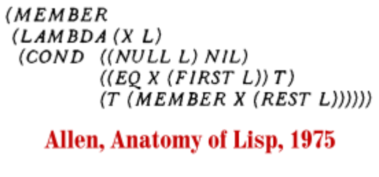
`DreamCoder`: Growing libraries of concepts with wake-sleep program induction

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

October 2018

MIT

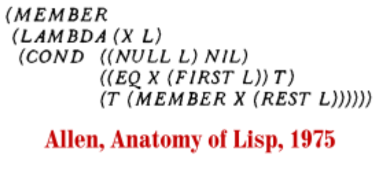
Human program induction everywhere



```
(MEMBER
 (LAMBDA (X L)
  (COND  ((NULL L) NIL)
         ((EQ X (FIRST L)) T)
         (T (MEMBER X (REST L))))))
```

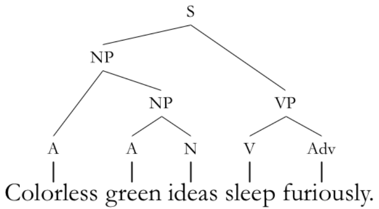
Allen, Anatomy of Lisp, 1975

Human program induction everywhere

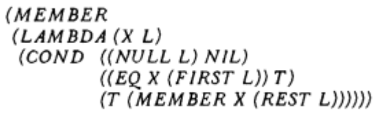


```
(MEMBER
 (LAMBDA (X L)
  (COND  ((NULL L) NIL)
         ((EQ X (FIRST L)) T)
         (T (MEMBER X (REST L))))))
```

Allen, Anatomy of Lisp, 1975

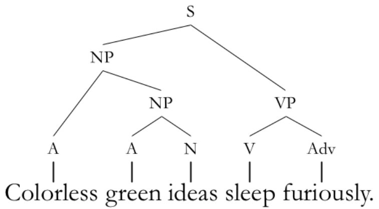


Human program induction everywhere

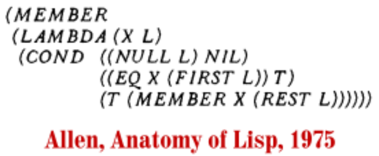


```
(MEMBER
 (LAMBDA (X L)
  (COND  ((NULL L) NIL)
         ((EQ X (FIRST L)) T)
         (T (MEMBER X (REST L))))))
```

Allen, Anatomy of Lisp, 1975

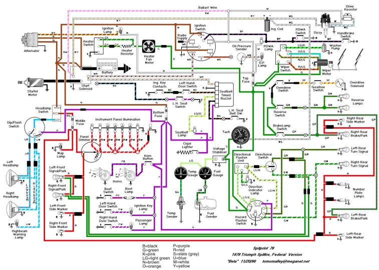


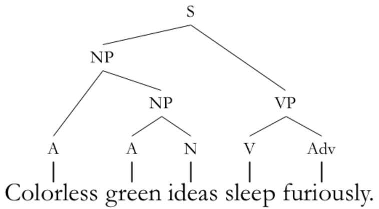
Human program induction everywhere



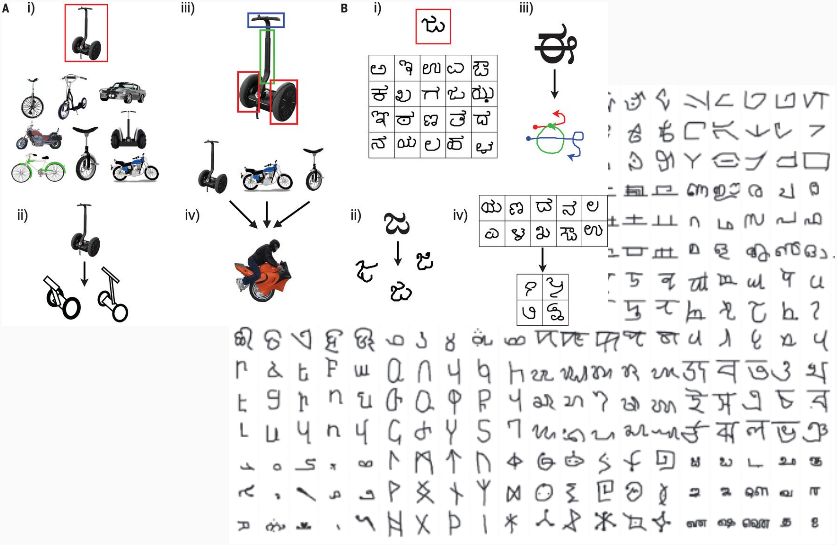
```
(MEMBER
 (LAMBDA (X L)
  (COND  ((NULL L) NIL)
         ((EQ X (FIRST L)) T)
         (T (MEMBER X (REST L))))))
```

Allen, Anatomy of Lisp, 1975





Human program induction everywhere



Lake et al 2015

Growing domain-specific knowledge

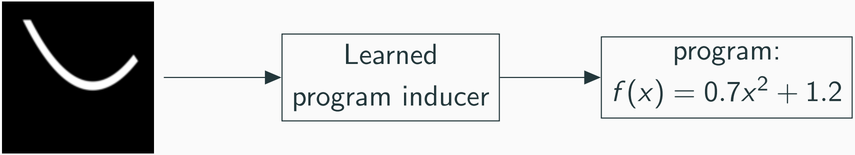
Goal: acquire domain-specific knowledge needed to induce a class of programs

- Library of concepts (declarative knowledge; generative model over programs)
- Inference strategy (procedural knowledge)

Growing domain-specific knowledge

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

- Library of concepts (declarative knowledge; generative model over programs)
- Inference strategy (procedural knowledge)



Learned program inducer

program: $f(x) = 0.7x^2 + 1.2$

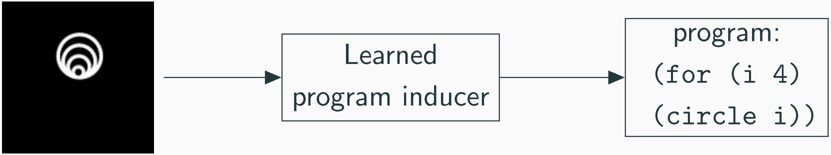
Concepts: x^2, etc

Inference strategy: neurosymbolic search for programs

Growing domain-specific knowledge

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

- Library of concepts (declarative knowledge; generative model over programs)
- Inference strategy (procedural knowledge)



Learned program inducer

program:

```
(for (i 4)
 (circle i))
```

Concepts: `circle`, etc

Inference strategy: neurosymbolic search for programs

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

DreamCoder

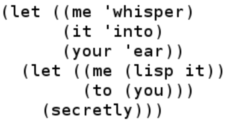
- **Wake:** Solve problems by writing programs
- **Sleep:** Improve DSL and neural recognition model:
  - **Sleep-G:** Improve DSL (**G**enerative model)
  - **Sleep-R:** Improve **R**ecognition model

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

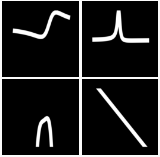
DreamCoder

- **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

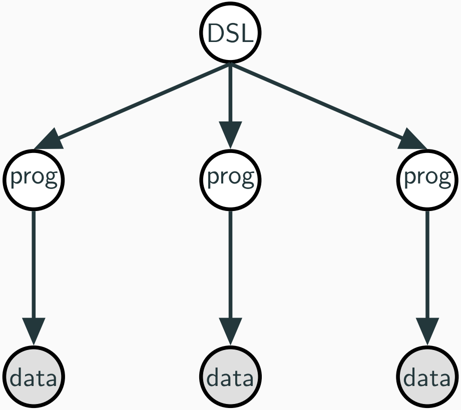
Combines ideas from Wake-Sleep & Exploration-Compression algorithm by Eyal Dechter



```
(let ((me 'whisper)
      (it 'into)
      (your 'ear))
  (let ((me (lisp it))
        (to (you)))
    (secretly)))
```



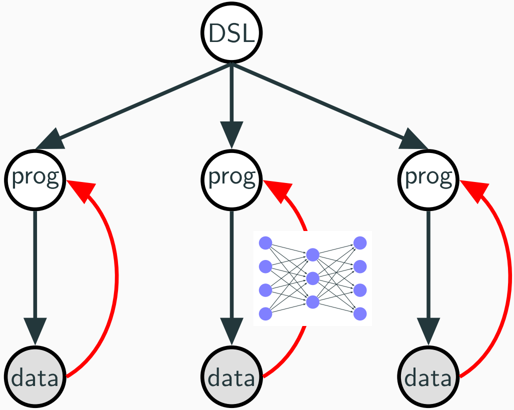
DSL learning as Bayesian inference



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

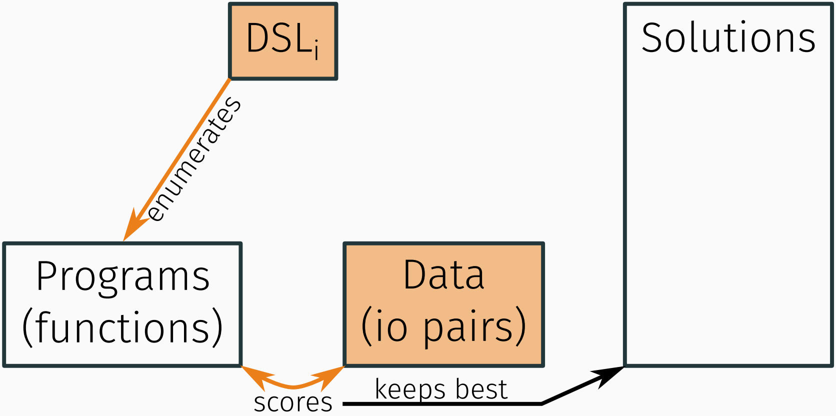
Dechter et al.: Exploration-Compression. Inspiration for DreamCoder.

DSL learning as amortized Bayesian inference



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

Wake — as in Exploration/Compression Algorithm



DSL$_i$

enumerates

Programs (functions)

Data (io pairs)

scores

keeps best

Solutions

DSL

```
f_0(l,r) = foldr r l cons
f_1(l,p) = foldr r nil
         (λ (x a)
            (if (p x)
               (cons x a)
               a))
```

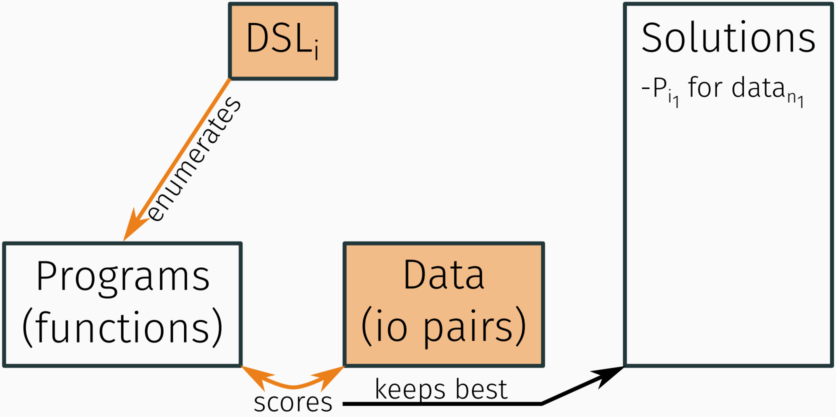
Programs

```
f(l) = (f_0 l l)


f(l) = (f_1 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_i

enumerates

Programs (functions)

Data (io pairs)

scores

keeps best

Solutions

-P_{i_1} for $data_{n_1}$

DSL

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

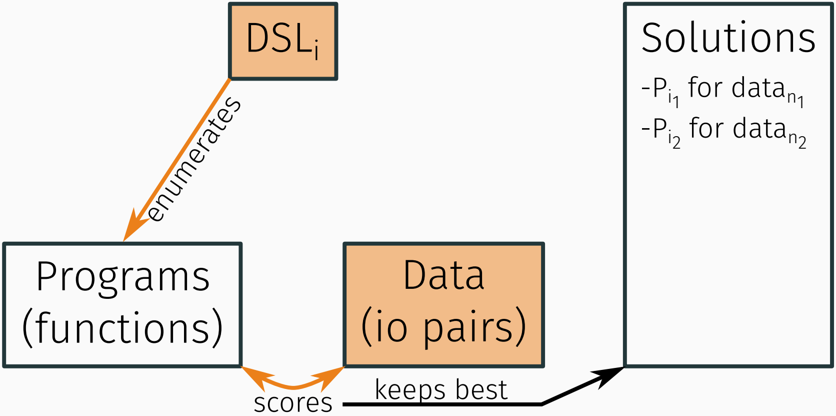
Programs

```
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$_i$

enumerates

Programs (functions)

Data (io pairs)

scores

keeps best

Solutions

-P$_{i_1}$ for data$_{n_1}$

-P$_{i_2}$ for data$_{n_2}$

DSL

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

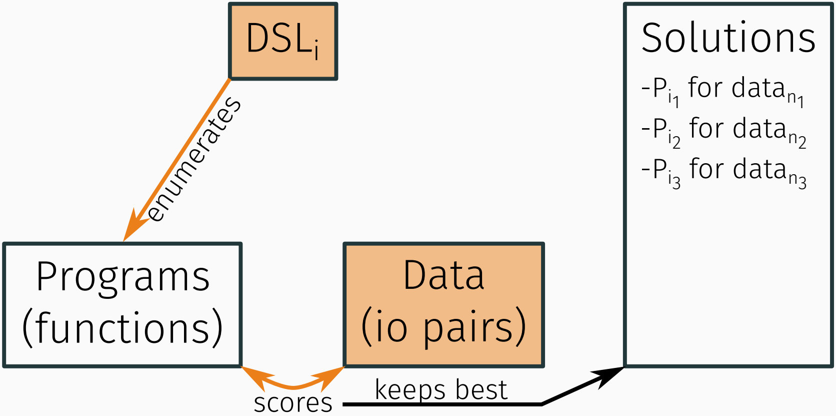
Programs

```
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_i

enumerates

Programs (functions)

Data (io pairs)

scores

keeps best

Solutions

-P_{i_1} for $data_{n_1}$

-P_{i_2} for $data_{n_2}$

-P_{i_3} for $data_{n_3}$

DSL

```
f_0(l,r) = foldr r l cons
f_1(l,p) = foldr r nil
          (λ (x a)
            (if (p x)
              (cons x a)
              a))
```

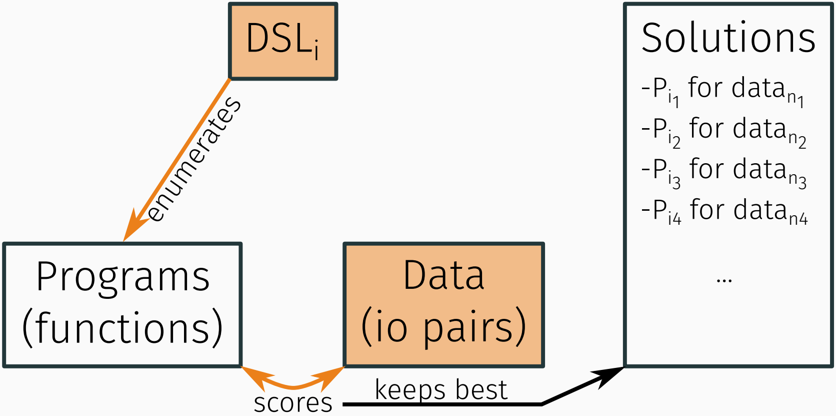
Programs

```
f(l) = (f_0 l l)

f(l) = (f_1 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_i

enumerates

Programs (functions)

Data (io pairs)

scores

keeps best

Solutions

-P_{i_1} for $data_{n_1}$

-P_{i_2} for $data_{n_2}$

-P_{i_3} for $data_{n_3}$

-P_{i4} for $data_{n4}$

...

DSL

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

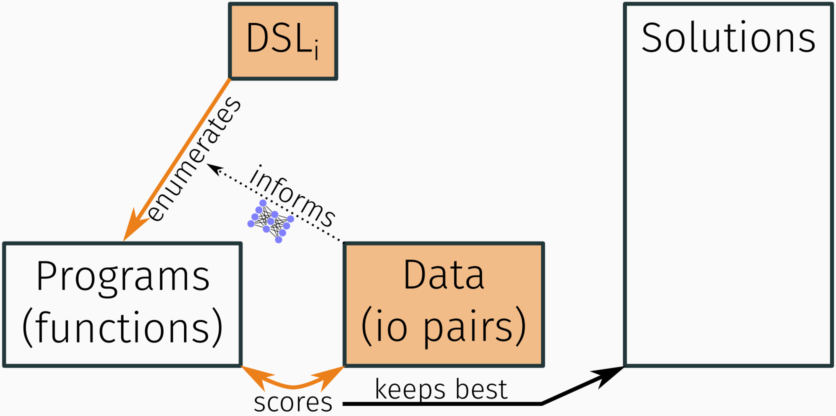
Programs

```
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

```
f_0(l,r) = foldr r l cons
f_1(l,p) = foldr r nil
             (λ (x a)
               (if (p x)
                 (cons x a)
                 a))
```

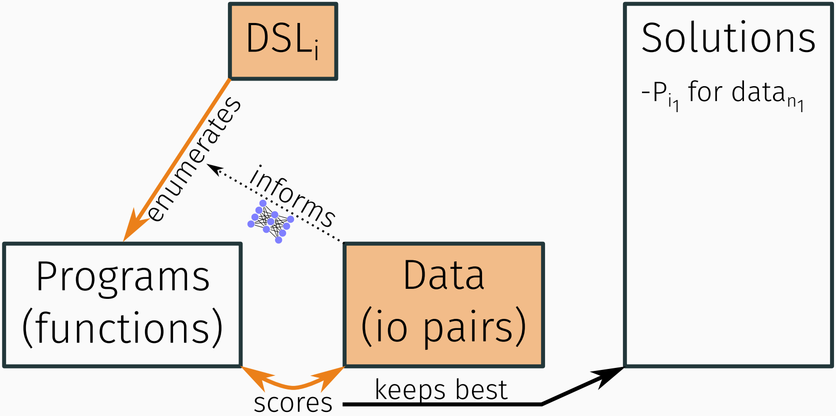
Programs

```
f(l) = (f_0 l l)

f(l) = (f_1 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_i

enumerates

informs

Programs (functions)

Data (io pairs)

scores

keeps best

Solutions

-P_{i_1} for data_{n_1}

DSL

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

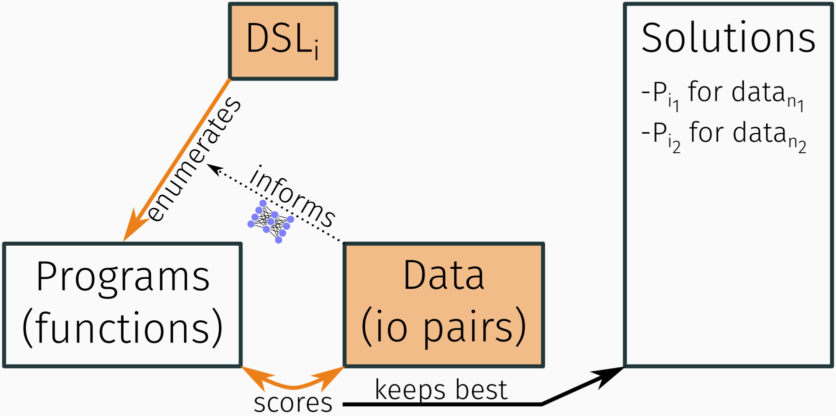
Programs

```
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_i

enumerates

informs

Programs (functions)

Data (io pairs)

scores

keeps best

Solutions

-P_{i_1} for data_{n_1}

-P_{i_2} for data_{n_2}

DSL

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

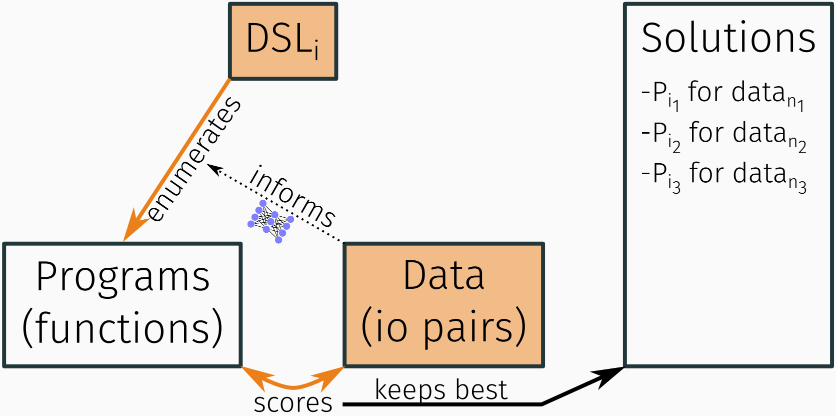
Programs

```
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_i

enumerates

informs

Programs (functions)

Data (io pairs)

scores

keeps best

Solutions

-P_{i_1} for data_{n_1}

-P_{i_2} for data_{n_2}

-P_{i_3} for data_{n_3}

DSL

```
f_0(l,r) = foldr r l cons
f_1(l,p) = foldr r nil
             (λ (x a)
               (if (p x)
                 (cons x a)
                 a))
```

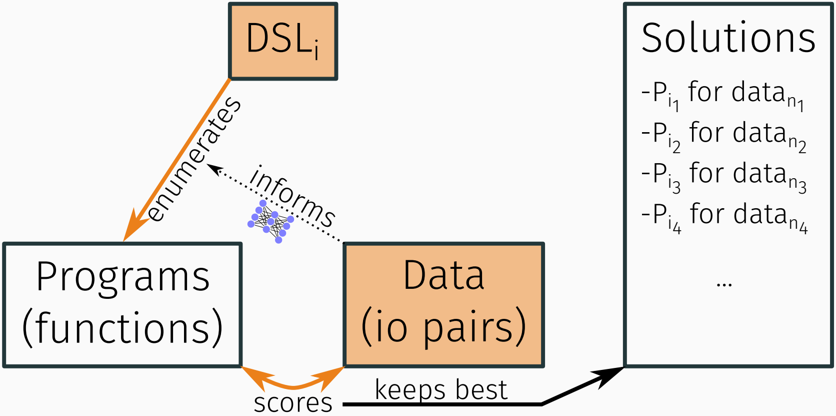
Programs

```
f(l) = (f_0 l l)

f(l) = (f_1 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_i

enumerates

informs

Programs (functions)

Data (io pairs)

scores

keeps best

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}

…

DSL

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

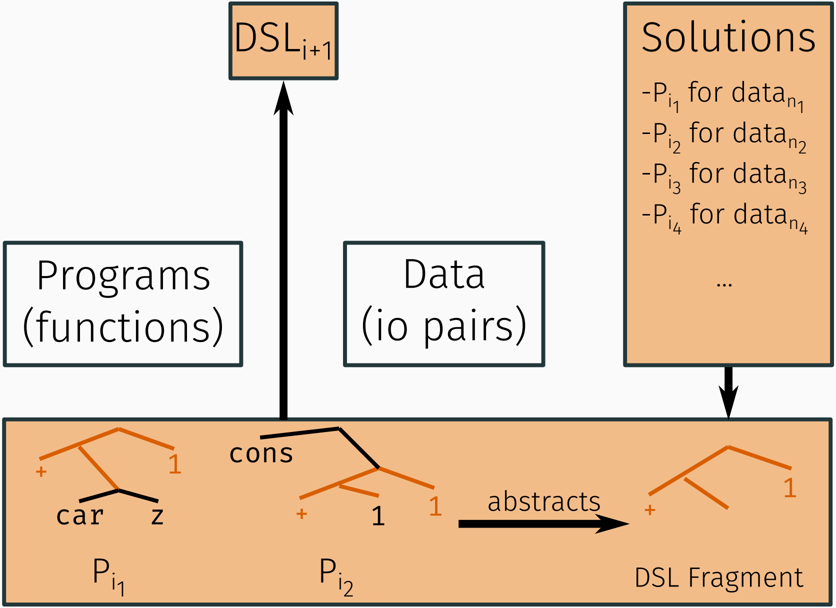
Programs

```
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



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$_{n_4}$

...

P_{i_1}

P_{i_2}

DSL Fragment

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))))))`

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)

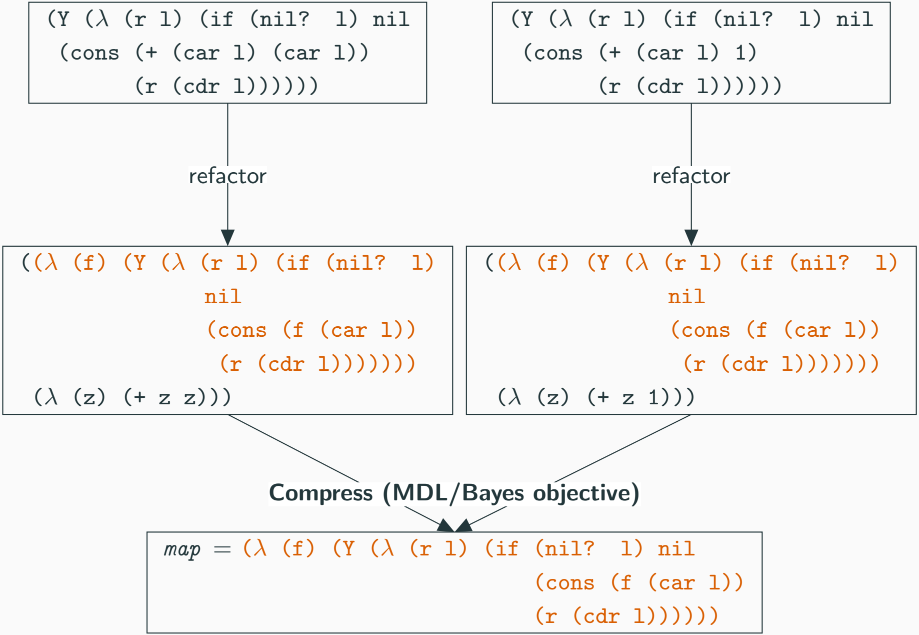
Learning higher-order `map` function

Task	Program

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

`(1 2 3)→(2 3 4)`
`(1 9 2)→(2 10 3)`
	`((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))))))
```



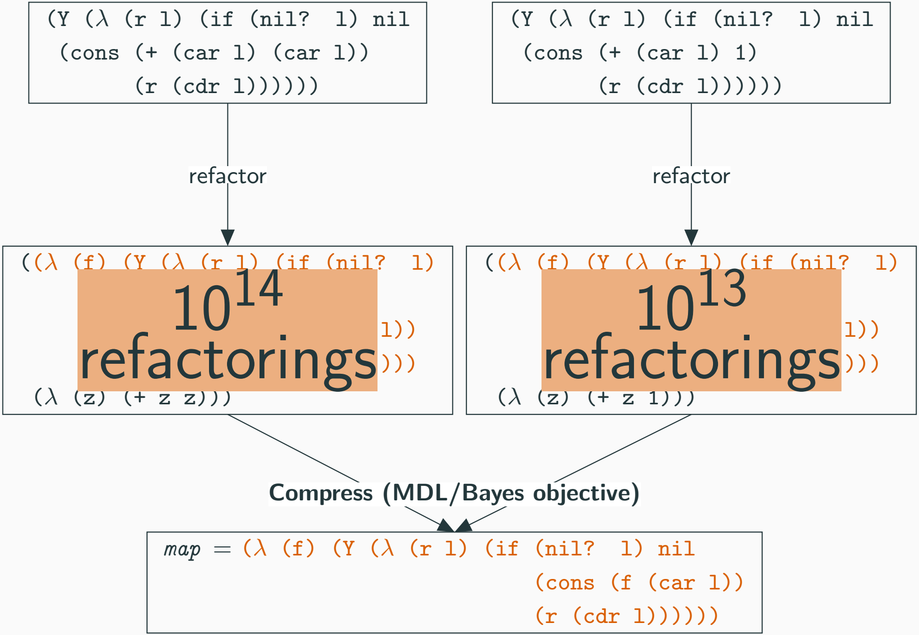
```
(Y (λ (r l) (if (nil?  l) nil
 (cons (+ (car l) (car l))
       (r (cdr l))))))
```

```
(Y (λ (r l) (if (nil?  l) nil
 (cons (+ (car l) 1)
       (r (cdr l))))))
```

```
((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)))
```

```
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))))))
```

```
((λ (f) (Y (λ (r l) (if (nil? l)
                                l))
                                )))
 (λ (z) (+ z z)))
```

10^{14} refactorings

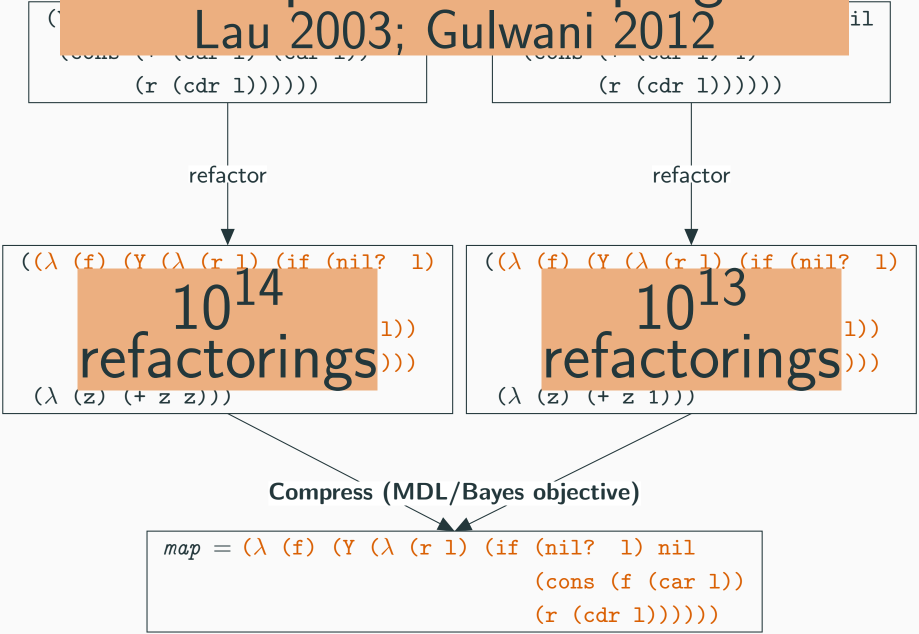
```
((λ (f) (Y (λ (r l) (if (nil? l)
                                l))
                                )))
 (λ (z) (+ z 1)))
```

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                                   il
 (cons (+ (car l) (car l))
       (r (cdr l))))))
```

```
 (cons (+ (car l) 1)
       (r (cdr l))))))
```

refactor

refactor

```
((λ (f) (Y (λ (r l) (if (nil?  l)
                                      l))
                                      )))
 (λ (z) (+ z z)))
```

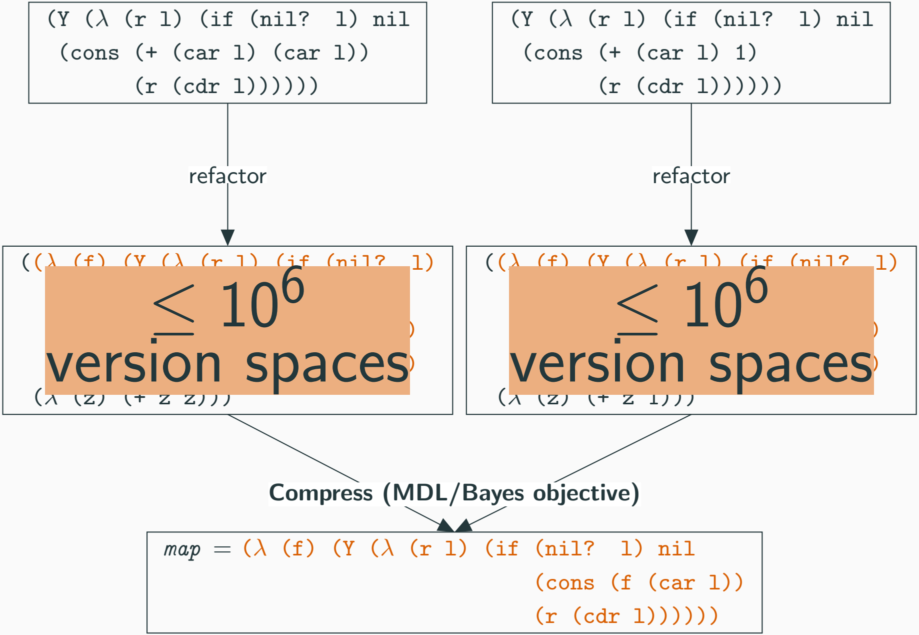
10^{14} refactorings

```
((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))))))
```



```
(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)
 (λ (z) (+ z z)))
```

$\leq 10^6$ version spaces

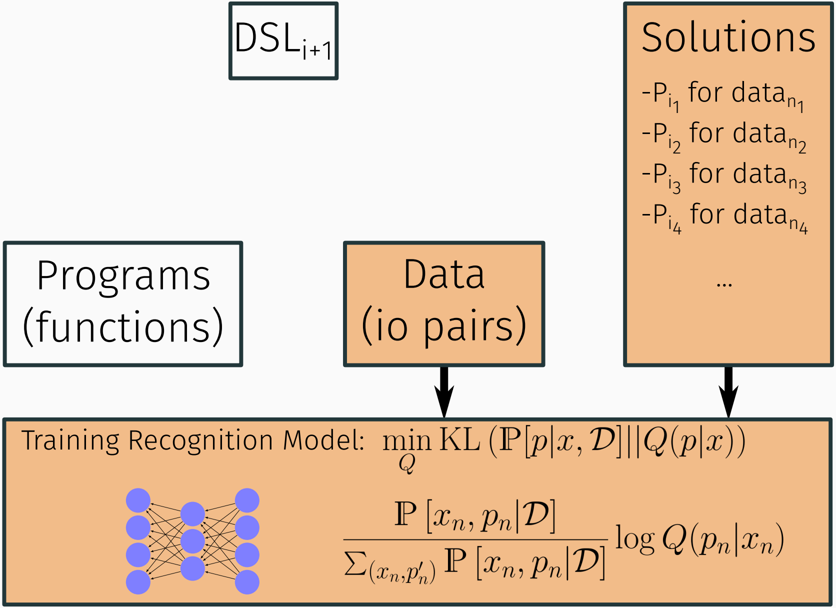
```
((λ (f) (Y (λ (r l) (if (nil? l)
 (λ (z) (+ z 1)))
```

$\leq 10^6$ 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}

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}

...

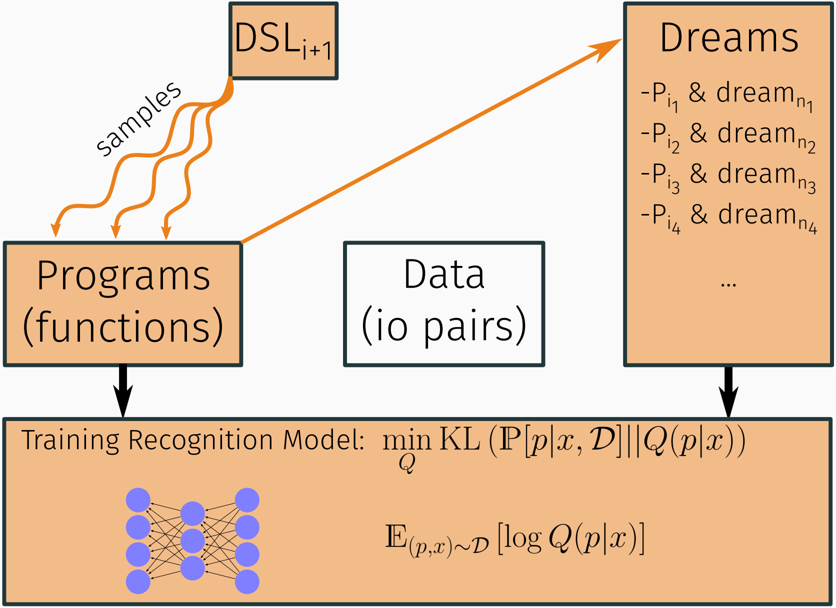
Programs (functions)

Data (io pairs)

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)



DSL$_{i+1}$

samples

Dreams

-P$_{i_1}$ & dream$_{n_1}$

-P$_{i_2}$ & dream$_{n_2}$

-P$_{i_3}$ & dream$_{n_3}$

-P$_{i_4}$ & dream$_{n_4}$

...

Programs (functions)

Data (io pairs)

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

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

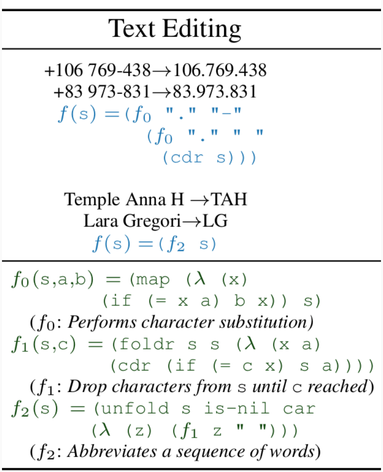
List functions — Created & investigated by Lucas Morales

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’.

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) =(`f_0` "." "-"`
`(`f_0` "." " "`
`(cdr s)))`

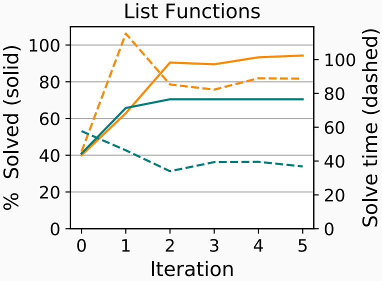
Temple Anna H →TAH
Lara Gregori→LG
f`(s) =(`f_2` s)`

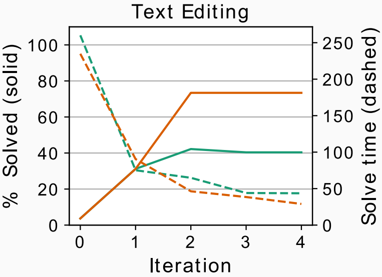
f_0`(s,a,b) = (map (`λ` (x)`
`(if (= x a) b x)) s)`
(f_0: *Performs character substitution*)
f_1`(s,c) = (foldr s s (`λ` (x a)`
`(cdr (if (= c x) s a))))`
(f_1: *Drop characters from* `s` *until* `c` *reached*)
f_2`(s) = (unfold s is-nil car`
`(`λ` (z) (`f_1` z " ")))`
(f_2: *Abbreviates a sequence of words*)

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

Best prior work: 80%

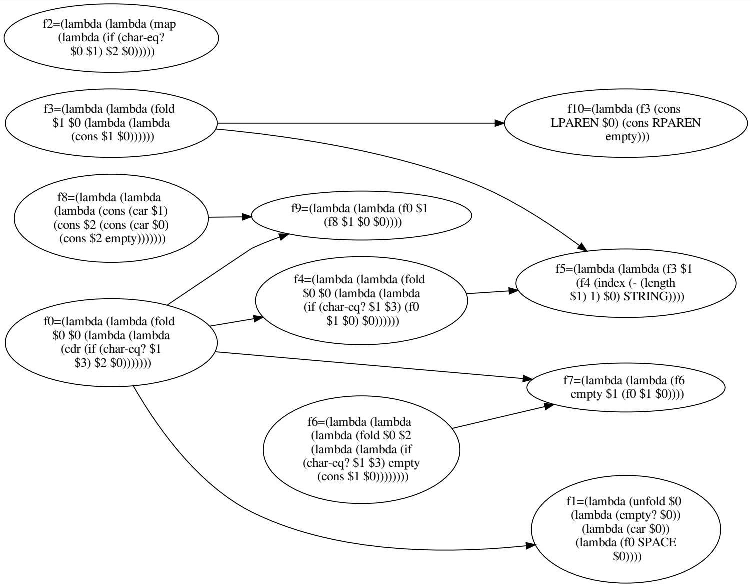
List functions & Text editing: Learning curves on hold out tasks





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.

Learned text processing DSL



f2=(lambda (lambda (map (lambda (if (char-eq? $0 $1) $2 $0)))))

f3=(lambda (lambda (fold $1 $0 (lambda (lambda (cons $1 $0))))))

f10=(lambda (f3 (cons LPAREN $0) (cons RPAREN empty)))

f8=(lambda (lambda (lambda (cons (car $1) (cons $2 (cons (car $0) (cons $2 empty)))))))

f9=(lambda (lambda (f0 $1 (f8 $1 $0 $0))))

f4=(lambda (lambda (fold $0 $0 (lambda (lambda (if (char-eq? $1 $3) (f0 $1 $0) $0))))))

f5=(lambda (lambda (f3 $1 (f4 (index (- (length $1) 1) $0) STRING))))

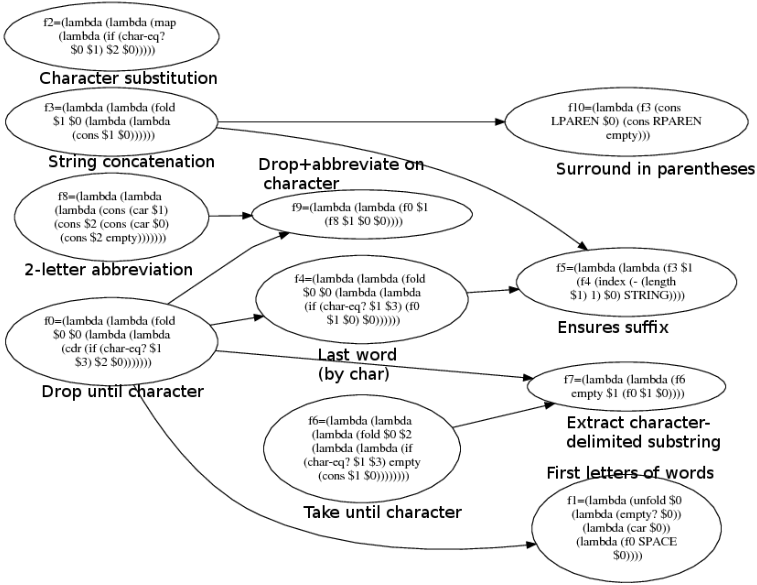
f0=(lambda (lambda (fold $0 $0 (lambda (lambda (cdr (if (char-eq? $1 $3) $2 $0)))))))

f7=(lambda (lambda (f6 empty $1 (f0 $1 $0))))

f6=(lambda (lambda (lambda (fold $0 $2 (lambda (lambda (if (char-eq? $1 $3) empty (cons $1 $0))))))))

f1=(lambda (unfold $0 (lambda (empty? $0)) (lambda (car $0)) (lambda (f0 SPACE $0))))

Learned text processing DSL



```
f2=(lambda (lambda (map (lambda (if (char-eq? $0 $1) $2 $0)))))
```

Character substitution

```
f3=(lambda (lambda (fold $1 $0 (lambda (lambda (cons $1 $0))))))
```

String concatenation

```
f10=(lambda (f3 (cons LPAREN $0) (cons RPAREN empty)))
```

Surround in parentheses

```
f8=(lambda (lambda (lambda (cons (car $1) (cons $2 (cons (car $0) (cons $2 empty)))))))
```

2-letter abbreviation

```
f9=(lambda (lambda (f0 $1 (f8 $1 $0 $0))))
```

Drop+abbreviate on character

```
f4=(lambda (lambda (fold $0 $0 (lambda (lambda (if (char-eq? $1 $3) (f0 $1 $0) $0))))))
```

Last word (by char)

```
f5=(lambda (lambda (f3 $1 (f4 (index (- (length $1) 1) $0) STRING))))
```

Ensures suffix

```
f0=(lambda (lambda (fold $0 $0 (lambda (lambda (cdr (if (char-eq? $1 $3) $2 $0)))))))
```

Drop until character

```
f7=(lambda (lambda (f6 empty $1 (f0 $1 $0))))
```

Extract character-delimited substring

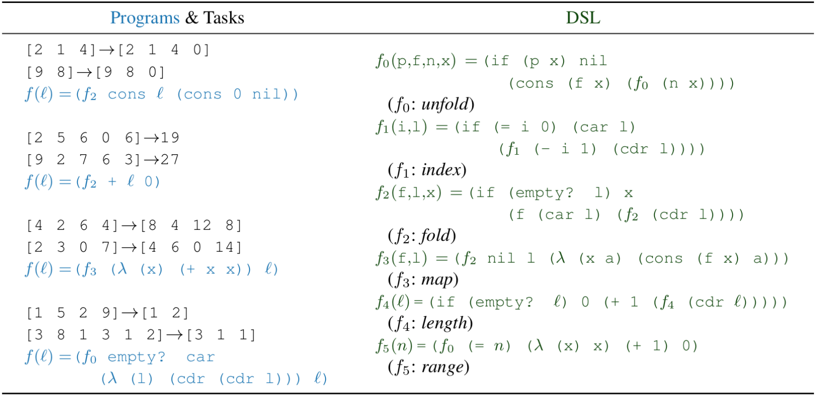
```
f6=(lambda (lambda (lambda (fold $0 $2 (lambda (lambda (if (char-eq? $1 $3) empty (cons $1 $0))))))))
```

Take until character

```
f1=(lambda (unfold $0 (lambda (empty? $0)) (lambda (car $0)) (lambda (f0 SPACE $0))))
```

First letters of words

Learning the fundamentals of programming

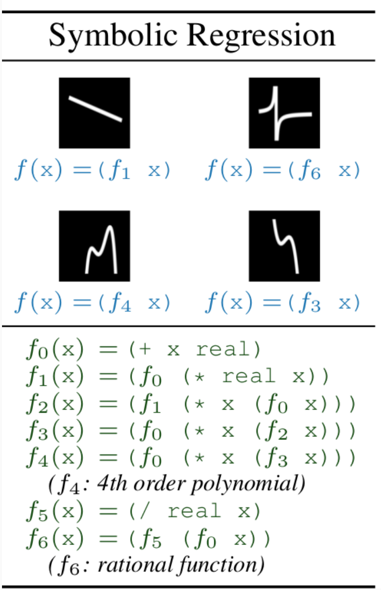


Programs & Tasks	DSL
[2 1 4]→[2 1 4 0] [9 8]→[9 8 0] $f(\ell) =$ (f_2 cons ℓ (cons 0 nil)) [2 5 6 0 6]→19 [9 2 7 6 3]→27 $f(\ell) =$ (f_2 + ℓ 0) [4 2 6 4]→[8 4 12 8] [2 3 0 7]→[4 6 0 14] $f(\ell) =$ (f_3 (λ (x) (+ x x)) ℓ) [1 5 2 9]→[1 2] [3 8 1 3 1 2]→[3 1 1] $f(\ell) =$ (f_0 empty? car (λ (l) (cdr (cdr l))) ℓ)	f_0(p,f,n,x) = (if (p x) nil (cons (f x) (f_0 (n x)))) (f_0: *unfold*) f_1(i,l) = (if (= i 0) (car l) (f_1 (- i 1) (cdr l)))) (f_1: *index*) f_2(f,l,x) = (if (empty? l) x (f (car l) (f_2 (cdr l)))) (f_2: *fold*) f_3(f,l) = (f_2 nil l (λ (x a) (cons (f x) a))) (f_3: *map*) $f_4(\ell)$ = (if (empty? ℓ) 0 (+ 1 (f_4 (cdr ℓ))))) (f_4: *length*) $f_5(n)$ = (f_0 (= n) (λ (x) x) (+ 1) 0) (f_5: *range*)

McCarthy 1959 Lisp ⟶ Modern functional programming

22 tasks. 64 CPUs. 93 hours.

Symbolic regression from visual input



Symbolic Regression

$f(\mathrm{x}) = (f_1\ \texttt{x})$

$f(\mathrm{x}) = (f_6\ \texttt{x})$

$f(\mathrm{x}) = (f_4\ \texttt{x})$

$f(\mathrm{x}) = (f_3\ \texttt{x})$

$f_0(\mathrm{x}) = (\texttt{+ x real})$

$f_1(\mathrm{x}) = (f_0\ (\star\ \texttt{real x}))$

$f_2(\mathrm{x}) = (f_1\ (\star\ \texttt{x}\ (f_0\ \texttt{x})))$

$f_3(\mathrm{x}) = (f_0\ (\star\ \texttt{x}\ (f_2\ \texttt{x})))$

$f_4(\mathrm{x}) = (f_0\ (\star\ \texttt{x}\ (f_3\ \texttt{x})))$

(f_4: 4th order polynomial)

$f_5(\mathrm{x}) = (\texttt{/ real x})$

$f_6(\mathrm{x}) = (f_5\ (f_0\ \texttt{x}))$

(f_6: rational function)

Turtle graphics — Created & investigated by Mathias Sablé–Meyer

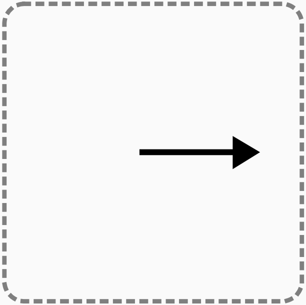
DSL

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

Tasks

```
task :  image
```

```
FW 1
```



Turtle graphics — Created & investigated by Mathias Sablé–Meyer

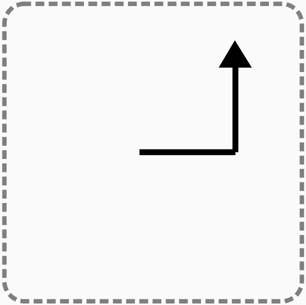
DSL

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

Tasks

```
task :  image
```

```
FW 1
RT $\frac{\pi}{2}$
FW 1
```



Turtle graphics — Created & investigated by Mathias Sablé–Meyer

DSL

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

Tasks

```
task :  image
```

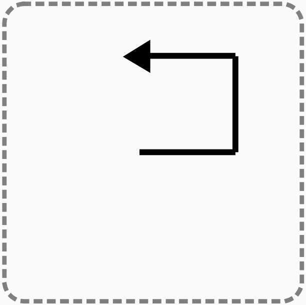
FW 1

RT $\frac{\pi}{2}$

FW 1

RT $\frac{\pi}{2}$

FW 1



Turtle graphics — Created & investigated by Mathias Sablé–Meyer

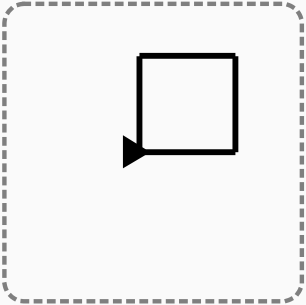
DSL

`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

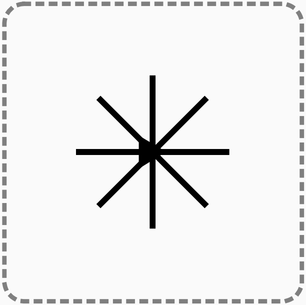
DSL

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

Tasks

`task :  image`

```
for i in range(8)
> FW 1
> SET origin
> RT $\frac{2\pi}{8}$
```



Turtle graphics — Created & investigated by Mathias Sablé–Meyer

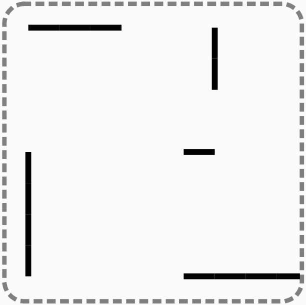
DSL

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

Tasks

```
task :  image
```

```
for i in range(8)
> PU
> FW $\frac{i}{2}$
> PD
> FW $\frac{i}{2}$
> RT $\frac{\pi}{2}$
```



Turtle graphics — Created & investigated by Mathias Sablé–Meyer

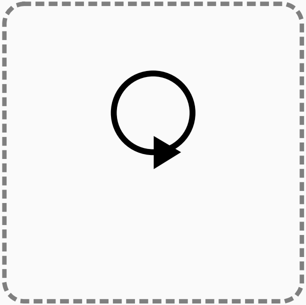
DSL

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

Tasks

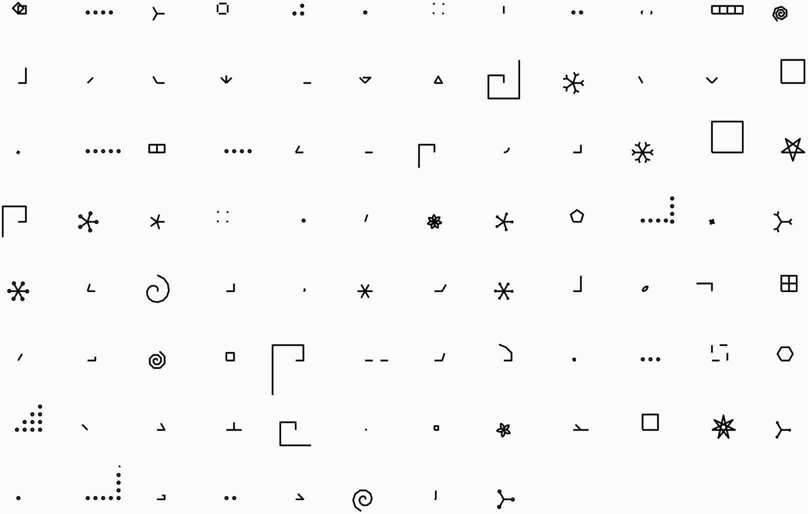
`task :  image`

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

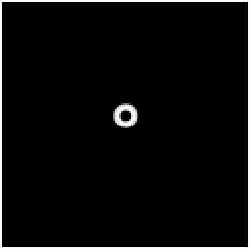


NUM ::= 1 | π | ∞ | ε | + | - | * | /

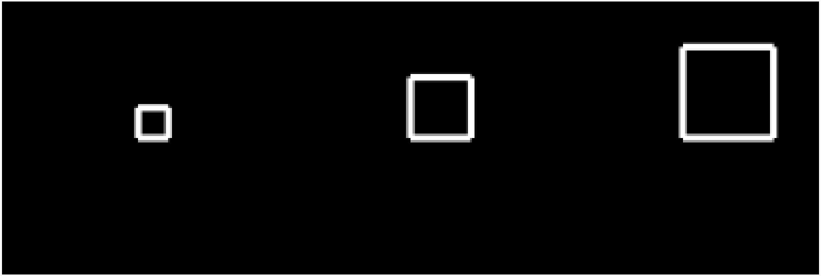
Turtle graphics — Training tasks



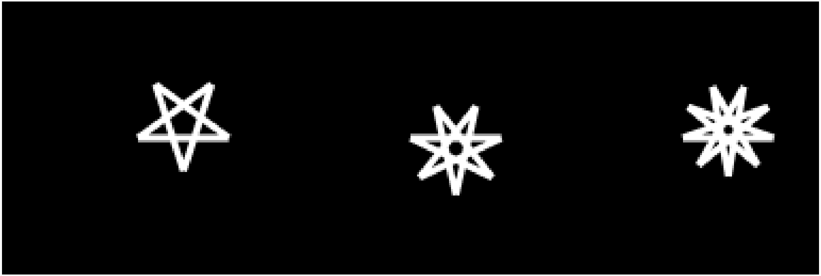
Turtle graphics — Illustration of learned DSL



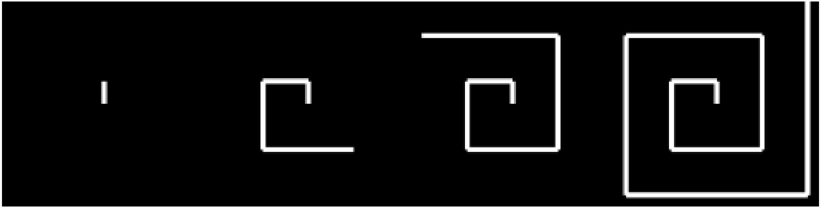
Turtle graphics — Illustration of learned DSL



Turtle graphics — Illustration of learned DSL

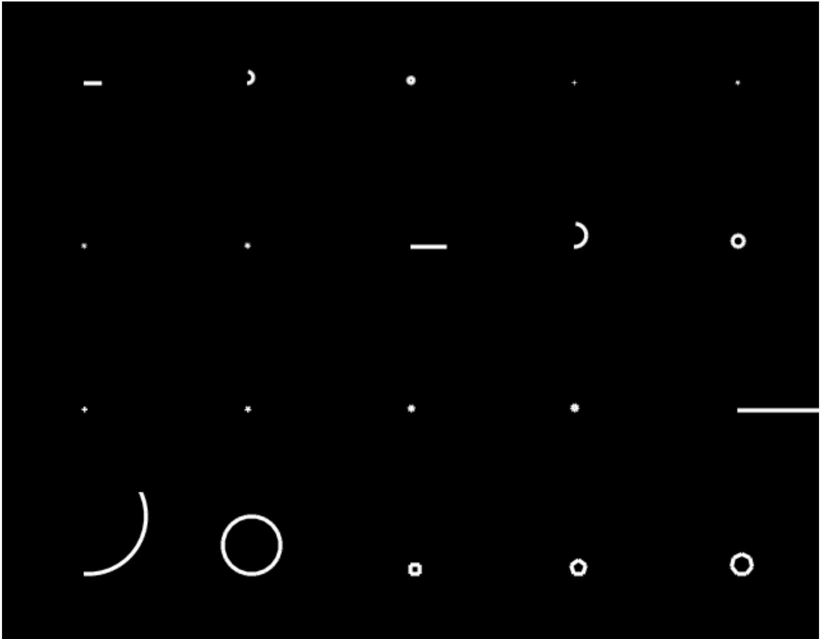


Turtle graphics — Illustration of learned DSL



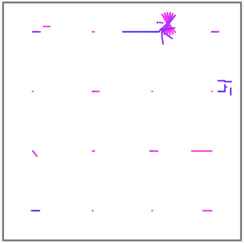
Turtle graphics — Illustration of learned DSL

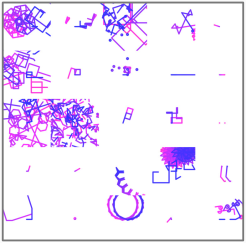
Turtle graphics — Illustration of learned DSL

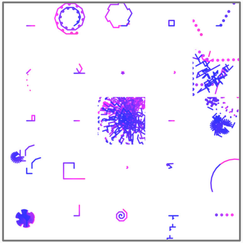


Turtle graphics — Dreams

Before training



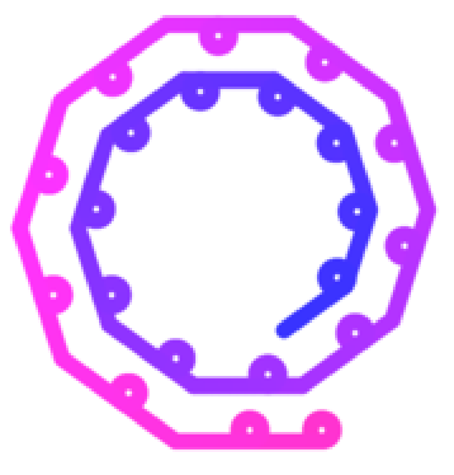




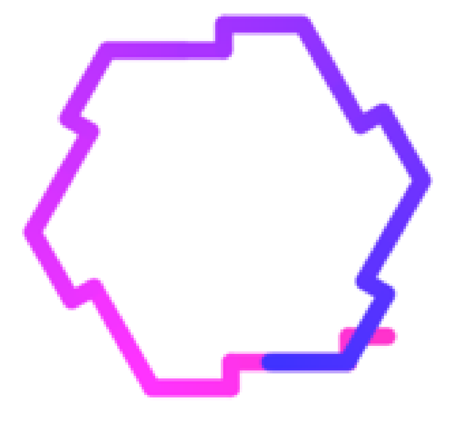
Plateau 5 minutes

Plateau 2 hours

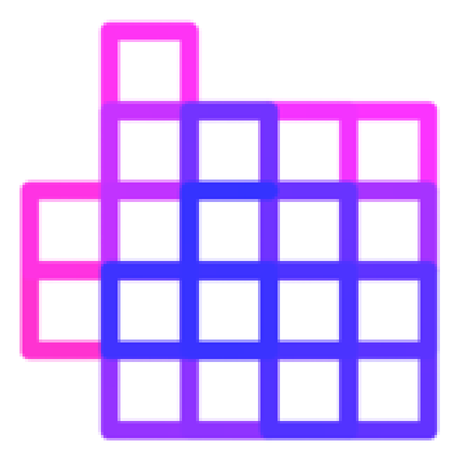
Turtle graphics — Dreaming from learned generative model



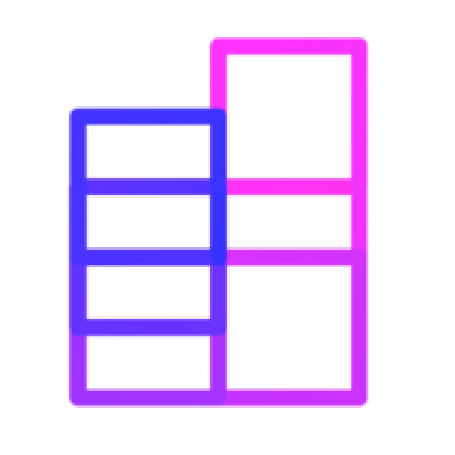
Turtle graphics — Dreaming from learned generative model



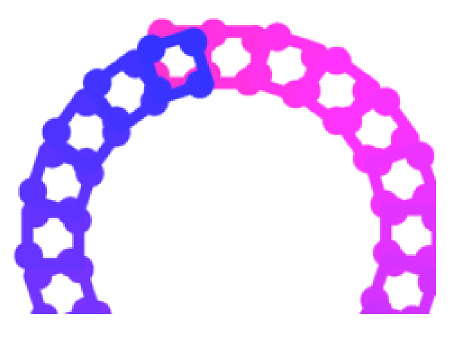
Turtle graphics — Dreaming from learned generative model



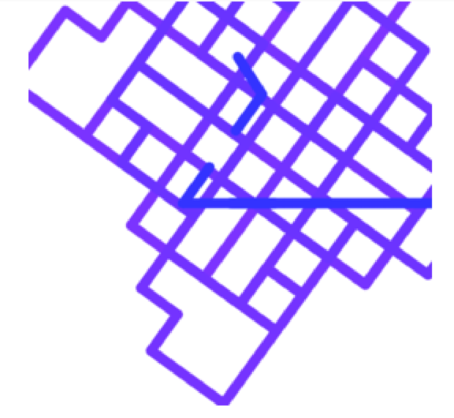
Turtle graphics — Dreaming from learned generative model



Turtle graphics — Dreaming from learned generative model



Turtle graphics — Dreaming from learned generative model

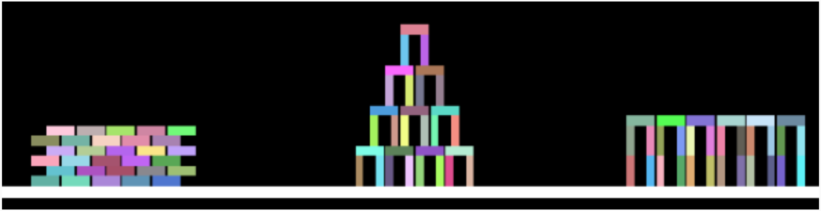


Turtle graphics — Dreaming from learned generative model

Turtle graphics — Dreaming from learned generative model

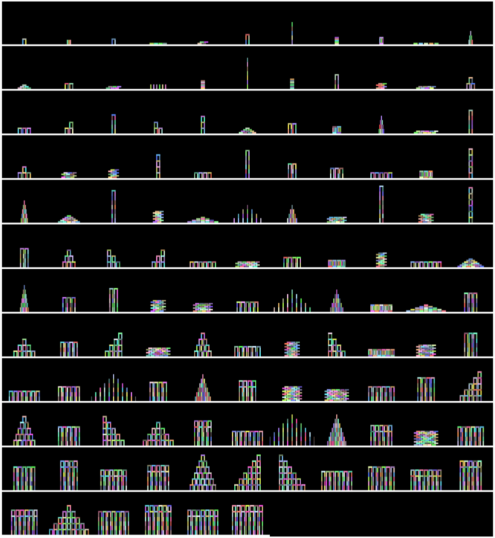
Tower building in blocks world

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



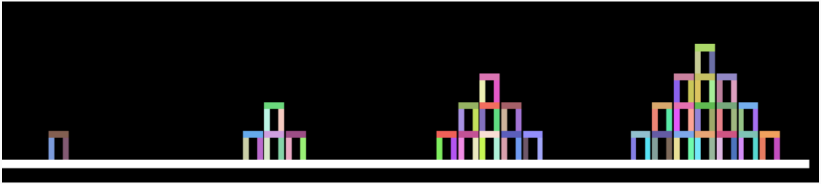
Tower building in blocks world

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



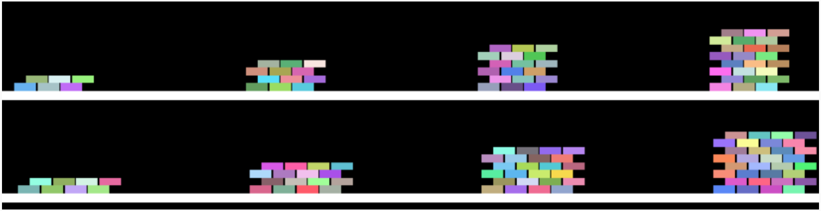
Tower building in blocks world: Learned concepts

Parametric planning primitives. Example pyramid concept:



Tower building in blocks world: Learned concepts

Parametric planning primitives. Example brickwall concept:

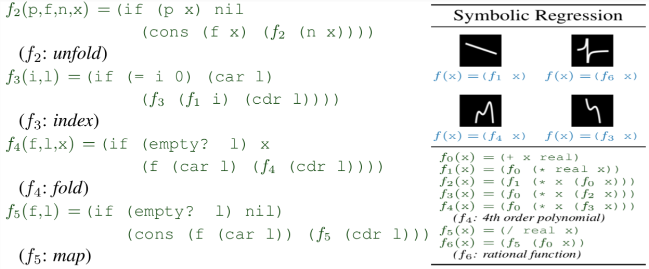


Vision

More human-like machine intelligence

- Acquiring a domain-specific representation (DSL)
- Learning to use that representation (recognition model)

DreamCoder: an algorithm for jointly realizing these goals

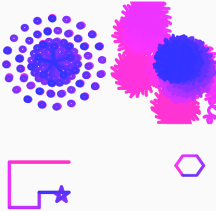


```
f2(p,f,n,x) = (if (p x) nil
                  (cons (f x) (f2 (n x))))
  (f2: unfold)
f3(i,l) = (if (= i 0) (car l)
                  (f3 (f1 i) (cdr l))))
  (f3: index)
f4(f,l,x) = (if (empty? l) x
                  (f (car l) (f4 (cdr l))))
  (f4: fold)
f5(f,l) = (if (empty? l) nil)
          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

Symbolic Regression

```
f(x) = (f1 x)    f(x) = (f6 x)
f(x) = (f4 x)    f(x) = (f3 x)
```

```
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)
```

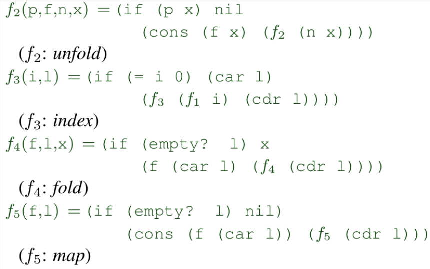


Vision

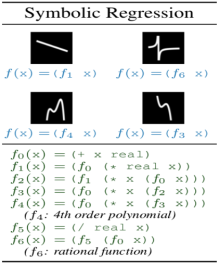
More human-like machine intelligence

- Acquiring a domain-specific representation (DSL)
- Learning to use that representation (recognition model)

DreamCoder: an algorithm for jointly realizing these goals

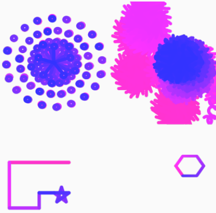


```
f2(p,f,n,x) = (if (p x) nil
                  (cons (f x) (f2 (n x))))
 (f2: unfold)
f3(i,l) = (if (= i 0) (car l)
                  (f3 (f1 i) (cdr l))))
 (f3: index)
f4(f,l,x) = (if (empty? l) x
                  (f (car l) (f4 (cdr l))))
 (f4: fold)
f5(f,l) = (if (empty? l) nil)
             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



Symbolic Regression

```
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)
```



The End.