Explore/Compress/Compile (EC$^2$) learns to solve programming tasks like these by growing a library of code and training a neural net to search for programs written using the library.
<table>
<thead>
<tr>
<th>Tasks and Programs</th>
<th>DSL</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>[7 2 3] → [7 3]</code></td>
<td>( f_0(\ell, r) = (\text{foldr } r \ \ell \ \text{cons}) )</td>
</tr>
<tr>
<td><code>[1 2 3 4] → [3 4]</code></td>
<td>( (f_0: \text{Append lists } r \text{ and } \ell) )</td>
</tr>
<tr>
<td><code>[4 3 2 1] → [4 3]</code></td>
<td>( f_1(\ell, p) = (\text{foldr } \ell \ \text{nil} \ (\lambda (x \ a) \ (\text{if} \ (p \ x) \ (\text{cons} \ x \ a) \ a))) )</td>
</tr>
<tr>
<td><code>f(\ell) = (f_1 \ \ell \ (\lambda (x) (&gt; x 2)))</code></td>
<td>( (f_1: \text{Higher-order filter function}) )</td>
</tr>
<tr>
<td><code>[7 3] → \text{False}</code></td>
<td>( f_2(\ell) = (\text{foldr } \ell \ 0 \ (\lambda (x \ a) \ (\text{if} \ (&gt; a \ x) \ a x))) )</td>
</tr>
<tr>
<td><code>[9 0 0] → \text{True}</code></td>
<td>( (f_2: \text{Maximum element in list } \ell) )</td>
</tr>
<tr>
<td><code>[0] → \text{True}</code></td>
<td>( f_3(\ell, k) = (\text{foldr } \ell \ (\text{is-nil } \ell) \ (\lambda (x \ a) \ (\text{if} \ a \ a \ (= k x)))) )</td>
</tr>
<tr>
<td><code>[0 7 3] → \text{True}</code></td>
<td>( (f_2: \text{Whether } \ell \text{ contains } k) )</td>
</tr>
<tr>
<td><code>f(\ell) = (f_2 \ \ell)</code></td>
<td></td>
</tr>
</tbody>
</table>

- Learned DSL primitives can call each other
- RedisCOVERs higher-order functions like filter
Explore/Compress/Compile as Amortized Bayesian Inference

[Diagram showing the process of Explore/Compress/Compile with nodes labeled DSL, prog, task, and arrows indicating the flow of information.]
Explore/Compress/Compile as Amortized Bayesian Inference

**Explore**: Infer programs, fixing DSL and neural net

- **DSL**
  - Recog. model
  - Task

- **Search** → **Programs**

- **Compress**: Update DSL, fixing programs

- **Compile**: Train recognition model

- **DSL**
  - **prog** → **task**
  - **prog** → **task**
  - **prog** → **task**

Image: Diagram illustrating the Explore/Compress/Compile process with arrows connecting **DSL**, **Search**, **Programs**, **Compress**, and **Compile**.
Explore/Compress/Compile as Amortized Bayesian Inference

**Explore:** Infer programs, fixing DSL and neural net

**Compress:** Update DSL, fixing programs

DSL → Search → Programs

DSL

Recog. model

Task

DSL

DSL+

progs. for task

progs. for task

DSL

prog

task

prog

task

prog

task

is

cons

+ 1 1

+ 1

car z

DSL+
Explore/Compress/Compile as Amortized Bayesian Inference

**Explore**: Infer programs, fixing DSL and neural net

DSL → Search → Programs → Task

**Compress**: Update DSL, fixing programs

progs. for task_1 → cons + 1 1 → DSL+

progs. for task_2 → + 1 → car z

**Compile**: Train recognition model

DSL → sample → program → run → task

DSL+
Explore/Compress/Compile as Amortized Bayesian Inference

**Explore**: Infer programs, fixing DSL and neural net

**Compress**: Update DSL, fixing programs

**Compile**: Train recognition model
Domain: List processing

Starts with: foldr, unfold, if, map, length, index, =, +, −, 0, 1, cons, car, cdr, nil, is-nil, mod, *, >, is-square, is-prime.

236 human-interpretable list processing tasks.

Discovers 38 new DSL primitives, including filter

<table>
<thead>
<tr>
<th>Name</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>repeat-2</td>
<td>[7 0]</td>
<td>[7 0 7 0]</td>
</tr>
<tr>
<td>drop-3</td>
<td>[0 3 8 6 4]</td>
<td>[6 4]</td>
</tr>
<tr>
<td>rotate-2</td>
<td>[8 14 1 9]</td>
<td>[1 9 8 14]</td>
</tr>
<tr>
<td>count-head-in-tail</td>
<td>[1 2 1 1 3]</td>
<td>2</td>
</tr>
<tr>
<td>keep-mod-5</td>
<td>[5 9 14 6 3 0]</td>
<td>[5 0]</td>
</tr>
<tr>
<td>product</td>
<td>[7 1 6 2]</td>
<td>84</td>
</tr>
</tbody>
</table>

With functional programming “problem set” + 93 hours on 64 CPUs, rediscovers: map, foldr, unfold, range, length, index, zip, +some arithmetic routines
Domain: Text Editing

In the style of FlashFill (Gulwani 2012). Starts with: `foldr, unfold, if, map, length, index, =, +, −, 0, 1, cons, car, cdr, nil, is-nil`, plus string & character constants.

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>+106 769-438</td>
<td>106.769.438</td>
</tr>
<tr>
<td>+83 973-831</td>
<td>83.973.831</td>
</tr>
</tbody>
</table>

Temple Anna H    TAH
Lara Gregori     LG

Figure: 11 learned DSL primitives over 3 successive iterations (3 columns). Learned primitives call each other (arrows).
Domain: Symbolic regression from visual input

Starts with: plus, times, divide, real-number. Autograds through program Bayesian likelihood $P[\text{data}|\text{prog}]$ favors fewer continuous parameters

<table>
<thead>
<tr>
<th>Programs &amp; Tasks</th>
<th>DSL</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f(x) = (f_1 \ x)$</td>
<td>$f_0(x) = (+ \ x \ \text{real})$</td>
</tr>
<tr>
<td>$f(x) = (f_5 \ x)$</td>
<td>$f_1(x) = (f_0 (* \ x \ \text{real} \ x))$</td>
</tr>
<tr>
<td>$f(x) = (f_4 \ x)$</td>
<td>$f_2(x) = (f_1 (* \ x (f_0 x)))$</td>
</tr>
<tr>
<td>$f(x) = (f_3 \ x)$</td>
<td>$f_3(x) = (f_0 (* \ x (f_2 x)))$</td>
</tr>
<tr>
<td>$f(x) = (f_4 \ x)$</td>
<td>$f_4(x) = (f_0 (* \ x (f_3 x)))$</td>
</tr>
<tr>
<td>$f(x) = (f_3 \ x)$</td>
<td>$(f_4: \ 4\text{th order polynomial})$</td>
</tr>
<tr>
<td>$f(x) = (f_5 \ x)$</td>
<td>$f_5(x) = (/ \ \text{real} \ (f_0 x))$</td>
</tr>
<tr>
<td></td>
<td>$(f_5: \ \text{rational function})$</td>
</tr>
</tbody>
</table>
New domain: Generative graphics programs (Turtle/LOGO)

Training tasks

DSL samples before learning

DSL samples after learning

Learned DSL contains parametric drawing routines:
- Semicircle
- Spiral
- Greek Spiral
- S-Curves
- Polygons & Stars
- Circles
New domain: Generative graphics programs (Turtle/LOGO)

Training tasks

DSL samples before learning

DSL samples after learning

Learned DSL contains parametric drawing routines:

Semicircle:

Greek Spiral:

Polygons & Stars:

Spiral:

S-Curves:

Circles:
\( f_2(p, f, n, x) = (\text{if} \ (p \ x) \ \text{nil} \ \ \ \ \ \ \ \ \ \ (\text{cons} \ (f \ x) \ (f_2 \ (n \ x))))) \)

\( (f_2: \text{ unfold}) \)

\( f_3(i, l) = (\text{if} \ \ (= \ i \ 0) \ \ (\text{car} \ l) \ (f_3 \ (f_1 \ i) \ (\text{cdr} \ l))) \)

\( (f_3: \text{ index}) \)

\( f_4(f, l, x) = (\text{if} \ \ (\text{empty?} \ l) \ \ x \ (\text{f} \ (\text{car} \ l) \ (f_4 \ (\text{cdr} \ l)))) \)

\( (f_4: \text{ fold}) \)

\( f_5(f, l) = (\text{if} \ \ (\text{empty?} \ l) \ \ \text{nil} \ (\text{cons} \ (f \ (\text{car} \ l)) \ (f_5 \ (\text{cdr} \ l)))) \)

\( (f_5: \text{ map}) \)