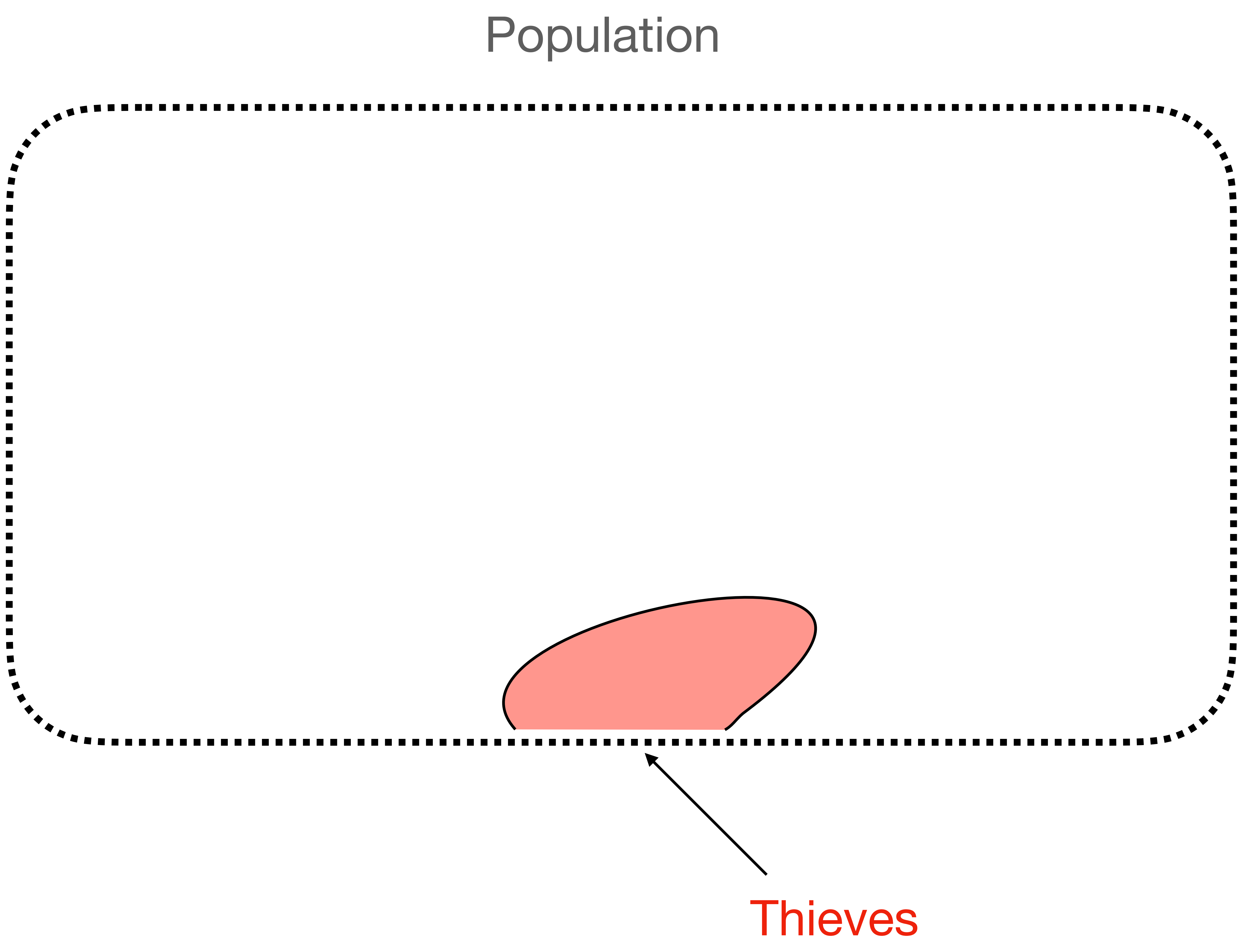
Neural Guided Program Synthesis

AAAI'22 & **JOSS'22**
with N.Fijalkow, G.Lagarde, T.Matricon, K.Ellis, P.Ohlmann, A.Potta

Séminaire LINKS - 05/04/2024

Disclaimer: this introduction does not reflect my own political views

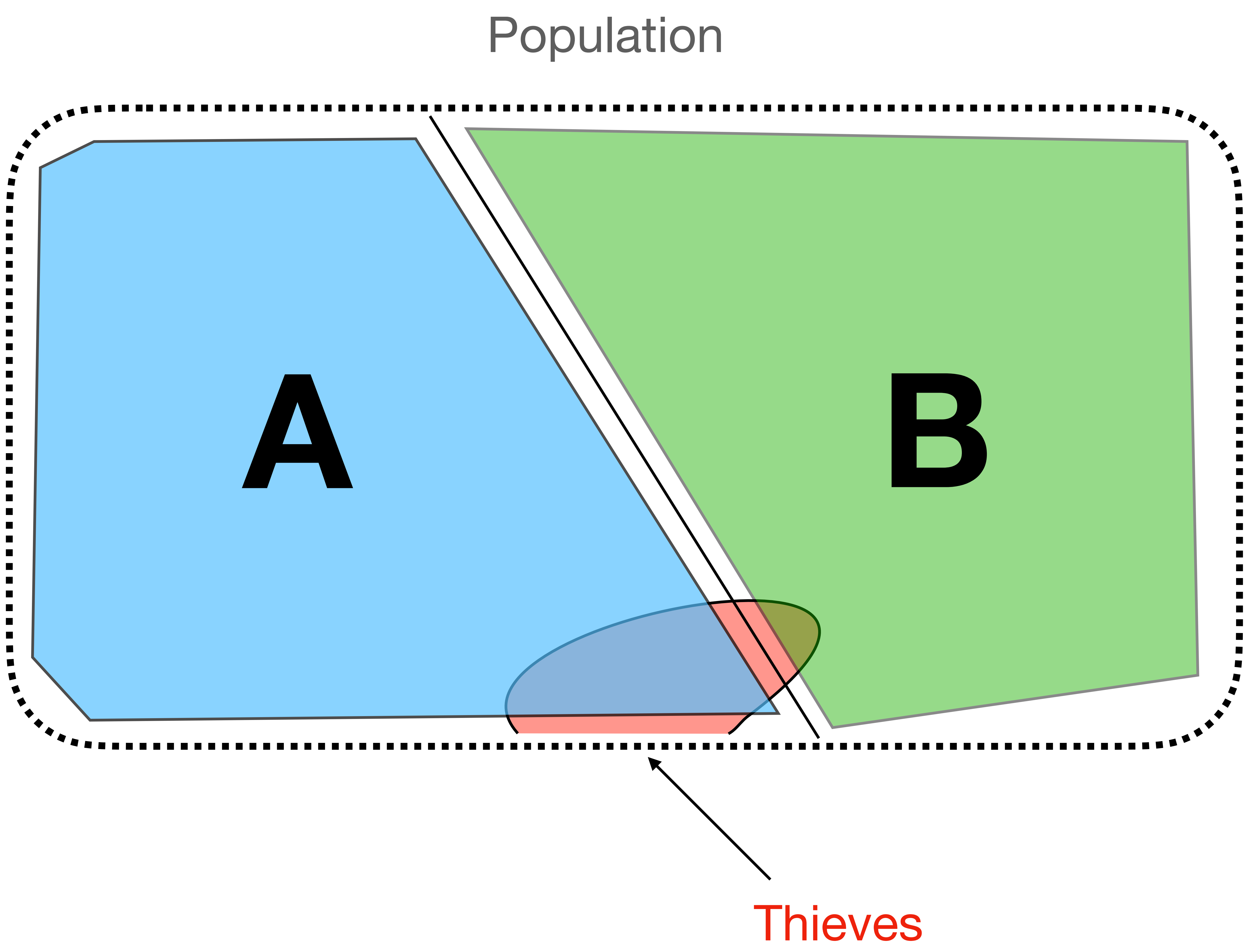
Airport Screening



Who to scan in order to catch thieves?

Disclaimer: this introduction does not reflect my own political views

Airport Screening



Who to scan in order to catch thieves?

Assumption: prior knowledge
Say: someone in group A is 9 times more likely to be a thief

Airport Screening

We have a distribution D
We want to do sampling *with replacement* using another distribution, say D'

$$\min_{D'} \mathbb{E}[\text{number of scans before finding a thief}]$$

$$Loss(D, D')$$

Airport Screening

Naive: use D

Optimal: $\sqrt{D} = \dfrac{\sqrt{D(x)}}{\sum_y \sqrt{D(y)}}$

If group A is 9 times more likely to be a thief, it should be sampled 3 times more often.

Airport Screening

Simplest instance:

$$D : \quad p \,.\, HEAD + (1-p) \,.\, TAIL$$

$$D' : \quad p' \,.\, HEAD + (1-p') \,.\, TAIL$$

Airport Screening

Simplest instance:

$$D: \quad p \,.\, HEAD + (1-p) \,.\, TAIL$$

$$D' : \quad p' \,.\, HEAD + (1-p') \,.\, TAIL$$

$$Loss(D, D') = \frac{p}{p'} + \frac{1-p}{1-p'} \longrightarrow \quad p' = \frac{\sqrt{p}}{\sqrt{p} + \sqrt{1-p}}$$

Airport Screening

Simplest instance:

$$D: \quad p\,.\,HEAD + (1-p)\,.\,TAIL$$

$$D' : \quad p'\,.\,HEAD + (1-p')\,.\,TAIL$$

$$Loss(D, D') = \frac{p}{p'} + \frac{1-p}{1-p'} \longrightarrow p' = \frac{\sqrt{p}}{\sqrt{p} + \sqrt{1-p}}$$

Program Synthesis?

An old dream: Church's Problem (1957)

Specification → **Program Synthesis** → Program

Program Synthesis?

An old dream: Church's Problem (1957)

Specification ⟶ **Program Synthesis** ⟶ Program

Logical formulas

Specification = **ϕ** a logical formula
A program **P** such that for all x, ϕ(x, P(x)) = True

Natural language

« A program that removes odd elements and sort the rest »

A set of I/O examples

```
[1, 5, 4, 2]  ⟶  [2, 4]
[6, 3, 0, 8]  ⟶  [0, 6, 8]
```

Today's setting! aka « *Inductive Program Synthesis* »

Program Synthesis?

An old dream: Church's Problem (1957)

Specification → **Program Synthesis** → Program

Logical formulas

Specification = **ϕ** a logical formula
A program **P** such that for all x, ϕ(x, P(x)) = True

Natural language

« A program that removes odd elements and sort the rest »

A set of I/O examples

```
[1, 5, 4, 2]  →  [2, 4]
[6, 3, 0, 8]  →  [0, 6, 8]
```

SORT; FILTER(EVEN)

Today's setting! aka « *Inductive Program Synthesis* »

Program Synthesis?

An old dream: Church's Problem (1957)

Specification ⟶ **Program Synthesis** ⟶ Program

Logical formulas

Specification = **ϕ** a logical formula
A program **P** such that for all x, ϕ(x, P(x)) = True

Natural language

« A program that removes odd elements and sort the rest »

A set of I/O examples

```
[1, 5, 4, 2]  ⟶  [2, 4]
[6, 3, 0, 8]  ⟶  [0, 6, 8]
```

Today's setting! aka « *Inductive Program Synthesis* »

Similarities with machine learning

Similarities with machine learning

Program Synthesis	Machine learning
Small number of examples	Large amount of data
Combinatorial/Symbolic search	Optimisation
Satisfies all specifications (reliable)	Minimizes a loss function (noisy)
Program (interpretable)	Model (hard to understand)

DeepCoder

Microsoft (Balog et al., 2017) — it manipulates list of integers

Program 4:
```
x ← [int]
y ← [int]
c ← SORT x
d ← SORT y
e ← REVERSE d
f ← ZIPWITH (*) d e
g ← SUM f
```

Input-output example:
Input:
[7 3 8 2 5],
[2 8 9 1 3]
Output:
79

Description:
Xavier and Yasmine are laying sticks to form non-overlapping rectangles on the ground. They both have fixed sets of pairs of sticks of certain lengths (represented as arrays x and y of numbers). Xavier only lays sticks parallel to the x axis, and Yasmine lays sticks only parallel to y axis. Compute the area their rectangles will cover at least.

TF-Coder

Google (Shi et al. 2019) — it manipulates tensors in Tensorflow

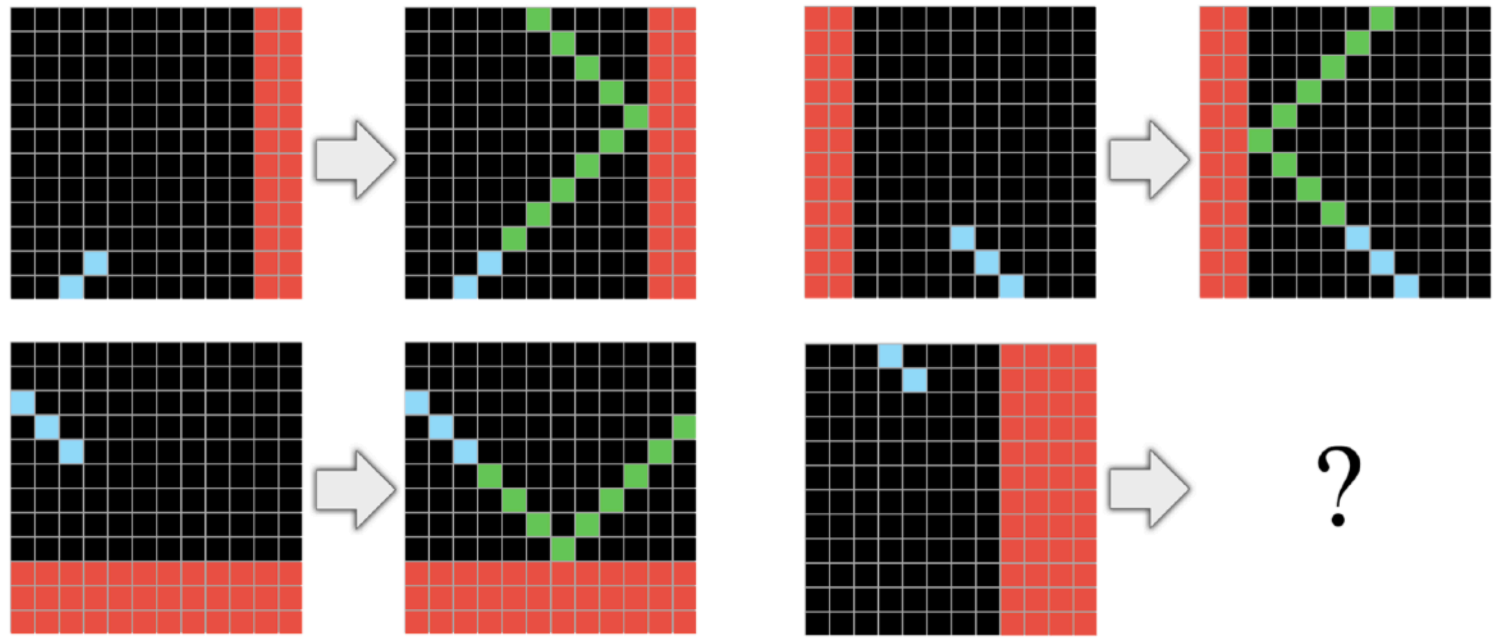
```
# Input-output example
inputs = {
    'scores': [[0.7, 0.2, 0.1],
               [0.4, 0.5, 0.1],
               [0.4, 0.4, 0.2],
               [0.3, 0.4, 0.3],
               [0.0, 0.0, 1.0]],
}
output = [[1, 0, 0],
          [0, 1, 0],
          [1, 0, 0],
          [0, 1, 0],
          [0, 0, 1]]
```

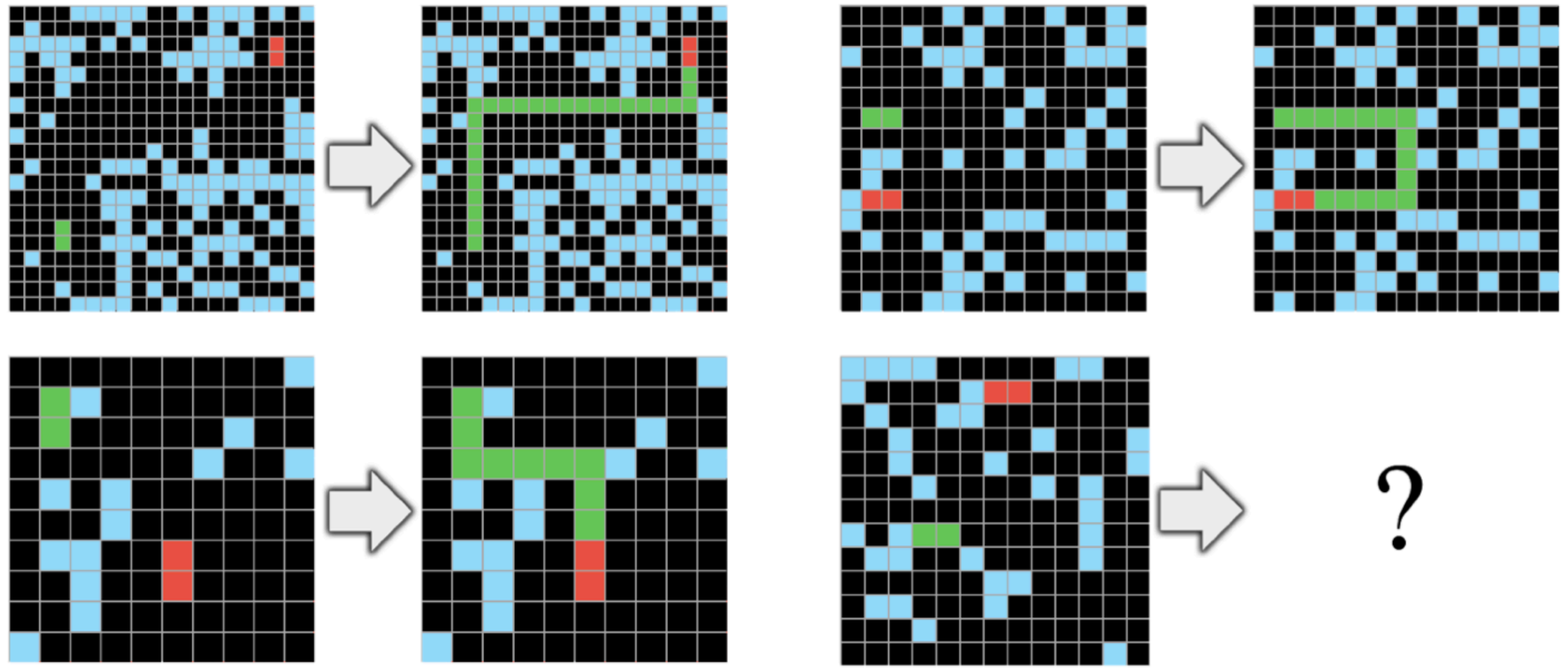
TF-Coder uses a combination of `tf.one_hot` and `tf.argmax` in a short solution to this problem:

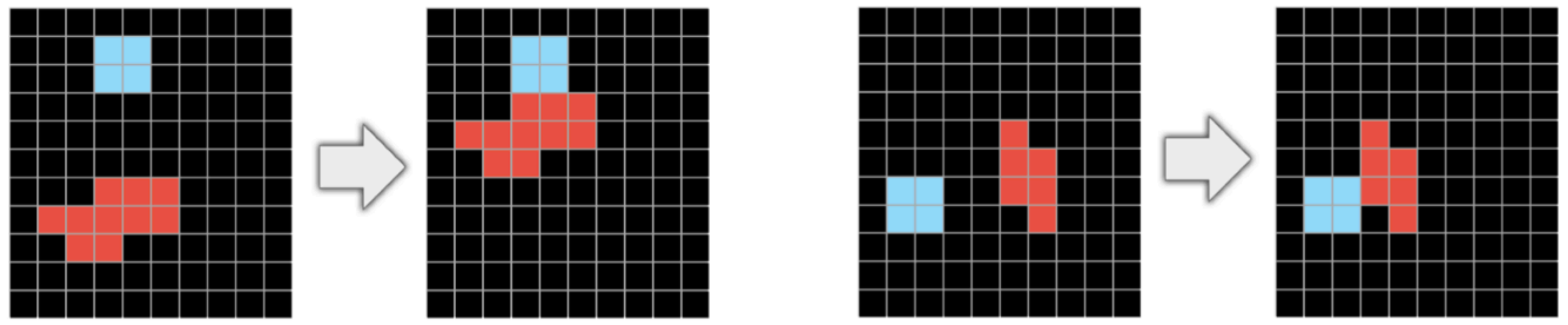
```
tf.cast(tf.one_hot(tf.argmax(scores, axis=1), 3), tf.int32)
```

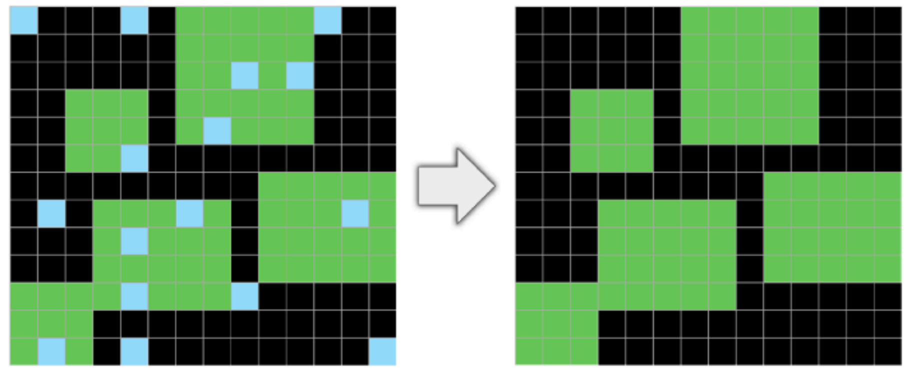
ARC Dataset

« The Abstraction and Reasoning Corpus », in « On the measure of intelligence » François Chollet, 2019









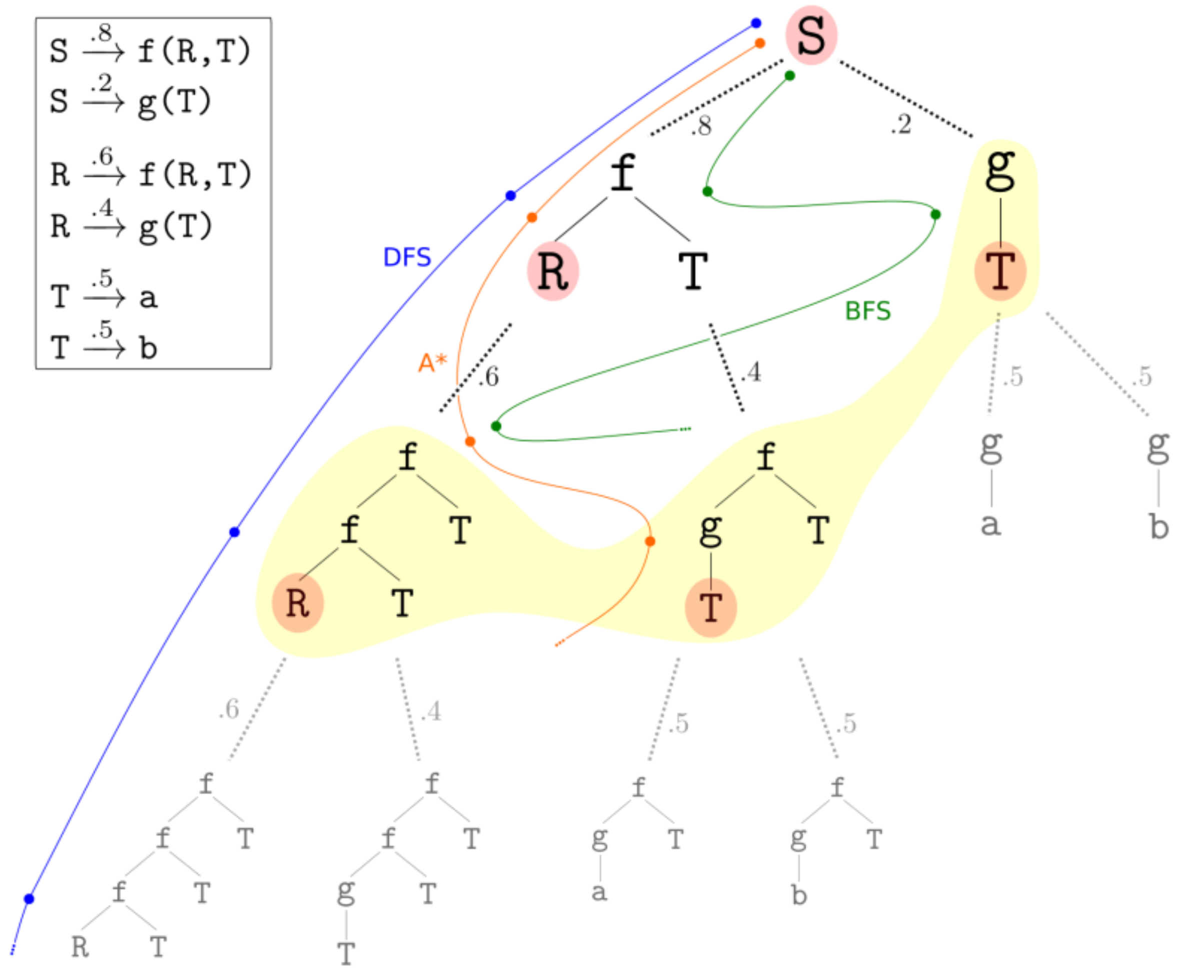
Neural-Guided Program Synthesis

Combination of formal methods and machine learning

reliable

efficient

Symbolic Search



Candidate program

Feedback

Evaluation

PROG ⊧ SPEC ?

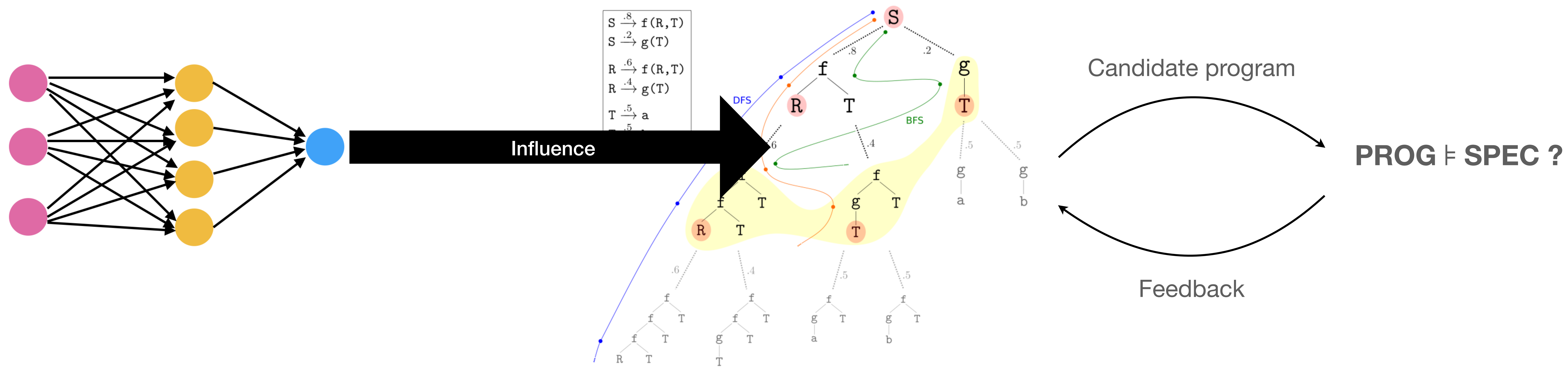
Neural-Guided Program Synthesis

Combination of formal methods and machine learning

reliable

efficient

Machine Learning Predictions **Symbolic Search** **Evaluation**

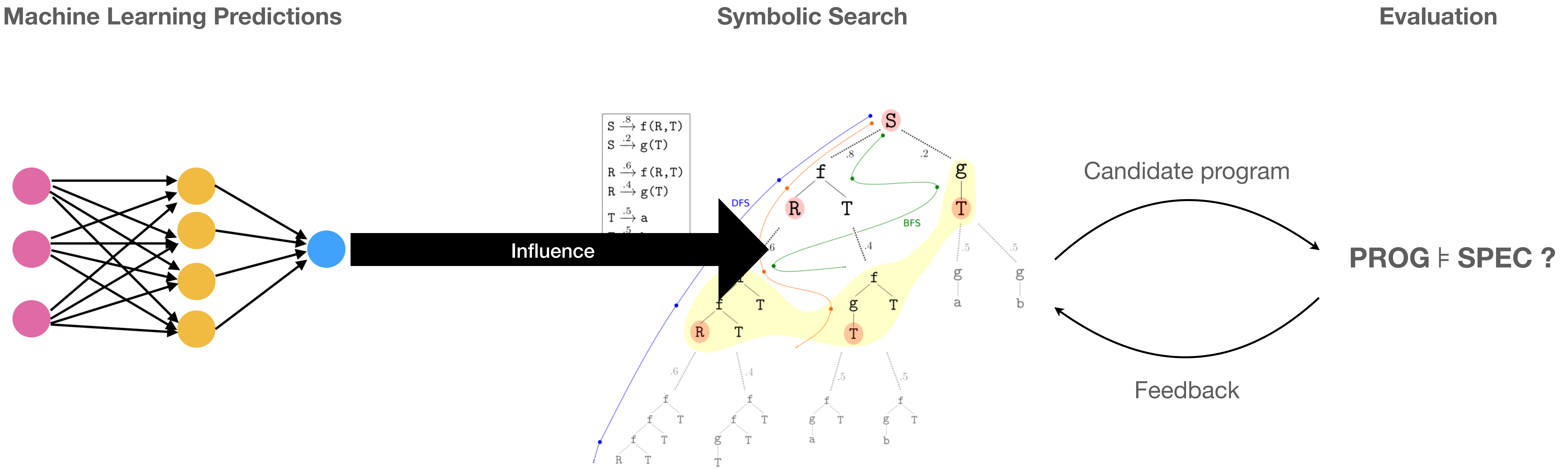


Neural-Guided Program Synthesis

Combination of formal methods and machine learning

reliable

efficient



First practical instance: DeepCoder, 2017

Natural idea: patterns found in examples reveal which primitives are used in a solution program

[1, 5, 4, 2] → [2, 4]

[6, 3, 0, 8] → [0, 6, 8]

Likely primitives such that:

- SORT
- FILTER[EVEN]
- MULTIPLY[2]
- …

Program representation

DSL: domain specific language

- Designed for a specific class of tasks
- List of primitives + associated types + semantics

DSL

sort : list(int) → list(int)
car : list(t) → t
cdr : list(t) → list(t)
map : (t1 → t2) → list(t1) → list(t2)
range : int → list(int)
+ : int → int → int
1 : int
⋮

SyGuS
Syntax-Guided Synthesis

SyGUS in its own words

The Problem

A Syntax-Guided Synthesis problem (SyGuS, in short) is specified with respect to a background theory $\mathbb{T}$, such as Linear-Integer-Arithmetic (LIA), that fixes the types of variables, operations on types, and their interpretation.

To synthesize a function f of a given type, the input consists of two constraints: **(1)** a *semantic constraint* given as a formula φ built from symbols in theory $\mathbb{T}$ along with f, and **(2)** a *syntactic constraint* given as a (possibly infinite) set $\mathcal{E}$ of expressions from $\mathbb{T}$ specified by a context-free grammar.
The computational problem then is to find an implementation for the function f, *i.e.* an expression $e \in \mathcal{E}$ such that the formula $\varphi[f \leftarrow e]$ is valid.

DSL —> Context-free Grammar

DSL

- sort : list(int) → list(int)
- car : list(t) → t
- cdr : list(t) → list(t)
- map : (t1 → t2) → list(t1) → list(t2)
- range : int → list(int)
- + : int → int → int
- 1 : int
- ⋮

Syntactic constraints

- program type list(int) → int
- program depth ≤ 6
- type size ≤ 10
- type nesting ≤ 4
- variable depth ≥ 3
- no two consecutive sort
- ⋮

compilation

CFG

```
S → I,0
```

```
I,d → one
I,d → car_I(L[I],d+1)
I,d → plus(I,d+1; I,d+1)
      ⋮
                              d=0,1,2,3,4,5
```

```
L[I],d → range(I,d+1)
L[I],d → var    if d ≥ 3
L[I],d → sort(L[I],d+1)
L[I],d → cdr_I(L[I],d+1)
L[I],d → car_L[I](L[L[[I]],d+1)
L[I],d → map_I,I(ItoI,d+1;
                 L[I],d+1)
      ⋮
                              d=0,1,2,3,4,5
```

```
ItoI,d → car_ItoI(L[ItoI],d+1)
ItoI,d → plus(I,d+1)
      ⋮
                              d=0,1,2,3,4,5
```

From CFG to PCFG

CFG

$$S \rightarrow \quad f(S,S) \;+\; g(T)$$

$$T \rightarrow \quad a \;+\; b$$

From CFG to PCFG

CFG

$$S \to f(S,S) + g(T)$$
$$T \to a + b$$

PCFG

$$S \to \frac{1}{4} \cdot f(S,S) + \frac{3}{4} \cdot g(T)$$
$$T \to \frac{2}{3} \cdot a + \frac{1}{3} \cdot b$$

From CFG to PCFG

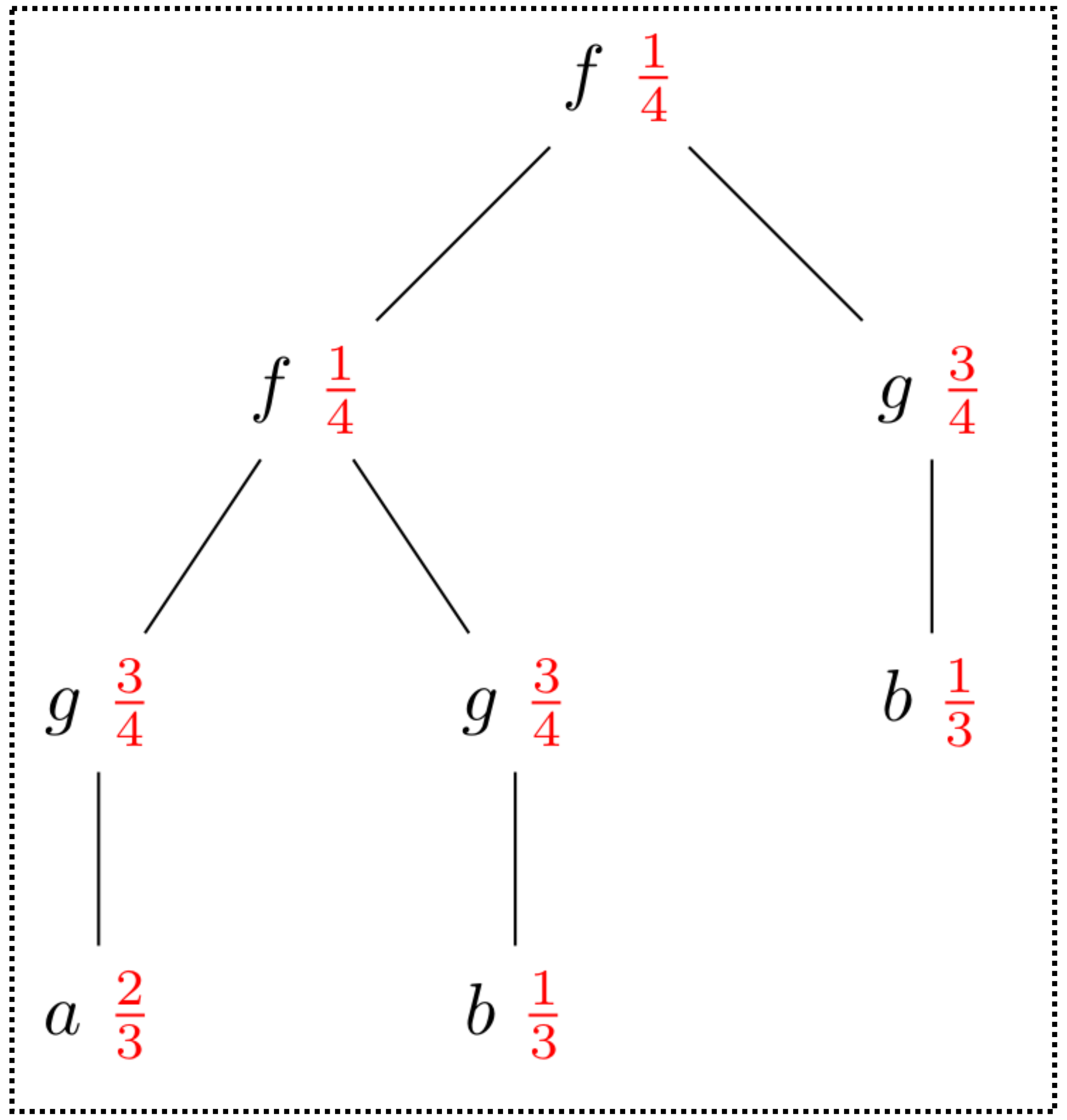
A PCFG *induces* a distribution D over **trees = programs**

CFG

$$S \to \quad f(S,S) \; + \; g(T)$$
$$T \to \quad a \; + \; b$$

PCFG

$$S \to \frac{1}{4} \cdot f(S,S) \; + \frac{3}{4} \cdot g(T)$$
$$T \to \frac{2}{3} \cdot a \; + \frac{1}{3} \cdot b$$



Probability $= \frac{1}{4} \cdot \frac{1}{4} \cdot \frac{3}{4} \cdot \frac{2}{3} \cdots$

Predictions: learning the PCFG

Examples

[1, 5, 4, 2] → [2, 4]

[6, 3, 0, 8] → [0, 6, 8]

→ **A prediction model** →

Probabilistic weights for the CFG

$$S \to \frac{1}{4} \cdot f(S, S) \ + \frac{3}{4} \cdot g(T)$$

$$T \to \frac{2}{3} \cdot a \ + \frac{1}{3} \cdot b$$

Predictions: learning the PCFG

$Pr(SORT \mid FILTER)$ is high
$Pr(SORT \mid REV)$ is low

Examples

[1, 5, 4, 2] → [2, 4]

[6, 3, 0, 8] → [0, 6, 8]

→ A prediction model →

Probabilistic weights for the CFG

$$S \to \frac{1}{4} \cdot f(S, S) \; + \frac{3}{4} \cdot g(T)$$
$$T \to \frac{2}{3} \cdot a \; + \frac{1}{3} \cdot b$$

Predictions: learning the PCFG

$Pr(SORT \mid FILTER)$ is high
$Pr(SORT \mid REV)$ is low

Examples

[1, 5, 4, 2] → [2, 4]

[6, 3, 0, 8] → [0, 6, 8]

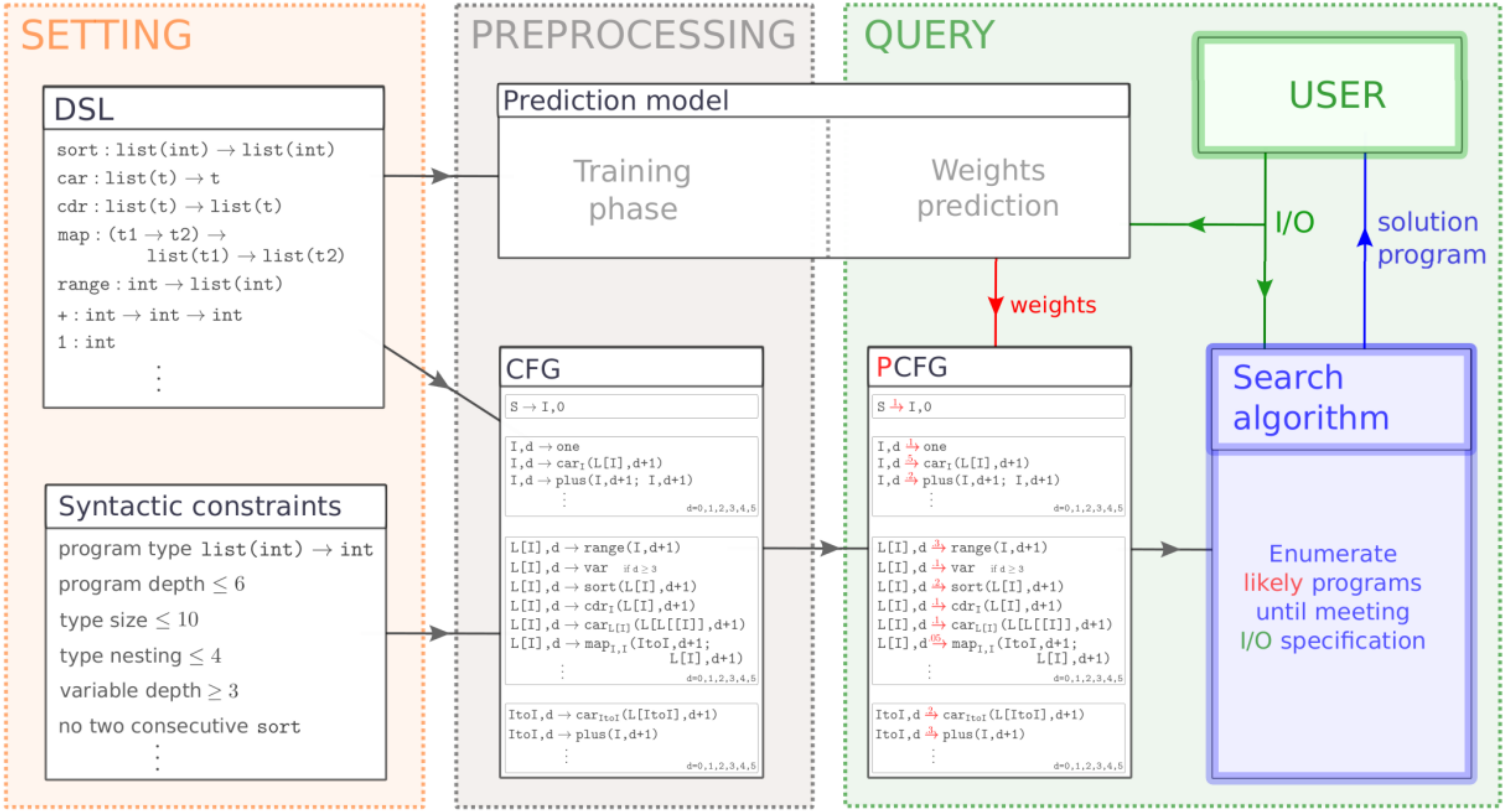
→ **A prediction model** →

Probabilistic weights for the CFG

$$S \to \frac{1}{4} \cdot f(S, S) \ + \frac{3}{4} \cdot g(T)$$
$$T \to \frac{2}{3} \cdot a \ + \frac{1}{3} \cdot b$$

The prediction model induces a **prior** distribution $D = Pr(program \mid examples)$

Summary so far



We are here!

How to use predictions for the search?

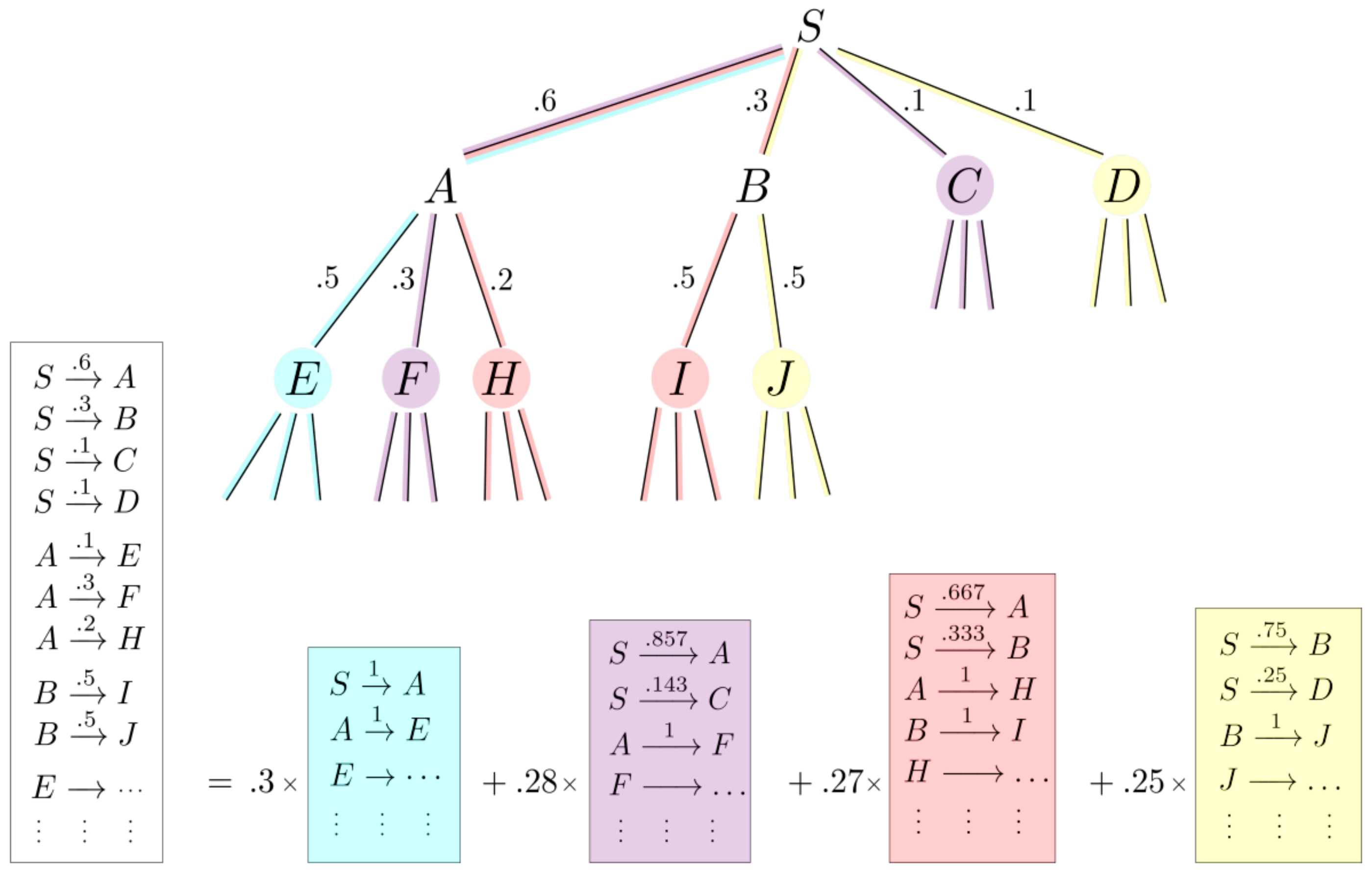
Toy example on the board

Technical contributions

Technical contributions

Heap Search.
Bottom-up enumeration strategy preserving the order on programs
- **Idea**: collections of heaps for storing partial programs
- **Guarantee**: i-th program in time $O(\log i)$

Grammar Splitting.
Divide the search on a parallel architecture
- **Idea**: partition derivations in a fair way



Technical contributions

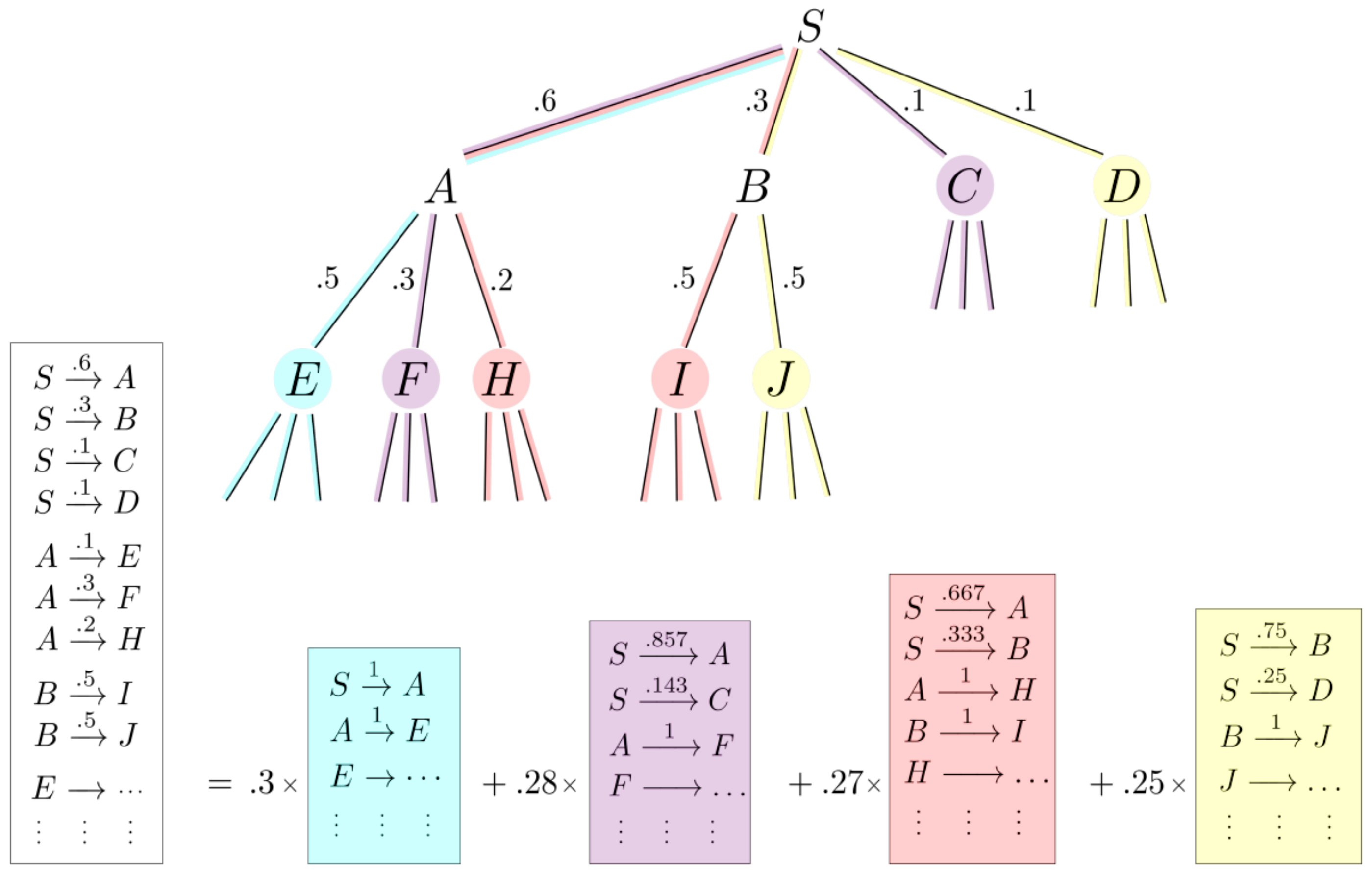
Heap Search.
Bottom-up enumeration strategy preserving the order on programs

- **Idea**: collections of heaps for storing partial programs
- **Guarantee**: i-th program in time $O(\log i)$

Is it optimal?

Grammar Splitting.
Divide the search on a parallel architecture

- **Idea**: partition derivations in a fair way



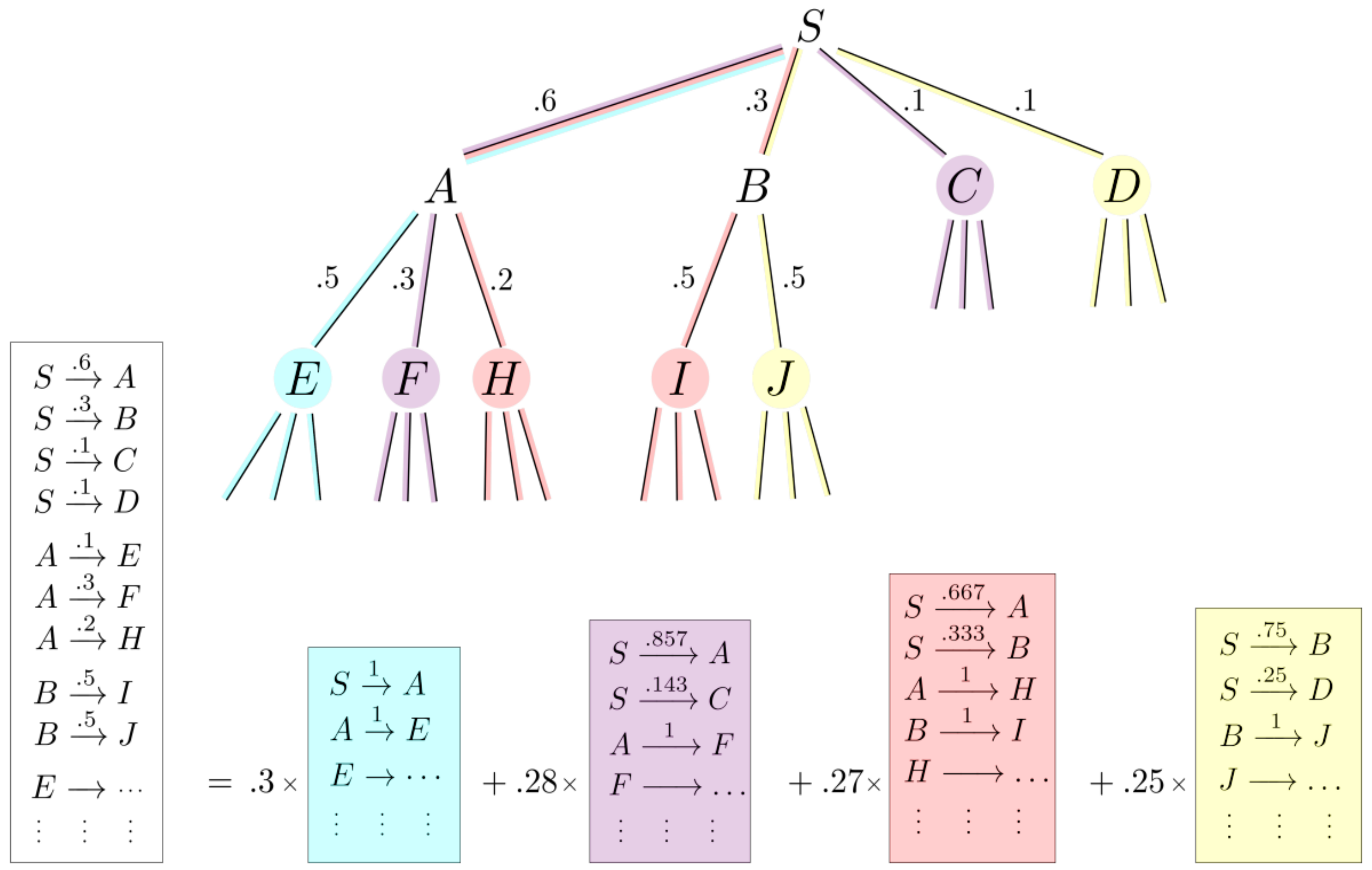
Technical contributions

Heap Search.
Bottom-up enumeration strategy preserving the order on programs
- **Idea**: collections of heaps for storing partial programs
- **Guarantee**: i-th program in time $O(\log i)$

Is it optimal?

SQRT Sampling.
Optimal sampling strategy with no memory
Interesting for simplicity and parallel search
- **Idea**: sample programs according to $D' \propto \sqrt{D}$

Grammar Splitting.
Divide the search on a parallel architecture
- **Idea**: partition derivations in a fair way



Technical contributions

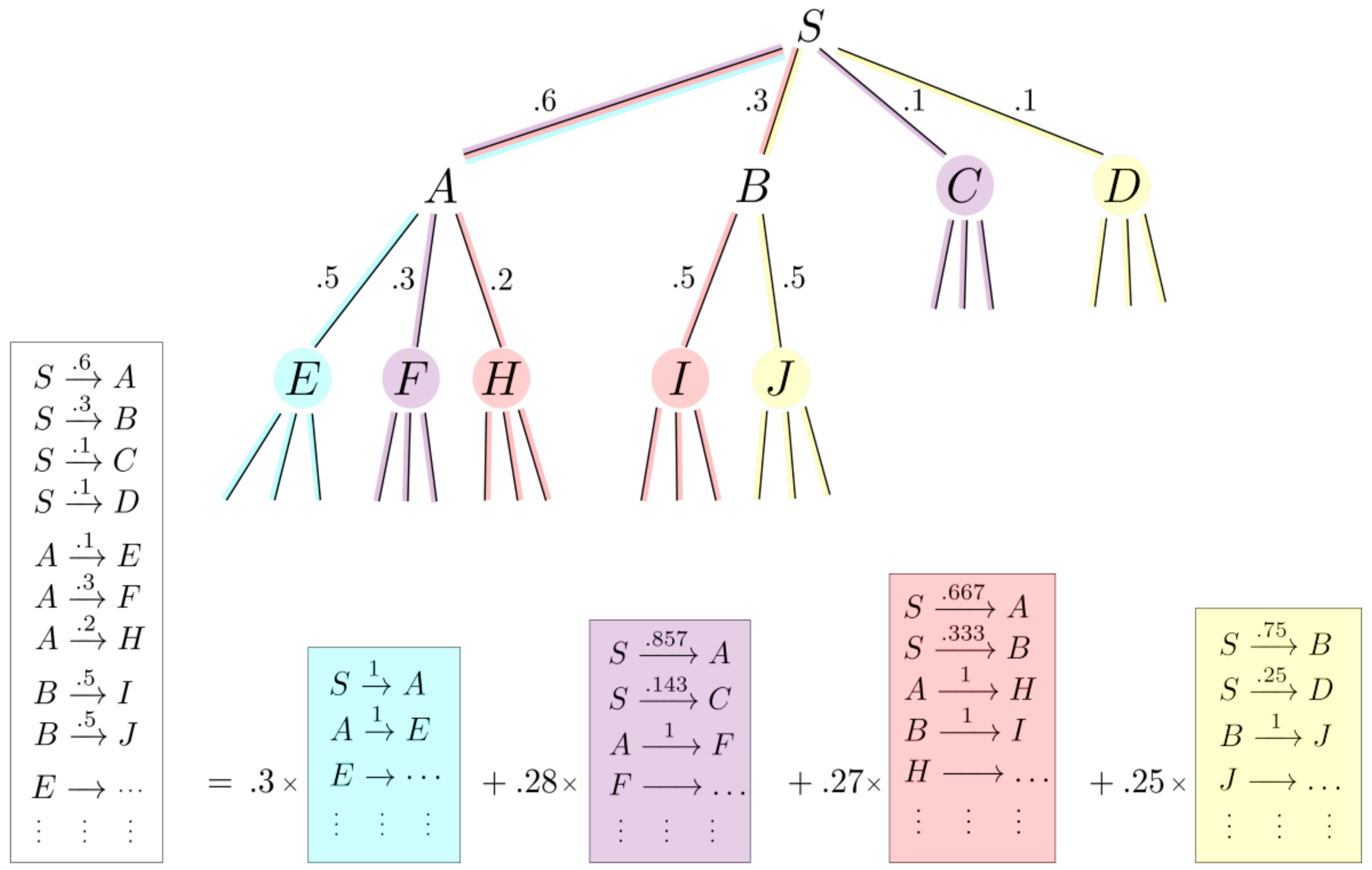
Heap Search.
Bottom-up enumeration strategy preserving the order on programs
- **Idea**: collections of heaps for storing partial programs
- **Guarantee**: i-th program in time $O(\log i)$

Is it optimal?

SQRT Sampling.
Optimal sampling strategy with no memory
Interesting for simplicity and parallel search
- **Idea**: sample programs according to $D' \propto \sqrt{D}$

How to sample? —> Construct a new PCFG for $\sqrt{D}$

Grammar Splitting.
Divide the search on a parallel architecture
- **Idea**: partition derivations in a fair way



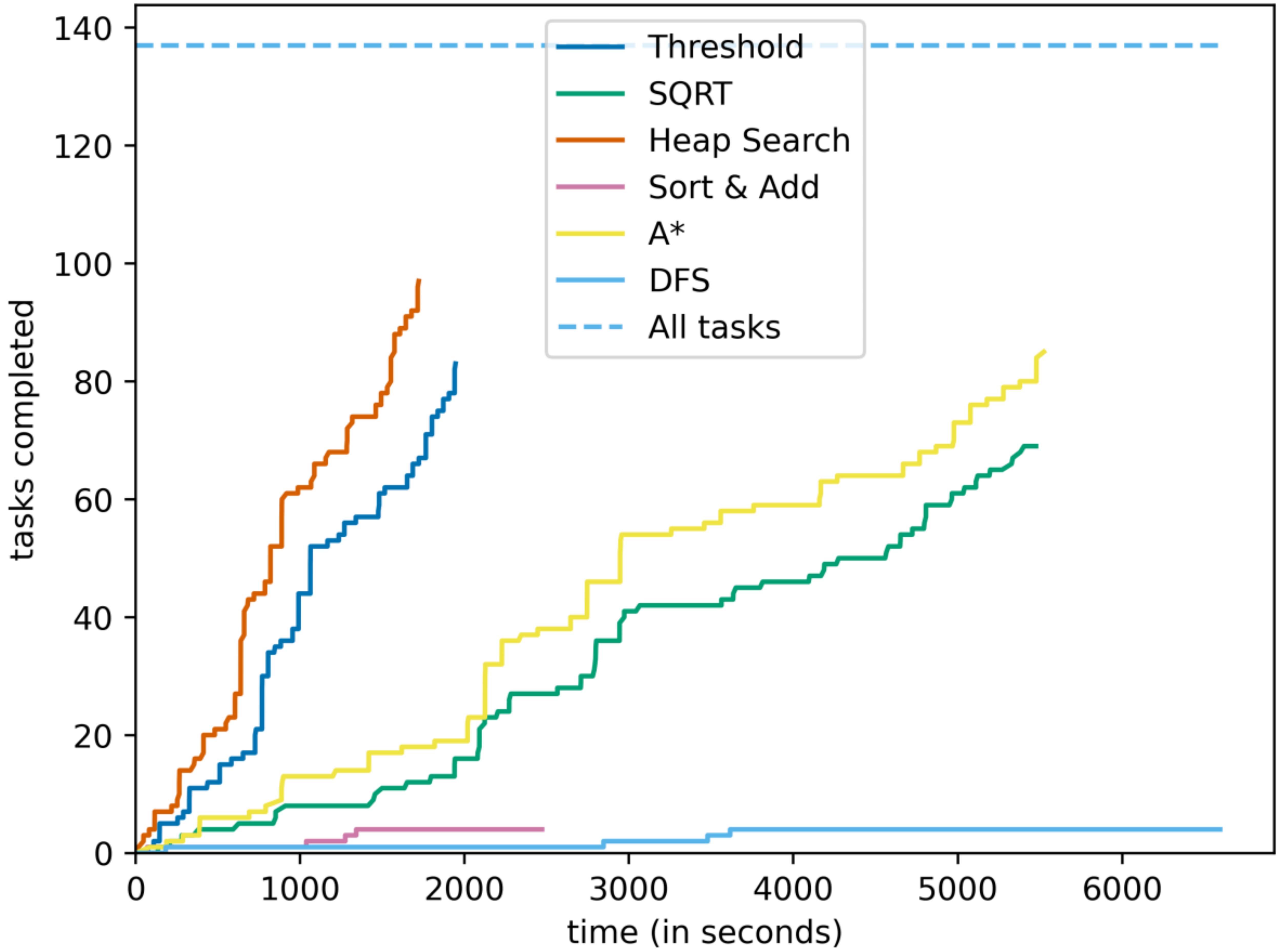


Figure 7: Comparing all search algorithms on the DreamCoder reduced dataset with machine-learned PCFGs

Thanks!

- Generic pipeline to do symbolic search on grammars enhanced with Machine Learning Predictions
- Check out **DeepSynth** (open-source tool) — https://deepsynth.labri.fr/
- Is heap search optimal (from an enumeration complexity POV)?
- Better sampling if using memory?
- Use feedback to refine the search?

Example

$$\mathcal{G}\ \begin{cases} S \longrightarrow \frac{1}{2} f(S,T) \ \mid\ \frac{1}{2}\, a \\ T \longrightarrow b \end{cases}$$

Natural candidate:

$$\mathcal{G}'\ \begin{cases} S \longrightarrow \frac{1}{\sqrt{2}} f(S,T) \ \mid\ \frac{1}{\sqrt{2}}\, a \\ T \longrightarrow b \end{cases}$$

$\mathcal{G}'$ indeed computes $\sqrt{\mathcal{D}}$ but it is not a PCFG, it needs **renormalisation**

for X non-terminal $(X = S, T)$ define

$$[X] = \sum \{ \mathcal{D}(t) \ : \ t \text{ generated from } X \}$$

$$\begin{cases} [S] = \frac{1}{\sqrt{2}} [S][T] + \frac{1}{\sqrt{2}} \\ [T] = 1 \end{cases} \implies \begin{cases} [S] = 1 + \sqrt{2} \\ [T] = 1 \end{cases}$$

The PCFG computing $\sqrt{\mathcal{D}}$ is

$$\sqrt{\mathcal{G}}\ \begin{cases} S \longrightarrow \frac{1}{1+\sqrt{2}} \left(\frac{1}{\sqrt{2}} f(S,T) \ \mid\ \frac{1}{\sqrt{2}}\, a \right) \\ T \longrightarrow b \end{cases}$$