

Thesis for Bachelor's Degree

Abductive Symbolic Solver on Abstraction and Reasoning Corpus

Lim, Mintaek (임 민택 林 旻澤)

School of Electrical Engineering and Computer Science

Gwangju Institute of Science and Technology

2024

Abductive Symbolic Solver on Abstraction and Reasoning Corpus

ARC 문제에서 귀추법을 활용한 심볼릭 모델에
대한 연구

Abductive Symbolic Solver on Abstraction and Reasoning Corpus

Advisor : Professor Kim, Sundong

by

Lim, Mintaek

School of Electrical Engineering and Computer Science
Gwangju Institute of Science and Technology

A thesis submitted to the faculty of Gwangju Institute of Science and
Technology in partial fulfillment of the requirements for the degree of
Bachelor of Science in Electrical Engineering and Computer Science

Gwangju, Republic of Korea

2024. 6. 17.

Approved by

Professor Kim, Sundong

Committee Chair

Abductive Symbolic Solver on Abstraction and Reasoning Corpus

Lim, Mintaek

Accepted in partial fulfillment of requirements
for the degree of Bachelor of Science

June. 17. 2024.

Committee Chair



Prof. Sundong Kim

Committee Member

Prof. Sunbeom So

BS/EC
20185152

Lim, Mintaek (임 민택). Abductive Symbolic Solver on Abstraction and Reasoning Corpus (ARC 문제에서 귀추법을 활용한 심볼릭 모델에 대한 연구). School of Electrical Engineering and Computer Science. 2024. 17p. Advisor Prof. Kim, Sundong.

Abstract

This paper addresses the challenge of enhancing artificial intelligence reasoning capabilities, focusing on logicity within the Abstraction and Reasoning Corpus (ARC). Humans solve such visual reasoning tasks based on their observations and hypotheses, and they can explain their solutions with a proper reason. However, many previous approaches focused only on the grid transition and it is not enough for AI to provide reasonable and human-like solutions. By considering the human process of solving visual reasoning tasks, we have concluded that the thinking process is likely the abductive reasoning process. Thus, we propose a novel framework that symbolically represents the observed data into a knowledge graph and extracts core knowledge that can be used for solution generation. This information limits the solution search space and helps provide a reasonable mid-process. Our approach holds promise for improving AI performance on ARC tasks by effectively narrowing the solution space and providing logical solutions grounded in core knowledge extraction.

Contents

Abstract	i
Contents	ii
Chapter 1. Introduction	1
Chapter 2. Related Works	3
2.1 Domain Specific Languages (DSL)	3
2.2 Graph in ARC	3
2.3 Abductive Reasoning	3
Chapter 3. Method	4
3.1 Structure of the ARC Knowledge Graph	4
3.2 Domain Specific Language for Knowledge Graph Construc- tion and Symbolic ARC solver	5
3.2.1 Data Types	5
3.2.2 DSL Categories	6
3.3 Symbolic ARC solver and Abductive Reasoning Learning . .	7
3.3.1 Solution Inference	7
3.3.2 Abductive Reasoning Learning	7
Chapter 4. Experiment and Results	10
4.1 Experimental Setup	10
4.2 Result	10
4.2.1 Comparison of Solver Performance with and without the Use of Knowledge Graph	10
4.2.2 Differences in Performance by Size of Synthesizer . . .	11
Chapter 5. Conclusion	12
Summary (한글 요약문)	13
Acknowledgments (감사의 글)	14
Curriculum Vitae (약력)	15

Chapter 1. Introduction

Artificial intelligence nowadays exhibits impressive problem-solving skills in many domains. Though they provide valuable assistance, not all responses make sense due to the hallucination issue and lack of logical stability. According to Pan Lu et al., especially within the category of mathematical reasoning, logical reasoning, and numeric commonsense, AI agents underperformed compared to other areas such as scientific, statistical, and algebraic reasoning. Moreover, the "puzzle test" and "abstract scene" tasks showed averagely the biggest performance gap between current AI models and humans [1]. To enhance such weaknesses, various experiments have been conducted on logic and puzzle test datasets [2, 3, 4, 5]. Datasets corresponding to such categories that require complex logical capabilities with visual images are called Visual Reasoning tasks.

As an IQ test is one of the representative measurements of human intelligence, Abstraction and Reasoning Corpus (ARC) was invented by François Chollet to measure the intelligence of an AI [2]. The ARC dataset has 400 tasks in each training and evaluation set and each consists of multiple numbers of example pairs and a test pair as shown in Figure 1.1. The task is to formulate a pattern that applies to all the example pairs and then construct an answer with the given test input grid. All tasks are created based on four core knowledge priors, which are 1) objectness, including object cohesion, persistence, and its influence via contact, 2) goal-directedness, 3) numbers and counting, and 4) basic geometry and topology [2]. Due to these characteristics, solutions that have defined domain-specific languages (DSL) have emerged. Unlike other AI techniques, two representative solutions have utilized DSLs to make the essence of each not dissolved into a vector but preserved symbolically. Moreover, the performances have resulted in 1st place in the Kaggle ARC solving competition and ARCathon 2022 [6, 7]. Therefore, this research focuses on the symbolic representation of the ARC by applying DSLs and synthesizing DSLs for the solution.

Since the transformer-based models are considered the best-performing AI, various researchers have challenged solving ARC tasks with texts by providing additional descriptions [8], applying different prompting skills [9], or estimating hypotheses between the input and output grids [10]. However, such solutions can be improved in the following two ways, 1) by using a symbolic network to generate solutions that are understandable from the human perspective, and 2) by following human thinking processes to make solutions more reasonable and human-like. As humans explain their thoughts to verify their understanding, it is necessary to check both the solution and the answer for reasoning tasks. Thus, this research proposes a symbolic solver that returns understandable and reasonable solutions.

In visual reasoning, humans establish hypotheses based on their observation [11]. Inductive reasoning is well-known as a method to generate general solutions with sufficient observations, however, finding the best solution under limited observations is appropriate with abductive reasoning. Due to such property, the human thinking process of solving the ARC is more likely abductive reasoning. In each pair of the ARC task, the transition between two grids could be represented with multiple hypotheses including 1) what has changed, 2) how or how much it has changed, and 3) why it has changed in such a way. Considering the reason for the transition is the key to this research. In

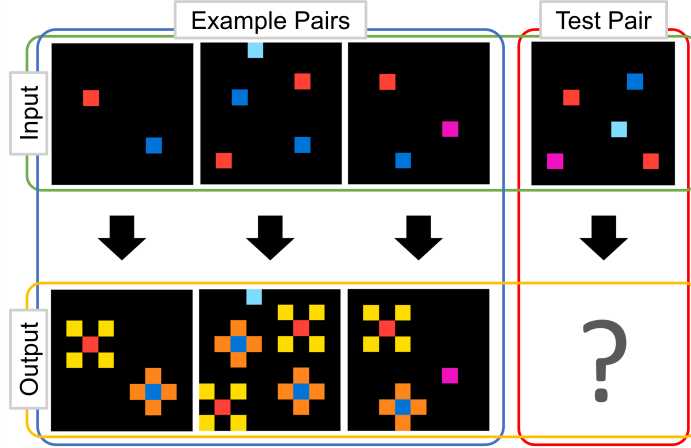


Figure 1.1: Example ARC task. Solvers are supposed to formulate a pattern that applies to all the given example pairs and then construct an answer with the given test input grid.

Figure 1.1, four orange pixels appeared around the blue pixel. With only the first pair, it is hard to guarantee a pattern for this task. Observing the second and third pairs provides more clues for formulating a solution. After checking all pairs, the reason for the orange pixel pattern can now be understood, ensuring that the target is blue. In other words, the color is the reason for the pattern not the other fundamental properties like position or counts.

Many previous approaches missed such information and struggled to select a target object to apply the pattern in the solution generation step. By emphasizing the weight of repeated features, we propose an experiment that extracts core knowledge which are the candidate arguments for the solution, and finds common transformations that utilize the extracted information to estimate the result. Our paper’s contribution is two-fold: first, it delineates the conversion of ARC tasks into knowledge graphs and the subsequent extraction of core knowledge from these graphs. Second, it presents an abductive symbolic solver that utilizes the extracted core knowledge.

Chapter 2. Related Works

2.1 Domain Specific Languages (DSL)

In tackling the ARC challenge, some researchers have designed DSLs by referencing specific ARC tasks and refining them after solving training tasks. While these DSLs prove their systematic stability through successful example pair augmentation based on handcrafted solutions, their adaptability to unseen tasks is limited [7, 12]. Recent studies have explored integrating neurodiversity-inspired methods with computational intelligence through DSLs. One such system, the Visual Imagery Reasoning Language (VIMRL), simulates human mental imagery processes in neurodivergent individuals but struggles to generalize across diverse ARC tasks [13]. Another study uses DreamCoder synthesis to create symbolic abstractions from solved tasks and design a reasoning algorithm, however, this approach heavily relies on previously solved tasks, making it less effective in novel situations [14].

2.2 Graph in ARC

The paper "Abstract Reasoning with Graph Abstractions (ARGA)" proposed using a graph-based representation to abstract input images into nodes and edges [15]. This method captures spatial and relational information but struggles with the complexity of real-world visual reasoning tasks due to its reliance on predefined graph structures and constraints, limiting its flexibility in diverse scenarios. Lastly, the paper "Mimicking Human Solutions with Object-Centric Decision Transformer" proposed an object-construction algorithm by transforming the ARC grid into a graph to cluster the nodes based on their distances [16]. Since the aim of the paper is limited to defining an object within only one layer of the graph, the current research gained motivation to expand the graph space by detecting multiple features.

2.3 Abductive Reasoning

Abductive reasoning is a type of logical inference aimed at determining the simplest and most probable explanation from observations. It is used in fields like logistics, design synthesis, and visual reasoning [17, 18, 19, 11]. Liang et al. introduced a task to evaluate machine intelligence in visual scenarios through abductive reasoning. This approach, reflecting human cognition via Observation (O) and Explanation (H), influenced our understanding of the ARC task [11].

Chapter 3. Method

The framework shown in Figure 3.1 is divided into two main stages: the conversion of ARC tasks into knowledge graphs and the derivation of answers based on these graphs. The former involves the formation of knowledge graphs using *syntax DSLs* and *property DSLs* through a program, with each example pair resulting in data containing symbolic information in a four-layered knowledge graph, shown in Figure 3.2. The latter stage is overseen by the symbolic ARC solver, comprising a *specifier* responsible for extracting specific nodes/knowledge from the knowledge graph and a *synthesizer* that utilizes this information to compose the answers.

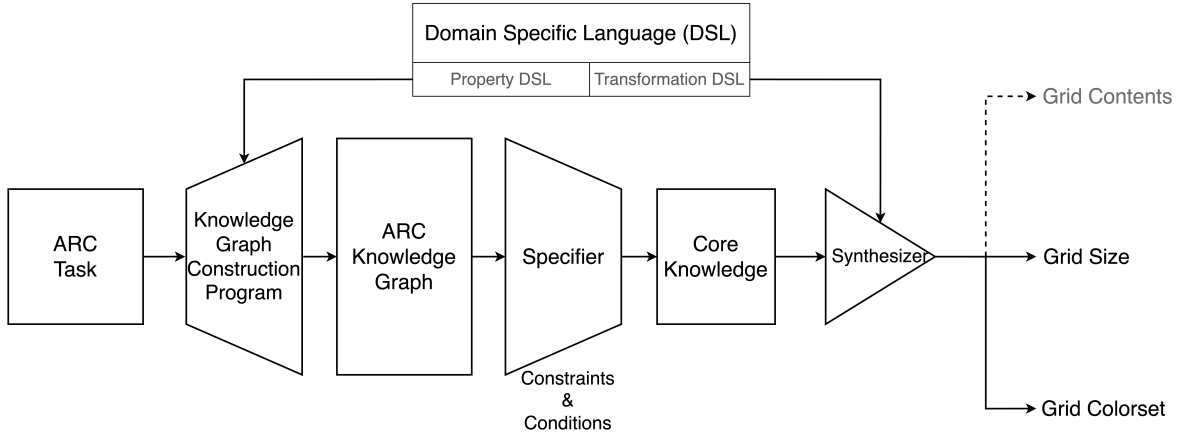


Figure 3.1: Overall framework of Symbolic ARC Solver. To tackle a given ARC problem, the first step involves generating a corresponding knowledge graph using a construction program based on DSL. Then, the trained *specifier* extracts core knowledge using which refers to objects in the knowledge graph that encompass specific conditions and constraints. Subsequently, the *synthesizer* applies transformation DSLs to this core knowledge to predict the answer.

3.1 Structure of the ARC Knowledge Graph

The original ARC data is provided in the form of a two-dimensional array, where each element of the array contains information corresponding to colors, ranging from 0 to 9. Therefore, It is challenging for machines to understand and infer rules from this data due to its limited information content. Thus, we propose a method to convert the 2D grid into a knowledge graph which captures information perceived by humans when viewing ARC problems. The knowledge graph is formed as units of one input-output example pair. A single knowledge graph consists of four layers, each characterized by the attributes of the nodes included in it. When representing the original ARC task’s example pairs as $Task = \{(I_1, O_1), (I_2, O_2), \dots, (I_n, O_n)\}$ the corresponding knowledge graphs are expressed as $KG = \{g_1, g_2, \dots, g_n\}$, where g_n is further represented as $\{NodeList_n, EdgeList_n\}$. Each $NodeList_n$ in g_n is a data structure containing all nodes found in the four layers, and $EdgeList_n$ is a data structure containing all edges found in the respective KG. The detailed description of each of the four layers is as follows.

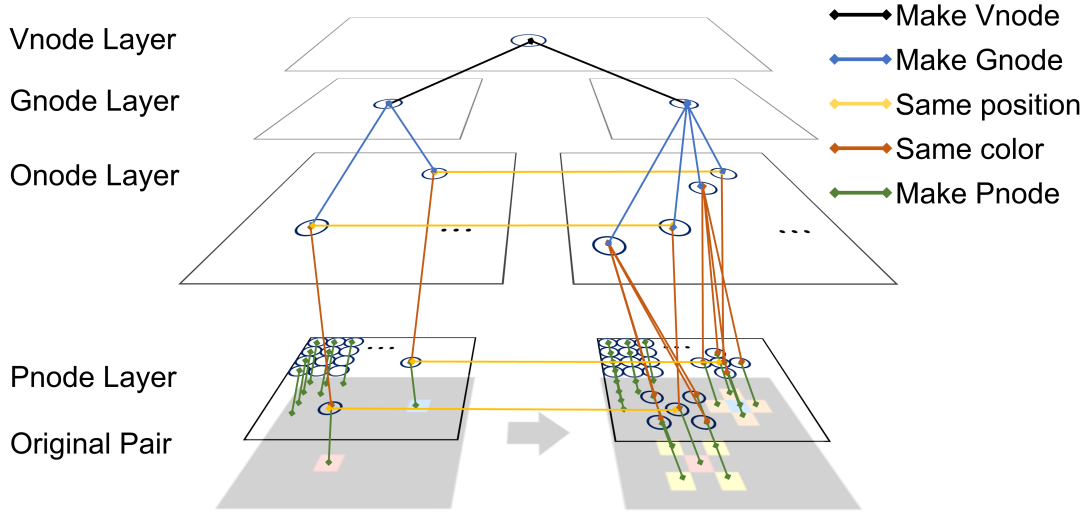


Figure 3.2: An example of a very simple, almost backbone-structured knowledge graph of the first pair of Figure 1.1. In practice, the knowledge graph generated by Algorithm 1 can contain up to millions of edges. The graph consists of four layers, with edges freely drawn between layers as well as between input and output by the property DSL. The edges highlighted in yellow represent connections between two nodes at the same position. The other (black, blue, green) indicated edges signify that nodes in the lower layer constitute nodes in the upper layer.

- **Pnode layer:** This first layer converts each pixel into a single node named *Pnode* and captures the relationships between these *Pnodes*, representing them as edges.
- **Onode layer:** This second layer contains nodes representing sets of one or more pixels forming objects. It captures the relationships between objects as edges. Nodes in this layer, which is named *Onode*, are connected to the *Pnodes* with edges.
- **Gnode layer:** This third layer represents the entire input or output grid as a single node named *Gnode*. Nodes in this layer are connected to all nodes in the lower layers including the first and second with edges.
- **Vnode layer:** This fourth layer combines the input and output grid into a single node. Each example pair is ultimately represented by one fourth-layered *Vnode*, which is connected to two *Gnodes* from the third layer through edges.

3.2 Domain Specific Language for Knowledge Graph Construction and Symbolic ARC solver

3.2.1 Data Types

In the realm of domain-specific languages (DSLs), data types form the backbone of how information is represented, manipulated, and interpreted. Table 3.1 provides an overview of the key data types

Table 3.1: Description of Data Types Used in creating DSLs

<i>Data type</i>	Description
<i>Pnode</i>	Represent a single pixel in the grid. Stores grid coordinates.
<i>Onode</i>	Represent objects in the grid formed by a collection of Pnode
<i>Gnode</i>	Represent whole grid holding Pnode and Onode as one big node.
<i>Vnode</i>	Represent a pair of input and output into one node that holds two Gnodes.
<i>Xnode</i>	Represent any type of node above.
<i>Edge</i>	Represent relationship between Pnode, Onode, Gnode, Vnode (provides connection in the graph).
<i>Color</i>	Represent a color of pixel by integer value.
<i>NodeList</i>	Represent a list of nodes.
<i>EdgeList</i>	Represent a list of edges.
<i>Coordinate</i>	Is used to represent coordinates which is a tuple of two integer values.
<i>ColorSet</i>	Is used to hold collections of color.

utilized in our DSL, each tailored to facilitate the unique requirements of nodes and their symbolic relationships.

3.2.2 DSL Categories

DSLs are classified into three categories based on their purpose. DSLs that symbolize the properties of nodes are referred to as *property DSLs* and are primarily used to draw edges in the knowledge graph. Refer to the Figure 3.3 to see what properties are defined. There are several conditions to draw an edge, such as when two nodes have the same property, when a node has a specific property, or when one node is contained within another node by some property. This category is further divided into more specific categories: *general* and *Pnode layer*. The former applies to all layers, generating edges, while the latter applies only to the Pnode layer. *Transformation DSLs* are utilized in the symbolic ARC solver and play a role in predicting the answer by applying transformations to the given nodes. Some DSLs belong to both *property DSLs* and *transformation DSLs* simultaneously, and the detailed classification is shown in Figure 3.3. *Syntax DSLs* handle the syntactical elements of DSLs and form the backbone of constructing the knowledge graph. They, in turn, are divided into DSLs for generating

Linear	 	get_number_of_nodes	 		get_background_color		Make_Edge_list	
get_identity_match	 	get_color_difference_set			get_dimension		Make_Pnode_list	
Onode_count	 	get_dimension_difference			get_color_of_node		Make_Onode_list	
get_number_of_color_set	 	get_dimension			get_coordinate		Make_Gnode_list	
get_subset_match	 	get_dominant_color			is_square		Make_Vnode_list	
get_superset_match	 	get_least_common_color			is_rectangle		Concat_list	
get_union	 	get_height_difference			is_ring		get_vertical_index	
get_height	 	get_width_difference			is_symmetric		get_manhattan_distance	
get_width	 	get_center_nodes			get_horizontal_index		get_polar_distance	
Legend  Transformation-S5  Transformation-S10  General DSL  Syntax DSL  Pnode-layer DSL								

Figure 3.3: Overview of Domain-Specific Languages (DSLs) categorized into Transformations, Pnode-layer, Syntax. The transformation DSLs are further divided into Transformation-S5 and Transformation-S10 which are classified based on the section 4.1. Transformation-S5s are used in *synthesizer-5* and Transformation-S10s are used in *synthesizer-10*.

edges, creating nodes, and combining the two lists, ultimately resulting in the knowledge graph being stored in the form of *odelist* and *edgelist*. The pseudocode for constructing the knowledge graph using this syntax is described in Algorithm 1.

Algorithm 1 Knowledge Graph Construction Program

Require: ARC Task

```

1:  $KG \leftarrow$  empty set
2: for each pair in Task do
3:   for each grid in pair do
4:      $node\_list \leftarrow \text{Make\_Pnode\_list}(grid)$ 
5:      $node\_list \leftarrow \text{Make\_Onode\_list}(node\_list)$ 
6:      $node\_list \leftarrow \text{Make\_Gnode\_list}(node\_list)$ 
7:   end for
8:   # Now, we have two node_lists from input and output grid
9:    $node\_list \leftarrow \text{Concat\_list}(node\_list\_input, node\_list\_input)$ 
10:   $node\_list \leftarrow \text{Make\_Vnode\_list}(node\_list)$ 
11:  for each Property_DSL do
12:     $edge\_list \leftarrow \text{Make\_Edge\_list}(node\_list, Property\_DSL)$ 
13:  end for
14:  Add ( $node\_list\_pair, edge\_list$ ) to  $KG$ 
15: end for
16: return  $KG$ 

```

3.3 Symbolic ARC solver and Abductive Reasoning Learning

3.3.1 Solution Inference

To address ARC problems, the inference process of our proposed symbolic ARC solver is divided into two main steps. First, core symbolic knowledge, implemented as a node form graph that can be utilized to solve the problem is extracted from the knowledge graph generated beforehand. Second, the extracted node is processed into a sequence of *transformation DSLs* to derive the correct answers. In the first step, the *specifier* is responsible for traversing the knowledge graph and extracting core knowledge (specific node) by considering some constraints. In the second step, the *synthesizer* takes charge, synthesizing the path of DSLs to the answer from the extracted specific node.

3.3.2 Abductive Reasoning Learning

The symbolic ARC solver utilizes the concept of abductive reasoning for the learning stage. Consisting of two trainable components, the *synthesizer*, and *specifier*, the learning process unfolds in reverse order of inference, starting from the *synthesizer*. Starting from each node of the input graph treated as a leaf and extending up to the root of the output grid, exploring all possible paths through the search tree. The edges of the search tree are composed of *transformation DSLs*, originating from the leaves and branching towards the root by applying each *transformation DSL*. The search halts when the tree reaches a certain depth, at which point the paths connected to the root become candidates for core knowledge used in the inference stage. At this stage, it identifies all possible (*node, path*) pairs, where the path represents the sequence of *transformation DSLs*. Applying this path to the nodes in the pair yields the desired output targeted during the process. An example of

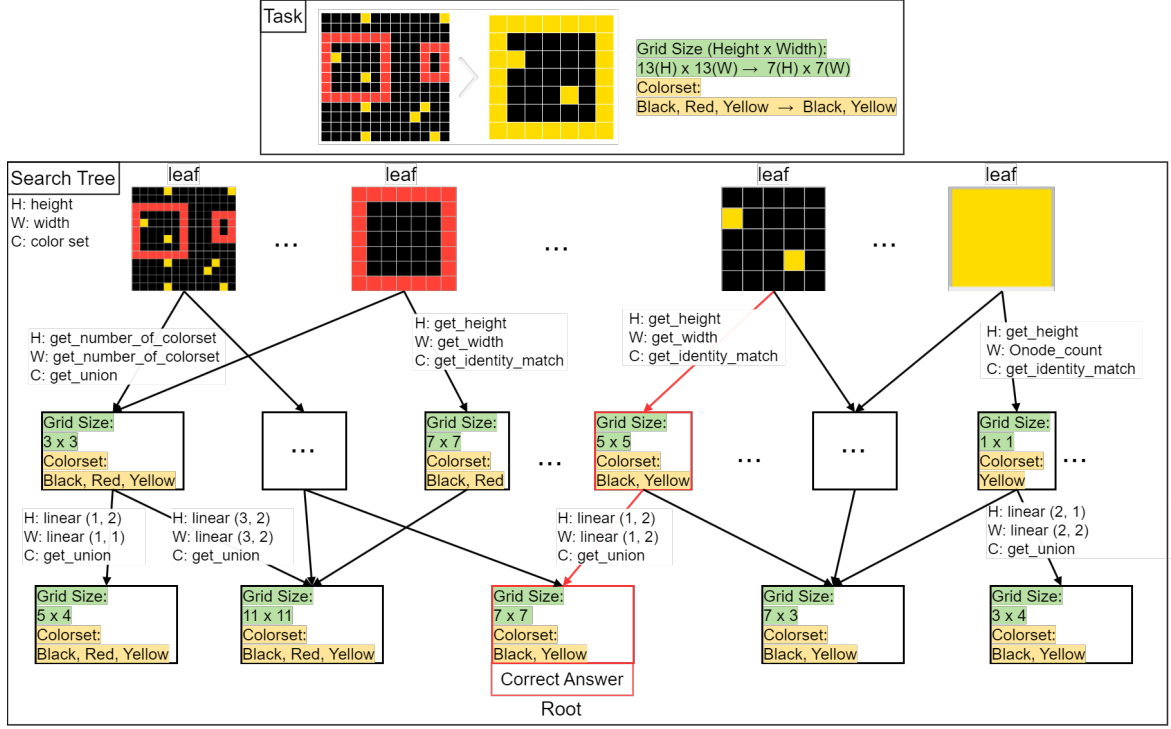


Figure 3.4: Training session of the *synthesizer* and its expanded search tree. The task is to find the largest rectangle in the input and change the color to its interior single-pixel color. First, all nodes generated from the input are placed at the top (leaf) of the search tree, with the output node at the bottom (root), commented as Correct Answer in the figure. Then, *transformation DSLs* are applied to draw paths. This example shows *Synthesizer-10* targeting grid size and color set. Among the DSLs used, *get_height* returns the height of the node, *get_width* returns the width, *get_number_of_colorset* returns the number of colors other than the background, and *Onode_count* returns the number of included objects. The *linear(a, b)* DSL performs the transformation $ax + b$ on the previous value x . *get_union* returns the union of colors between the previous node and the target node, while *get_identity_match* returns the color set of the previous node. The path that reaches the root is highlighted in red, forming a pair with the corresponding leaf.

the *synthesizer*'s training can be found in Figure 3.4, which corresponds to the setup in Section 4.1 and is an example of *Synthesizer-10*.

Next, the *specifier* is a component that extracts specific node(s), from a given knowledge graph. It learns a method to uniquely identify nodes from the candidate pairs generated by the *synthesizer*. For instance, if the node is specified as "the largest object composed of adjacent pixels of the same color," the *specifier* generates a function that identifies the minimal features in the knowledge graph that uniquely designate this node, returning them as constraints. The objective of this process is to traverse the knowledge graph and find the smallest subset that satisfies the criteria of the given node, such as "same color," "adjacent pixels," and "largest." The example can be found in Figure 3.5. Consequently, the constraint becomes a function that extracts node(s) in the knowledge graph, ultimately generating a hypothesis in the form of a pair (*constraints*, *path*). This hypothesis can be applied to all knowledge graphs of the same task by the

$$path(constraints(KG)) \Rightarrow prediction$$

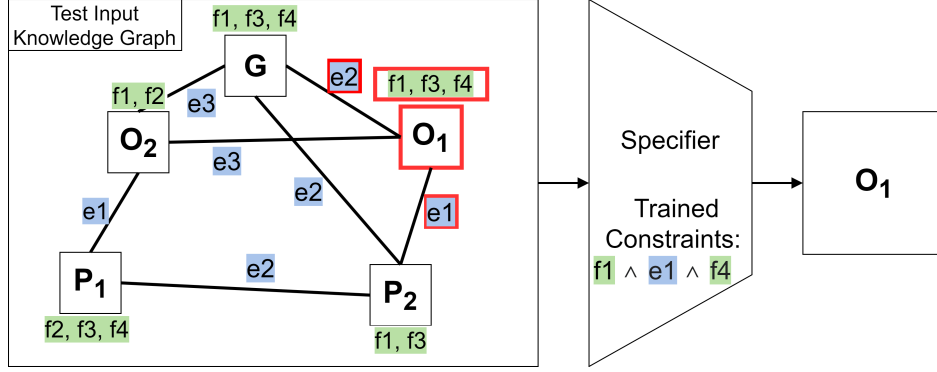


Figure 3.5: The figure demonstrates the learning process of the *specifier* for extracting the O_1 node. Here, G represents Gnodes, O_n represents Onodes, P_n represents Pnodes, e_n represents edges, and f_n represents features. In the knowledge graph on the left, the *specifier* identifies unique conditions (e_n and f_n information) that can uniquely specify the O_1 node, bundles these conditions into constraints, and returns them.

returning the predicted answer.

After obtaining a set of possible hypotheses from the observations of the first example pair, the final solution is adopted through the process of evaluating whether these hypotheses can consistently explain other observations. Due to the nature of ARC problems, observations are highly limited by the number of example pairs and exhibit characteristics of few-shot learning. By applying hypotheses to the given pairs and iteratively selecting only those hypotheses that correctly derive the answers, the remaining hypotheses are adopted as the final solution for this task. This solution ensures that our observations are well explained.

Chapter 4. Experiment and Results

The primary objective of this experiment is to leverage a knowledge graph (KG) and domain-specific language (DSL) to solve tasks within the Abstraction and Reasoning Corpus (ARC). Below are the hypotheses raised in this paper:

- **H1:** The knowledge graphs effectively encapsulate symbolic knowledge, facilitating human-like problem-solving and enhancing performance.
- **H2:** The number of transformation DSLs is positively correlated with the performance of the symbolic ARC solver.

4.1 Experimental Setup

For the evaluation of the DSL-based symbolic ARC solver and knowledge graphs, a small set of simple and basic transformation DSLs was applied. The answers (outputs) of ARC problems consist of three elements: 1) the size of the grid, 2) the color set of the grid, and 3) the contents of the grid. Though all three are crucial, predicting and modifying the target values of the first two hold significant importance as they represent steps inherent in human problem-solving of ARC tasks. Therefore, we prioritized these aspects in our experimental setup, focusing primarily on them and enabling the utilization of minimal and straightforward *transformation DSLs* during the synthesis process. For the color set, all colors appearing in the correct grid must be matched with the predicted value to be considered as the correct answer, while for the grid’s size, separate integer values for height and width were predicted. In the generation of the knowledge graph, a total of 22 property DSLs were employed.

4.2 Result

4.2.1 Comparison of Solver Performance with and without the Use of Knowledge Graph

Figure 4.1 presents the accuracy scores of a solver’s performance on different target values, comparing the use of a knowledge graph against not using one. For each target on the x-axis, the solver’s accuracy is consistently higher when utilizing the knowledge graph. In particular, when not utilizing the knowledge graph, a significant decrease in the prediction performance of C and HWC can be observed. This indicates the crucial role of symbolic information contained in the knowledge graph in predicting the color set. The solver achieves nearly perfect accuracy for the H, W, and HW with the knowledge graph. These results confirm that the use of knowledge graphs effectively enhances performance, supporting **H1** by demonstrating their capability to encapsulate symbolic knowledge and facilitate human-like problem-solving.

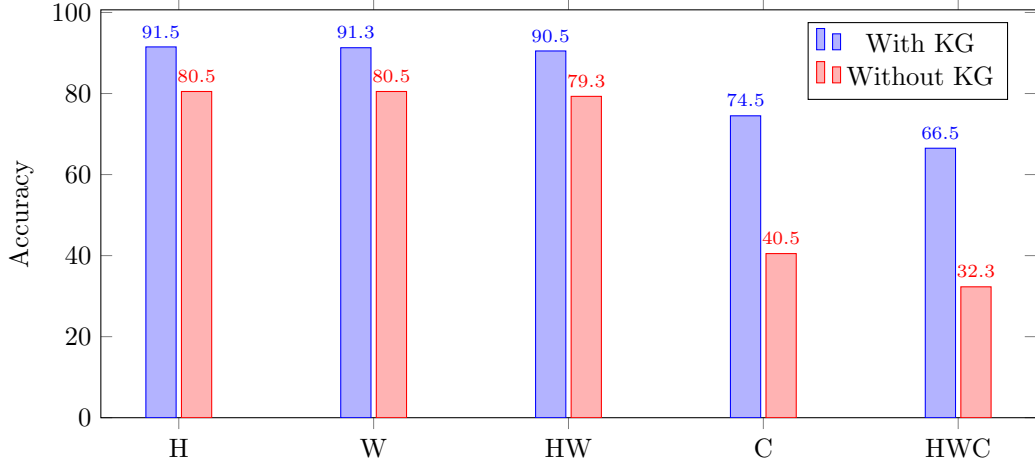


Figure 4.1: Accuracy score comparison of solver with and without utilizing knowledge graph on each target. Here, "KG" refers to the knowledge graph. The targets assessed are Height (H), Width (W), Color (C), and their combinations: Height and Width (HW), and Height, Width, and Color (HWC).

4.2.2 Differences in Performance by Size of Synthesizer

To explore the relationship between the number of transformation DSLs and accuracy, two *synthesizers* of different sizes were prepared, both with a depth limit of 2 for the search tree. The results show that *Synthesizer-10* consistently achieves higher accuracy across all categories compared to *Synthesizer-5*. Notably, in the HWC category, *Synthesizer-10* outperforms *Synthesizer-5* by over three times. These findings support H2, confirming that the number of *transformation DSLs* is positively correlated with the performance of the symbolic ARC solver. Additionally, they suggest that employing more sophisticated and diverse *transformation DSLs* enhances the model's accuracy and its potential to predict content.

Table 4.1: The comparison presented here delves into the accuracy scores of solvers utilizing different *synthesizer* sizes. *Synthesizer-10*, employing 10 *transformation DSLs*, is contrasted with *Synthesizer-5*, which utilizes only 5. For details on DSL adopted by each *synthesizer*, see Figure 3.3.

	Synthesizer-10			Synthesizer-5		
	Correct	Incorrect	Accuracy (%)	Correct	Incorrect	Accuracy (%)
H	366	34	91.5	209	191	52.25
W	365	35	91.25	203	197	50.75
C	299	101	74.75	176	224	44
HW	362	38	90.5	197	203	49.25
HWC	266	134	66.5	84	316	21

Chapter 5. Conclusion

We introduced a framework for ARC problem-solving, integrating knowledge graph conversion and abductive reasoning learning with a symbolic ARC Solver. This approach, inspired by human thought processes, offers systematic, interpretable, and scalable solutions. Leveraging knowledge graphs, we decode ARC tasks symbolically, providing crucial insights for inferring problem rules. Impressively, even with a naive *synthesizer* using limited transformation DSLs, our framework achieves high accuracy in predicting grid sizes (90.5%) and color sets (74.5%). Furthermore, as DSLs increase, we anticipate significant performance improvement, potentially extending to grid content prediction.

Summary

Abductive Symbolic Solver on Abstraction and Reasoning Corpus

ARC는 시각적 추론 작업을 포함하는 문제 모음으로, 인간은 이를 관찰과 가설을 통해 해결하고, 해결 방법을 논리적으로 설명할 수 있습니다. 하지만 이전의 많은 접근법들은 단순히 그리드의 변환에만 집중했기 때문에, 이러한 접근만으로는 AI가 인간처럼 합리적이고 논리적인 해결책을 제공하기에 충분하지 않았습니다. 인간이 시각적 추론 작업을 해결하는 과정은 가설과 검증을 기반으로 하는 귀추논증(abductive reasoning) 과정이라는 결론에 도달하였고 이를 적용한 모델을 제안합니다. 먼저, 2차원 배열 형태의 원본 데이터를 인간의 선험 지식이 담긴 도메인 특화 언어를 활용하여 지식 그래프로 표현합니다. 입력과 출력 한 쌍은 하나의 거대한 지식그래프로 표현되며 각 그래프는 4개의 다른 계층에 각기 다른 종류의 노드를 가지고 있으며 노드들 간의 관계는 심볼릭 정보로서 엣지에 담기게 됩니다. 이후 이 지식 그래프를 이용하여 솔루션을 생성하게 되는데 이는 핵심 지식을 추출하는 과정과 해답을 합성하는 과정으로 나누어집니다. 두 컴포넌트는 귀추 논증을 통해 학습되며 각 문제에 따라 올바른 해답을 도출하는 것을 목표로 합니다. 여기서 도메인 특화 언어는 솔루션 탐색 공간을 제한하고, 합리적인 중간 과정을 제공하는 데 도움이 되며 해답을 구성하는 요소가 됩니다. 이 모델은 적은 수의 기초적인 도메인 특화 언어만을 활용하고도 정답 그리드의 색깔, 크기를 예측하는 작업에서 높은 성능을 보였습니다. 또한 적용된 도메인 특화 언어의 개수에 따라 성능의 향상을 보임으로써 보다 정교한 확장을 그리드 내용의 예측으로 나아갈 수 있는 가능성을 확인하였고 궁극적으로 강인공지능으로 나아가는 초석이 되기를 기대합니다.

감 사 의 글

지난 1년 반 동안 즐겁게 연구를 시작하고, 이와 같은 멋진 논문으로 결실을 맺을 수 있도록 지도해주신 김선동 교수님께 깊은 감사의 말씀을 드립니다. 또한 귀중한 시간을 내어 논문 심사를 해주신 소순범 교수님께도 감사드립니다. 이 외에도 함께 연구를 진행하며 많은 도움을 준 이석기, Liyew, 그리고 연구실 동료분들께 감사의 인사를 전합니다.

약 력

이 름: 임민택

생 년 월 일: 1999년 6월 21일

출 생 지: 대한민국 인천광역시

학 력

2015. 3. – 2018. 2. 양서고등학교

2018. 2. – 2024. 8. 광주과학기술원 전기전자컴퓨터공학부 (학사)

References

1. Pan Lu, Hritik Bansal, Tony Xia, Jiacheng Liu, Chunyuan Li, Hannaneh Hajishirzi, Hao Cheng, Kai-Wei Chang, Michel Galley, and Jianfeng Gao. Mathvista: Evaluating mathematical reasoning of foundation models in visual contexts. *arXiv preprint arXiv:2310.02255*, 2023.
2. François Chollet. On the measure of intelligence, 2019.
3. Jean Raven. Raven progressive matrices. In *Handbook of nonverbal assessment*, pages 223–237. Springer, 2003.
4. Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C Lawrence Zitnick, and Devi Parikh. Vqa: Visual question answering. In *Proceedings of the IEEE international conference on computer vision*, pages 2425–2433, 2015.
5. Deepanway Ghosal, Vernon Toh Yan Han, Chia Yew Ken, and Soujanya Poria. Are language models puzzle prodigies? algorithmic puzzles unveil serious challenges in multimodal reasoning. *arXiv preprint arXiv:2403.03864*, 2024.
6. Top-quarks. Arc-solution. <https://github.com/top-quarks/ARC-solution>, 2021.
7. Michael Hodel. arc-dsl. <https://github.com/michaelhodel/arc-dsl>, 2022.
8. Yudong Xu, Wenhao Li, Pashootan Vaezipoor, Scott Sanner, and Elias B Khalil. Llms and the abstraction and reasoning corpus: Successes, failures, and the importance of object-based representations. *arXiv preprint arXiv:2305.18354*, 2023.
9. Seungpil Lee, Woonchang Sim, Donghyeon Shin, Sanha Hwang, Wongyu Seo, Jiwon Park, Seokki Lee, Sejin Kim, and Sundong Kim. Reasoning abilities of large language models: In-depth analysis on the abstraction and reasoning corpus. *arXiv preprint arXiv:2403.11793*, 2024.
10. Ruocheng Wang, Eric Zelikman, Gabriel Poesia, Yewen Pu, Nick Haber, and Noah D Goodman. Hypothesis search: Inductive reasoning with language models. *arXiv preprint arXiv:2309.05660*, 2023.
11. Chen Liang, Wenguan Wang, Tianfei Zhou, and Yi Yang. Visual abductive reasoning, 2022.
12. Michael Hodel. Addressing the abstraction and reasoning corpus via procedural example generation. *arXiv preprint arXiv:2404.07353*, 2024.
13. James Ainooson, Deepayan Sanyal, Joel P Michelson, Yuan Yang, and Maithilee Kunda. A neurodiversity-inspired solver for the abstraction & reasoning corpus (arc) using visual imagery and program synthesis. *arXiv preprint arXiv:2302.09425*, 2023.
14. Simon Alford, Anshula Gandhi, Akshay Rangamani, Andrzej Banburski, Tony Wang, Sylee Dandekar, John Chin, Tomaso Poggio, and Peter Chin. Neural-guided, bidirectional program search for abstraction and reasoning. In *Complex Networks & Their Applications X: Volume 1, Proceedings*

of the Tenth International Conference on Complex Networks and Their Applications *COMPLEX NETWORKS 2021 10*, pages 657–668. Springer, 2022.

15. Yudong Xu, Elias B. Khalil, and Scott Sanner. Graphs, constraints, and search for the abstraction and reasoning corpus. *arXiv preprint arXiv:2210.09880*, December 2022. Available: <https://arxiv.org/abs/2210.09880>.
16. Jaehyun Park, Jaegyun Im, Sanha Hwang, Mintaek Lim, Sabina Ualibekova, Sejin Kim, and Sundong Kim. Unraveling the arc puzzle: Mimicking human solutions with object-centric decision transformer. *arXiv preprint arXiv:2306.08204*, 2023.
17. Gyöngyi Kovács and Karen M Spens. Abductive reasoning in logistics research. *International journal of physical distribution & logistics management*, 35(2):132–144, 2005.
18. Stephen C-Y Lu and Ang Liu. Abductive reasoning for design synthesis. *CIRP annals*, 61(1):143–146, 2012.
19. Paul Thagard and Cameron Shelley. Abductive reasoning: Logic, visual thinking, and coherence. In *Logic and Scientific Methods: Volume One of the Tenth International Congress of Logic, Methodology and Philosophy of Science, Florence, August 1995*, pages 413–427. Springer, 1997.