Chair of Computer Science 6 (Machine Learning and Reasoning)

RWTH AACHEN UNIVERSITY

The present work was submitted to the Chair of Machine Learning and Reasoning.
Diese Arbeit wurde vorgelegt am Lehrstuhl für Maschinelles Lernen und Inferenz.

Solving ARC as a Generalized Planning Problem using ASP

Bachelor Thesis
Bachelorarbeit

Presented by / Vorgelegt von

Jan Dornhege
434697

Supervised by / Betreut von Prof. Hector Geffner, Ph.D.

1st Examiner / 1. Prüfer Prof. Hector Geffner, Ph.D.

2nd Examiner / 2. Prüfer Juniorprof. Dr. rer. nat. Christopher Morris

Aachen, January 26, 2025

Abstract

The Abstraction and Reasoning Corpus (ARC) is a benchmark for evaluating the general intelligence of artificial intelligence systems. A recent public competition with a high prize has significantly increased its popularity. It consists of a collection of tasks in which a system is provided with a very limited number of training examples and common patterns must be identified and applied to some test inputs. The tasks involve transformations of images with a maximum size of 30×30 pixels.

This thesis presents a novel approach for solving ARC tasks using Answer Set Programming (ASP). The method involves abstracting input images into a structured representation of objects and relations and applying a logic-based solver to find a sequence of actions that solves the task. The approach builds on the approach called Abstract Reasoning with Graph Abstractions (ARGA) proposed by Xu *et al.* [15]. Both the abstraction process and the formulation of transformations are adapted to better fit the ASP framework and with an aim to improve expressiveness.

The final system performed comparably well as ARGA, solving only a few more tasks. However, it does not yet surpass Generalized Planning for Abstract Reasoning (GPAR), an alternative approach that adapts ARGA proposed by Lei *et al.* [10].

While ASP shows some promise for ARC, it has not significantly advanced symbolic methods, which may never reach the levels comparable to top neural approaches, that currently drastically outperform them.

Contents

1 Introduction **1**

2 Background **3**
2.1 Core Knowledge . . . 3
2.2 Clingo . . . 4

3 Related Work **6**
3.1 Alternative Datasets . . . 6
3.2 Related Symbolic Approaches . . . 6
3.3 Neural Approaches . . . 8

4 Methods **10**
4.1 Formulation of the Approach . . . 10
4.1.1 Abstract States . . . 10
4.1.2 Transformation Definition . . . 14
4.1.3 Policy Application . . . 16
4.1.4 Solution Formulation . . . 17
4.2 Policy Search . . . 19
4.2.1 Pruning . . . 19
4.2.2 Clingo Encoding . . . 21
4.3 Examples . . . 23
4.3.1 Task 22eb0ac0 . . . 23
4.3.2 Task 42a50994 . . . 24

5 Results **26**
5.1 Evaluation on ARC subset . . . 26
5.2 Evaluation on MiniARC . . . 28
5.3 Further Analysis . . . 28

6 Conclusion **30**
6.1 How the Approach Implements Core Knowledge . . . 30

A Appendix **32**
A.1 Refinement Definitions . . . 32
A.2 Node Relations and Properties . . . 34

A.3 Abstract Solutions Are Direct Solutions 34

List of Acronyms **36**

List of Figures **37**

List of Tables **38**

List of Listings **39**

List of References **40**

1 Introduction

The Abstraction and Reasoning Corpus (ARC) was introduced by François Chollet in 2019 alongside the paper ”On the Measure of Intelligence” [3] as a benchmark to measure if a system is intelligent. ARC consists of 1000 tasks, each is handcrafted to incorporate concepts that are understandable by humans and can be solved by reasoning based on a set of prior knowledge called Core Knowledge. The concept of Core Knowledge is a theory from the field of psychology that states that every intelligent entity, like animals or humans, inherently possesses capabilities to reason based on knowledge from four broad categories: ”Objectness”, ”Agentness”, ”Natural numbers and simple arithmetic”, and ”elementary Geometry and Topology” [14]. These are formulated so that they can be applied to the format of the ARC tasks.

The 1000 tasks are split into 400 training tasks and 600 evaluation tasks. Two hundred evaluation tasks are not publicly accessible and are used to evaluate models in competitions. Each task consists of a few (on average 3.3) *demonstration examples* and usually one *test example*. Each of the examples consists of an input grid and an output grid. All grid sizes range from 1×1 to 30×30. The values in each cell range from 0 to 9 and are interpretable as colors in an image. The size of the output grid might differ from the size of the corresponding input grid. The system that tries to solve a task is given input and output images of the demonstration examples and the input image of the test example.

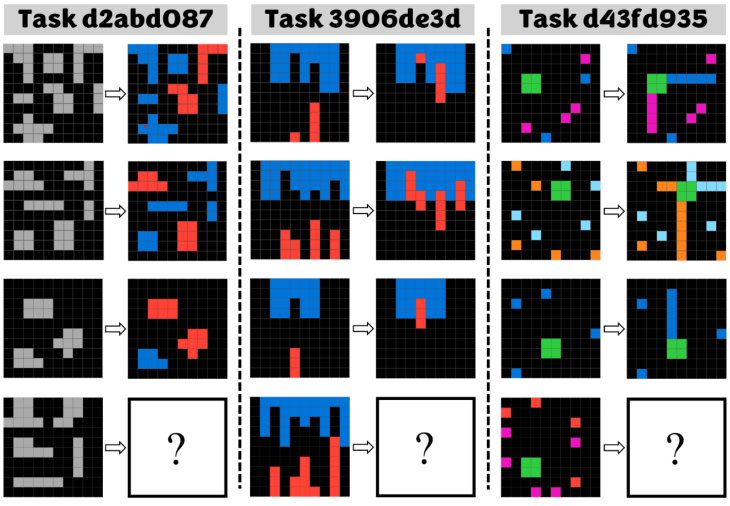


Figure 1.1: Image adapted from [15]. Three tasks from the ARC dataset. For each task, three demonstration examples and one test example.

Three of the ARC training tasks are depicted in figure Figure 1.1. In the first one, every gray connected component is recolored to red if it consists of exactly 6 pixels and blue otherwise. In the second task, the red vertical stripes are moved to the top until they collide with a blue pixel. In the third one, lines are created between the unique 2×2 green square and non-black isolated pixels that lie horizontally or vertically to the green square. The lines are colored according to the non-black pixel they connect to.

The following chapters provide background on Core Knowledge and the Clingo solver, an overview of related work, a detailed description of the proposed approach and an experimental evaluation. Finally, a conclusion summarizes the results and gives an outlook for extending the approach.

2 Background

This chapter provides a more detailed introduction to Core Knowledge and the Answer Set Programming (ASP) solver Clingo.

2.1 Core Knowledge

Historically, two predominant views have emerged in cognitive science regarding the operation of the brain. The first view proposes that the brain recognizes regularities in the environment and adapts to new situations based on past experiences. The second one, in contrast, describes the brain as a collection of fixed, specialized mechanisms. Core Knowledge proposes that evolution has developed a small set of distinct systems (called Core Knowledge) in the brain and new skills can be learned using these as a foundation [14].

Studies on human infants and animals found that all of them possess abilities in four broad knowledge categories:

- Objectness,
- Agentness,
- Natural numbers and simple arithmetic,
- Elementary Geometry and Topology.

More recently, evidence for a fifth category that centers around social interactions has been proposed, but it has not been considered in the design of ARC.

Objectness, in general, refers to the ability to reason about objects. It can be further categorized into three domains: object ”cohesion”, ”continuity”, and ”contact”. Object cohesion is the ability to recognize that objects are bounded and connected such that an object moves as one entity. In ARC, object cohesion is formulated as recognizing objects through spatial or color contiguity, parsing grids into zones, and partitioning them [3]. Object continuity means that objects move on continuous, connected and unobstructed paths. In ARC, this is formulated as object persistence. It describes how objects persist even if they are obstructed by other objects or noise. Object contact describes that objects may interact in some way if they are in contact with each other. For example, an object may move in some direction until it is in contact with another object and then stop.

Research has shown that the number of objects considered in reasoning is limited. This limit is about 3, for human infants, and for monkeys, it is 4. However, this specific result has not been explicitly considered in the formulation of ARC.

The Agentness category describes that agents may intentionally perform actions to efficiently reach some goal. It is not explicitly incorporated in ARC, but in some tasks, it may be required to apply a sequence of actions that lead to a goal.

It has not been fully understood how the category Natural numbers and simple arithmetic is implemented in nature. Observations suggest that number representations follow three properties. The First, number representations are imprecise and become more imprecise for larger numbers. Secondly, number representations are abstract, meaning that representations are not tied to specific objects and are universally applicable. The third property is that numbers can be compared, and addition and subtraction operations can be performed. Comparing, sorting, additions, and subtractions are needed to solve the ARC tasksand tasks are designed to be solvable by reasoning about natural numbers up to 10.

Elementary Geometry and Topology describes the ability to recognize distances, angles, and other geometric properties while unable to represent non-geometric properties, like colors. Many geometrical properties are featured in ARC:

- Lines, rectangular shapes.
- Symmetries, rotations, translations.
- Shape upscaling or downscaling, elastic distortions.
- Containing, being contained, being inside or outside of a perimeter.
- Drawing lines, connecting points, orthogonal projections.
- Copying, repeating objects.

By providing only a few demonstration examples per task and a large space of possible transformations, the design of ARC requires the solving system to explicitly or implicitly use these Core Knowledge systems.

Implementing the Core Knowledge systems was not the primary focus when developing the approach used in this thesis. However, many parts of the approach directly implement specific properties of the Core Knowledge, while some aspects formulated in the Core Knowledge are missing from the approach and seem difficult to implement comprehensively.

2.2 Clingo

ASP describes a methodology for solving NP-hard problems that can be formulated as logic programs. A logic program is a set of rules and facts expressed in a formal language representing problems and solutions in ASP.

Clingo is part of the Potsdam answer set solving collection (Potassco) [6]. It combines the grounder *Gringo* and the solver *Clasp*. *Gringo* is the grounder, which converts the high-level program into a propositional format. *Clasp* is the solver that computes the answer sets based on the propositional problem obtained from calling Gringo. Grounding a logic program refers to the

process of instantiating variables in the logic program with atoms, resulting in a propositional formula. The support of simple integer arithmetic in Clingo was particularly useful in this work's application. In contrast, a pure SAT encoding would be more complex.

ASPs are solved by finding so-called *stable models* (or answer sets) of a given program. As a model of the program, the *stable model* is consistent and satisfies all rules and facts given in the program. Additionally, it is required that the satisfaction of a predicate is either due to a fact stating the truth of this predicate or the predicate can be derived by a rule. The reasoning is not allowed to include any circularity, meaning that the truth of predicates is ultimately based on the initial facts.

A Clingo program comprises *facts*, *rules*, and *constraints* [5]. Facts must be true in any model and are represented as ground literals (literals without variables). Rules are of the form

$$A_0 :\text{-} B_0, \ldots B_n.$$

The rule states that if a model satisfies all *body literals* $B_0, \ldots, B_n$, it must also satisfy the *head literal* A_0. Constraints are written as

$$:\text{-} B_0, \ldots, B_n.$$

and prohibit any model from satisfying all $B_0, \ldots, B_n$ simultaneously.

Beyond computing valid solutions, Clingo supports *optimization*. It allows users to specify preferences for certain solutions by assigning costs or rewards to the satisfaction of specific predicates. This feature makes it possible to identify not only feasible solutions but also the most desirable ones according to a defined metric.

3 Related Work

A study found that humans can successfully solve approximately 84% of the ARC tasks [7]. Exceeding the state-of-the-art of artificial systems. The current leading system is the winner of the ArcPrize 2024 competition [4], which solved 55% of the tasks of the private evaluation set. The ArcPrize was a competition hosted on Kaggle.com, held from June to November 2024. It was the third competition on Kaggle with the goal of solving the tasks in the ARC. Before the competition, the highest achieved score was 33%.

3.1 Alternative Datasets

To simplify solving the ARC tasks, some additional ARC-inspired benchmarks have been introduced [8, 12, 15, 16]. The approach is evaluated on two of them in Chapter 5.

Xu *et al.* [15] classified a set of 160 tasks in the training set of ARC to follow an object-centric theme. They further categorized these 160 tasks into three categories: *recolor*, *movement*, and *augmentation*. The width and height of the images do not change from input to output in all of those tasks. The restriction to the subset simplifies the problem of solving a task since the width and height of the output image do not need to be predicted and all tasks have an object-centric theme. The focus of this work was this subset; hence, the algorithm is incapable of predicting width and height.

The second benchmark on which the approach is evaluated is the MiniARC [9]. Its tasks are restricted so that all images have dimensions 5x5. This reduces computational complexity and allows testing more resource-demanding approaches.

3.2 Related Symbolic Approaches

The presented approach is very closely related to Abstract Reasoning with Graph Abstractions (ARGA) by Xu *et al.* [15]. They use a similar way of abstracting the images into an abstract representation. Then, they search for policies that solve the tasks.

The authors use some so-called *graph abstractions*, to abstract the images into a representation as a graph. These extract objects from the image, which then correspond to nodes in a graph. Edges in the graph correspond to relations between these nodes. They independently apply their solver to the different graphs from different *graph abstractions*.

Their actions to transform the images can modify individual objects by 11 possible transformations. The parameters for the transformation can be dynamically derived from the nodes and their relation to other nodes. Some of the transformations are *updateColor(Node, Color)*, *move(Node, Direction)*, *rotate(Node)*, *mirror(Node, Pixel, Direction)*, *extend(Node, Direction)*, and *flip(Node, Direction)*.

To select the objects to which these transformations are applied, they define $filters$. A $filter$ is a formula in a subset of first-order logic. It can refer to the relations of a node to other nodes. Filters are recursively constructed by following recursive definition:

$$
\begin{aligned}
Filter(x) &::= Type(x) \\
&::= Filter(x) \wedge Filter(x) \\
&::= Filter(x) \vee Filter(x) \\
&::= \neg Filter(x) \\
&::= \exists y Rel(x, y) \wedge Filter(y) \\
&::= \exists y Rel(y, x) \vee Filter(y) \\
&::= \forall y Rel(x, y) \Rightarrow Filter(y) \\
&::= \forall y Rel(y, x) \Rightarrow Filter(y) \\
&::= Rel(x, c) \text{where } c \text{ is a constant} \\
&::= Rel(c, x) \text{where } c \text{ is a constant}
\end{aligned}
$$

An action consists of a $filter$ and a $transformation$. When applying an action, the transformation is applied to all nodes that satisfy the formula of the $filter$.

Finding filters and corresponding transformations involved searching over the possible filters, combined with some pruning and a heuristic that rewards correctly colored pixels in the output image.

A similar approach is taken by Lei *et al.* [10] in an approach called Generalized Planning for Abstract Reasoning (GPAR). They utilize an abstraction method similar to the ones used in ARGA to derive abstract objects but proceed by defining a domain-specific language that uses pointers to select nodes or other objects and *goto* statements that control the program flow. This language defines general policies or programs that solve the task.

Both ARGA and GPAR were evaluated on the subset of 160 ARC tasks introduced in Section 3.1. This work mainly focuses on solving the tasks of this dataset. ARGA learned 63/160 transformations that correctly transform the demonstration examples of a task, 57 of those also

solve the test examples of these tasks. GPAR was able to solve 86/160 of the demonstration examples, 81 of these also solved the test examples.

Ainooson *et al.* [2] defined image transformations like *scale_2x* that scales an object by two or *change_color* that changes the color of a node. Then, their approach searches for a sequence of these actions such that they solve a task. The actions directly modify pixel-based images instead of abstract representations. This approach was able to solve 111 of the 400 training tasks.

3.3 Neural Approaches

All approaches that performed well in the ArcPrize competition relied on neural networks somehow.

Li *et al.* [11] describe two general ways to successfully use neural networks for the ARC. Transduction describes the model directly predicting the output images based on the demonstration examples and an input image. This approach is often augmented by test time training (TTT), where the neural network is trained on a lot of training data. Then, when solving a task, the model is fine-tuned based on the given task. All of the top transductive models used test time training.

On the other hand, solving a task by Induction involves inferring a program or function in some language that solves all of the given demonstration examples of a given task. The language can be a special domain-specific language; others use general programming languages like Python. Then, this program is applied to the test input image. The neural model can either infer this program directly or guide a search over the programs [4]. Some specialized code Large Language Models (LLM) were utilized for inductive models.

It has been observed that the tasks that transductive models solve differ greatly from those solved by inductive models, making ensembles between both approaches most successful.

One commonly used method is data augmentation, where additional tasks are invented by some program or created by hand. This allows the neural network to be trained on a lot more data. For example, the BARC systems were trained using a corpus of 400 000 additional tasks [11].

Directly prompting non-specialized LLM in a transductive manner did solve some of the tasks, but it did not perform as well as fine-tuned models.

Xu *et al.* [16] tried some methods to use LLMs. An overview of their approach is given in Figure 3.1. Their approach consists of encoding the task into a textual encoding. This is done by directly encoding pixels of the images or deriving an abstract graph using the method from the ARGA approach [15] and embedding this graph into text. Then, this is combined with

demonstration examples and a detailed description of what the GPT is expected to do. The answer from the GPT is parsed into an image. To improve this, they used prompting methods like Chain of Thought. Their best method can solve 97 out of the 400 training tasks.

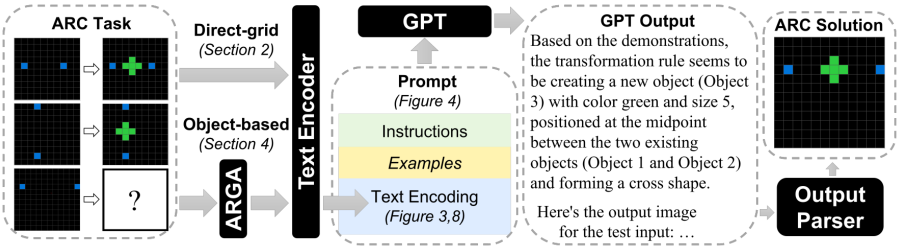


Figure 3.1: Image adapted from Xu *et al.* [16]; Approach to use GPT to learn ARC tasks.

4 Methods

The approach used in this work is introduced in this Chapter. First, a general overview of the approach is given. Then, a detailed mathematical formulation is introduced. Following this, the process of finding a solution to a task using Clingo is described.

4.1 Formulation of the Approach

ARC tasks are solved by finding an action sequence that transforms an abstract representation of the input image. Abstract representations of images are referred to as *abstract states*. An abstract state represents an image as a set of nodes. There are different ways to obtain such an abstract state from an image. Based on ARGA [15] and GPAR [10], which utilize a similar approach of finding transformations on abstract image representations, this work uses 23 predefined *abstraction functions*. Abstraction functions describe how an image is abstracted into an abstract state by forming nodes based on a partition of the pixels.

The actions modify *abstract states* by transforming individual nodes by selecting nodes of on abstract state that satisfy some preconditions, which are given as first-order logic formulas. These first-order logic formulas are evaluated based on the properties of the nodes and relations between them. An action can delete or modify the selected nodes or create new nodes.

A solution to a task is a pair (f, A), where $f \in \mathcal{A}$ is an abstraction function and A is a policy, given as a tuple of actions.

4.1.1 Abstract States

In the following, the abstract states, abstraction functions and the process of abstracting an image are defined. Also the formulation of actions and policies is presented.

In this work, an *image* refers to a matrix with values in $\{0, ..., 9\}$, while a *pattern* is a matrix with values in $\{-1, 0, \dots, 9\}$, where -1 encodes a transparent cell. In accordance with the task formulation of ARC, both images and patterns have dimensions between 1×1 and 30×30. Let $P_{all} = \{\{-1, 0, \dots, 9\}^{i \times j} : 1 \leq i \leq 30, 1 \leq j \leq 30\}$ be the set of all possible patterns and I_{all} the set of all possible images analogously. The pixels of an image or pattern M are all triplets $(x, y, c) \in pixels(M) = \{(x, y, c) : M_{x,y} = c\}$, where the image or pattern M has value c in the cell with coordinates (x, y).

A node n is a tuple with the following entries $n = \left(n_x, n_y, n_{rot}, n_{mirr}, n_{colors}, n_{shape}, n_{mod}\right)$, such that

- $n_x \in \{1, ..., 30\}$
- $n_y \in \{1, ..., 30\}$
- $n_{rot} \in \{0°, 90°, 180°, 270°\}$
- $n_{mirr} \in \{0, 1\}$
- $n_{shape} \in P_{all}$
- $n_{colors} \in \{0, ..., 9\}^9$
- $n_{mod} \in \{0, 1\}$.

The x and y properties indicate the object's position in the image. *rot*, *mirr*, and *colors* specify how to embed the *shape* in an image. *mod* indicates whether a node has been modified by some action and is used for defining policies.

An *abstract state* represents an image as a set of nodes. It is a triplet (N, w, h), where $N = \{n_1, \dots, n_m\}$ is a set of nodes and $w, h \in \mathbb{N}$ refer to the width and height of the corresponding image, respectively.

For an abstract state $S = (\{n_1, \dots, n_m\}, w, h)$ let

$$shapes(S) := \{n_{i,shape} : 1 \leq i \leq m\}$$

be the set of all used shapes.

Abstraction Functions

For the set of all possible pixels $M = \{(x, y, c) : x, y \in \{1, \dots, 30\}, c \in \{0, \dots, 9\}\}$ any mapping $r : \mathcal{P}(\mathcal{P}(M)) \to \mathcal{P}(\mathcal{P}(M))$, such that for all $P_0 \in \mathcal{P}(\mathcal{P}(M))$ and $P_1 := r(P_0)$, the elements in P_1 are pairwise disjunct and

$$\forall p_1 \in P_1, \exists p_0 \in P_0 : p_1 \subseteq p_0$$

holds are called refinement operators. This states that for each element in the output, there has to be one elements in the input, such that it is a superset. If the function is given a set in of $\mathcal{P}(\mathcal{P}(M))$, it can only split up its elements or alternatively delete some. But it can not add new ones or combine existing ones.

Elements in $\mathcal{P}(\mathcal{P}(M))$ are sets that contain subsets of pixels. Every partition of pixels of an image is an element of $\mathcal{P}(\mathcal{P}(M))$. In the usage, partitions of all (sometimes only non-black) pixels of a given image are considered. The following basic refinement operators have been used:

- black
- connect8
- connect4

- color
- x
- y
- *remove_black*
- remove_empty

Formal definitions of these refinement operators are provided in Chapter A. For example, the *color* refinement operator refines a partition by grouping all the same colored pixels, which have been in the same element. Since the color black (=0) is often used as a background color, some of the *abstraction functions* give a partition of the non-black pixels by applying *remove_black*, eliminating all black pixels. If the partition contains the empty set, remove_empty removes it, as it should not form a node in the abstract state.

These basic refinement operators were concatenated in different ways to form the 23 *abstraction functions*; see the full list and definitions in Chapter A. Let $\mathcal{A}$ be the set of all those *abstraction functions*. Because any concatenation of refinement operators is also a refinement operator, abstraction functions are refinement operators. For example
$abstract\_4connected := remove\_empty \circ connect4 \circ color$ groups all pixels that have the same color and are connected.

Nodes where the shapes only differ in rotation, mirroring, or color replacement should be threated the same by the actions. In the first step, the elements of the partition are converted into so-called *normalized patterns*. A normalized pattern for a set of pixels p is denoted as $\tilde{p}$. The pixels in p are placed in a matrix, where -1 is placed in cells that are not mentioned in p. The empty (all -1) rows and columns at the borders are removed such that the colored pixels touch all four edges.

All the colors for all pixels of the most often occurring color are replaced by 0, the colors of the second most occurring color are replaced by 1, and so on. If two colors occur equally often, an arbitrary order is chosen. Patterns are cropped to make shapes comparable and recoloring is done such that the actions can reference or change the colors according to their frequency.

Figure 4.1 shows an example of how a normalized pattern $\tilde{p}$ is obtained from p.

Abstracting a Single Image

For an image $I \in \{1, \dots, 9\}^{w \times h}$, the abstract state that is obtained by using an abstraction function $f \in \mathcal{A}$ is denoted as $abstract_{f,P}(I)$, where $P \subset P_{all}$ is a set of patterns. P enables the reuse of shapes observed in other images of the same task. Converting an abstract state S into an image does not depend on the function and is denoted as $render(S)$.

The first step to obtain $abstract_{f,P}(I)$, is to use f to get a partititon $f(pixels(I)) = \{p_1, ..., p_l\}$ of the pixels in I. S_I is created incrementally by starting with an abstract state that does not yet

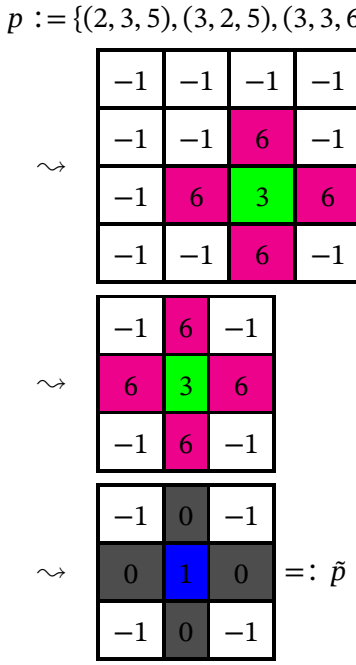


Figure 4.1: Illustration of normalizing a pattern.

contain any nodes. For each $p_i \in \{p_1, ..., p_l\}$, a new node n_i is added. Let $S_0, S_1, \dots, S_l$ be the intermediate *abstract states* with $S_0 := (\varnothing, w, h)$ and $S_l =: abstract_{f,P}(I)$.

Then for $i \in \{1, \dots, l\}$, let $S_i = (N_i, w, h)$, such that $N_i := N_{i-1} \cup \{n_i\}$, where n_i is a node.

The x and y properties of n_i are determined by finding the pixels with the least x or y coordinate, respectively.

Then, the algorithm searches if there is a pattern $s \in shapes(N_{i-1}) \cup P$ such that it is congruent to the normalized pattern $\tilde{p}_i$. Congruency here refers to being able to transform one pattern into the other one by remapping the colors, rotating, and, mirroring along the horizontal axis. If such a s is found, the values for *rot*,*mirr colors*, and *shape* can be determined by finding a transformation of mirroring, rotating, and remapping of colors, that map $\tilde{p}_i$ to s. The *shape* property of n_i is set to s.

If no such pattern s exists, *rot* and *mirr* are set to zero and $shape_p := \tilde{p}_i$. *colors* are chosen to match the remapping in the process of obtaining $\tilde{p}_i$. So the $colors_0$ property is set to the most frequently occurring color in p_i, $colors_1$ to the second most frequent color, etc.

Finding nodes with congruent shapes is done to allow actions to reference specific shapes independent of rotation, mirroring, and colors. It also allows the reuse of shapes that have been observed in the demonstration examples. Neumann and Pintér [13] used a similar approach.

For example, in task *0962bcdd* (see Figure 4.2), the *abstraction function* abstract_8connected_multicolor_bg_rm, formes nodes by finding all non-black pixels that are connected and removing the black background. Actions may replace the shape property of all nodes with the shape that appears in the output nodes. Since the output shape has been observed in the demonstration examples, it can be correctly predicted in the test examples.

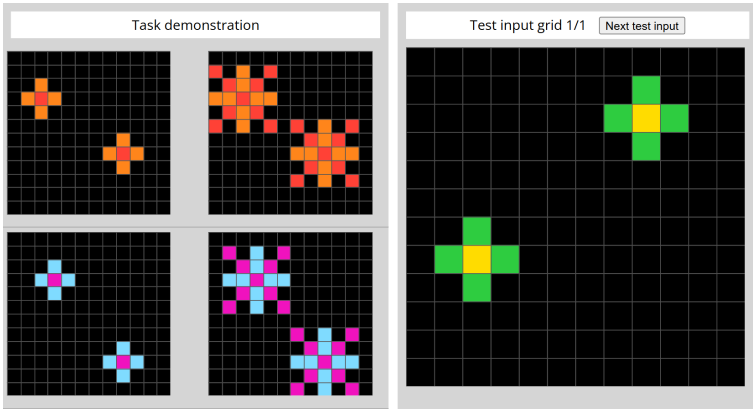


Figure 4.2: Task 0962bcdd; The shape of each multicolor-connected object is changed.

The converse process of obtaining an *image* from an *abstract state* is called rendering and is denoted as $render(S)$ for an abstract stats S. The exact process is not explained in detail here, but in general, the shape of each node is rotated, mirrored, and colors are remapped according to the properties *mirr*, *rot*, and *colors* of the node. The resulting pattern is added to the $w \times h$, initially all-black, matrix at the position indicated by the x and y properties of the node while ignoring -1 values.

In general, the rendering process is not deterministic; it is unclear which node to show if two nodes overlap. However, abstract states $abstract_{f,P}(I)$ that have been created by the process previously described do not contain any overlapping nodes. Therefore, it is deterministic. This is because the used *abstraction functions* create partitions of the pixels or non-black pixels and individual nodes are rendered to exactly the pixels in the corresponding element of the partition. For all abstraction functions $f \in \mathcal{A}$, all images I, and all sets of patterns P

$$render\big(abstract_{f,P}(I)\big) = I \tag{4.1}$$

holds, while in general, for an abstract state S, $abstract_{f,P}(render(S))$ is not equal to S.

4.1.2 Transformation Definition

A transformation is a tuple of actions $A = \big(a_i, \dots a_{n_A}\big)$, where a_i for $i \in \{0, \dots, n_A\}$ is an action and n_A is a parameter for the number of allowed actions. Actions map *abstract states* into other *abstract states*.

Some building blocks need to be introduced to define actions.

Value bindings are mappings, that for a given abstract state $S = (N, w, h)$, derive a natural number for each node $n \in N$, $VB : N \to \mathbb{N}$.

A recursive definition of these mappings is given in the following:

$$\begin{aligned} VB(n) &::= c \text{ for } c \in \{1, \dots, 10\} \\ &::= n_p \text{ for } p \in \{x, y, rot, mirr, shape, colors_0, \dots, colors_9\} \\ &::= VB'(n) + VB''(n) \\ &::= VB'(n) - VB''(n) \\ &::= VB'(x) \bmod 2 \\ &::= VB'(x) \bmod 3 \\ &::= |N| \\ &::= w \text{ (width of the state)} \\ &::= h \text{ (height of the state)} \\ &::= VB'(k) \text{ for } r(n, k) \text{ for a relation } r \in R_{VB} \\ &::= A(n) \text{ for a node-attribute } A \in A_{VB} \end{aligned} \tag{4.2}$$

All patterns in $shapes(S)$ are enumerated, such that each shape is assigned a unique natural number. $VB(n) := n_{shape}$ evaluates to that unique number.

Node attributes $A \in A_{VB}$ that occur in the last rule allow to express additional information, such as the number of pixels in the shape of a node or the number of neighbors, for a full list see Section A.2. R_{VB} is a set of precalculated $functional$-relations between nodes of the same image. Functional relations exhibit the property that for each $n \in N$, there is exactly one $m \in N$ such that $R_{VB}(n, m)$ holds. This allows to reference the information of certain other nodes in the state. For example, R_{VB} contains a relation that relates each node to the biggest node in the state and a relation that relates each node to its closest neighbor. A list of all used relations is also listed in Section A.2.

The next building block of actions are $filters$. They are first-order formulas that use value bindings to reference nodes' properties and express equalities and inequalities. Given an abstract state $S = (N, w, h)$, a $filter$ is used to select some nodes based on their properties. A $filter\ F$ is a recursively defined mapping $F : N \to \{true, false\}$.

$$
\begin{aligned}
F(n) &::= true \\
&::= false \\
&::= \neg F'(n) \\
&::= F'(n) \wedge F''(n) \\
&::= F'(n) \vee F''(n) \\
&::= VB_1(n) = VB_2(n) \\
&::= VB_1(n) < VB_2(n) \\
&::= P(n) \text{ for a node-property } P \in P_F
\end{aligned}
\tag{4.3}
$$

The $P \in P_F$ are node properties. One example of a node property is $sym_{hor}(n)$, which indicates if the shape of a node is horizontally symmetric. See Section A.2 for a full list of all used node-properties P_F.

VB_1 and VB_2, which occur in the last two rules, are value bindings as previously defined.

In the implementation, parameters restrict maximal recursion depth of filters and value bindings.

Given this, an action $a = (F_a, V_a, n_a)$ consists of a filter F_a, an integer $0 \leq n_a \leq n_C$, and a set of value bindings $V_a = \{V_{a,j,p} : j \in \{1 \dots n_a\}, p \in \{x, y, rot, mirr, shape, color_0, ..., color_9\}\}$. n_C is a parameter indicating the maximal number an action can copy a node.

If $n_a = 0$, this means that all selected nodes are deleted. For $n_a = 1$, this corresponds to modifying all selected nodes. If $n_a > 1$, additional nodes are created with values based on the properties of the origin node. $V_{a,j,p}$ is a value binding that derives the property p of the j-th created node by action a.

4.1.3 Policy Application

As previously defined, a solution $S = (f, A)$ is given as an abstraction function f and a tuple of actions $A = (a_1, ..., a_{n_A})$. The number of actions n_A is a parameter. Some actions may be noop-actions, meaning they do not have any effect. They can be implemented by an action with an unsatisfiable filter (for example, $F_a := false$). The solution is applied on an *image* by first obtaining an abstract state $S_0 = abstract_{f,P}(I)$ following the process described in Section 4.1.1. Then, the actions in A are applied sequentially to derive the output state S_o.

$$
abstract_{f,P}(I) = S_0 \rightarrow_{a_1} S_1 \rightarrow_{a_2} \cdots \rightarrow_{a_{n_A}} S_{n_A} = S_o
$$

where $S \rightarrow_a S'$ indicates the application of action a to S, resulting in S'. $S \rightarrow_A S'$ indicates that S' results from applying the actions in A sequentially.

An action $a = (F_a, V_a)$ is applied by first selecting all nodes n in an abstract state $S = (N, P, w, h)$, that satisfy its filter F_a and that have not been modified by any other action before.

$$select_a(S) := \{n \in N : F_a(n) = true \text{ and } mod_n = 0\}$$

The action then transforms an abstract state $S = (N, w, h)$ into the abstract state $S' = (N', w, h)$ ($S \rightarrow_a S'$) in the following way:

$$N' = (N \backslash select_a(S)) \cup \bigcup_{i=0 \ldots n_a} \{a^i(n) : n \in select_a(S)\}$$

where $a^i(n) := \left(V_{a,i,x}(n), V_{a,i,y}(n), \ldots, V_{a,i,color_0}(n), \ldots, V_{a,i,color_9}(n), 1\right)$ is a node, where the properties are derived by evaluating the value bindings given by the action. The action may modify only very few properties by having value bindings like $V_{a,i,p} := n_p$ that do not change the node properties.

The restriction to only allow a node to be modified once was made to decrease the size of the Clingo encoding that is introduced in Section 4.2. Otherwise, all possible intermediate node configurations would need to be encoded. The w and h values of the state do not change during action application. This is unnecessary because in all tasks in the subset selected by Xu *et al.* [15], the width and height of the input images do not change from input to output images.

4.1.4 Solution Formulation

For a task, image transitions $D_1 \hookrightarrow \overline{D}_1, \ldots, D_k \hookrightarrow \overline{D}_k$ of the demonstration examples and the input images $T_1, \ldots T_l$ of the test examples are given. The algorithm needs to correctly predict the output images $\overline{T}_1, \ldots \overline{T}_l$ of the test examples. This is approached by finding a *solution* that solves all $D_i \hookrightarrow \overline{D}_i$.

Let (f, A) be a pair of *abstraction function* f and action tuple A.

For each given image, an abstract state is created in a specific order, such that each abstract state reuses as many of the previously used shapes as possible.

$$\begin{aligned}
d_j &:= abstract_{f,P_{j-1}}\left(D_j\right) && \text{for } j \in \{1, \ldots, k\} \\
\overline{d}_j &:= abstract_{f,P_{j-1} \cup shapes\left(d_j\right)}\left(\overline{D_j}\right) && \text{for } j \in \{1, \ldots, k\} \\
\text{with } P_j &:= \bigcup_{i=1}^{j}\left(shapes(d_i) \cup shapes\left(\overline{d}_i\right)\right) && \text{for } j \in \{1, \ldots, k\}
\end{aligned}$$

Here, the P_j-s are the set of all shapes that have been created by abstracting the images until example j. The d_j is the abstract state for the input image D_j. $\overline{d}_j$ is an abstract state of the output image $\overline{D}_j$.

$P_k := \bigcup_{i=1}^{k}\left(shapes(d_i) \cup shapes\left(\overline{d}_i\right)\right)$ includes all patterns observed in the demonstration examples.

Test input images are abstracted to use shapes observed in any of the images in the demonstration examples or in a previous test image.

$$t_j := abstract_{f,P_k \cup Q_{j-1}}\left(T_j\right) \qquad \text{for } j \in \{1, \dots, l\}$$
$$\text{with } Q_j := \bigcup_{i=1}^{j} shapes(t_i) \qquad \text{for } j \in \{1, \dots, l\}$$

And $Q_l := \bigcup_{t=1}^{l}\left(shapes(t_i) \cup shapes\left(\overline{d}_t\right)\right)$

For the evaluation of a solution, the output test images are also abstracted:

$$\overline{t_j} := abstract_{f,Q_l \cup P_k}\left(\overline{T}_j\right) \quad \text{for } j \in \{1, \dots, l\}$$

There are two formulations- *abstract* and *direct* - to decide if (f, A) solves an image transition $D \hookrightarrow \overline{D}$. These are visualized in Figure 4.3. In the abstract formulation, the input image D is abstracted to d, then the actions in A are applied on d, resulting in $\overline{d}$. The task is then solved *abstractly* if $\overline{d}$ is the same abstract state as the abstract generated by abstracting the output image. In the *direct* formulation, the input is also abstracted and then the actions are applied to derive an abstract state $\overline{d}$. But then $\overline{d}$ is rendered into an image. If the resulting image is the output image, the task is solved.

(f, A) solving a transition *abstractly* is denoted by $(f, A) \vDash D \hookrightarrow \overline{D}$, while the *direct* solution formulation is denoted as $(f, A) \vDash_d D \hookrightarrow \overline{D}$. Formally, these conditions can be written as

$$(f, A) \vDash D \hookrightarrow \overline{D} :\Leftrightarrow abstract_f(D) \rightarrow_A abstract_f\left(\overline{D}\right),$$
$$(f, A) \vDash_d D \hookrightarrow \overline{D} :\Leftrightarrow abstract_f(D) \rightarrow_A S \text{ and } render(S) = \overline{D}.$$

Where $abstract_f(I) := abstract_{f,P}(I)$, such that the set of patterns P is chosen according to the previously described order in which abstract states are generated for a task.

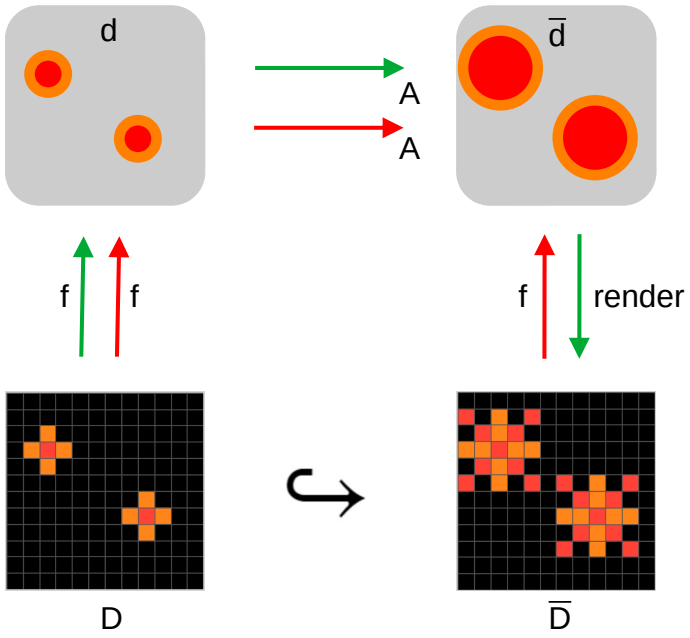


Figure 4.3: Illustration steps performed to evaluate abstract (red) and direct (green) formulation. (This is ARC-task 0962bcdd)

The *abstract* solution formulation is more strict than the *direct* one. This means that every *abstract* solution is also a *direct* solution. The other direction does not hold; there are *direct* solutions that are not *abstract* solutions. This follows from Equation (4.1). A proof sketch is given in Section A.3.

Any pair (f, A) *learns* a task *abstractly* or *directly*, if it solves all demonstration image transitions *abstractly* or *directly*. And a pair (f, A) *solves* a task if it learns the task and the test images are transformed into the correct outputs.

4.2 Policy Search

A solution pair (f, A) is found by evaluating for each abstraction functions $f \in \mathcal{A}$ individually if a solution exists. The problem of finding a policy for a given abstraction function is encoded as a Clingo program. Clingo is called to evaluate if a policy exists and find one. The optimization directives in Clingo are used to search for the ”simplest” policy according to some directives.

4.2.1 Pruning

Not all abstraction functions were evaluated on every task. To reduce runtime, states with a lot of nodes were not considered. Additionally, some conditions allow to exclude some of the abstraction functions because there is no possible policy to solve the task.

Table 4.1 shows which pruning criterion was used for which solution formulation and the number of task abstraction function pairs it eliminated.

Criterion	Pruned in direct	Pruned in abstract	Excluded pairs
Duplicate abstract state	Yes	Yes	756
Number of Nodes	Yes	Yes	392
New shape	Yes	No	1886
Too many output nodes(1)	Yes	No	1279
Too many output nodes(2)	Yes	No	339

Table 4.1: Number of task-abstraction function pairs pruned from the 3280 possible ones by pruning criterion. *Too many output nodes(i)* depends on the parameter $n_C = i$ that limits the maximal number of copy steps allowed. One task-abstraction function pair may be excluded by multiple critera.

All abstraction functions that created more than 200 nodes in all abstract states of the demonstration examples combined were pruned (*Number of Nodes*). Also, if two abstraction functions produced the same partition or abstract state, only one was used for the task (*Duplicate abstract state*).

If the *abstract* solution method is used and the abstraction function generates a node in one of the output images of a test examples, that has a shape that did not occur in any of the other images. No action can generate this shape and the program prunes this abstraction function. Expressed in the notation of Section 4.1.4, the condition states that an abstraction function is pruned if for any $\bar{t}_i$, $shapes(\bar{t}_i) \nsubseteq P_{all} \cup Q_{all}$ holds (*New shape*).

If in any abstract state of an output state, the number of nodes exceeds the number of nodes in the input states multiplied by n_C, the parameter for the maximal number of new nodes an action can create, that abstraction function is pruned. The actions can not create this many new nodes (*Too many output nodes*). For example, if $n_C = 2$ and the abstract state of an output image contains more than two times as many nodes as the abstract state of the input image, then the actions can not create all of the nodes in the output since for each node at most two new ones can be created.

For the abstract solution formulation, this reduced the number of candidate task-abstraction function pairs from 3680 of 160 tasks to 726 pairs of 126 tasks for $n_C = 1$, meaning that each action can, at most, create one new node per selected node. Here, in 34 tasks, all abstraction functions were prunded. For $n_C = 2$, 987 pairs of 139 tasks were not pruned. 21 tasks were excluded, since no abstraction function was applicable.

4.2.2 Clingo Encoding

The abstract states $d_1, \dots d_k$ and $\overline{d}_1, \dots, \overline{d}_k$ are encoded in Clingo. For all abstract states $d_i = \big(\{n_1, \dots, n_f\}, w_i, h_i\big)$ or $\overline{d}_i = \big(\{n_1, \dots, n_f\}, w_i, h_i\big)$, predicates

$$nodeProperty((i, input, k), P, n_P)$$

are created for all $n_i \in N$ and $P \in \{x, y, rot, \dots\}$. Such that $input = 0$ if the node is from an input image and $input = 1$ if it is from an output image.

For each d_i, predicates $width(i, w_i, h_i)$ and $height(i, w_i, h_i)$ are created. These are used to evaluate the number bindings that reference these values. The evaluations of node properties P_F in filters, node attributes A_{VB} and relations R_{VB} in value bindings are precalculated in Python and encoded as predicates.

The parameters limiting transformations' maximal complexity are encoded as predicates. The following parameters are given to the Clingo program:

Predicate	Definition
$maxFilters(m_F)$	Number of filters per action
$maxVBinFilter(m_V)$	Number of value bindings per filter
$maxVBinTransformation(m_T)$	Number of value bindings per transformation
$numActions(n_A)$	Number of actions (may be noop)
$numCopySteps(n_C)$	Number of copy-steps for each action

If the *direct* solution formulation is used, additional input is required. For each pattern predicates $shape\_img(P, R, M, X, Y, C)$ encode that the pattern with index P has color C at position (X, Y) if it is rotated R-times by 90° and mirrored if $M = 1$. The output images are encoded by predicates $out\_img(E, X, Y, C)$, indicating that the output image of example E has color C at position (X, Y).

Clingo Encoding of Policy Syntax and Semantic

The part of learning transformations in Clingo can be separated into three parts: *generate*, *derive*, and *check* (or $check_{render}$).

The *generate* part defines how actions can be formed from filters and value bindings. It defines their structure by encoding the recursive definitions of Equations (4.2) and (4.3). It also ensures that the resulting actions obey all parameters restricting their complexity. Thus, it ensures that all correctly formed policies are candidate solutions.

The *derive* part evaluates filters and number bindings for a given representation of the abstract state by implementing the semantics of filters and value bindings. It accesses the encoding of the abstract state provided as input, additionally, the predicates that encode a policy.

The *check* part verifies if a policy solves the demonstration examples abstractly. For each given example, it applies the actions using the evaluations of filters and value bindings obtained in the derive part. It is satisfiable if and only if the set of created nodes is the same as the set of nodes in the output state. This is ensured by finding a bijection between the created nodes and the nodes in the output abstract state, such that all connected nodes have equal properties values.

$check_{render}$ implements the *direct* solution formulation. Each derived output nodes is rendered into an individual image with dimensions of the output image, which are then combined. Overlapping pixels are handled by prioritizing nodes that have been modified most recently. The encoding does not scale well and it is only feasible to use in tasks with very small image dimmensions.

Clingo Optimization Directives

Following the assumption that a simpler policy is more likely to generalize, the Clingo encoding contains optimization statements that minimize the complexity of the found policy.

Finding simple solutions is achieved by minimizing certain optimization directives in Clingo. The optimization directives in Clingo allow the expression of priorities over these directives. Optimizing a directive with a lower priority value is more important. If two objectives have the same priority, their sum is minimized. The optimization is set up to minimize the following objectives at the given priority.

Minimization objective	Definition	Priority
#(p)	Number of transformed properties	1
#(a)	Number of non-trivial actions	1
#(F)	Number of filters	2
#(VB)	Number of value bindings	2
#(F-T)	Number of filter types used	3
#(VB-T)	Number of value-binding types used	3

Here, #(p) is the number of properties modified by any action (i.e. $V_{a,j,p} \neq n_p$ for any action a and any j). #(F) denotes the number of filters and #(VB) refers to the number of value bindings. Recursively defined filters and value bindings are included in this count. Value bindings used in a transformation of a property p with structure $V_{a,j,p}(n) := n.p$ are not considered in this count since the action does not modify the node's property in this case. The last two rules minimize the number of filter and value binding definition types from Equations (4.2) and (4.3) that are used by any filter or value binding.

Clingo produces intermediate non-optimal solutions during optimization. If the experiment's time limit is reached and Clingo is still in the optimization process, the most recently discovered policy is be selected.

Listing 4.1: Clingo Optimization Directives.

```
#minimize{1@1, A : is_transformed(A)}.
#minimize{1@1, A : action(A), not noop_action(A)}.

#minimize{1@2, Y : numberExpr(Y), not identity_Number(Y)}.
#minimize{1@2, X : filter(X), not filterType(X, fnot)}.

#minimize{1@3, A : usedNumberTypes(A)}.
#minimize{1@3, A : usedFilterTypes(A)}.
```

The Clingo implementation of these objectives is presented in Listing 4.1

All the previous encodings are combined such that the Clingo program consists of 5 parts: *input*, *generate*, *derive*, *check* (or $check_{render}$), and *optimization*.

4.3 Examples

The following Section presents two of the solutions of the ARC tasks that were correctly found by Clingo.

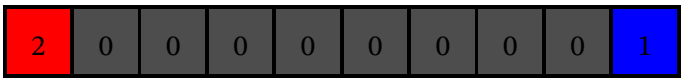
4.3.1 Task 22eb0ac0

Figure 4.4: Task 22eb0ac0; Same colored pixels are connected.

Listing 4.2: Policy encoding of task 22eb0ac0 and abstraction function ”abstract_y_slices_no_abstraction”.

```
action(0).
actionType(0,copy).
copySteps(0,1).
filterType((0,1),symVert).
filter((0,1)).
numberType((((0,0),color(2)),1),prop).
numberExpr((((0,0),color(2)),1)).
is_transformed(color(2)).
numberpropValueLink((((0,0),color(2)),1),color(0)).
```

The abstraction functions create objects such that each object corresponds to a horizontal slice of the images and all nodes have the same shape:



The policy consists of a single action. This action transforms all vertically symmtric nodes by changing thir $color_0$ property to the value of $color_1$. The slices with differently colored pixels on either side are not selected because they are not vertically symmetric. The filter selects the all-black slices, which remain unchanged because $color_0 = 0$ and $color_1 = 0$. Slices where the left and right pixels have the same color are also selected and the middle part is recolored to match the color of the leftmost pixel.

This solution is given as $(abstract\_y\_slices\_no\_abstraction, (a))$, where the policy is a single action $a = (F_a, V_a, n_a)$ with

- $F_a(x) := symVert(x)$
- $V_a := \{V_{a,0,p} : p \in \{x, y, rot, mirr, shape, color_0, ..., color_9\}\}$
- $V_{a,0,color_0}(n) := n_{color_1}$
- $V_{a,0,p} := n_p$ for $p \in \{x, y, rot, mirr, shape, color_1, ..., color_9\}$
- $n_a := 1$

4.3.2 Task 42a50994

This policy consists of a single action that deletes all nodes symmetric under a 90° rotation. This condition applies to all isolated pixels, while all larger objects do not follow this symmetry.

The solution is given as the tuple $(abstract\_8connected\_bg\_rm, (a))$ with $a = (symrot90, \emptyset, 0)$.

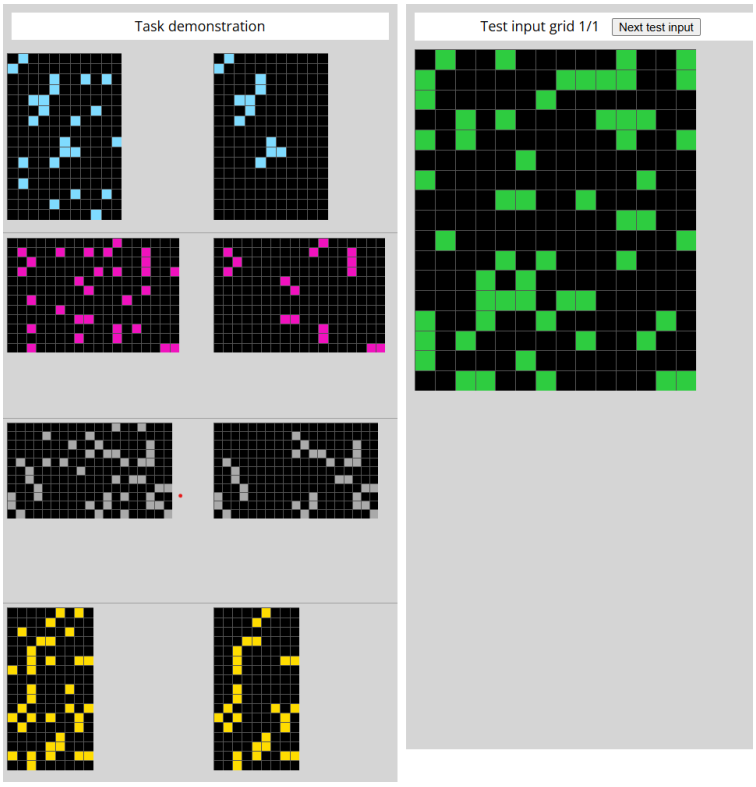


Figure 4.5: Task 42a50994; In all isolated pixels are deleted.

Listing 4.3: Policy encoding of task 42a50994 and abstraction function ”abstract_8connected_bg_rm”.

```
1 action(0).
2 actionType(0,copy).
3 copySteps(0,0).
4 filter((0,1)).
5 filterType((0,1),symrot90).
```

5 Results

This section presents some results gathered from implementing the previously described approach. The abstract and direct solution formulations are both implemented in Clingo. Abstraction functions, the encoding of abstract states as prediates, and the calculation of node properties, attributes, and relations are implemented in Python.

The following parameters that restrict the complexity of possible solutions are selectable in the implementation.

- m_F: Maximal number of filters
- m_V: Maximal number of value bindings per filter
- m_T: Maximal number of value bindings per transformation
- n_A: Number of actions
- n_C: Maximal number of nodes created for each node

In the following, the abstract solution formulation is evaluated on the object-centric ARC subset [15] and compared to the results of ARGA [15] and GPAR [10]. The direct solution formulation could not be evaluated on this subset because image sizes are too large, while evaluation on the MiniARC is possible since image sizes are restricted there. Abstract and direct solution formulation are compared by evaluating them on the MiniARC [8] benchmark.

5.1 Evaluation on ARC subset

Experiment name	m_F	m_V	m_T	n_A	n_C	Training Accuracy		Testing Accuracy	
Experiment 1	1	1	1	1	1	23/160	14.38%	22/160	13.75%
Experiment 2	3	3	3	1	1	55/160	34.38%	44/160	27.50%
Experiment 3	3	3	3	2	1	68/160	42.50%	45/160	28.13%
Experiment 4	1	1	3	2	2	70/160	43.75%	49/160	30.63%
Experiment 5	3	1	1	3	1	68/160	42.50%	47/160	29.38%
Ensemble (top 3)	-	-	-	-	-	85/160	53.13%	60/160	37.50%

Table 5.1: Results for the subset of 160 ARC tasks and abstract solution formulation; Overview of parameters and results of performed experiments. The Clingo solver was restricted to run for at most 15 seconds per task and abstraction function. Testing accuracy describes how often one of thet top three solutions with the least complexity solved the test examples. Ensemble considers the top three least complex solution of any experiment.

The Clingo encoding grows quite fast with growing parameter values. Multiple experiments with different parameter configurations were run, to allow testing larger parameter values. An overview of performed experiments and achieved results is given in Table 5.1. All of these experiments use the abstract solution formulation. Since the rules of ARC allow three guesses, testing accuracy is calculated by counting the number of tasks where any of the three solutions with minimal complexity solves the test examples. Complexity is measured by the Clingo optimization directives described in Section 4.2.2. In *Experiment 4*, only considering the top three solutions reduced the solved tasks from 50 to 49. In all other experiments, it did not have any effect. In Ensemble, the top three solutions of all Experiments are considered. They solved 60 tasks, while overall for 65 tasks an solution was found.

As illustrated in Table 5.2, the Ensemble of experiments 1-5 outperforms ARGA in the categories movement and recolor and in the combined score of all tasks, while ARGA solves more tasks in the augmentation category. This is probably due to the actions in ARGA that allow to extend or augment certain objects, while the approach in this work can not produce new shapes. It performs slightly worse than the Kaggle first place from 2020 [1] but solves more tasks correctly in the recolor category. The Ensemble has the worst generalization ability. The solutions generalize to solve the test examples in 70% of the tasks where the demonstration examples are learned. In ARGA, this is 90,5%, Kaggle First Place with 81% and GPAR is best with 94%.

Model	Task Type	Training Accuracy		Testing Accuracy	
ARGA	movement	18/31	(58.06%)	17/31	(54.84%)
	recolor	25/62	(40.32%)	23/62	(37.10%)
	augmentation	20/67	(29.85%)	17/67	(25.37%)
	all	63/160	(39.38%)	57/160	(35.62%)
Kaggle First Place	movement	21/31	(67.74%)	15/31	(48.39%)
	recolor	23/62	(37.10%)	28/62	(45.16%)
	augmentation	**35**/67	(**52.24%**)	21/67	(31.34%)
	all	79/160	(49.38%)	64/160	(40.00%)
GPAR	movement	20/31	(64.52%)	**19**/31	(**61.30%**)
	recolor	41/62	(66.13%)	**39**/62	(**62.90%**)
	augmentation	25/67	(37.31%)	**23**/67	(**34.33%**)
	all	**86**/160	(**53.75%**)	**81**/160	(**50.63%**)
Ensemble Exp. 1-5	movement	**23**/31	(**74.13%**)	14/31	(45.16%)
	recolor	**44**/62	(**70.97%**)	35/62	(51.61%)
	augmentation	18/67	(25.37%)	11/67	(16.42%)
	all	85/160	(53.13%)	60/160	(37.50%)

Table 5.2: Performance of ARGA, Kaggle First Place (2020), GPAR, and Ensemble of Experiments 1-5 over 160 object-centric ARC tasks. Training accuracy is the number of tasks where the solution solves all the training instances. Testing accuracy is the number of tasks where the solution generates the correct output images for all test instances. The best results are in bold.

5.2 Evaluation on MiniARC

The direct solution formulation is only evaluated on the MiniARC benchmark because the large number of pixels on normal ARC tasks leads to a very high runtime. The limited number of pixels in the MiniARC images (5x5 images) made it easier to evaluate the approach.

Multiple parameter configurations have been tried for the MiniARC aswell. Table 5.3 shows the results of the performed experiments on the MiniARC dataset. The *direct* solution formulation solves more tasks than the *abstract* solution formulation for every parameter configuration. In the third experiment, with two actions, the direct solution generalized very badly; in 49% of the tasks did the solutions correctly solve the test examples.

Formulation	m_F	m_V	m_T	n_A	n_C	Training Accuracy	Testing Accuracy	Timeouts
abstract	1	1	1	1	1	30/149	26/160	0/1396
direct						43/149	31/160	62/2762
abstract	3	1	1	1	1	31/149	26/160	0/1396
direct						47/149	34/160	18/2762
abstract	1	1	1	2	1	55/149	41/160	5/1396
direct						92/149	45/160	606/2762

Table 5.3: MiniARC results: Overview of parameters in performed experiments. The Clingo solver was restricted to run for a maximum of 60 seconds per task and abstraction function. Numbers in bold indicate which solution formulation performs better. The timeout column indicates the number of times the Clingo encoding exceeded the time limit of 60 seconds.

5.3 Further Analysis

One observation is that smaller tasks seem to be easier to solve. Table 5.4 illustrates this, showing that in the object-centric ARC subset, the images of solved tasks have, on average, fewer pixels than the images of tasks where no solution is found.

Train solved	Test solved	Average Number of Pixels
Yes	Yes	221.39
Yes	No	404.67
No	No	540.86

Table 5.4: Average number of pixels in task images for different train and test solved cases.

Another observation is illustrated in Table 5.5, which shows that the solved tasks overlap a lot between the abstraction functions. Almost half of the abstraction functions (10/23) did not solve any unique tasks and the top 5 abstraction functions solve the majority of tasks. The table was generated by iteratively adding the abstraction function that solved the most tasks that had not been solved by any previous one. The *solved tasks* column describes the number of tasks

correctly solved in any of the experiments using that abstraction function. *New tasks* refers to the number of tasks solved by the abstraction function but not by any further up in the table.

Abstraction function	Solved tasks	New tasks
abstract_4connected_bg_rm	25	25
abstract_pixel_bg_rm	22	14
abstract_4connected	22	10
abstract_y_slices_multicolor_bg_rm	11	4
abstract_8connected_multicolor_bg_rm	11	3
abstract_x_slices	14	2
abstract_no_abstraction	11	2
abstract_x_slices_bg_rm	19	1
abstract_4connected_multicolor_bg_rm	13	1
abstract_y_slices_no_abstraction	12	1
abstract_color_bg_rm	8	1
abstract_x_slices_no_abstraction	13	1
abstract_x_slices_multicolor_bg_rm	10	0
abstract_color	8	0
abstract_8connected	10	0
abstract_pixel	13	0
abstract_8connected_bg_rm	8	0
abstract_y_slices_multicolor	13	0
abstract_y_slices	15	0
abstract_y_slices_bg_rm	15	0
abstract_x_slices_multicolor	13	0
abstract_4connected_multicolor	10	0
abstract_8connected_multicolor	4	0

Table 5.5: Number of tasks solved by each abstraction function in the second column and the number of tasks not solved by any previous abstraction function in the third column. 11 of the 23 tasks did not solve any new tasks.

6 Conclusion

The approach was tested on a set of object-centric tasks of the original ARC and on the MiniARC dataset, which consists of tasks with smaller image sizes. The results indicate that the approach can solve some of them effectively. Specifically, the direct formulation method showed high expressiveness but was inefficient in computation time. On the other hand, the abstract formulation was more efficient but had limited expressiveness. Both struggled to generalize well to the test tasks. The results of the MiniARC demonstrated that the direct formulation is, in practice, more powerful but showed its high computational demands; even on the smaller tasks, only small parameter values could be evaluated. Overall, the approach produced some good results, but more refinement is needed to improve expressiveness and generalization capabilities. Adding new abstraction functions did not seem to have a big influence on the systems' performance; a small set of abstraction functions managed to solve a big fraction of all solved tasks.

One challenge of the abstract formulation is that only previously observed shapes could be predicted. One possible way to overcome this would be to allow actions to select shapes from a larger list. This list could include cropped versions of previously observed shapes or standard shapes like rectangles or lines. Another extension could be to consider the abstractions as actions. These actions would modify an abstract state by dividing nodes using the refinement operators seen in Section 4.1.1. If the division of nodes can be efficiently implemented in Clingo, these actions could become part of the policy and be searched simultaneously with the other actions. Allowing the normal action and these new abstraction actions to be applied alternating could increase expressivity even further.

6.1 How the Approach Implements Core Knowledge

The presented approach implements some of the Core Knowledge systems, as introduced in Section 2.1, while some others are missing.

The Objectness prior is implemented in the abstraction process. Each node is an object in the image. The object cohesion prior is implemente by recognizing objects using the abstraction functions that group pixels by colors or connectivity. The algorithm lacks a good implementation of object persistence. In the *direct* solution method objects can be obstructed by other objects, in the *abstract* formulation, this is not possible and thus object persistence is not possible. The prior of interaction via contact can be partly implemented using the *adjacent*

node relation. However rebounding or tasks that require more complex interactions are not possible.

Agentness or goal-directedness is not encoded. The actions are performed in parallel and they can not work step by step toward a goal.

The natural numbers and arithmetics prior is implemented in the actions by allowing nodes to be filtered based on equalities and comparisons and to dynamically transform their properties based on numbers derived from arithmetic functions.

The priors concerning Geometry or Topology, only ”Symmetries, rotations, translations”, and ”Copying, repeating objects” are implemented. Some node properties and relations are predefined and can be accessed in the filters and value bindings. However, they do not fully capture all of the formulated priors in this category. Additionally, the structure of the actions allows for rotations, translations, and to generate symmetries.

A Appendix

A.1 Refinement Definitions

The following *refinement operators* are used $O = \{black, connect8, connect4, color, x, y\}$

Given an image $I \in \{0, \dots, 9\}^{n \times m}$. Let $P = \{(x, y, I_{x,y}) : 0 \leq x \leq n, 0 \leq y \leq m\}$ and $P_{black} = \{(x, y, c) \in P : c = 0\}$.

Refinement operators, as used in this work, operate on partitions of P or $P \backslash P_{black}$. But they can be defined more generally to operate on sets that contain subsets of pixels (elements in the powerset of the powerset of P).

A *refinement operator* $o \in O$ is a mapping $o : \mathcal{P}(\mathcal{P}(P)) \to \mathcal{P}(\mathcal{P}(P))$ with following definition:

$$
\begin{aligned}
black(x) &:= \bigcup_{p \in x} \{\{(x, y, c) \in p : c \neq 0\}\} \cup \{\{(x, y, c) \in p : c = 0\}\} \\
color(x) &:= \bigcup_{p \in x} \bigcup_{c' \in \{0,\dots,9\}} \{\{(x, y, c) \in p : c = c'\}\} \\
x(x) &:= \bigcup_{p \in x} \bigcup_{x' \in \{0,\dots,n\}} \{\{(x, y, c) \in p : x = x'\}\} \\
y(x) &:= \bigcup_{p \in x} \bigcup_{y' \in \{0,\dots,m\}} \{\{(x, y, c) \in p : y = y'\}\} \\
connect4(x) &:= \bigcup_{p \in x} \bigcup_{v \in p} \{\{(x, y, c) \in p : reachable4_p((x, y, c), v)\}\} \\
connect8(x) &:= \bigcup_{p \in x} \bigcup_{v \in p} \{\{(x, y, c) \in p : reachable8_p((x, y, c), v)\}\}
\end{aligned}
$$

$reachable4_p(v_1, v_2)$ is true if there is a path of pixels in p form v_1 to v_2, where pixels are adjacent if they have horizontal or vertical contact. Since $reachable4_p$ is symmetric, reflexive, and transitive, it is an equivalence relation and thus, the function is well-defined.

$reachable8_p$ is analogously defined, but for pixels to be considered adjacent also diagonal connections are allowed. By the same argument it is also well-defined.

Additionally another operator, $remove\_black : \mathcal{P}(\mathcal{P}(P)) \to \mathcal{P}(\mathcal{P}(P \backslash P_{black}))$, are used. It removes all elements that contain at least one black pixel:

$$remove\_black(x) := \{p \backslash P_{black} \ : \ p \in x\}$$

The abstraction functions used can be obtained by concatenations of these operators. The following shows the definition of all abstraction functions:

$$\begin{aligned}
abstract_no_abstraction &:= id \\
abstract_4connected &:= connect4 \circ color \\
abstract_4connected_multicolor &:= connect4 \circ black \\
abstract_8connected &:= connect8 \circ color \\
abstract_8connected_multicolor &:= connect8 \circ black \\
abstract_color &:= color \\
abstract_pixel &:= x \circ y \\
abstract_x_slices &:= x \circ col \\
abstract_x_slices_multicolor &:= x \circ black \\
abstract_x_slices_no_abstraction &:= x \\
abstract_y_slices &:= x \circ col \\
abstract_y_slices_multicolor &:= c \circ black \\
abstract_y_slices_no_abstraction &:= y \\
abstract_4connected_br &:= remove_black \circ connect4 \circ color \\
abstract_4connected_multicolor_br &:= remove_black \circ connect4 \circ black \\
abstract_8connected_br &:= remove_black \circ connect8 \circ color \\
abstract_8connected_multicolor_br &:= remove_black \circ connect8 \circ black \\
abstract_color_br &:= remove_black \circ color \\
abstract_pixel_br &:= remove_black \circ x \circ y \\
abstract_no_abstraction_br &:= remove_black \circ id \\
abstract_x_slices_br &:= remove_black \circ x \circ col \\
abstract_x_slices_multicolor_br &:= remove_black \circ x \circ black \\
abstract_x_slices_no_abstraction_br &:= remove_black \circ x \\
abstract_y_slices_br &:= remove_black \circ x \circ col \\
abstract_y_slices_multicolor_br &:= remove_black \circ c \circ black \\
abstract_y_slices_no_abstraction_br &:= remove_black \circ y
\end{aligned}$$

A.2 Node Relations and Properties

R_{VB} contains the following elements:

- nearestNeighborLeft
- nearestNeighborRight
- nearestNeighborUp
- nearestNeighborDown
- biggestNode
- smallestNode

Additional node properties P_F are:

- size
- numNeighbors
- numNeighbors8
- height
- width
- mostFrequentColor
- mostFrequentShape
- mindistancetoBorder
- distanceToBorderLeft
- distanceToBorderRight
- distanceToBorderUp
- distanceToBorderDown

The following node attributes A_{VB} were used

- symHor
- symVert
- symDiag1
- symDiag2
- symRot90
- symRot180
- symRot270
- biggestNode
- smallestNode

A.3 Abstract Solutions Are Direct Solutions

Let (f, A) be a *abstract* solution to the image transitions $I_1 \hookrightarrow I_2$ with abstract states $S_1 = abstract_f(I_1)$ and $S_2 = abstract_f(I_2)$ be the abstract states that result from abstracting the

images I_1 and I_2 using the abstraction function f. Then $S_1 \rightarrow_A S_2$, i.e., The actions transform the abstract state of the input image to the abstract state of the output image.

Since S_2 resulted from an abstraction process and by Equation (4.1), which states: $render\big(abstract_f(I)\big) = I$. Hence $I_2' := render(S_2) = render\big(abstract_f(I_2)\big) = I_2$and thus $(f, A) \vDash_d I_1 \hookrightarrow I_2$ holds, meaning that (f, A) is a *direct* solution to the image transition.

List of Acronyms

ARC Abstraction and Reasoning Corpus

ARGA Abstract Reasoning with Graph Abstractions

ASP Answer Set Programming

GPAR Generalized Planning for Abstract Reasoning

Potassco Potsdam answer set solving collection

List of Figures

1.1 Image adapted from [15]. Three tasks from the ARC dataset. For each task, three demonstration examples and one test example. 1

3.1 Image adapted from Xu *et al.* [16]; Approach to use GPT to learn ARC tasks. . . 9

4.1 Illustration of normalizing a pattern. 13
4.2 Task 0962bcdd; The shape of each multicolor-connected object is changed. . . 14
4.3 Illustration steps performed to evaluate abstract (red) and direct (green) formulation. (This is ARC-task 0962bcdd) 19
4.4 Task 22eb0ac0; Same colored pixels are connected. 23
4.5 Task 42a50994; In all isolated pixels are deleted. 25

List of Tables

4.1 Number of task-abstraction function pairs pruned from the 3280 possible ones by pruning criterion. *Too many output nodes(i)* depends on the parameter $n_C = i$ that limits the maximal number of copy steps allowed. One task-abstraction function pair may be excluded by multiple critera. 20

5.1 Results for the subset of 160 ARC tasks and abstract solution formulation; Overview of parameters and results of performed experiments. The Clingo solver was restricted to run for at most 15 seconds per task and abstraction function. Testing accuracy describes how often one of thet top three solutions with the least complexity solved the test examples. Ensemble considers the top three least complex solution of any experiment. 26

5.2 Performance of ARGA, Kaggle First Place (2020), GPAR, and Ensemble of Experiments 1-5 over 160 object-centric ARC tasks. Training accuracy is the number of tasks where the solution solves all the training instances. Testing accuracy is the number of tasks where the solution generates the correct output images for all test instances. The best results are in bold. 27

5.3 MiniARC results: Overview of parameters in performed experiments. The Clingo solver was restricted to run for a maximum of 60 seconds per task and abstraction function. Numbers in bold indicate which solution formulation performs better. The timeout column indicates the number of times the Clingo encoding exceeded the time limit of 60 seconds. 28

5.4 Average number of pixels in task images for different train and test solved cases. 28

5.5 Number of tasks solved by each abstraction function in the second column and the number of tasks not solved by any previous abstraction function in the third column. 11 of the 23 tasks did not solve any new tasks. 29

List of Listings

4.1 Clingo Optimization Directives. 23
4.2 Policy encoding of task 22eb0ac0 and abstraction function ”abstract_y_slices_no_abstraction”. 24
4.3 Policy encoding of task 42a50994 and abstraction function ”abstract_8connected_bg_rm”. 25

List of References

[1] *Abstraction and Reasoning Challenge*, https://kaggle.com/abstraction-and-reasoning-challenge. Accessed: Jan. 8, 2025.

[2] J. Ainooson, D. Sanyal, J. P. Michelson, Y. Yang, and M. Kunda, *A Neurodiversity-Inspired Solver for the Abstraction \& Reasoning Corpus (ARC) Using Visual Imagery and Program Synthesis*, Oct. 2023. DOI: 10.48550/arXiv.2302.09425. arXiv: 2302.09425 [cs]. Accessed: Feb. 25, 2024.

[3] F. Chollet, *On the Measure of Intelligence*, Nov. 2019. DOI: 10.48550/arXiv.1911.01547. arXiv: 1911.01547 [cs]. Accessed: Mar. 15, 2024.

[4] F. Chollet, M. Knoop, G. Kamradt, and B. Landers, *ARC Prize 2024: Technical Report*, Dec. 2024. DOI: 10.48550/arXiv.2412.04604. arXiv: 2412.04604 [cs]. Accessed: Jan. 8, 2025.

[5] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and S. Thiele, "A User's Guide to gringo, clasp, clingo, and iclingo,"

[6] S. J. Gershman, "Complex Probabilistic Inference: From Cognition to Neural Computation,"

[7] A. Johnson and B. M. Lake, "Fast and flexible: Human program induction in abstract reasoning tasks,"

[8] S. Kim, P. Phunyaphibarn, D. Ahn, and S. Kim, "Playgrounds for Abstraction and Reasoning," in *NeurIPS 2022 Workshop on Neuro Causal and Symbolic AI (nCSI)*, Oct. 2022. Accessed: Jan. 8, 2025.

[9] S. Kim, P. Phunyaphibarn, D. Ahn, and S. Kim, "Playgrounds for Abstraction and Reasoning,"

[10] C. Lei, N. Lipovetzky, and K. A. Ehinger, *Generalized Planning for the Abstraction and Reasoning Corpus*, Jan. 2024. arXiv: 2401.07426 [cs]. Accessed: Apr. 7, 2024.

[11] W.-D. Li, K. Hu, C. Larsen, Y. Wu, S. Alford, C. Woo, S. M. Dunn, H. Tang, M. Naim, D. Nguyen, W.-L. Zheng, Z. Tavares, Y. Pu, and K. Ellis, *Combining Induction and Transduction for Abstract Reasoning*, Dec. 2024. DOI: 10.48550/arXiv.2411.02272. arXiv: 2411.02272 [cs]. Accessed: Dec. 15, 2024.

[12] A. Moskvichev, V. V. Odouard, and M. Mitchell, *The ConceptARC Benchmark: Evaluating Understanding and Generalization in the ARC Domain*, May 2023. DOI: 10.48550/arXiv.2305.07141. arXiv: 2305.07141. Accessed: Nov. 19, 2024.

[13] N. Neumann and Á. Pintér, "Solving ARC with non-procedural program induction," in *2023 IEEE 23rd International Symposium on Computational Intelligence and Informatics (CINTI)*, Budapest, Hungary: IEEE, Nov. 2023, pp. 000 271–000 276, ISBN: 9798350342949. DOI: 10.1109/CINTI59972.2023.10382009. Accessed: Apr. 7, 2024.

[14] E. S. Spelke and K. D. Kinzler, "Core knowledge," *Developmental Science*, vol. 10, no. 1, pp. 89–96, 2007, ISSN: 1467-7687. DOI: 10.1111/j.1467-7687.2007.00569.x. Accessed: Apr. 8, 2024.

[15] Y. Xu, E. B. Khalil, and S. Sanner, *Graphs, Constraints, and Search for the Abstraction and Reasoning Corpus*, Dec. 2022. DOI: 10.48550/arXiv.2210.09880. arXiv: 2210.09880 [cs]. Accessed: Jan. 8, 2025.

[16] Y. Xu, W. Li, P. Vaezipoor, S. Sanner, and E. B. Khalil, *LLMs and the Abstraction and Reasoning Corpus: Successes, Failures, and the Importance of Object-based Representations*, Feb. 2024. arXiv: 2305.18354 [cs]. Accessed: Jul. 15, 2024.