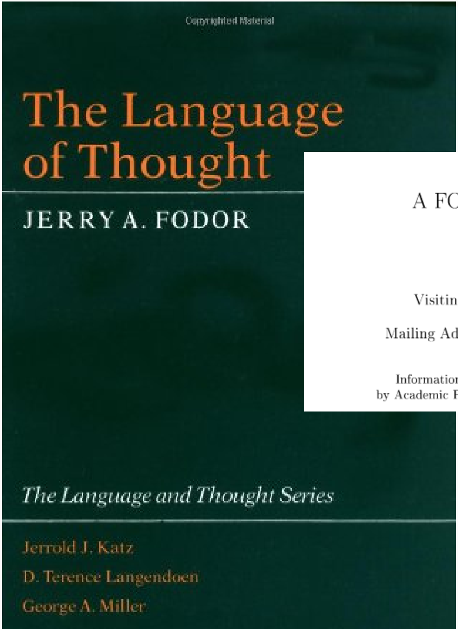
Growing libraries of concepts with wake-sleep program induction

Kevin Ellis
Joint with: Lucas Morales, Mathias Sablé Meyer, Armando Solar-Lezama, Joshua B. Tenenbaum
Heavy inspiration from: Eyal Dechter

October 2018

MIT

The Language of Thought



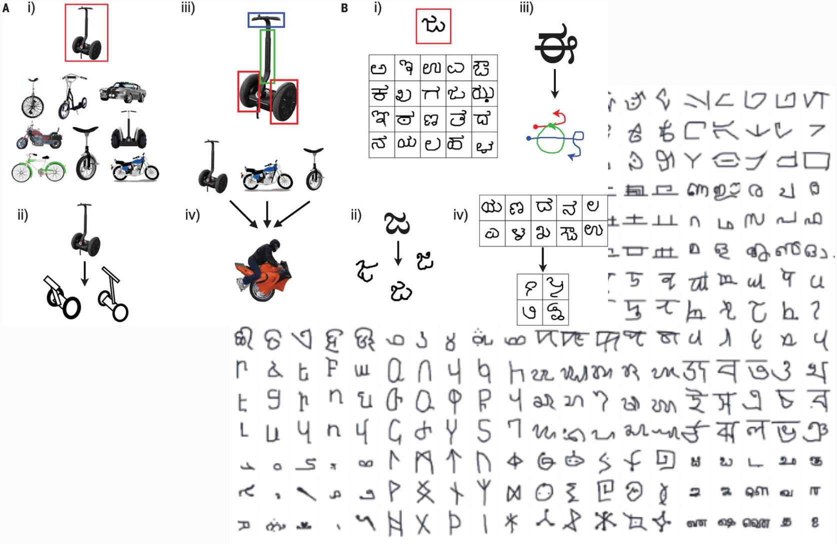
A FORMAL THEORY OF INDUCTIVE INFERENCE, Part I*†

Ray J. Solomonoff

Visiting Professor, Computer Learning Research Center
Royal Holloway, University of London
Mailing Address: P.O.B. 400404, Cambridge, Ma. 02140, U.S.A.

Information and Control, Volume 7, No. 1, Pp. 1-22, March 1964 Copyright by Academic Press Inc.

Engineering the language of thought



Lake et al 2015

Growing a domain-specific language of thought

Goal: acquire domain-specific knowledge needed to induce a class of programs

Growing a domain-specific language of thought

Goal: acquire domain-specific knowledge needed to induce a class of programs

- Library of concepts (declarative knowledge)
- Inference strategy (procedural knowledge)

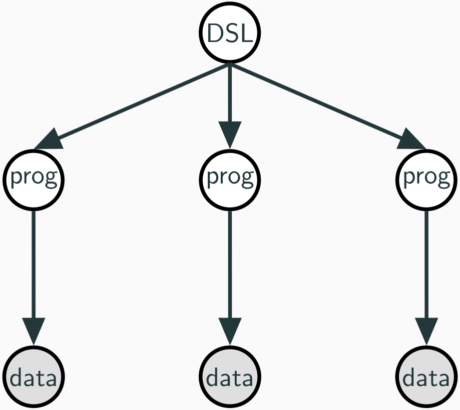
DSL: Library of concepts

Tasks and Programs	DSL
[7 2 3]→[7 3] [1 2 3 4]→[3 4] [4 3 2 1]→[4 3] $f(\ell) = $ (f₁ ℓ (λ (x) (> x 2))) [7 3]→False [3]→False [9 0 0]→True [0]→True [0 7 3]→True $f(\ell) = $ (f₃ ℓ 0) [2 7 8 1]→8 [3 19 14]→19 $f(\ell) = $ (f₂ ℓ)	$f_0(\ell, \mathtt{r}) = $ (foldr r ℓ cons) (f_0: *Append lists* r *and* ℓ) $f_1(\ell, \mathtt{p}) = $ (foldr ℓ nil (λ (x a) (if (p x) (cons x a) a))) (f_1: *Higher-order filter function*) $f_2(\ell) = $ (foldr ℓ 0 (λ (x a) (if (> a x) a x))) (f_2: *Maximum element in list* ℓ) $f_3(\ell, \mathtt{k}) = $ (foldr ℓ (is-nil ℓ) (λ (x a) (if a a (= k x)))) (f_2: *Whether* ℓ *contains* k)

- **Wake:** Solve problems by writing programs
- **Sleep:** Improve DSL and neural recognition model:
  - **Sleep-G:** Improve DSL (**G**enerative model)
  - **Sleep-R:** Improve **R**ecognition model

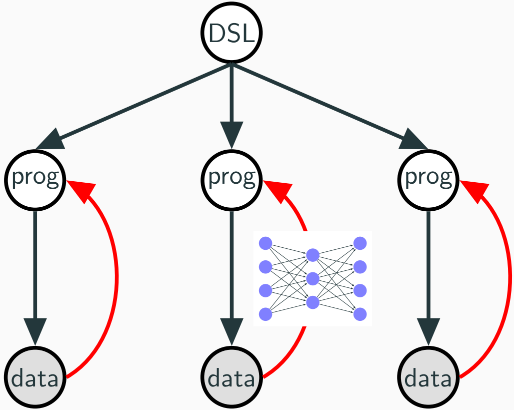
Combines ideas from Wake-Sleep & Exploration-Compression algorithm by Eyal Dechter

DSL learning as Bayesian inference

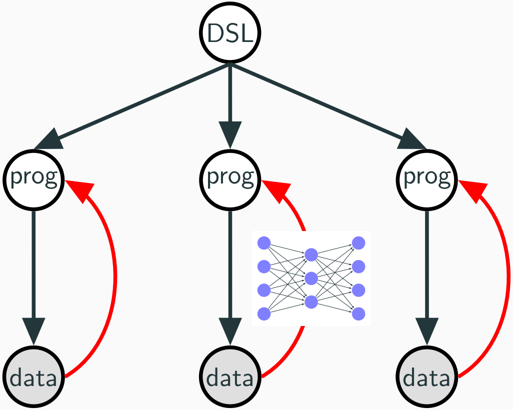


[Dechter et al., 2013] [Liang et al, 2010]; [Lake et al, 2015]

DSL learning as amortized Bayesian inference

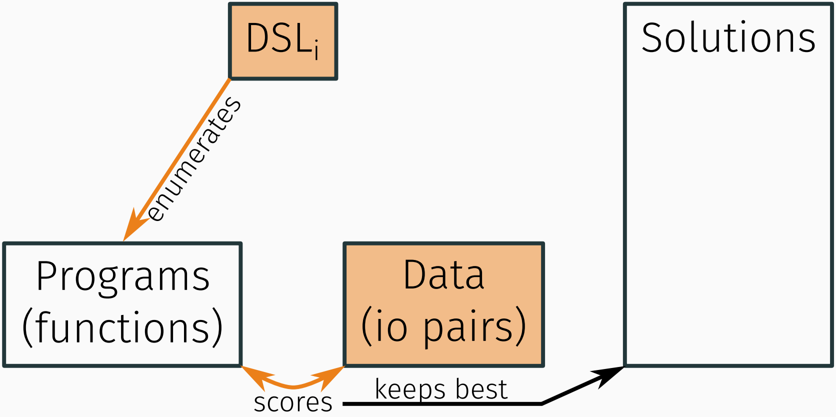


DSL learning as amortized Bayesian inference



New: amortized inference +
better program representation (Lisp) +
better DSL inference

Wake — as in Exploration/Compression Algorithm



DSL

```
f0(l,r) = foldr r l cons
f1(l,p) = foldr r nil
           (λ (x a)
              (if (p x)
                 (cons x a)
                 a))
```

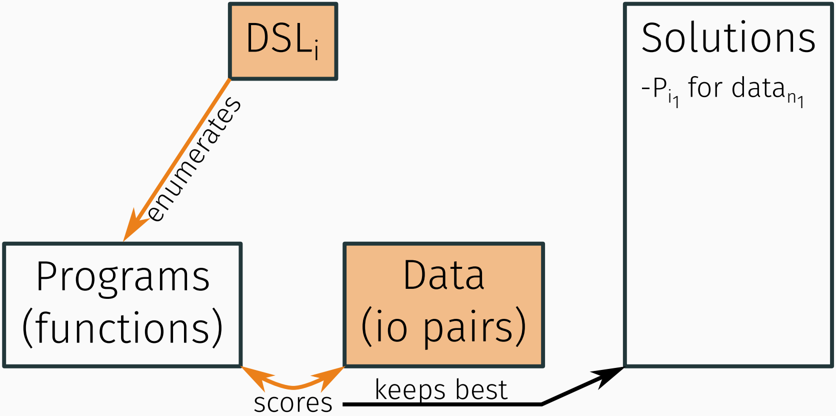
Programs

```
f(l) = (f0 l l)

f(l) = (f1 l (λx. > x 2))
```

Data

```
[7 2]→[7 2 7 2]
["a"]→["a" "a"]

[7 2 3]→[7 3]
[1 2 3 4]→[3 4]
[4 3 2 1]→[4 3]
```

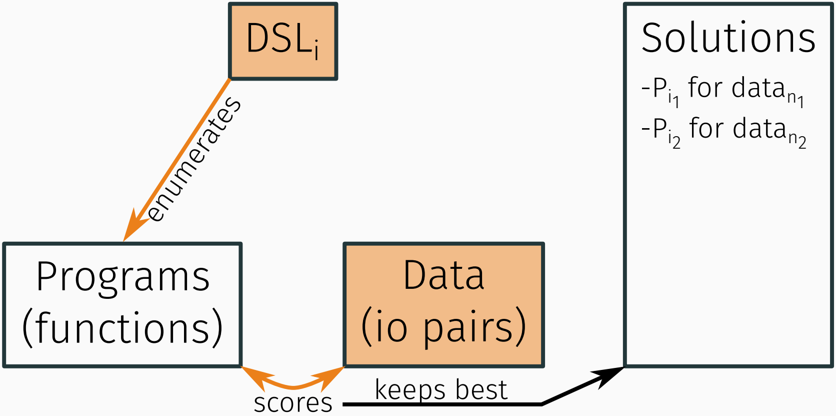
Wake — as in Exploration/Compression Algorithm



DSL	Programs	Data
`f0(l,r) = foldr r l cons`	`f(l) = (f0 l l)`	`[7 2]→[7 2 7 2]` `["a"]→["a" "a"]`

`f1(l,p) = foldr r nil (λ (x a) (if (p x) (cons x a) a))`	`f(l) = (f1 l (λx. > x 2))`	`[7 2 3]→[7 3]`
		 `[1 2 3 4]→[3 4]`
		 `[4 3 2 1]→[4 3]`

Wake — as in Exploration/Compression Algorithm



DSL

```
f0(l,r) = foldr r l cons
f1(l,p) = foldr r nil
          (λ (x a)
            (if (p x)
              (cons x a)
              a))
```

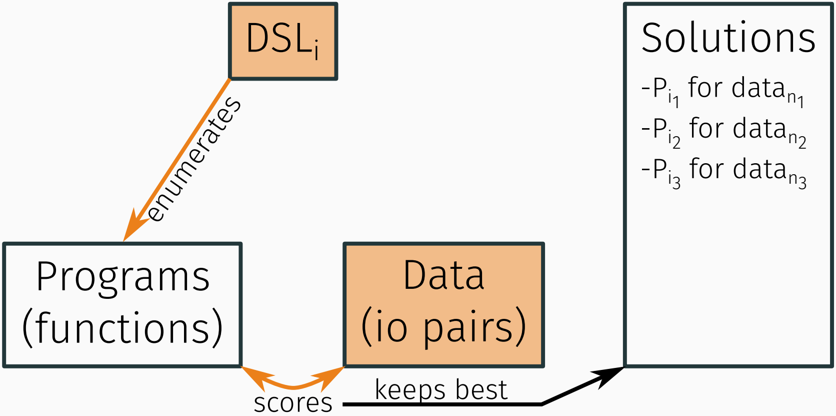
Programs

```
f(l) = (f0 l l)

f(l) = (f1 l (λx. > x 2))
```

Data

```
[7 2]→[7 2 7 2]
["a"]→["a" "a"]

[7 2 3]→[7 3]
[1 2 3 4]→[3 4]
[4 3 2 1]→[4 3]
```

Wake — as in Exploration/Compression Algorithm



DSL

```
f0(l,r) = foldr r l cons
f1(l,p) = foldr r nil
          (λ (x a)
            (if (p x)
              (cons x a)
              a))
```

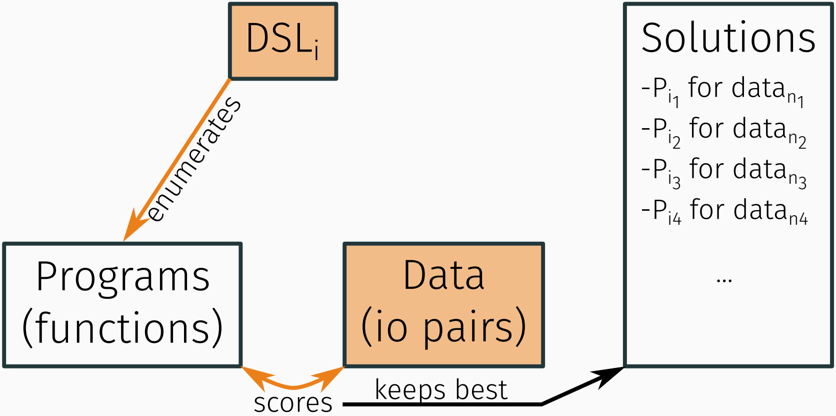
Programs

```
f(l) = (f0 l l)

f(l) = (f1 l (λx. > x 2))
```

Data

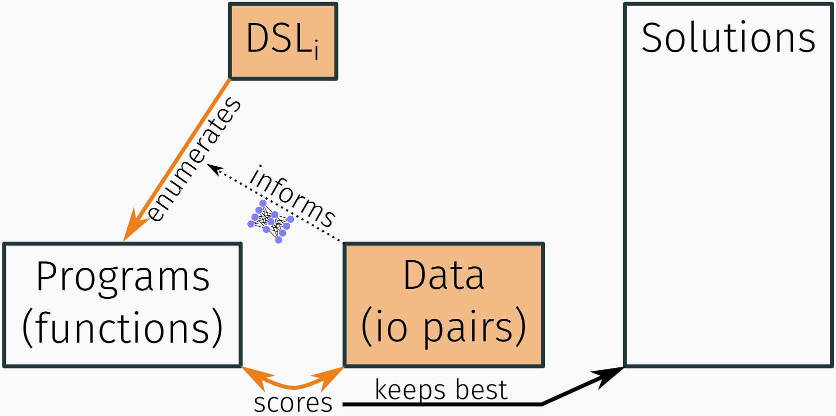
```
[7 2]→[7 2 7 2]
["a"]→["a" "a"]

[7 2 3]→[7 3]
[1 2 3 4]→[3 4]
[4 3 2 1]→[4 3]
```

Wake — as in Exploration/Compression Algorithm



DSL	Programs	Data
`f0(l,r) = foldr r l cons`	`f(l) = (f0 l l)`	`[7 2]→[7 2 7 2]` `["a"]→["a" "a"]`

`f1(l,p) = foldr r nil (λ (x a) (if (p x) (cons x a) a))`	`f(l) = (f1 l (λx. > x 2))`	`[7 2 3]→[7 3]`
		`[1 2 3 4]→[3 4]`
		`[4 3 2 1]→[4 3]`



DSL

```
f0(l,r) = foldr r l cons
f1(l,p) = foldr r nil
           (λ (x a)
             (if (p x)
               (cons x a)
               a))
```

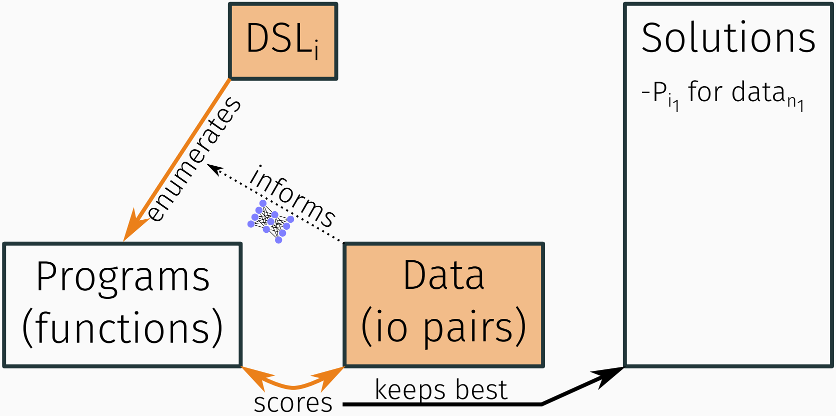
Programs

```
f(l) = (f0 l l)

f(l) = (f1 l (λx. > x 2))
```

Data

```
[7 2]→[7 2 7 2]
["a"]→["a" "a"]

[7 2 3]→[7 3]
[1 2 3 4]→[3 4]
[4 3 2 1]→[4 3]
```

DreamCoder — Wake



DSL

```
f₀(l,r) = foldr r l cons
f₁(l,p) = foldr r nil
          (λ (x a)
            (if (p x)
              (cons x a)
              a))
```

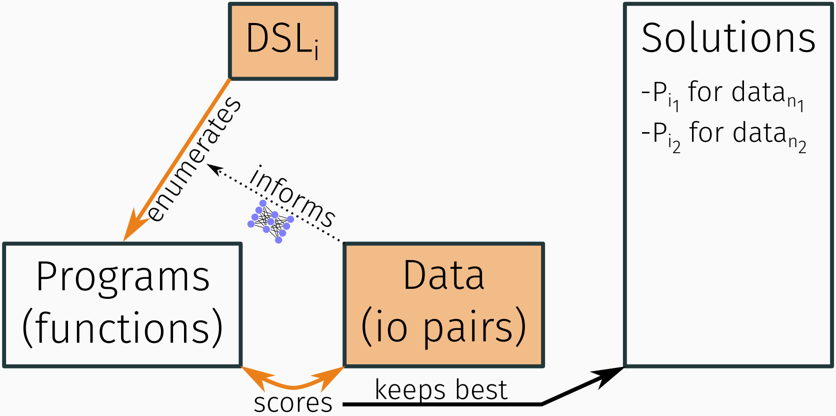
Programs

```
f(l) = (f₀ l l)

f(l) = (f₁ l (λx. > x 2))
```

Data

```
[7 2]→[7 2 7 2]
["a"]→["a" "a"]

[7 2 3]→[7 3]
[1 2 3 4]→[3 4]
[4 3 2 1]→[4 3]
```

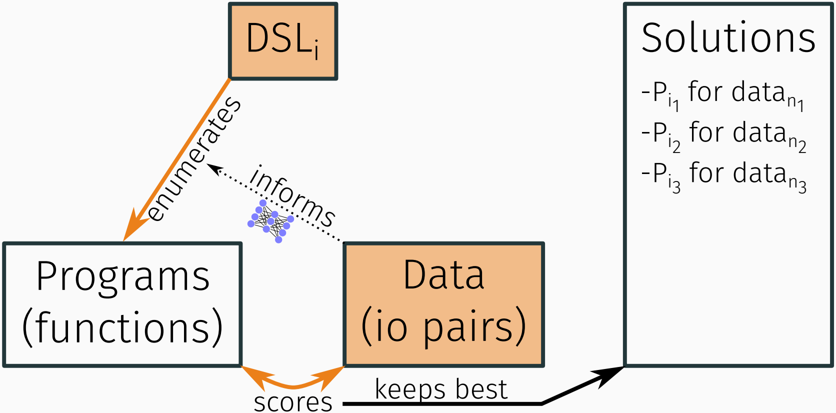
DreamCoder — Wake



DSL	Programs	Data
`f0(l,r) = foldr r l cons`	`f(l) = (f0 l l)`	`[7 2]→[7 2 7 2]` `["a"]→["a" "a"]`

`f1(l,p) = foldr r nil (λ (x a) (if (p x) (cons x a) a))`	`f(l) = (f1 l (λx. > x 2))`	`[7 2 3]→[7 3]`
		`[1 2 3 4]→[3 4]`
		`[4 3 2 1]→[4 3]`

DreamCoder — Wake



DSL

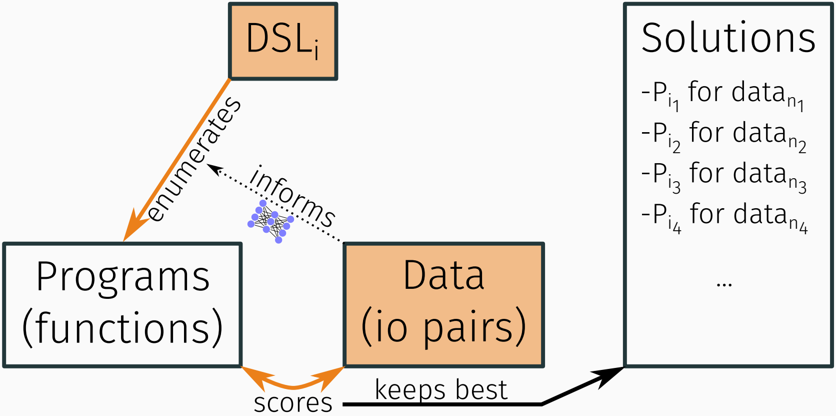
```
f0(l,r) = foldr r l cons
f1(l,p) = foldr r nil
          (λ (x a)
            (if (p x)
              (cons x a)
              a))
```

Programs

```
f(l) = (f0 l l)

f(l) = (f1 l (λx. > x 2))
```

Data

```
[7 2]→[7 2 7 2]
["a"]→["a" "a"]

[7 2 3]→[7 3]
[1 2 3 4]→[3 4]
[4 3 2 1]→[4 3]
```



DSL

```
f0(l,r) = foldr r l cons
f1(l,p) = foldr r nil
          (λ (x a)
             (if (p x)
                 (cons x a)
                 a))
```

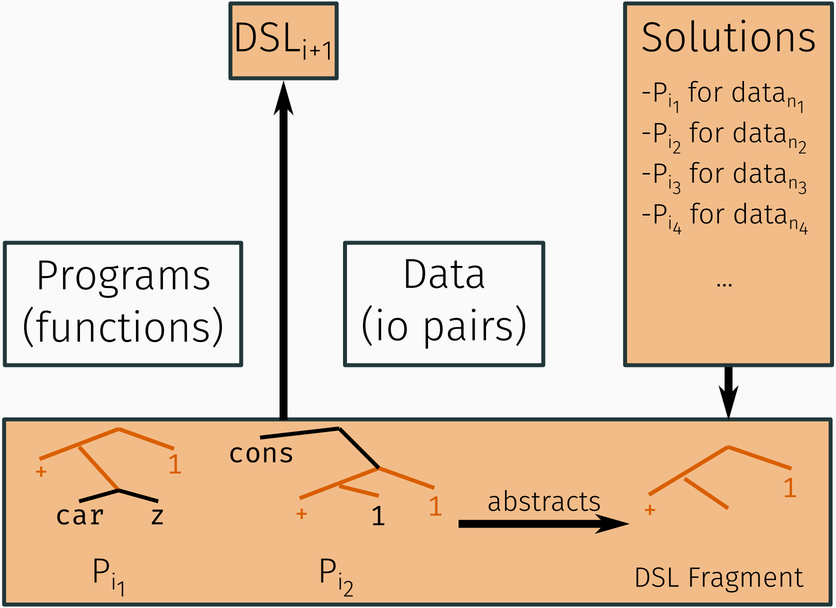
Programs

```
f(l) = (f0 l l)

f(l) = (f1 l (λx. > x 2))
```

Data

```
[7 2]→[7 2 7 2]
["a"]→["a" "a"]

[7 2 3]→[7 3]
[1 2 3 4]→[3 4]
[4 3 2 1]→[4 3]
```

DreamCoder — Sleep-G



DreamCoder — Sleep-G (Refactoring)

Learning higher-order `map` function

Task	Program

`(1 2 3)`$\rightarrow$`(2 4 6)`
`(1 9 2)`$\rightarrow$`(2 18 4)`
	`(Y (`λ` (r l) (if (nil? l) nil`
	`(cons (+ (car l) (car l))`
	`(r (cdr l))))))`

`(1 2 3)`$\rightarrow$`(2 3 4)`
`(1 9 2)`$\rightarrow$`(2 10 3)`
	`(Y (`λ` (r l) (if (nil? l) nil`
	`(cons (+ (car l) 1)`
	`(r (cdr l))))))`

DreamCoder — Sleep-G (Refactoring)

Learning higher-order `map` function

Task	Program
`(1 2 3)→(2 4 6)` `(1 9 2)→(2 18 4)`	`(Y (λ (r l) (if (nil? l) nil (cons (+ (car l) (car l)) (r (cdr l))))))`
`(1 2 3)→(2 3 4)` `(1 9 2)→(2 10 3)`	`(Y (λ (r l) (if (nil? l) nil (cons (+ (car l) 1) (r (cdr l))))))`

```
map = (λ (f) (Y (λ (r l) (if (nil?  l) nil
                         (cons (f (car l))
                         (r (cdr l)))))))
```

DreamCoder — Sleep-G (Refactoring)

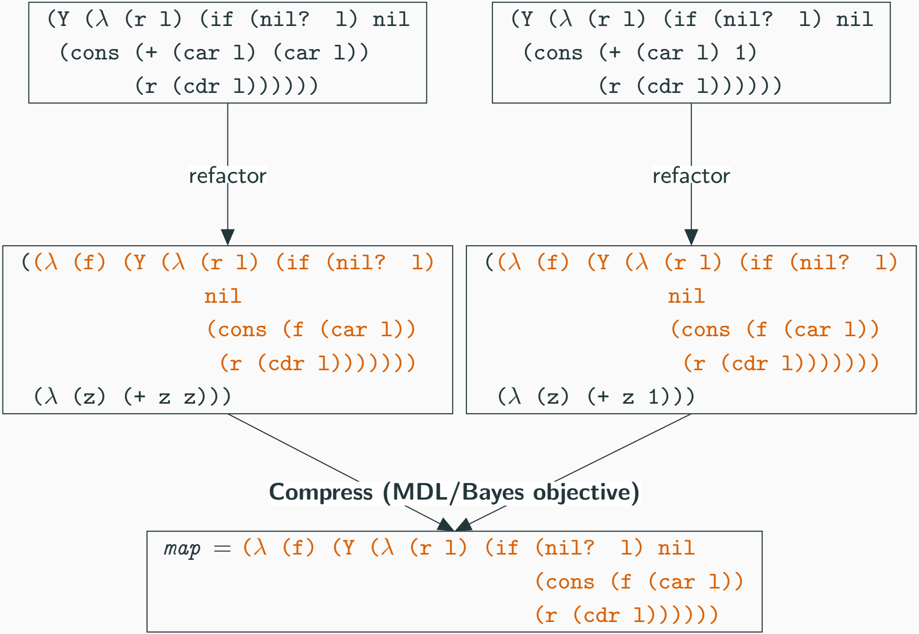
Learning higher-order `map` function

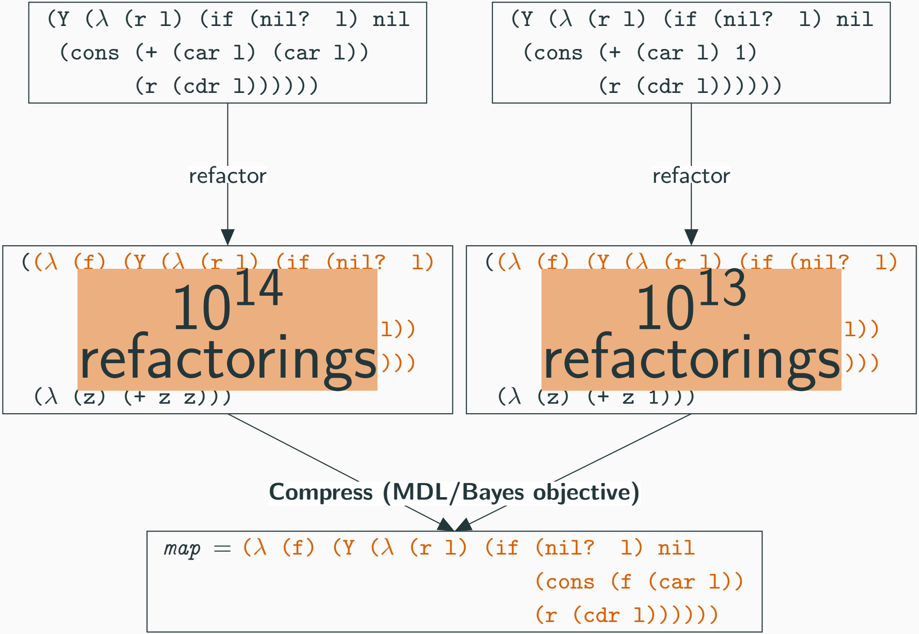
Task	Program

`(1 2)→(2 4)`
`(1 9)→(2 18)`
	`((λ (f) (Y (λ (r l) (if (nil? l) nil`
	`(cons (f (car l))`
	`(r (cdr l)))))))`
	`(λ (z) (+ z z)))`

`(1 2)→(2 3)`
`(1 9)→(2 10)`
	`((λ (f) (Y (λ (r l) (if (nil? l) nil`
	`(cons (f (car l))`
	`(r (cdr l)))))))`
	`(λ (z) (+ z 1)))`

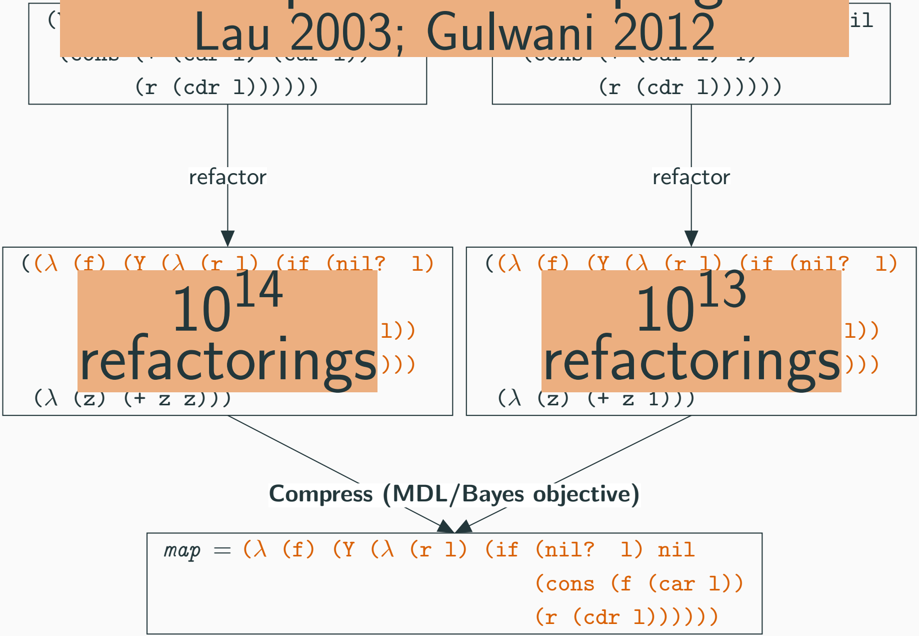
```
map = (λ (f) (Y (λ (r l) (if (nil? l) nil
                              (cons (f (car l))
                              (r (cdr l)))))))
```

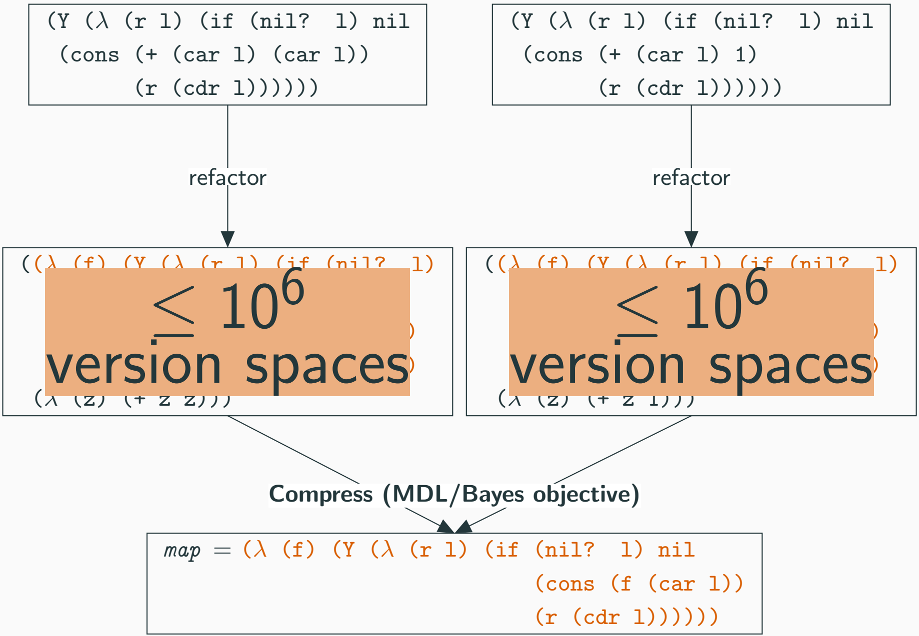

(Y (λ (r l) (if (nil? l) nil
(cons (+ (car l) (car l))
(r (cdr l))))))
(Y (λ (r l) (if (nil? l) nil
(cons (+ (car l) 1)
(r (cdr l))))))
refactor
refactor
((λ (f) (Y (λ (r l) (if (nil? l)
nil
(cons (f (car l))
(r (cdr l)))))))
(λ (z) (+ z z)))
((λ (f) (Y (λ (r l) (if (nil? l)
nil
(cons (f (car l))
(r (cdr l)))))))
(λ (z) (+ z 1)))
Compress (MDL/Bayes objective)
map = (λ (f) (Y (λ (r l) (if (nil? l) nil
(cons (f (car l))
(r (cdr l)))))))


(Y (λ (r l) (if (nil? l) nil
(cons (+ (car l) (car l))
(r (cdr l))))))
(Y (λ (r l) (if (nil? l) nil
(cons (+ (car l) 1)
(r (cdr l))))))
refactor
refactor
((λ (f) (Y (λ (r l) (if (nil? l)
l))
)))
(λ (z) (+ z z)))
10^{14} refactorings
((λ (f) (Y (λ (r l) (if (nil? l)
l))
)))
(λ (z) (+ z 1)))
10^{13} refactorings
Compress (MDL/Bayes objective)
map = (λ (f) (Y (λ (r l) (if (nil? l) nil
(cons (f (car l))
(r (cdr l))))))

version space: set of programs

Lau 2003; Gulwani 2012




(Y (λ (r l) (if (nil? l) nil
(cons (+ (car l) (car l))
(r (cdr l))))))
(Y (λ (r l) (if (nil? l) nil
(cons (+ (car l) 1)
(r (cdr l))))))
refactor
refactor
≤ 10^6
version spaces
≤ 10^6
version spaces
Compress (MDL/Bayes objective)
map = (λ (f) (Y (λ (r l) (if (nil? l) nil
(cons (f (car l))
(r (cdr l))))))

DreamCoder — Sleep-R (Experience Replay)

DSL_{i+1}

Programs
(functions)

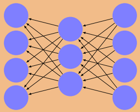
Data
(io pairs)

Solutions

-P_{i_1} for data_{n_1}
-P_{i_2} for data_{n_2}
-P_{i_3} for data_{n_3}
-P_{i_4} for data_{n_4}

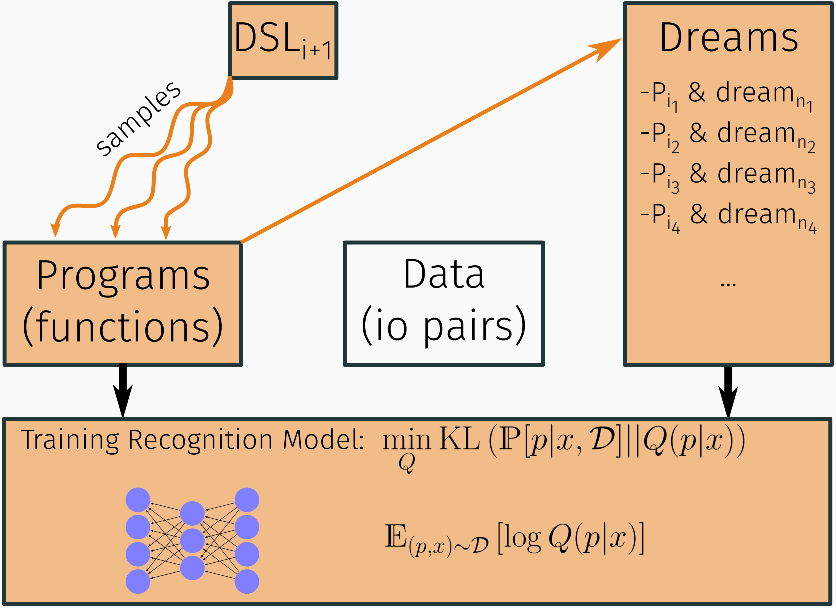
...

Training Recognition Model: $\min_{Q} \mathrm{KL}\left(\mathbb{P}[p|x,\mathcal{D}]||Q(p|x)\right)$



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

DreamCoder — Sleep-R (Dreaming)



List functions — Created & investigated by Lucas Morales

Name	Input	Output
repeat-3	[7 0]	[7 0 7 0 7 0]
drop-3	[0 3 8 6 4]	[6 4]
rotate-2	[8 14 1 9]	[1 9 8 14]
count-head-in-tail	[1 2 1 1 3]	2
keep-div-5	[5 9 14 6 3 0]	[5 0]
product	[7 1 6 2]	84

Discovers 38 concepts, including ‘filter’. With different tasks will also learn ‘map’, ‘fold’, ‘unfold’, etc. starting with 1950’s Lisp

Text editing

In the style of FlashFill (Gulwani 2012)

Text Editing

+106 769-438→106.769.438
+83 973-831→83.973.831

```
f(s) = (f0 "." "-"
         (f0 "." " "
           (cdr s)))
```

Temple Anna H →TAH
Lara Gregori→LG

```
f(s) = (f2 s)
```

```
f0(s,a,b) = (map (λ (x)
               (if (= x a) b x)) s)
```

(f_0: *Performs character substitution*)

```
f1(s,c) = (foldr s s (λ (x a)
               (cdr (if (= c x) s a))))
```

(f_1: *Drop characters from* s *until* c *reached*)

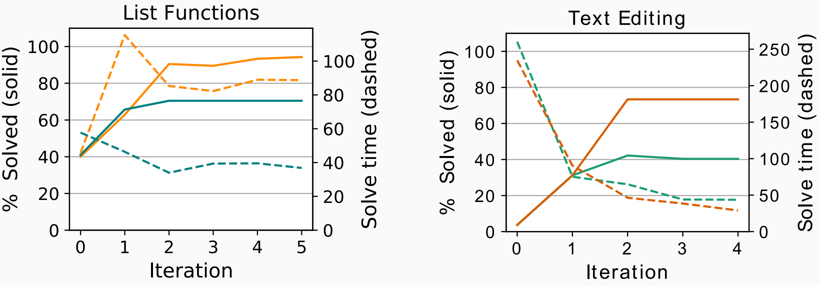
```
f2(s) = (unfold s is-nil car
            (λ (z) (f1 z " ")))
```

(f_2: *Abbreviates a sequence of words*)

SyGuS problems: solves 3% before learning, vs 75% after learning.

Best prior work: 80%

List functions & Text editing: Learning curves on hold out tasks



Learning curves for DreamCoder both with (in orange) and without (in teal) the recognition model. Solid lines: % holdout testing tasks solved w/ 10m timeout. Dashed lines: Average solve time, averaged only over tasks that are solved.

Symbolic regression from visual input

Symbolic Regression

$f(x) = (f_1 \; x)$

$f(x) = (f_6 \; x)$

$f(x) = (f_4 \; x)$

$f(x) = (f_3 \; x)$

$f_0(x) = (+ \; x \; \text{real})$

$f_1(x) = (f_0 \; (* \; \text{real} \; x))$

$f_2(x) = (f_1 \; (* \; x \; (f_0 \; x)))$

$f_3(x) = (f_0 \; (* \; x \; (f_2 \; x)))$

$f_4(x) = (f_0 \; (* \; x \; (f_3 \; x)))$

(f_4: *4th order polynomial*)

$f_5(x) = (/ \; \text{real} \; x)$

$f_6(x) = (f_5 \; (f_0 \; x))$

(f_6: *rational function*)

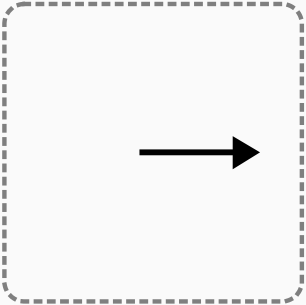
DSL

```
OP ::= FW x | RT x | UP | DOWN | SET state
```

Tasks

```
task :  image
```

```
FW 1
```



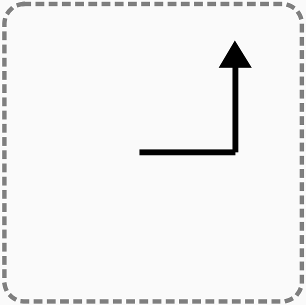
DSL

`OP ::= FW x | RT x | UP | DOWN | SET state`

Tasks

`task : image`

```
FW 1
RT π/2
FW 1
```



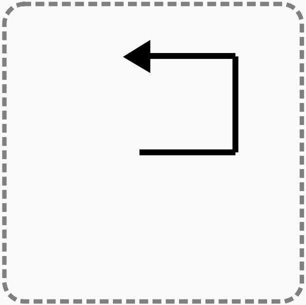
DSL

`OP ::= FW x | RT x | UP | DOWN | SET state`

Tasks

`task : image`

```
FW 1
RT π/2
FW 1
RT π/2
FW 1
```



Turtle graphics — Created & investigated by Mathias Sablé–Meyer

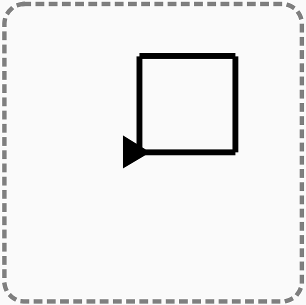
DSL

`OP ::= FW x | RT x | UP | DOWN | SET state`

Tasks

`task :  image`

```
for i in range(4)
> FW 1
> RT π/2
```



Turtle graphics — Created & investigated by Mathias Sablé–Meyer

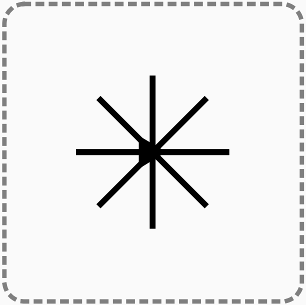
DSL

`OP ::= FW x | RT x | UP | DOWN | SET state`

Tasks

`task :  image`

```
for i in range(8)
> FW 1
> SET origin
> RT 2π/8
```



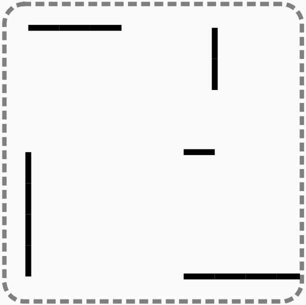
DSL

`OP ::= FW x | RT x | UP | DOWN | SET state`

Tasks

`task : image`

```
for i in range(8)
> PU
> FW i/2
> PD
> FW i/2
> RT π/2
```



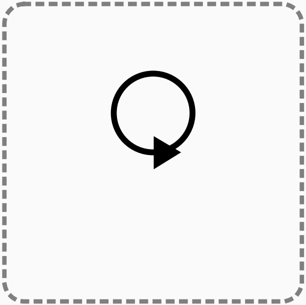
DSL

`OP ::= FW x | RT x | UP | DOWN | SET state`

Tasks

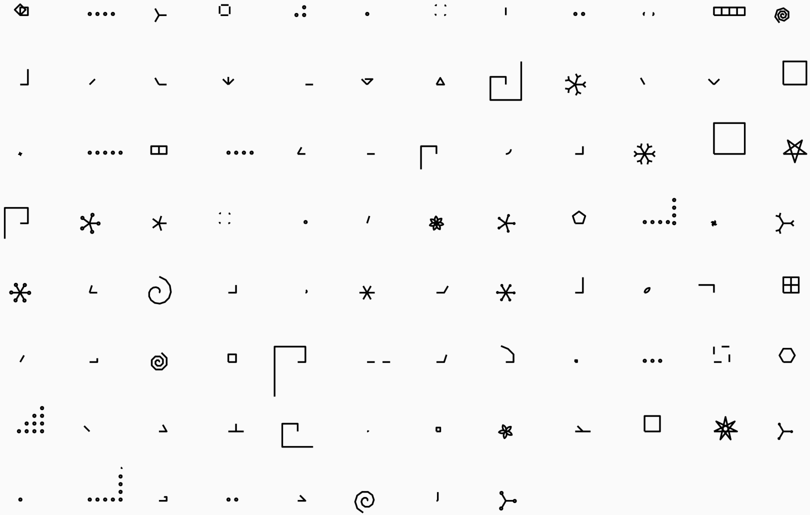
`task :  image`

```
for i in range(∞)
> FW ε
> RT ε
```

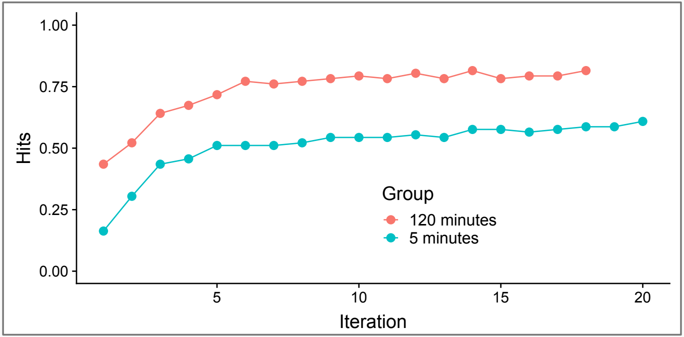


NUM ::= 1 | π | ∞ | ε | + | - | * | /

Turtle graphics — Training tasks



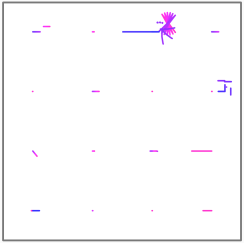
Turtle graphics — Learning curves

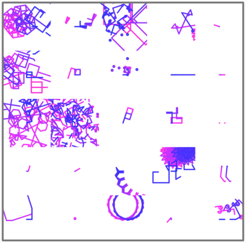


- $\frac{\pi}{2}$ and $\frac{\pi}{4}$ from π, 2, $+$ and $/$
- A line of length `n` followed with a right angle
- Loops of length `n` that uses the number `n` inside.
- Unit line then teleport back to origin
- ...

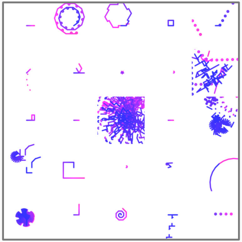
Turtle graphics — Dreams

Before training



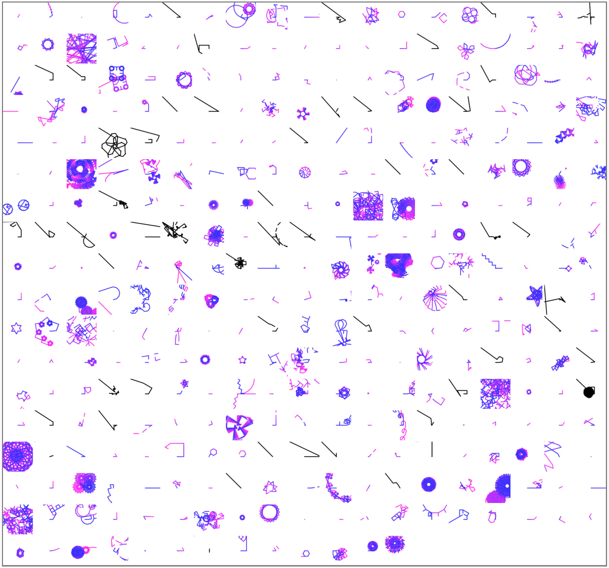


Plateau 5 minutes

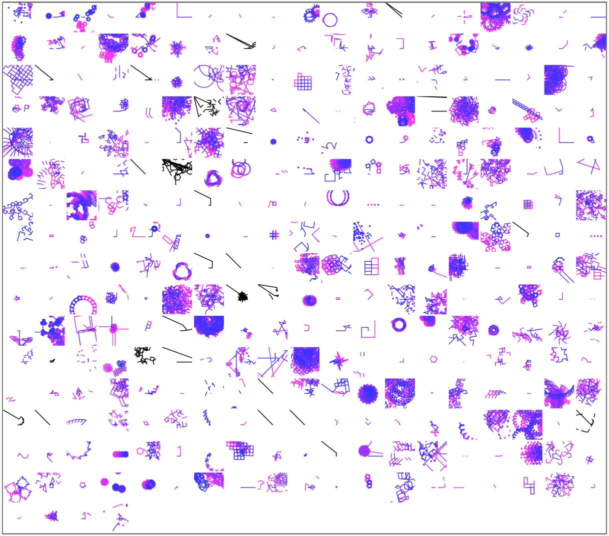


Plateau 2 hours

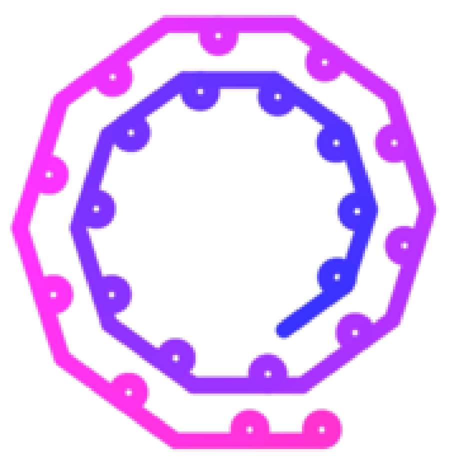
Turtle graphics — More dreams, 5 minutes, 1st iteration



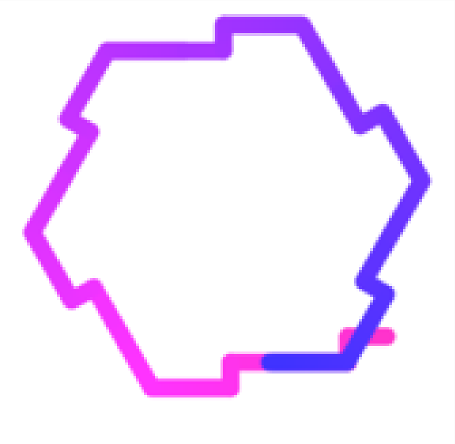
Turtle graphics — More dreams, 5 minutes, last iteration



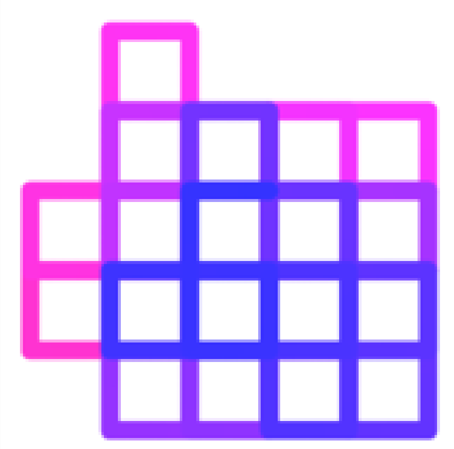
Turtle graphics — Dreaming from learned generative model



Turtle graphics — Dreaming from learned generative model

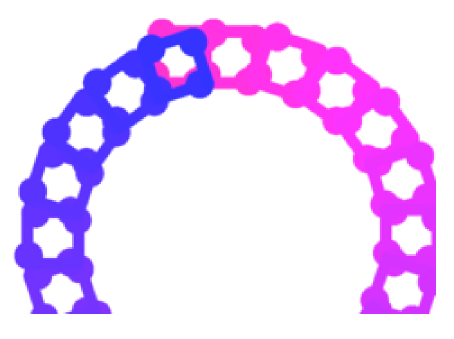


Turtle graphics — Dreaming from learned generative model

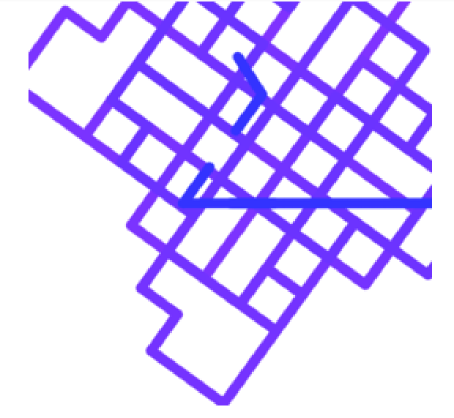


Turtle graphics — Dreaming from learned generative model

Turtle graphics — Dreaming from learned generative model



Turtle graphics — Dreaming from learned generative model

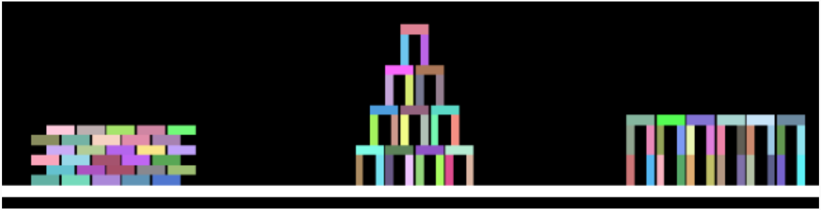


Turtle graphics — Dreaming from learned generative model

Turtle graphics — Dreaming from learned generative model

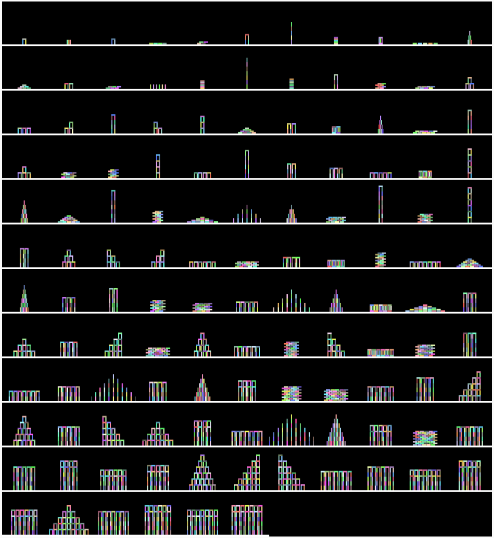
Tower building in blocks world

Control a hand that puts down blocks
(turtle: control a pen that puts down ink)



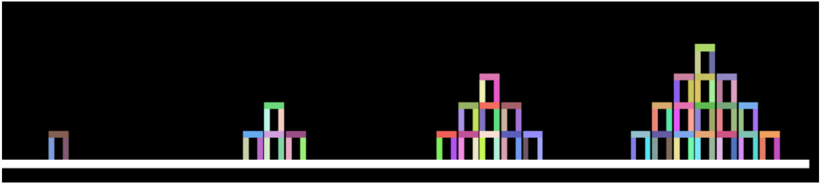
Tower building in blocks world

Control a hand that puts down blocks
(turtle: control a pen that puts down ink)



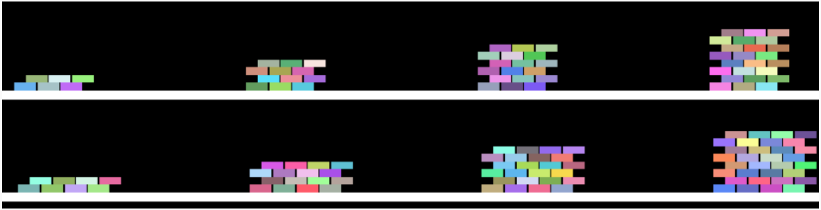
Tower building in blocks world: Learned concepts

Parametric planning primitives. Example pyramid concept:



Tower building in blocks world: Learned concepts

Parametric planning primitives. Example brickwall concept:



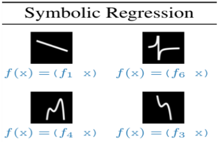
Vision

More human-like machine intelligence

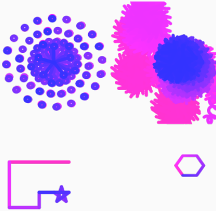
- Acquiring a domain-specific representation (DSL)
- Learning to write programs (recognition model)

DreamCoder: an algorithm for jointly realizing these goals

```
f2(p,f,n,x) = (if (p x) nil
                  (cons (f x) (f2 (n x))))
  (f2: unfold)
f3(i,l) = (if (= i 0) (car l)
                  (f3 (f1 i) (cdr l))))
  (f3: index)
f4(f,l,x) = (if (empty? l) x
                  (f (car l) (f4 (cdr l))))
  (f4: fold)
f5(f,l) = (if (empty? l) nil)
                  (cons (f (car l)) (f5 (cdr l)))
  (f5: map)
```




```
f0(x) = (+ x real)
f1(x) = (f0 (* real x))
f2(x) = (f1 (* x (f0 x)))
f3(x) = (f0 (* x (f2 x)))
f4(x) = (f0 (* x (f3 x)))
  (f4: 4th order polynomial)
f5(x) = (/ real x)
f6(x) = (f5 (f0 x))
  (f6: rational function)
```

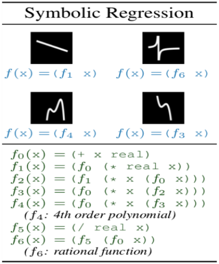


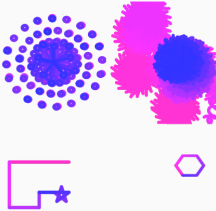
More human-like machine intelligence

- Acquiring a domain-specific representation (DSL)
- Learning to write programs (recognition model)

DreamCoder: an algorithm for jointly realizing these goals

```
f2(p,f,n,x) = (if (p x) nil
                  (cons (f x) (f2 (n x))))
  (f2: unfold)
f3(i,l) = (if (= i 0) (car l)
              (f3 (f1 i) (cdr l))))
  (f3: index)
f4(f,l,x) = (if (empty? l) x
                (f (car l) (f4 (cdr l))))
  (f4: fold)
f5(f,l) = (if (empty? l) nil
              (cons (f (car l)) (f5 (cdr l)))
  (f5: map)
```





The End.