THE SYMBOLIC INNER CORE HYPOTHESIS FOR AGI

A PREPRINT

Robert Lizée
robert.lizee@gmail.com

November 26, 2023

ABSTRACT

Our research aims to use the ARC challenge to flesh out and test the symbolic inner core hypothesis. The idea is to solve problems relying on a symbolic inner core solver by abstracting a problem in a representation suitable for the symbolic inner core to analyze and solve it. The abstracting of the problem may rely on a partly neural architecture that could learn new representations, while the inner core solver is purely symbolic and can't learn. Thus, we want to investigate the many aspects of the solution, like the representation used by the inner core solver, the process of abstracting the problems, the process of finding new abstractions, and the process of solving the problems once abstracted.

1 Introduction

1.1 Motivation

The ARC Challenge [1] proposed by François Chollet is a great test bed to produce, flesh out, and validate hypotheses on AGI.

- It is not solved yet;
- It is not dependent on a vast amount of acquired knowledge;
- It seems to capture the essence of human reasoning.

Trying to resolve this challenge raises some questions:

- Is a neural network only approach sufficient?
- Is a symbolic component needed?
- How the two are related?
- Can the brain do symbolic computations?

Concerning the latter, in *the Neuro-Symbolic Brain* [4], we argue that neural networks can do symbolic computations in a limited fashion:

- With that framework, neural networks are slow at symbolic computations;
- However, it can leverage the neural architecture to compensate;
- Also, many symbolic processors could work in parallel, which is great for building a solver.

From that, we conclude that we should not restrict ourselves to strictly deep learning solutions to solve ARC from the perception that it is more closely in line with how the brain works.

Thus, our initial strategy to tackle the ARC Challenge was:

- To consider it as a programming process first;
- To abstract away the parts where we would need deep learning;

- To do generalization as we go using all the technics we know;
- To refine our solution as we try to solve more ARC tasks.

After the initial trials, we opt to solve ARC tasks in two stages repeated until a solution is found:

Stage 1 Use two decomposition methods to decompose the input and output grids, respectively, in a symbolic representation;

Stage 2 Use the symbolic inner core solver to discover the relationship between the input and output representations.

Where:

- The *decomposition methods* are obtained by the composition of primitive decomposition methods handed-coded initially and eventually learned;
- The *inner core solver* is hardcoded, not learned, symbolic, fast, and systematic but not thorough as it should observe relationships humans would.

We are still fleshing out the idea but present the current solution and its planned evolutions here.

1.2 Related Work

There have been various technics used to try to solve the ARC Challenge.

- Ours differs from neural network-based technics as centered around a symbolic solver where neural networks would play a role of providing heuristics for the search for the right decomposition method to use;
- It differs also from DSL-based program generation techniques;
- The solution proposed falls closer to the family object-centric technics like *Tackling the Abstraction and Reasoning Corpus (ARC) with Object-centric Models and the MDL Principle* [3] and [2]. However, in our case:
  - The decomposition methods are considered separately from the representation model;
  - The decomposition methods don't search for regularity between the samples; this job is for the solver;
  - The representation models are not necessarily object-centric;
  - the MDL principle does not guide item the search; we leave it as an investigation of what guides the search;
  - We focus on the characteristics of the solver.

2 Solution Proposed

We decompose the solution we are investigating for the ARC Challenge in the following tasks:

- Define a recursive **symbolic representation** of a pixel grid (in progress);
- Provide the **decomposition methods** used for each task of the training set (in progress);
- Design and implement the **symbolic inner core solver** using the decomposition provided (in progress);
- Design and implement the **solution finding process** using the solver;
- Design and implement the process to **suggest decomposition methods** for each given task (in progress);
- **Extend the solution** to cover more ARC tasks (in progress);

2.1 Symbolic Representation

We defined a symbolic representation that is still evolving, having the following characteristics:

- It provides a representation at a higher semantic level than the original pixel grid when available;
- It is defined recursively;
- It can reconstruct the grid;
- It exposes an API for the inner core solver, which is all it knows about the representation:

  - *number functions*;
  - *color functions*;
  - *grid functions*;
  - *sub images functions*; note that the inner core solver is not aware of the hierarchy of the symbolic representation, but it has access to its sub-objects through these sub-image functions.
- It has a *builder solver function* to build a function to build itself as a function of the input image, using the core solver to find functions to build each of its parameters as a function of the input image.

For example, we have:

BasicGrid A basic grid is essentially the end of the symbolic decomposition as it is the image itself. It does expose properties of the image, such as its width, height, whether it is symmetrical, the count for each color, etc... It also exposes the grid itself as a property.

MonochromeImage Exposes its color as a property plus all the properties of its child image, which only has the colors 0 and 1.

SubImages Exposes the method for combining its child's sub-images, which are symbolic images. It also exposes the array of sub-images to the solver through a sub-images function.

2.2 Decomposition Methods

We define various decomposition methods and decomposition method producers. The latter are functions that produce a decomposition method given an input, which often is a decomposition method used to decompose the inner part of the grid. The decomposition methods have the following characteristics:

- Given a grid image, it produces a symbolic representation;
- It might fail; in that case, the decomposition is just not possible;
- It is constructed using decomposition method producers and primitive decomposition methods;
- Images decomposed with the same decomposition method will have symbolic representations with the same structure; This aspect is crucial because it means that for images decomposed by the same decomposition method, the various functions exposed in the API for the solver will work for all of them.

For example, we have:

basicgrid It is a primitive decomposition method that produces a **BasicGrid** symbolic image.

monochrome It is a decomposition method producer that takes a decomposition method as argument; it computes the color of the image and then uses the decomposition method given as argument to compute the child's symbolic image with a monochrome version of the original image; finally, it creates a **MonochromeImage** symbolic image.

object list It is a decomposition method producer that takes a decomposition method as argument; it decomposes the original image in sub-images to which are applied the decomposition method given as argument; finally, it creates a **SubImages** symbolic image.

Note that we can have different decomposition methods that produce symbolic images with the same structure. For instance, we have various variations of the object list method depending on the rule it uses to separate the objects.

Further, note that we will need to extend this model because it assumes that the input and the output image are decomposable independently, which is not the case for all of the ARC tasks.

2.3 Symbolic Inner Core Solver

2.3.1 Design

This solver does the analysis. To do so, it uses the decomposition of the input and output images, which provides each image with an array of number functions, color functions, grid functions, and sub-images functions. Given an output function, it tries to reconstruct it using the function associated with the input images. Note that it leverages the fact that all the input images have the same symbolic structure and the same for the output images as provided by the decomposition methods used;

The solver operates on the following types when it makes its prediction:

boolean with boolean operations;

number with basic arithmetic and comparison operators;

color with equality;

grid with basic transforms and superposition.

As the project evolves, we might consider different categories of numbers to favor different types of operations for numbers that represent positions, sizes, counts, etc.

These design constraints for the solver are the following:

- It should be fast;
- It should represent the human analysis capacity;
- It should provide answers considered plausible by a human.

One hope is that by separating the inner core, we will eventually be able to define what it means for a solution to be considered as likely by a human;

In the future evolution of the solver, we want to integrate the notion of levels of analysis. The initial superficial analysis will serve to prune the unpromising decomposition methods to be used to solve a given ARC task. Even without solving the training ARC tasks, we can measure how close we are to a solution by counting the number of functions correctly predicted. For the non-predicted functions, the solver could give approximate functions that output a range of possible values instead of a single value; the narrower the range, the better the prediction.

2.3.2 Analysis done by the solver

To fulfill its mission, the solver, among other things, does the following analysis to determine if a given property of the output image can be computed as a function of the input image:

- Basic analysis;
- Table analysis;
- Object correspondance;
- Decision tree analysis;
- Mapping table analysis.

Basic Analysis First, test for simple input image functions to predict the output image's property.

- Use the output function, giving the property as the input function. Often, when the output image has the same structure as the input image, this function will work;
- Try a constant;
- Try one of the input image functions potentially scaled or translated if the type is a number;
- If the type is a grid, try basic image transformation of the grid or try superposing input grids on top of each other with different composition methods;

Table Analysis When the symbolic representations of input images contain sub-images, make a table of the properties of each sub-image:

- Compute the count of each property value;
- Compute the count of a combination of property values;
- Rank the property values;
- Rank the count.

With these, build functions to predict the output images' properties.

Object Correspondance If the symbolic representation of the input and output images contains sub-images, we want to predict one of the properties of the sub-images of output images.

- Find a correspondence between a subset of the sub-images of the input images and the sub-images of the output images comparing their properties;
- Find a function predicting which sub-image of the input images has a corresponding sub-image in the output images;

- Create a sub-solver to analyze from the perspective of the sub-images of the input images and the sub-images of the output images;
- Use it to predict the properties of the sub-images of the output images.
- This technique could be complemented by considering a subset of pairs of subimages of the input images corresponding to subimages of the output images.

Decision Tree Analysis Check if the properties of the output images could be produced with a decision tree using the properties of the input images.

Mapping Table Analysis Check if a mapping could be used to predict the output properties.

2.4 Solution finding process

Now, we describe the resolution process:

- Get an input and an output promising decomposition method for the input and the output images;
- Feed the solver the symbolic decomposition of the input image and the output image of the training samples;
- If needed, create a sub-solver with object correspondence to handle lists of objects in the output images;
- Using the recursive symbolic decomposition of the output images, find a function of the input image producing each of its parameters by querying the solver;
- Using these parameter functions, build a function producing the symbolic decomposition of the output image given the symbolic decomposition of the input image;
- If a function producing the symbolic decomposition of the output image cannot be found, repeat the process with other input and output decomposition methods;
- Otherwise, given the input image of a test sample:
  - Decompose it into its symbolic representation;
  - Use the function built to produce the symbolic representation of the output image;
  - Generate using the symbolic representation of the image.

2.5 Suggest Decomposition Methods

Assuming we've got an inner symbolic solver that can solve ARC tasks given the proper decomposition. Then, the problem of solving ARC reduces to searching for the proper decomposition methods. Here are some strategies we have in mind:

- Use of assembles of alternatives symbolic representation;
- Use the different levels of analysis of the symbolic inner core solver;
- Investigate how to refine decomposition methods gradually;
- Use of neural networks to predict the most promising decompositions.

2.5.1 Use of assembles of alternatives symbolic representation

The idea is to cover many compatible potential symbolic representations in a single step. To do so:

- We assemble the result of different decomposition methods in a single symbolic image containing all the alternatives computed;
- We prune the alternative decomposition methods that do not produce a symbolic image for every image;
- The symbolic image containing the alternative symbolic images uses the solver to try rebuilding any of its alternatives.

2.5.2 Use the different levels of analysis of the symbolic inner core solver

Use it to target the most promising symbolic decomposition for deeper analysis.

2.5.3 Investigate how to refine decomposition methods gradually

To optimize the search, it would be useful to tell we are getting closer, which a partial solution of the solver could provide. Also, if we could tweak or refine the decomposition methods used, which we want to investigate, then we could:

- Start with a fast superficial analysis to target decomposition methods with promising potential solutions;
- Refine the decomposition methods used;
- Augment the depth of the analysis of the inner core solver for the most promising decomposition methods.

2.5.4 Use of neural networks to predict the most promising decompositions

Finding the right decomposition problem seems similar to a recognition problem for which neural networks excel. We plan to investigate how to use neural networks to help find the proper decomposition method.

2.6 Extend the Solution

The solution as presented is too simplistic to cover all of ARC. Now, we present ideas on how to extend it.

- Learning to draw;
- Making abstractions;
- Learning representations;
- Extend the inner core.

2.6.1 Learning to Draw

Some ARC tasks require finding drawing rules. The output image is obtained from the input image by drawing using the discovered drawing rules.

- Even though this seems different from our current model, we can leverage the inner core solver by decomposing the input and output images in overlapping tiles;
- Each tile gives a window of attributes to the solver to do its analysis to discover which pixel should be drawn next.;
- As pixels are drawn, more tiles are added to the drawn pixel for the solver to refine its analysis;
- This proceeds until all the pixels that need to be drawn are drawn, or the solver fails to find better rules.

2.6.2 Making abstractions

Instead of using the solver to find the relationship between the input and output images, one could use the solver to find out what is common between the input and output images and create abstractions.

- In that case, the decomposition method marks an image, composed of sub-images, as an abstraction to be found;
- Then the solver looks at all the symbolic images marked as the same abstraction and searches for a formula to produce it given a small set of parameters;
- To do so, it integrates the symbolic sub-images composing the abstraction one at a time. It tries to predict its parameters with a formula given the previously added sub-images and adds it to the formula to build the abstraction. If it finds one, it adds it to the set of input parameters of the abstraction.

2.6.3 Learning Representations

We plan to combine the techniques for learning to draw and for making abstraction to build even more powerful abstractions.

These discovered abstractions could create new representations and decomposition methods.

This way, the decomposition methods are not merely obtained by the composition of primitive functions.

2.6.4 Extend the Inner Core

As we incorporate more ARC tasks, we want to investigate the minimal capabilities the inner core solver needs to solve all the ARC tasks.

3 Discussion

3.1 Results

Training Set By specifying which decomposition methods to use, we solve 134 of 400 problems in the training set in less than 10 seconds for all of them in a web browser on a portable computer. It is still early, and the goal is to have the inner core solver be able to solve all 400 of them.

Evaluation Set By just trying each of the decomposition methods used in the training set, we solve 40 of 400 problems of the evaluation set in less than 3 minutes for all of them in a web browser on a portable computer. It is still very early because we haven't implemented a reliable method to propose promising decomposition methods.

Private Set With the same technique as the evaluation set, we solved 1 of 100 problems. It is surprising because we thought these problems had the same distribution as the evaluation set. In any case, we need a more reliable way to propose decomposition methods.

3.2 Questions

While pursuing this research, we are hoping to answer these questions:

- Can the symbolic inner core be hardcoded or need to be learned?
- What are the capabilities of the symbolic inner core?
- How can we learn new representations and decomposition methods?
- Can the symbolic inner core idea account for all ARC tasks?
- What would be the role of Deep Learning: can it be restricted to a recognition role in finding the right decomposition, or should it play a bigger part?

3.3 Conclusion

We are satisfied with the speed of the inner core solver so far and can build around this idea to cover the rest of the ARC tasks. Then, the challenge will be to find a way to propose automatically promising decompositions to feed the solver. Finally, we must automatically learn new representations and decomposition methods to solve the remaining ARC tasks.

3.4 Code

The code for our ARC-Solver is available at https://github.com/robertlizee/arc-solver.

References

[1] François Chollet. *On the Measure of Intelligence*. 2019. arXiv: 1911.01547 [cs.AI].

[2] Sébastien Ferré. *First Steps of an Approach to the ARC Challenge based on Descriptive Grid Models and the Minimum Description Length Principle*. 2021. arXiv: 2112.00848 [cs.AI].

[3] Sébastien Ferré. *Tackling the Abstraction and Reasoning Corpus (ARC) with Object-centric Models and the MDL Principle*. 2023. arXiv: 2311.00545 [cs.AI].

[4] Robert Lizée. *The Neuro-Symbolic Brain*. 2022. DOI: 10.48550/ARXIV.2205.13440. URL: https://arxiv.org/abs/2205.13440.