program synthesis

Nadia Polikarpova
goal: automate programming
program synthesis

specification → program
program synthesis

specification ➔ search ➔ program

program space
program synthesis

specification → search → program

program space → program
program synthesis

specification

search

program

program space
program synthesis

specification
- examples
demonstrations
- programs
- logic
- types
- natural language
...

search

program space

program
program synthesis

specification → search

VSA

enumerative

stochastic

constraint-based

...

program
this talk

specifications

search strategies
this talk

specifications

search strategies
what makes a good spec?

1. human-friendly
2. informative
3. synthesizer-friendly
what makes a good spec?

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. types
4. natural language

search strategies
this talk

specifications

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

search strategies
this talk

specifications

1. examples (PBE)
2. programs
3. types
4. natural language
Excel 2013’s coolest new feature that should have been available years ago

Excel

Excel is designed for people who aren’t spreadsheet- and chart-making pros. The application’s new feature recognizes patterns, and will offer auto-complete options for your data. For example, if you have a column of first names and a column of last names, and want to create a new column that concatenates the two, FlashFill will recognize you want to create a full name and auto-complete the formula for you. And you don’t even have to click on the formula bar to access this feature—just hover over a cell and the FlashFill field will appear as a drop-down menu. And it’s available in Excel 2013 and above.

[FlashFill Demo]

[Image of Excel interface]

[Image of FlashFill in action]
pbe / pbd: discussion

Domains:
- text transformations: FlashFill, FlashExtract
- web scraping: WebRelate, Rousillon
- data science: Morpheus, Wrex
- programmer’s assistant: FrAngel, Snippy
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. types
4. natural language


**Problem:** isolate the least significant zero bit in a word

\[ \text{Sketch: } 0010 \ 0101 \rightarrow 0000 \ 0010 \]
Sketch

Problem: isolate the least significant zero bit in a word
Easy to implement with a loop

```c
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; }
}
```
Problem: isolate the least significant zero bit in a word
Easy to implement with a loop

```c
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?
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: 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 ~gen(x, depth-1);
    if(??){
        return { | expr(x, depth-1) (+ | & | ^) expr(x, depth-1) |};
    }
}
Sketch: synthesis goal

```cpp
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 ~gen(x, depth-1);
    if(??){
        return { | expr(x, depth-1) (+ | & | ^) expr(x, depth-1) |};
    }
}
```
Sketch: output

```c
bit[W] isolate0fast (bit[W] x) {
    return (~x) & (x + 1);
}
```
reference programs: discussion

Domains:
• superoptimization: Stoke, Lens
• verified lifting: QBS, STNG, other Alvin Cheung’s work
reference programs: discussion

Domains:
- superoptimization: Stoke, Lens
- verified lifting: QBS, STNG, other 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. types
4. natural language
Problem: remove adjacent duplicates from a list using library functions

\[ [1, 1, 1, 2, 2, 1] \rightarrow [1, 2, 1] \]
inspiration: Hooble

(a -> b) -> [a] -> [b]
map :: (a -> b) -> [a] -> [b]

Data.List

map f xs is the list obtained by applying f to each element of xs, i.e.,

map f [x1, x2, ..., xn] == [f x1, f x2, ..., f xn]
Hoogle needs synthesis!

[a] -> [a]
Hoogle needs synthesis!

Eq \( a \Rightarrow [a] \rightarrow [a] \)

Search
The `nub` function removes duplicate elements from a list.

The `tail` function extracts elements after the head of the list.
**Hoogle needs synthesis!**

**nub :: Eq a => [a] -> [a]**

*Data.List*

The nub function removes duplicate elements from a list.

**tail :: [a] -> [a]**

*Prelude*

Extract elements after the head of the list.
Hoogle+

specification

\[ \text{Eq } a \Rightarrow [a] \rightarrow [a] \]

->

\[ \text{H}^+ \]

->

 programs

Haskell libraries

1. ___
2. ___
3. ___
4. ___
Hoogle+

specification

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

demo!

Haskell libraries

programs

1. ___
2. ___
3. ___
4. ___
Problem: duplicate every element in a list using recursion

\[ [1, 2, 3] \rightarrow [1, 1, 2, 2, 3, 3] \]
type-driven synthesis

specification: $[a] \rightarrow [a]$

search

program space

program
type-driven synthesis

specification

search

program

ambiguous!
type-driven synthesis

specification \[\Rightarrow\] search \[\Rightarrow\] program

ambiguous!

POWER TO THE TYPES!
refinement types

Int
refinement types

\{ v:\text{Int} \mid 0 \leq v \}
Synquid

specification  refinement  types  program

components
Synquid

specification

refinement types

demo!

components

program
types: discussion

Domains:
• API discovery: Sypet, Hoogle+
• verified functional programs: Synquid
types: discussion

Domains:
- API discovery: Sypet, Hoogle+
- verified functional programs: Synquid

+ programmer friendly
  + concise
  + automatic checking
  + can express non-functional properties
  + guide search

- simple types are ambiguous
- advanced types require expertise
- some things are hard to express / check
this talk

specifications

1. examples (PBE)
2. programs
3. types
4. natural language
“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.”
“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.”

Positive examples:
123456789.123
12345.1

Negative examples:
1234567891234567
123.1234
“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.”

```plaintext
Concat(
  RepeatRange(<num>,1,15),
  Optional(Concat(
    <.>,
    RepeatRange(<num>,1,3)))
```

Positive examples:

123456789.123
12345.1

Negative examples:

1234567891234567
123.1234
123.1234
natural language: discussion

Domains:
- queries (SQL, regex): SQLizer, Regel
- programmer’s assistant: SWIM, GPT-3?
natural language: discussion

Domains:

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

+ beginner-friendly
+ expressive

- ambiguous
- verbose
specifications: conclusions

synthesis is more than just PBE/PBD!
specifications: conclusions

synthesis is more than just PBE/PBD!
depends on target domain and audience
targeting programmers? can use programs or types!
specifications: conclusions

synthesis is more than just PBE/PBD!
depends on target domain and audience
  targeting programmers? can use programs or types!
best results when combined
  types + examples, NL + examples
help eliminate ambiguity
this talk

specifications

search strategies
this talk

specifications

search strategies
program synthesis

specification → search → program

program space
program synthesis

specification → search → program

program space

DSL
context-free grammars (CFGs)

a grammar of list expressions

L ::= sort(L) | L[N..N] | L + L | [N] | x
N ::= find(L,N) | 0
context-free grammars (CFGs)

a grammar of list expressions

terminals

```
L ::= sort(L) | L[N..N] | L + L | [N] | x
N ::= find(L,N) | 0
```
context-free grammars (CFGs)

a grammar of list expressions

L ::= sort(L) | L[N..N] | L + L | [N] | x

N ::= find(L,N) | 0
context-free grammars (CFGs)

a grammar of list expressions

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

terminals
nonterminals
productions
context-free grammars (CFGs)

A grammar of list expressions:

Let $L$ denote the list of elements, $N$ denote the element number, $x$ denote an element,

Starting nonterminal: $L$


Nonterminals: $N$, $\theta$

Productions:

- $L ::= \text{sort}(L)$
- $L[N..N]$
- $L + L$
- $[N]$
- $x$
- $N ::= \text{find}(L,N)$
- $\theta$
this talk

specifications

search strategies

1. enumerative
2. VSA & co
3. stochastic
4. constraint-based
this talk

search strategies

1. enumerative
2. VSA & co
3. stochastic
4. constraint-based
enumerative search

idea: sample programs from the grammar one by one and check if they satisfy the spec

challenge: how do we systematically enumerate all programs?
enumerative search

idea: sample programs from the grammar one by one and check if they satisfy the spec

challenge: how do we systematically enumerate all programs?

bottom-up vs top-down
bottom-up enumeration

start from variables / literals
combine sub-programs into larger programs using productions

\[
\begin{align*}
L & ::= \text{sort}(L) \\
 & \quad | \quad L[N..N] \\
 & \quad | \quad L + L \\
 & \quad | \quad [N] \\
 & \quad | \quad x \\
N & ::= \text{find}(L,N) \\
 & \quad | \quad 0 \\
\end{align*}
\]

\[[1,4,0,6] \rightarrow [1,4]\]
bottom-up: example

Program bank

height 0:

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

[[1,4,0,6] → [1,4]]
bottom-up: example

Program bank

height 0:

\[
L ::= \text{sort}(L) \quad | \\
L[N..N] \quad | \\
L + L \quad | \\
[N] \quad | \\
x \quad |
\]

\[
N ::= \text{find}(L,N) \quad | \\
\emptyset
\]

[[1,4,0,6] → [1,4]]
bottom-up: example

Program bank

height 0:  x

\[ L ::= \text{sort}(L) \mid L[N..N] \mid L + L \mid [N] \mid x \]

\[ N ::= \text{find}(L,N) \mid \emptyset \]

\[ [[1,4,0,6] \rightarrow [1,4]] \]
bottom-up: example

Program bank

height 0:  x

L ::= sort(L) | L[N..N] | L + L | [N] | x
N ::= find(L,N) | 0
[[1,4,0,6] → [1,4]]
bottom-up: example

Program bank

height 0:  x  0

L ::= sort(L)  |  L[N..N]  |  L + L  |  [N]  |  x  
N ::= find(L,N)  |  \emptyset  

[[1,4,0,6] \rightarrow [1,4]]
bottom-up: example

Program bank

height 0: \( x \quad 0 \)

height 1:

\[
\begin{align*}
L & ::= \text{sort}(L) \\
& \quad L[N..N] \\
& \quad L + L \\
& \quad [N] \\
& \quad x \\
N & ::= \text{find}(L,N) \\
& \quad 0 \\
[[1,4,0,6]] \rightarrow [1,4]
\end{align*}
\]
bottom-up: example

Program bank

height 0:  x  0

height 1:

L ::= sort(L)  
    L[N..N]  
    L + L  
    [N]  
    x

N ::= find(L,N)  
    0

[[1,4,0,6]  →  [1,4]]
bottom-up: example

Program bank

height 0:  \( x \)  \( \emptyset \)

height 1:  sort(x)

\[
L ::= \text{sort}(L) \\
L[N..N] \\
L + L \\
[N] \\
x \\
N ::= \text{find}(L,N) \\
\emptyset \\
[[1,4,0,6] \rightarrow [1,4]]
\]
bottom-up: example

Program bank

height 0: \( x \quad \emptyset \)

height 1: \( \text{sort}(x) \)

\[
L ::= \text{sort}(L) \\
L[N..N] \\
L + L \\
[N] \\
x \\
N ::= \text{find}(L,N) \\
\emptyset \\
[[1,4,0,6] \rightarrow [1,4]]
\]
bottom-up: example

Program bank

height 0:  x  0

height 1:  \textbf{sort}(x)  x[0..0]

\[
L ::= \text{sort}(L) \\
N ::= \text{find}(L,N) \\
\]

[[1,4,0,6] \rightarrow [1,4]]
bottom-up: example

Program bank

height 0:  x  ∅

height 1:  sort(x)  x[0..0]

L ::= sort(L)  |  L[N..N]
    L + L
    [N]
    x

N ::= find(L,N)  |  ∅

[[1,4,0,6]  →  [1,4]]
bottom-up: example

Program bank

height 0: \[ x \quad \emptyset \]

height 1: \[ \text{sort}(x) \quad x[\emptyset..\emptyset] \quad x + x \]

\[
\begin{align*}
L & ::= \text{sort}(L) \\
    & | \quad L[N..N] \\
    & | \quad L + L \\
    & | \quad [N] \\
    & | \quad x \\
N & ::= \text{find}(L,N) \\
    & | \quad \emptyset \\
    & | \quad [1,4,0,6] \rightarrow [1,4]
\end{align*}
\]
bottom-up: example

Program bank

height 0:  x  0

height 1:  sort(x)  x[0..0]  x + x

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

[[1,4,0,6] → [1,4]]
bottom-up: example

Program bank

height 0:  \( x \quad 0 \)

height 1:  \textbf{sort}(x) \quad x[0..0] \quad x + x \quad [0]

\[
\begin{align*}
L & ::= \text{sort}(L) \\
L[N..N] & \\
L + L & \\
[N] & \\
x & \\
N & ::= \text{find}(L,N) \\
\emptyset & \\
[[1,4,0,6] & \rightarrow [1,4]]
\end{align*}
\]
bottom-up: example

Program bank

height 0:  x  0

height 1:  sort(x)  x[0..0]  x + x  [0]

L ::= sort(L)  |
L[N..N]  |
L + L  |
[N]  |
x  |

N ::= find(L,N)  |
0  |

[[1,4,0,6]  →  [1,4]]
bottom-up: example

Program bank

height 0: \( x \ 0 \)

height 1: \( \text{sort}(x) \ x[0..0] \ x + x \ [0] \)
\( \text{find}(x,0) \)

\[
L ::= \text{sort}(L) \ \\
L[N..N] \ \\
L + L \ \\
[N] \ \\
x \ \\
N ::= \text{find}(L,N) \ \\
0 \ \\
[1,4,0,6] \rightarrow [1,4]
\]
bottom-up: example

Program bank

height 0:  \( x \quad \emptyset \)

height 1:  \( \text{sort}(x) \quad x[0..0] \quad x + x \quad [0] \)

height 2:

\[
L ::= \text{sort}(L) \quad | \\
L[N..N] \quad | \\
L + L \quad | \\
[N] \quad | \\
x \quad | \\
N ::= \text{find}(L,N) \quad | \\
\emptyset
\]

\([1,4,0,6] \rightarrow [1,4]\)
bottom-up: example

Program bank

height 0: \( x \quad \emptyset \)

height 1: \( \text{sort}(x) \quad x[0..0] \quad x + x \quad [\emptyset] \)
\( \text{find}(x,\emptyset) \)

height 2:

\[
L ::= \text{sort}(L) \quad L[N..N] \quad L + L \quad [N] \quad x
\]
\[
N ::= \text{find}(L,N) \quad \emptyset
\]

\([1,4,0,6] \rightarrow [1,4]\\)
**bottom-up: example**

Program bank

height 0:  \( x \quad 0 \)

height 1:  \( \text{sort}(x) \quad x[0..0] \quad x + x \quad [0] \)
           \( \text{find}(x,0) \)

height 2:  \( \text{sort}(\text{sort}(x)) \quad \text{sort}(x[0..0]) \quad \text{sort}(x + x) \)
           \( \text{sort}([0]) \)

\[ L ::= \text{sort}(L) \]
\[ L[N..N] \]
\[ L + L \]
\[ [N] \]
\[ x \]
\[ N ::= \text{find}(L,N) \]
\[ 0 \]

\([[1,4,0,6] \to [1,4]]\)
Program bank

height 0: \( x \quad \emptyset \)

height 1:
- \( \text{sort}(x) \)
- \( x[0..0] \)
- \( x + x \)
- \( \emptyset \)
- \( \text{find}(x,\emptyset) \)

height 2:
- \( \text{sort}(\text{sort}(x)) \)
- \( \text{sort}(x[0..0]) \)
- \( \text{sort}(x + x) \)
- \( \text{sort}([\emptyset]) \)

\[
L ::= \text{sort}(L) \quad | \quad L[N..N] \quad | \quad L + L \quad | \quad [N] \quad | \quad x
\]

\[
N ::= \text{find}(L,N) \quad | \quad \emptyset
\]

\([1,4,0,6] \rightarrow [1,4]\]
bottom-up: example

Program bank

height 0: \( x \ 0 \)

height 1: \[
\text{sort}(x) \quad x[0..0] \quad x + x \quad [0] \\
\text{find}(x,0)
\]

height 2: \[
\text{sort}(\text{sort}(x)) \quad \text{sort}(x[0..0]) \quad \text{sort}(x + x) \\
\text{sort}([0]) \quad x[0..\text{find}(x,0)] \quad x[\text{find}(x,0)..0] \\
x[\text{find}(x,0)..\text{find}(x,0)] \quad \text{sort}(x[0..0]) \\
x[0..0][0..0] \quad (x + x)[0..0] \quad [0][0..0]
\]

\[
L ::= \text{sort}(L) \quad L[N..N] \quad L + L \\
[N] \\
\text{x} \\
N ::= \text{find}(L,N) \quad 0 \\
[[1,4,0,6] \rightarrow [1,4]]
\]
bottom-up: example

Program bank

height 0:   \( x \quad 0 \)

height 1:   \[
\begin{align*}
&\text{sort}(x) \quad x[0..0] \quad x + x \quad [0] \\
&\text{find}(x,0)
\end{align*}
\]

height 2:   \[
\begin{align*}
&\text{sort}(\text{sort}(x)) \quad \text{sort}(x[0..0]) \quad \text{sort}(x + x) \\
&\text{sort}([0]) \quad x[0..\text{find}(x,0)] \quad x[\text{find}(x,0)..0] \\
&x[\text{find}(x,0)..\text{find}(x,0)] \quad \text{sort}(x)[0..0] \\
&x[0..0][0..0] \quad (x + x)[0..0] \quad [0][0..0]
\end{align*}
\]

\( L ::= \text{sort}(L) \quad | \quad L[N..N] \quad | \quad L + L \quad | \quad [N] \quad | \quad x \)

\( N ::= \text{find}(L,N) \quad | \quad 0 \)

\([1,4,0,6] \rightarrow [1,4]\)
bottom-up: example

Program bank

height 0:  x  0

height 1:  sort(x)  x[0..0]  x + x  [0]  find(x,0)

height 2:  sort(sort(x))  sort(x[0..0])  sort(x + x)
           sort([0])  x[0..find(x,0)]  x[find(x,0)..0]
           x[find(x,0)..find(x,0)]  sort(x)[0..0]
           x[0..0][0..0]  (x + x)[0..0]  [0][0..0]
           x + (x + x)  x + [0]  sort(x) + x  x[0..0] + x
           (x + x) + x  [0] + x  x + x[0..0]  x + sort(x)

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

[[1,4,0,6]  →  [1,4]]
bottom-up: example

Program bank

height 0: \( x \quad 0 \)

height 1: \( \text{sort}(x) \quad x[0..0] \quad x + x \quad [0] \)
\( \text{find}(x,0) \)

height 2: \( \text{sort}(\text{sort}(x)) \quad \text{sort}(x[0..0]) \quad \text{sort}(x + x) \)
\( \text{sort}([0]) \quad x[0..\text{find}(x,0)] \quad x[\text{find}(x,0)..0] \)
\( x[\text{find}(x,0)..\text{find}(x,0)] \quad \text{sort}(x)[0..0] \)
\( x[0..0][0..0] \quad (x + x)[0..0] \quad [0][0..0] \)
\( x + (x + x) \quad x + [0] \quad \text{sort}(x) + x \quad x[0..0] + x \)
\( (x + x) + x \quad [0] + x \quad x + x[0..0] \quad x + \text{sort}(x) \)
\( \ldots \)
bottom-up: example

Program bank

height 0:  
x  0

height 1:  
\text{sort}(x)  \quad x[0..0]  \quad x + x  \quad [0]  
\text{find}(x,0)

height 2:  
\text{sort}(\text{sort}(x))  \quad \text{sort}(x[0..0])  \quad \text{sort}(x + x)  
\text{sort}([0])  \quad x[0..\text{find}(x,0)]  \quad x[\text{find}(x,0)..0]  
x[\text{find}(x,0)..\text{find}(x,0)]  \quad \text{sort}(x)[0..0]  
x[0..0][0..0]  \quad (x + x)[0..0]  \quad [0][0..0]  
x + (x + x)  \quad x + [0]  \quad \text{sort}(x) + x  \quad x[0..0] + x  
(x + x) + x  \quad [0] + x  \quad x + x[0..0]  \quad x + \text{sort}(x)

\ldots
top-down enumeration

start from the start non-terminal
expand remaining non-terminals using productions

\[
L ::= L[N..N] \mid x \\
N ::= \text{find}(L,N) \mid 0
\]

\[[1,4,\emptyset,6] \rightarrow [1,4] \]
top-down: example

Worklist

\[
L ::= L[N..N] \mid x \\
N ::= \text{find}(L,N) \mid 0
\]

\[
[[1,4,0,6] \rightarrow [1,4]]
\]
top-down: example

Worklist

\L ::= L[N..N] | x

N ::= find(L,N) | \emptyset

[[1,4,0,6] \rightarrow [1,4]]
top-down: example

Worklist

L

iter 1:

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

[[1,4,0,6] → [1,4]]
top-down: example

Worklist

iter 1:

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

[[1,4,0,6] → [1,4]]
**top-down: example**

Worklist

$L$

**iter 1:** $x$

$L ::= \text{L}[N..N] \mid \text{x}$

$N ::= \text{find(}L,N\text{)} \mid \emptyset$

$[[1,4,0,6] \rightarrow [1,4]]$
top-down: example

Worklist

iter 1:  x

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

[[1,4,0,6] → [1,4]]
top-down: example

Worklist

iter 1: \( x \)  \( L[N..N] \)

\[
L ::= L[N..N] \\
N ::= \text{find}(L,N) \\
0 \\
\]

\([1,4,0,6] \rightarrow [1,4]\]
top-down: example

Worklist

```
L := L[N..N] | x
N := find(L,N) | 0
[[1,4,0,6] → [1,4]]
```
top-down: example

Worklist

$L \ ::= \ L[N..N]$  

$x \ N \ ::= \ find(L,N)$  

$\emptyset$  

$[1,4,0,6] \rightarrow [1,4]$
top-down: example

Worklist

\[
\begin{align*}
L & ::= L[N..N] \\
N & ::= \text{find}(L,N) \\
0 & \\
[[1,4,0,6]] & \rightarrow [1,4]
\end{align*}
\]
### top-down: example

**Worklist**

\[
L \\
\text{iter 1: } L[N..N] \times \\
\text{iter 2: } L[N..N] \\
\text{iter 3: }
\]

\[
L ::= L[N..N] | \\
   x \\
N ::= \text{find}(L,N) | \\
   \emptyset \\
[[1,4,0,6] \rightarrow [1,4]]
\]
top-down: example

Worklist

iter 1: \( \times \) \[L[N..N]\]
iter 2: \[L[N..N]\]
iter 3:

\[
\begin{align*}
L ::= & \ L[N..N] \quad | \\
N ::= & \ \text{find}(L, N) \quad | \\
0 & \\
[[1,4,0,6] \rightarrow [1,4]]
\end{align*}
\]
top-down: example

Worklist

iter 1:  \( \times \quad \text{L}[N..N] \)

iter 2:  \( \text{L}[N..N] \)

iter 3:  \( \text{x}[N..N] \)

\[
\begin{align*}
L & ::= L[N..N] \quad | \\
   & \quad x \\
N & ::= \text{find}(L, N) \quad | \\
   & \quad 0 \\
[[1,4,0,6] & \rightarrow [1,4]]
\end{align*}
\]
top-down: example

Worklist

Iter 1: \( \times L[N..N] \)
Iter 2: \( L[N..N] \)
Iter 3: \( x[N..N] \)

\[
L ::= L[N..N] \mid x \\
N ::= \text{find}(L,N) \mid 0 \\
[[1,4,0,6] \rightarrow [1,4]]
\]
top-down: example

Worklist

<table>
<thead>
<tr>
<th>iter 1:</th>
<th>x</th>
<th>L[N..N]</th>
</tr>
</thead>
<tbody>
<tr>
<td>iter 2:</td>
<td>L[N..N]</td>
<td></td>
</tr>
</tbody>
</table>

N ::= find(L,N) | 0

[[1,4,0,6] → [1,4]]
top-down: example

Worklist

iter 1: \( \times \) \[N..N\]
iter 2: \[N..N\]
iter 3: \( x \)[N..N] \[N..N]\]

\[
L ::= L[N..N] | x \\
N ::= \text{find}(L,N) | \theta \\
\]

\([[1,4,\theta,6] \rightarrow [1,4]]\)
top-down: example

Worklist

iter 1: \( \times \) L[N..N]
iter 2: L[N..N]

\[
L ::= L[N..N] \mid x \\
N ::= \text{find}(L,N) \mid 0 \\
[[1,4,0,6] \rightarrow [1,4]]
\]
top-down: example

Worklist

| iter 1: | L[N..N] |
| iter 2: | L[N..N] |
| iter 4: | x[0..N]  L[N..N][N..N] |

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

[[1,4,0,6] → [1,4]]
```
top-down: example

Worklist

iter 1: $x \times L[N..N]$
iter 2: $L[N..N]$
iter 4: $x[\emptyset..N] \quad L[N..N][N..N]$

L ::= L[N..N] | x
N ::= find(L,N) | \emptyset

[[1,4,\emptyset,6] \rightarrow [1,4]]
**top-down: example**

Worklist

L

iter 1: \( \times \) \( L[N..N] \)

iter 2: \( L[N..N] \)


iter 4: \( x[0..N] \) \( x[\text{find}(L,N)..N] \) \( L[N..N][N..N] \)

\[ [1,4,0,6] \rightarrow [1,4] \]
top-down: example

**Worklist**

```
iter 1:  \( \times \) \( L[N..N] \)
iter 2:  \( L[N..N] \)
iter 4:  \( x[0..N] \quad x[\text{find}(L,N)..<N] \quad L[N..N][N..N] \)
iter 5:  \( x[\text{find}(L,N)..<N] \quad \ldots \)
```

```
L ::= L[N..N]  |  
    x
N ::= \text{find}(L,N)  |  
    0

[[1,4,0,6] \rightarrow [1,4]]
```
top-down: example

Worklist

\[
L \::= \begin{cases} 
L[N..N] & | \\
x & | \\
N \::= \text{find}(L,N) & | \\
\emptyset & |
\end{cases}
\]

\[
[[1,4,0,6] \rightarrow [1,4]]
\]
## top-down: example

### Worklist

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Worklist</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iter 1:</td>
<td>x (\times) (L[N..N])</td>
</tr>
<tr>
<td>Iter 2:</td>
<td>(L[N..N])</td>
</tr>
<tr>
<td>Iter 4:</td>
<td>(x[0..N]) (x[\text{find}(L,N)\ldots N]) (L[N..N][N..N])</td>
</tr>
<tr>
<td>Iter 5:</td>
<td>(x[0..0]) (x[\text{find}(L,N)\ldots N])</td>
</tr>
</tbody>
</table>

\[
L ::= L[N..N] \mid x \\
N ::= \text{find}(L,N) \mid 0 \\
[[1,4,0,6] \rightarrow [1,4]]
\]
**top-down: example**

Worklist

iter 1: \[\times, L[N..N]\]

iter 2: \[L[N..N]\]


iter 4: \[x[0..N], x[\text{find}(L,N)..N], L[N..N][N..N]\]

iter 5: \[x[0..0], x[\text{find}(L,N)..N]\]

---

L ::= L[N..N] | \times 
N ::= \text{find}(L,N) | 0

[[1,4,0,6] \rightarrow [1,4]]
top-down: example

Worklist

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>iter 1:</td>
<td>x</td>
<td>L[N..N]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iter 2:</td>
<td>L[N..N]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iter 4:</td>
<td>x[0..N]</td>
<td>x[find(L,N)..N]</td>
<td>L[N..N][N..N]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iter 5:</td>
<td>x[0..0]</td>
<td>x[0.. find(L,N)]</td>
<td>x[find(L,N)..N]</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[L ::= L[N..N] \mid x\]
\[N ::= \text{find}(L,N) \mid 0\]
\[[[1,4,0,6] \rightarrow [1,4]]\]
top-down: example

Worklist

| iter 1: | x[0..0] L[N..N] |
| iter 2: | L[N..N] |
| iter 4: | x[0..N] x[find(L,N)..N] L[N..N][N..N] |
| iter 5: | x[0..0] x[0.. find(L,N)] x[find(L,N)..N] |
| iter 6: |

L ::= L[N..N] | x
N ::= find(L,N) | 0
[[1,4,0,6] → [1,4]]
top-down: example

Worklist

iter 1: \(x\) L[N..N]
iter 2: L[N..N]
iter 4: x[0..N] x[find(L,N)..N] L[N..N][N..N]
iter 5: x[0..0] x[0.. find(L,N)] x[find(L,N)..N] ...
iter 6:

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

[[1,4,0,6] \rightarrow [1,4]]
**top-down: example**

Worklist

```
L

iter 1: x^ L[N..N]

iter 2: L[N..N]


iter 4: x[0..N]  x[find(L,N)..N]  L[N..N][N..N]

iter 5: x[0..0]  x[0.. find(L,N)]  x[find(L,N)..N]  ...

iter 6: x[0.. find(L,N)]  x[find(L,N)..N]  ...
```

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

[[1,4,0,6]  →  [1,4]]
```
top-down: example

Worklist

L

iter 1: \(\times L[N..N]\)

iter 2: \(L[N..N]\)


iter 4: \(x[0..N], x[\text{find}(L,N)..N], L[N..N][N..N]\)

iter 5: \(x[0..0], x[0.. \text{find}(L,N)], x[\text{find}(L,N)..N], \ldots\)

iter 6: \(x[0.. \text{find}(L,N)], x[\text{find}(L,N)..N], \ldots, \ldots\)

iter 7: \(x[0.. \text{find}(x,N)], x[0.. \text{find}(L[N..N],N)], \ldots, \ldots, \ldots\)

\[L ::= L[N..N] \quad | \quad x\]

\[N ::= \text{find}(L,N) \quad | \quad 0\]

\([\{1,4,0,6\} \rightarrow [1,4]]\)
top-down: example

Worklist

```
L

iter 1: x[×] L[N..N]

iter 2: L[N..N]


iter 5: x[∅..∅][×] x[∅.. find(L,N)] x[find(L,N) ..N] ...

iter 6: x[∅.. find(L,N)] x[find(L,N) ..N] ...

iter 7: x[∅.. find(x,N)] x[∅.. find(L[N..N],N)] ...

iter 8:
```

```
L ::= L[N..N] | x
N ::= find(L,N) | 0
[[1,4,∅,6] → [1,4]]
```
top-down: example

Worklist

L

iter 1:  x \neq L[N..N]

iter 2:  L[N..N]


iter 5:  x[\emptyset..\emptyset]  x[\emptyset.. \text{find}(L,N)]  x[\text{find}(L,N)..N] ...

iter 6:  x[\emptyset.. \text{find}(L,N)]  x[\text{find}(L,N)..N]  ...

iter 7:  x[\emptyset.. \text{find}(x,N)]  x[\emptyset.. \text{find}(L[N..N],N)]  ...

iter 8:  x[\emptyset.. \text{find}(x,\emptyset)]  x[\emptyset.. \text{find}(x,\text{find}(L,N))]  ...

L ::= L[N..N]  |  x
N ::= \text{find}(L,N)  |  \emptyset
[[1,4,\emptyset,6] \rightarrow [1,4]]
top-down: example

Worklist

L

iter 1: x L[N..N]

iter 2: L[N..N]


iter 4: x[0..N] x[find(L,N)..N] L[N..N][N..N]

iter 5: x[0..0] x[0.. find(L,N)] x[find(L,N)..N] ...

iter 6: x[0.. find(L,N)] x[find(L,N)..N] ...

iter 7: x[0.. find(x,N)] x[0.. find(L[N..N],N)] ...

iter 8: x[0.. find(x,0)] x[0.. find(x,find(L,N))] ...

iter 9:

L ::= L[N..N] | x
N ::= find(L,N) | 0
[[1,4,0,6] → [1,4]]
### top-down: example

#### Worklist

L

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Worklist</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><img src="image" alt="Worklist 1" /></td>
</tr>
<tr>
<td>2</td>
<td><img src="image" alt="Worklist 2" /></td>
</tr>
<tr>
<td>3</td>
<td><img src="image" alt="Worklist 3" /></td>
</tr>
<tr>
<td>4</td>
<td><img src="image" alt="Worklist 4" /></td>
</tr>
<tr>
<td>5</td>
<td><img src="image" alt="Worklist 5" /></td>
</tr>
<tr>
<td>6</td>
<td><img src="image" alt="Worklist 6" /></td>
</tr>
<tr>
<td>7</td>
<td><img src="image" alt="Worklist 7" /></td>
</tr>
<tr>
<td>8</td>
<td><img src="image" alt="Worklist 8" /></td>
</tr>
<tr>
<td>9</td>
<td><img src="image" alt="Worklist 9" /></td>
</tr>
</tbody>
</table>

---

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

[[1,4,0,6] → [1,4]]
how to make it scale

Prune
Discard useless subprograms

Prioritize
Explore more promising candidates first
how to make it scale

Prune
Discard useless subprograms

Prioritize
Explore more promising candidates first

\[ m \times N^2 \quad m \times (N - 1)^2 \]
how to make it scale

Prune
Discard useless subprograms

Prioritize
Explore more promising candidates first

\[ P = \{ [\emptyset][N..N], x[N..N], \ldots \} \]

\[ m \times N^2 \quad m \times (N - 1)^2 \]
how to make it scale

Prune
Discard useless subprograms

Prioritize
Explore more promising candidates first

\[ P = \{ \text{[0][N..N]}, \text{x[N..N]}, \ldots \} \]

dequeue this first

\[ m \times N^2 \quad m \times (N - 1)^2 \]
when can we discard a subprogram?
when can we discard a subprogram?

It’s equivalent to something we have already explored

\[
\text{sort}(x)
\]
when can we discard a subprogram?

It’s equivalent to something we have already explored

\[
\text{sort(sort(x))}
\]

\[
\text{sort(x)}
\]
when can we discard a subprogram?

It’s equivalent to something we have already explored

\( \texttt{sort(sort(x))} \)
when can we discard a subprogram?

It’s equivalent to something we have already explored

Equivalence reduction
when can we discard a subprogram?

It's equivalent to something we have already explored

No matter what we combine it with, it cannot satisfy the spec

Equivalence reduction
when can we discard a subprogram?

It’s equivalent to something we have already explored

No matter what we combine it with, it cannot satisfy the spec

Equivalent reduction

\[
[\_] \rightarrow [\_]
\]

\[
\ldots
\]
when can we discard a subprogram?

It’s equivalent to something we have already explored

No matter what we combine it with, it cannot satisfy the spec

Equivalence reduction
when can we discard a subprogram?

It’s equivalent to something we have already explored

No matter what we combine it with, it cannot satisfy the spec

Equivalence reduction
when can we discard a subprogram?

It’s equivalent to something we have already explored

No matter what we combine it with, it cannot satisfy the spec

Equivalence reduction

\[
\begin{align*}
@ & \\
sort(sort(x)) & \\
sort(x) & \\
\end{align*}
\]

\[
\begin{align*}
L & + L \\
[N] + L & \\
\end{align*}
\]

\[
[] \rightarrow []
\]

...
when can we discard a subprogram?

It’s equivalent to something we have already explored

No matter what we combine it with, it cannot satisfy the spec

Equivalence reduction

Top-down propagation

\[ \text{sort}(\text{sort}(x)) \]

\[ \text{sort}(x) \]

\[ L + L \]

\[ [N] + L \]
when can we discard a subprogram?

It’s equivalent to something we have already explored

Equivalence reduction
equivalent programs

\[
\begin{align*}
L &::= \text{sort}(L) \\
& \quad | \\
& L[N..N] \\
& \quad | \\
& L + L \\
& \quad | \\
& [N] \\
& \quad | \\
x & \\
N &::= \text{find}(L,N) \\
& \quad | \\
& 0
\end{align*}
\]
equivalent programs

\[
\begin{align*}
L &::= \text{sort}(L) \mid L[N..N] \mid L + L \mid [N] \mid x \\
N &::= \text{find}(L,N) \mid 0
\end{align*}
\]
### equivalent programs

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

---

bottom_up

x  θ
equivalent programs

\[
\begin{align*}
L & ::= \text{sort}(L) \\
    & \mid L[N..N] \\
    & \mid L + L \\
    & \mid [N] \\
    & \mid x \\
N & ::= \text{find}(L,N) \\
    & \mid 0
\end{align*}
\]
equivalent programs

\[
L ::= \text{sort}(L) \mid \text{L}[N..N] \mid L + L \mid [N] \\
N ::= \text{find}(L,N) \mid 0
\]

\[
bottom\_up
\]

\[
x \ 0
\]

\[
\text{sort}(x) \ x[0..0] \ x + x \ [0] \ \text{find}(x,0)
\]

\[
\text{sort(sort}(x)) \ \text{sort}(x + x) \ \text{sort}(x[0..0]) \\
\text{sort([0])} \ x[0..\text{find}(x,0)] \ x[\text{find}(x,0)..0] \\
x[\text{find}(x,0)..\text{find}(x,0)] \ \text{sort}(x)[0..0] \\
x[0..0][0..0] \ (x + x)[0..0] \ [0][0..0] \\
x + (x + x) \ x + [0] \ \text{sort}(x) + x \ x[0..0] + x \\
(x + x) + x \ [0] + x \ x + x[0..0] \ x + \text{sort}(x) \\
\ldots
\]
equivalent programs

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

bottom_up

\[
\begin{align*}
\text{sort}(x) & \quad x[0..0] \quad x + x \quad [0] \quad \text{find}(x,0) \\
\text{sort}(\text{sort}(x)) & \quad \text{sort}(x + x) \quad \text{sort}(x[0..0]) \\
\text{sort}([0]) & \quad x[0..\text{find}(x,0)] \quad x[\text{find}(x,0)..0] \\
x[\text{find}(x,0)..\text{find}(x,0)] & \quad \text{sort}(x)[0..0] \\
x[0..0][0..0] & \quad (x + x)[0..0] \quad [0][0..0] \\
x + (x + x) & \quad x + [0] \quad \text{sort}(x) + x \quad x[0..0] + x \\
(x + x) + x & \quad [0] + x \quad x + x[0..0] \quad x + \text{sort}(x) \\
\ldots
\end{align*}
\]
equivalent programs

\[
L ::= \text{sort}(L) \mid L[N..N] \mid L + L \mid [N] \mid x
\]

\[
N ::= \text{find}(L, N) \mid 0
\]
equivalent programs

\[
L ::= \text{sort}(L) \\
L[N..N] \\
L + L \\
[N] \\
x \\
N ::= \text{find}(L, N) \\
\theta
\]

\[
x + (x + x) \\
\text{sort}(x) + x \\
\theta + x \\
x + \text{sort}(x)
\]

How do we check program equivalence?
idea: In PBE we only care about equivalence on the given inputs! easy to check efficiently even more programs are equivalent
observational equivalence

idea: In PBE we only care about equivalence on the given inputs!
   easy to check efficiently
   even more programs are equivalent

[[0] → [0]]
observational equivalence

idea: In PBE we only care about equivalence on the given inputs!
   easy to check efficiently
   even more programs are equivalent

\[
[[0] \rightarrow [0]]
\]
idea: In PBE we only care about equivalence on the given inputs!

easy to check efficiently
even more programs are equivalent

\[[0] \rightarrow [0]\]

\(x + (x + x)\)
how to make it scale

Prune
Discard useless subprograms

Prioritize
Explore more promising candidates first

$$P = \{ [\emptyset][N..N], x[N..N], \ldots \}$$

dequeue this first

$$m \times N^2$$

$$m \times (N - 1)^2$$
how to make it scale

Prioritize
Explore more promising candidates first

\[ P = \{ [\emptyset][N..N], x[N..N], ... \} \]
top-down search (revisited)

Turn off the rightmost sequence of 1s:

00101 → 00100
01010 → 01000
10110 → 10000
top-down search (revisited)

Turn off the rightmost sequence of 1s:

\[
\begin{align*}
00101 & \rightarrow 00100 \\
01010 & \rightarrow 01000 \\
10110 & \rightarrow 10000
\end{align*}
\]

\[
\begin{array}{cccc}
S & \rightarrow & 0 & 1 \\
S + S & | & | \\
S - S & | & | \\
S \& S & | & | \\
S | S & | & | \\
S << S & | & | \\
S >> S & | & |
\end{array}
\]
top-down search (revisited)

Turn off the rightmost sequence of 1s:

00101 → 00100
01010 → 01000
10110 → 10000

\[
x \& (1 + (x | x - 1))
\]
top-down search (revisited)

Turn off the rightmost sequence of 1s:

\[ \begin{align*}
00101 & \rightarrow 00100 \\
01010 & \rightarrow 01000 \\
10110 & \rightarrow 10000 \\
\end{align*} \]

\[ S \rightarrow 0 | 1 | x | \]

\[ S + S \]
\[ S - S \]
\[ S \land S \]
\[ S | S \]
\[ S << S \]
\[ S >> S \]

\[ x \& (1 + (x | x - 1)) \]
top-down search (revisited)

Turn off the rightmost sequence of 1s:

00101 → 00100
01010 → 01000
10110 → 10000

\[
\begin{align*}
S & \rightarrow 0 \mid 1 \mid x \\
S & + S \\
S & - S \\
S & \& S \\
S & | S \\
S & \ll S \\
S & \gg S \\
\end{align*}
\]

x&(1+(x|x-1))
top-down search (revisited)

Turn off the rightmost sequence of 1s:

- 00101 → 00100
- 01010 → 01000
- 10110 → 10000

Rules:

- $S \rightarrow 0 \mid 1 \mid x$
- $S + S$
- $S - S$
- $S \& S$
- $S \mid S$
- $S \ll S$
- $S \gg S$
- $x \& (1 + (x \mid x - 1))$
### top-down search (revisited)

Turn off the rightmost sequence of 1s:

- $00101 \rightarrow 00100$
- $01010 \rightarrow 01000$
- $10110 \rightarrow 10000$

<table>
<thead>
<tr>
<th>$S$</th>
<th>$0$</th>
<th>$1$</th>
<th>$x$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S + S$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$S - S$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$S &amp; S$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$S \mid S$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$S \ll S$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$S \gg S$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$S$:

- $x \& (x+1)$
- $x \&(1+(x\mid x-1))$

$0$

$1$

$x$

$S \& S$

$x \gg S$

$S \gg S$
top-down search (revisited)

Turn off the rightmost sequence of 1s:

00101 → 00100
01010 → 01000
10110 → 10000

S → 0 | 1 | x |
    S + S
    S - S
    S & S
    S | S
    S << S
    S >> S
top-down search (revisited)

Turn off the rightmost sequence of 1s:

00101 → 00100
01010 → 01000
10110 → 10000

S → 0 | 1 | x |
   S + S
   S - S
   S & S
   S | S
   S << S
   S >> S
top-down search (revisited)

Turn off the rightmost sequence of 1s:

00101 → 00100
01010 → 01000
10110 → 10000

S -> 0 | 1 | x |
   S + S
   S - S
   S & S
   S | S
   S << S
   S >> S

...
top-down search (revisited)

Turn off the rightmost sequence of 1s:

00101 → 00100
01010 → 01000
10110 → 10000

\[ S \rightarrow 0 \mid 1 \mid x \mid S + S \mid S - S \mid S \& S \mid S \mid S \mid S << S \mid S >> S \]

Explores many unlikely programs!
weighted top-down search

Idea: explore programs in the order of likelihood, not size

[Euphony. Lee at al’15]
weighted top-down search

Idea: explore programs in the order of likelihood, not size

\[ Euphony. Lee at al'15 \]
weighted top-down search

Idea: explore programs in the order of likelihood, not size

[Euphony. Lee at al’15]
weighted top-down search

Idea: explore programs in the order of likelihood, not size

\[
Euphony. \text{Lee at al'15}
\]
weighted top-down search

Idea: explore programs in the order of likelihood, not size

[Euphony. Lee et al’15]
weighted top-down search

Idea: explore programs in the order of likelihood, not size

1. Assign weights \( w(e) \) to edges such that \( d(p) < d(p') \) iff \( p \) is more likely than \( p' \)

---

[Euphony. Lee at al’15]
weighted top-down search

**Idea:** explore programs in the order of likelihood, not size

1. Assign weights $w(e)$ to edges such that $d(p) < d(p')$ iff $p$ is more likely than $p'$

2. Use A* algorithm to find closest leaves

[1] Euphony. Lee at al’15
weighted top-down search

Idea: explore programs in the order of likelihood, not size

1. Assign weights $w(e)$ to edges such that $d(p) < d(p')$ iff $p$ is more likely than $p'$

2. Use A* algorithm to find closest leaves

3. Weights can be learned offline or online

[Euphony. Lee at al’15]
bottom-up vs top-down

down-bottom

top-down
bottom-up vs top-down

bottom-up

+ dynamic programming
  (result caching)
  + can evaluate candidates
  (good for equivalence reduction)

top-down
bottom-up vs top-down

bottom-up

+ dynamic programming (result caching)
+ can evaluate candidates (good for equivalence reduction)

top-down

+ goal-directed
+ good for spec propagation
+ can be guided with context-dependent weights
enumerative search: discussion

examples: Synquid, Morpheus, Snippy
enumerative search: discussion

examples: Synquid, Morpheus, Snippy

+ works for any language you can evaluate / check

- might not fully leverage spec / DSL
this talk

search strategies

1. enumerative
2. VSA & co
3. stochastic
4. constraint-based
representation-based search

idea: build a graph that compactly represents all programs that satisfy the spec

examples: FlashhFill, FlashExtract, WebRelate

+ efficient
+ returns a ranked list of solutions

- restricted class of DSLs
this talk

search strategies

1. enumerative
2. VSA & co
3. stochastic
4. constraint-based
search space
search space

Enumerative search
search space

- CFG (small)
- PCFG (made from likely rules)
search space

Enumerative search

Weighted enumerative search

PCFG

Made from likely rules
search space

Enumeration search

Small

PCFG

Made from likely rules

Weighted enumerative search

Local search
naïve local search
naïve local search
naïve local search
naïve local search

program

better program

can generate $p_2$ from $p_1$ (and vice versa) via mutation
naïve local search

To find the best program:

\[
p := \text{random()}
\]

\[
\text{while (true) {}
    p' := \text{mutate}(p);
    \text{if (cost}(p') < \text{cost}(p))
    \quad p := p';
\}
\]
naïve local search

To find the best program:

\[
p := \text{random()}
\]

\[
\text{while (true) } \{
    p' := \text{mutate}(p);
    \text{if (cost}(p') < \text{cost}(p))
    \quad p := p';
\}
\]

Will never get to \(p_1\) from \(p_2\)!
MCMC sampling

Avoid getting stuck in local minima:

\[
p := \text{random()}
\]
\[
\text{while (true)} \{
    p' := \text{mutate}(p);
    \text{if (random}(A(p \rightarrow p')))
    \quad p := p';
\}
\]

where

if \( p' \) is better than \( p \): \( A(p \rightarrow p') = 1 \)
otherswise: \( A(p \rightarrow p') \) decreases with difference in cost between \( p' \) and \( p \)
MCMC for superoptimization

[STOKE. Schkufza et al ‘13]
stochastic search: discussion

examples: STOKÉ, FrAngel

+ can explore program spaces with no a-priori bias
+ good for sampling from a distribution

- needs cost function that faithfully approximates correctness
search strategies

1. enumerative
2. VSA & co
3. stochastic
4. constraint-based
constraint-based search

idea: reduce synthesis to a SAT/SMT problem and let a solver deal with it

challenge: how to encode a synthesis problem into SAT/SMT?
what is an encoding?

\[ C = \{ c_1, \ldots, c_n \} \]

\[ \{ C \mid \text{wf}(C) \} \]

\[ \{ C \mid \text{wf}(C) \wedge \varphi(C, i, o) \} \]
how to define an encoding

• define the parameter space $C = \{c_1, \ldots, c_n\}$
  
  • encode : $\text{Prog} \rightarrow C$
  • decode : $C \rightarrow \text{Prog}$ (might not be defined for all C)

• define a formula $\text{wf}(c_1, \ldots, c_n)$
  • that holds iff $\text{decode}[c]$ is a “well-formed” program

• define a formula $\varphi(c_1, \ldots, c_n, i, o)$
  • that holds iff $(\text{decode}[C])(i) = o$
SAT encoding: example

\(x\) is a two-bit word
\((x = x_h x_l)\)

program space
\(\{x, x \& 1\}\)

parameter space
\(C = \{c : \text{Bool}\}\)

\(E = [11 \rightarrow 01]\)

decode[\(\emptyset\)] \(\rightarrow x\)
decode[1] \(\rightarrow x \& 1\)

\(\text{Model } \{c \rightarrow 1\}\)

\(\text{SAT solver}\)

\(\text{return } \text{decode[1]} \text{ i.e. } x \& 1\)
SMT encoding: example

**Program Space**

\[ x + N \mid x \times N \]

- \( x \) is an integer input
- \( N \) is an integer literal

**Parameter Space**

\[ C = \{ c_{op} : \text{Bool}, c_N : \text{Int} \} \]

- \( \text{decode}[0,N] \rightarrow x + N \)
- \( \text{decode}[1,N] \rightarrow x \times N \)

\[ E = [2 \rightarrow 9] \]

\[ \text{Model} \{ c_{op} \rightarrow 0, \ c_N \rightarrow 7 \} \]

**SMT Solving**

\[ \text{SAT}(\varphi(c_{op}, c_N, 2, 9)) \]

\[ \text{SAT}((\neg c_{op} \Rightarrow 9 = 2 + c_N) \land (c_{op} \Rightarrow 9 = 2 \times c_N)) \]

- \( \text{return} \) \( \text{decode}[0,7] \) i.e. \( x + 7 \)
constraint-based search: discussion

examples: Sketch, Hoogle+

+ efficient
+ SMT solvers can discover constants

- finite number of parameters
  (no recursive grammars)
- program semantics expressible as a SAT/SMT formula
Comparison of search strategies

- VSA
- Constraint-based
- Stochastic
- Enumerative
this talk

specifications

search strategies
program synthesis