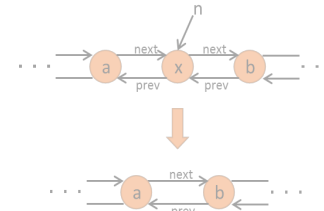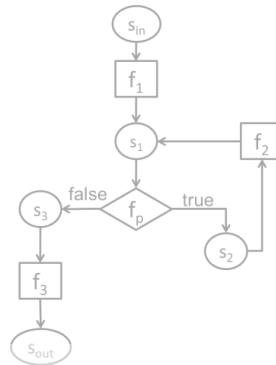$$\exists c \forall in \; Q(c, in)$$

```
/* Average of x and y without using x+y (avoid overflow)*/
int avg(int x, int y){
    int t = expr({x/2, y/2, x%2, y%2, 2 }, {PLUS, DIV});
    assert t == (x+y)/2;
    return t;
}
```
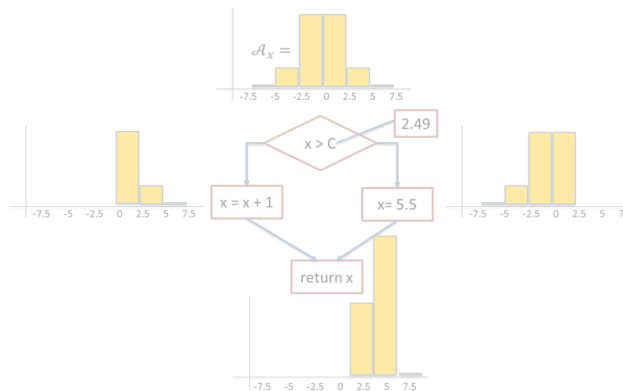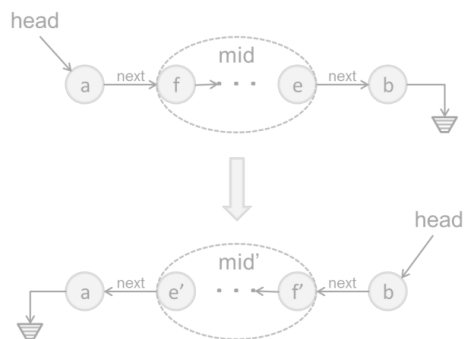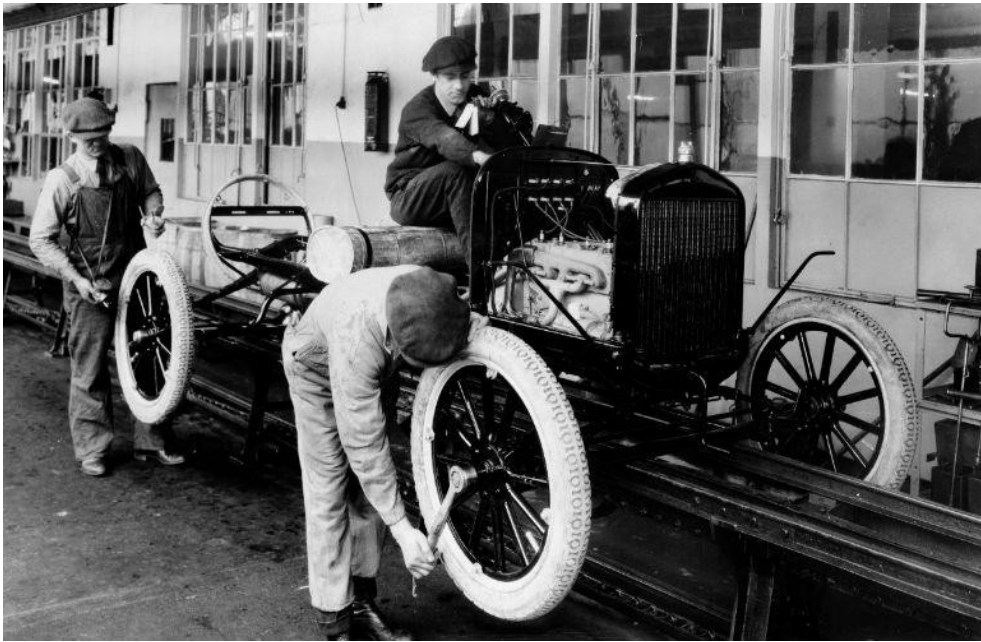
Program Synthesis

$$\varphi(p)$$

$$Sk[c](in)$$

Lecture 1

Introduction to Synthesis

The goal: automate programming

Modern program synthesis: FlashFill

[Gulwani 2011]

FlashFill: a feature of Excel 2013

[Gulwani 2011]

FlashFill: a feature of Excel 2013

Modern program synthesis: SQLizer

[Yaghmazadeh et al. 2017]

Problem: "Find the number of papers in OOPSLA 2010"
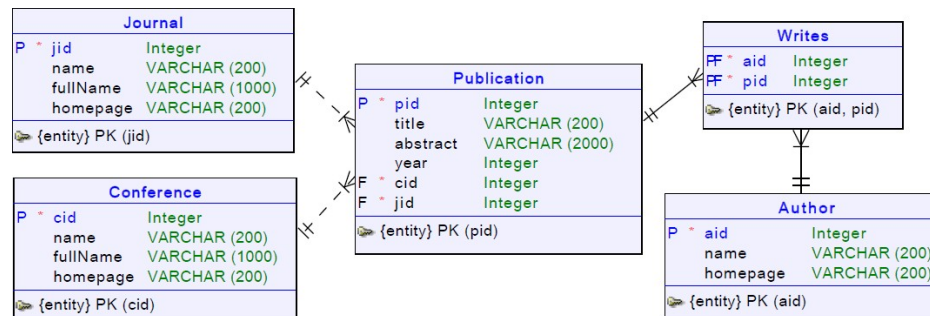
Output:

```
SELECT count(Publication.pid)
FROM Publication JOIN Conference ON Publication.cid = Conference.cid
WHERE Conference.name = "OOPSLA" AND Publication.year = 2010
```

Modern program synthesis: Sketch

[Solar-Lezama 2013]

Problem: isolate the least significant zero bit in a word
- example: 0010 0101 →   0000 0010

Easy to implement with a loop

```
int W = 32;

bit[W] isolate0 (bit[W] x) {        // W: word size
        bit[W] ret = 0;
        for (int i = 0; i < W; 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: space of possible implementations

```
/**
 * Generate the set of all bit-vector expressions
 * involving +, &, xor and bitwise negation (~).
 */

generator bit[W] gen(bit[W] x){
    if(??) return x;
    if(??) return ??;
    if(??) return ~gen(x);
    if(??){
        return {| gen(x) (+ | & | ^) gen(x) |};
    }
}
```

Sketch: synthesis goal

```
generator bit[W] gen(bit[W] x, int depth){
    assert depth > 0;
    if(??) return x;
    if(??) return ??;
    if(??) return ~gen(x, depth-1);
    if(??){
        return {| gen(x, depth-1) (+ | & | ^) gen(x, depth-1) |};
    }
}

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

Sketch: output

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

Modern program synthesis: Synquid

Problem: intersection of strictly sorted lists
- example: intersect [4, 8, 15, 16, 23, 42] [8, 16, 32, 64] →　　[8, 16]

Also: we want a guarantee that it's correct on all inputs!

Synquid: synthesis goal and components

Step 1: define synthesis goal as a *type*

sorted list

```
intersect :: xs:List a   →   ys:List a   →
                 List a
```

the set of elements

Step 2: define a set of components
- Which primitive operations is our function likely to use?
- Here: {`Nil`, `Cons`, `<`}

Synquid: synthesis goal and components

Example: Synquid

specification

program

```
intersect :: xs:SList a  →
  ys:SList a  →
  {v:SList a | elems v = elems xs ∩
                        elems ys}
```

→

```
intersection = \xs . \ys .
  match xs with
    Nil -> xs
    Cons x xt ->
      match ys with
        Nil -> ys
        Cons y yt ->
          if x < y
          then intersection xt ys
          else
            if y < x
            then intersection xs yt
            else Cons x (intersection xt yt)
```

Synquid: output

```
intersection = \xs . \ys .
  match xs with
    Nil -> xs
    Cons x xt ->
      match ys with
        Nil -> ys
        Cons y yt ->
          if x < y
          then intersection xt ys
          else
            if y < x
            then intersection xs yt
            else Cons x (intersection xt yt)
```

xs	ys	result
[4, 8, 15, 16, 23, 42]	[8, 16, 32, 64]	
[8, 15, 16, 23, 42]	[8, 16, 32, 64]	[8]
[15, 16, 23, 42]	[16, 32, 64]	
[16, 23, 42]	[16, 32, 64]	[8, 16]
[23, 42]	[32, 64]	
[42]	[32, 64]	
[42]	[64]	
[]	[64]	

What is program synthesis?

specification   search   program
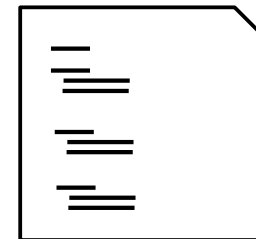
program
space

What is program synthesis?

Automatic programming?

- but I still have to tell the computer what I want...

level of
abstraction

???

Python, Haskell, ...

C

assembly

machine code

Synthesis
=
an unusually concise / intuitive
programming language
+
a compiler based on search

Dimensions in program synthesis

[Gulwani 2010]

Behavioral constraints (aka specification):
how do you tell the system
what the program should do?

Search strategy:
How does the system find the
program you want?

Structural constraints:
what is the space of programs
to explore?
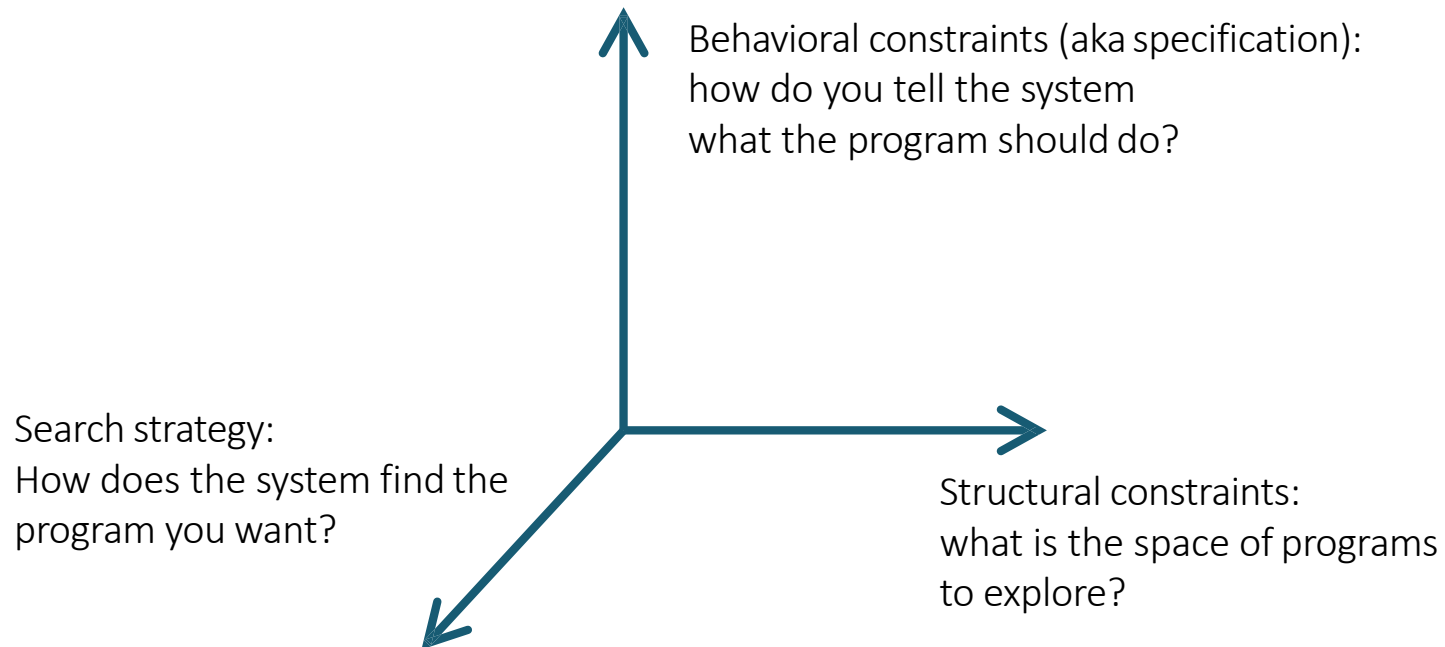
Behavioral constraints
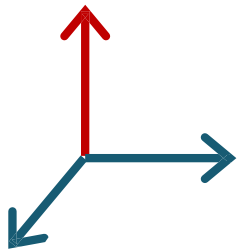
How do you tell the system what the program should do?
- What is the input language / format?
- What is the interaction model?
- What happens when the intent is ambiguous?

Q: What did behavioral constraints look like in FlashFill / Sketch / Synquid / SQLizer?

Behavioral constraints: examples

Input/output examples

Equivalent program

Formal specifications (pre/post conditions, types, …)

Natural language
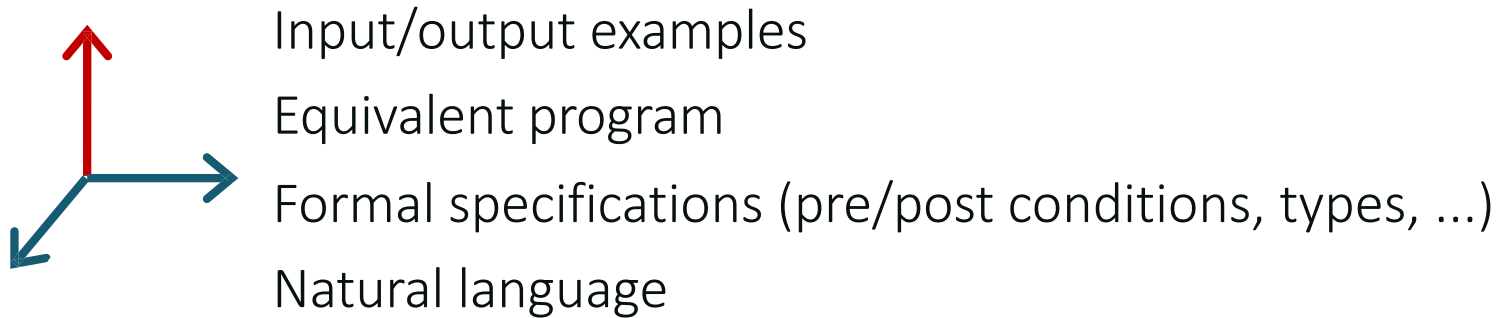
Structural constraints
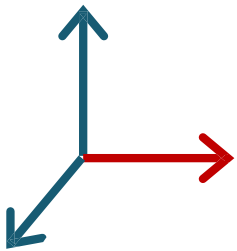
What is the space of programs to explore?

- Large enough to contain interesting programs, yet small enough to exclude garbage and enable efficient search
- Built-in or user defined?
- Can we extract domain knowledge from existing code?

Q: What did structural constraints look like in FlashFill / Sketch / Synquid / SQLizer?

Structural constraints: examples
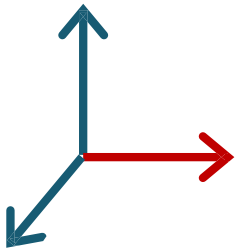
Built-in DSL

User-defined DSL (grammar)

User-provided components

Languages with synthesis constructs
- e.g. generators in Sketch

Search strategies
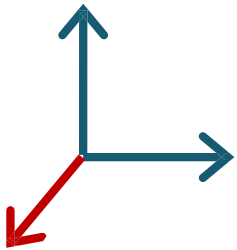
Synthesis is search:

- Find a program in the space defined by *structural constraints* that satisfies *behavioral constraints*

Challenge: the space is astronomically large

- The search algorithm is the heart of a synthesis technique

How does the system find the program you want?

- How does it know it's the program you want?
- How can it leverage structural constraints to guide the search?
- How can it leverage behavioral constraints to guide the search?

Search strategies: examples
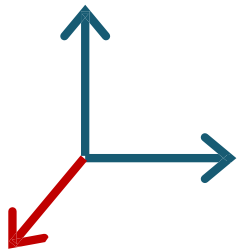
Enumerative (explicit) search
- exhaustively enumerate all programs in the language in the order of increasing size

Stochastic search
- random exploration of the search space guided by a fitness function

Representation-based search
- use a data structure to represent a large set of programs

Constraint-based search
- translate to constraints and use a solver

Structure of the Course

Module 1: Searching for Simple Programs
- Easy to decide when a program is correct
- Challenge: search in a large space

Module 2: Searching for Complex Programs
- Deciding when a program is correct can be hard
- Search in a large space is still a problem

Module 3: Applications of Synthesis
- We can search for programs. Now what?

Module 1: Searching for Simple Programs

Example: FlashFill

specification

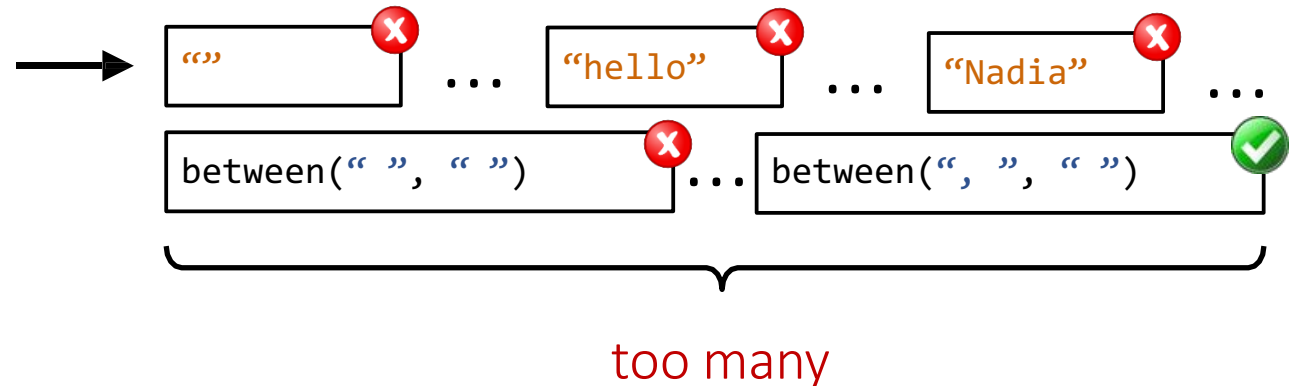1: "Polikarpova, Nadia" → "Nadia"

2: "Van Damme, Jean Claude" → "Jean"

program space

constant string:
  "…"

or substring of input:
  between("…", "…")

"" ... "hello" ... "Nadia" ...

between(" ", " ") ... between(", ", " ")

too many

Module 2: Searching for Complex Programs

Example: Synquid

specification

```
intersect :: xs:SList a  →
  ys:SList a  →
  {v:SList a | elems v = elems xs ∩
                     elems ys}
```

program

```
intersection = \xs . \ys .
  match xs with
    Nil -> xs
    Cons x xt ->
      match ys with
        Nil -> ys
        Cons y yt ->
          if x < y
          then intersection xt ys
          else
            if y < x
            then intersection xs yt
            else Cons x (intersection xt yt)
```

How do we know this program always
produces a sorted list that is the
intersection?

Module 3: Applications of synthesis

Synthesis as a Programming Tool
- How can synthesis help programmers?
- What is the right user interaction model?

Domain-Specific Synthesis
- Super-optimization
- Cryptographic schemes
- SQL Query synthesis
- Graphics kernels

Synthesis as Machine Learning
- Learning interpretable models from few examples