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

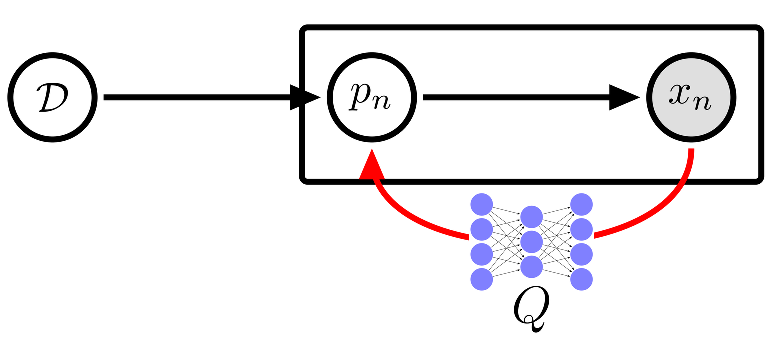
Domain Specific Language (DSL): A finely-tuned program representation, specialized to a domain of programming tasks. Prior work in program learning largely uses hand-engineered DSLs.

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, alternatingly:

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

Program representation: ≈Lisp; conditionals, variables, λ abstraction

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: $\mathbb{P}[x|\mathcal{D}]$

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

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

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

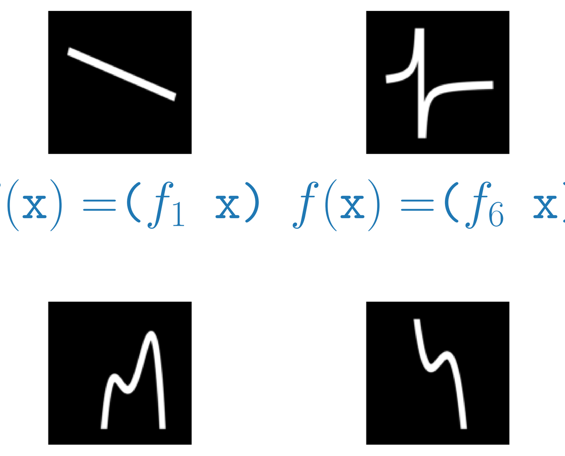
$$\underbrace{\min_Q \mathrm{KL}\left(\mathbb{P}[p|x,\mathcal{D}] || Q(p|x)\right)}_{\text{Sleep-R}}$$

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:
$$\mathbb{E}_{(p,x)\sim\mathcal{D}}\left[\log Q(p|x)\right]$$
- **Self-Supervised**: (x_n, p_n) pairs discovered during waking. Loss:
$$\frac{\mathbb{P}\left[x_n, p_n|\mathcal{D}\right]}{\sum_{(x_n,p_n')}\mathbb{P}\left[x_n,p_n|\mathcal{D}\right]}\log Q(p_n|x_n)$$

Model outputs for three different task domains

	List Functions	Text Editing	Symbolic Regression

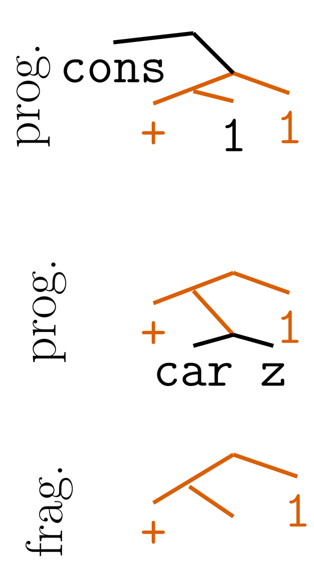
Programs & Tasks	`[7 2 3]→[7 3]`
	`[1 2 3 4]→[3 4]`
	`[4 3 2 1]→[4 3]`
	$f(\ell)$ `=(`f_1 ℓ `(λ (x) (> x 2)))`
	
	`[7 3]→False`
	`[3]→False`
	`[9 0 0]→True`
	`[0]→True`
	`[0 7 3]→True`
	$f(\ell)$ `=(`f_3 ℓ `0)`
	
	`[2 7 8 1]→8`
	`[3 19 14]→19`
	$f(\ell)$ `=(`f_2 ℓ`)`
		+106 769-438→106.769.438
		+83 973-831→83.973.831
		`f(s) =(`f_0 `"." "-" (`f_0 `"." " " (cdr s)))`
		
		Temple Anna H →TAH
		Lara Gregori→LG
		`f(s) =(`f_2 `s)`
			
			`f(x) =(`f_1 `x)` `f(x) =(`f_6 `x)`
			`f(x) =(`f_4 `x)` `f(x) =(`f_3 `x)`

DSL	f_1`(ℓ,p) =(foldr ℓ nil (λ (x a) (if (p x) (cons x a) a)))` (f_1: *Higher-order filter function*) f_2`(ℓ) =(foldr ℓ 0 (λ (x a) (if (> a x) a x)))` (f_2: *Maximum element in list* ℓ) f_3`(ℓ,k) =(foldr ℓ (is-nil ℓ) (λ (x a) (if a a (= k x))))` (f_3: *Whether* ℓ *contains* `k`)	f_0`(s,a,b) =(map (λ (x) (if (= x a) b x)) s)` (f_0: *substitutes characters*) f_1`(s,c) =(foldr s s (λ (x a) (cdr (if (= c x) s a))))` (f_1: *Drop characters from* `s` *until* `c` *reached*) f_2`(s) =(unfold s is-nil car (λ (z) (`f_1 `z " ")))` (f_2: *Abbreviates words*)	f_0`(x) =(+ x real)` f_1`(x) =(`f_0 `(* 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: *4th order polynomial*) f_5`(x) =(/ real x)` f_6`(x) =(`f_5 `(`f_0 `x))` (f_6: *rational function*)

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)

Fragment Grammars: Inducing a DSL

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

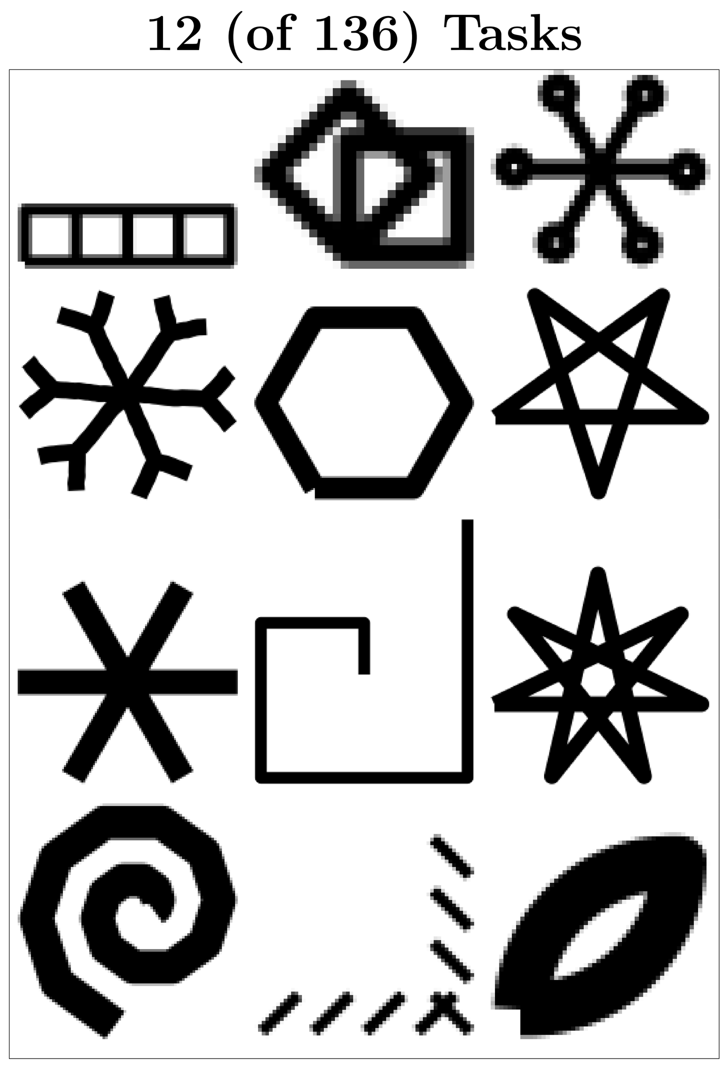


Example synthesized programs	Proposed λ-expression
`(λ (s) (map (λ (x) (if (= x '.') '-' x))) s)` `(λ (s) (map (λ (x) (if (= x '-') ',' x))) s)`	`(λ (s) (map (λ (x) (if (= x α) β x))) s)`

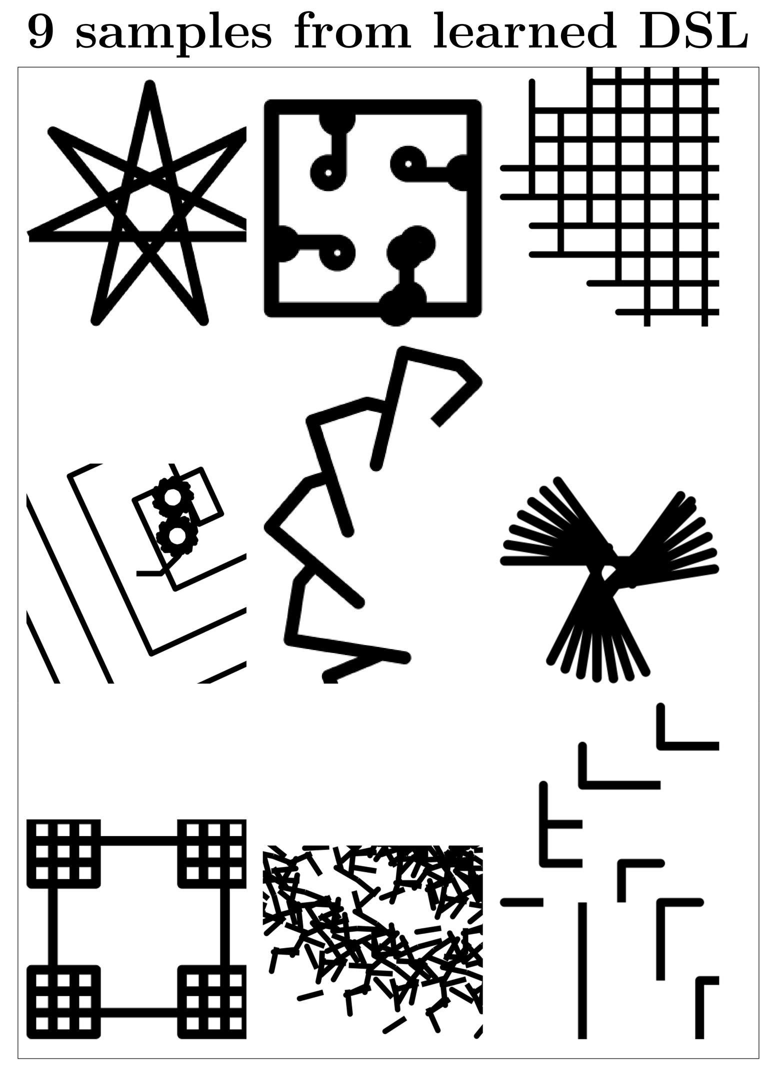
Figure 1: **Left:** syntax trees of two programs sharing common structure, highlighted in orange, from which we extract a fragment and add it to the DSL (bottom). **Right:** actual programs, from which we extract fragments that perform character substitutions.

Generative models of images: Turtle/LOGO Graphics

12 (of 136) Tasks



9 samples from learned DSL

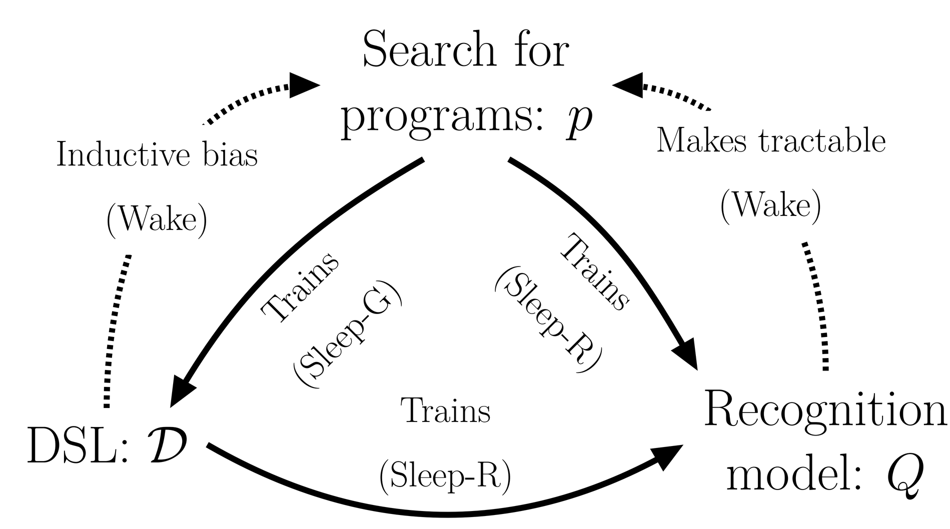


Left: Agent controls a 'pen' – tasked with drawing pictures. **Right:** During Sleep-R, 'dream' by sampling programs from learned DSL and rendering them.

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.\d+)`	`((\u\u)*)|F`	`\d\d\d.\d`

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

Learned DSL

$f_1 =$ `\u\w*` $f_2(x) =$`(`x`|`f_1`)*`
$f_3 = f_2($`" "`$) =$`( |\u\w*)*`
(f_3: *whitespace delimited words*)
$f_4(x) =$ `(`x`*`x`)` (f_4: *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`.