

# DreamCoder: Bootstrapping Domain-Specific Languages for Neurally-Guided Bayesian Program Learning

Kevin Ellis, Lucas Morales, Mathias Sablé Meyer, Maxwell Nye, Luke Hewitt,  
Armando Solar-Lezama, Joshua B. Tenenbaum

Massachusetts Institute of Technology & MIT-IBM Watson AI lab

## Wake/Sleep DSL Induction

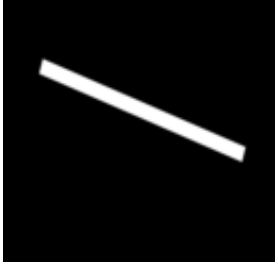



**Domain Specific Language (DSL):** A finely-tuned program representation, specialized to a domain of programming tasks. Prior work in program learning largely uses hand-engineered DSLs.

**Approach:** DREAMCODER algorithm, which bootstraps a learned DSL while jointly training a neural net to search for programs in the learned DSL. Given a few hundred programming tasks, alternately:

- **Wake:** synthesize programs
- **Sleep-R:** train neural net (Recognition model)
- **Sleep-G:** improve DSL (Generative model)

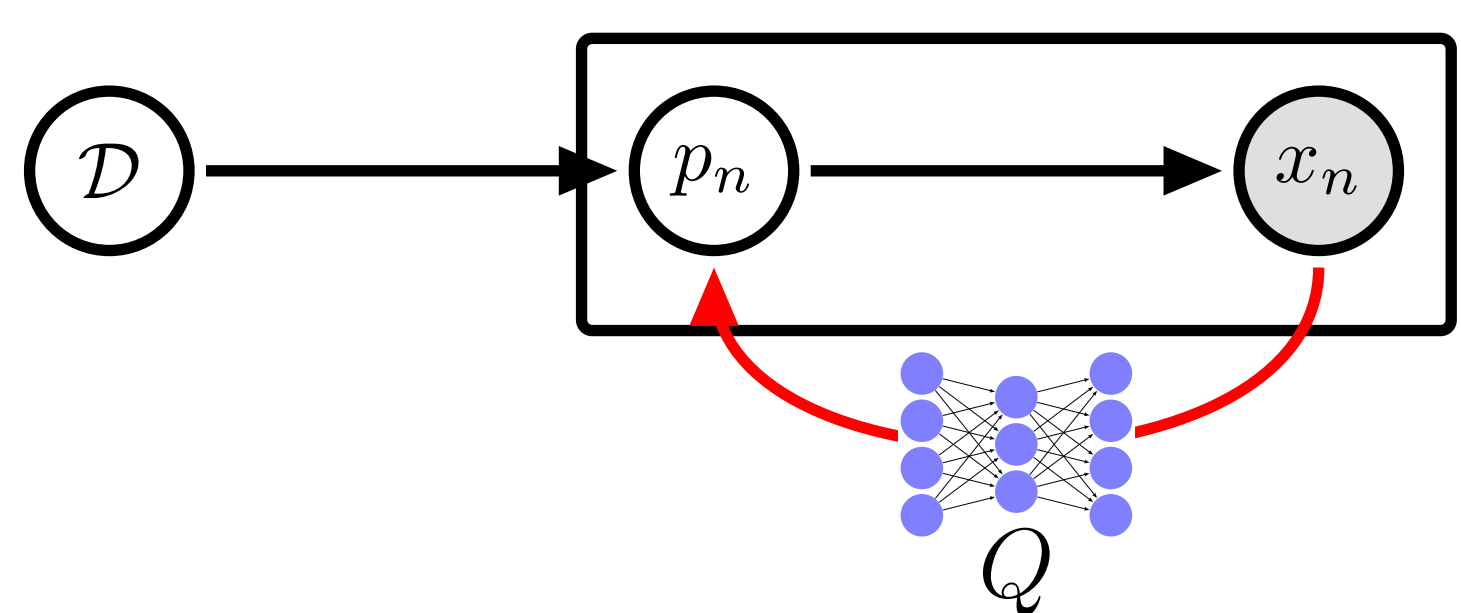
**Program representation:**  
 $\approx$ Lisp; conditionals, variables,  $\lambda$  abstraction

## Model outputs for three different task domains

	List Functions	Text Editing	Symbolic Regression
Programs & Tasks	$[7\ 2\ 3] \rightarrow [7\ 3]$ $[1\ 2\ 3\ 4] \rightarrow [3\ 4]$ $[4\ 3\ 2\ 1] \rightarrow [4\ 3]$ $f(\ell) = (f_1\ \ell\ (\lambda\ (x)\ (>\ x\ 2)))$	$+106\ 769\text{-}438 \rightarrow 106.769.438$ $+83\ 973\text{-}831 \rightarrow 83.973.831$ $f(s) = (f_0\ \text{"."}\ \text{"-"}\ (f_0\ \text{"."}\ \text{" "}\ (\text{cdr}\ s)))$	  $f(x) = (f_1\ x)$ $f(x) = (f_6\ x)$
	$[2\ 7\ 8\ 1] \rightarrow 8$ $[3\ 19\ 14] \rightarrow 19$ $f(\ell) = (f_2\ \ell)$	$[7\ 3] \rightarrow \text{False}$ $[3] \rightarrow \text{False}$ $[9\ 0\ 0] \rightarrow \text{True}$ $[0] \rightarrow \text{True}$ $[0\ 7\ 3] \rightarrow \text{True}$ $f(\ell) = (f_3\ \ell\ 0)$	  $f(x) = (f_4\ x)$ $f(x) = (f_3\ x)$
DSL	$f_1(\ell, p) = (\text{foldr}\ \ell\ \text{nil}\ (\lambda\ (x\ a)\ (\text{if}\ (p\ x)\ (\text{cons}\ x\ a)\ a))))$ $(f_1: \text{Higher-order filter function})$	$f_0(s, a, b) = (\text{map}\ (\lambda\ (x)\ (\text{if}\ (= x\ a)\ b\ x))\ s)$ $(f_0: \text{substitutes characters})$	$f_0(x) = (+\ x\ \text{real})$ $f_1(x) = (f_0\ (*\ \text{real}\ x))$ $f_2(x) = (f_1\ (*\ x\ (f_0\ x)))$ $f_3(x) = (f_0\ (*\ x\ (f_2\ x)))$ $f_4(x) = (f_0\ (*\ x\ (f_3\ x)))$ $(f_4: \text{4th order polynomial})$
	$f_2(\ell) = (\text{foldr}\ \ell\ 0\ (\lambda\ (x\ a)\ (\text{if}\ (>\ a\ x)\ a\ x)))$ $(f_2: \text{Maximum element in list } \ell)$	$f_1(s, c) = (\text{foldr}\ s\ s\ (\lambda\ (x\ a)\ (\text{cdr}\ (\text{if}\ (= c\ x)\ s\ a))))$ $(f_1: \text{Drop characters from } s \text{ until } c \text{ reached})$	$f_5(x) = (/ \text{real}\ x)$ $f_6(x) = (f_5\ (f_0\ x))$ $(f_6: \text{rational function})$
	$f_3(\ell, k) = (\text{foldr}\ \ell\ (\text{is-nil}\ \ell)\ (\lambda\ (x\ a)\ (\text{if}\ a\ a\ (= k\ x))))$ $(f_3: \text{Whether } \ell \text{ contains } k)$	$f_2(s) = (\text{unfold}\ s\ \text{is-nil}\ \text{car}\ (\lambda\ (z)\ (f_1\ z\ \text{" "})))$ $(f_2: \text{Abbreviates words})$	

Top: Tasks from three domains we apply our algorithm to, each followed by the programs DREAMCODER discovers for them. Bottom: Several examples from learned DSL. Notice that learned DSL primitives can call each other, and that DREAMCODER rediscovers higher-order functions like **filter** ( $f_1$  under List Functions)

## Bayesian framing



Observe  $N$  **tasks**, written  $\{x_n\}_{n=1}^N$ , each a program synthesis problem.

Solve task  $x_n$  with latent program  $p_n$

**Likelihood model**  $\mathbb{P}[x_n|p_n]$  scores program  $p_n$  on task  $x_n$

Latent **DSL**  $\mathcal{D}$  acts as generative model over programs:  $\mathbb{P}[p_n|\mathcal{D}]$

$$\underbrace{p_n^* = \arg \max_{p_n} \mathbb{P}[x_n|p_n] \mathbb{P}[p_n|\mathcal{D}^*]}_{\text{Wake}}$$

$$\underbrace{\mathcal{D}^* = \arg \max_{\mathcal{D}} \mathbb{P}[\mathcal{D}] \prod_n \sum_{p_n} \mathbb{P}[x_n|p_n] \mathbb{P}[p_n|\mathcal{D}]}_{\text{Sleep-G}}$$

## Neural recognition model

Neural network  $Q(p|x)$  predicts distribution over programs conditioned on tasks. Simple  $Q$ : just predicts probabilities of DSL productions. Goal: learn to invert generative model

$$\underbrace{\min_Q \text{KL}(\mathbb{P}[p|x, \mathcal{D}] || \mathbb{Q}(p|x))}_{\text{Sleep-R}}$$

Train on two sources of data:

- **Samples (“Dreams”) from DSL:** Unlimited data, but only high-quality if generative model  $\mathcal{D}$  is good. Like Helmholtz Machine’s recognition model training. Loss:

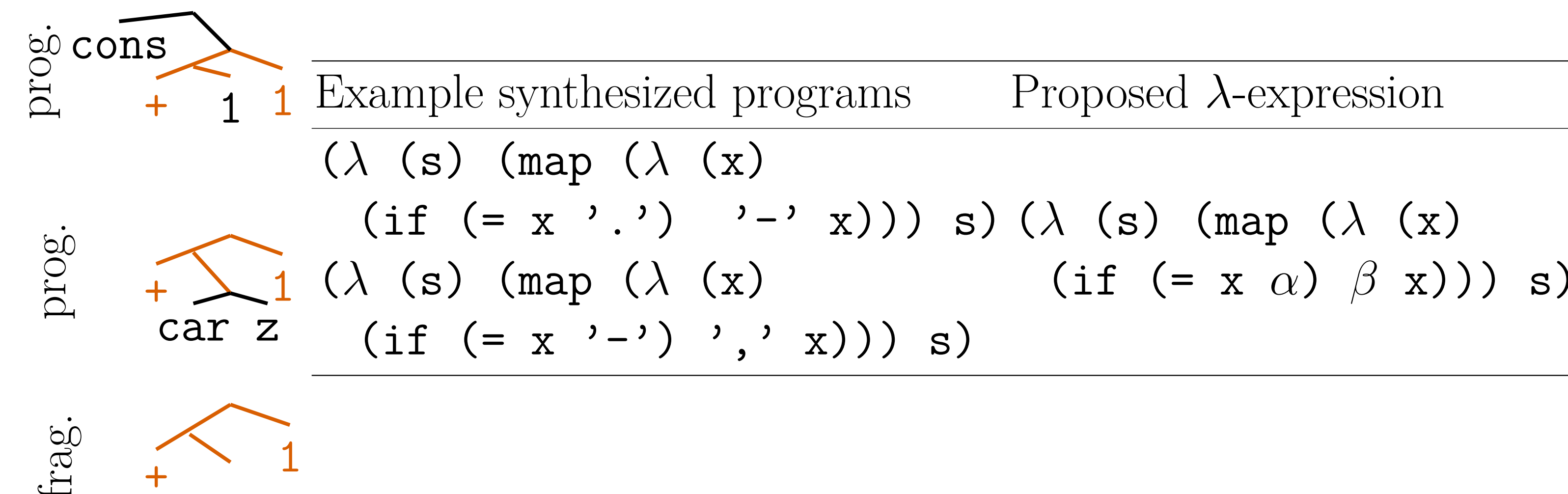
$$\mathbb{E}_{(p,x) \sim \mathcal{D}} [\log \mathbb{Q}(p|x)]$$

- **Self-Supervised:**  $(x_n, p_n)$  pairs discovered during waking. Loss:

$$\frac{\mathbb{P}[x_n, p_n|\mathcal{D}]}{\sum_{(x_n, p'_n)} \mathbb{P}[x_n, p'_n|\mathcal{D}]} \log \mathbb{Q}(p_n|x_n)$$

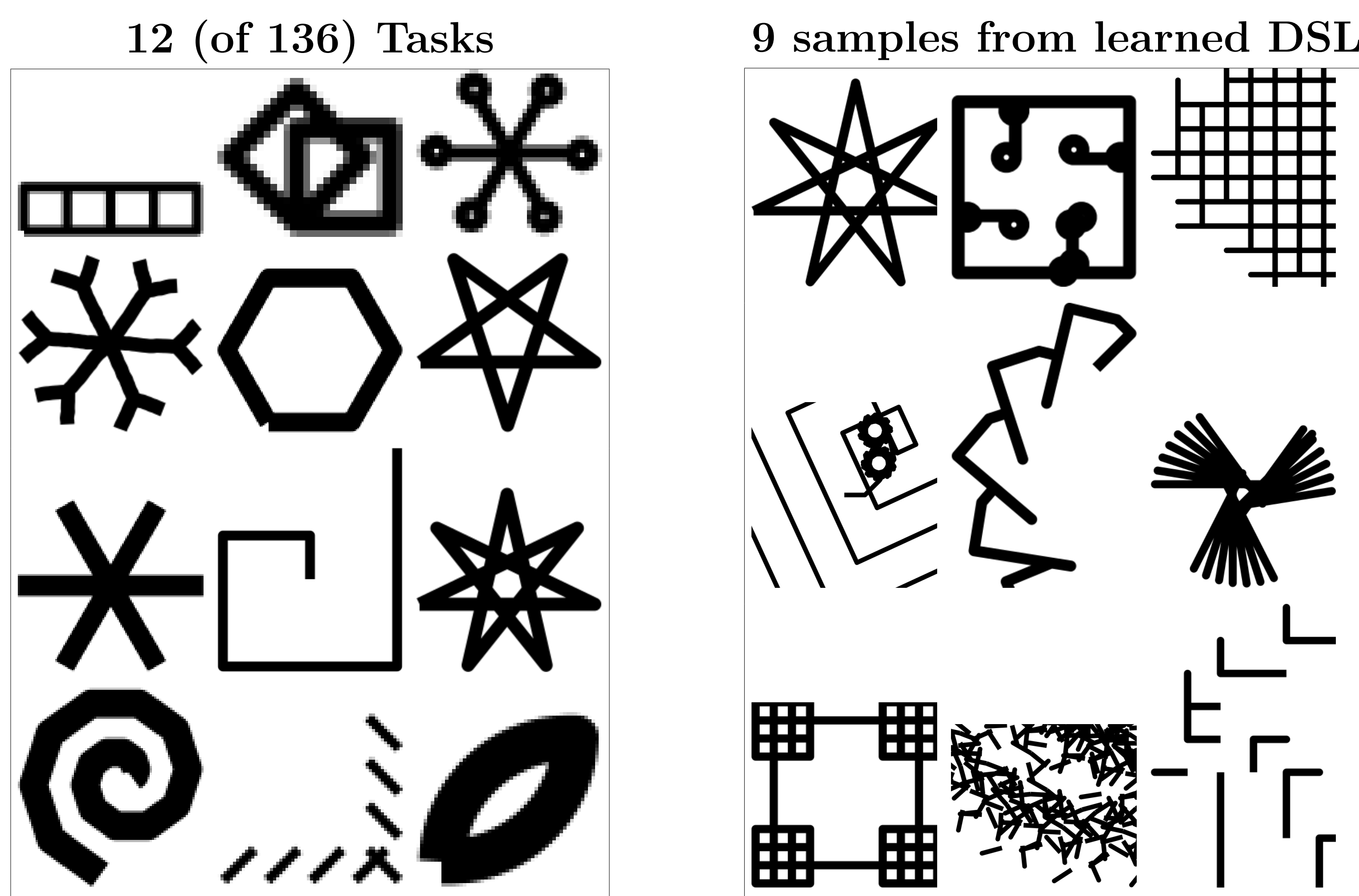
## Fragment Grammars: Inducing a DSL

Fragment grammars: introduced in computational linguistics (O’Donnell 2015)



**Figure 1: Left:** syntax trees of two programs sharing common structure, highlighted in orange, from which we extract a fragment and add it to the DSL (bottom). **Right:** actual programs, from which we extract fragments that perform character substitutions.

## Generative models of images: Turtle/LOGO Graphics

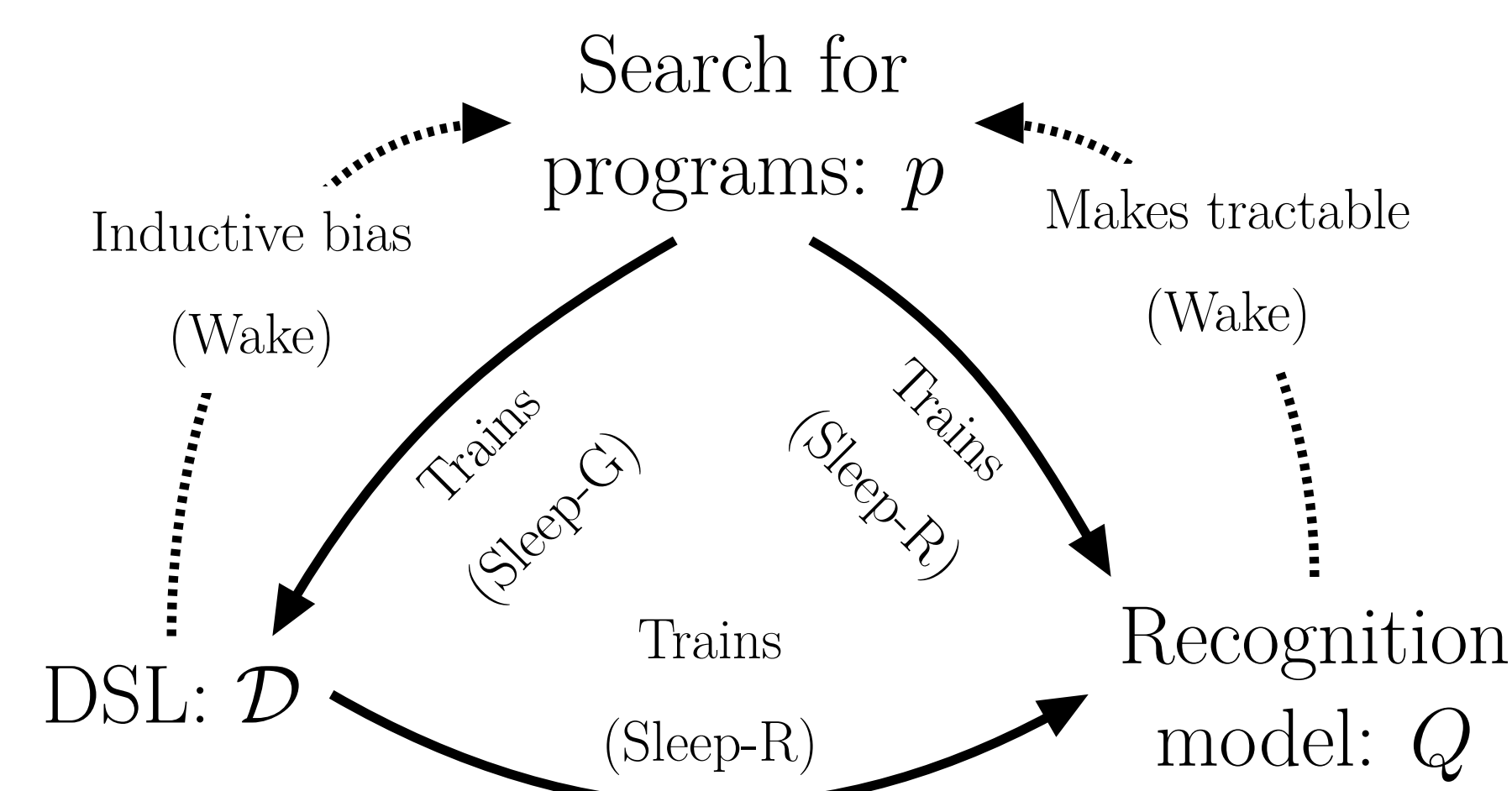


**Left:** Agent controls a ‘pen’ – tasked with drawing pictures. **Right:** During Sleep-R, ‘dream’ by sampling programs from learned DSL and rendering them.

## Acknowledgements

We gratefully acknowledge collaboration with Eyal Dechter, whose EC algorithm (Dechter et al, IJCAI 2013) provided the inspiration for DreamCoder, and Luke Hewitt, who graciously provided us with a regex learning data set.

## Why this works: Bootstrapping



## Generative models of text

Three Tasks		
1.14531	F	110.9
?	CL	163.2
1.29857	F	207.3
?	PCFL	143.3

Three learned generative models		
$? (1.\backslash d+)((\backslash u\backslash u)*) F\backslash d\backslash d\backslash d.\backslash d$		

Samples from generative models		
1.61	DQDF	343.8
?	F	241.2
?	F	647.5
1.2	KI	246.8

Learned DSL		
$f_1 = \backslash u\backslash w*$	$f_2(x) = (x   f_1)*$	
$f_3 = f_2(\text{" "}) = (\backslash u\backslash w*)*$		
$(f_3: \text{whitespace delimited words})$		
$f_4(x) = (x*x)$	$(f_4: \text{regex 'plus'})$	

## Learning from Scratch

Start w/ McCarthy 1959 Lisp: recursion, conditionals, lists. Train on 22 programming exercises. After 93 hours on 64 CPUs, rediscovers: **map**, **fold**, **zip**, **unfold**, **index**, **length**, **range**, **incr**, **decr**.