{y} \mapsto a \,\} \mid \texttt{let a2 = *x; ??}}\;\text{(Read)}$$

$$\dfrac{\dfrac{\dfrac{\{\, \mathrm{x}, \mathrm{y}, \mathrm{a2}, \mathrm{b2} \,\} ; \{\, \mathrm{x} \mapsto \mathrm{b2} * \mathrm{y} \mapsto \mathrm{b2} \,\} \rightsquigarrow \{\, \mathrm{x} \mapsto \mathrm{b2} * \mathrm{y} \mapsto \mathrm{a2} \,\} \mid \texttt{??}}{\{\, \mathrm{x}, \mathrm{y}, \mathrm{a2}, \mathrm{b2} \,\} ; \{\, \mathrm{x} \mapsto \mathrm{a2} * \mathrm{y} \mapsto \mathrm{b2} \,\} \rightsquigarrow \{\, \mathrm{x} \mapsto \mathrm{b2} * \mathrm{y} \mapsto \mathrm{a2} \,\} \mid \texttt{*x = b2; ??}}\;\text{(Write)}}{\{\, \mathrm{x}, \mathrm{y}, \mathrm{a2} \,\} ; \{\, \mathrm{x} \mapsto \mathrm{a2} * \mathrm{y} \mapsto b \,\} \rightsquigarrow \{\, \mathrm{x} \mapsto b * \mathrm{y} \mapsto \mathrm{a2} \,\} \mid \texttt{let b2 = *y; ??}}\;\text{(Read)}}{\{\, \mathrm{x}, \mathrm{y} \,\} ; \{\, \mathrm{x} \mapsto a * \mathrm{y} \mapsto b \,\} \rightsquigarrow \{\, \mathrm{x} \mapsto b * \mathrm{y} \mapsto a \,\} \mid \texttt{let a2 = *x; ??}}\;\text{(Read)}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\{\, x, y, a2, b2 \,\} ; \{\, y \mapsto b2 \,\} \rightsquigarrow \{\, y \mapsto a2 \,\} \mid \texttt{??}}{\{\, x, y, a2, b2 \,\} ; \{\, x \mapsto b2 \ast y \mapsto b2 \,\} \rightsquigarrow \{\, x \mapsto b2 \ast y \mapsto a2 \,\} \mid \texttt{??}}\ \text{(Frame)}}{\{\, x, y, a2, b2 \,\} ; \{\, x \mapsto a2 \ast y \mapsto b2 \,\} \rightsquigarrow \{\, x \mapsto b2 \ast y \mapsto a2 \,\} \mid \texttt{*x = b2; ??}}\ \text{(Write)}}{\{\, x, y, a2 \,\} ; \{\, x \mapsto a2 \ast y \mapsto \mathit{b} \,\} \rightsquigarrow \{\, x \mapsto \mathit{b} \ast y \mapsto a2 \,\} \mid \texttt{let b2 = *y; ??}}\ \text{(Read)}}{\{\, x, y \,\} ; \{\, x \mapsto \mathit{a} \ast y \mapsto \mathit{b} \,\} \rightsquigarrow \{\, x \mapsto \mathit{b} \ast y \mapsto \mathit{a} \,\} \mid \texttt{let a2 = *x; ??}}\ \text{(Read)}$$

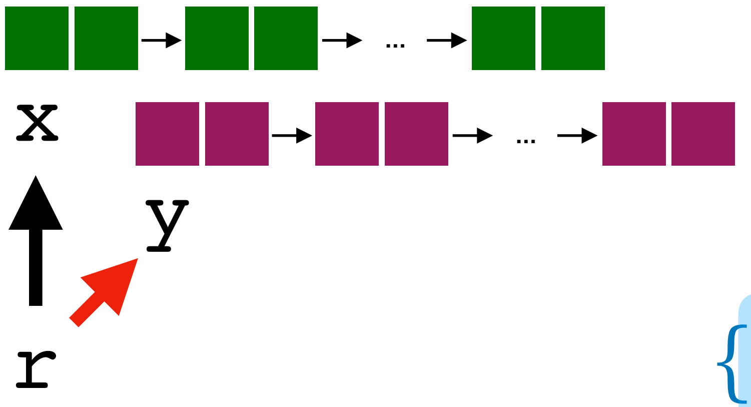
$$
\begin{array}{cl}
\{\, \mathrm{x}, \mathrm{y}, \mathrm{a2}, \mathrm{b2} \,\} ; \{\, \mathrm{y} \mapsto \mathrm{a2} \,\} \rightsquigarrow \{\, \mathrm{y} \mapsto \mathrm{a2} \,\} \mid \texttt{??} & \\
\hline
\{\, \mathrm{x}, \mathrm{y}, \mathrm{a2}, \mathrm{b2} \,\} ; \{\, \mathrm{y} \mapsto \mathrm{b2} \,\} \rightsquigarrow \{\, \mathrm{y} \mapsto \mathrm{a2} \,\} \mid \texttt{*y = a2; ??} & (\text{Write}) \\
\hline
\{\, \mathrm{x}, \mathrm{y}, \mathrm{a2}, \mathrm{b2} \,\} ; \{\, \mathrm{x} \mapsto \mathrm{b2} * \mathrm{y} \mapsto \mathrm{b2} \,\} \rightsquigarrow \{\, \mathrm{x} \mapsto \mathrm{b2} * \mathrm{y} \mapsto \mathrm{a2} \,\} \mid \texttt{??} & (\text{Frame}) \\
\hline
\{\, \mathrm{x}, \mathrm{y}, \mathrm{a2}, \mathrm{b2} \,\} ; \{\, \mathrm{x} \mapsto \mathrm{a2} * \mathrm{y} \mapsto \mathrm{b2} \,\} \rightsquigarrow \{\, \mathrm{x} \mapsto \mathrm{b2} * \mathrm{y} \mapsto \mathrm{a2} \,\} \mid \texttt{*x = b2; ??} & (\text{Write}) \\
\hline
\{\, \mathrm{x}, \mathrm{y}, \mathrm{a2} \,\} ; \{\, \mathrm{x} \mapsto \mathrm{a2} * \mathrm{y} \mapsto b \,\} \rightsquigarrow \{\, \mathrm{x} \mapsto b * \mathrm{y} \mapsto \mathrm{a2} \,\} \mid \texttt{let b2 = *y; ??} & (\text{Read}) \\
\hline
\{\, \mathrm{x}, \mathrm{y} \,\} ; \{\, \mathrm{x} \mapsto a * \mathrm{y} \mapsto b \,\} \rightsquigarrow \{\, \mathrm{x} \mapsto b * \mathrm{y} \mapsto a \,\} \mid \texttt{let a2 = *x; ??} & (\text{Read})
\end{array}
$$

$$\begin{array}{cl}
\{\, x, y, a2, b2 \,\} \,;\, \{\, \mathsf{emp} \,\} \rightsquigarrow \{\, \mathsf{emp} \,\} \mid \texttt{??} & \\
\hline
\{\, x, y, a2, b2 \,\} \,;\, \{\, y \mapsto a2 \,\} \rightsquigarrow \{\, y \mapsto a2 \,\} \mid \texttt{??} & (\text{Frame}) \\
\hline
\{\, x, y, a2, b2 \,\} \,;\, \{\, y \mapsto b2 \,\} \rightsquigarrow \{\, y \mapsto a2 \,\} \mid \texttt{*y = a2; ??} & (\text{Write}) \\
\hline
\{\, x, y, a2, b2 \,\} \,;\, \{\, x \mapsto b2 * y \mapsto b2 \,\} \rightsquigarrow \{\, x \mapsto b2 * y \mapsto a2 \,\} \mid \texttt{??} & (\text{Frame}) \\
\hline
\{\, x, y, a2, b2 \,\} \,;\, \{\, x \mapsto a2 * y \mapsto b2 \,\} \rightsquigarrow \{\, x \mapsto b2 * y \mapsto a2 \,\} \mid \texttt{*x = b2; ??} & (\text{Write}) \\
\hline
\{\, x, y, a2 \,\} \,;\, \{\, x \mapsto a2 * y \mapsto b \,\} \rightsquigarrow \{\, x \mapsto b * y \mapsto a2 \,\} \mid \texttt{let b2 = *y; ??} & (\text{Read}) \\
\hline
\{\, x, y \,\} \,;\, \{\, x \mapsto a * y \mapsto b \,\} \rightsquigarrow \{\, x \mapsto b * y \mapsto a \,\} \mid \texttt{let a2 = *x; ??} & (\text{Read})
\end{array}$$

$$
\dfrac{
  \dfrac{
    \dfrac{
      \dfrac{
        \dfrac{
          \dfrac{
            \dfrac{}{\{\, x, y, a2, b2 \,\} ;\ \{\, \mathrm{emp} \,\} \rightsquigarrow \{\, \mathrm{emp} \,\} \mid \texttt{skip}}\;(\text{Emp})
          }{\{\, x, y, a2, b2 \,\} ;\ \{\, y \mapsto a2 \,\} \rightsquigarrow \{\, y \mapsto a2 \,\} \mid \texttt{??}}\;(\text{Frame})
        }{\{\, x, y, a2, b2 \,\} ;\ \{\, y \mapsto b2 \,\} \rightsquigarrow \{\, y \mapsto a2 \,\} \mid \boxed{\texttt{*y = a2;}}\ \texttt{??}}\;(\text{Write})
      }{\{\, x, y, a2, b2 \,\} ;\ \{\, x \mapsto b2 * y \mapsto b2 \,\} \rightsquigarrow \{\, x \mapsto b2 * y \mapsto a2 \,\} \mid \texttt{??}}\;(\text{Frame})
    }{\{\, x, y, a2, b2 \,\} ;\ \{\, x \mapsto a2 * y \mapsto b2 \,\} \rightsquigarrow \{\, x \mapsto b2 * y \mapsto a2 \,\} \mid \boxed{\texttt{*x = b2;}}\ \texttt{??}}\;(\text{Write})
  }{\{\, x, y, a2 \,\} ;\ \{\, x \mapsto a2 * y \mapsto b \,\} \rightsquigarrow \{\, x \mapsto b * y \mapsto a2 \,\} \mid \boxed{\texttt{let b2 = *y;}}\ \texttt{??}}\;(\text{Read})
}{\{\, x, y \,\} ;\ \{\, x \mapsto a * y \mapsto b \,\} \rightsquigarrow \{\, x \mapsto b * y \mapsto a \,\} \mid \boxed{\texttt{let a2 = *x;}}\ \texttt{??}}\;(\text{Read})
$$

```
void swap(loc x, loc y) {
    let a2 = *x;
    let b2 = *y;
    *x = b2;
    *y = a2;
}
```

Copying a linked list



r “points-to” x

$$\{\mathtt{r} \mapsto x * \mathsf{sll}(x, s)\}$$

inductive predicate “singly-linked list”

```
void sll_copy(loc r)
```

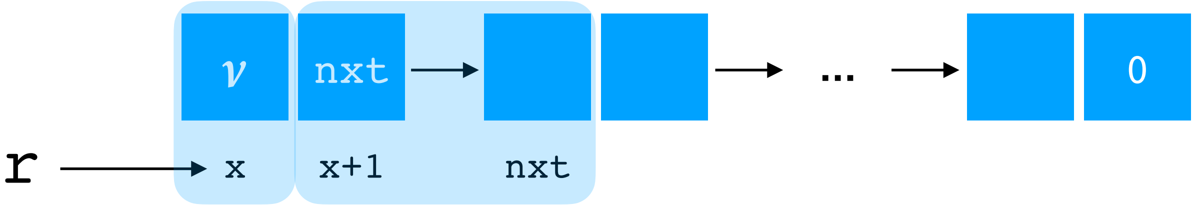
$$\{\mathtt{r} \mapsto y * \mathsf{sll}(x, s) * \mathsf{sll}(y, s)\}$$

separating conjunction

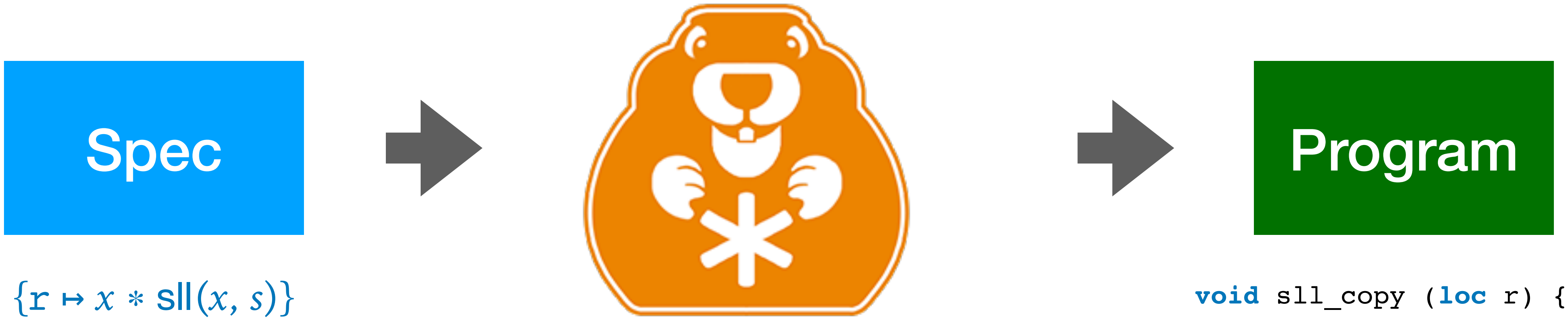
The linked list predicate

$$\textbf{sll}(x, s) \triangleq x = 0 \land \{s = \emptyset, \textbf{emp}\}$$

$$\mid\ x \neq 0 \land \{s = \{v\} \cup s_1 \land [x,2] \ * \ x \mapsto v \ * \ (x+1) \mapsto nxt \ * \ \textbf{sll}(nxt, s_1)\}$$



Generating code from logical spec



Spec

Program

$\{\mathtt{r} \mapsto x * \mathsf{sll}(x, s)\}$

```
void sll_copy(loc r)
```

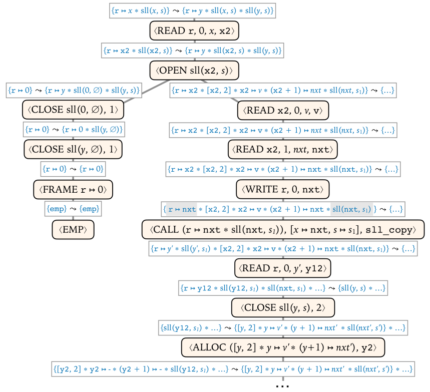
$\{\mathtt{r} \mapsto y * \mathsf{sll}(x, s) * \mathsf{sll}(y, s)\}$

Automatically produce
an implementation

```
void sll_copy (loc r) {
  let x2 = *r;
  if (x2 == 0) {}
  else {
    let v = *x2;
    let nxt = *(x2 + 1);
    *r = nxt;
    sll_copy(r);
    let y12 = *r;
    let y2 = malloc(2);
    *r = y2;
    *(y2 + 1) = y12;
    *y2 = v;
  }
}
```

Deductive Program Synthesis: Summary

Derivation trace


{r ↦ x ∗ sll(x, s)} ⤳ {r ↦ y ∗ sll(x, s) ∗ sll(y, s)}
⟨READ r, 0, x, x2⟩
{r ↦ x2 ∗ sll(x2, s)} ⤳ {r ↦ y ∗ sll(x2, s) ∗ sll(y, s)}
⟨OPEN sll(x2, s)⟩
{r ↦ 0} ⤳ {r ↦ y ∗ sll(0, ∅) ∗ sll(y, s)}
⟨CLOSE sll(0, ∅), 1⟩
{r ↦ 0} ⤳ {r ↦ 0 ∗ sll(y, ∅)}
⟨CLOSE sll(y, ∅), 1⟩
{r ↦ 0} ⤳ {r ↦ 0}
⟨FRAME r ↦ 0⟩
{emp} ⤳ {emp}
⟨EMP⟩
{r ↦ x2 ∗ [x2, 2] ∗ x2 ↦ v ∗ (x2 + 1) ↦ nxt ∗ sll(nxt, s₁)} ⤳ {...}
⟨READ x2, 0, v, v⟩
{r ↦ x2 ∗ [x2, 2] ∗ x2 ↦ v ∗ (x2 + 1) ↦ nxt ∗ sll(nxt, s₁)} ⤳ {...}
⟨READ x2, 1, nxt, nxt⟩
{r ↦ x2 ∗ [x2, 2] ∗ x2 ↦ v ∗ (x2 + 1) ↦ nxt ∗ sll(nxt, s₁)} ⤳ {...}
⟨WRITE r, 0, nxt⟩
{ r ↦ nxt ∗ [x2, 2] ∗ x2 ↦ v ∗ (x2 + 1) ↦ nxt ∗ sll(nxt, s₁) } ⤳ {...}
⟨CALL (r ↦ nxt ∗ sll(nxt, s₁)), [x ↦ nxt, s ↦ s₁], sll_copy⟩
{r ↦ y′ ∗ sll(y′, s₁) ∗ [x2, 2] ∗ x2 ↦ v ∗ (x2 + 1) ↦ nxt ∗ sll(nxt, s₁)} ⤳ {...}
⟨READ r, 0, y′, y12⟩
{r ↦ y12 ∗ sll(y12, s₁) ∗ sll(nxt, s₁) ∗ ...} ⤳ {sll(y, s) ∗ ...}
⟨CLOSE sll(y, s), 2⟩
{sll(y12, s₁) ∗ ...} ⤳ {[y, 2] ∗ y ↦ v′ ∗ (y + 1) ↦ nxt′ ∗ sll(nxt′, s′)} ∗ ...}
⟨ALLOC ([y, 2] ∗ y ↦ v′ ∗ (y+1) ↦ nxt′), y2⟩
{[y2, 2] ∗ y2 ↦ - ∗ (y2 + 1) ↦ - ∗ sll(y12, s₁) ∗ ...} ⤳ {[y, 2] ∗ y ↦ v′ ∗ (y + 1) ↦ nxt′ ∗ sll(nxt′, s′)} ∗ ...}
...

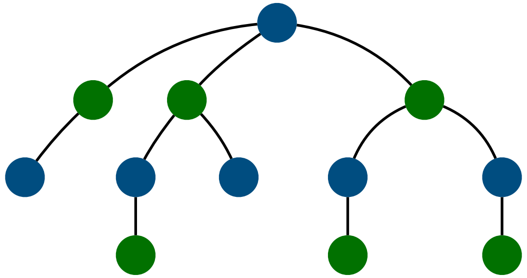
Initial specification

Proof search

$$\{r \mapsto x * \mathsf{sll}(x, S)\}$$

```
void sll_copy(loc r)
```

$$\{r \mapsto y * \mathsf{sll}(x, S) * \mathsf{sll}(y, S)\}$$



Program (byproduct)

```
void sll_copy (loc r) {
  let x2 = *r;
  if (x2 == 0) {}
  else {
    let v = *x2;
    let nxt = *(x2 + 1);
    *r = nxt;
    sll_copy(r);
    let y12 = *r;
    let y2 = malloc(2);
    *(y2 + 1) = y12;
    *y2 = v;
  }
}
```

Deductive Program Synthesis: Summary

Initi

$\{r$

voi

$\{r \mapsto y$

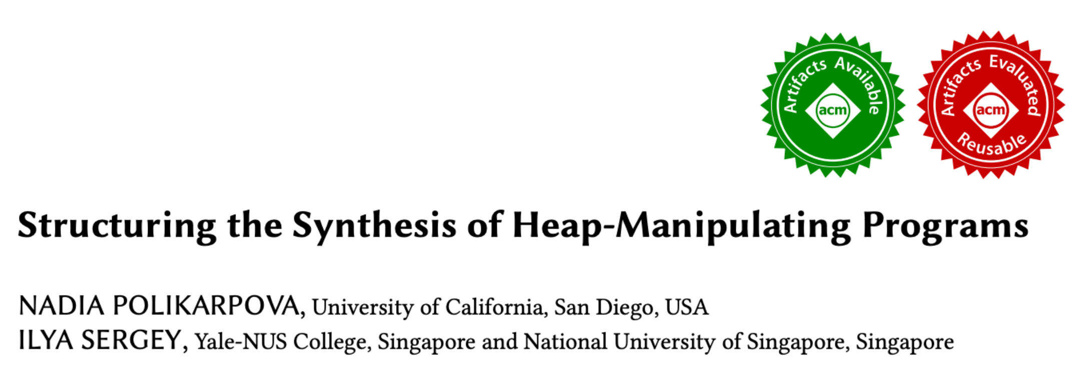
ct)

```
    *y2 = v;
  }
}
```

SuSLik

A **deductive synthesizer** that uses inference rules of **Synthetic Separation Logic (SSL)** to generate imperative, **heap-manipulating programs**

demo!



Artifacts Available

Artifacts Evaluated Reusable

Structuring the Synthesis of Heap-Manipulating Programs

NADIA POLIKARPOVA, University of California, San Diego, USA
ILYA SERGEY, Yale-NUS College, Singapore and National University of Singapore, Singapore

Data Structure	*Id*	*Description*	*Proc*	*Stmt*	*Code/Spec*	*Time*
Integers	1	swap two	1	4	1.0x	0.2
	2	min of two[1]	1	3	1.1x	0.8
Singly Linked List	3	length[2]	1	6	1.4x	0.4
	4	max[2]	1	11	1.9x	3.0
	5	min[2]	1	11	1.9x	2.9
	6	singleton[1]	1	4	0.9x	0.2
	7	deallocate	1	4	5.5x	0.2
	8	initialize	1	4	1.6x	0.4
	9	copy[3]	1	11	2.7x	0.6
	10	append[3]	1	6	1.1x	0.4
	11	delete[3]	1	12	2.6x	1.2
	12	deallocate two	2	9	6.2x	0.2
	13	append three	2	14	2.3x	1.0
	14	non-destructive append	2	21	3.0x	8.0
	15	union	2	23	5.5x	4.3
	16	intersection[4]	3	32	7.0x	101.1
	17	difference[4]	2	21	5.1x	4.7
	18	deduplicate[4]	2	22	7.3x	1.8
Sorted list	19	prepend[2]	1	4	0.4x	0.2
	20	insert[2]	1	19	3.1x	1.0
	21	insertion sort[2]	1	7	1.2x	0.7
	22	sort[4]	2	13	4.9x	1.0
	23	reverse[4]	2	11	4.0x	0.7
	24	merge[2]	2	30	4.4x	55.6
Doubly Linked List	25	singleton[1]	1	5	1.1x	0.2
	26	copy	1	22	4.3x	7.2
	27	append[3]	1	10	1.6x	1.7
	28	delete[3]	1	19	3.7x	3.4
	29	single to double	1	23	6.0x	0.7

Data Structure	*Id*	*Description*	*Proc*	*Stmt*	*Code/Spec*	*Time*
Doubly Linked List	25	singleton[1]	1	5	1.1x	0.2
	26	copy	1	22	4.3x	7.2
	27	append[3]	1	10	1.6x	1.7
	28	delete[3]	1	19	3.7x	3.4
	29	single to double	1	23	6.0x	0.7
List of Lists	30	deallocate	2	11	10.7x	0.2
	31	flatten[4]	2	17	4.4x	0.6
	32	length[5]	2	21	5.5x	22.8
Binary Tree	33	size	1	9	2.5x	0.4
	34	deallocate	1	6	8.0x	0.2
	35	deallocate two	1	16	11.8x	0.4
	36	copy	1	16	3.8x	2.5
	37	flatten w/append	1	17	4.8x	0.4
	38	flatten w/acc	1	12	2.1x	0.6
	39	flatten	2	23	7.1x	1.5
	40	flatten to dll in place	2	15	9.6x	11.3
	41	flatten to dll w/null[5]	2	17	11.2x	106.1
BST	42	insert[2]	1	19	2.8x	14.6
	43	rotate left[2]	1	5	0.2x	6.2
	44	rotate right[2]	1	5	0.2x	4.9
	45	find min[5]	1	11	1.4x	66.3
	46	find max[5]	1	18	2.2x	58.0
	47	delete root[2]	1	18	1.3x	13.9
	48	from list[4]	2	27	5.7x	10.0
	49	to sorted list[4]	3	32	7.7x	20.8
Rose Tree	50	deallocate	2	9	12.0x	0.2
	51	flatten	3	25	8.0x	11.0
	52	copy[5]	2	32	7.9x	-
Packed Tree	53	pack[5]	1	16	1.6x	-
	54	unpack[5]	1	23	2.9x	21.0

logic and types: discussion

Domains:

- API discovery: Sypet, Hoogle+
- verified functional programs: Synquid
- verified programs with pointers: SuSLik

+ concise
+ provably correct
+ can express non-functional properties
+ guide search

- simple specs are ambiguous
- advanced specs require expertise
- some things are hard to express / check

Discussion: trust in synthesis

$$\{r \mapsto x * \mathsf{sll}(x, S)\}$$

```
void sll_copy(loc r)
```

$$\{r \mapsto y * \mathsf{sll}(x, S) * \mathsf{sll}(y, S)\}$$

```
void sll_copy (loc r) {
  let x2 = *r;
  if (x2 == 0) {}
  else {
    let v = *x2;
    let nxt = *(x2 + 1);
    *r = nxt;
    sll_copy(r);
    let y12 = *r;
    let y2 = malloc(2);
    *(y2 + 1) = y12;
    *y2 = v;
  }
}
```

$$\{\mathsf{r} \mapsto x * \mathsf{sll}(x, S)\} \rightsquigarrow \{\mathsf{r} \mapsto y * \mathsf{sll}(x, S) * \mathsf{sll}(y, S)\}$$

What’s wrong?

```
void sll_copy (loc r) {
  let x2 = *r;
  if (x2 == 0) {}
  else {
    let v = *x2;
    let nxt = *(x2 + 1);
    *r = nxt;
    sll_copy(r);
    let y12 = *r;
    let y2 = malloc(2);
    *(y2 + 1) = y12;
    *y2 = v;
  }
}
```

$$\{\mathsf{r} \mapsto x * \mathsf{sll}(x, S)\} \rightsquigarrow \{\mathbf{r} \mapsto y * \mathsf{sll}(x, S) * \mathsf{sll}(y, S)\}$$

There’s a bug.

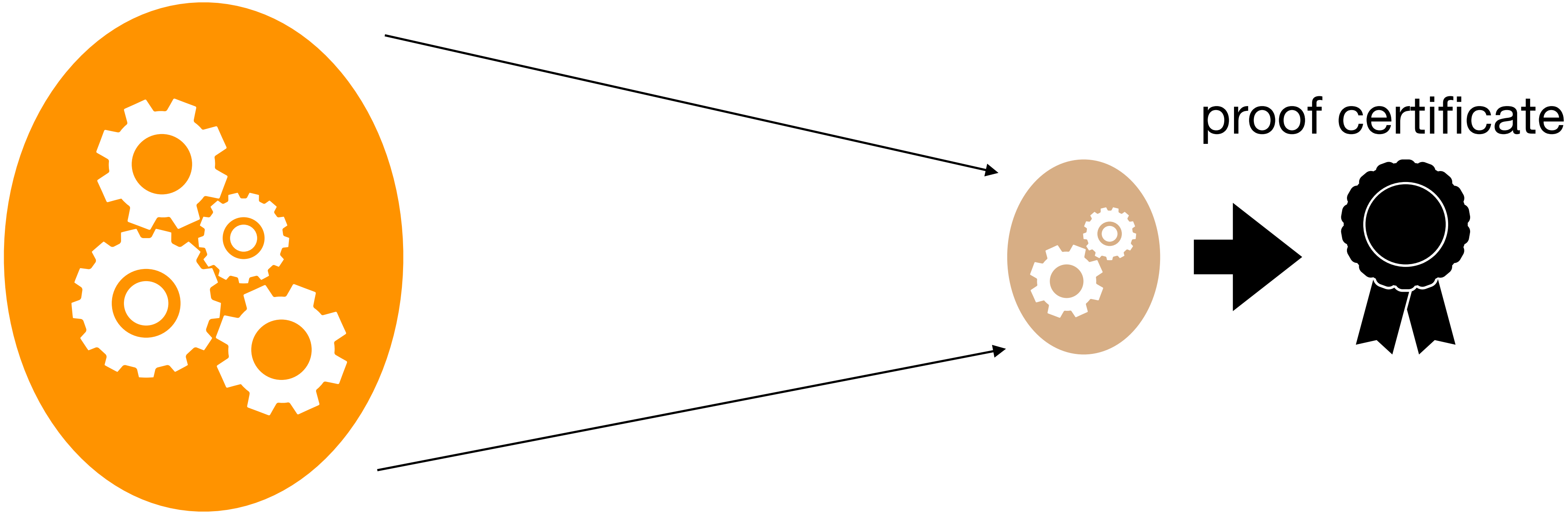
```
void sll_copy (loc r) {
  let x2 = *r;
  if (x2 == 0) {}
  else {
    let v = *x2;
    let nxt = *(x2 + 1);
    *r = nxt;
    sll_copy(r);
    let y12 = *r;
    let y2 = malloc(2);
    *r = y2;
    *(y2 + 1) = y12;
    *y2 = v;
  }
}
```

How can we trust what the synthesiser gives us?

Meet the Coq Proof Assistant

- *State-of-the* art verification framework
- Based on *dependently typed functional language*
- *Interactive* — requires a human in the loop
- Very small *trusted code base*
- Used to implement fully verified
  - *compilers*
  - *operating systems*
  - *distributed protocols (including blockchains)*

Shifting the burden of trust

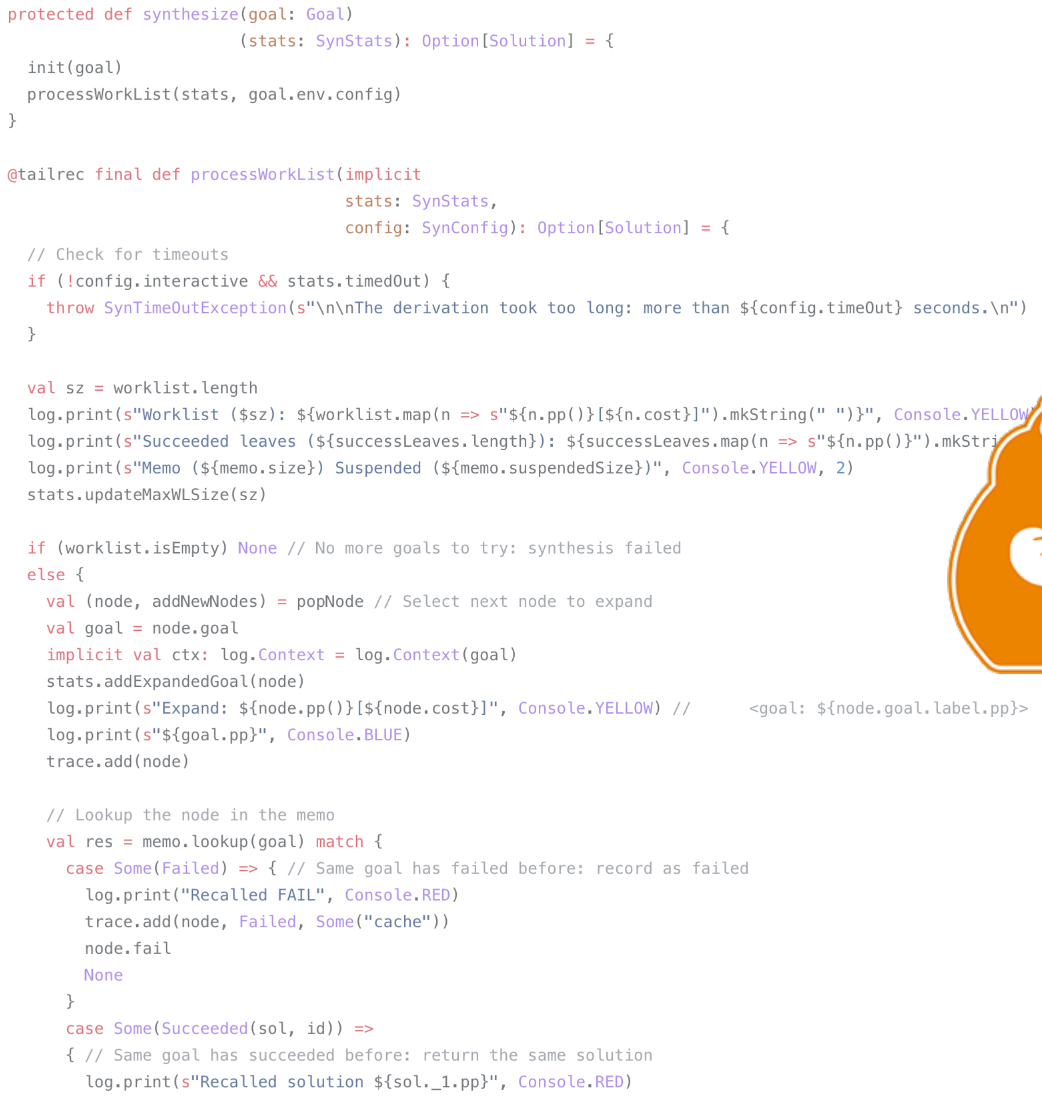


proof certificate

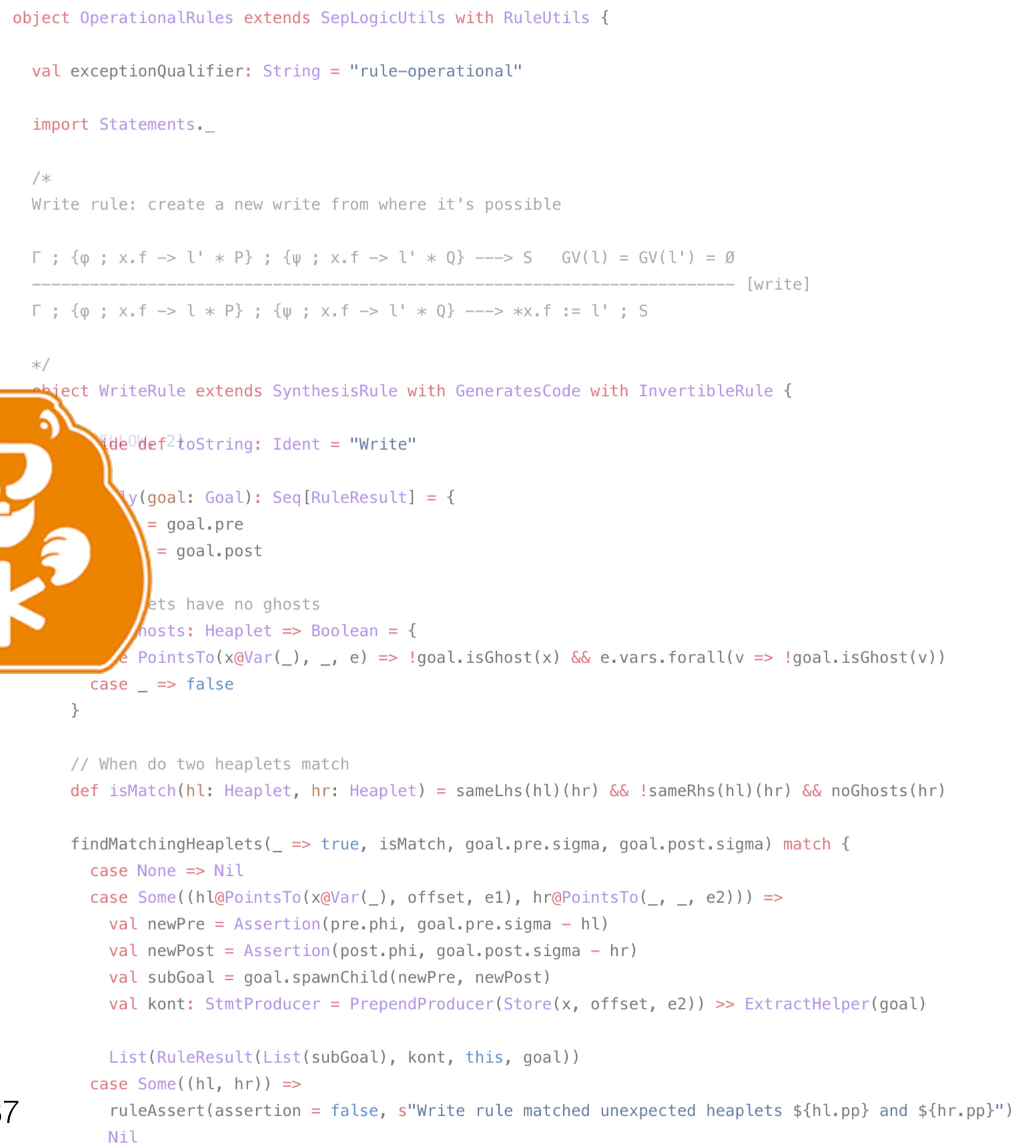
SuSLik: Large TCB

Coq: Small TCB

SuSLik codebase: too large to verify

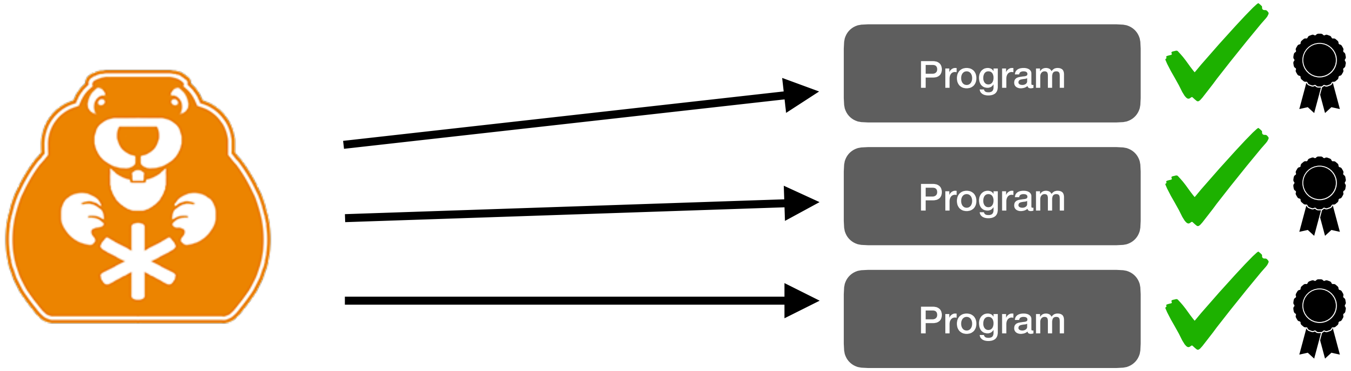


```
protected def synthesize(goal: Goal)
                        (stats: SynStats): Option[Solution] = {
  init(goal)
  processWorkList(stats, goal.env.config)
}

@tailrec final def processWorkList(implicit
                                   stats: SynStats,
                                   config: SynConfig): Option[Solution] = {
  // Check for timeouts
  if (!config.interactive && stats.timedOut) {
    throw SynTimeOutException(s"\n\nThe derivation took too long: more than ${config.timeOut} seconds.\n")
  }

  val sz = worklist.length
  log.print(s"Worklist ($sz): ${worklist.map(n => s"${n.pp()}[${n.cost}]").mkString(" ")}", Console.YELLOW
  log.print(s"Succeeded leaves (${successLeaves.length}): ${successLeaves.map(n => s"${n.pp()}").mkStri
  log.print(s"Memo (${memo.size}) Suspended (${memo.suspendedSize})", Console.YELLOW, 2)
  stats.updateMaxWLSize(sz)

  if (worklist.isEmpty) None // No more goals to try: synthesis failed
  else {
    val (node, addNewNodes) = popNode // Select next node to expand
    val goal = node.goal
    implicit val ctx: log.Context = log.Context(goal)
    stats.addExpandedGoal(node)
    log.print(s"Expand: ${node.pp()}[${node.cost}]", Console.YELLOW) //      <goal: ${node.goal.label.pp}>
    log.print(s"${goal.pp}", Console.BLUE)
    trace.add(node)

    // Lookup the node in the memo
    val res = memo.lookup(goal) match {
      case Some(Failed) => { // Same goal has failed before: record as failed
        log.print("Recalled FAIL", Console.RED)
        trace.add(node, Failed, Some("cache"))
        node.fail
        None
      }
      case Some(Succeeded(sol, id)) =>
      { // Same goal has succeeded before: return the same solution
        log.print(s"Recalled solution ${sol._1.pp}", Console.RED)
```



```
object OperationalRules extends SepLogicUtils with RuleUtils {

  val exceptionQualifier: String = "rule-operational"

  import Statements._

  /*
  Write rule: create a new write from where it's possible

  Γ ; {φ ; x.f -> l' * P} ; {ψ ; x.f -> l' * Q} ---> S   GV(l) = GV(l') = Ø
  ------------------------------------------------------------------------ [write]
  Γ ; {φ ; x.f -> l * P} ; {ψ ; x.f -> l' * Q} ---> *x.f := l' ; S

  */
  ject WriteRule extends SynthesisRule with GeneratesCode with InvertibleRule {

          ide def toString: Ident = "Write"

             y(goal: Goal): Seq[RuleResult] = {
               = goal.pre
                = goal.post

              ets have no ghosts
             hosts: Heaplet => Boolean = {
            e PointsTo(x@Var(_), _, e) => !goal.isGhost(x) && e.vars.forall(v => !goal.isGhost(v))
        case _ => false
      }

      // When do two heaplets match
      def isMatch(hl: Heaplet, hr: Heaplet) = sameLhs(hl)(hr) && !sameRhs(hl)(hr) && noGhosts(hr)

      findMatchingHeaplets(_ => true, isMatch, goal.pre.sigma, goal.post.sigma) match {
        case None => Nil
        case Some((hl@PointsTo(x@Var(_), offset, e1), hr@PointsTo(_, _, e2))) =>
          val newPre = Assertion(pre.phi, goal.pre.sigma - hl)
          val newPost = Assertion(post.phi, goal.post.sigma - hr)
          val subGoal = goal.spawnChild(newPre, newPost)
          val kont: StmtProducer = PrependProducer(Store(x, offset, e2)) >> ExtractHelper(goal)

          List(RuleResult(List(subGoal), kont, this, goal))
        case Some((hl, hr)) =>
          ruleAssert(assertion = false, s"Write rule matched unexpected heaplets ${hl.pp} and ${hr.pp}")
          Nil
```

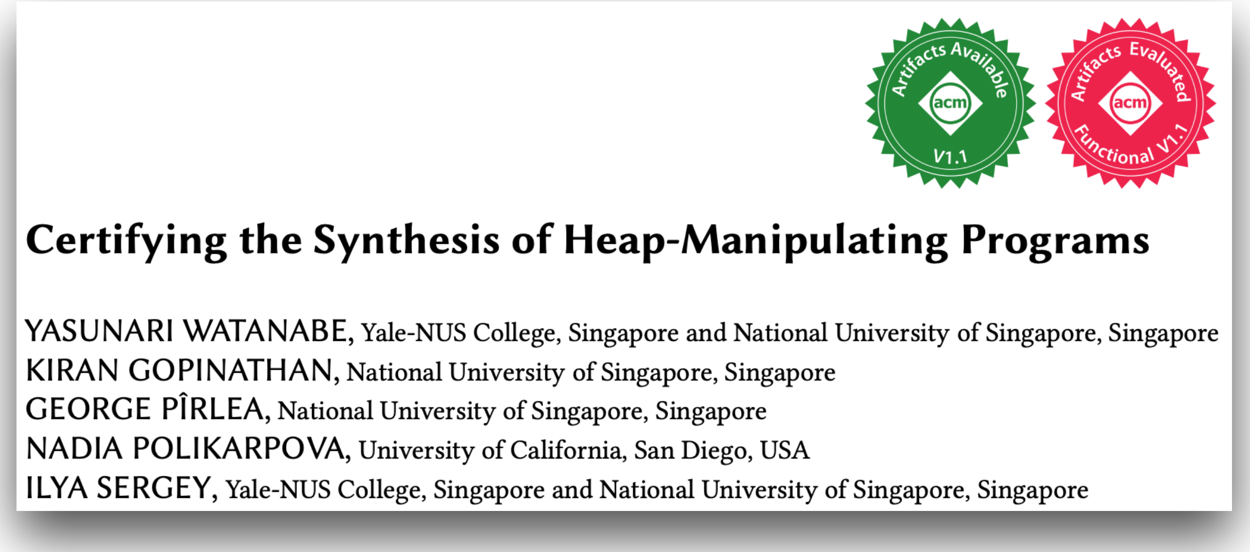
Deductive insight → post-hoc certification



Program ✓

Program ✓

Program ✓



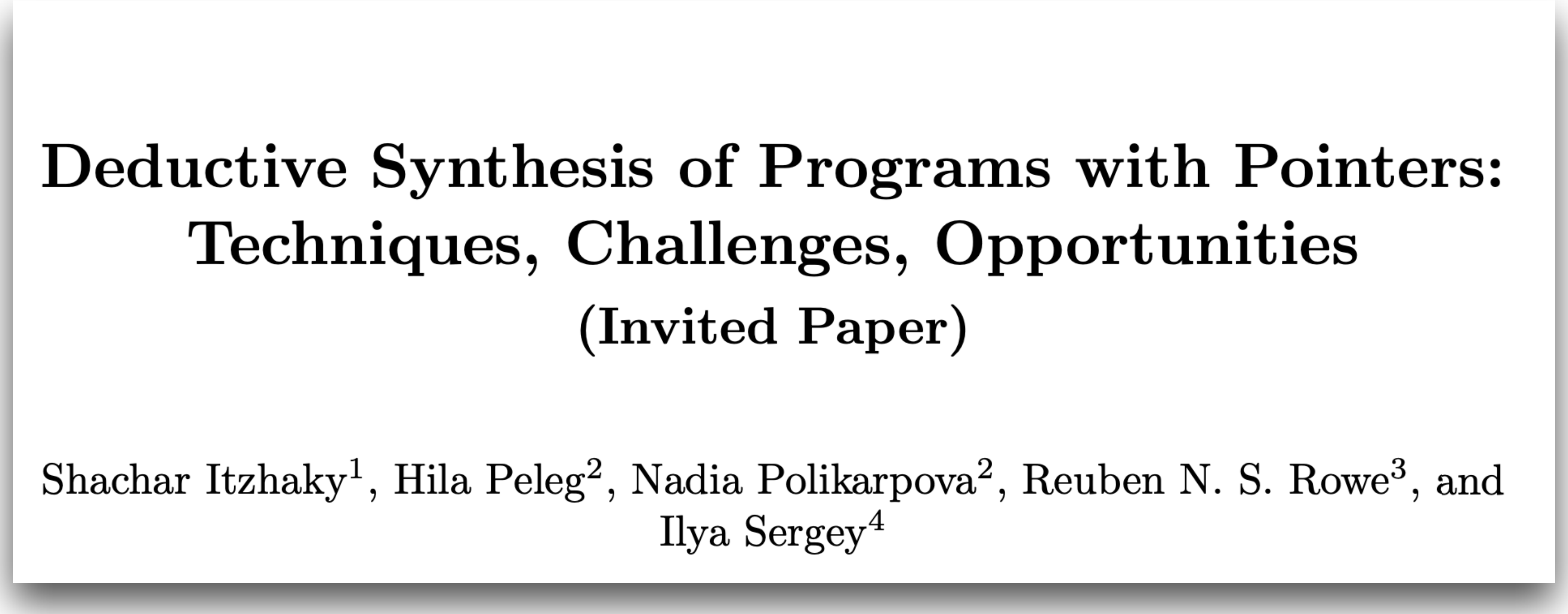
Artifacts Available acm V1.1

Artifacts Evaluated acm Functional V1.1

Certifying the Synthesis of Heap-Manipulating Programs

YASUNARI WATANABE, Yale-NUS College, Singapore and National University of Singapore, Singapore
KIRAN GOPINATHAN, National University of Singapore, Singapore
GEORGE PÎRLEA, National University of Singapore, Singapore
NADIA POLIKARPOVA, University of California, San Diego, USA
ILYA SERGEY, Yale-NUS College, Singapore and National University of Singapore, Singapore

Future Directions



Deductive Synthesis of Programs with Pointers: Techniques, Challenges, Opportunities

(Invited Paper)

Shachar Itzhaky[1], Hila Peleg[2], Nadia Polikarpova[2], Reuben N. S. Rowe[3], and Ilya Sergey[4]

synthesis: approaches and challenges

synthesis is more than just PBE/PBD!

depends on target domain and audience

targeting programmers? can use programs or logical specs!

good specifications are difficult to write

one needs to capture the intent *exactly*

challenge: trust in synthesis

NL + examples: how many examples are needed?

using proof assistants

Thanks!