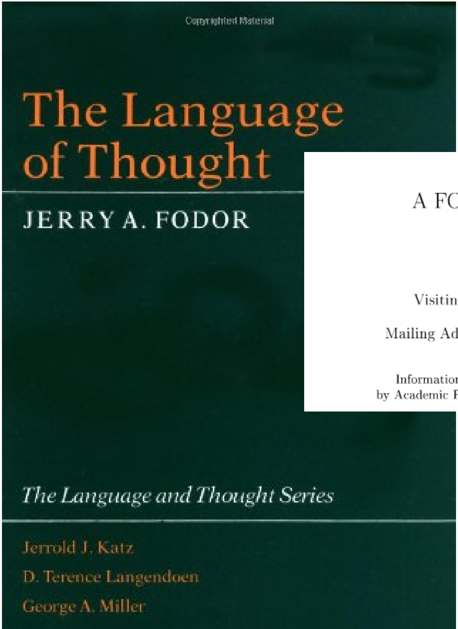
Growing libraries of concepts with wake-sleep program induction

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

July 24, 2018

MIT

The Language of Thought



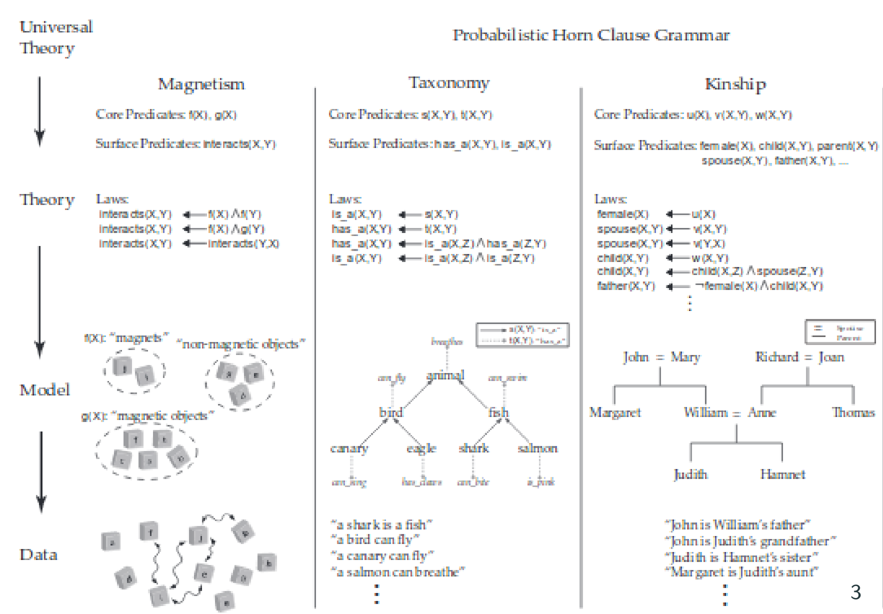
A FORMAL THEORY OF INDUCTIVE INFERENCE, Part 1*†

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.

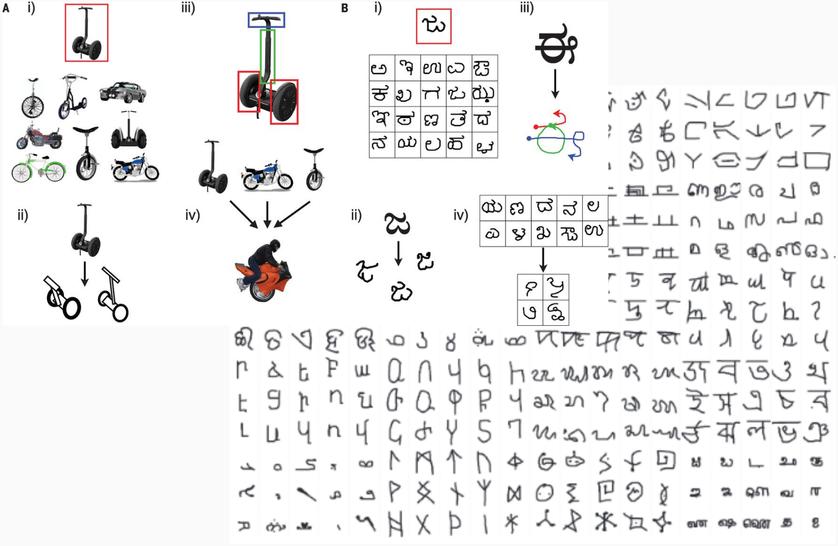
Information and Control, Volume 7, No. 1, Pp. 1-22, March 1964 Copyright by Academic Press Inc.

Engineering the language of thought



Ullman et al 2012

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

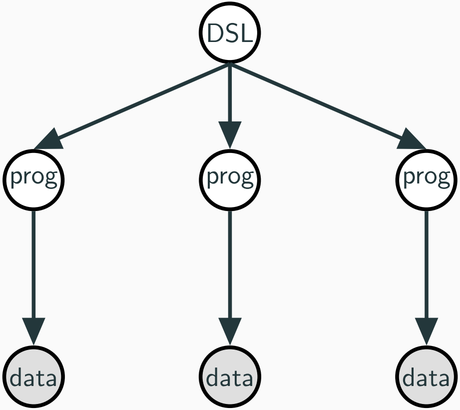
Tasks and Programs	DSL
[7 2 3]→[7 3] [1 2 3 4]→[3 4] [4 3 2 1]→[4 3] $f(\ell) =$(f$_1$ ℓ (λ (x) (> x 2))) [7 3]→False [3]→False [9 0 0]→True [0]→True [0 7 3]→True $f(\ell) =$(f$_3$ ℓ 0) [2 7 8 1]→8 [3 19 14]→19 $f(\ell) =$(f$_2$ ℓ)	$f_0(\ell,$r$) =$ (foldr r ℓ cons) (f_0: *Append lists* r *and* ℓ) $f_1(\ell,$p$) =$ (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) $f_3(\ell,$k$) =$ (foldr ℓ (is-nil ℓ) (λ (x a) (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

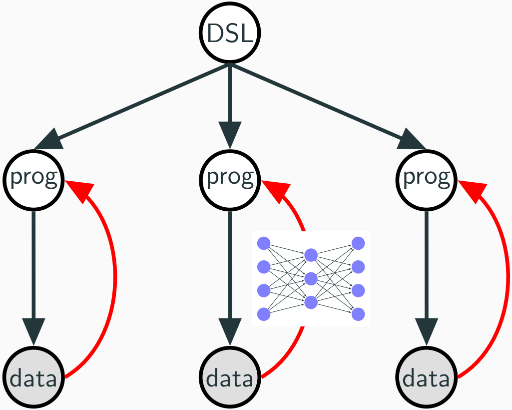
Combines ideas from Wake-Sleep & Exploration-Compression algorithm by Eyal Dechter

DSL learning as Bayesian inference

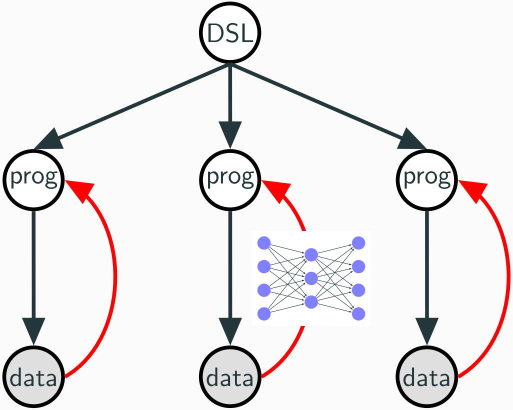


[Dechter et al., 2013] [Liang et al, 2010]; [Lake et al, 2015]

DSL learning as amortized Bayesian inference

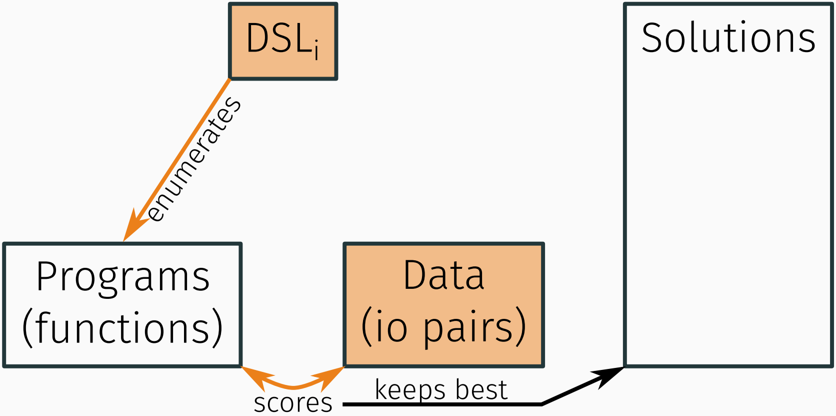


DSL learning as amortized Bayesian inference



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

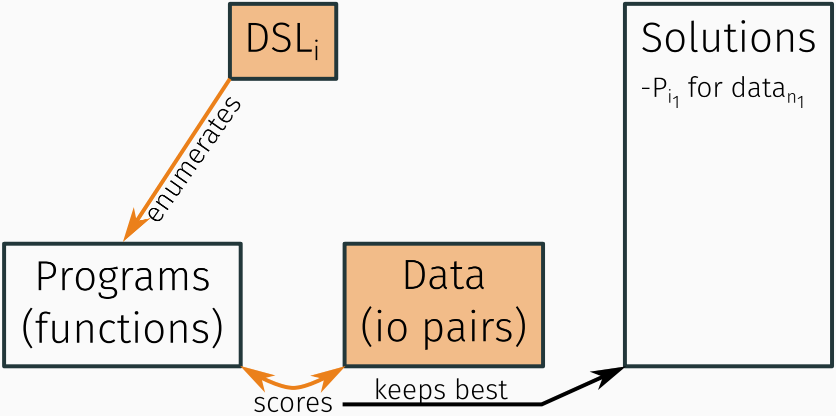
Wake — as in Exploration/Compression Algorithm



DSL	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))
```

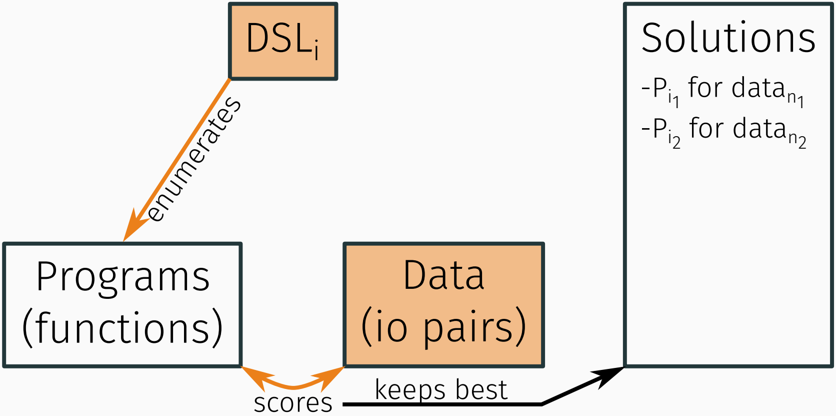
Programs

```
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))
```

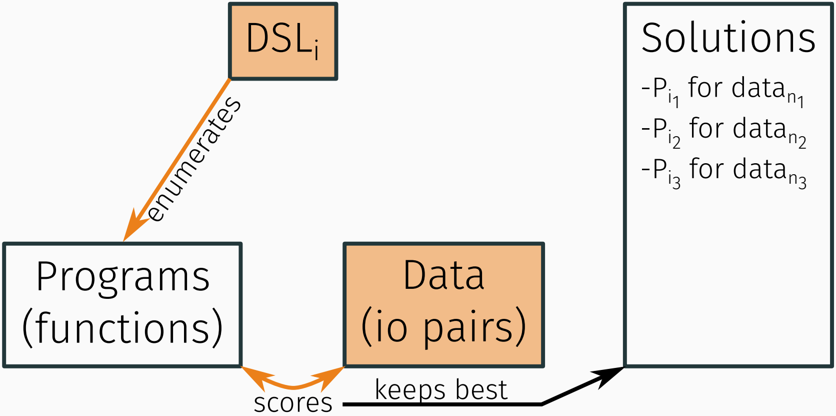
Programs

```
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

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

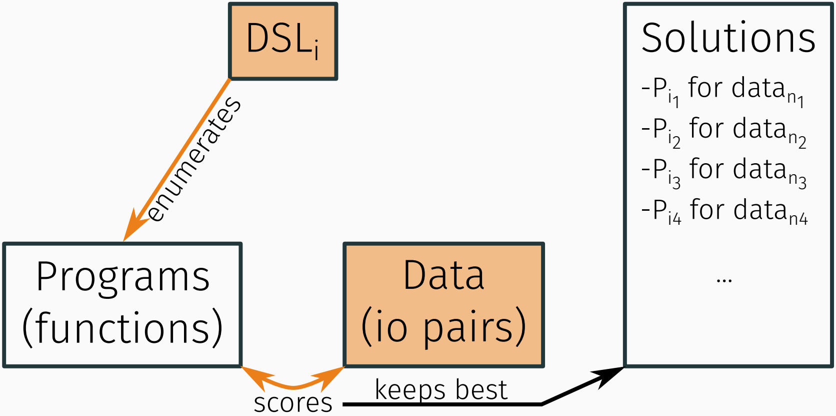
Programs

```
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))
```

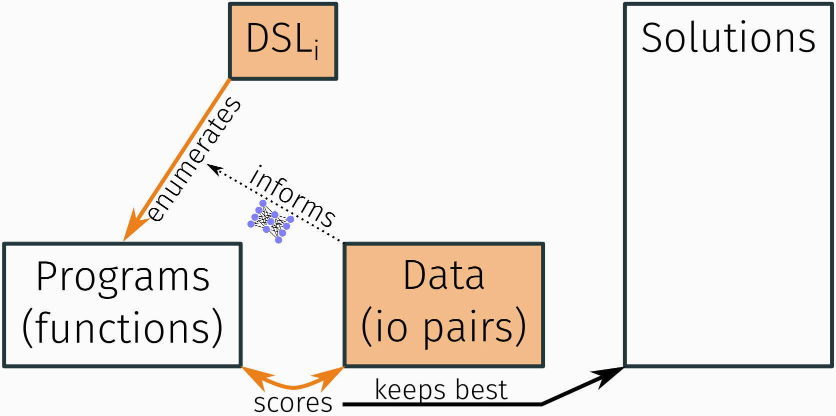
Programs

```
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]
```

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))
```

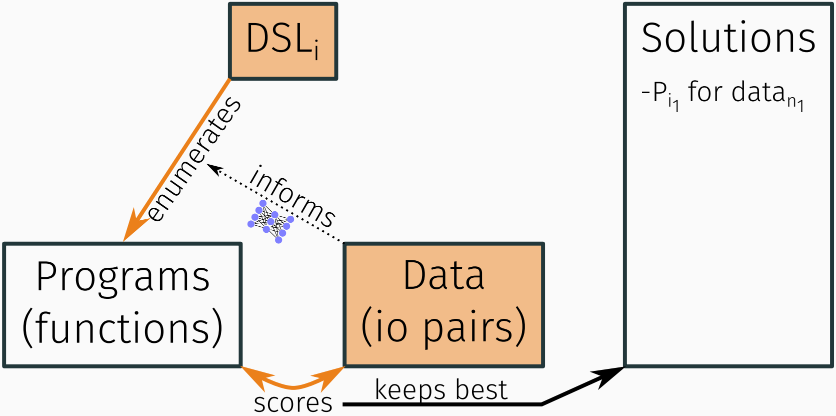
Programs

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

f(l) = (f1 l (λx. > x 2))
```

Data

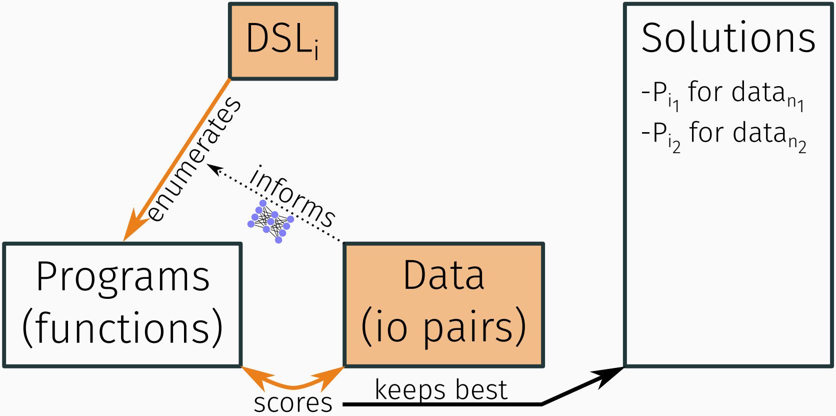
```
[7 2]→[7 2 7 2]
["a"]→["a" "a"]

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

DreamCoder — Wake



DSL	Programs	Data
f_0(l,r) = foldr r l cons	f(l) = (f_0 l l)	[7 2]→[7 2 7 2] ["a"]→["a" "a"]
f_1(l,p) = foldr r nil (λ (x a) (if (p x) (cons x a) a))	f(l) = (f_1 l (λx. > x 2))	[7 2 3]→[7 3] [1 2 3 4]→[3 4] [4 3 2 1]→[4 3]

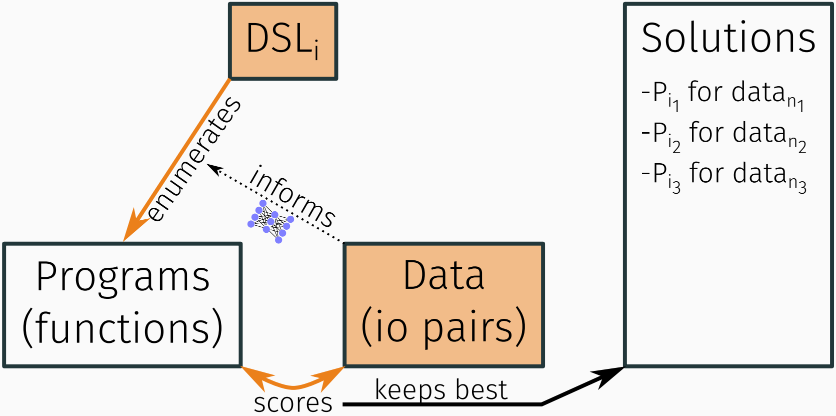
DreamCoder — Wake



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))
```

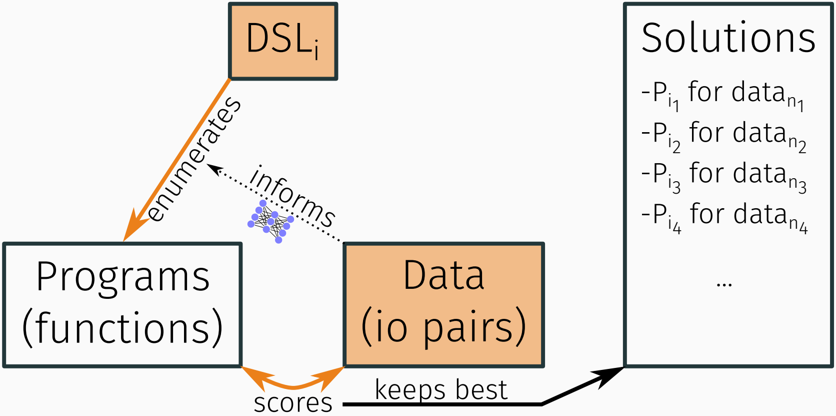
Programs

```
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]
```

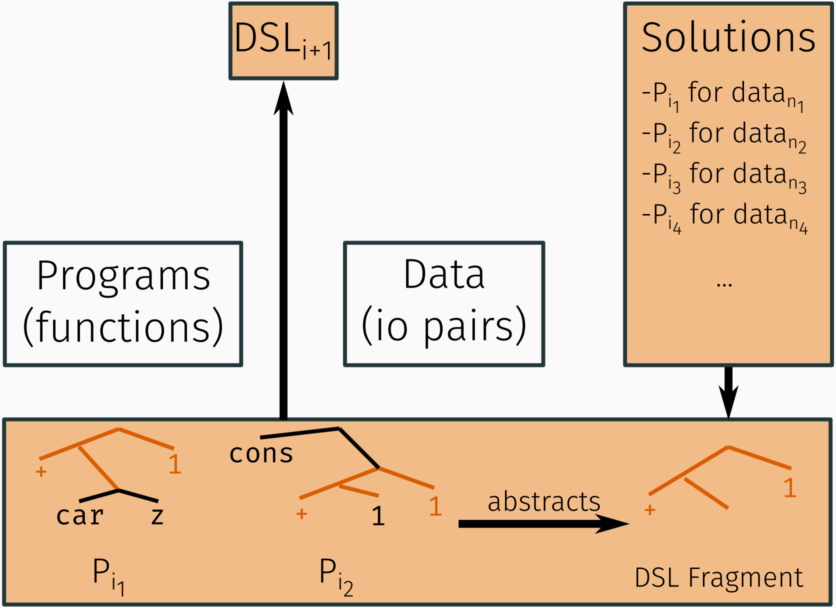
DreamCoder — Wake



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 — Sleep-G



DreamCoder — Sleep-R (Experience Replay)

DSL_{i+1}

Solutions

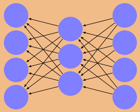
- P_{i_1} for data_{n_1}
- P_{i_2} for data_{n_2}
- P_{i_3} for data_{n_3}
- P_{i_4} for data_{n_4}

...

Programs (functions)

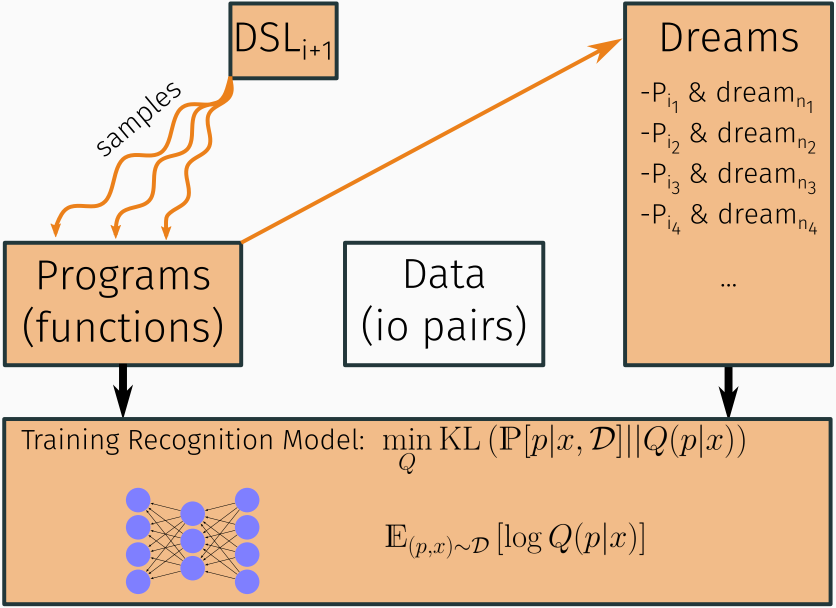
Data (io pairs)

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)

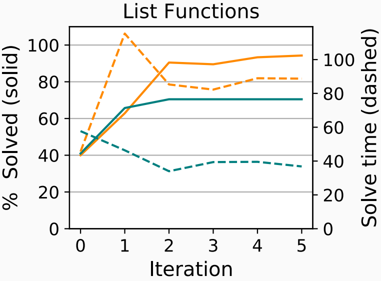


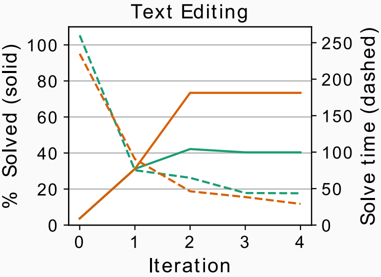
List functions — Created & investigated by Lucas Morales

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 suitable curriculum can also learn 'map', 'fold', etc.

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.

Turtle graphics — Created & investigated by Mathias Sablé–Meyer

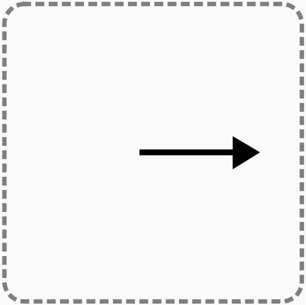
DSL

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

Tasks

`task :  image`

```
FW 1
```



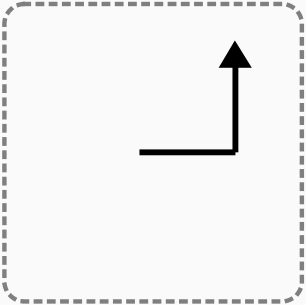
DSL

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

Tasks

`task : image`

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



DSL

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

Tasks

task : image

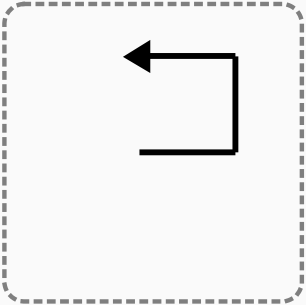
FW 1

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

FW 1

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

FW 1



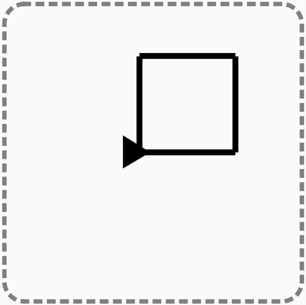
DSL

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

Tasks

`task : image`

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



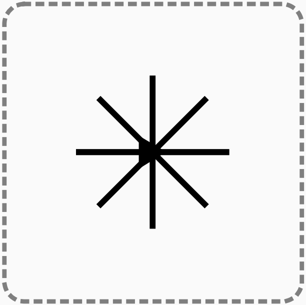
DSL

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

Tasks

`task :  image`

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



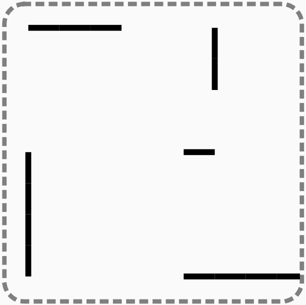
DSL

`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
```



Turtle graphics — Created & investigated by Mathias Sablé–Meyer

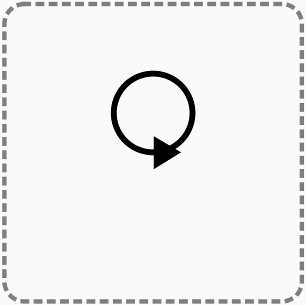
DSL

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

Tasks

```
task :  image
```

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



DSL

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

Tasks

`task :  image`

```
for i in range(5 × ∞)
> FW i × ε
> RT ε
```

DSL

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

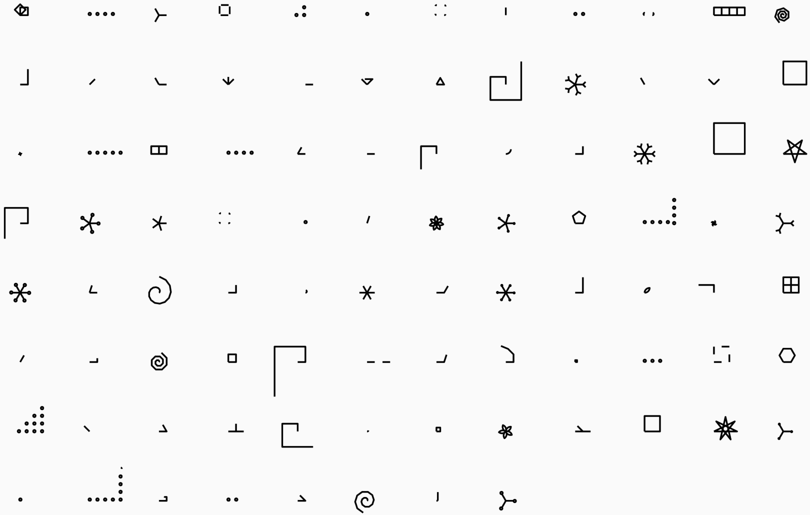
Tasks

`task :  image`

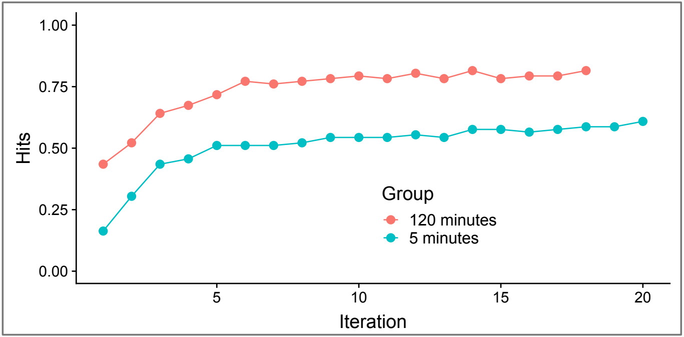
```
for i in range(5 × ∞)
> FW i × ε
> RT ε
```

`NUM ::= 1 | π | ∞ | ε | + | - | * | /`

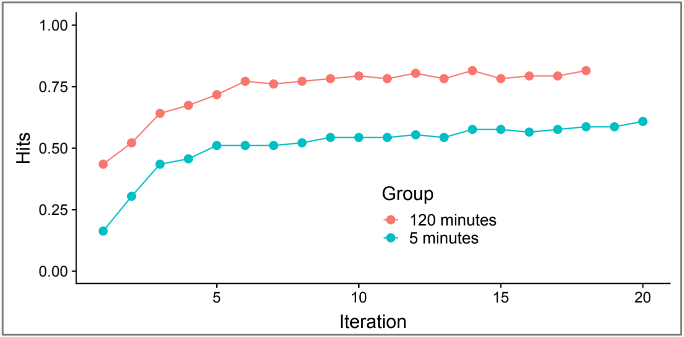
Turtle graphics — Training tasks



Turtle graphics — Learning curves



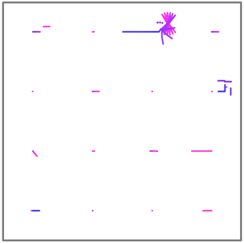
Turtle graphics — Learning curves

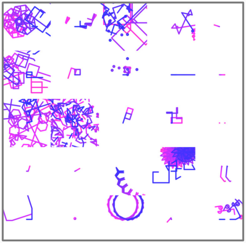


- $\frac{\pi}{2}$ and $\frac{\pi}{4}$ from π, 2, $+$ and $/$
- A line of length `n` followed with a right angle
- Loops of length `n` that uses the number `n` inside.
- Unit line then teleport back to origin
- ...

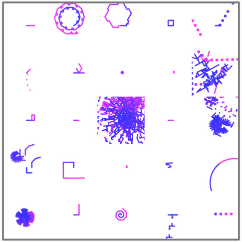
Turtle graphics — Dreams

Before training



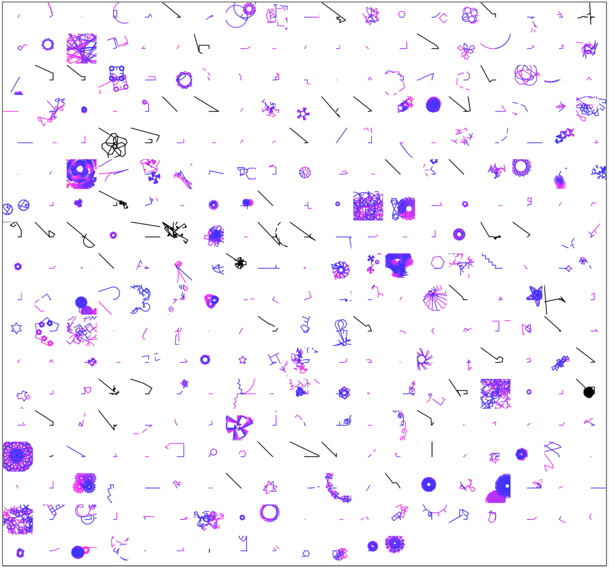


Plateau 5 minutes

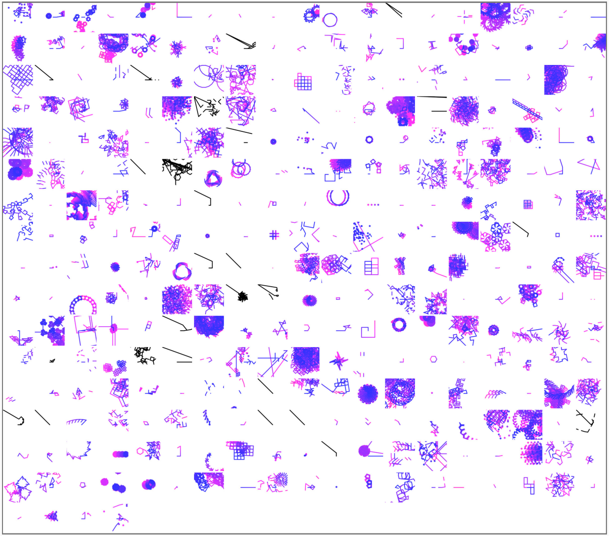


Plateau 2 hours

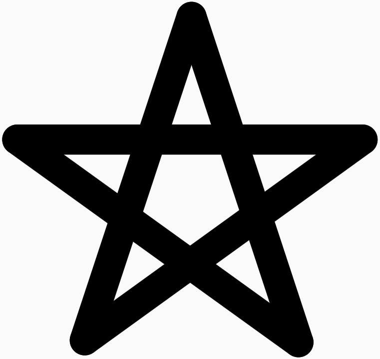
Turtle graphics — More dreams, 5 minutes, 1st iteration



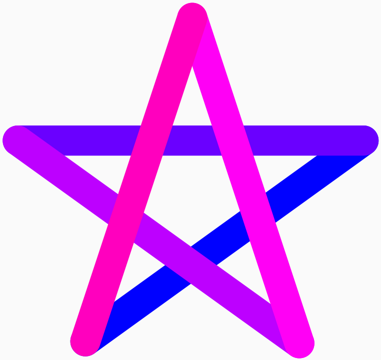
Turtle graphics — More dreams, 5 minutes, last iteration



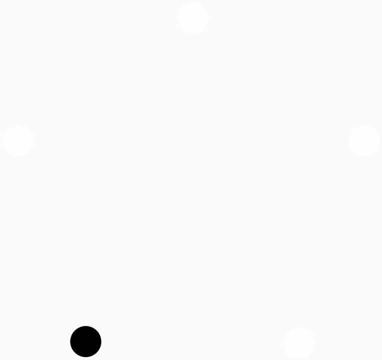
Turtle graphics — All the way down



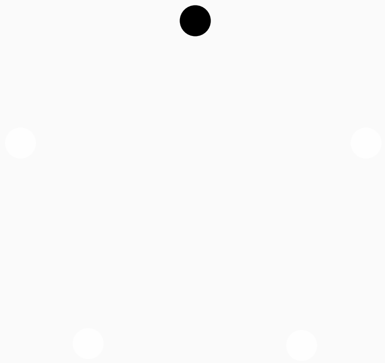
Turtle graphics — All the way down



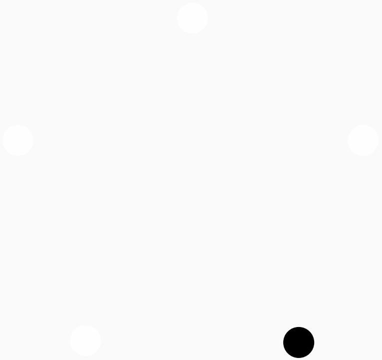
Turtle graphics — All the way down



Turtle graphics — All the way down



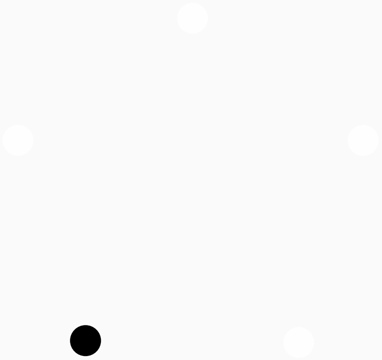
Turtle graphics — All the way down



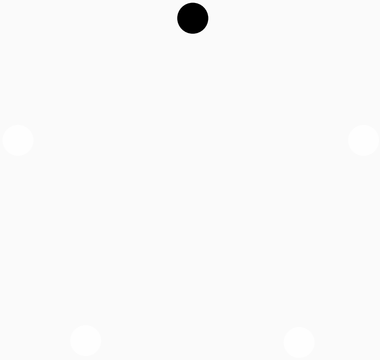
Turtle graphics — All the way down

Turtle graphics — All the way down

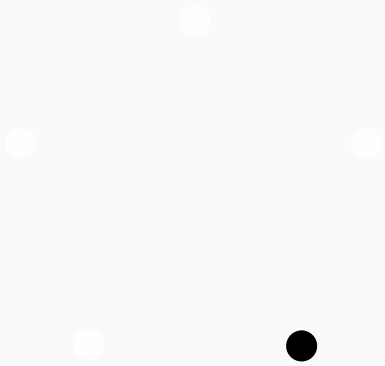
Turtle graphics — All the way down



Turtle graphics — All the way down



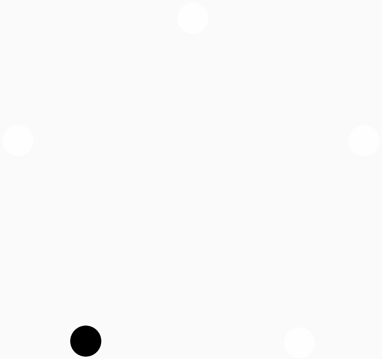
Turtle graphics — All the way down



Turtle graphics — All the way down

Turtle graphics — All the way down

Turtle graphics — All the way down



Turtle graphics — All the way down

Turtle graphics — All the way down

Turtle graphics — All the way down

Turtle graphics — All the way down

Turtle graphics — All the way down

Turtle graphics — All the way down

Turtle graphics — All the way down

Turtle graphics — All the way down

Turtle graphics — All the way down

Turtle graphics — All the way down

Turtle graphics — All the way down

Turtle graphics — All the way down

Turtle graphics — All the way down

Turtle graphics — All the way down

Turtle graphics — All the way down

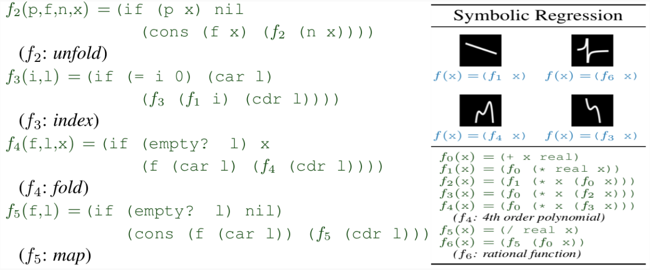
Turtle graphics — All the way down

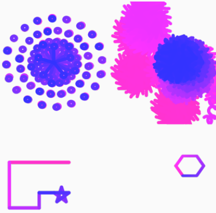
Turtle graphics — All the way down

Vision

More human-like machine intelligence

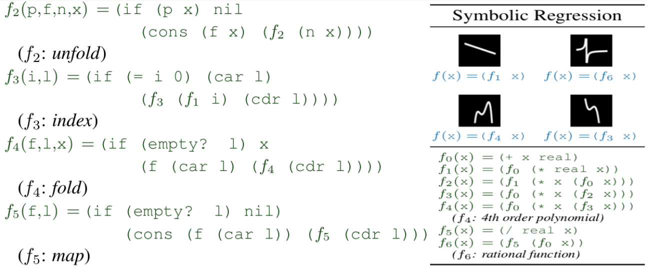
Flexibly adapting to new problem domains: acquiring a domain-specific language-of-thought (DSL); learning how to use the language of thought (recognition model)

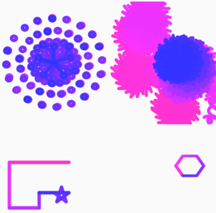




More human-like machine intelligence

Flexibly adapting to new problem domains: acquiring a domain-specific language-of-thought (DSL); learning how to use the language of thought (recognition model)





The End.