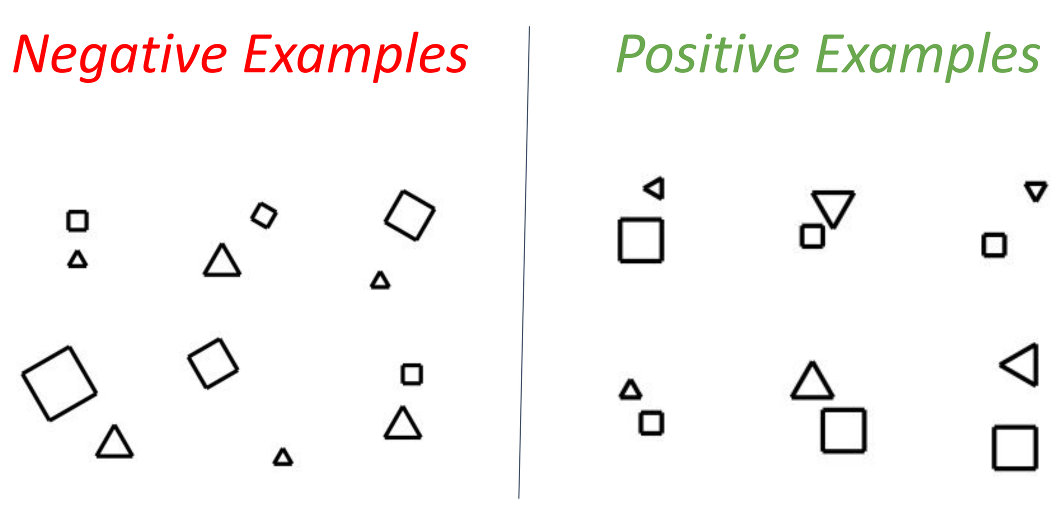
Using Program Synthesis and Inductive Logic Programming to solve Bongard Problems

Atharv Sonwane*, Sharad Chitlangia*, Tirtharaj Dash, Lovekesh Vig,
Gautam Shroff and Ashwin Srinivasan

Bongard Problems

Visual Reasoning tasks with 6 positive and 6 negative example images for a particular concept. Given the examples images, the task is to find the differentiating concept. For example -



Concept: ***Triangles above squares***.

We evaluate system on adaptations of: #4, #14, #16, #21, #23, #24, #36, #40, #53, #60, #75, #85, #94 and #96 from www.foundalis.com/res/bps

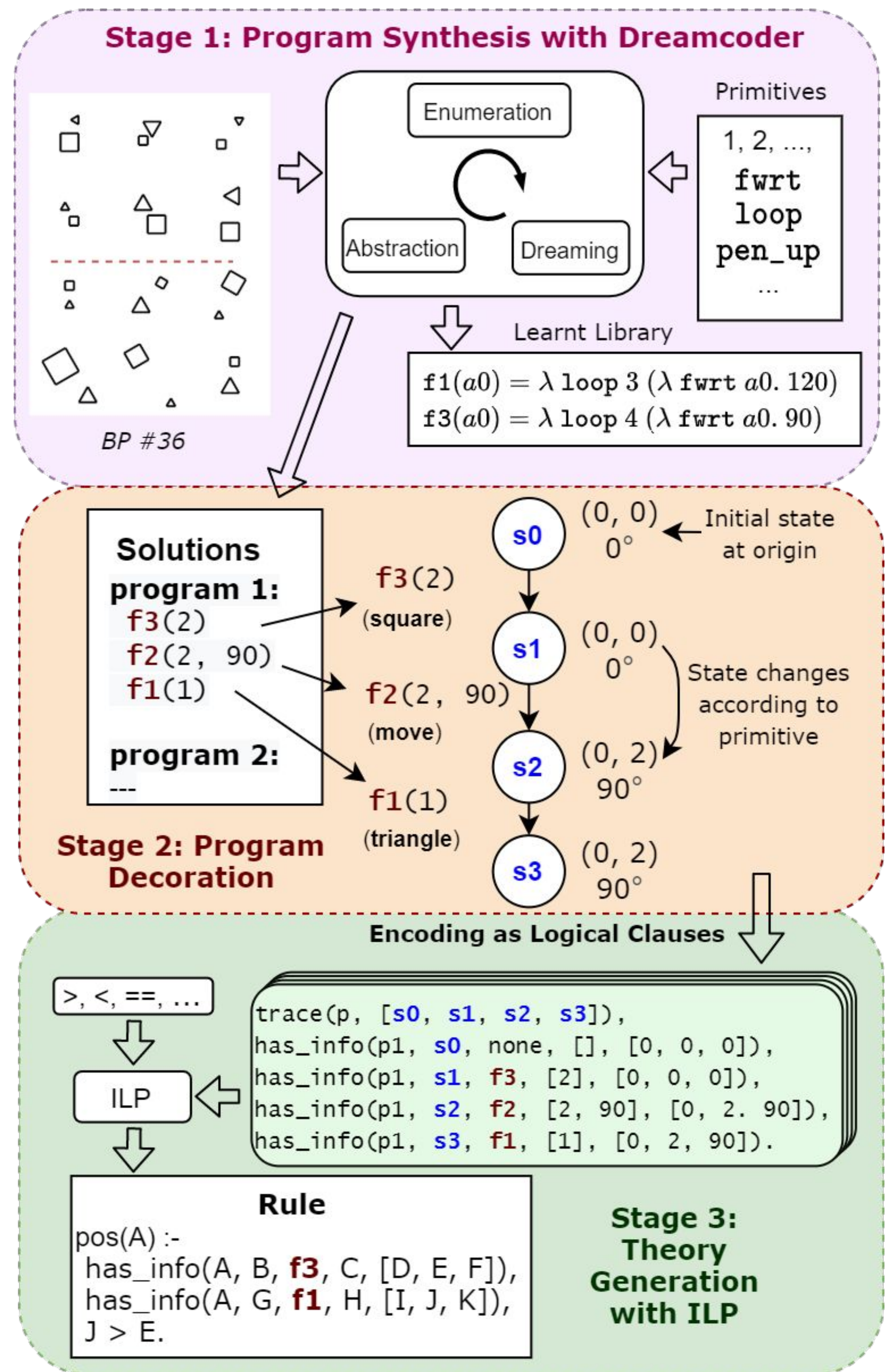
Overview

Solving reasoning tasks requires the use of suitable representations which can encapsulate relevant concepts. Such a representation should also allow flexibility in abstraction formation at various levels in the hierarchy.

In our 3-staged inductive programming system, we use decorated graphical programs to represent the images for Bongard Problems. We postulate that this allows for formation of concepts:

- At the first stage through invention by abstraction in functional λ-calculus programs using Dreamcoder. Such as learning a polygon from line instructions.
- By allowing for additional methods of information extraction from the solution program at the decoration phase during a debugger style step-by-step execution of the λ-calculus programs.
- On top of the decorated functional λ-calculus programs, using logical programs, through Inductive Logic Programming, to learn higher level concepts such as *Triangle above Square, Concavity / Convexity*

Method



1. **Synthesis**: All positive and negative images for a Bongard Problem are input to Dreamcoder to obtain a library of higher level primitives and solution programs for each image
2. **Decoration**: Each Program is converted into a decorated state transition diagram via a *debugger*-styled execution. Transitions are decorated with primitive calls and states with information such as the position and orientation of the cursor during the program execution.
3. **Theory Identification**: A FOL representation of the transitions using `has_info` and `trace` predicates, along with comparison predicates, are input, as background knowledge, to Aleph (an ILP Engine) to find a theory that differentiates between the positives and negatives.

Predicate Definitions

```
has_info(+Program,-State,#Primitive,-Args,[-X,-Y,-Angle])
trace(+Program,[-state0,-state1,-state2, ...])
```

Results

Our system is able to solve 8 of the 14 Bongard Problems considered. Some are illustrated below -

<table>
<tr><th>Concept</th><th>Invented Primitives (Dreamcoder)</th><th>Theory</th><th>Explanation</th></tr>
<tr><td>Anti-clockwise vs Clockwise

BP #16</td><td>f2(a0), f3(a0): both draw anti- clockwise spirals with different step lengths and with a0 controlling tightness of the spiral.</td><td>pos(A):-
has_info(A,B,f3,C,[D,E,F]).

pos(A):-
has_info(A,B,f2,C,[D,E,F]).</td><td>Presence of invented primitive for drawing spirals that are anticlockwise.</td></tr>
<tr><td>Smaller shape present

BP #21</td><td>f1(a0, a1): Draw an a0-sided polygon with sides of length a1</td><td>pos(A): -
has_info(A,B,rtfwint,C,[D,E,F]), C=[G|H], H=[I|J], G>I, has_info(A,K,f1,L,[D,E,F]).

pos(A):-
has_info(A,B,f1,C,[D,E,F]), C=[G|H], H=[I|J], G>I.</td><td>Program contains a move primitive where the division factor for angle is greater than multiplication factor for distance, or there is a polygon with side length less than number of sides. Indicating the shape is small.</td></tr>
<tr><td>Triangle above Square

BP #36</td><td>f1(a0): Draws triangle of side length a0.

f3(a0): Draws square of side length a0</td><td>pos(A):-
has_info(A,B,f3,C,[D,E,F]), has_info(A,G,f1,H,[I,J,K]), J>E.</td><td>Triangle exists with y coordinate greater than that of square</td></tr>
<tr><td>Enclosed shape has fewer sides

BP #53</td><td>f1(a0, a1): Draw an a0-sided polygon with sides of length a1</td><td>pos(A):-
has_info(A,B,f1,C,[D,E,F]), has_info(A,R,pt,Q,[K,L,M]), has_info(A,I,f1,J,[K,L,M]), C=[G|H],J=[N|O],O=[P|Q], G>N, N>P.</td><td>Smaller polygon (having length of side smaller than number of sides) has has fewer sides than larger (enclosing) polygon.</td></tr>
</table>

The main reasons where the system fails are -

- **Representation**: Inability to represent solid fills, arbitrary curves and other irregular features using current DSL.
- **Search**: High number of shapes / lines to be drawn meaning intractable search due to large program lengths of the solution.

Next Steps

Our work can be improved in 3 key areas:

- **Graphical Program Synthesis**:
  - A learned metric for comparison rather than pixel level comparison
  - Execution-guided synthesis rather than enumeration
- **State Decorations**:
  - Learned automated feature extractors to work on top of produced programs/images.
- **Final theory learning step**:
  - Construction of meta-rules among programs of different problems to learn general concepts such as *smallness*, etc
  - Construction of meta-rules, in the 2nd order, over sub-programs of the same problem