CS409
Software Testing

TAN, Shin Hwei
陈馨慧
Southern University of Science and Technology

Adapted from CSE 503 University of Washington

Administrative Info

- MP3 is released and due on **18 December 2020** (this week) at 11.59pm.
  - Need you to debug and repair open issues in your app!
  - Some common mistakes:
  - Forget to include the link for each of the selected issues (for App A and from similar issue)
- Final Project Presentation is released.
  - 20% of your total grade for the class
  - Presentation will be in lab session (4.20-6.10pm) on 21 December 2020 (next week). All members need to attend and present.
  - Come to the lab today to discuss the final project with your groupmates.
- Final exam currently scheduled to be on 2021-01-04 at 16:30-18:30.

Administrative Info: Schedule

- Dec 18: MP3 due
- Dec 21 Lecture: Final Exam Review
- Dec 21 Lab: Final Presentation (All members need to attend and present)
- Dec 25: Final Report due
- Dec 25: All lab assignments due
- Jan 4: Final exam at 16:30-18:30 荔园2栋101（Lychee Hill, Building 2, Room 101）

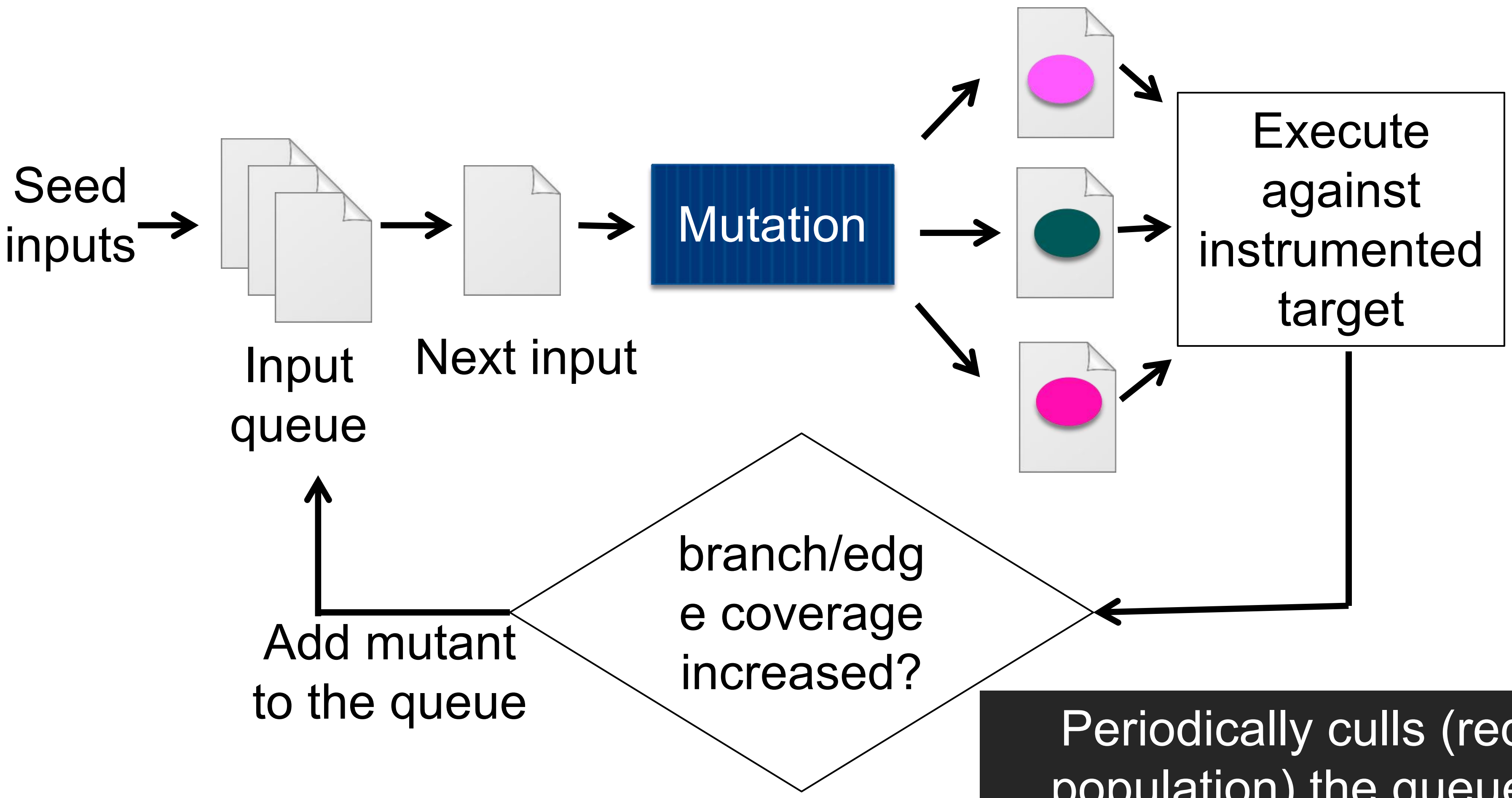
Administrative Info

All lab assignments due on 25 December 2020, 11.59pm:

Remember to write the answers for all question in README.md and include your name and student id for all assignments:

- Android-graph Lab: https://classroom.github.com/a/-wVDOh_I
- Fuzzing Lab: https://classroom.github.com/a/WOPbCjnZ
- Graph Lab: https://classroom.github.com/a/rkg8YIET
- ISP-Lab(group assignment): https://classroom.github.com/g/tCTOdiKH
- Junit-Lab1: https://classroom.github.com/a/TnI4NoVY
- Junit2: https://classroom.github.com/a/8TQabGyd
- Logic coverage lab: https://classroom.github.com/a/6i6xSkX7
- Logic source code lab: https://classroom.github.com/a/WbKNTVOr
- Monkey delta lab: https://classroom.github.com/a/yq3B85aj
- TDD lab(group assignment): https://classroom.github.com/g/Rf2Mkwo7

Recap: American Fuzzy Lop (AFL)



Periodically culls (reduce the population) the queue without affecting total coverage

AFL

- Instrument the binary at compile-time
- Regular mode: instrument assembly
- Recent addition: LLVM compiler instrumentation mode
- Provide 64K counters representing all edges in the app
- Hashtable keeps track of # of execution of edges
  - 8 bits per edge (# of executions: 1, 2, 3, 4-7, 8-15, 16-31, 32-127, 128+)
  - Imprecise (edges may collide) but very efficient
- AFL-fuzz is the driver process, the target app runs as separate process(es)

Data-flow-guided fuzzing

- Intercept the data flow, analyze the inputs of comparisons
  - Incurs extra overhead
- Modify the test inputs, observe the effect on comparisons
- Prototype implementations in libFuzzer and go-fuzz

Fuzzing challenges

- How to seed a fuzzer?
  - Seed inputs must cover different branches
  - Remove duplicate seeds covering the same branches
  - Small seeds are better (Why?)

- Some branches might be very hard to get past as the # of inputs statisfying the conditions are very small
  - Manually/automatically transform/remove those branches

Hard to fuzz code

```
void test (int n) {
  if (n==0x12345678)
      crash();
}
```

needs 2^32 or 4 billion attempts
In the worst case

Make it easier to fuzz

```
void test (int n) {
  int dummy = 0;
  char *p = (char *)&n;
  if (p[3]==0x12) dummy++;
  if (p[2]==0x34) dummy++;
  if (p[1]==0x56) dummy++;
  if (p[0]==0x78) dummy++;
  if (dummy==4)
    crash();
}
```

needs around 2^10 attempts

Fuzzing rules of thumb

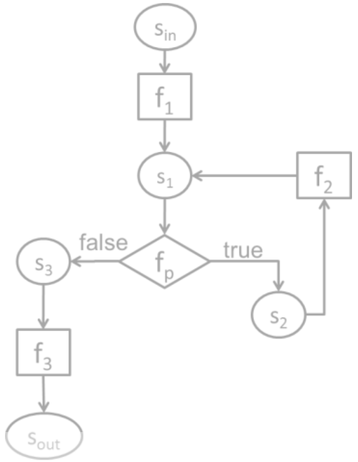
- Input-format knowledge is very helpful
- Generational tends to beat random, better specs make better fuzzers
- Each implementation will vary, different fuzzers find different bugs
  - More fuzzing with is better
- The longer you run, the more bugs you may find
  - But it reaches a plateau and saturates after a while
- Best results come from guiding the process
- Notice where you are getting stuck, use profiling (gcov, lcov)!

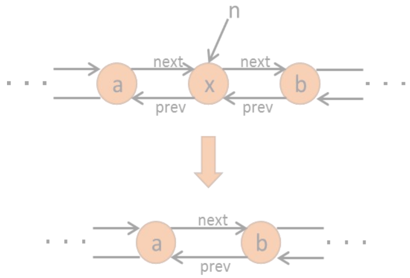
Program Synthesis

Slides adapted from https://github.com/nadia-polikarpova/cse291-program-synthesis

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

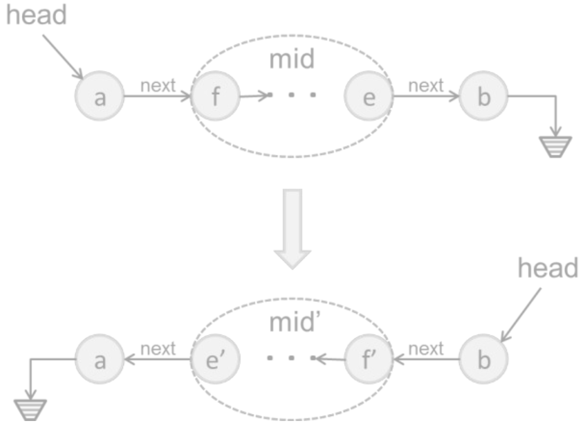
```
/* Average of x and y without using x+y (avoid overflow)*/
int avg(int x, int y){
  int t = expr({x/2, y/2, x%2, y%2, 2 }, {PLUS, DIV});
  assert t == (x+y)/2;
  return t;
}
```

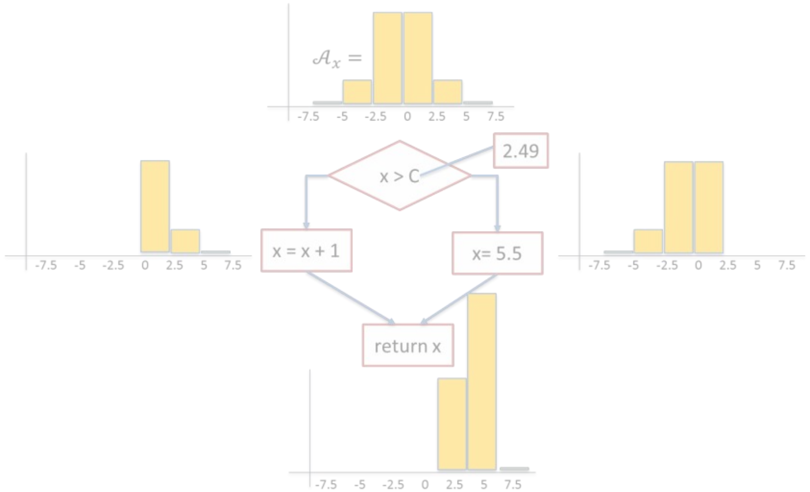





```
{
  s = n.succ;
  p = n.pred;
  p.succ = s;
  s.pred = p;
}
```

Program Synthesis



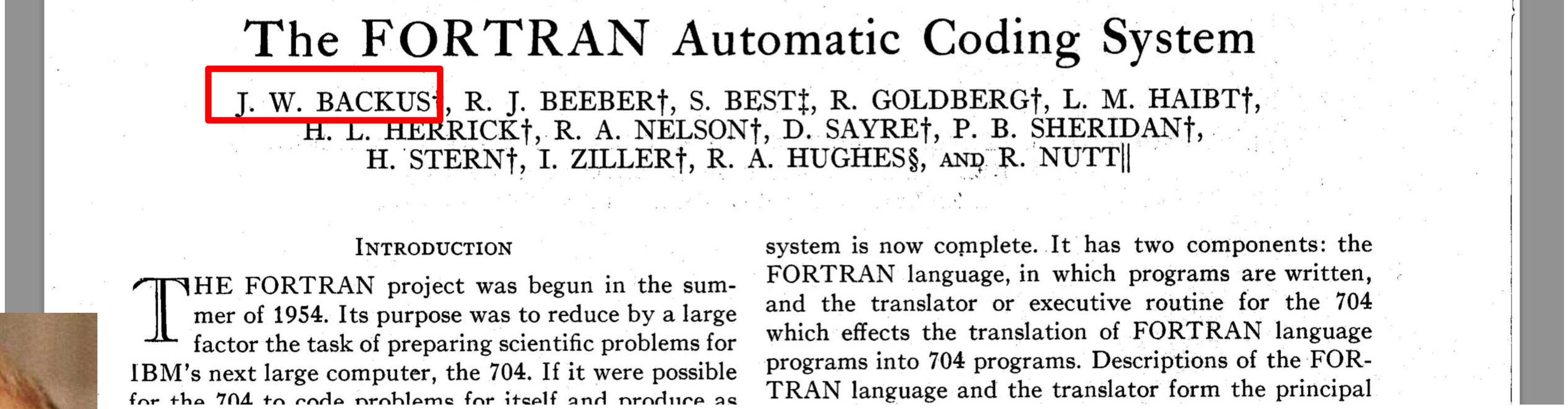


$\varphi(p)$

$Sk[c](in)$

History of Program Synthesis

Published on 1957



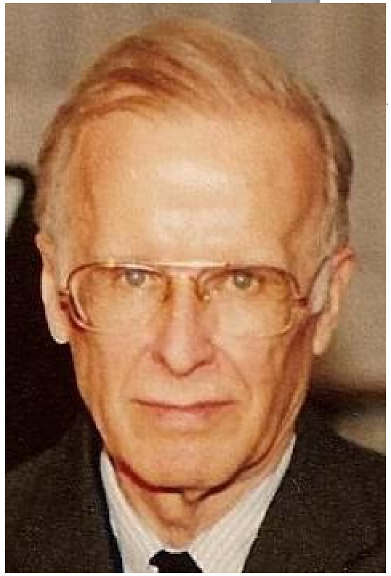
The FORTRAN Automatic Coding System

J. W. BACKUS†, R. J. BEEBER†, S. BEST‡, R. GOLDBERG†, L. M. HAIBT†, H. L. HERRICK†, R. A. NELSON†, D. SAYRE†, P. B. SHERIDAN†, H. STERN†, I. ZILLER†, R. A. HUGHES§, AND R. NUTT||

INTRODUCTION

THE FORTRAN project was begun in the summer of 1954. Its purpose was to reduce by a large factor the task of preparing scientific problems for IBM's next large computer, the 704. If it were possible for the 704 to code problems for itself and produce as

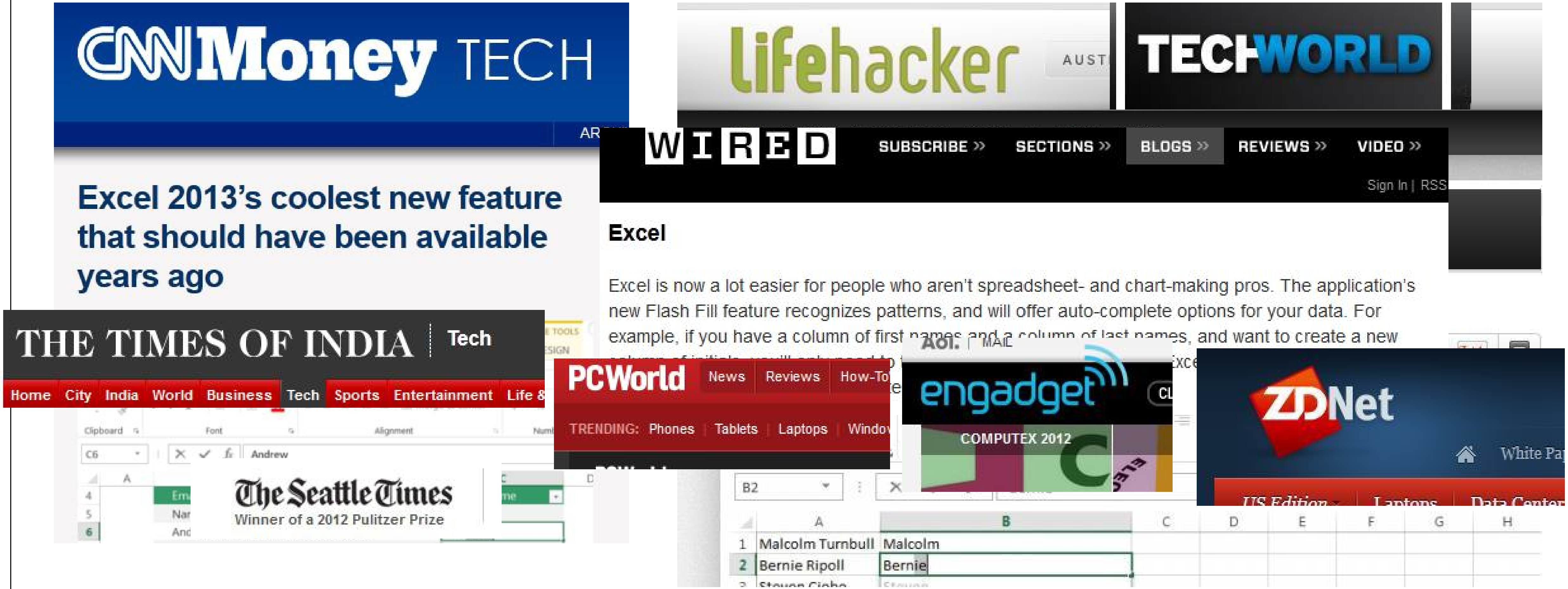
system is now complete. It has two components: the FORTRAN language, in which programs are written, and the translator or executive routine for the 704 which effects the translation of FORTRAN language programs into 704 programs. Descriptions of the FORTRAN language and the translator form the principal



John Backus

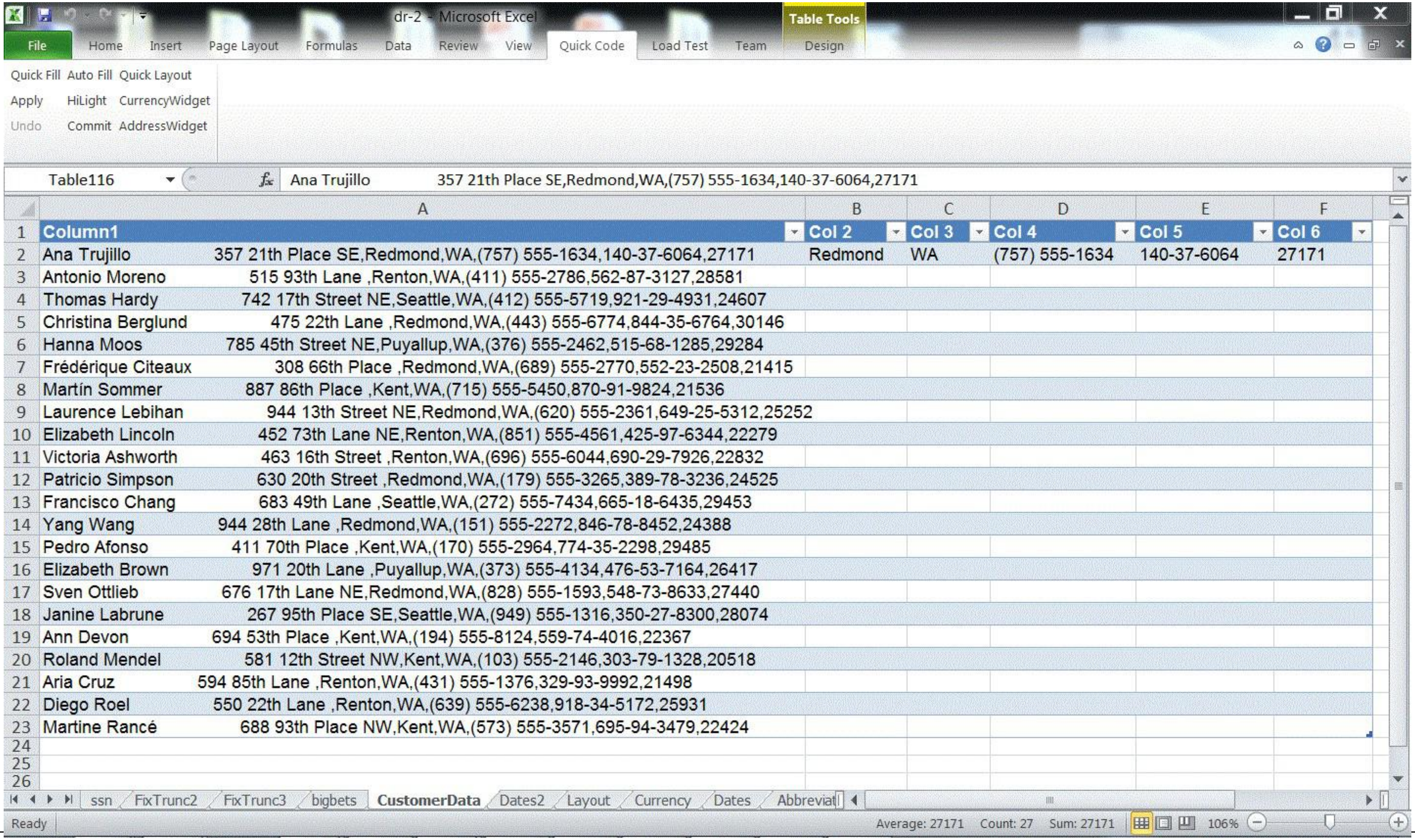
- Inventor of FORTRAN
- Inventor of Backus–Naur form (BNF)

Modern program synthesis: FlashFill



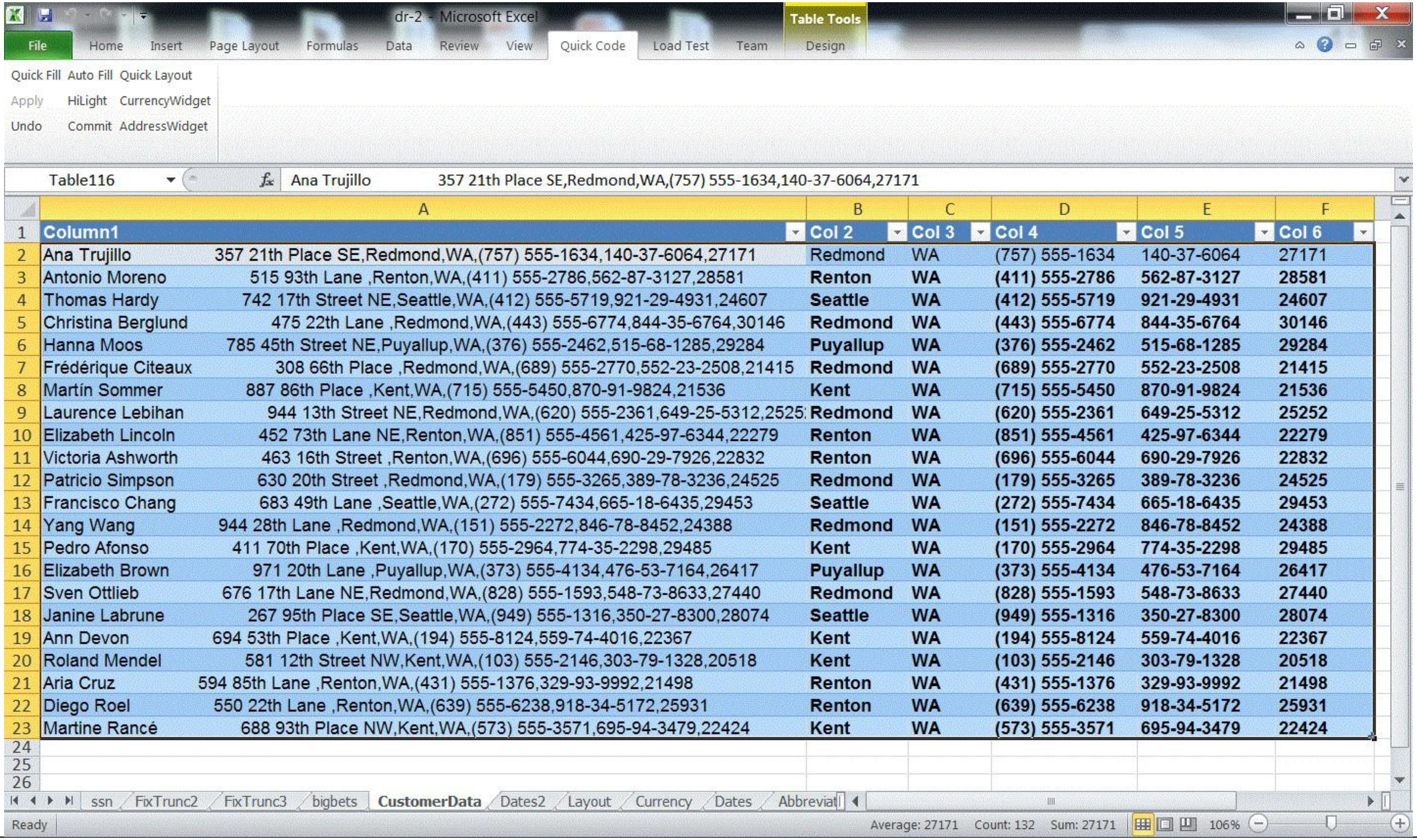
FlashFill: a feature of Excel 2013

[Gulwani 2011]



	A	B	C	D	E	F
1	Column1	Col 2	Col 3	Col 4	Col 5	Col 6
2	Ana Trujillo 357 21th Place SE,Redmond,WA,(757) 555-1634,140-37-6064,27171	Redmond	WA	(757) 555-1634	140-37-6064	27171
3	Antonio Moreno 515 93th Lane ,Renton,WA,(411) 555-2786,562-87-3127,28581					
4	Thomas Hardy 742 17th Street NE,Seattle,WA,(412) 555-5719,921-29-4931,24607					
5	Christina Berglund 475 22th Lane ,Redmond,WA,(443) 555-6774,844-35-6764,30146					
6	Hanna Moos 785 45th Street NE,Puyallup,WA,(376) 555-2462,515-68-1285,29284					
7	Frédérique Citeaux 308 66th Place ,Redmond,WA,(689) 555-2770,552-23-2508,21415					
8	Martín Sommer 887 86th Place ,Kent,WA,(715) 555-5450,870-91-9824,21536					
9	Laurence Lebihan 944 13th Street NE,Redmond,WA,(620) 555-2361,649-25-5312,25252					
10	Elizabeth Lincoln 452 73th Lane NE,Renton,WA,(851) 555-4561,425-97-6344,22279					
11	Victoria Ashworth 463 16th Street ,Renton,WA,(696) 555-6044,690-29-7926,22832					
12	Patricio Simpson 630 20th Street ,Redmond,WA,(179) 555-3265,389-78-3236,24525					
13	Francisco Chang 683 49th Lane ,Seattle,WA,(272) 555-7434,665-18-6435,29453					
14	Yang Wang 944 28th Lane ,Redmond,WA,(151) 555-2272,846-78-8452,24388					
15	Pedro Afonso 411 70th Place ,Kent,WA,(170) 555-2964,774-35-2298,29485					
16	Elizabeth Brown 971 20th Lane ,Puyallup,WA,(373) 555-4134,476-53-7164,26417					
17	Sven Ottlieb 676 17th Lane NE,Redmond,WA,(828) 555-1593,548-73-8633,27440					
18	Janine Labrune 267 95th Place SE,Seattle,WA,(949) 555-1316,350-27-8300,28074					
19	Ann Devon 694 53th Place ,Kent,WA,(194) 555-8124,559-74-4016,22367					
20	Roland Mendel 581 12th Street NW,Kent,WA,(103) 555-2146,303-79-1328,20518					
21	Aria Cruz 594 85th Lane ,Renton,WA,(431) 555-1376,329-93-9992,21498					
22	Diego Roel 550 22th Lane ,Renton,WA,(639) 555-6238,918-34-5172,25931					
23	Martine Rancé 688 93th Place NW,Kent,WA,(573) 555-3571,695-94-3479,22424					
24						
25						
26						

FlashFill: a feature of Excel 2013



	Column1 (A)	Col 2 (B)	Col 3 (C)	Col 4 (D)	Col 5 (E)	Col 6 (F)
2	Ana Trujillo 357 21th Place SE,Redmond,WA,(757) 555-1634,140-37-6064,27171	Redmond	WA	(757) 555-1634	140-37-6064	27171
3	Antonio Moreno 515 93th Lane ,Renton,WA,(411) 555-2786,562-87-3127,28581	Renton	WA	(411) 555-2786	562-87-3127	28581
4	Thomas Hardy 742 17th Street NE,Seattle,WA,(412) 555-5719,921-29-4931,24607	Seattle	WA	(412) 555-5719	921-29-4931	24607
5	Christina Berglund 475 22th Lane ,Redmond,WA,(443) 555-6774,844-35-6764,30146	Redmond	WA	(443) 555-6774	844-35-6764	30146
6	Hanna Moos 785 45th Street NE,Puyallup,WA,(376) 555-2462,515-68-1285,29284	Puyallup	WA	(376) 555-2462	515-68-1285	29284
7	Frédérique Citeaux 308 66th Place ,Redmond,WA,(689) 555-2770,552-23-2508,21415	Redmond	WA	(689) 555-2770	552-23-2508	21415
8	Martín Sommer 887 86th Place ,Kent,WA,(715) 555-5450,870-91-9824,21536	Kent	WA	(715) 555-5450	870-91-9824	21536
9	Laurence Lebihan 944 13th Street NE,Redmond,WA,(620) 555-2361,649-25-5312,2525	Redmond	WA	(620) 555-2361	649-25-5312	25252
10	Elizabeth Lincoln 452 73th Lane NE,Renton,WA,(851) 555-4561,425-97-6344,22279	Renton	WA	(851) 555-4561	425-97-6344	22279
11	Victoria Ashworth 463 16th Street ,Renton,WA,(696) 555-6044,690-29-7926,22832	Renton	WA	(696) 555-6044	690-29-7926	22832
12	Patricio Simpson 630 20th Street ,Redmond,WA,(179) 555-3265,389-78-3236,24525	Redmond	WA	(179) 555-3265	389-78-3236	24525
13	Francisco Chang 683 49th Lane ,Seattle,WA,(272) 555-7434,665-18-6435,29453	Seattle	WA	(272) 555-7434	665-18-6435	29453
14	Yang Wang 944 28th Lane ,Redmond,WA,(151) 555-2272,846-78-8452,24388	Redmond	WA	(151) 555-2272	846-78-8452	24388
15	Pedro Afonso 411 70th Place ,Kent,WA,(170) 555-2964,774-35-2298,29485	Kent	WA	(170) 555-2964	774-35-2298	29485
16	Elizabeth Brown 971 20th Lane ,Puyallup,WA,(373) 555-4134,476-53-7164,26417	Puyallup	WA	(373) 555-4134	476-53-7164	26417
17	Sven Ottlieb 676 17th Lane NE,Redmond,WA,(828) 555-1593,548-73-8633,27440	Redmond	WA	(828) 555-1593	548-73-8633	27440
18	Janine Labrune 267 95th Place SE,Seattle,WA,(949) 555-1316,350-27-8300,28074	Seattle	WA	(949) 555-1316	350-27-8300	28074
19	Ann Devon 694 53th Place ,Kent,WA,(194) 555-8124,559-74-4016,22367	Kent	WA	(194) 555-8124	559-74-4016	22367
20	Roland Mendel 581 12th Street NW,Kent,WA,(103) 555-2146,303-79-1328,20518	Kent	WA	(103) 555-2146	303-79-1328	20518
21	Aria Cruz 594 85th Lane ,Renton,WA,(431) 555-1376,329-93-9992,21498	Renton	WA	(431) 555-1376	329-93-9992	21498
22	Diego Roel 550 22th Lane ,Renton,WA,(639) 555-6238,918-34-5172,25931	Renton	WA	(639) 555-6238	918-34-5172	25931
23	Martine Rancé 688 93th Place NW,Kent,WA,(573) 555-3571,695-94-3479,22424	Kent	WA	(573) 555-3571	695-94-3479	22424

Modern program synthesis: Sketch

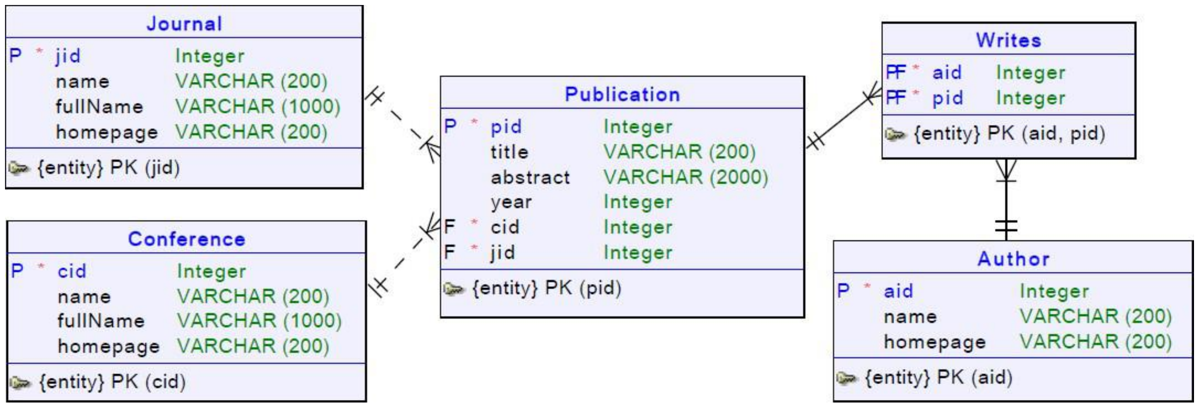
[Solar-Lezama 2013]

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

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

Modern program synthesis: SQLizer

[Yaghmazadeh et al. 2017]

Problem: “Find the number of papers in OOPSLA2010”



Output:

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

What is program synthesis?

Automatic programming?

- but I 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

Synthesis

Goal: Synthesize a computational concept in some underlying language from user intent using some search technique.

State of the art: We can synthesize programs of size 10-20.

Dimensions in Synthesis

- Language (Application)
  - Programs
    - Straight-line programs
  - Automata
  - Queries
- User Intent (Ambiguity)
  - Logic, Natural Language
  - Examples, Demonstrations/Traces
  - Program
- Search Technique (Algorithm)
  - SAT/SMT solvers (Formal Methods)
  - A*-style goal-directed search (AI)
  - Version space algebras (Machine Learning)

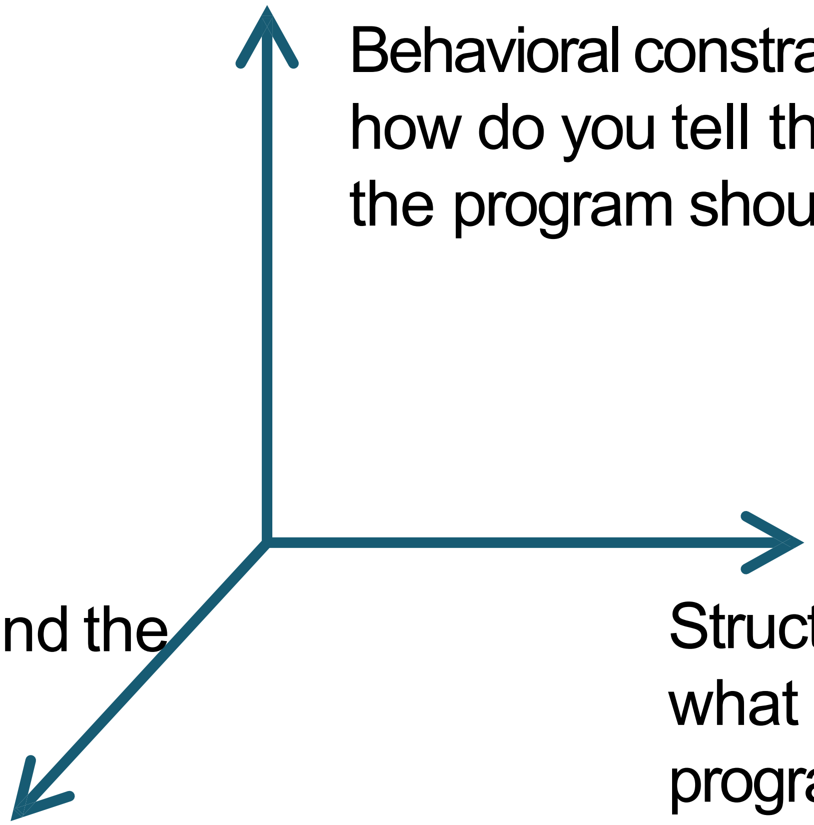
PPDP 2010: “Dimensions in Program Synthesis”, Gulwani.

Compilers vs. Synthesizers

Dimension	Compilers	Synthesizers
Concept Language	Executable Program	Variety of concepts: Program, Automata, Query, Sequence
User Intent	Structured language	Variety/mixed form of constraints: logic, examples, traces
Search Technique	Syntax-directed translation (No new algorithmic insights)	Uses some kind of search (Discovers new algorithmic insights)

Dimensions in program synthesis

[Gulwani 2010]

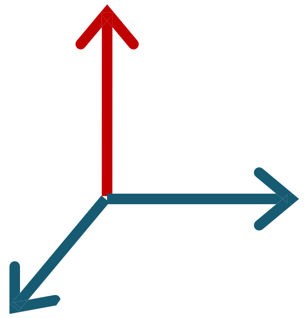


Behavioral constraints:
how do you tell the system what the program should do?

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

Structural constraints:
what is the space of programs to explore?

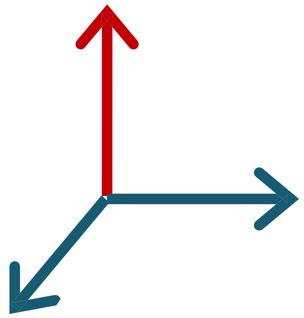
Behavioral constraints



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?

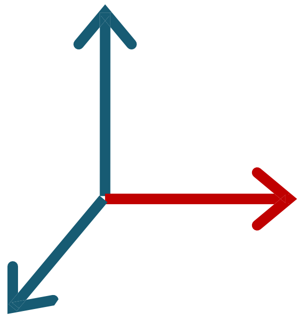
Behavioral constraints: examples



Input/output examples:

- Equivalent program
- Formal specifications (pre/post conditions, types, ...)
- Natural language

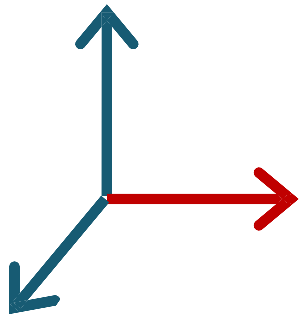
Structural constraints



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?

Structural constraints: examples



Built-in DSL (Domain-Specific Language)

User-defined DSL (grammar)

- + statistical models

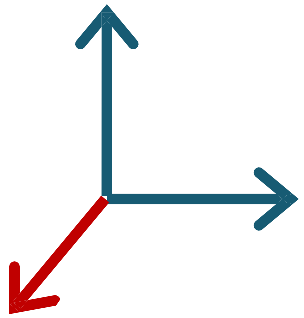
User-provided components

- within straight-line code
- within recursive functional programs

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*

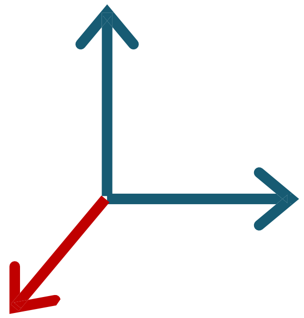
Challenge: the space is astronomically large

- 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

Programming by Examples

User Intent = Examples

Synthesis from Examples

=

Programming by Example

=

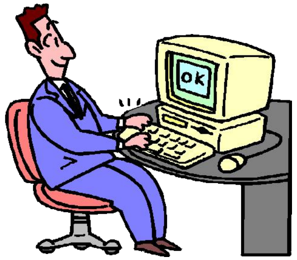
Inductive Synthesis
(Inductive Learning)

Programming by Example: Motivation

Algorithm Designers

Software Developers

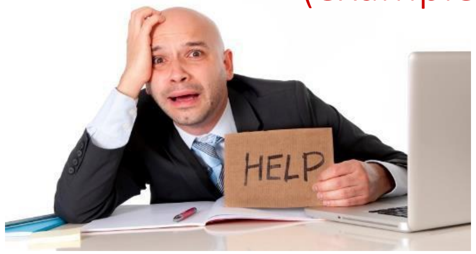
End-Users



(logics, automata, etc.)

(code)

(examples!)



The Zendo Game

Let's play a game to learn about how programming by example(PBE) works!

The modified Zendo (禪道) game: Rules

Original Zendo Game

1. The teacher keeps a secret rule about arrangements of 0s and 1s
   - e.g. has odd number of zeros

2. The teacher builds two koans (公案) (a positive & a negative)
   Examples:
   - 00111 (Follows the hidden rule)
   - 1001 (Does not follow the hidden rule)

3. Students take turns to build koans and ask the teacher to label them
4. A student can try to guess the rule
   - if they are right, they win
   - otherwise, the teacher builds a koan on which the two rules disagree

Trial 1

Hidden Rule

Follows the rule	Does not follow the rule
00011	01

Give me more公案 and guess if it follows the rules

Joint down what do you think the rule is.

Trial 2

Hidden Rule

Follows the rule	Does not follow the rule
10011	010

Give me more公案 and guess if it follows the rules

Joint down what do you think the rule is.

Zendo: Inductive Learning

- Generalizability and Biased Samples:
  - Starting with only 2 data points and extrapolating from that information can be dangerous and lead us astray. As we add data, a biased perspective can lead to confirmation bias problems, rather than a genuine test of our hypotheses.
- General ideas of generating and testing hypotheses
  - Same in the research process where student generates a hypothesis and test the hypotheses through experiments

A little bit of history: inductive learning

MIT/LCS/TR-76

LEARNING STRUCTURAL DESCRIPTIONS FROM EXAMPLES

Patrick H. Winston

September 1970

Patrick Winston

Explored the question of generalizing from a set of observations

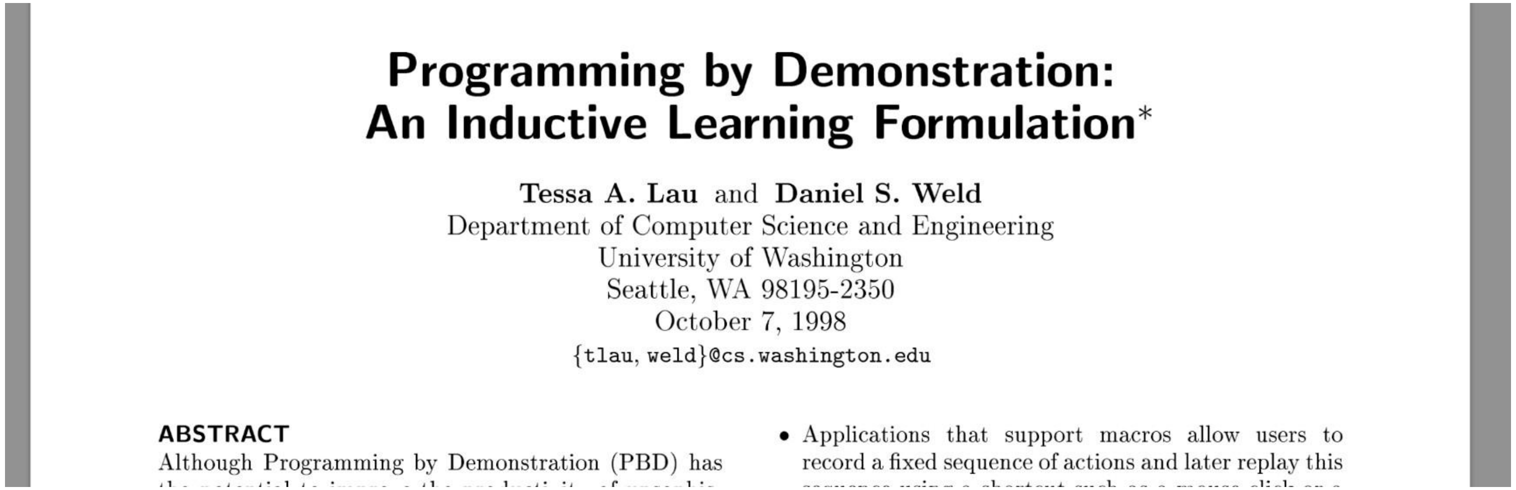
- Similar to Zendo

Became the foundation of machine learning

A little bit of history: PBE/PBD

Early systems searched a predefined list of programs

Tessa Lau: bring inductive learning techniques into PBE



Programming by Demonstration: An Inductive Learning Formulation*

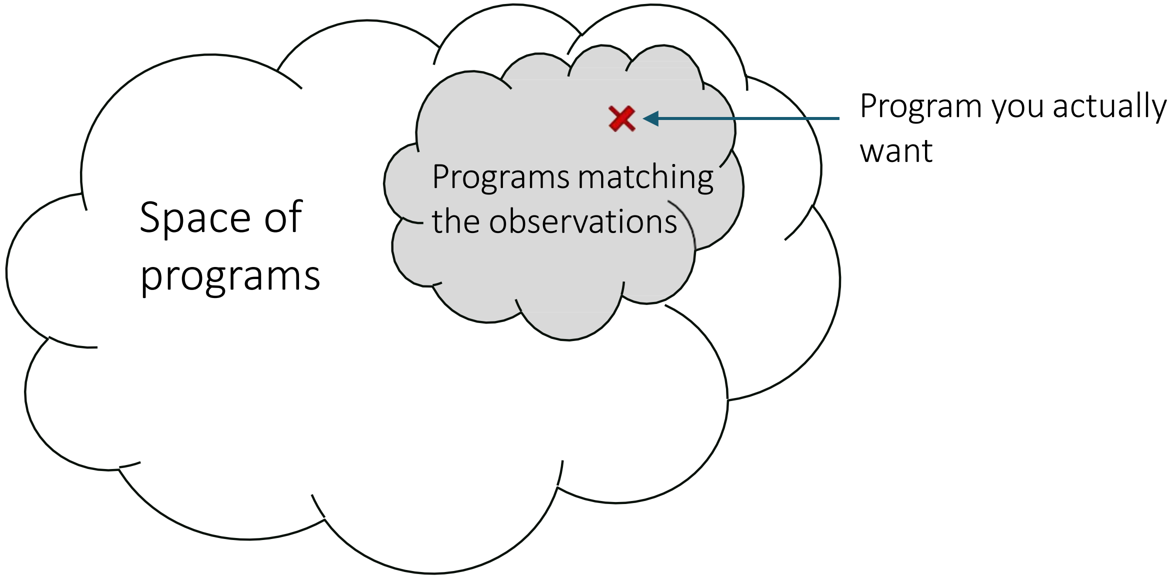
Tessa A. Lau and **Daniel S. Weld**
Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195-2350
October 7, 1998
{tlau, weld}@cs.washington.edu

ABSTRACT
Although Programming by Demonstration (PBD) has

- Applications that support macros allow users to record a fixed sequence of actions and later replay this

Tessa Lau

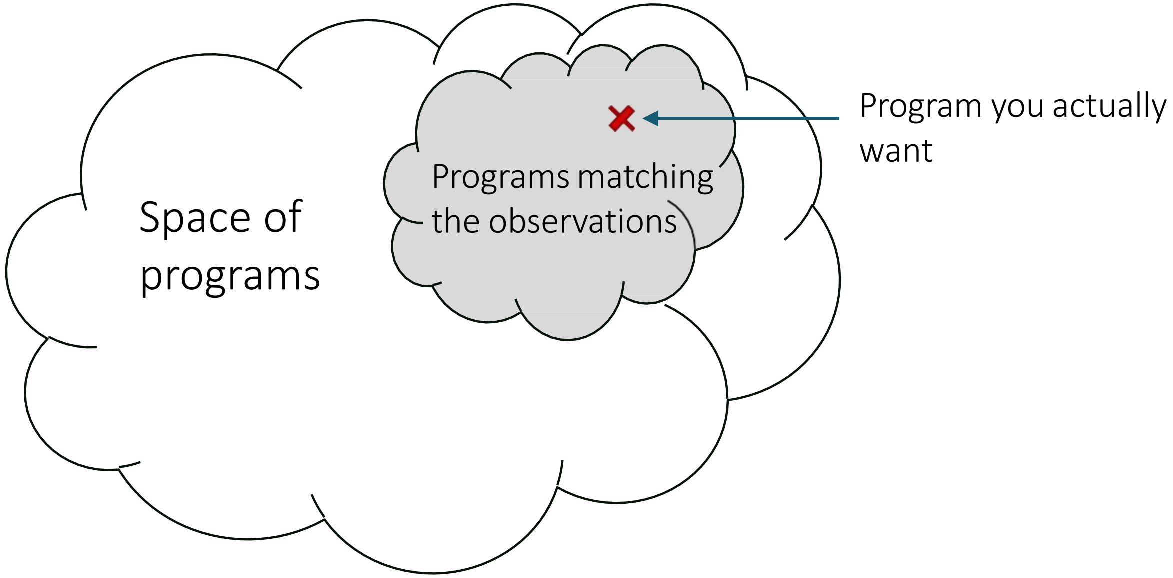
Key issues in inductive learning



(1) How do you find a program that matches the observations?

(2) How do you know it is the program you are looking for?

Key issues in inductive learning



Traditional ML emphasizes (2)

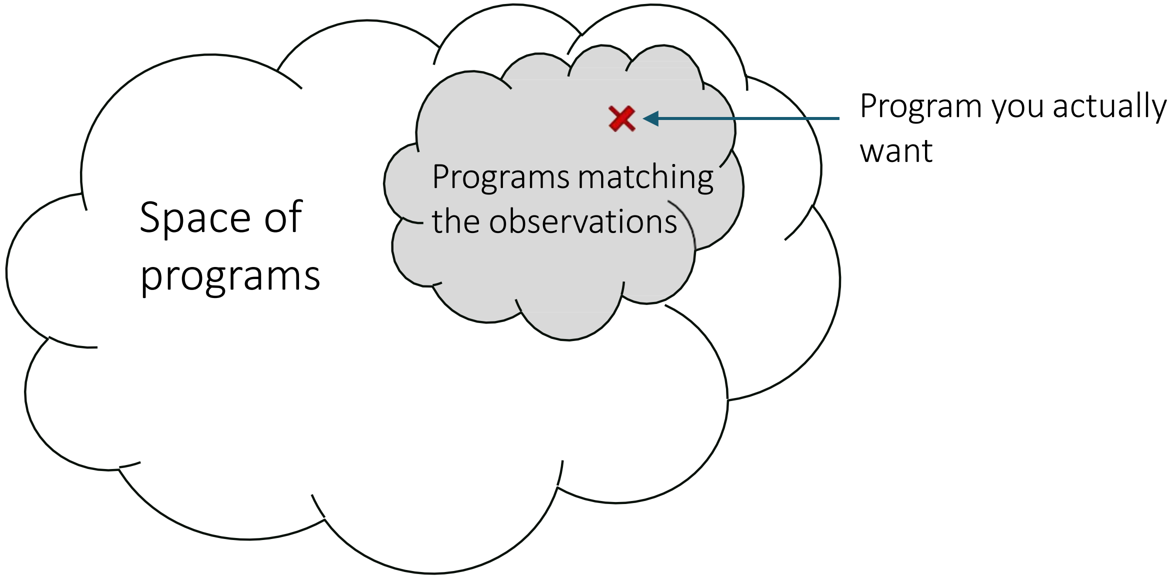
- Fix the space so that (1) is easy

So did a lot of PBD work

(1) How do you find a program that matches the observations?

(2) How do you know it is the program you are looking for?

The synthesis approach

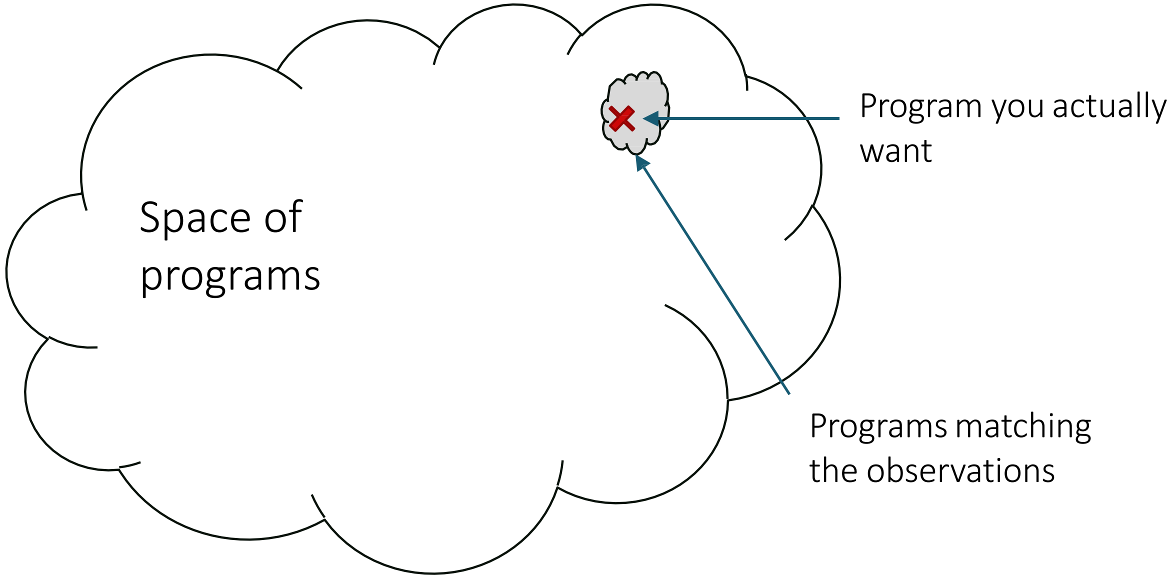


Modern emphasis

(1) How do you find a program that matches the observations?

(2) How do you know it is the program you are looking for?

The synthesis approach



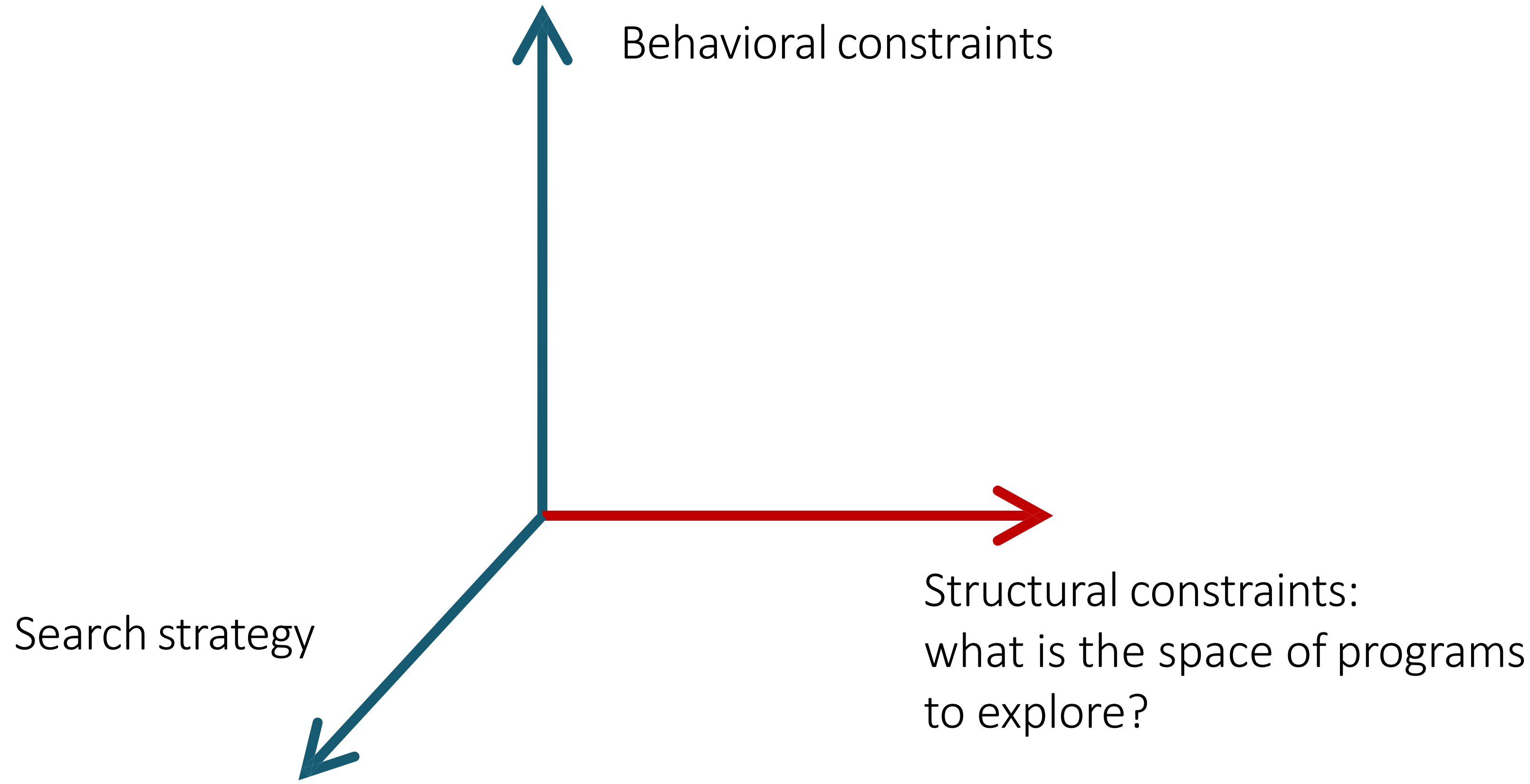
Modern emphasis

- If you can do really well with (1) you can win
- (2) is still important

(1) How do you find a program that matches the observations?

(2) How do you know it is the program you are looking for?

Dimensions in program synthesis



Syntax-Guided Synthesis

Example

[1,4,7,2,0,6,9,2,5,0] → [1,2,4,7,0]

```
f(x) := sort(x[0..find(x, 0)]) + [0]
```

```
L ::= sort(L)     |
      L[N..N]     |
      L + L       |
      [N]         |
      x
N ::= find(L,N)  |
         0
```

Context-free grammars (CFGs)

starting nonterminal

terminals

nonterminals

```
L ::= sort(L)    |
      L[N..N]    |
      L + L      |
      [N]        |
      x
N ::= find(L,N)  |
      0
```

productions

Context-free grammars (CFGs)

nonterminals rules (productions)

terminals starting nonterminal

`<T, N, R, S>`

Sentential forms: $\{\alpha \in (N \cup T)^*\}$	`x N x`
Rewrites to: $\alpha A \beta \to \alpha \gamma \beta \;\equiv\; (A \to \gamma) \in R$	`x N x -> x find(L,N) x`
(Incomplete) terms/programs: $\{\alpha \in (N \cup T)^* \vert A \to^* \alpha\}$	`find(L+L,N)`
Ground programs: $\{\alpha \in T^* \vert A \to^* \alpha\}$ = programs without holes, complete programs	`find(x+x,0)`
Whole programs: $\{\alpha \in (N \cup T)^* \vert S \to^* \alpha\}$ = roughly, programs of the right type	`sort(L+L)`

CFGs as structural constraints

Space of programs
=
all ground, whole programs

```
L ::= sort(L)      |
      L[N..N]      |
      L + L        |
      [N]          |
      x
N ::= find(L,N)    |
        0
```

```
x    sort(x)    x + x    x[0..0]
...
x[0..find(x, 0)]
...
sort(x[0..find(x, 0)]) + [0]
...
```

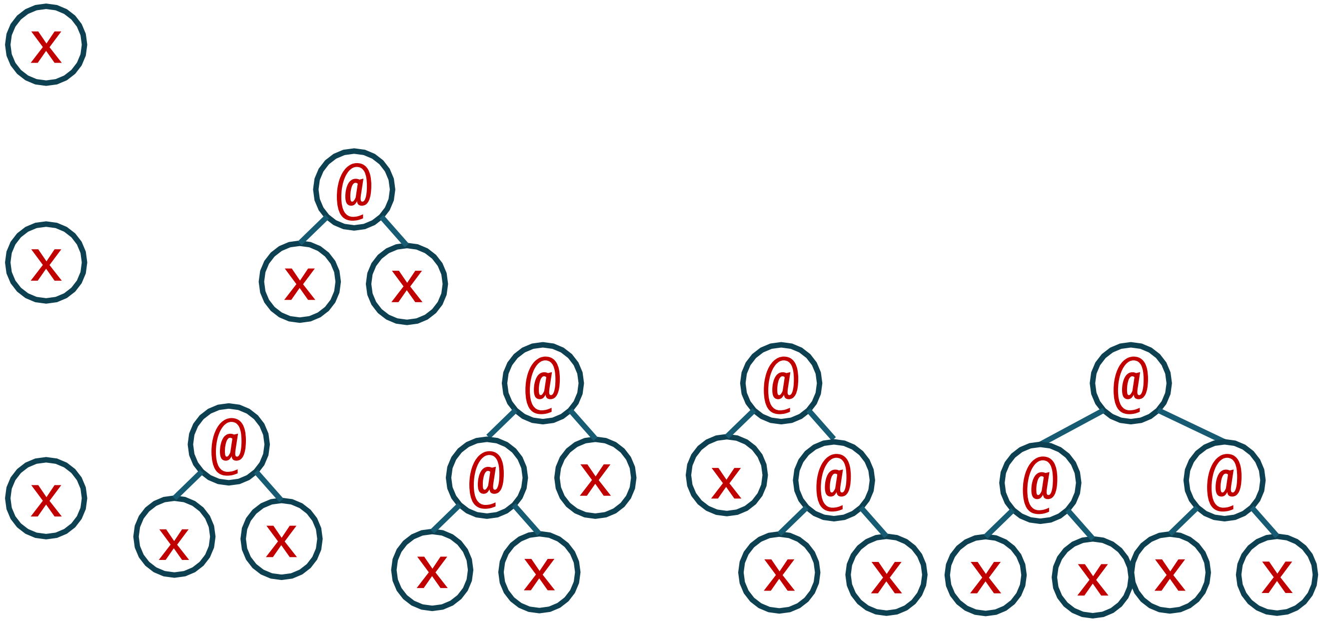
How big is the space?

`E ::= x | E @ E`

depth <= 1 — $N(1) = 1$

depth <= 2 — $N(2) = 2$

depth <= 3 — $N(3) = 5$



$$N(d) = 1 + N(d-1)^2$$

How big is the space?

`E ::= x | E @ E`

$N(d) = 1 + N(d - 1)^2$ $\qquad N(d) \sim c^{2^d}$ $\qquad (c > 1)$

N(1) = 1
N(2) = 2
N(3) = 5
N(4) = 26
N(5) = 677
N(6) = 458330
N(7) = 210066388901
N(8) = 44127887745906175987802
N(9) = 1947270476915296449559703445493848930452791205
N(10) = 3791862310265926082868235028027893277370233152247388584761734150717768254410341175325352026

How big is the space?

```
E ::=      x₁ | ... | xₖ |
      E @₁ E | ... | E @ₘ E
```

$$N(0) = 0$$

$$N(d) = k + m * N(d - 1)^2$$

N(1) = 3
N(2) = 30
N(3) = 2703
N(4) = 21918630
N(5) = 1441279023230703
N(6) = 6231855668414547953818685622630
N(7) = 116508075215851596766492219468227024724121520304443212304350703

k = m = 3

CFGs as structural constraints

Pros:

- Clean declarative description
- Easy to sample
- Easy to explore exhaustively

Cons:

- Insufficiently expressive

What if we know the following:

- Sort can be called at most once
- Sub-list is never called on a concatenation of singletons
- In a call to sub-list, the start index is <= the end index

Grammars vs generators

- Grammars
- Pros:
  - Clean declarative description
  - Easy to sample
  - Easy to explore exhaustively
- Cons:
  - Insufficiently expressive

Generators

- Programs that produce programs

Pros:

- Extremely general
  - easy to enforce arbitrary constraints

Cons:

- Extremely general
  - Hard to analyze and reason about
  - Hard to automatically discover structure of the space

The SyGuS project

[Alur et al. 2013]

SyGuS problem = < theory, spec, grammar >

A “library” of types and function symbols

Example: Linear Integer Arithmetic (LIA)

```
True, False
0,1,2,...
∧, ∨, ¬, +, ≤, ite
```

CFG with terminals in the theory (+ input variables)

Example: Conditional LIA expressions w/o sums

```
E ::= x | ite C E E
C ::= E ≤ E | C ∧ C | ¬C
```

The SyGuS project

SyGuS problem = < theory, spec, grammar >

A first-order logic formula over the theory

Examples:

```
f(0, 1) = 1 ∧
f(1, 0) = 1 ∧
f(1, 1) = 1 ∧
f(2, 0) = 2
```

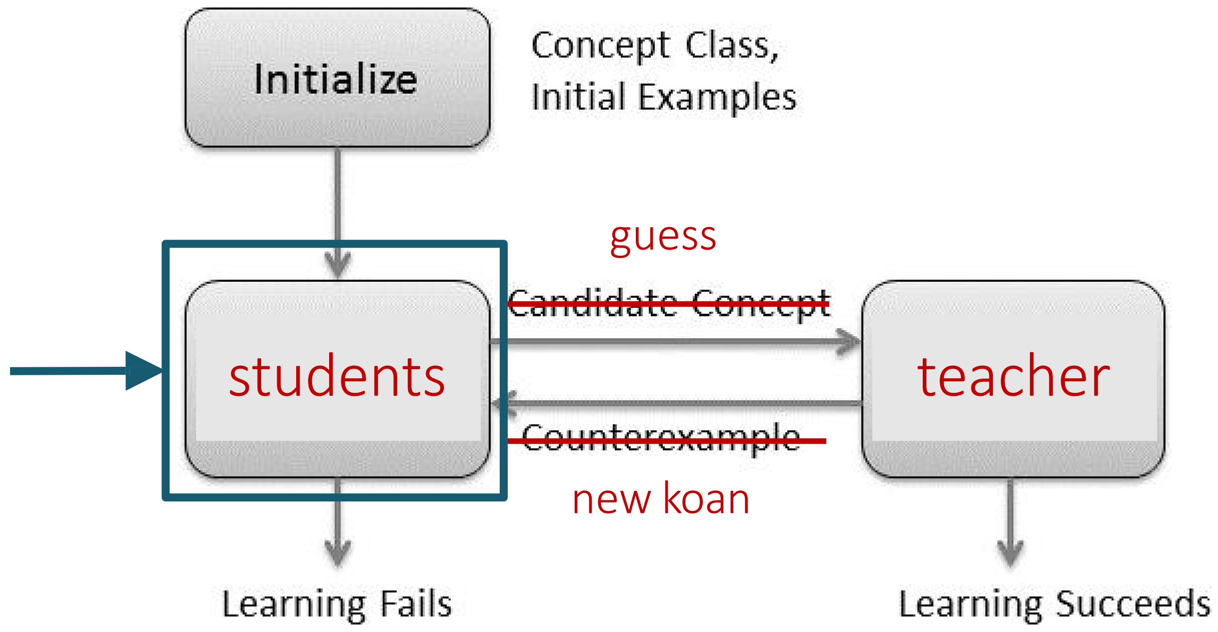
Formula with free variables:

```
x.≤ f(x, y) ∧
y.≤ f(x, y) ∧
(f(x, y) = x ∨ f(x, y) = y)
```

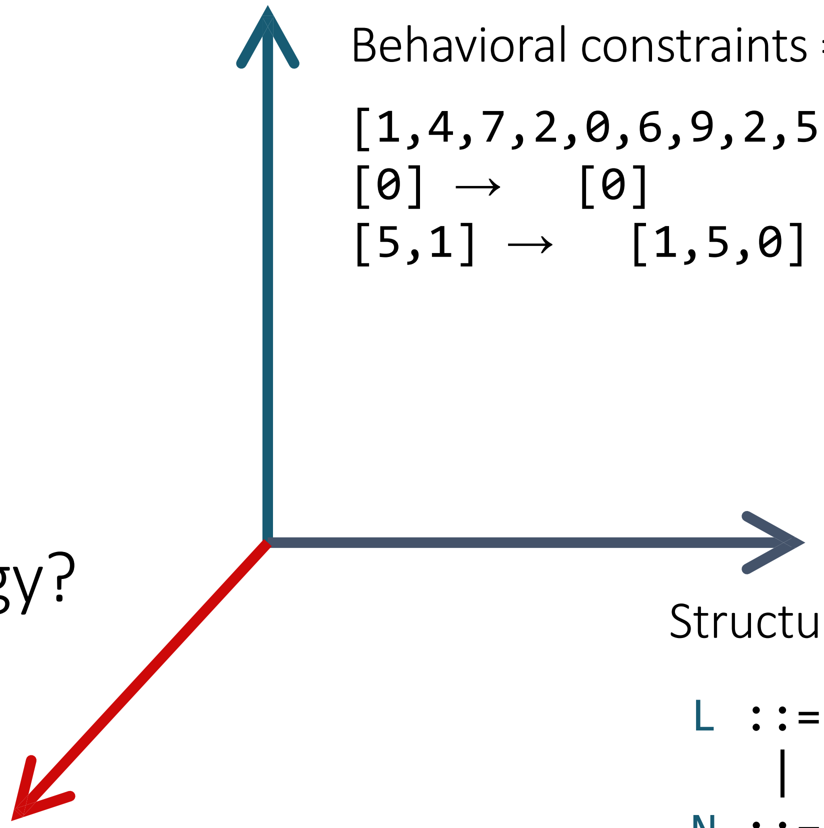
Counter-example guided inductive synthesis

The Zendo of program synthesis

Our focus in this module



The problem statement



Behavioral constraints = examples

```
[1,4,7,2,0,6,9,2,5]  →  [1,2,4,7,0]
[0] →  [0]
[5,1] →  [1,5,0]
```

Search strategy?

Structural constraints = grammar

```
L ::= sort(L)  |  L[N..N]
    |  L + L  |  [N]  |  x
N ::= find(L,N)  |  0
```

Enumerative Search

Enumerative search

- =
- Explicit / Exhaustive Search (穷竭搜索/暴力搜索)

- Idea: Sample programs from the grammar one by one and test them on the examples
  - order?

bottom-up

top-down

Bottom-up enumeration

Start from terminals

Combine sub-programs into larger programs using productions

Q: “Run” bottom-up on the board with

```
L ::= sort(L)     |
      L[N..N]     |
      L + L       |
      [N]         |
      x
N ::= find(L,N)   |
      0

[[1,4,0,6] → [1,4]]
```

terminals

nonterminals

rules (productions)

starting nonterminal

```
bottom-up (<T, N, R, S>, [i → o]) { P := [t | t in T && t is nullary] while (true)
   P += grow(P);
   forall (p in P)
      if (whole(p) && p([i]) = [o])
         return p;
}
grow (P) {
P’ := []
forall (A ::= rhs in R)
   P’ += [rhs[B -> p] | p in P]
return P’;
}
```

Top-down enumeration

Start from the start non-terminal

Expand remaining non-terminals using productions

Q: “Run” top-down on the board with

```
    L ::= L[N..N]  |
          x
N ::= find(L,N)  |
          0

[[1,4,0,6]  →  [1,4]]
```

```
top-down(<T, N, R, S>, [i →  o]){
  P := [S]
  while (P != [])
    p := P.dequeue();
    if (ground(p) && p([i]) = [o])
      return p;  P.enqueue(unroll(p));
}

unroll(p) {
  P’ := []
  forall (A in p)
    forall (A ::= rhs in R)
      P’ += p[A -> rhs]
  return P’;
}
```

Bottom-up vs top-down

Bottom-up

Top-down

Smaller to larger

- Has to explore between $3*10^9$ and 10^{23} programs to find `sort(x[0..find(x, 0)]) + [0]` (depth 6)

Candidates are ground but might not be whole

- Can always run on inputs
- Cannot always relate to outputs

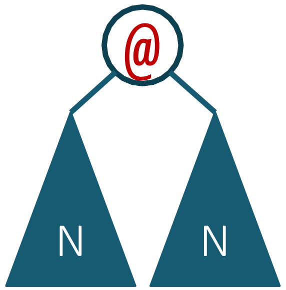
Candidates are whole but might not be ground

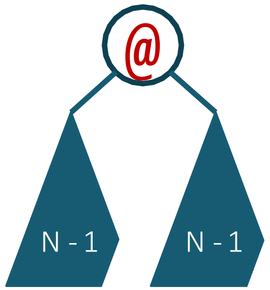
- Cannot always run on inputs
- Can always relate to outputs (?)

How to make it scale

Prune

Discard useless subprograms





$m * N^2$ $m * (N - 1)^2$

Prioritize

Explore more promising candidates first

```
P = { [0][N..N] ,
      x[N..N]   ,
      ... }
```

dequeue this first

Programming by Sketching

Slides from:

Armando Solar-Lezama, Liviu Tancau, Gilad Arnold,
Rastislav Bodik, Sanjit Seshia UC Berkeley, Rodric Rabbah MIT,
Kemal Ebcioglu, Vijay Saraswat, Vivek Sarkar IBM

Merge sort

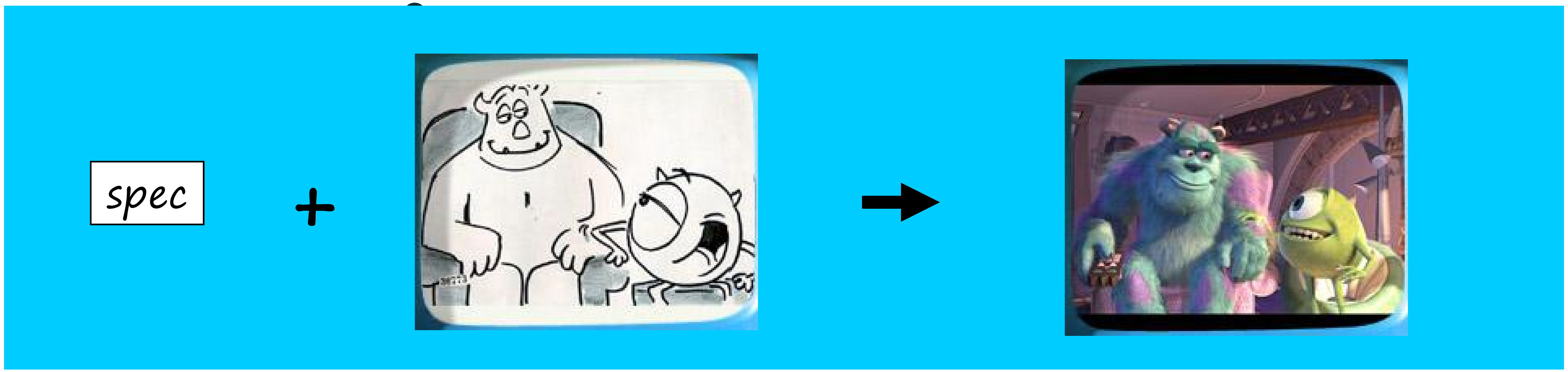
```
int[] mergeSort (int[] input, int n) {

                                                              , n);

int[] merge (int[] a, int b[], int n) {
        int j=0, k=0;
        for (int i = 0; i < n; i++)
                if ( a[j] < b[k] ) {
                        result[i] = a[j++];
                } else  {
                        result[i] = b[k++];
                }
        }
        return result;
}
```

looks simple to code, but there is a bug

Merge sort

```
int[] mergeSort (int[] input, int n) {
        return merge(mergeSort (input[0::n/2]),
                        mergeSort (input[n/2+1::n])    , n);
}
int[] merge (int[] a, int b[], int n) {
        int j, k;
        for (int i = 0; i < n; i++)
                if ( j<n && ( !(k<n) || a[j] < b[k]) ) {
                        result[i] = a[j++];
                } else  {
                        result[i] = b[k++];
                }
        }
        return result;
}
```

The sketching



specification

sketch

implementation
(completed sketch)

The spec: bubble sort

```
int[] sort (int[] input, int n) {
    for (int i=0; i<n; ++i)
        for (int j=i+1; j<n; ++j)
            if (input[j] < input[i])
                swap(input, j, i);
}
```

Merge sort: sketched

```
int[] mergeSort (int[] input, int n) {
        return merge(mergeSort (input[0::n/2]),
                              mergeSort (input[n/2+1::n])  , n);
}
int[] merge (int[] a, int b[], int n) {
        int j, k;
        for (int i = 0; i < n; i++)
                if (          hole          ) {
                        result[i] = a[j++];
                } else  {
                        result[i] = b[k++];
                }
        }
        return result;
}
```

Merge sort: synthesized

```
int[] mergeSort (int[] input, int n) {
        return merge(mergeSort (input[0::n/2]),
                        mergeSort (input[n/2::n])     );
}
int[] merge (int[] a, int b[], int n) {
        int j, k;
        for (int i = 0; i < n; i++)
                if ( j<n && ( !(k<n) || a[j] < b[k]) ) {
                        result[i] = a[j++];
                } else  {
                        result[i] = b[k++];
                }
        }
        return result;
}
```

Sketching: spec vs. sketch

Specification

executable: easy to debug, serves as a prototype
a reference implementation: simple and sequential
written by domain experts: crypto, bio, MPEG committee

Sketched implementation

program with holes: filled in by synthesizer
programmer sketches strategy: machine provides details
written by performance experts: vector wizard; cache guru

Ex1: Isolate rightmost 0-bit. 1010 0111 → 0000 1000

```
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; break; }
return ret;
}

bit[W] isolate0Fast (bit[W] x) implements isolate0 {
          return ~x & (x+1);
}

bit[W] isolate0Sketched (bit[W] x) implements isolate0 {
          return ~(x + ??) & (x + ??);
}
```

Programmer's view of sketches

the **??** operator replaced with a suitable constant as directed by the **implements** clause.

the **??** operator introduces non-determinism the **implements** clause constrains it.

Beyond synthesis of literals

Synthesizing values of ?? already very useful

parallelization machinery: bitmasks, tables in crypto codes
array indices: A[i+??,j+??]

We can synthesize more than constants

semi-permutations: functions that select and shuffle bits
polynomials: over one or more variables
actually, arbitrary expressions, programs

Summary

- Synthesis Techniques
  - Programming by Example
  - Syntax-guided Synthesis
  - Counter-example guided inductive synthesis
  - Programming by Sketching

References

- https://github.com/nadia-polikarpova/cse291-program-synthesis
- http://activelearningps.com/2016/02/24/zendo-revisited-a-simple-methods-game-for-large-classes/