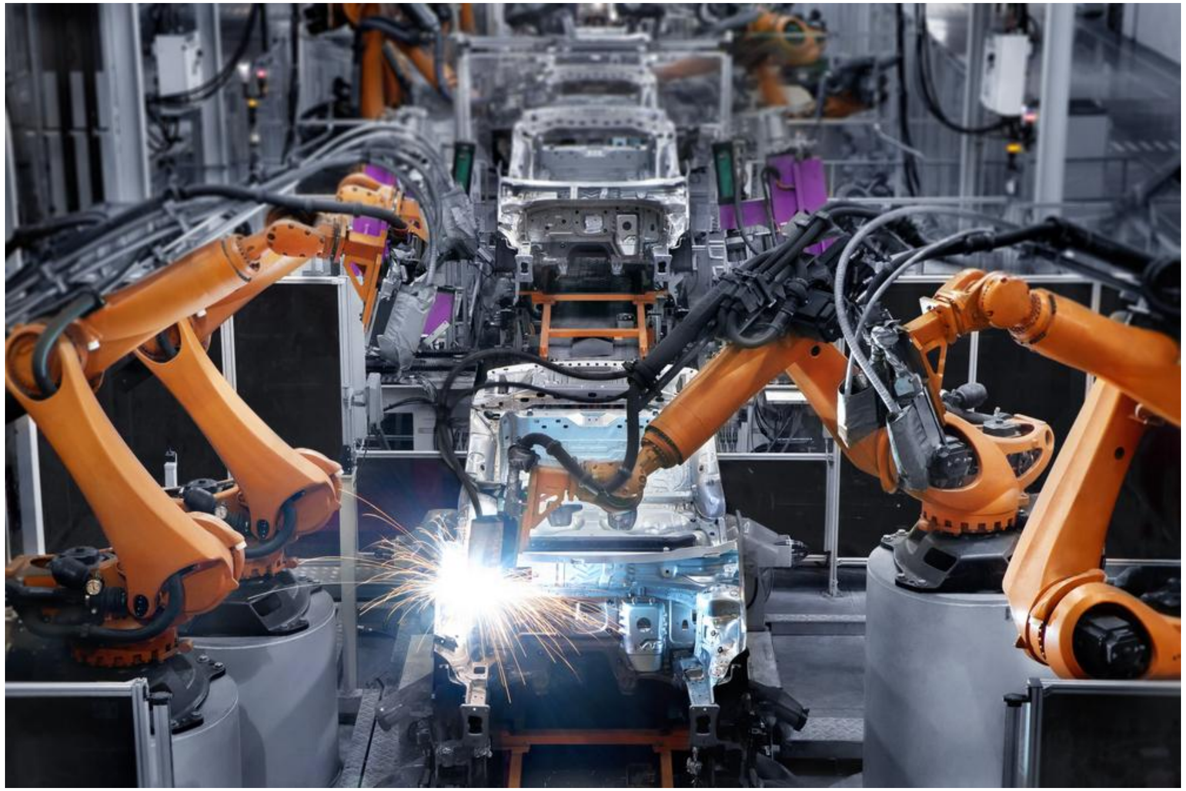
Generating correct code for your programmers

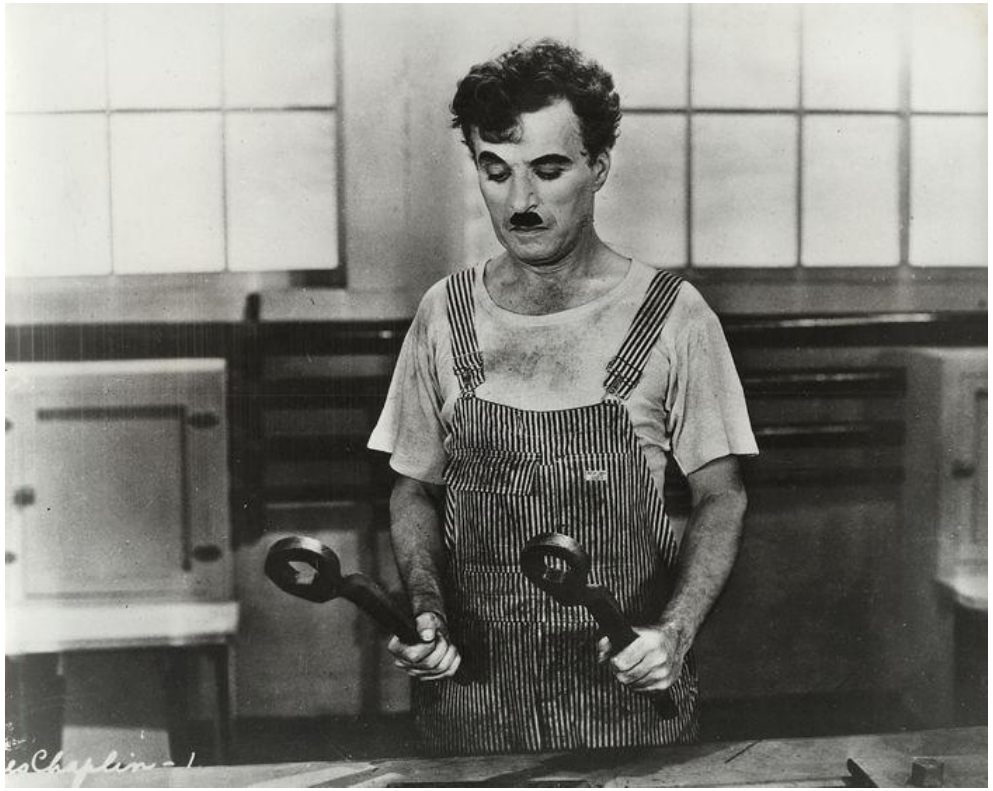
PLISS 2025 – Part I

Hila Peleg - Technion

Funded by the European Union (ERC, EXPLOSYN, 101117232). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.

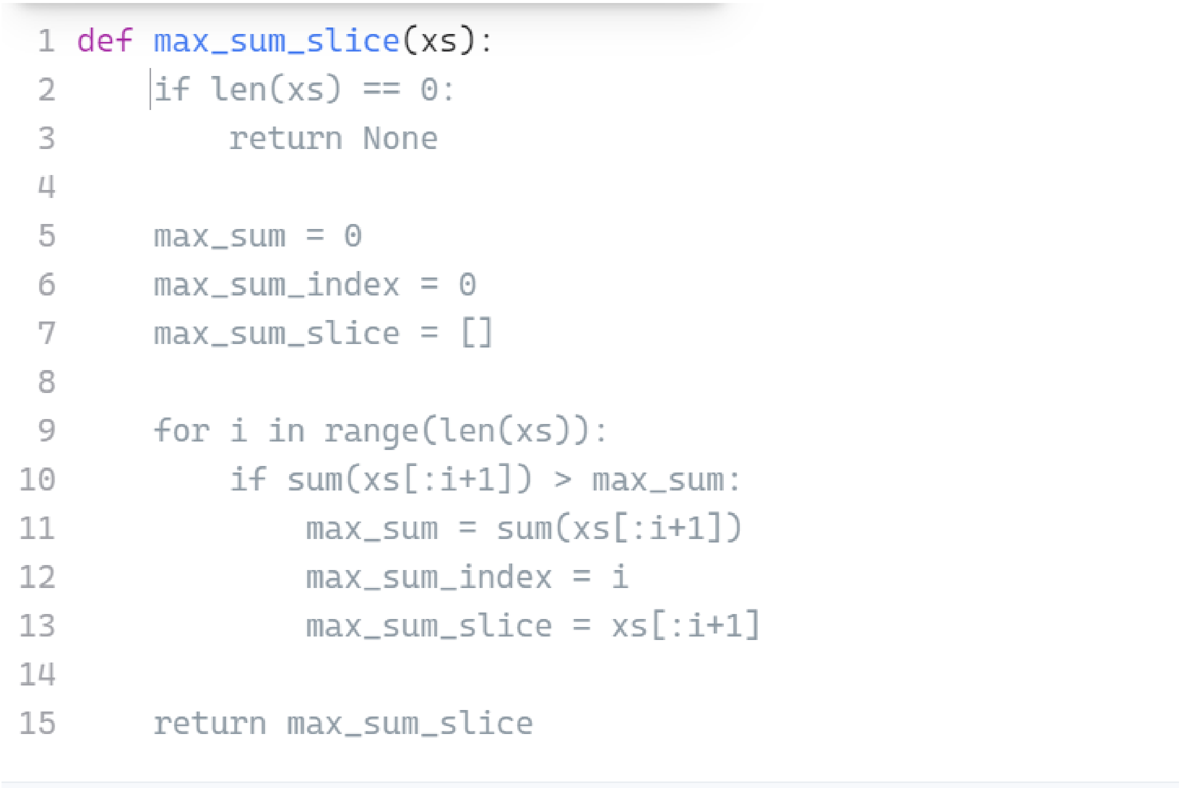
The goal: Automate some programming





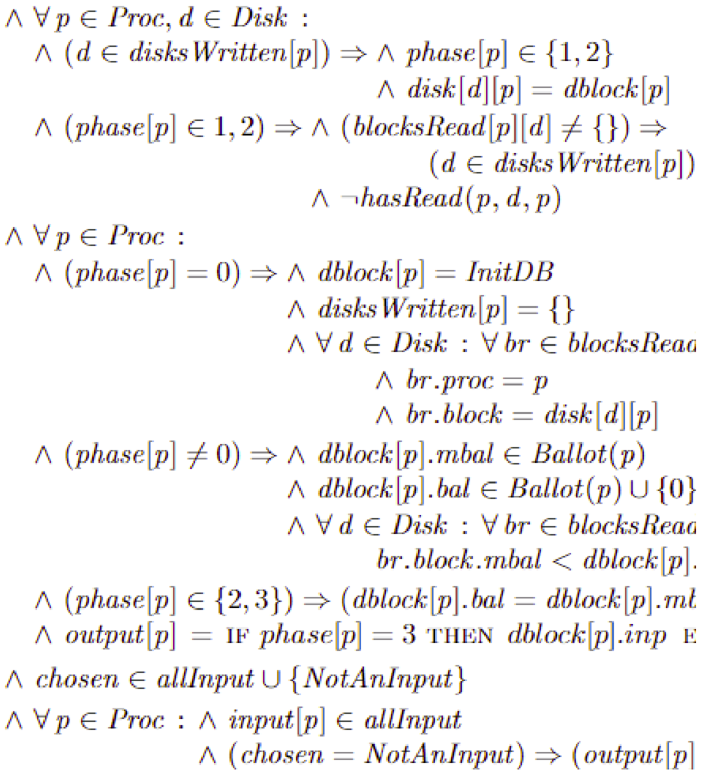
…in a *correct by construction* way

GitHub Copilot



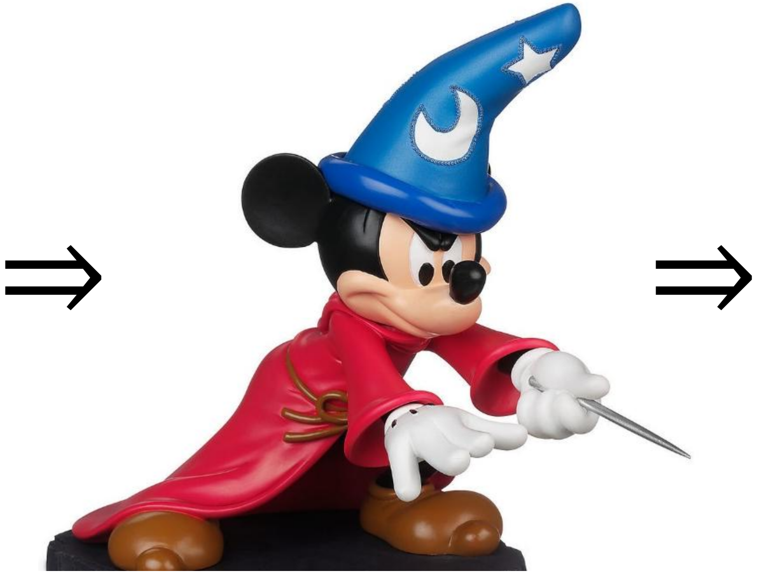
```python
def max_sum_slice(xs):
    if len(xs) == 0:
        return None

    max_sum = 0
    max_sum_index = 0
    max_sum_slice = []

    for i in range(len(xs)):
        if sum(xs[:i+1]) > max_sum:
            max_sum = sum(xs[:i+1])
            max_sum_index = i
            max_sum_slice = xs[:i+1]

    return max_sum_slice
```

Specifications go in, code comes out

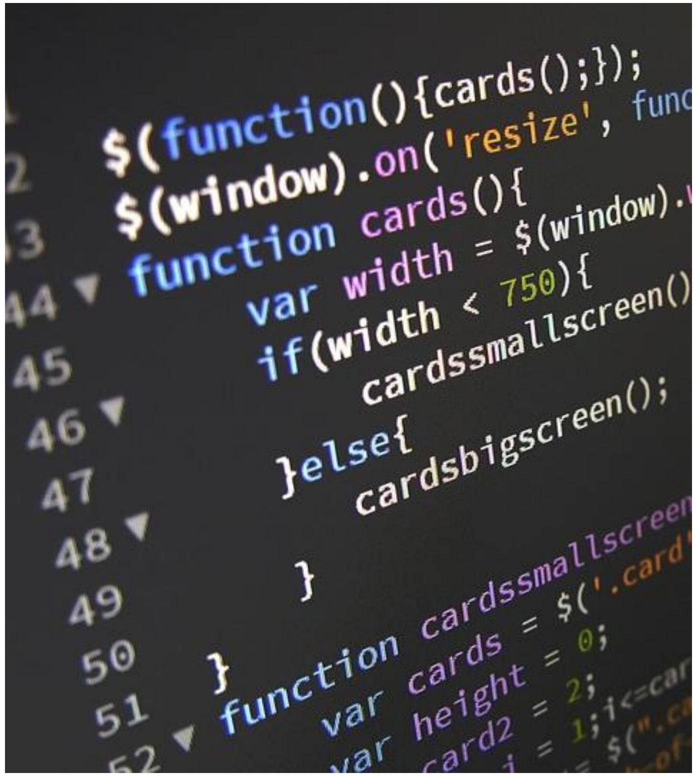


$$
\begin{aligned}
&\land \forall\, p \in \mathit{Proc}, d \in \mathit{Disk} \;:\\
&\quad \land\, (d \in \mathit{disksWritten}[p]) \Rightarrow \land\; \mathit{phase}[p] \in \{1, 2\}\\
&\qquad\qquad\qquad\qquad\qquad\qquad\;\; \land\; \mathit{disk}[d][p] = \mathit{dblock}[p]\\
&\quad \land\, (\mathit{phase}[p] \in 1, 2) \Rightarrow \land\, (\mathit{blocksRead}[p][d] \neq \{\}) \Rightarrow\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad (d \in \mathit{disksWritten}[p])\\
&\qquad\qquad\qquad\qquad\qquad\quad \land\, \neg \mathit{hasRead}(p, d, p)\\
&\land \forall\, p \in \mathit{Proc} \;:\\
&\quad \land\, (\mathit{phase}[p] = 0) \Rightarrow \land\; \mathit{dblock}[p] = \mathit{InitDB}\\
&\qquad\qquad\qquad\qquad\qquad \land\; \mathit{disksWritten}[p] = \{\}\\
&\qquad\qquad\qquad\qquad\qquad \land\, \forall\, d \in \mathit{Disk} \;:\; \forall\, \mathit{br} \in \mathit{blocksReac}\\
&\qquad\qquad\qquad\qquad\qquad\qquad\quad \land\; \mathit{br.proc} = p\\
&\qquad\qquad\qquad\qquad\qquad\qquad\quad \land\; \mathit{br.block} = \mathit{disk}[d][p]\\
&\quad \land\, (\mathit{phase}[p] \neq 0) \Rightarrow \land\; \mathit{dblock}[p].\mathit{mbal} \in \mathit{Ballot}(p)\\
&\qquad\qquad\qquad\qquad\qquad \land\; \mathit{dblock}[p].\mathit{bal} \in \mathit{Ballot}(p) \cup \{0\}\\
&\qquad\qquad\qquad\qquad\qquad \land\, \forall\, d \in \mathit{Disk} \;:\; \forall\, \mathit{br} \in \mathit{blocksReac}\\
&\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathit{br.block.mbal} < \mathit{dblock}[p].\\
&\quad \land\, (\mathit{phase}[p] \in \{2, 3\}) \Rightarrow (\mathit{dblock}[p].\mathit{bal} = \mathit{dblock}[p].\mathit{ml}\\
&\quad \land\; \mathit{output}[p] \;=\; \textsc{if}\; \mathit{phase}[p] = 3 \;\textsc{then}\; \mathit{dblock}[p].\mathit{inp}\;\; \textsc{e}\\
&\land\; \mathit{chosen} \in \mathit{allInput} \cup \{\mathit{NotAnInput}\}\\
&\land \forall\, p \in \mathit{Proc} \;:\; \land\; \mathit{input}[p] \in \mathit{allInput}\\
&\qquad\qquad\qquad\quad\;\; \land\, (\mathit{chosen} = \mathit{NotAnInput}) \Rightarrow (\mathit{output}[p]
\end{aligned}
$$

⇒



⇒



```
$(function(){cards();});
$(window).on('resize', func
function cards(){
    var width = $(window).
    if(width < 750){
        cardssmallscreen()
    }else{
        cardsbigscreen();
    }
}
function cardssmallscreen
    var cards = $('.card
    var height = 0;
    var card2 = 2;
    i = 1;i<=car
```

What is program synthesis?

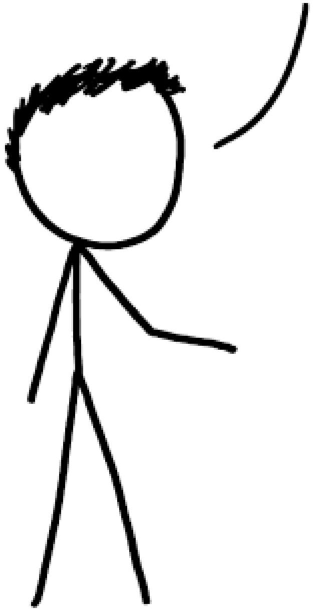
$$\overbrace{\exists p}^{\text{Find a program}} \in \underbrace{\mathcal{L}(G)}_{\text{From a language}}.\overbrace{\forall x.\phi(p,x)}^{\text{A correctness criterion}}$$

How do you find a program?

How do you express correctness?

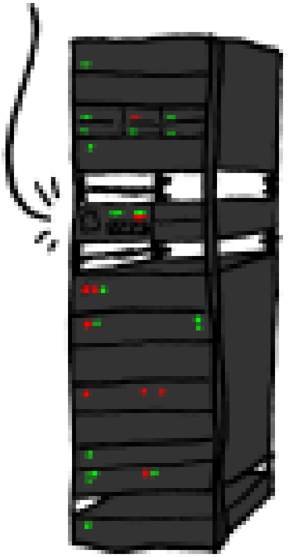
How do you express correctness?

Specification of intent – the olden days



$$
\begin{array}{l}
\land \forall p \in \mathit{Proc}, d \in \mathit{Disk} : \\
\quad \land (d \in \mathit{disksWritten}[p]) \Rightarrow \land \mathit{phase}[p] \in \{1,2\} \\
\qquad\qquad\qquad\qquad\qquad\quad\;\; \land \mathit{disk}[d][p] = \mathit{dblock}[p] \\
\quad \land (\mathit{phase}[p] \in 1,2) \Rightarrow \land (\mathit{blocksRead}[p][d] \neq \{\}) \Rightarrow \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad (d \in \mathit{disksWritten}[p]) \\
\qquad\qquad\qquad\qquad\quad\;\; \land \neg \mathit{hasRead}(p, d, p) \\
\land \forall p \in \mathit{Proc} : \\
\quad \land (\mathit{phase}[p] = 0) \Rightarrow \land \mathit{dblock}[p] = \mathit{InitDB} \\
\qquad\qquad\qquad\qquad\quad \land \mathit{disksWritten}[p] = \{\} \\
\qquad\qquad\qquad\qquad\quad \land \forall d \in \mathit{Disk} : \forall br \in \mathit{blocksRea} \\
\qquad\qquad\qquad\qquad\qquad\quad \land br.\mathit{proc} = p \\
\qquad\qquad\qquad\qquad\qquad\quad \land br.\mathit{block} = \mathit{disk}[d][p] \\
\quad \land (\mathit{phase}[p] \neq 0) \Rightarrow \land \mathit{dblock}[p].\mathit{mbal} \in \mathit{Ballot}(p) \\
\qquad\qquad\qquad\qquad\quad \land \mathit{dblock}[p].\mathit{bal} \in \mathit{Ballot}(p) \cup \{0\} \\
\qquad\qquad\qquad\qquad\quad \land \forall d \in \mathit{Disk} : \forall br \in \mathit{blocksRea} \\
\qquad\qquad\qquad\qquad\qquad\quad br.\mathit{block}.\mathit{mbal} < \mathit{dblock}[p]. \\
\quad \land (\mathit{phase}[p] \in \{2,3\}) \Rightarrow (\mathit{dblock}[p].\mathit{bal} = \mathit{dblock}[p].\mathit{m} \\
\quad \land \mathit{output}[p] = \text{IF } \mathit{phase}[p] = 3 \text{ THEN } \mathit{dblock}[p].\mathit{inp} \text{ E} \\
\land \mathit{chosen} \in \mathit{allInput} \cup \{\mathit{NotAnInput}\} \\
\land \forall p \in \mathit{Proc} : \land \mathit{input}[p] \in \mathit{allInput} \\
\qquad\qquad\qquad\quad\;\; \land (\mathit{chosen} = \mathit{NotAnInput}) \Rightarrow (\mathit{output}[p]
\end{array}
$$

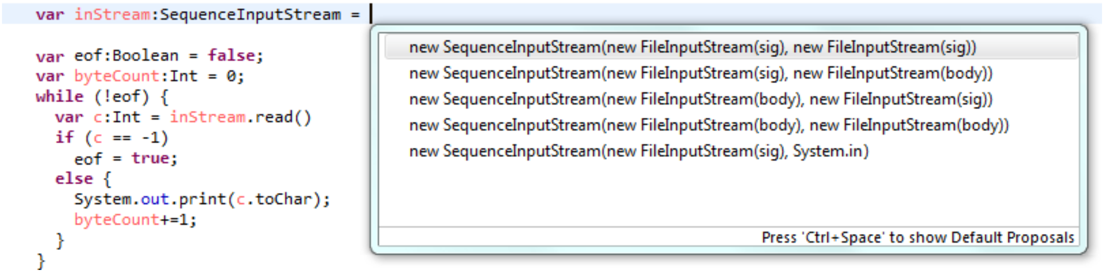
Right-O, here's some code:



Specification of *partial* intent

$$\mathcal{E} = \{\iota_i \rightarrow \omega_i\}$$

Examples

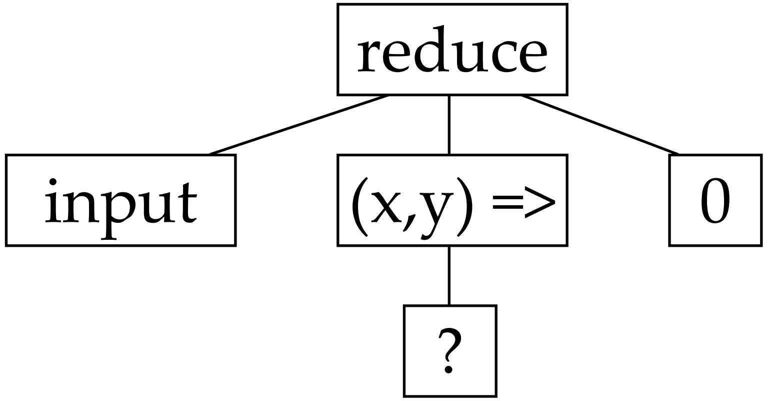


```
var inStream:SequenceInputStream = 

var eof:Boolean = false;
var byteCount:Int = 0;
while (!eof) {
  var c:Int = inStream.read()
  if (c == -1)
    eof = true;
  else {
    System.out.print(c.toChar);
    byteCount+=1;
  }
}
```

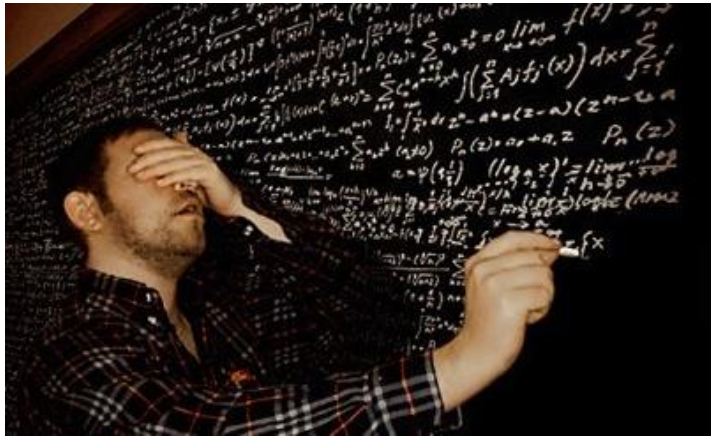
```
new SequenceInputStream(new FileInputStream(sig), new FileInputStream(sig))
new SequenceInputStream(new FileInputStream(sig), new FileInputStream(body))
new SequenceInputStream(new FileInputStream(body), new FileInputStream(sig))
new SequenceInputStream(new FileInputStream(body), new FileInputStream(body))
new SequenceInputStream(new FileInputStream(sig), System.in)
```

Press 'Ctrl+Space' to show Default Proposals

Types



Sketching



Logical specifications

Programming by Example

$$\phi(p, x) = (x = \iota_1 \Rightarrow [\![p]\!](x) = \omega_1) \wedge (x = \iota_2 \Rightarrow [\![p]\!](x) = \omega_2) \wedge \cdots$$

or in other words

$$\mathcal{E} = \{\iota_i \rightarrow \omega_i\}$$

values of all the variables/the
state of the environment

output of an expression/an effect/
the new state of the environment

Type-Driven Synthesis

$$(\tau_1, \tau_2, \ldots, \tau_k) \rightarrow \tau_{out}$$

or in other words,

“Use variables of these types that are in scope to make something of this type”

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

(Fancy-)Type-Driven Synthesis

$$(\tau_1, \tau_2, \dots, \tau_k) \rightarrow \tau_{out}$$

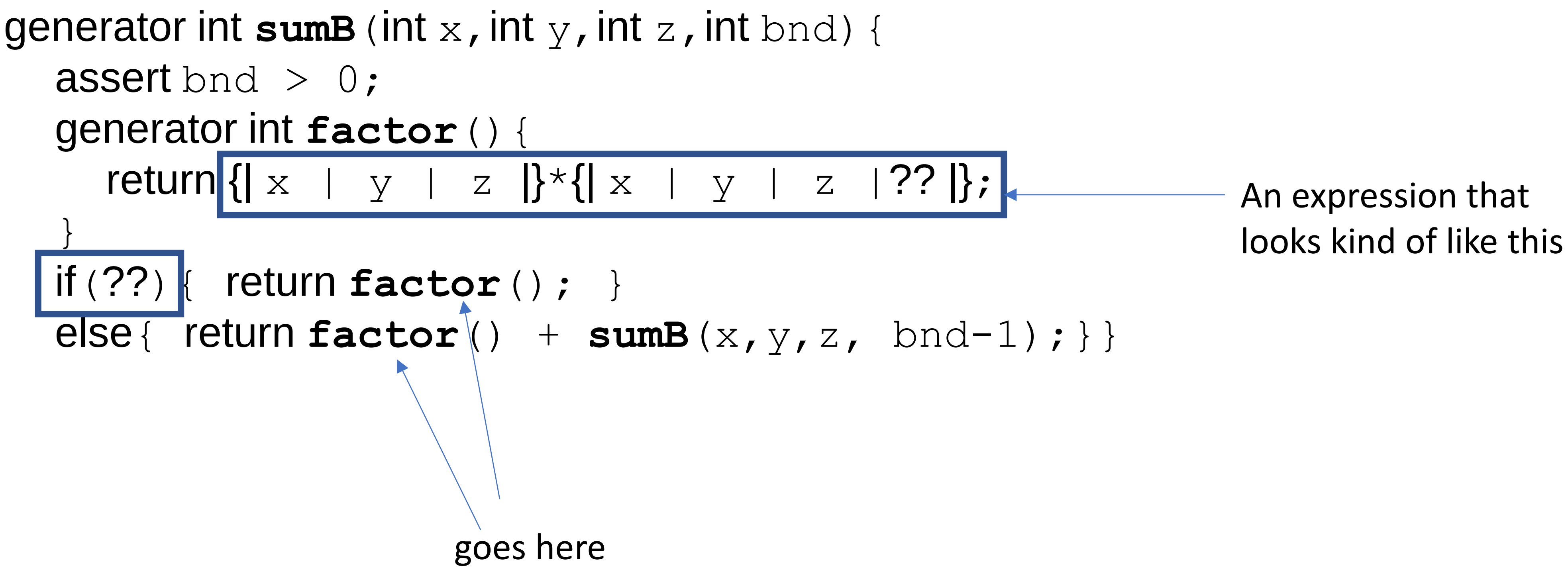
or in other words,

"Use variables of these types that are in scope to make something of this type"

```
n:Nat -> x:a -> {List a | len _v = n}
```

Sketching

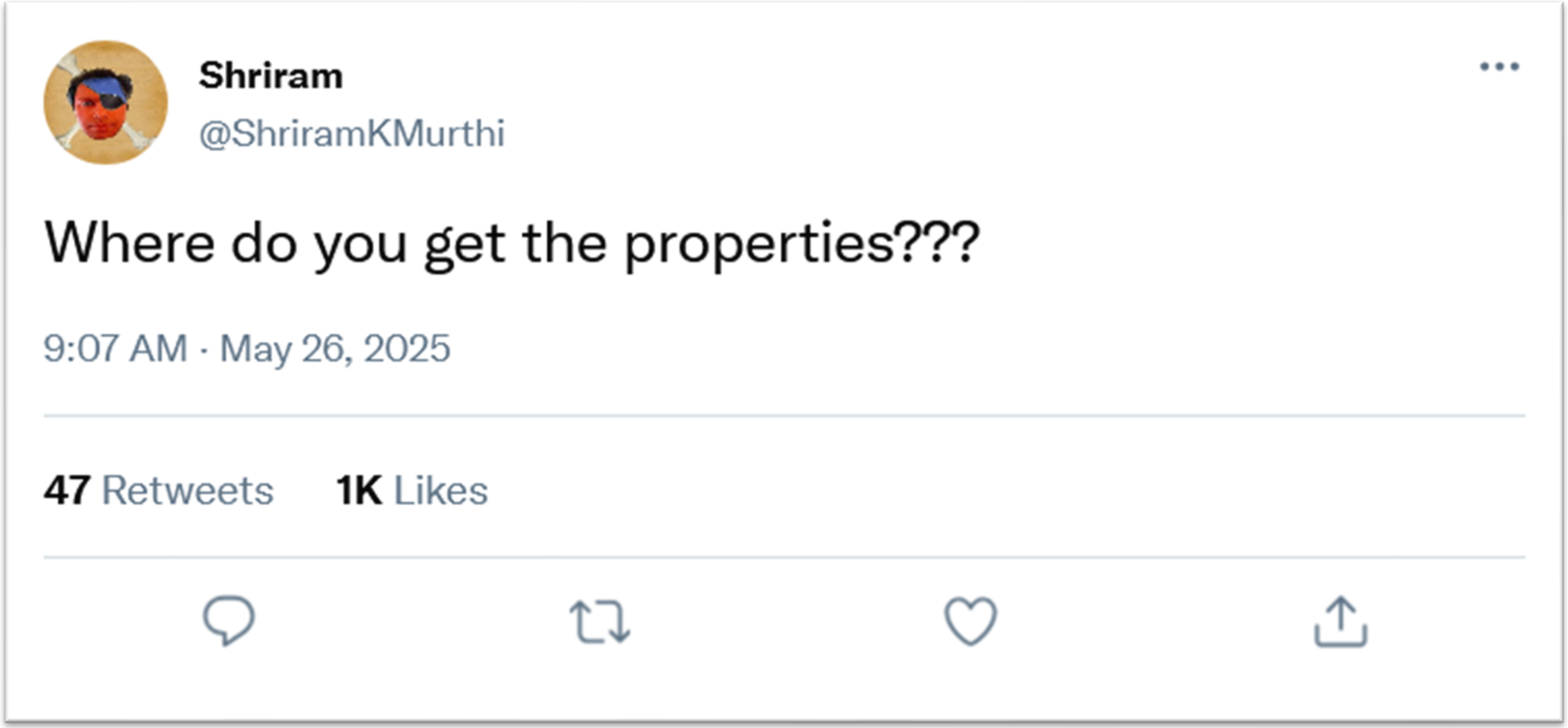
“I already know some of the code for my program”



```
generator int sumB(int x, int y, int z, int bnd) {
    assert bnd > 0;
    generator int factor() {
        return {| x | y | z |}*{| x | y | z |?? |};
    }
    if(??){ return factor(); }
    else{ return factor() + sumB(x,y,z, bnd-1);}}
```

An expression that looks kind of like this

goes here



Shriram
@ShriramKMurthi

Where do you get the properties???

9:07 AM · May 26, 2025

47 Retweets 1K Likes

(next time, I promise)

How do you find a program?

Generic synthesis recipe

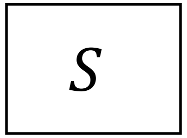
1. Generate a candidate program

- Enumerate trees
  - Top-down
  - Bottom-up
- Traverse automata
- Graph reachability
- Enumerate deduction rules

2. Test against specification

- Run tests
  - Examples
  - Unit
- Encode for SMT solver
- Apply typing rules

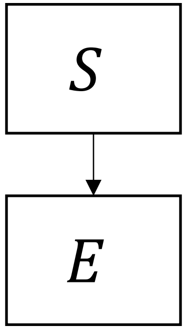
Enumerating trees



S

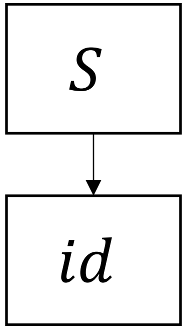
$$S \rightarrow E$$
$$E \rightarrow E + E$$
$$E \rightarrow E\ *\ E$$
$$E \rightarrow [EList]$$
$$EList \rightarrow E \mid EList, E$$
$$E \rightarrow len(E)$$
$$E \rightarrow id$$
$$E \rightarrow num$$

Enumerating trees



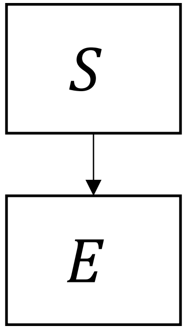
$S \to E$
$E \to E + E$
$E \to E \ * \ E$
$E \to [EList]$
$EList \to E \mid EList, E$
$E \to len(E)$
$E \to id$
$E \to num$

Enumerating trees



$$
\begin{aligned}
&S \to E \\
&E \to E + E \\
&E \to E \ * \ E \\
&E \to [EList] \\
&EList \to E \mid EList, E \\
&E \to len(E) \\
&E \to id \\
&E \to num
\end{aligned}
$$

Enumerating trees



$S \rightarrow E$

$E \rightarrow E + E$

$E \rightarrow E \ * \ E$

$E \rightarrow [EList]$

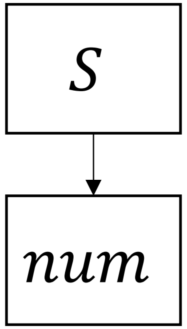
$EList \rightarrow E \mid EList, E$

$E \rightarrow len(E)$

$E \rightarrow id$

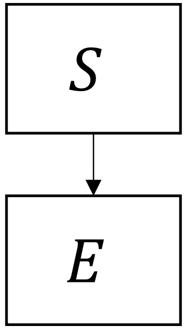
$E \rightarrow num$

Enumerating trees



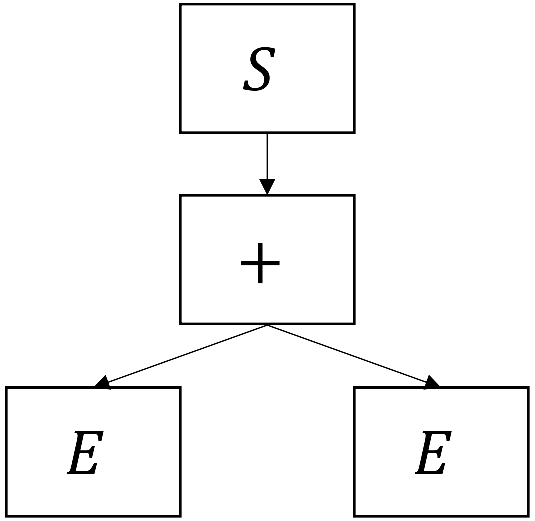
$S \to E$
$E \to E + E$
$E \to E \, * \, E$
$E \to [EList]$
$EList \to E \mid EList, E$
$E \to len(E)$
$E \to id$
$E \to num$

Enumerating trees



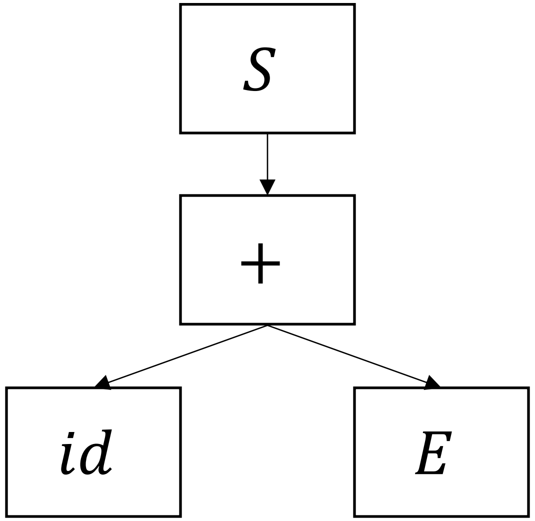
$S \rightarrow E$
$E \rightarrow E + E$
$E \rightarrow E \; * \; E$
$E \rightarrow [EList]$
$EList \rightarrow E \mid EList, E$
$E \rightarrow len(E)$
$E \rightarrow id$
$E \rightarrow num$

Enumerating trees



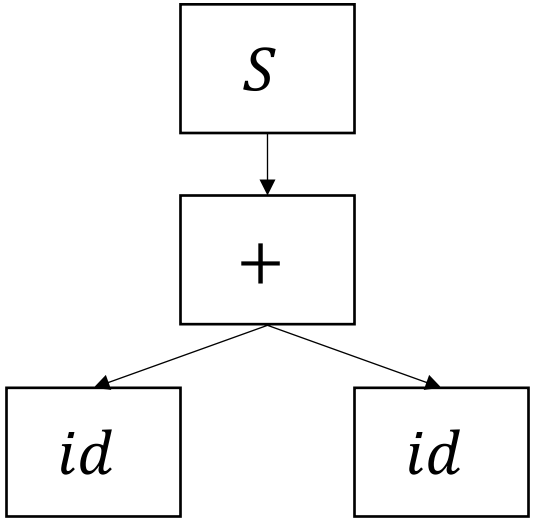
$S \to E$
$E \to E + E$
$E \to E \ * \ E$
$E \to [EList]$
$EList \to E \mid EList, E$
$E \to len(E)$
$E \to id$
$E \to num$

Enumerating trees



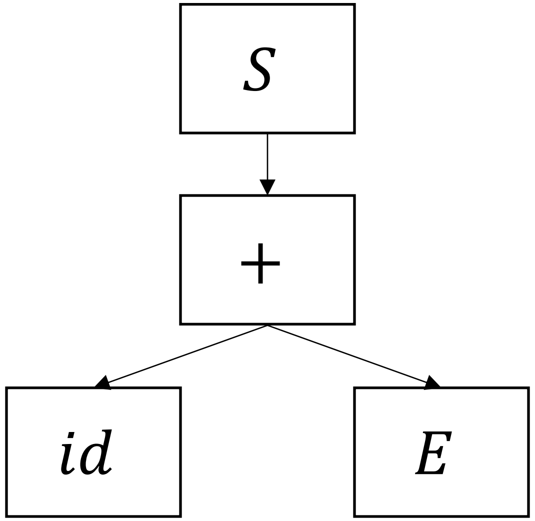
$S \rightarrow E$
$E \rightarrow E + E$
$E \rightarrow E \ * \ E$
$E \rightarrow [EList]$
$EList \rightarrow E \mid EList, E$
$E \rightarrow len(E)$
$E \rightarrow id$
$E \rightarrow num$

Enumerating trees



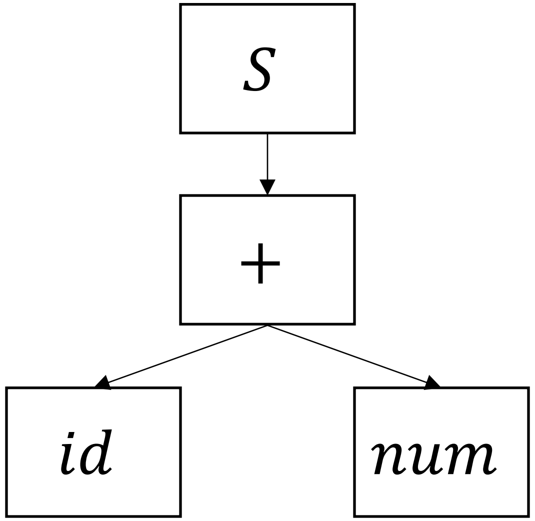
$S \rightarrow E$
$E \rightarrow E + E$
$E \rightarrow E \ * \ E$
$E \rightarrow [EList]$
$EList \rightarrow E \mid EList, E$
$E \rightarrow len(E)$
$E \rightarrow id$
$E \rightarrow num$

Enumerating trees



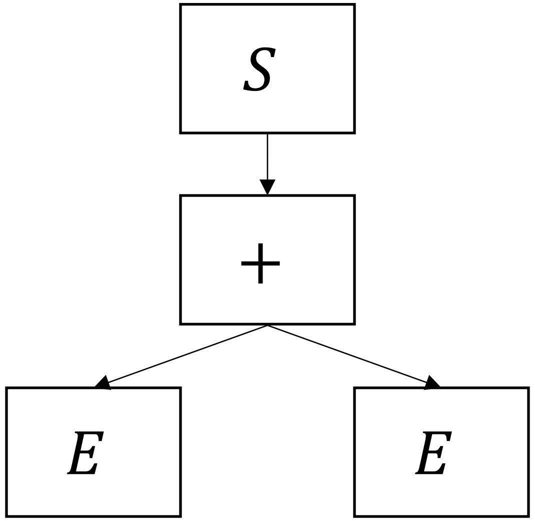
$$
\begin{aligned}
&S \to E\\
&E \to E + E\\
&E \to E\ *\ E\\
&E \to [EList]\\
&EList \to E \mid EList, E\\
&E \to len(E)\\
&E \to id\\
&E \to num
\end{aligned}
$$

Enumerating trees



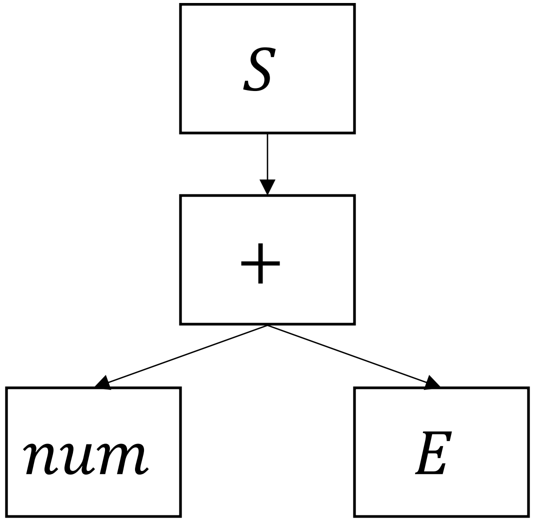
$$S \rightarrow E$$
$$E \rightarrow E + E$$
$$E \rightarrow E \ * \ E$$
$$E \rightarrow [EList]$$
$$EList \rightarrow E \mid EList, E$$
$$E \rightarrow len(E)$$
$$E \rightarrow id$$
$$E \rightarrow num$$

Enumerating trees



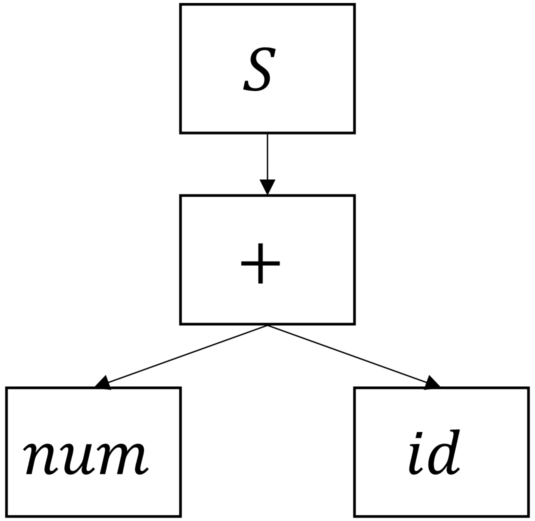
$S \rightarrow E$
$E \rightarrow E + E$
$E \rightarrow E\ *\ E$
$E \rightarrow [EList]$
$EList \rightarrow E \mid EList, E$
$E \rightarrow len(E)$
$E \rightarrow id$
$E \rightarrow num$

Enumerating trees



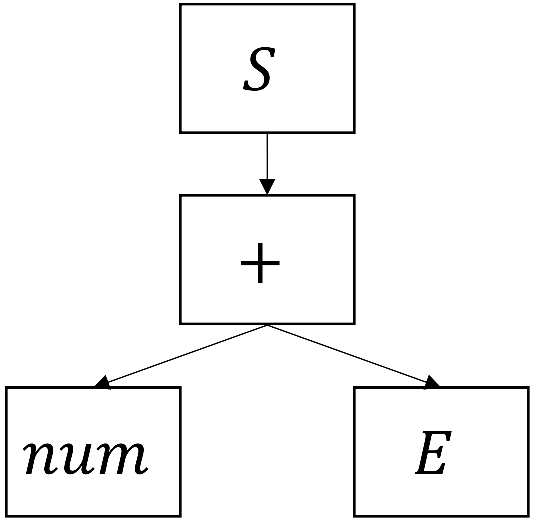
$S \to E$
$E \to E + E$
$E \to E \; * \; E$
$E \to [EList]$
$EList \to E \mid EList, E$
$E \to len(E)$
$E \to id$
$E \to num$

Enumerating trees



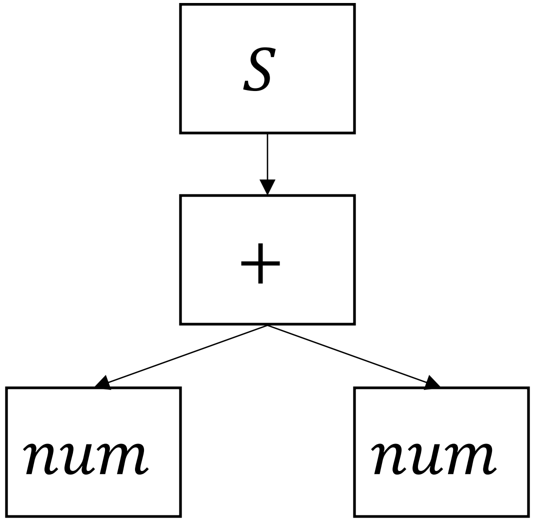
$$S \rightarrow E$$
$$E \rightarrow E + E$$
$$E \rightarrow E\ *\ E$$
$$E \rightarrow [EList]$$
$$EList \rightarrow E \mid EList, E$$
$$E \rightarrow len(E)$$
$$E \rightarrow id$$
$$E \rightarrow num$$

Enumerating trees



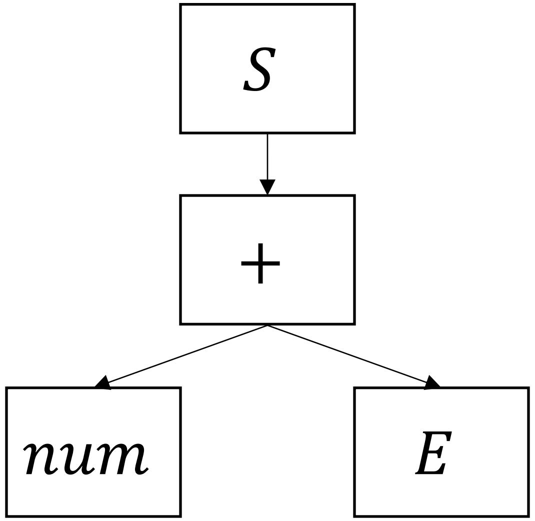
$S \to E$
$E \to E + E$
$E \to E \ * \ E$
$E \to [EList]$
$EList \to E \mid EList, E$
$E \to len(E)$
$E \to id$
$E \to num$

Enumerating trees



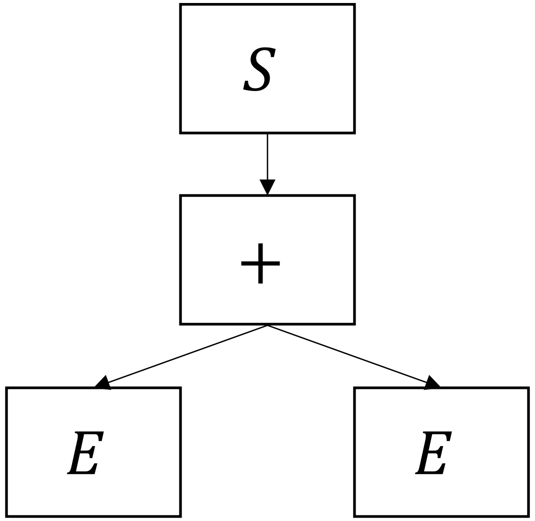
$$
\begin{aligned}
&S \rightarrow E\\
&E \rightarrow E + E\\
&E \rightarrow E \, * \, E\\
&E \rightarrow [EList]\\
&EList \rightarrow E \mid EList, E\\
&E \rightarrow len(E)\\
&E \rightarrow id\\
&E \rightarrow num
\end{aligned}
$$

Enumerating trees



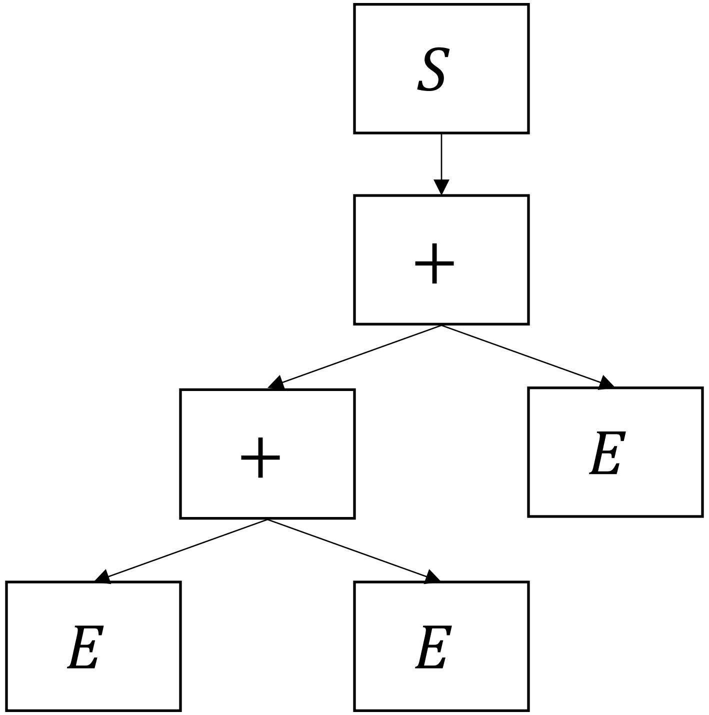
$S \rightarrow E$
$E \rightarrow E + E$
$E \rightarrow E \, * \, E$
$E \rightarrow [EList]$
$EList \rightarrow E \mid EList, E$
$E \rightarrow len(E)$
$E \rightarrow id$
$E \rightarrow num$

Enumerating trees



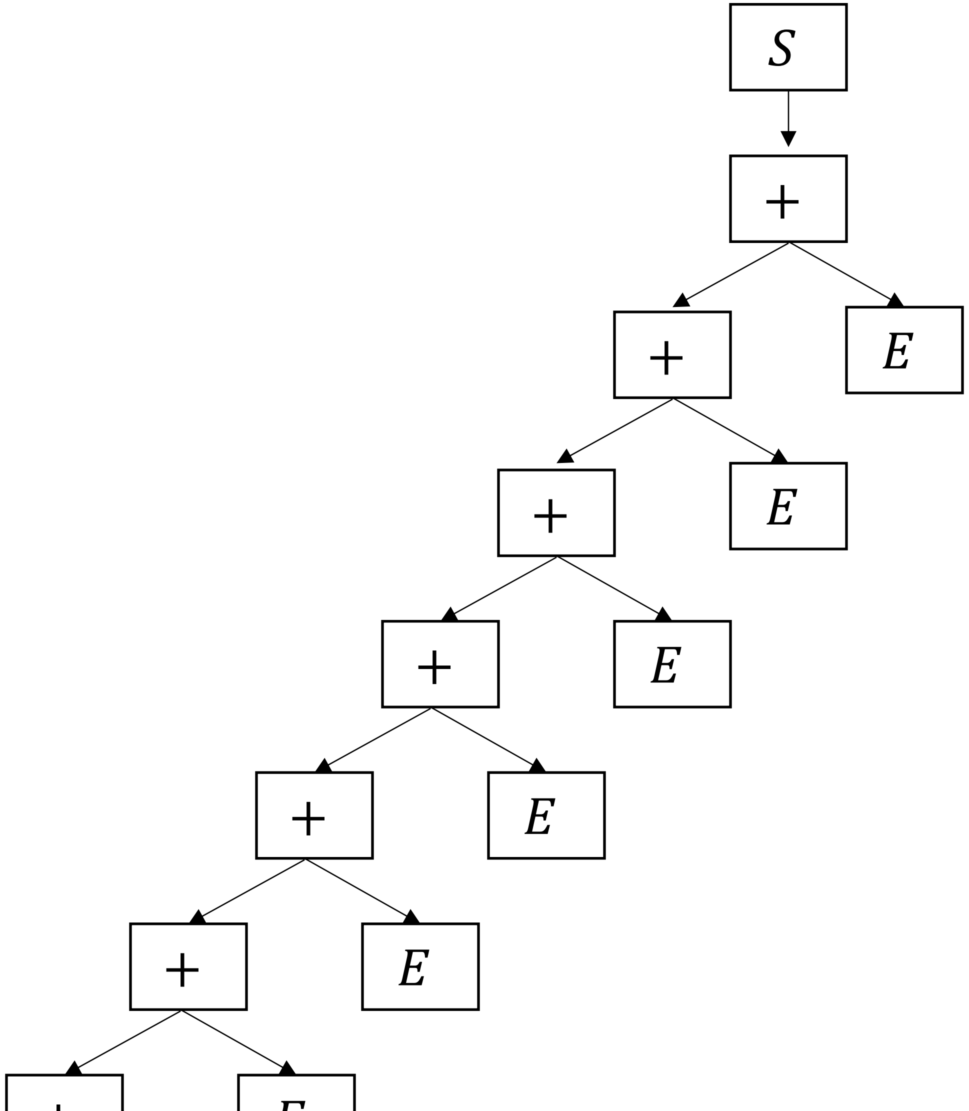
$S \rightarrow E$
$E \rightarrow E + E$
$E \rightarrow E * E$
$E \rightarrow [EList]$
$EList \rightarrow E \mid EList, E$
$E \rightarrow len(E)$
$E \rightarrow id$
$E \rightarrow num$

Enumerating trees



$S \to E$
$E \to E + E$
$E \to E \; * \; E$
$E \to [EList]$
$EList \to E \mid EList, E$
$E \to len(E)$
$E \to id$
$E \to num$

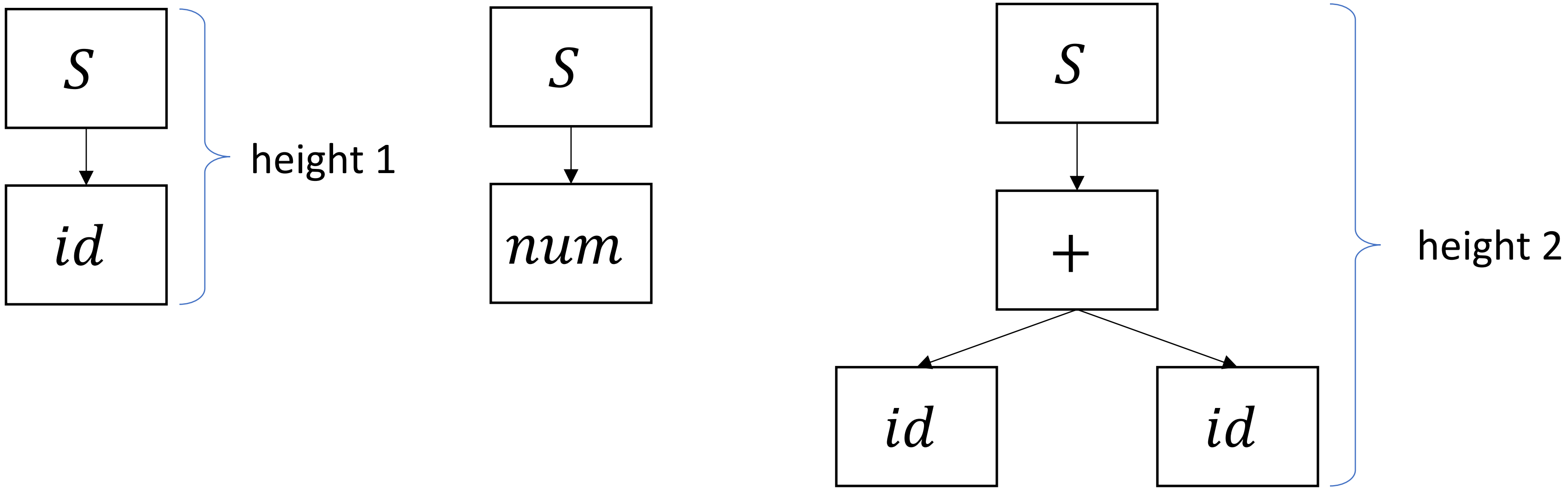
Enumeration stack overflow



$S \rightarrow E$
$E \rightarrow E + E$
$E \rightarrow E \ * \ E$
$E \rightarrow [EList]$
$EList \rightarrow E \mid EList, E$
$E \rightarrow len(E)$
$E \rightarrow id$
$E \rightarrow num$

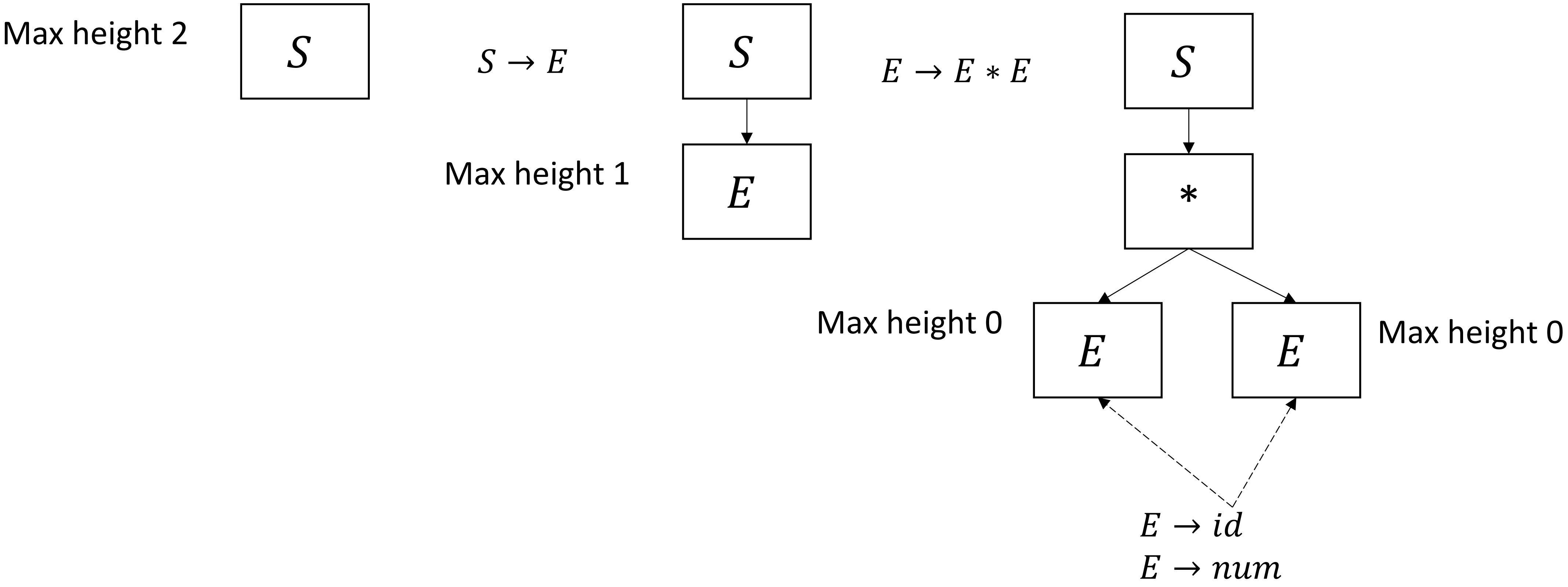
Solution: iterative deepening

- Limit the height of programs being enumerated
- First, all programs of height 0
- then 1
- then 2…

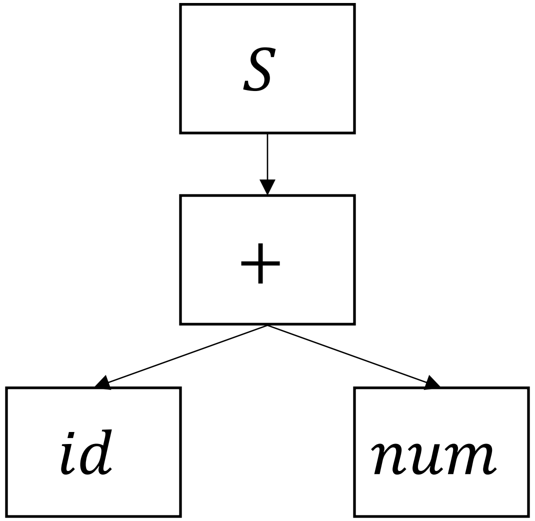


Iterative deepening

To enumerate height 2:

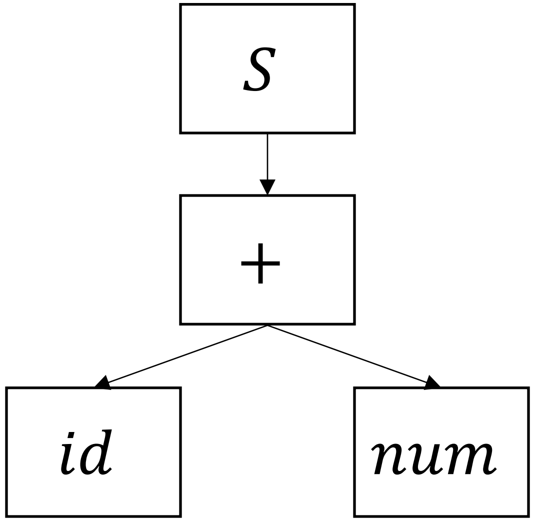


An infinite space

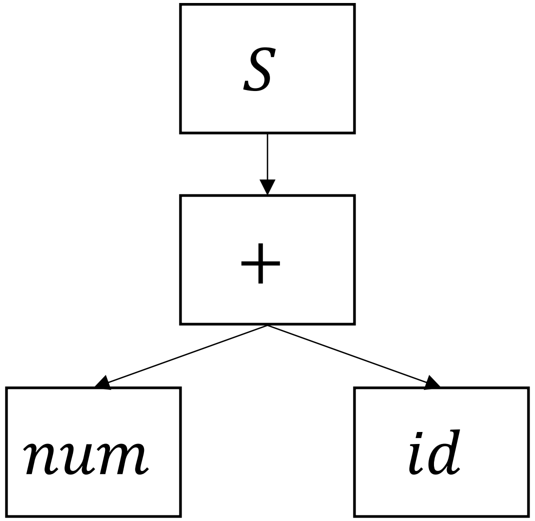


```
x  +  2
x  +  3
x  +  42
y  +  2
y  +  -3
```

An infinite space full of redundancy

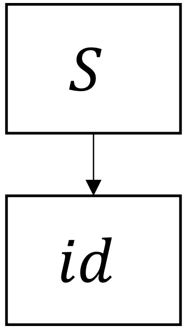


x + 2

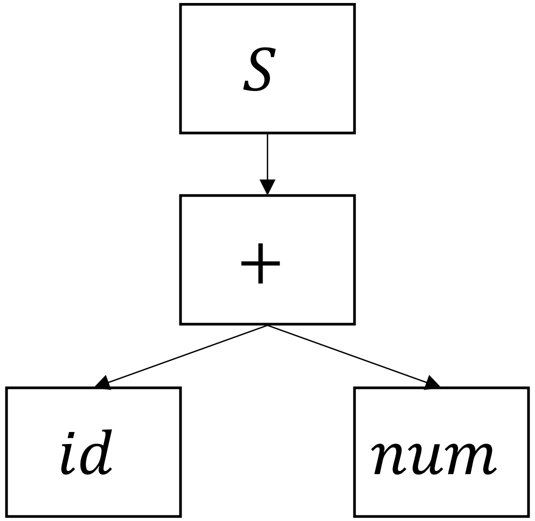


2 + x

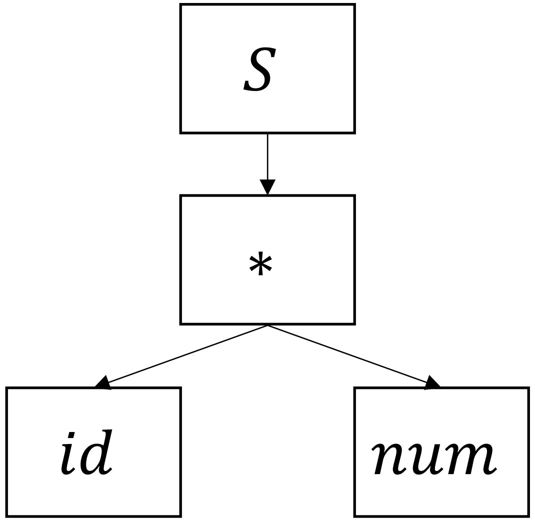
An infinite space full of redundancy



x

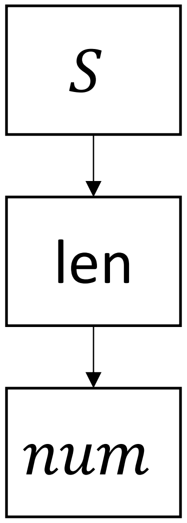


x + 0

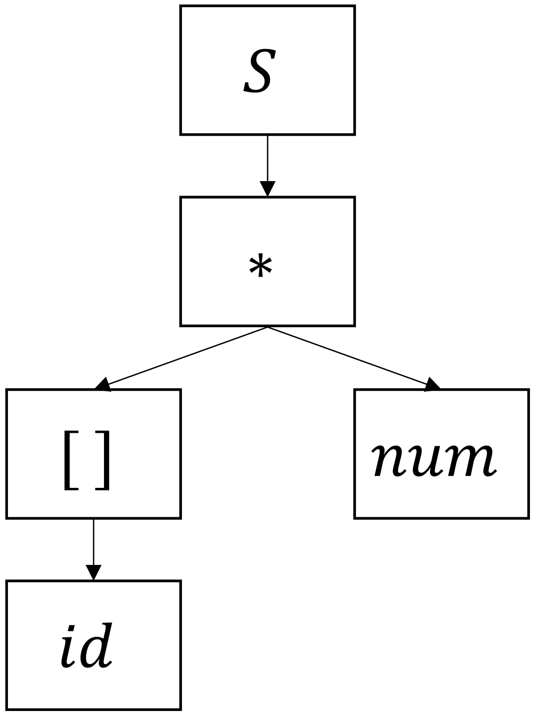


x * 1

An infinite space full of bad programs



```
len(2)
```



```
[x]   *   1
```

Maybe enumerating the space in full is bad, actually?

Pruning the space 1: well-typedness

$$\frac{e_1 : int \quad e_2 : int}{e_1 + e_2 : int} \qquad \frac{e_1 : [\tau] \quad e_2 : [\tau]}{e_1 + e_2 : [\tau]}$$

$$\frac{e_1 : int \quad e_2 : int}{e_1 * e_2 : int} \qquad \frac{e_1, \ldots, e_k : \tau}{[e_1, \ldots, e_k] : [\tau]}$$

$$\frac{e : [\tau]}{len(e) : int} \qquad \frac{var\ x\ has\ type\ \tau}{x : \tau}$$

$$\frac{}{num : int}$$

$$\begin{aligned}
&S \rightarrow E \\
&E \rightarrow E + E \\
&E \rightarrow E\ *\ E \\
&E \rightarrow [EList] \\
&EList \rightarrow \varepsilon \mid EList, E \\
&E \rightarrow len(E) \\
&E \rightarrow id \\
&E \rightarrow num
\end{aligned}$$

Generic synthesis recipe

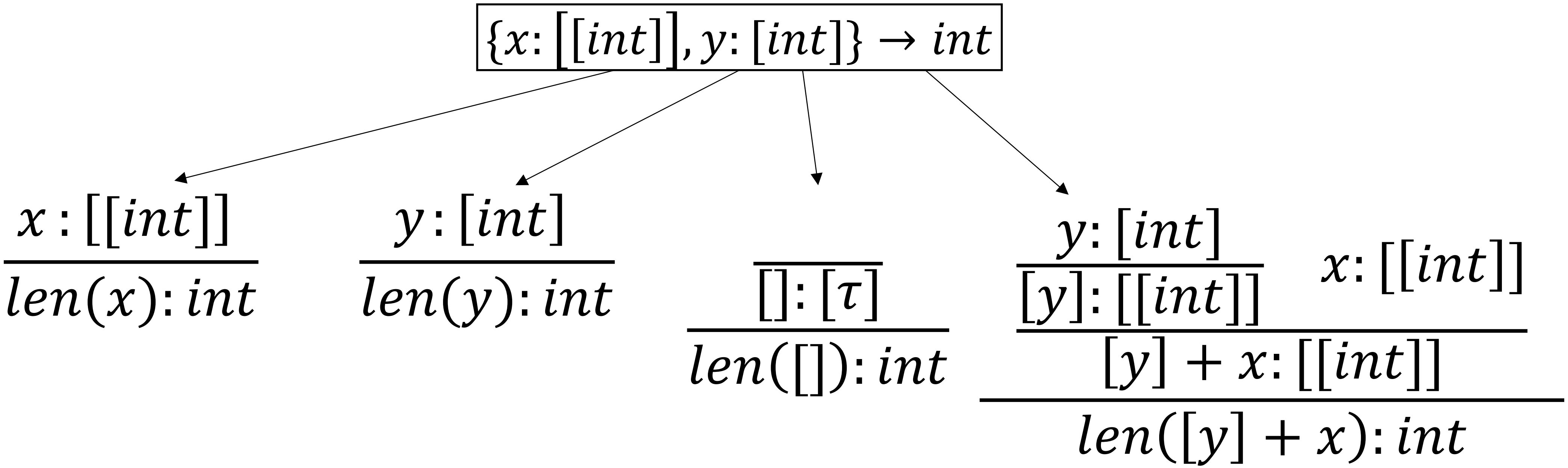
1. Generate a candidate program

- Enumerate trees
  - Top-down
  - Bottom-up
- Traverse automata
- Graph reachability
- Enumerate deduction rules
- Cheat* by looking at spec

2. Test against specification

- Run tests
  - Examples
  - Unit
- Encode for SMT solver
- Apply typing rules

Looking at spec with types



$$\boxed{\{x: [[int]], y: [int]\} \to int}$$

$$\frac{x : [[int]]}{len(x): int}$$

$$\frac{y: [int]}{len(y): int}$$

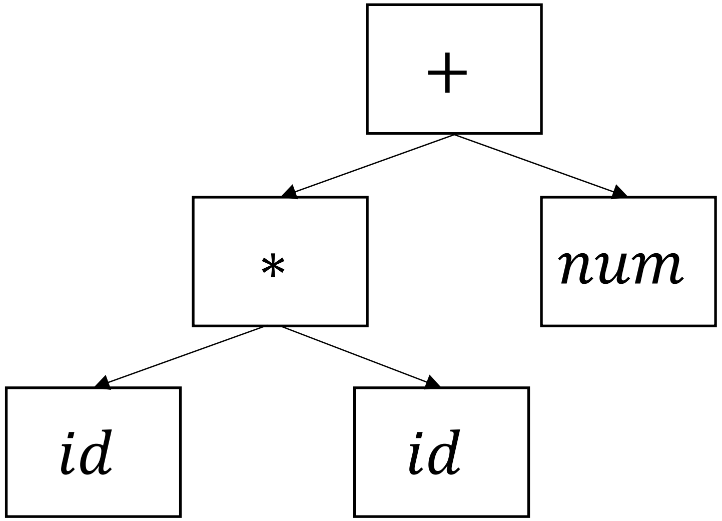
$$\frac{\overline{[]: [\tau]}}{len([]): int}$$

$$\frac{\dfrac{\dfrac{y: [int]}{[y]: [[int]]} \qquad x: [[int]]}{[y] + x: [[int]]}}{len([y] + x): int}$$

Deductive proof search

What about examples?

What we enumerated was not a concrete program



Can we combine that with examples?

What about examples?

What we enumerated was not a concrete program

Given:

$$\epsilon_1\text{: } \{x \mapsto 6, y \mapsto 8\} \to 49$$
$$\epsilon_2\text{:} \{x \mapsto 4, y \mapsto 7\} \to 29$$

And a (meta) candidate: `(id`$_1$` * id`$_2$`) + num`

Can we solve for id_1, id_2, num?

SMT solvers to the rescue

Given:

$$\epsilon_1\colon \{x \mapsto 6, y \mapsto 8\} \to 49$$
$$\epsilon_2\colon \{x \mapsto 4, y \mapsto 7\} \to 29$$

And a (meta) candidate: $(\texttt{id}_1\ *\ \texttt{id}_2)\ +\ \texttt{num}$

Can we solve for id_1, id_2, num?

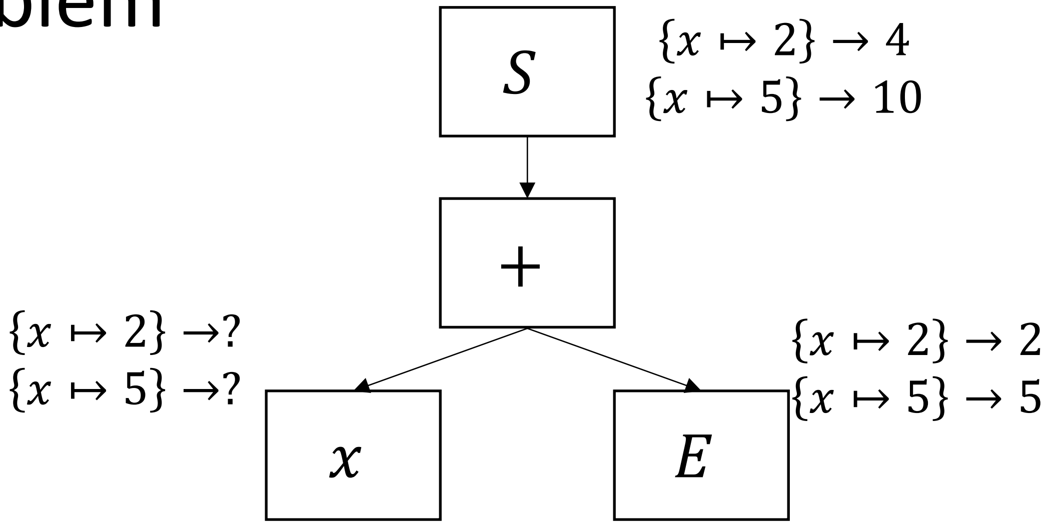
$$\forall arr \in \overset{x}{\overset{\downarrow}{\mathbb{Z}}} \times \overset{y}{\overset{\downarrow}{\mathbb{Z}}}. (arr = [6{,}8] \Rightarrow (arr[i_1] * arr[i_2]) + num = 49) \wedge$$
$$(arr = [4{,}7] \Rightarrow (arr[i_1] * arr[i_2]) + num = 29)$$

Model: $i_1 = 1, i_2 = 0, num = 1$

free variables

Can we do better?

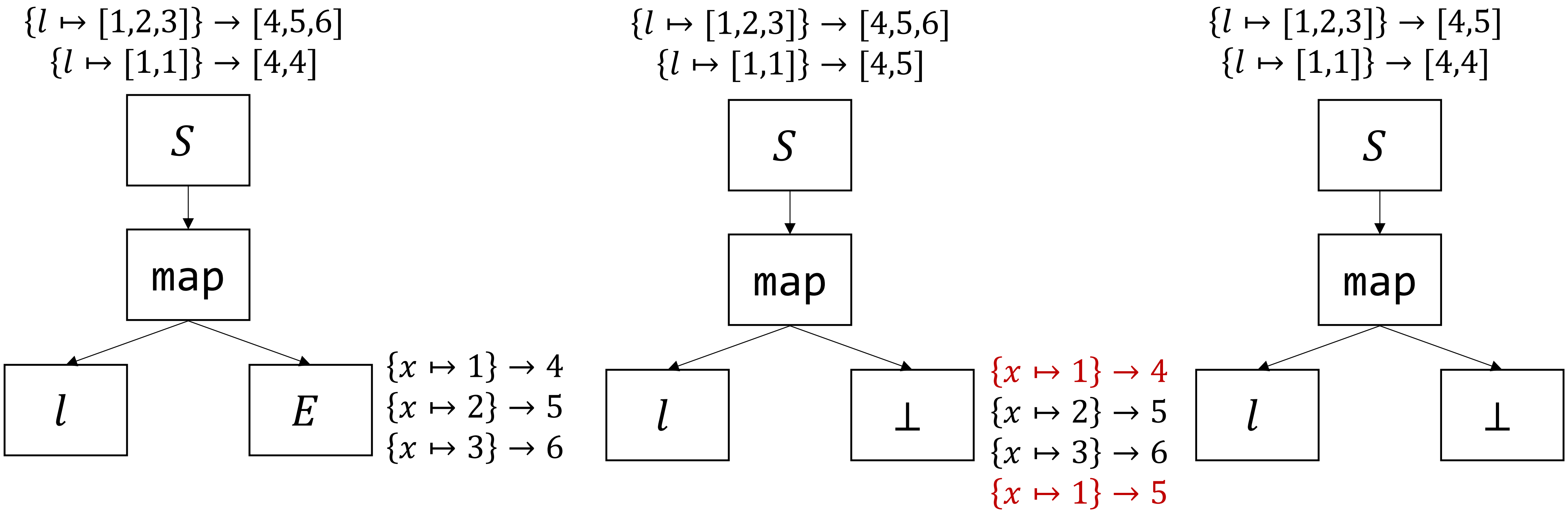
- Specification x Semantics of node = Specifications for remaining problem
  - … sometimes



S
$\{x \mapsto 2\} \rightarrow 4$
$\{x \mapsto 5\} \rightarrow 10$
$+$
$\{x \mapsto 2\} \rightarrow ?$
$\{x \mapsto 5\} \rightarrow ?$
x
E
$\{x \mapsto 2\} \rightarrow 2$
$\{x \mapsto 5\} \rightarrow 5$

- Called *Example Refinement*

Refinements and higher-order functions



$\{l \mapsto [1,2,3]\} \rightarrow [4,5,6]$
$\{l \mapsto [1,1]\} \rightarrow [4,4]$

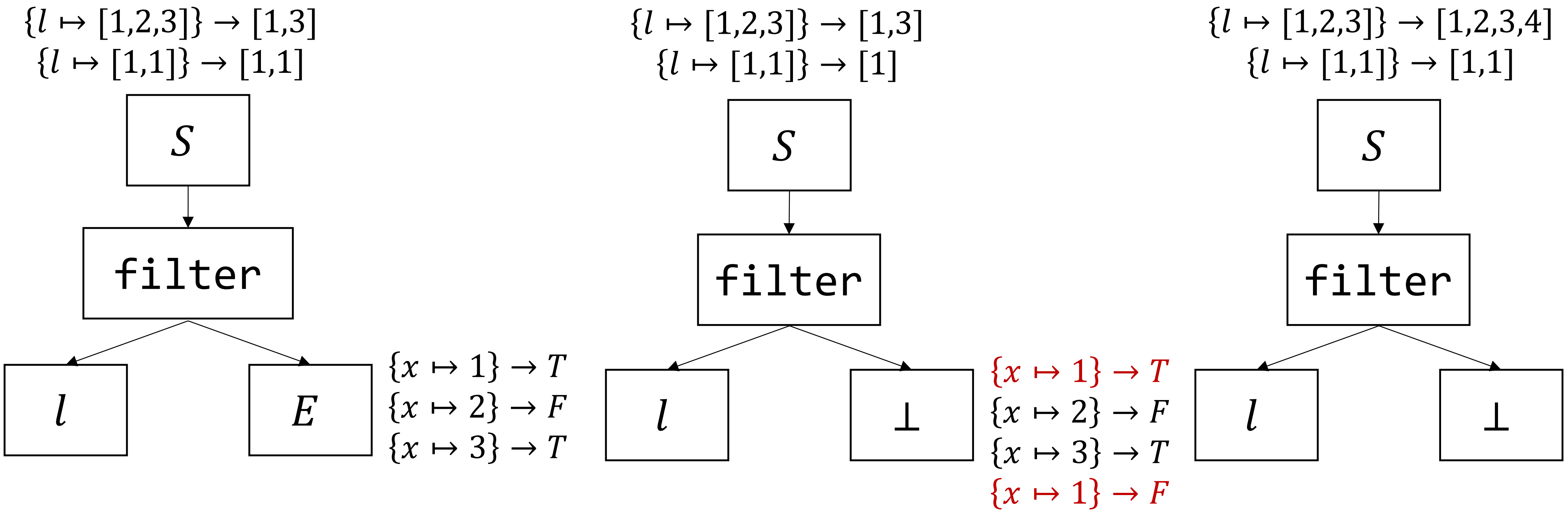
$\{x \mapsto 1\} \rightarrow 4$
$\{x \mapsto 2\} \rightarrow 5$
$\{x \mapsto 3\} \rightarrow 6$

$\{l \mapsto [1,2,3]\} \rightarrow [4,5,6]$
$\{l \mapsto [1,1]\} \rightarrow [4,5]$

$\{x \mapsto 1\} \rightarrow 4$
$\{x \mapsto 2\} \rightarrow 5$
$\{x \mapsto 3\} \rightarrow 6$
$\{x \mapsto 1\} \rightarrow 5$

$\{l \mapsto [1,2,3]\} \rightarrow [4,5]$
$\{l \mapsto [1,1]\} \rightarrow [4,4]$

Refinements and higher-order functions



$\{l \mapsto [1,2,3]\} \rightarrow [1,3]$

$\{l \mapsto [1,1]\} \rightarrow [1,1]$

S

filter

l

E

$\{x \mapsto 1\} \rightarrow T$

$\{x \mapsto 2\} \rightarrow F$

$\{x \mapsto 3\} \rightarrow T$

$\{l \mapsto [1,2,3]\} \rightarrow [1,3]$

$\{l \mapsto [1,1]\} \rightarrow [1]$

S

filter

l

$\bot$

$\{x \mapsto 1\} \rightarrow T$

$\{x \mapsto 2\} \rightarrow F$

$\{x \mapsto 3\} \rightarrow T$

$\{x \mapsto 1\} \rightarrow F$

$\{l \mapsto [1,2,3]\} \rightarrow [1,2,3,4]$

$\{l \mapsto [1,1]\} \rightarrow [1,1]$

S

filter

l

$\bot$

Generic synthesis recipe

1. Generate a candidate program

- Enumerate trees
  - Top-down
  - Bottom-up
- Traverse automata
- Graph reachability
- Enumerate deduction rules

2. Test against specification

- Run tests
  - Examples
  - Unit
- Encode for SMT solver
- Apply typing rules

Enumerative Search

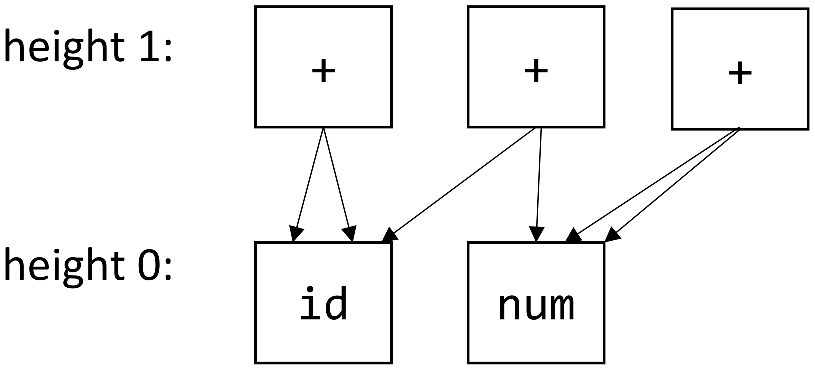
We did great with: Generate programs from the grammar, one by one, and test them on the specification

Enumerating trees

$$S \rightarrow E$$
$$E \rightarrow E + E$$
$$E \rightarrow E \;*\; E$$
$$E \rightarrow [EList]$$
$$EList \rightarrow E \mid EList, E$$
$$E \rightarrow len(E)$$
$$E \rightarrow id$$
$$E \rightarrow num$$

height 0: `id` `num`

Enumerating trees



$S \to E$

$\boxed{E \to E + E}$

$E \to E \, * \, E$

$E \to [EList]$

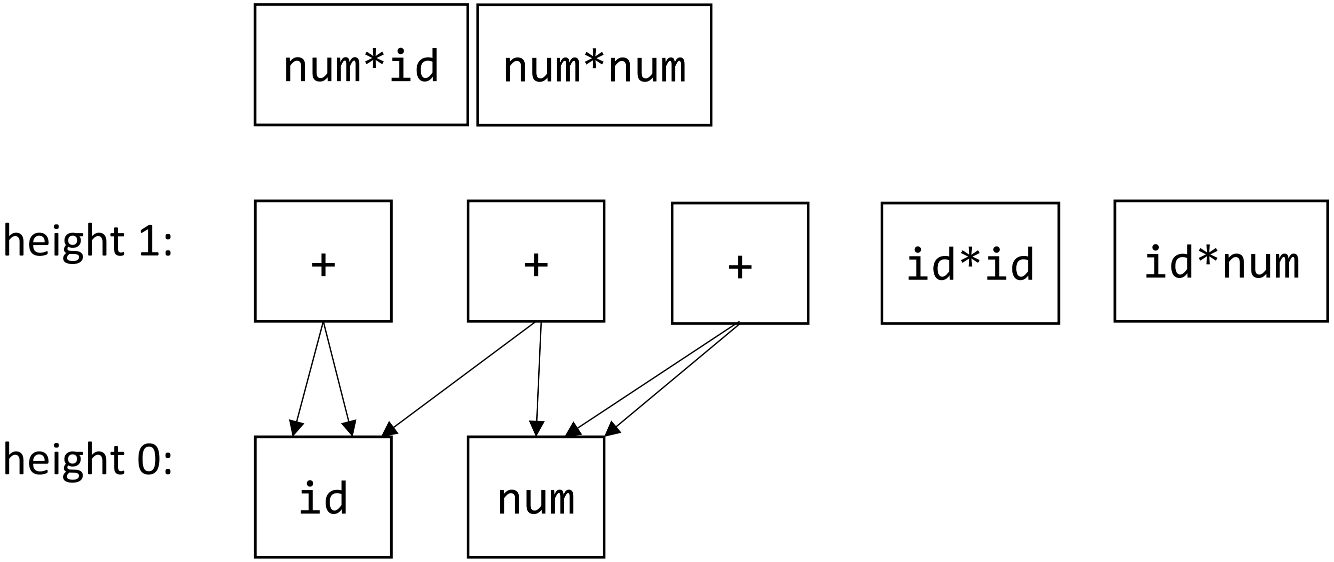
$EList \to E \mid EList, E$

$E \to len(E)$

$E \to id$

$E \to num$

Enumerating trees



$S \rightarrow E$

$E \rightarrow E + E$

$\boxed{E \rightarrow E \; * \; E}$

$E \rightarrow [EList]$

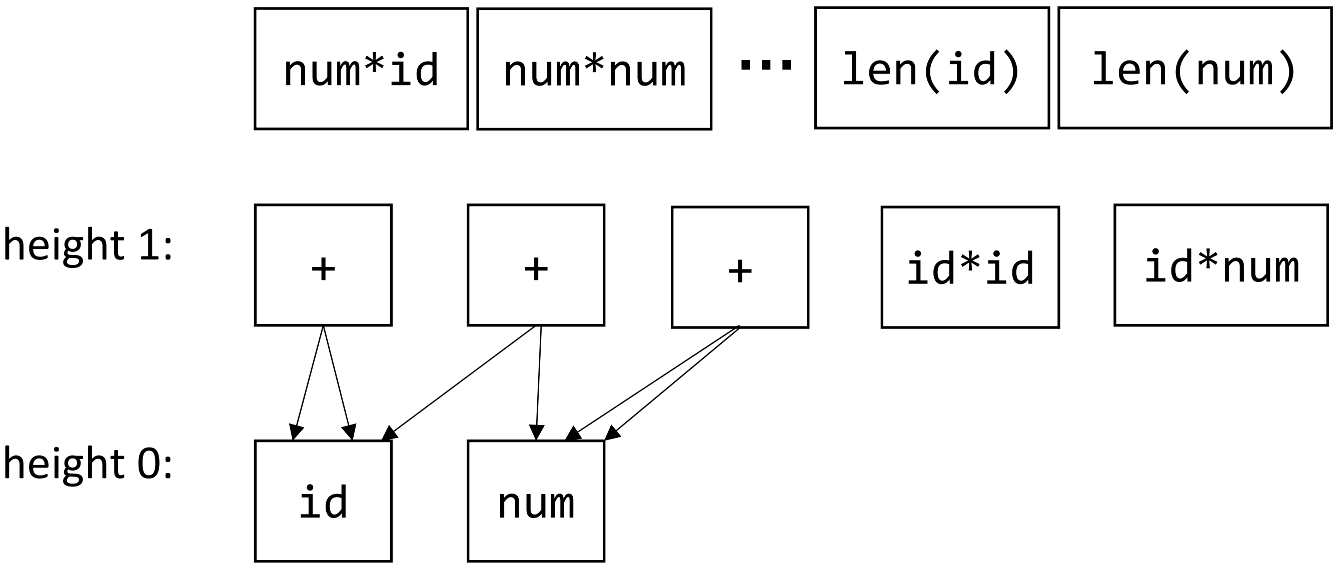
$EList \rightarrow E \mid EList, E$

$E \rightarrow len(E)$

$E \rightarrow id$

$E \rightarrow num$

Enumerating trees



$S \rightarrow E$

$E \rightarrow E + E$

$E \rightarrow E * E$

$E \rightarrow [EList]$

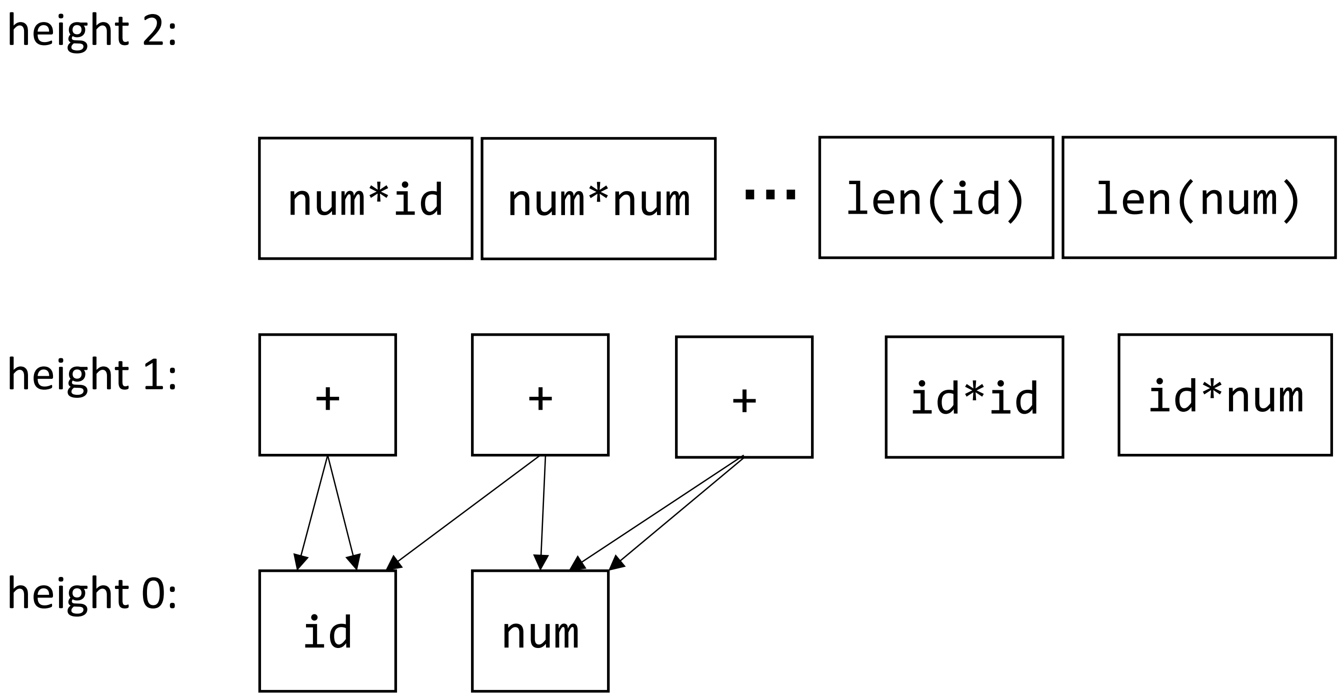
$EList \rightarrow E \mid EList, E$

$\boxed{E \rightarrow len(E)}$

$E \rightarrow id$

$E \rightarrow num$

Enumerating trees



$S \rightarrow E$

$\boxed{E \rightarrow E + E}$

$E \rightarrow E \ * \ E$

$E \rightarrow [EList]$

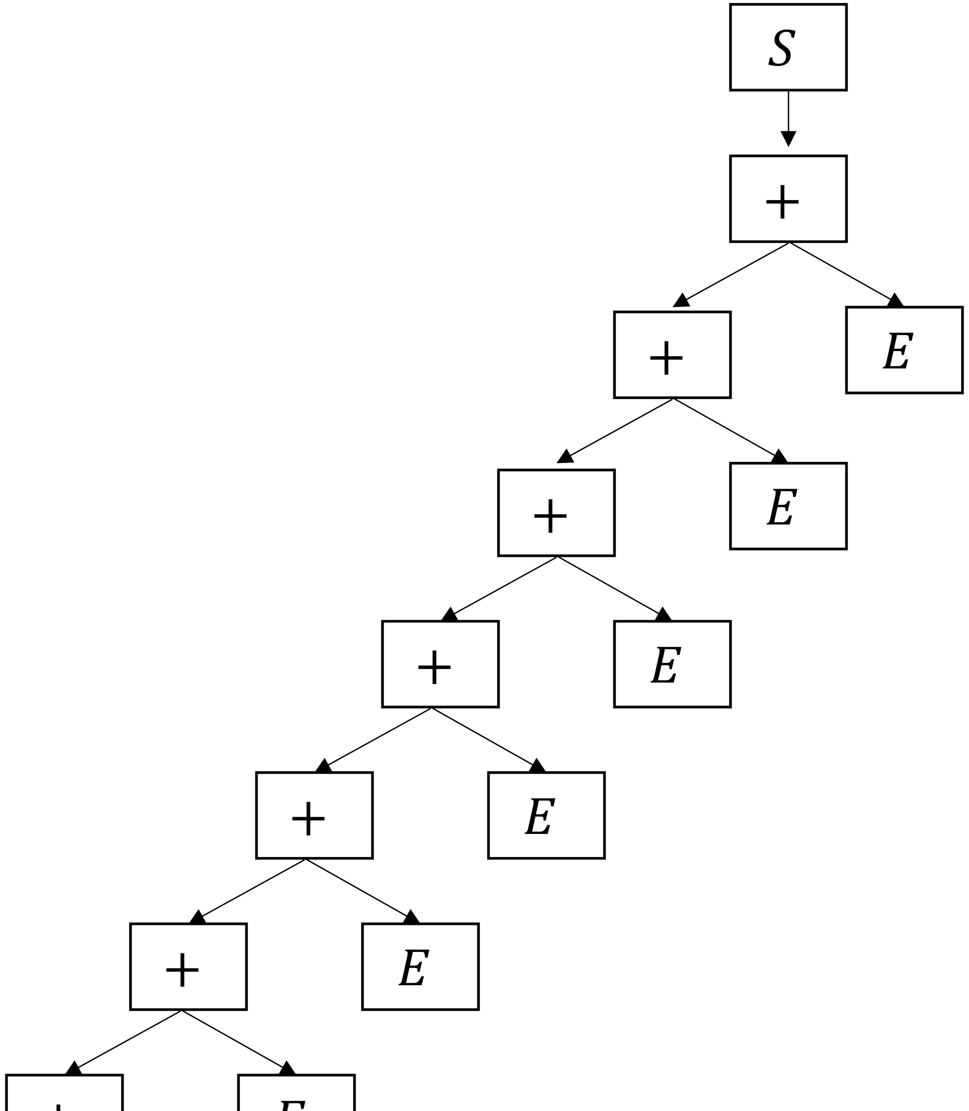
$EList \rightarrow E \mid EList, E$

$E \rightarrow len(E)$

$E \rightarrow id$

$E \rightarrow num$

On one hand: no stack overflow

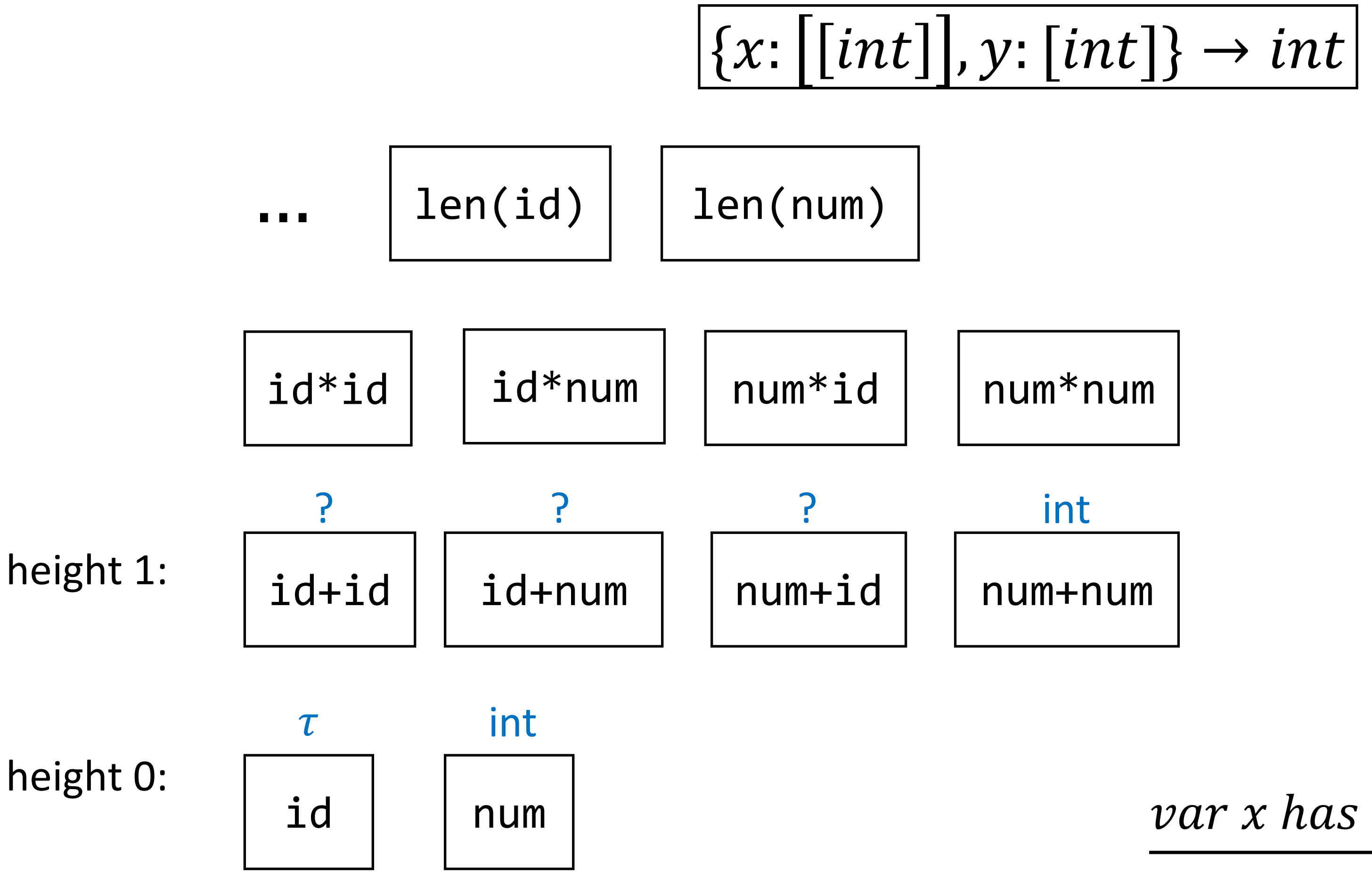


$S \rightarrow E$
$E \rightarrow E + E$
$E \rightarrow E \, * \, E$
$E \rightarrow [EList]$
$EList \rightarrow E \mid EList, E$
$E \rightarrow len(E)$
$E \rightarrow id$
$E \rightarrow num$

On the other hand:

A lot of things suddenly got harder

What if we have a type specification?



$\{x: [[int]], y: [int]\} \rightarrow int$

...

`len(id)` `len(num)`

`id*id` `id*num` `num*id` `num*num`

height 1:

?	?	?	int
`id+id`	`id+num`	`num+id`	`num+num`

height 0:

τ	int
`id`	`num`

$$S \rightarrow E$$
$$E \rightarrow E + E$$
$$E \rightarrow E * E$$
$$E \rightarrow [EList]$$
$$EList \rightarrow E \mid EList, E$$
$$E \rightarrow len(E)$$
$$E \rightarrow id$$
$$E \rightarrow num$$

$$\frac{var\ x\ has\ type\ \tau}{x: \tau} \qquad \frac{}{num: int} \qquad \frac{e_1: int \quad e_2: int}{e_1 + e_2: int}$$

Generic synthesis recipe

1. Generate a candidate program

- Enumerate trees
  - Top-down
  - Bottom-up
- Traverse automata
- Graph reachability
- Enumerate deduction rules
- Cheat* by looking at spec

2. Test against specification

- Run tests
  - Examples
  - Unit
- Encode for SMT solver
- Apply typing rules

Programming by Example

$$\phi(p,x) = (x = \iota_1 \Rightarrow [\![p]\!](x) = \omega_1) \wedge (x = \iota_2 \Rightarrow [\![p]\!](x) = \omega_2) \wedge \cdots$$

or in other words

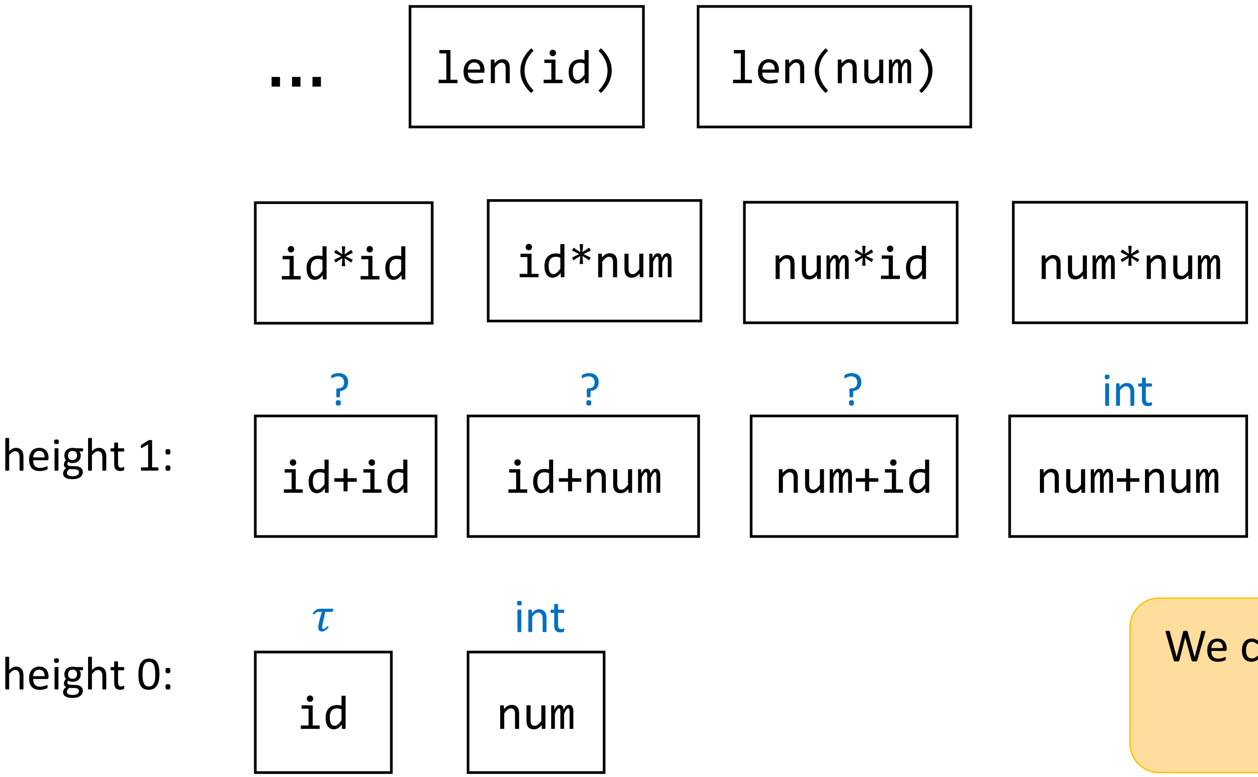
$$\mathcal{E} = \{\iota_i \rightarrow \omega_i\}$$

values of all the variables/the
state of the environment

output of an expression/an effect/
the new state of the environment

Programming by Example

$$\epsilon_1\text{: } \{x \mapsto 6, y \mapsto 8\} \rightarrow 49$$
$$\epsilon_2\text{: } \{x \mapsto 4, y \mapsto 7\} \rightarrow 29$$



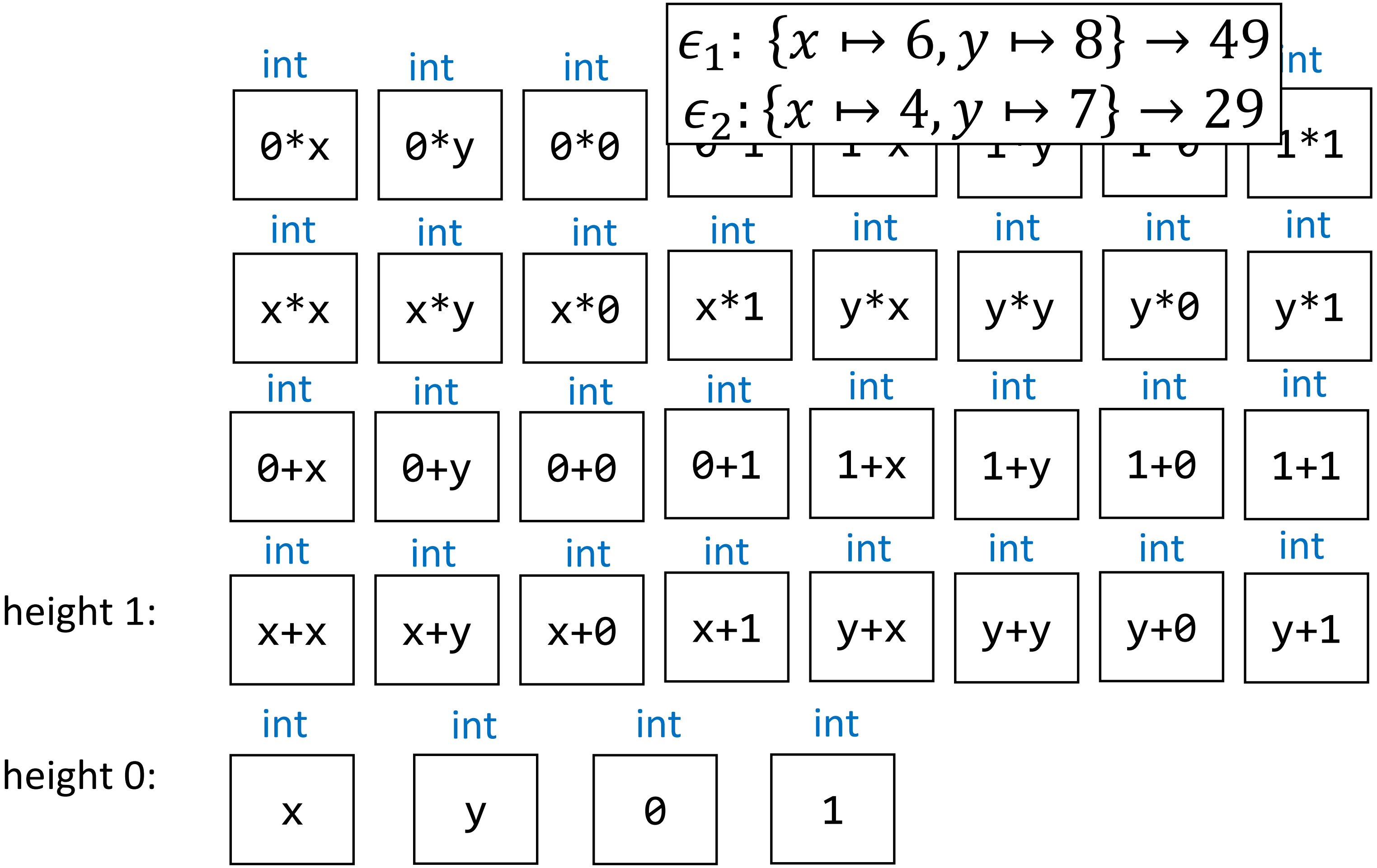
$$S \rightarrow E$$
$$E \rightarrow E + E$$
$$E \rightarrow E * E$$
$$E \rightarrow [EList]$$
$$EList \rightarrow E \mid EList, E$$
$$E \rightarrow len(E)$$
$$E \rightarrow id$$
$$E \rightarrow num$$

We can evaluate every program!

…almost

Programming by Example

ϵ_1: $\{x \mapsto 6, y \mapsto 8\} \rightarrow 49$
ϵ_2: $\{x \mapsto 4, y \mapsto 7\} \rightarrow 29$



int 0*x | int 0*y | int 0*0 | 0*1 | 1*x | 1*y | 1*0 | int 1*1

int x*x | int x*y | int x*0 | int x*1 | int y*x | int y*y | int y*0 | int y*1

int 0+x | int 0+y | int 0+0 | int 0+1 | int 1+x | int 1+y | int 1+0 | int 1+1

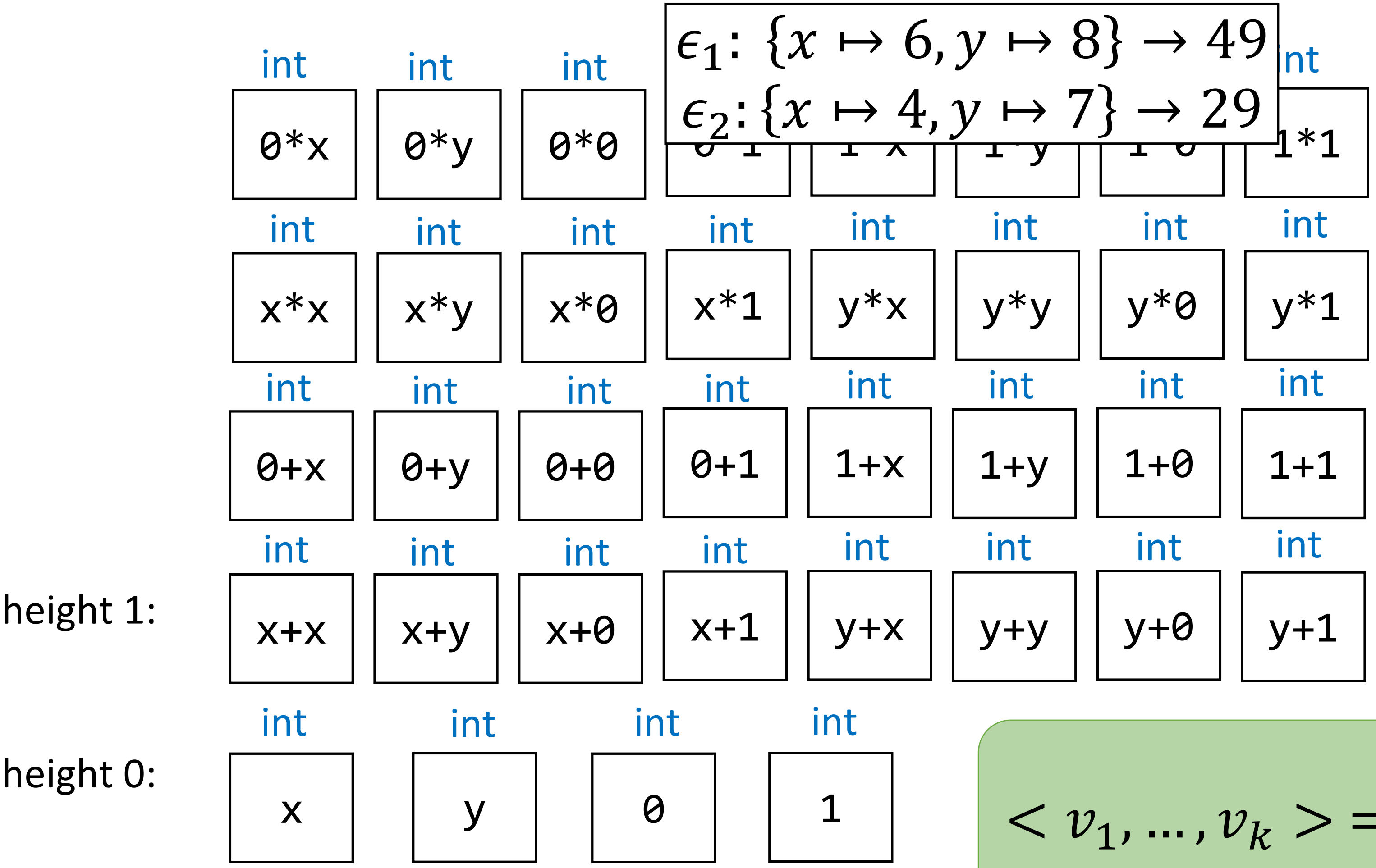
height 1: int x+x | int x+y | int x+0 | int x+1 | int y+x | int y+y | int y+0 | int y+1

height 0: int x | int y | int 0 | int 1

$$S \rightarrow E$$
$$E \rightarrow E + E$$
$$E \rightarrow E \; * \; E$$
$$E \rightarrow [EList]$$
$$EList \rightarrow E \mid EList, E$$
$$E \rightarrow len(E)$$
$$E \rightarrow x \mid y$$
$$E \rightarrow 0 \mid 1$$

Programming by Example

ϵ_1: $\{x \mapsto 6, y \mapsto 8\} \to 49$
ϵ_2: $\{x \mapsto 4, y \mapsto 7\} \to 29$



int 0*x | int 0*y | int 0*0 | 0*1 | 1*x | 1*y | 1*0 | int 1*1

int x*x | int x*y | int x*0 | int x*1 | int y*x | int y*y | int y*0 | int y*1

int 0+x | int 0+y | int 0+0 | int 0+1 | int 1+x | int 1+y | int 1+0 | int 1+1

height 1: int x+x | int x+y | int x+0 | int x+1 | int y+x | int y+y | int y+0 | int y+1

height 0: int x | int y | int 0 | int 1

$$S \to E$$
$$E \to E + E$$
$$E \to E \ * \ E$$
$$E \to [EList]$$
$$EList \to E \mid EList, E$$
$$E \to len(E)$$
$$E \to x \mid y$$
$$E \to 0 \mid 1$$

$$< v_1, \dots, v_k > = < [\![p]\!](\iota_1), \dots, [\![p]\!](\iota_k) >$$

Programming by Example

ϵ_1: $\{x \mapsto 6, y \mapsto 8\} \rightarrow 49$
ϵ_2: $\{x \mapsto 4, y \mapsto 7\} \rightarrow 29$

	<0,0>	<0,0>	<0,0>					1,1>
	0*x	0*y	0*0	0*1	1*x	1*y	1*0	1*1
	<36,16>	<48,28>	<0,0>	<6,4>	<48,28>	<64,49>	<0,0>	<8,7>
	x*x	x*y	x*0	x*1	y*x	y*y	y*0	y*1
	<6,4>	<8,7>	<0,0>	<1,1>	<7,5>	<9,8>	<1,1>	<2,2>
	0+x	0+y	0+0	0+1	1+x	1+y	1+0	1+1
	<12,8>	<14,11>	<6,4>	<7,5>	<14,11>	<16,14>	<8,7>	<9,8>
height 1:	x+x	x+y	x+0	x+1	y+x	y+y	y+0	y+1
	<6,4>	<8,7>	<0,0>	<1,1>				
height 0:	x	y	0	1				

$$S \rightarrow E$$
$$E \rightarrow E + E$$
$$E \rightarrow E * E$$
$$E \rightarrow [EList]$$
$$EList \rightarrow E \mid EList, E$$
$$E \rightarrow len(E)$$
$$E \rightarrow x \mid y$$
$$E \rightarrow 0 \mid 1$$

$$< v_1, \ldots, v_k > = < [\![p]\!](\iota_1), \ldots, [\![p]\!](\iota_k) >$$

Equivalence classes

[Albarghouthi et al. 2013, Udupa et al. 2013]

ϵ_1: $\{x \mapsto 6, y \mapsto 8\} \to 49$

ϵ_2: $\{x \mapsto 4, y \mapsto 7\} \to 29$

	<0,0>	<0,0>	<0,0>					1,1>
	0*x	0*y	0*0	0*1	1*x	1*y	1*0	1*1
	<36,16>	<48,28>	<0,0>	<6,4>	<48,28>	<64,49>	<0,0>	<8,7>
	x*x	x*y	x*0	x*1	y*x	y*y	y*0	y*1
	<6,4>	<8,7>	<0,0>	<1,1>	<7,5>	<9,8>	<1,1>	<2,2>
	0+x	0+y	0+0	0+1	1+x	1+y	1+0	1+1
	<12,8>	<14,11>	<6,4>	<7,5>	<14,11>	<16,14>	<8,7>	<9,8>
height 1:	x+x	x+y	x+0	x+1	y+x	y+y	y+0	y+1
	<6,4>	<8,7>	<0,0>	<1,1>				
height 0:	x	y	0	1				

$$S \to E$$
$$E \to E + E$$
$$E \to E * E$$
$$E \to [EList]$$
$$EList \to E \mid EList, E$$
$$E \to len(E)$$
$$E \to x \mid y$$
$$E \to 0 \mid 1$$

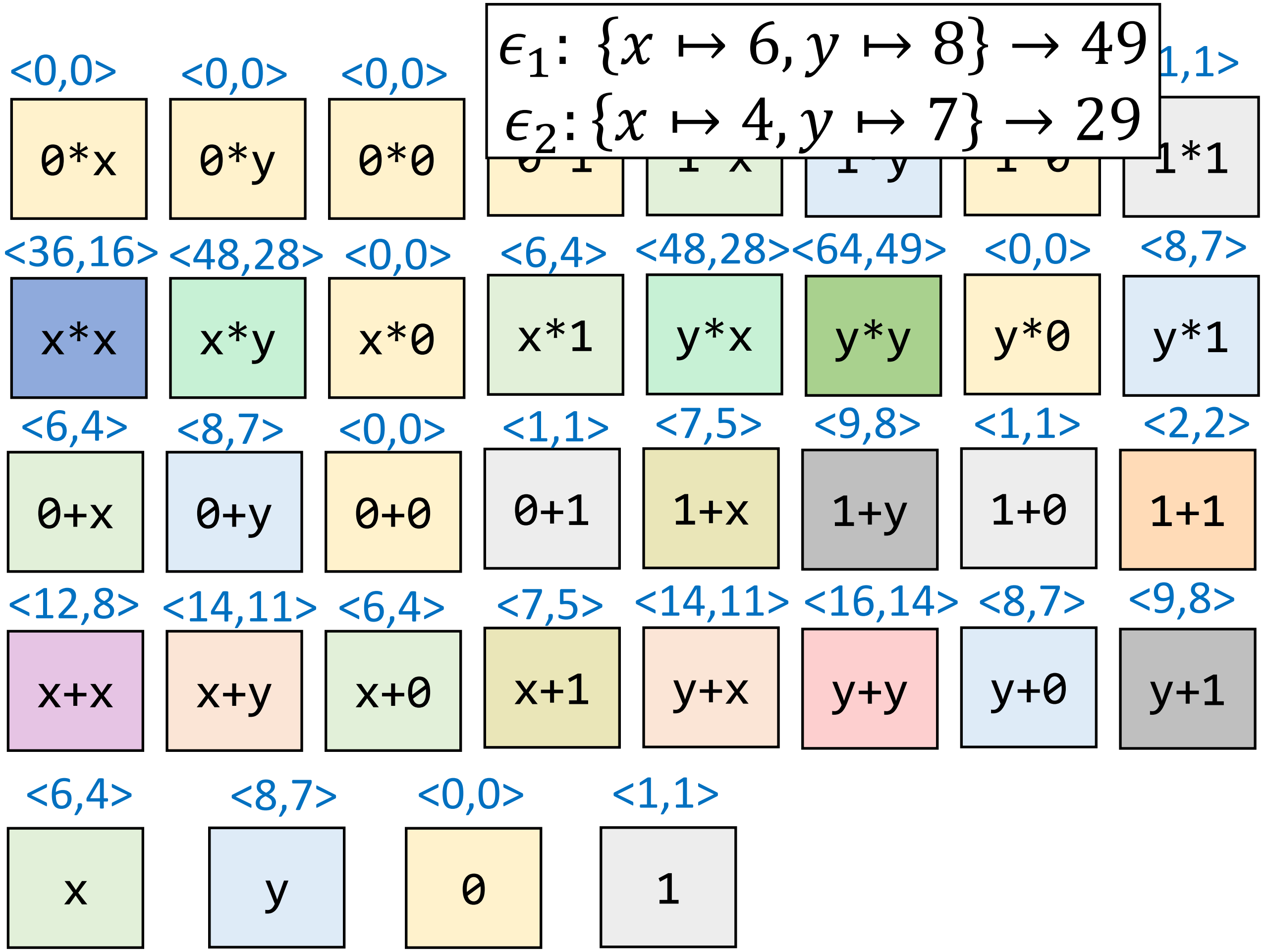
$$< v_1, \ldots, v_k > = < [\![p]\!](\iota_1), \ldots, [\![p]\!](\iota_k) >$$

Equivalence classes

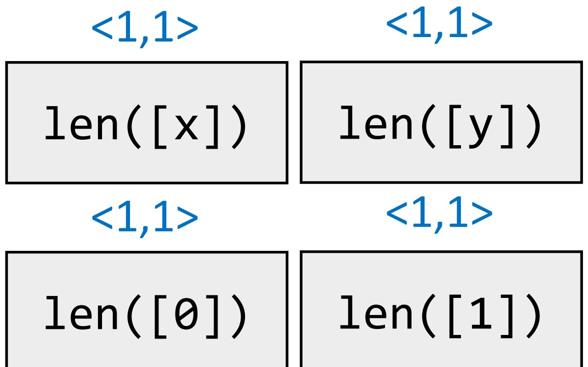
[Albarghouthi et al. 2013, Udupa et al. 2013]

ϵ_1: $\{x \mapsto 6, y \mapsto 8\} \rightarrow 49$
ϵ_2: $\{x \mapsto 4, y \mapsto 7\} \rightarrow 29$

$$S \rightarrow E$$
$$E \rightarrow E + E$$
$$E \rightarrow E \ * \ E$$
$$E \rightarrow [EList]$$
$$EList \rightarrow E \mid EList, E$$
$$E \rightarrow len(E)$$
$$E \rightarrow x \mid y$$
$$E \rightarrow 0 \mid 1$$



	<0,0>	<0,0>	<0,0>					1,1>
	0*x	0*y	0*0	0*1	1*x	1*y	1*0	1*1
	<36,16>	<48,28>	<0,0>	<6,4>	<48,28>	<64,49>	<0,0>	<8,7>
	x*x	x*y	x*0	x*1	y*x	y*y	y*0	y*1
	<6,4>	<8,7>	<0,0>	<1,1>	<7,5>	<9,8>	<1,1>	<2,2>
	0+x	0+y	0+0	0+1	1+x	1+y	1+0	1+1
	<12,8>	<14,11>	<6,4>	<7,5>	<14,11>	<16,14>	<8,7>	<9,8>
height 1:	x+x	x+y	x+0	x+1	y+x	y+y	y+0	y+1
	<6,4>	<8,7>	<0,0>	<1,1>				
height 0:	x	y	0	1				



<1,1>	<1,1>
len([x])	len([y])
<1,1>	<1,1>
len([0])	len([1])

Observational equivalence

[Albarghouthi et al. 2013, Udupa et al. 2013]

Equivalence:

$p_1 \equiv p_2$ i.f.f. for every possible input i ever, $[\![p_1]\!](i) = [\![p_2]\!](i)$

Observational equivalence:

$p_1 \equiv_{OE} p_2$ i.f.f. for every input i the user cares about, $[\![p_1]\!](i) = [\![p_2]\!](i)$

Equivalence classes

[Albarghouthi et al. 2013, Udupa et al. 2013]

$$\epsilon_1\!: \{x \mapsto 6, y \mapsto 8\} \to 49$$
$$\epsilon_2\!: \{x \mapsto 4, y \mapsto 7\} \to 29$$

	<0,0> 0*x	<0,0> 0*y	<0,0> 0*0	0*1	1*x	1*y	1*0	1,1> 1*1
	<36,16> x*x	<48,28> x*y	<0,0> x*0	<6,4> x*1	<48,28> y*x	<64,49> y*y	<0,0> y*0	<8,7> y*1
	<6,4> 0+x	<8,7> 0+y	<0,0> 0+0	<1,1> 0+1	<7,5> 1+x	<9,8> 1+y	<1,1> 1+0	<2,2> 1+1
height 1:	<12,8> x+x	<14,11> x+y	<6,4> x+0	<7,5> x+1	<14,11> y+x	<16,14> y+y	<8,7> y+0	<9,8> y+1
height 0:	<6,4> x	<8,7> y	<0,0> 0	<1,1> 1				

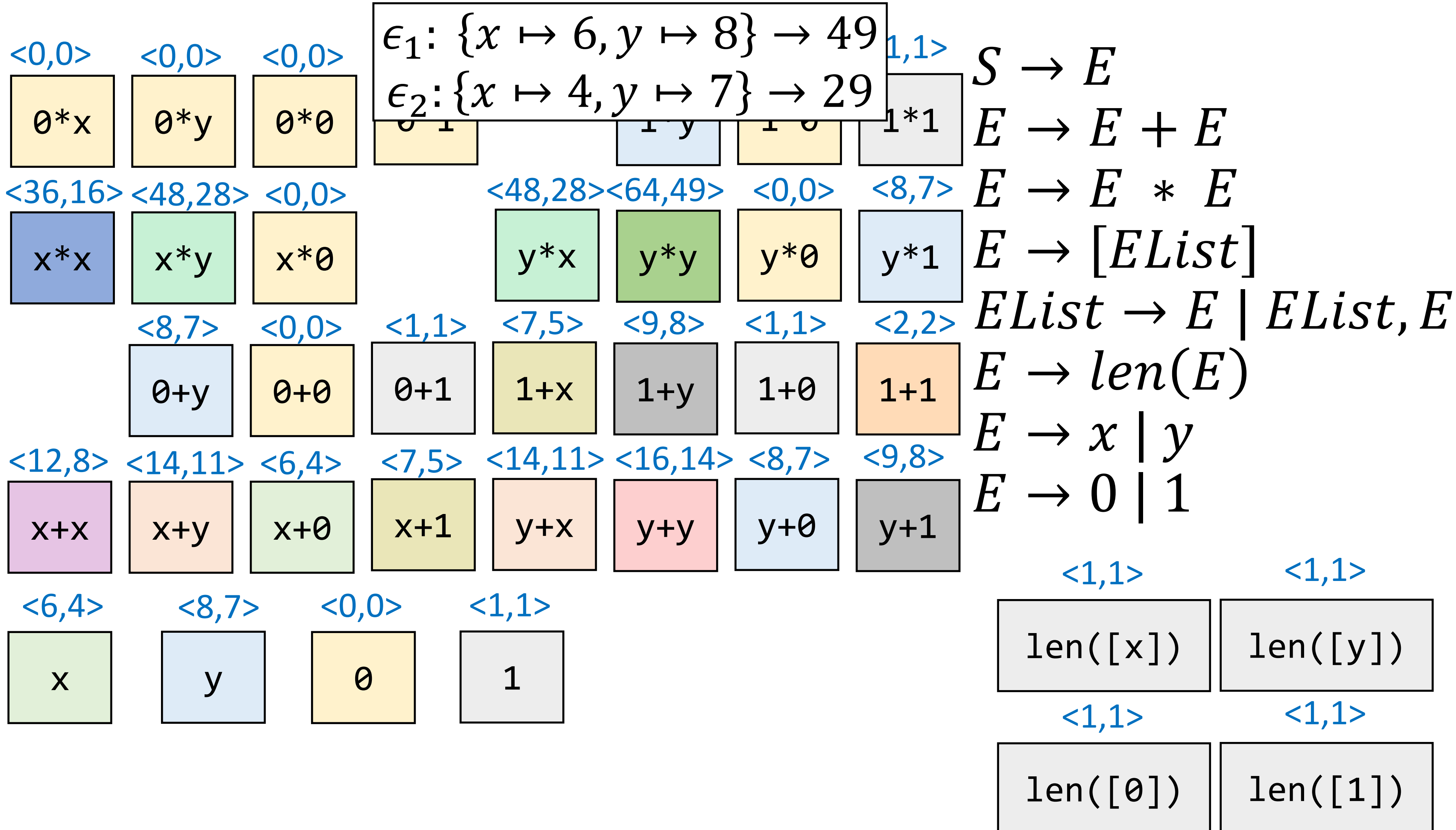
$$
\begin{aligned}
S &\to E \\
E &\to E + E \\
E &\to E * E \\
E &\to [EList] \\
EList &\to E \mid EList, E \\
E &\to len(E) \\
E &\to x \mid y \\
E &\to 0 \mid 1
\end{aligned}
$$

<1,1> len([x])	<1,1> len([y])
<1,1> len([0])	<1,1> len([1])

Equivalence classes

[Albarghouthi et al. 2013, Udupa et al. 2013]

ϵ_1: $\{x \mapsto 6, y \mapsto 8\} \to 49$
ϵ_2: $\{x \mapsto 4, y \mapsto 7\} \to 29$



<0,0> 0*x | <0,0> 0*y | <0,0> 0*0 | 0*1 | 1*y | 1*0 | 1,1> 1*1

<36,16> x*x | <48,28> x*y | <0,0> x*0 | <48,28> y*x | <64,49> y*y | <0,0> y*0 | <8,7> y*1

<8,7> 0+y | <0,0> 0+0 | <1,1> 0+1 | <7,5> 1+x | <9,8> 1+y | <1,1> 1+0 | <2,2> 1+1

height 1: <12,8> x+x | <14,11> x+y | <6,4> x+0 | <7,5> x+1 | <14,11> y+x | <16,14> y+y | <8,7> y+0 | <9,8> y+1

height 0: <6,4> x | <8,7> y | <0,0> 0 | <1,1> 1

<1,1> len([x]) | <1,1> len([y])

<1,1> len([0]) | <1,1> len([1])

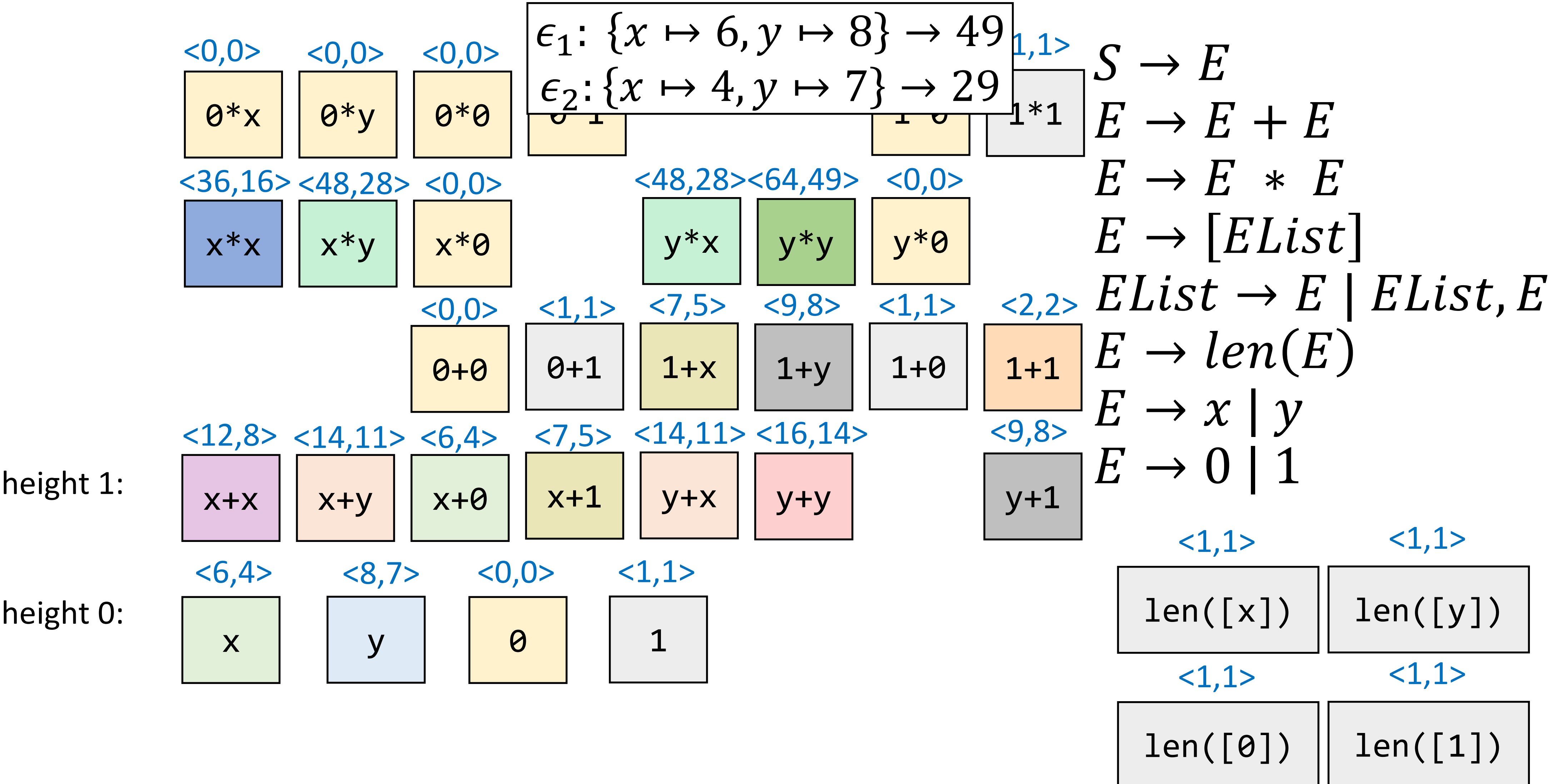
$$S \to E$$
$$E \to E + E$$
$$E \to E \; * \; E$$
$$E \to [EList]$$
$$EList \to E \mid EList, E$$
$$E \to len(E)$$
$$E \to x \mid y$$
$$E \to 0 \mid 1$$

Equivalence classes

[Albarghouthi et al. 2013, Udupa et al. 2013]

ϵ_1: $\{x \mapsto 6, y \mapsto 8\} \to 49$
ϵ_2: $\{x \mapsto 4, y \mapsto 7\} \to 29$

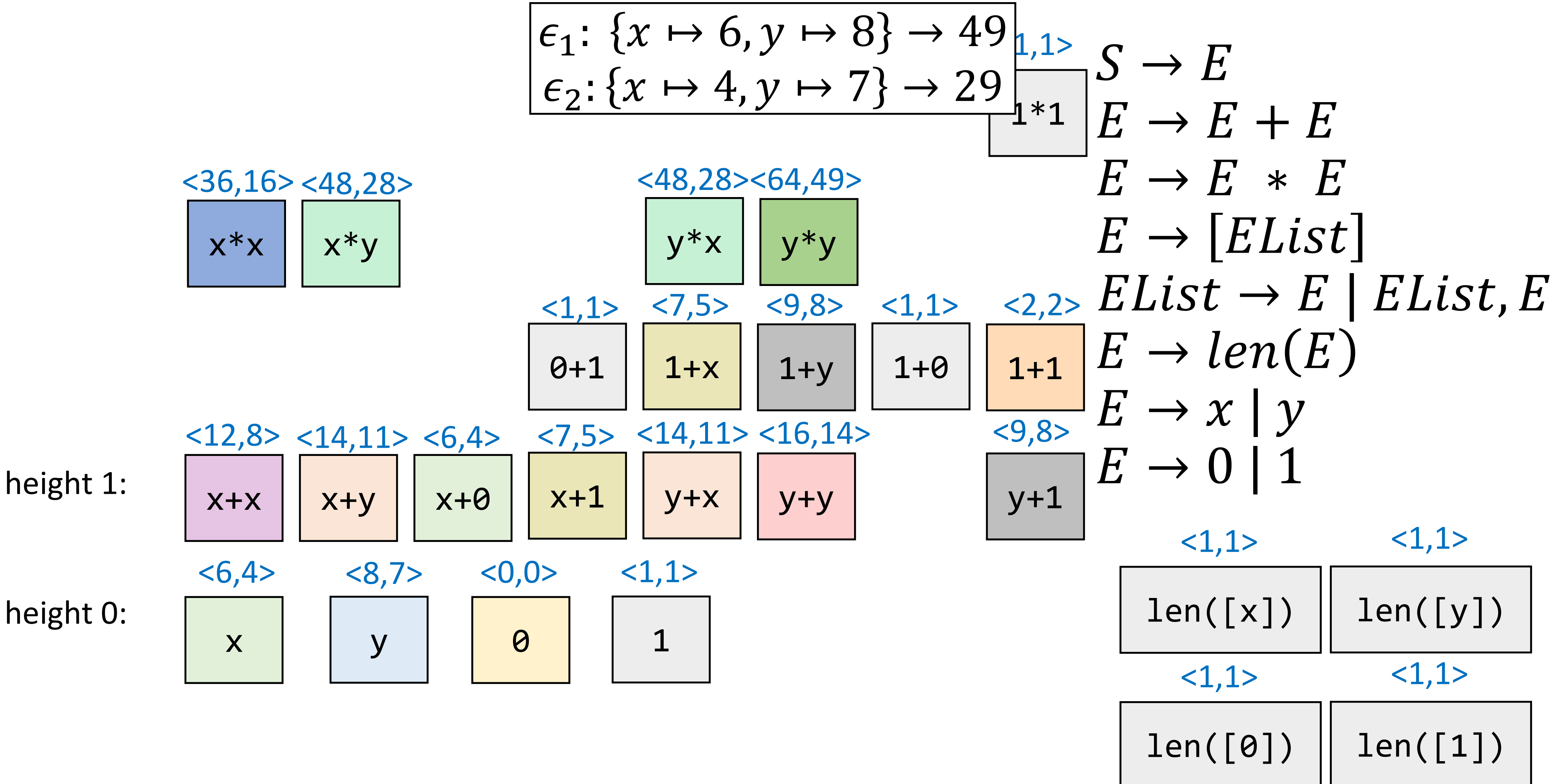
$$S \to E$$
$$E \to E + E$$
$$E \to E * E$$
$$E \to [EList]$$
$$EList \to E \mid EList, E$$
$$E \to len(E)$$
$$E \to x \mid y$$
$$E \to 0 \mid 1$$



Equivalence classes

[Albarghouthi et al. 2013, Udupa et al. 2013]

$$\epsilon_1\text{: } \{x \mapsto 6, y \mapsto 8\} \to 49$$
$$\epsilon_2\text{: } \{x \mapsto 4, y \mapsto 7\} \to 29$$

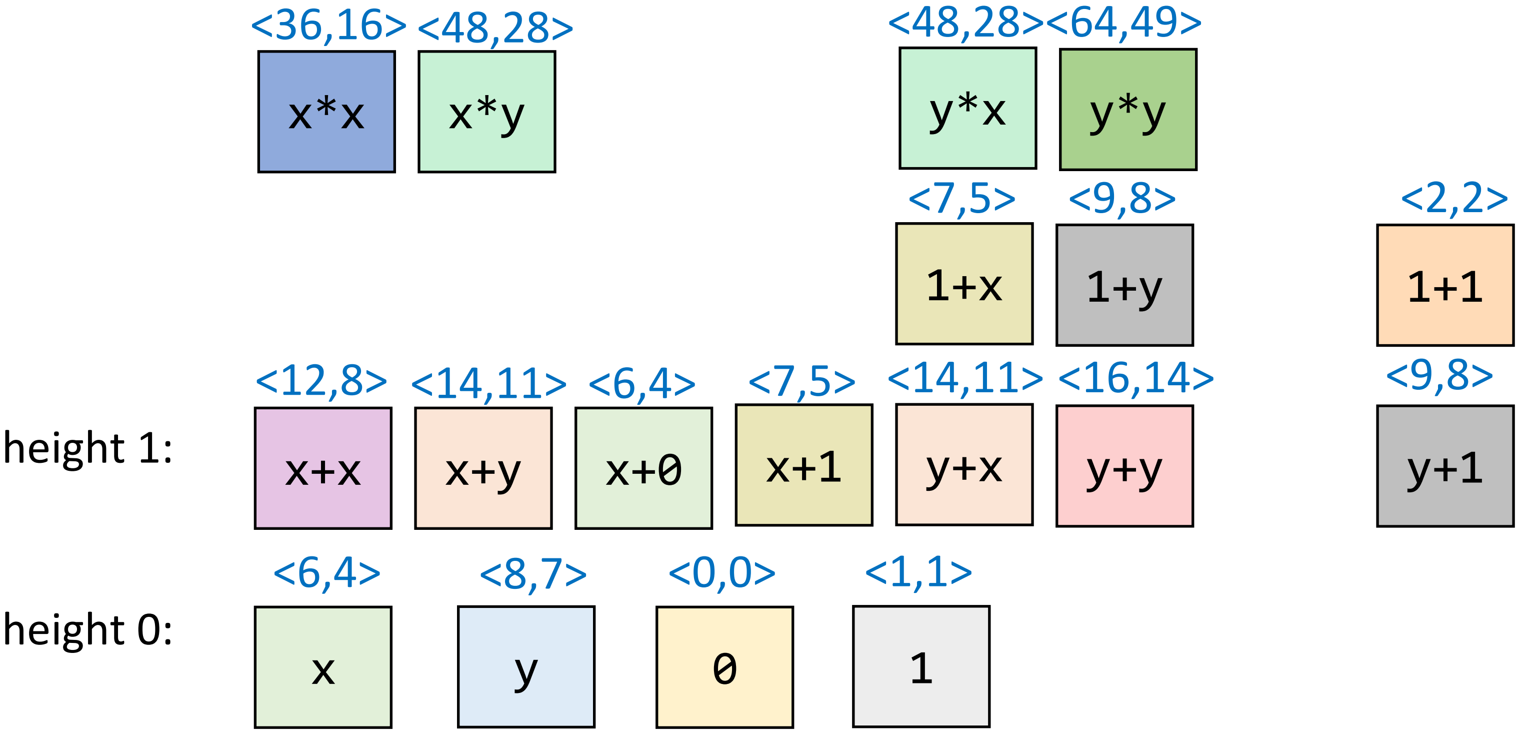


$$S \to E$$
$$E \to E + E$$
$$E \to E * E$$
$$E \to [EList]$$
$$EList \to E \mid EList, E$$
$$E \to len(E)$$
$$E \to x \mid y$$
$$E \to 0 \mid 1$$

Equivalence classes

[Albarghouthi et al. 2013, Udupa et al. 2013]

$$\boxed{\begin{array}{l} \epsilon_1\text{: } \{x \mapsto 6, y \mapsto 8\} \to 49 \\ \epsilon_2\text{: } \{x \mapsto 4, y \mapsto 7\} \to 29 \end{array}}$$



<36,16> x*x <48,28> x*y

<48,28> y*x <64,49> y*y

<7,5> 1+x <9,8> 1+y

<2,2> 1+1

height 1: <12,8> x+x <14,11> x+y <6,4> x+0 <7,5> x+1 <14,11> y+x <16,14> y+y

<9,8> y+1

height 0: <6,4> x <8,7> y <0,0> 0 <1,1> 1

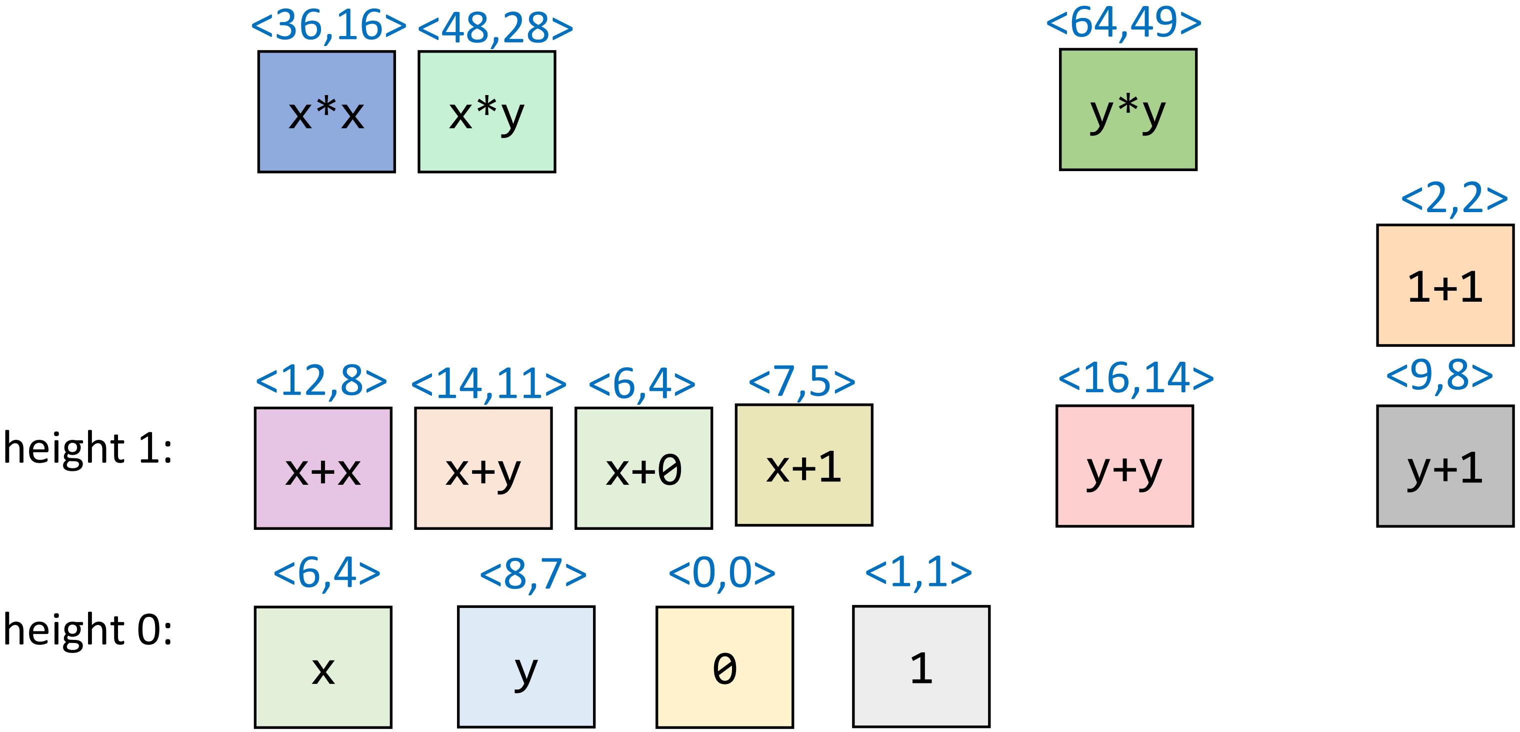
$$S \to E$$
$$E \to E + E$$
$$E \to E * E$$
$$E \to [EList]$$
$$EList \to E \mid EList, E$$
$$E \to len(E)$$
$$E \to x \mid y$$
$$E \to 0 \mid 1$$

Equivalence classes

[Albarghouthi et al. 2013, Udupa et al. 2013]

$$\epsilon_1 \colon \{x \mapsto 6, y \mapsto 8\} \to 49$$
$$\epsilon_2 \colon \{x \mapsto 4, y \mapsto 7\} \to 29$$

$$S \to E$$
$$E \to E + E$$
$$E \to E * E$$
$$E \to [EList]$$
$$EList \to E \mid EList, E$$
$$E \to len(E)$$
$$E \to x \mid y$$
$$E \to 0 \mid 1$$



<36,16> x*x
<48,28> x*y
<64,49> y*y
<2,2> 1+1

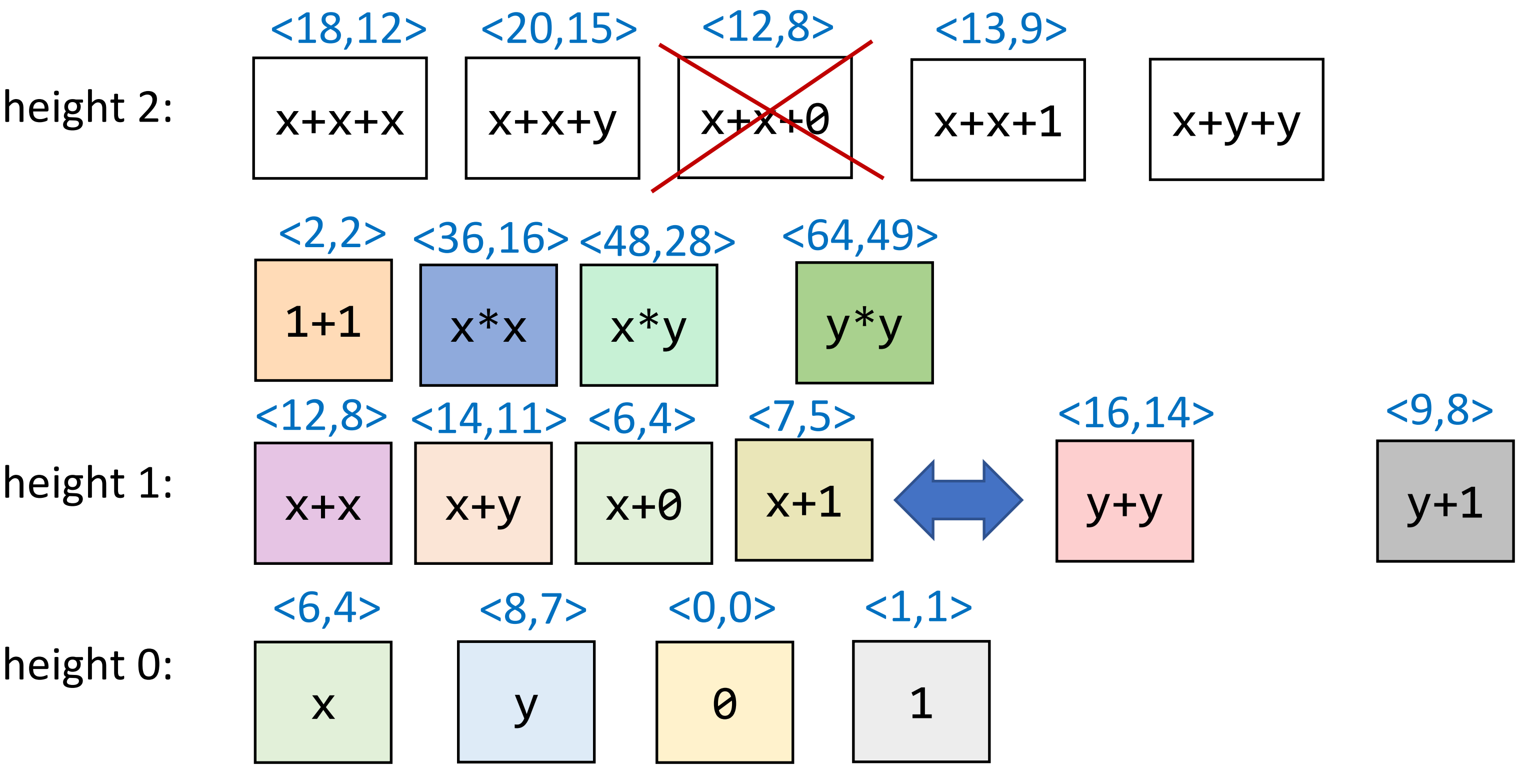
height 1:
<12,8> x+x
<14,11> x+y
<6,4> x+0
<7,5> x+1
<16,14> y+y
<9,8> y+1

height 0:
<6,4> x
<8,7> y
<0,0> 0
<1,1> 1

On the fly

[Albarghouthi et al. 2013, Udupa et al. 2013]

$$\epsilon_1\colon \{x \mapsto 6, y \mapsto 8\} \to 49$$
$$\epsilon_2\colon \{x \mapsto 4, y \mapsto 7\} \to 29$$



$$S \to E$$
$$E \to E + E$$
$$E \to E \ * \ E$$
$$E \to [EList]$$
$$EList \to E \mid EList, E$$
$$E \to len(E)$$
$$E \to x \mid y$$
$$E \to 0 \mid 1$$

Pros and cons of bottom-up

Pros:

- Iterative deepening is free
- No need to solver-encode anything

Cons:

- Have to specify literals
- Purity
- OE is very aggressive

Summary

- This time:
  - Program synthesis
  - Types of specification
  - Search algorithms
  - Directions for enumerative search
- Next time:
  - What happens when you give this to people?