ArChEs Proposal

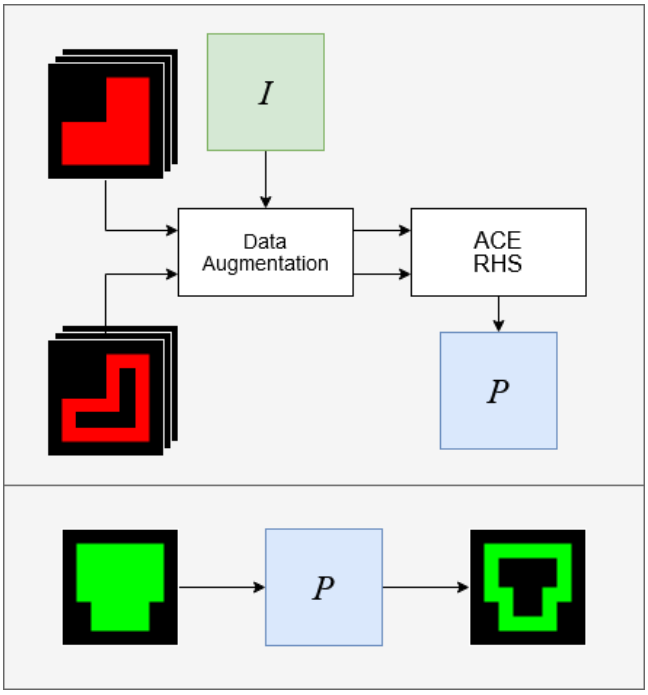
CS 560

Sean Flannery, Ryan Luu, Harjas Monga

February 2021

1 Introduction

$\begin{smallmatrix} A & C & E \\ R & H & S \end{smallmatrix}$ (pronounced "arches") stands for Abstraction on Constrained Examples, by Ryan, Harjas, and Sean. It is designed to synthesize programs that model reproducible image transformations using only examples (and optionally, knowledge of certain properties of the transformation). The figure below shows how input examples are augmented using invariants I. $\begin{smallmatrix} A & C & E \\ R & H & S \end{smallmatrix}$ searches our DSL for a sequence of instructions that match the transformation from the input examples and produces an optimally general program P according to an Occam's razor-based heuristic.



2 Specification

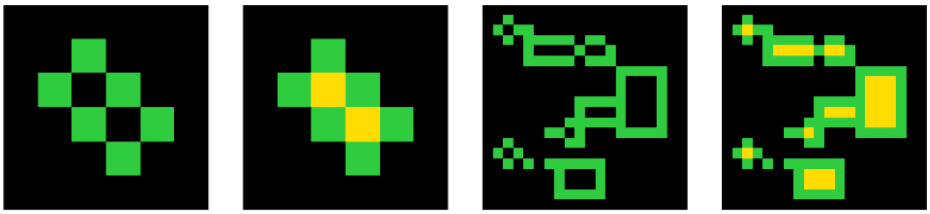
2.1 Description of Problem

Given a set of image pairs $S_I = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$, synthesize a program that models the function F, which satisfies $F(X_i) = Y_i$ for all $i \in 1, \ldots, n$. The program should also be able to generalize to images not found in S_I.

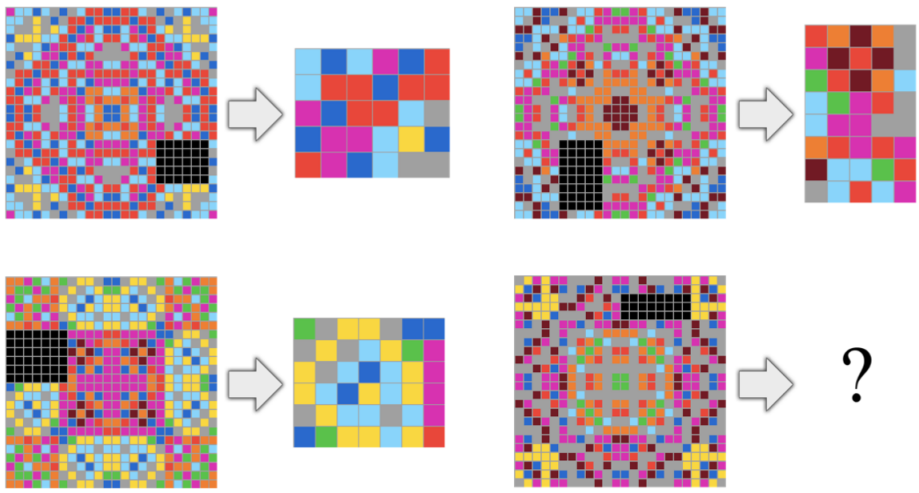
2.2 Sources of Input/Output Examples

The tasks (sets of image pairs) for this project are inspired by (or directly taken from) those used in the Abstraction & Reasoning Kaggle Competition. This competition was created by Francois Cholet, whose groundbreaking paper [1] on how to better measure the "intelligence" of AI systems. This work critiques the broader trend within the AI/ML community to compare human and machine intelligence on especially-specific tasks like Chess, Go, or Atari. The authors propose a series of improved AI benchmarks, which focus more on the ability of a system to perform generic abstraction and solving of visual tasks.

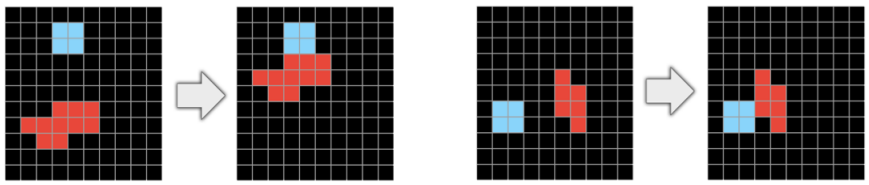
2.2.1 Sample Task: Flood-Fill



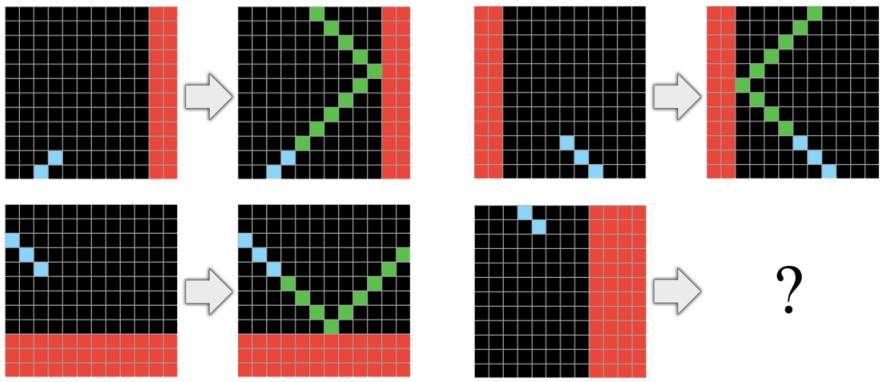
2.2.2 Sample Task: Pattern Completion



2.2.3 Sample Task: Collision Simulation



2.2.4 Sample Task: Trajectory Prediction



3 Approach

3.1 Semantic Properties

An important property that will help guide the data augmentation is the type of transformation being done. If the program we are trying to synthesize is focused on a structural component of the image, then we can mark the color as "unimportant" an property. We can then permute the color property to help augment the data set. On other hand, if we are attempting to synthesize a program that deals with color manipulation we could mark that the structure is the "unimportant" property. In this case we could apply transformations (rotations, flips, etc...) to the images to augment the data set.

3.2 Data Augmentation

For many of these types of tasks, we can augment the provided examples by applying color maps, translations, rotations, reflections along y, x, and diagonal axes. We will also explore increasing the number of examples beyond this automatically for user-provided task instances.

3.3 Search Space Reduction

Many of these tasks can have their search spaces substantially reduced by considering that

- most tasks only modify input boxes of a single color (typically black)
- for some tasks we can view contiguous bits of the same color on input as whole objects/shapes
- in creating our DSL, we'll avoid functions with several arguments to prevent search space explosion

Before applying such space-saving reductions, we can consider smaller images for input-output pairs to validate the correctness of our approach.

4 Ideal Deliverables

4.1 Data Augmentation Pipeline

We shall create a process to generate examples using input examples and known invariants.

4.2 Domain Specific Language

We shall create a high-level DSL for specifying programs that solve these tasks.

4.3 Integration with Microsoft PROSE

We shall integrate our semantics of the operators/transformations of our DSL within Microsoft's PROSE tool[2].

4.4 Synthesized Solutions from Nontrivial Tasks in the ARC

We shall successfully synthesize programs in our DSL using PROSE for nontrivial task examples from the Abstraction Reasoning Challenge Corpus.

4.5 Stretch Goal: "Create-Your-Own Task" Tool

This tool would enable us to have students/users design their own task and provide limited examples of desired input/output.

4.6 Stretch Goal: User-Interactivity

This would consist of us prompting a user with questions like "would this transformation make sense?" for inputs they've provided.

4.7 Stretch Goal: Live Demonstration for Novel Task

This would mean anonymously asking a person in the class to prepare an example for our system to solve *LIVE*.

References

[1] François Chollet. On the measure of intelligence. *arXiv preprint arXiv:1911.01547*, 2019.

[2] Sumit Gulwani and Prateek Jain. Programming by examples: Pl meets ml. In *Asian Symposium on Programming Languages and Systems*, pages 3–20. Springer, 2017.