Abstract Visual Reasoning: An Algebraic Approach for Solving Raven’s Progressive Matrices

Jingyi Xu1*, Tushar Vaidya2*, Yufei Wu2*, Saket Chandra1*, Zhangsheng Lai3* Kai Fong Ernest Chong1†
1Singapore University of Technology and Design
2Nanyang Technological University
3Singapore Polytechnic
jingyi_xu@mymail.sutd.edu.sg tushar.vaidya@ntu.edu.sg yufei002@e.ntu.edu.sg
lai_zhangsheng@sp.edu.sg {saket.chandra,ernest.chong}@sutd.edu.sg

Abstract

We introduce algebraic machine reasoning, a new reasoning framework that is well-suited for abstract reasoning. Effectively, algebraic machine reasoning reduces the difficult process of novel problem-solving to routine algebraic computation. The fundamental algebraic objects of interest are the ideals of some suitably initialized polynomial ring. We shall explain how solving Raven’s Progressive Matrices (RPMs) can be realized as computational problems in algebra, which combine various well-known algebraic subroutines that include: Computing the Gröbner basis of an ideal, checking for ideal containment, etc. Crucially, the additional algebraic structure satisfied by ideals allows for more operations on ideals beyond set-theoretic operations.

Our algebraic machine reasoning framework is not only able to select the correct answer from a given answer set, but also able to generate the correct answer with only the question matrix given. Experiments on the I-RAVEN dataset yield an overall 93.2% accuracy, which significantly outperforms the current state-of-the-art accuracy of 77.0% and exceeds human performance at 84.4% accuracy.

1. Introduction

When we think of machine reasoning, nothing captures our imagination more than the possibility that machines would eventually surpass humans in intelligence tests and general reasoning tasks. Even for humans, to excel in IQ tests, such as the well-known Raven’s progressive matrices (RPMs) [5], is already a non-trivial feat. A typical RPM instance is composed of a question matrix and an answer set; see Fig. 1. A question matrix is a 3 × 3 grid of panels that satisfy certain hidden rules, where the first 8 panels are filled with geometric entities, and the 9-th panel is “missing”. The goal is to infer the correct answer for this last panel from among the 8 panels in the given answer set.

The ability to solve RPMs is the quintessential display of what cognitive scientists call fluid intelligence. The word “fluid” alludes to the mental agility of discovering new relations and abstractions [28], especially for solving novel problems not encountered before. Thus, it is not surprising that abstract reasoning on novel problems is widely hailed as the hallmark of human intelligence [6].

Although there has been much recent progress in machine reasoning [15, 17, 30–33, 37, 38, 46, 47], a common criticism [9, 25, 26] is that existing reasoning frameworks have focused on approaches involving extensive training, even when solving well-established reasoning tests such as RPMs. Perhaps most pertinently, as [9] argues, reasoning tasks such as RPMs should not need task-specific perfor-
performance optimization. After all, if a machine optimizes performance by training on task-specific data, then that task cannot possibly be novel to the machine.

To better emulate human reasoning, we propose what we call “algebraic machine reasoning”, a new reasoning framework that is well-suited for abstract reasoning. Our framework solves RPMs without needing to optimize for performance on task-specific data, analogous to how a gifted child solves RPMs without needing practice on RPMs. Our key starting point is to define concepts as ideals of some suitably initialized polynomial ring. These ideals are treated as the “abstract objects of study” in algebraic machine reasoning, which do not require any numerical values to be assigned to them. We shall elucidate how the RPM task can be realized as a computational problem in algebra involving ideals.

Our reasoning framework can be broadly divided into two stages: (1) algebraic representation, and (2) algebraic machine reasoning; see Fig. 2. In the first stage, we represent RPM panels as ideals, based on perceptual attribute values extracted from object detection models. In the second stage, we propose 4 invariance modules to extract patterns from the RPM question matrix.

To summarize, our main contributions are as follows:

- We reduce “solving the RPM task” to “solving a computational problem in algebra”. Specifically, we present how the discovery of abstract patterns can be realized very concretely as algebraic computations known as primary decompositions of ideals.
- In our algebraic machine reasoning framework, we introduce 4 invariance modules for extracting patterns that are meaningful to humans.
- Our framework is not only able to select the correct answer from a given answer set, but also able to generate answers without needing any given answer set.
- Experiments conducted on RAVEN and I-RAVEN datasets demonstrate that our reasoning framework significantly outperforms state-of-the-art methods.

2. Related Work

RPM solvers. There has been much recent interest in solving RPMs with deep-learning-based methods [15, 23, 32, 47, 48, 51–54]. Most methods extract features from raw RPM images using neural networks, and select answers by measuring panel similarities. Several works instead focus on generating correct answers without needing the answer set [27, 34]. To evaluate the reasoning capabilities of these methods, RPM-like datasets such as PGM [32] and RAVEN [46] have been proposed. Subsequently, I-RAVEN [12] and RAVEN-FAIR [3] are introduced to overcome a shortcut flaw in the answer set generation of RAVEN.

Algebraic methods in AI. Using algebraic methods in AI is not new. Systems of polynomial equations are commonly seen in computer vision [29] and robotics [8], which are solved algebraically via Gröbner basis computations. In statistical learning theory, methods in algebraic geometry [41] and algebraic statistics [10] are used to study singularities in statistical models [22, 42, 43, 45], to analyze generalization error in hierarchical models [39, 40], to learn invariant subspaces of probability distributions [18, 20], and to model Bayesian networks [11, 36]. A common theme in these works is to study suitably defined algebraic varieties. In deep learning, algebraic methods are used to study the expressivity of neural nets [7, 16, 24, 50]. In automated theorem proving, Gröbner basis computations are used in proof-checking [35]. Recently, a matrix formulation of first-order logic was applied to the RPM task [49], where relations are approximated by matrices and reasoning is framed as a bilevel optimization task to find best-fit matrix operators. As far as we know, methods from commutative algebra have not been used in machine reasoning.

3. Proposed Algebraic Framework

In abstract reasoning, a key cognitive step is to “discover patterns from observations”, which can be formulated con-
cretely as “finding invariances in observations”. In this section, we describe how algebraic objects known as ideals are used to represent RPM instances, how patterns are extracted from such algebraic representations, and how RPMs can be solved, both for answer selection and answer generation, as computational problems in algebra.

3.1. Preliminaries

Throughout, let \( R = \mathbb{R}[x_1, \ldots, x_n] \) be the ring of polynomials in variables \( x_1, \ldots, x_n \), with real coefficients. In particular, \( R \) is closed under addition and multiplication of polynomials, i.e., for any \( a, b \in R \), we have \( a + b, ab \in R \).

3.1.1 Algebraic definitions

Ideals in polynomial rings. A subset \( I \subseteq R \) is called an ideal if there exist polynomials \( g_1, \ldots, g_k \) in \( R \) such that

\[
I = \{ f_1 g_1 + \cdots + f_k g_k | f_1, \ldots, f_k \in R \}
\]

contains all polynomial combinations of \( g_1, \ldots, g_k \). We say that \( G = \{ g_1, \ldots, g_k \} \) is a generating set for \( I \), we call \( g_1, \ldots, g_k \) generators, and we write either \( I = (g_1, \ldots, g_k) \) or \( I = \langle G \rangle \). Note that generating sets of ideals are not unique. If \( I \) has a generating set consisting only of monomials, then we say that \( I \) is a monomial ideal. (Recall that a monomial is a polynomial with a single term.) Given ideals \( J_1 = (g_1, \ldots, g_k) \) and \( J_2 = (h_1, \ldots, h_l) \), there are three basic operations (sums, products, intersections):

\[
J_1 + J_2 := (g_1, \ldots, g_k, h_1, \ldots, h_l);
\]

\[
J_1 J_2 := (g_i h_j | 1 \leq i \leq k, 1 \leq j \leq l);
\]

\[
J_1 \cap J_2 := \{ r \in R : r \in J_1 \text{ and } r \in J_2 \}.
\]

Most algebraic computations involving ideals, especially “advanced” operations (e.g. primary decompositions), require computing their Gröbner bases as a key initial step. More generally, Gröbner basis computation forms the backbone of most algorithms in algebra; see Appendix A.2.

Primary decompositions. In commutative algebra, primary decompositions of ideals are a far-reaching generalization of the idea of prime factorization for integers. Its importance to algebraists cannot be overstated. Informally, every ideal \( J \) has a decomposition \( J = J_1 \cap \cdots \cap J_s \) as an intersection of finitely many primary ideals. This intersection is called a primary decomposition of \( J \), and each \( J_j \) is called a primary component of the decomposition. In the special case when \( J \) is a monomial ideal, there is an unique minimal primary decomposition with maximal monomial primary components [2]; We denote this unique set of primary components by \( \text{pd}(J) \). See Appendix A.3 for details.

3.1.2 Concepts as monomial ideals

We define a concept to be a monomial ideal of \( R \). In particular, the zero ideal \( \langle 0 \rangle \subseteq R \) is the concept “null”, and could be interpreted as “impossible” or “nothing”, while the ideal \( \langle 1 \rangle = R \) is the concept “conceivable”, and could be interpreted as “possible” or “everything”. Given a concept \( J \subseteq R \), a monomial in \( J \) is called an instance of the concept. For example, \( x_{\text{black}} \cdot x_{\text{square}} \) is an instance of \( \langle x_{\text{square}} \rangle \) (the concept “square”). For each \( x_i \), we say \( \langle x_i \rangle \subseteq R \) is a primitive concept, and \( x_i \) is a primitive instance.

Theorem 3.1. There are infinitely many concepts in \( R \), even though there are finitely many primitive concepts in \( R \). Furthermore, if \( J \subseteq R \) is a concept, then the following hold:

(i) \( J \) has infinitely many instances, unless \( J = \langle 0 \rangle \).

(ii) \( J \) has a unique minimal generating set consisting of finitely many instances, which we denote by mingen(I).

(iii) If \( J \neq \langle 1 \rangle \), then \( J \) has a unique set of associated concepts \( \{ P_1, \ldots, P_k \} \), together with a unique minimal primary decomposition \( J = J_1 \cap \cdots \cap J_k \), such that each \( J_i \) is a concept contained in \( P_i \), that is maximal among all possible primary components contained in \( P_i \) that are concepts.

See Appendix A.4 for a proof of Theorem 3.1 and for more details on why defining concepts as monomial ideals captures the expressiveness of concepts in human reasoning.

3.2. Stage 1: Algebraic representation

We shall use the RPM instance depicted in Fig 1 as our running example, to show the entire algebraic reasoning process: (1) algebraic representation; and (2) algebraic machine reasoning. In this subsection, we focus on the first stage. Recall that every RPM instance is composed of 16 panels filled with geometric entities. For our running example, each entity can be described using 4 attributes: “color”, “size”, “type”, and “position”. We also need one additional attribute to represent the “number” of entities in the panel.

3.2.1 Attribute concepts

In human cognition, certain semantically similar concepts are naturally grouped to form a more general concept. For example, concepts such as “red”, “green”, “blue”, “yellow”, etc., can be grouped to form a new concept that represents “color”. Intuitively, we can think of “color” as an attribute, and “red”, “green”, “blue”, “yellow” as attribute values.

For our running example, the 5 attributes are represented by 5 concepts (monomial ideals). In general, all possible values for each attribute are encoded as generators for the concept representing that attribute. However, for ease of explanation, we shall consider only those attribute values that are involved in Fig. 1 to explain our example:

\[
A_{\text{num}} := \{ x_{\text{one}}, x_{\text{two}} \},
\]

\[
A_{\text{pos}} := \{ x_{\text{left}}, x_{\text{right}} \},
\]

\[
A_{\text{type}} := \{ x_{\text{triangle}}, x_{\text{square}}, x_{\text{pentagon}}, x_{\text{hexagon}}, x_{\text{circle}} \},
\]

\[
A_{\text{color}} := \{ x_{\text{white}}, x_{\text{gray}}, x_{\text{gray}}, x_{\text{black}} \},
\]

\[
A_{\text{size}} := \{ x_{\text{small}}, x_{\text{avg}}, x_{\text{large}} \}.
\]

Let \( L := \{ \text{num}, \text{pos}, \text{type}, \text{color}, \text{size} \} \) be the set of attribute labels, and let \( A_{\text{all}} := \bigcup_{L \in L} A_L \). Initialize the ring
\( R := \mathbb{R}[A_{\text{all}}] \) of all polynomials on the variables in \( A_{\text{all}} \) with real coefficients. For each \( \ell \in \mathcal{L} \), let \( J_{\ell} \) be the concept \( \langle A_{\ell} \rangle \subseteq R \). These concepts, which we call \textit{attribute concepts}, are task-specific. We assume humans tend to discover and organize complex patterns in terms of attributes. Thus for pattern extraction, we shall use the inductive bias that a concept representing a pattern is deemed meaningful if it is in some attribute concept.

### 3.2.2 Representation of RPM panels

In order to encode the RPM images algebraically, we first need to train perception modules to extract attribute information directly from raw images. One possible approach for perception, as used in our experiments, is to train 4 RetinaNet models (each with a ResNet-50 backbone) separately for all 4 attributes except “number”, which can be directly inferred by counting the number of bounding boxes.

After extracting attribute values for entities, we can represent each panel as a concept. For example, the top-left panel of the RPM in Fig. 1 can be encoded as the concept \( \langle \text{pentagon} \rangle \subseteq R \), which represents each panel as a concept. For example, the top-left panel can be encoded as \( \langle \text{pentagon} \rangle \subseteq R \), which represents each panel as a concept.

### 3.3.3 Proposed invariance modules

Our 4 proposed invariance modules are: (1) intra-invariance module, (2) inter-invariance module, (3) compositional invariance module, and (4) binary-operator invariance module.

1. **Intra-invariance module** extracts patterns where the set of values for some attribute within concept \( J_{\ell} \) remains invariant over all \( i \). First, we define \( J_{+} := J_{1} + \cdots + J_{k} \) and \( J_{\cap} := J_{1} \cap \cdots \cap J_{k} \); see Section 3.1.1. Intuitively, \( J_{+} \) and
$J_1$, are concepts that capture information about the entire sequence $J_1, \ldots, J_k$ in two different ways. Next, we compute the common primary components of $J_1$ and $J_1$ that are contained in attribute concepts. Finally, we return the attributes associated to these common primary components: 

$$P_{\text{comm}}(\{J_1, \ldots, J_k\}) := \{\text{attr} \in L | \exists I \in \text{pd}(J_1) \cap \text{pd}(J_k), I \subseteq \langle A_{\text{attr}} \rangle\}.$$ 

2. Inter-invariance module extracts patterns arising from the set difference between $\text{pd}(J_1)$ and $\text{pd}(J_1)$. Therefore, we check for the invariance of these extracted patterns across multiple sequences. The extracted set of patterns is:

$$P_{\text{intra}}(\{J_1, \ldots, J_k\}) := \{\text{attr} \in L | \exists I \subseteq \langle A_{\text{attr}} \rangle \forall I \in \mathcal{I}\},$$

where $\mathcal{I}$ is a set of concepts, and “$-$” refers to set difference. We omit $\text{pd}(J_1)$ so that we do not overcount the patterns already extracted in the previous module. Informally, for each pair $(\text{attr}, \mathcal{I})$, the concepts in $\mathcal{I}$ can be interpreted as those “primary” concepts that correspond to at least one of $J_1, \ldots, J_k$, that do not correspond to all of $J_1, \ldots, J_k$, and that are contained in $\langle A_{\text{attr}} \rangle$.

3. Compositional invariance module extracts patterns arising from invariant attribute values in the following new sequence of concepts:

$$[J_1, \ldots, J_k] := [f^{k-1}(J_1), f^{k-2}(J_2), \ldots, f(J_{k-1}), J_k],$$

where $f$ is some given function. Intuitively, for such patterns, there are some attributes whose values are invariant in $[f(J_i), J_{i+1}]$ for all $i = 1, \ldots, k-1$. By checking the intersection of primary components of the concepts in the new sequence, the extracted set of patterns is given by:

$$P_{\text{comp}}([J_1, \ldots, J_k]) := \{\text{attr} \in L | \exists I \subseteq \langle A_{\text{attr}} \rangle \forall I \in \mathcal{I}\},$$

The given function used for the RPM task is $f_{\text{next}}(\cdot \Delta)$, where $\Delta$ represents the number of steps; see Section 3.3.4.

4. Binary-operator module extracts numerical patterns, based on a given real-valued function $g$ on concepts, and a given set $\Lambda$ of binary operators. The extracted patterns are:

$$P_{\text{binary}}(J_i) := \{\bigotimes \in \mathcal{O} | \bigotimes = \otimes_1 \otimes \cdots \otimes_{k-2} \otimes g(J_1) \otimes \cdots \otimes g(J_{k-1}) = g(J_k)\}.$$ 

3.4.4 Extracting row-wise patterns

Given a concept matrix $J = [J_{11}, \ldots, J_{32}]$, how do we extract the patterns from its $i$-th row? We first begin by extracting the common position values among all 8 panels:

$$\text{comPos}(J) := \{p \in A_{\text{pos}} | \exists I \subseteq \bigcup_{J \in J} \text{pd}(J), p \in I\}.$$ 

For each common position $p \in \text{comPos}(J)$, we generate two new concept matrices $J^{(p)}_1$ and $J^{(p)}_3$, such that:

- Each concept $J^{(p)}_1$ in $J^{(p)}_1$ is generated by the unique generator in $J_{11}$ that is divisible by $p$;
- Each concept $J^{(p)}_3$ in $J^{(p)}_3$ is generated by all generators in $J_{11}$ that are not divisible by $p$.

(Recall that generators of a concept are polynomials.) Informally, we are splitting each panel in the RPM image into 2 panels, one that contains only the entity in the common position $p$, and the other that contains all remaining entities not in position $p$. This step allows us to reason about rules that involve only a portion of the panels.

Consequently, if $\text{comPos}(J) = \{p_1, \ldots, p_k\}$, then we can extend the single concept matrix into a list of concept matrices $[J, J^{(p_1)}_1, J^{(p_1)}_3, \ldots, J^{(p_k)}_1, J^{(p_k)}_3]$.

For each concept matrix $J$ from the extended list, we consider its $i$-th row $J_i = [J_{i1}, J_{i2}, J_{i3}]$ (left-to-right) and extract patterns from $J_i$ via the 4 modules from Section 3.3.3. Let $P(J_i) := \{\text{attr} \in L | \exists I \subseteq \langle A_{\text{attr}} \rangle \forall I \in \mathcal{I}\},$ where the union ranges over all concept matrices $J$ in the extended list, i.e. $J \in [J, J^{(p_1)}_1, J^{(p_1)}_3, \ldots, J^{(p_k)}_1, J^{(p_k)}_3]$. Note that $P_{\text{binary}}(J)$ can be regarded as all the patterns extracted from the $i$-th row of the original concept matrix $J$. If instead $J = [J_{11}, \ldots, J_{32}]$ is a list containing 9 concepts, then we can define $P_{\text{binary}}(J)$ analogously.

Algorithm 1 Answer selection.

**Inputs:** Concept matrix $J = [J_{11}, \ldots, J_{32}]$, and associated answer set $[J_{\text{ans1}}, \ldots, J_{\text{ans8}}].$

1. Initialize comPattern $= \{0, \ldots, 0\} \times 8$.
2. Compute $P_{\text{ans}}(J) := P_{\text{binary}}(J) \cap P_{\text{binary}}(J) \cap P_{\text{binary}}(J) \cap P_{\text{binary}}(J)$.
3. for all $i$ from 1 to 8 do
4. $J_i \leftarrow J_{\text{ans1}}, \ldots, J_{\text{ans8}}$
5. Compute $P_{\text{ans}}(J_i)$.
6. comPattern[index] $\leftarrow P_{\text{ans}}(J_i) \cap P_{\text{ans}}(J_i)$
7. return answer index $i \leftarrow \text{argmax}_{i'} \text{comPattern}[i']$.

3.4. Solving RPMs

3.4.1 Answer selection

In Section 3.3.4, we described how row-wise patterns can be extracted using the 4 invariance modules. Thus, a natural approach for answer selection is to determine which answer option, when inserted in place of the missing panel, would maximize the number of patterns that are common to all three rows. Consequently, answer selection is reduced to a simple optimization problem; see Algorithm 1.

3.4.2 Answer generation

Since our algebraic machine reasoning framework is able to extract common patterns that are meaningful to humans, hidden in the raw RPM images, it provides a new way to generate answers without needing a given answer set. This is similar to a gifted human who is able to solve the RPM task, by first recognizing the patterns in the first two rows, then inferring what the missing panel should be. Intuitively, we are applying “inverse” operations of the 4 invariance modules to generate the concept representing the missing panel; see Algorithm 2 for an overview.
Briefly speaking, for a given RPM concept matrix $J$, we first compute the common patterns among the first two rows via $\mathcal{P}_{1,2}(J) := \mathcal{P}^{\text{all}}_{1}(J) \cap \mathcal{P}^{\text{all}}_{2}(J)$; see (1). Each element in $\mathcal{P}_{1,2}(J)$ is a pair $(K, J)$, where $K$ is a common pattern (for rows 1 and 2) specific to one attribute, and $J$ is the corresponding concept matrix. (This represents the difficult step of pattern discovery by a gifted human.) Then, we go through all common patterns to compute the attribute values for the missing 9th panel. (This represents a routine consistency check of the discovered patterns; see Appendix B.2 for full algorithmic details, and B.3 for an example.)

In general, when integrating all the attribute values for $J_{3,3}$ derived from the patterns in $\mathcal{P}_{1,2}(J)$, it is possible that entities (i) have multiple possible values for a single attribute; or (ii) have missing attribute values. Case (i) occurs when there are multiple patterns extracted for a single attribute, while case (ii) occurs when there are no non-conflicting patterns for this attribute. For either case, we randomly select an attribute value from the possible values.

Algorithm 2 Answer generation.

Inputs: Concept matrix $J = [J_{1,3} \ldots J_{3,3}]$.

1: for $(K, J) \in \mathcal{P}^{\text{all}}_{1}(J) \cap \mathcal{P}^{\text{all}}_{2}(J)$ do // see (1)
2: if $[J_{3,1}, J_{3,2}]$ does not conflict with pattern $K$ then  
3: Compute attribute value for $J_{3,3}$ using pattern $K$.
4: Collect all the above attribute values for $J_{3,3}$.
5: while $\exists$ unique value for some attribute of an entity do
6: Randomly choose one valid attribute value.
7: Generate ideal $J_{3,3} \subseteq R$.
8: return $J_{3,3}$ and the corresponding image.

4. Discussion

Algebraic machine reasoning provides a fundamentally new paradigm for machine reasoning beyond numerical computation. Abstract notions in reasoning tasks are encoded very concretely as ideals, which are computable algebraic objects. We treat ideals as “actual objects of study”, and we do not require numerical values to be assigned to them. This means our framework is capable of reasoning on more qualitative or abstract notions that do not naturally have associated numerical values. Novel problem-solving, such as the discovery of new abstract patterns from observations, is realized concretely as computations on ideals (e.g. computing the primary decompositions of ideals). In particular, we are not solving a system of polynomial equations, in contrast to existing applications of algebra in AI (cf. Section 2). Variables (or primitive instances) are not assigned values. We do not evaluate polynomials at input values.

Theory-wise, our proposed approach breaks new ground. We established a new connection between machine reasoning and commutative algebra, two areas that were completely unrelated previously. There is over a century’s worth of very deep results in commutative algebra that have not been tapped. Could algebraic methods be the key to tackling the long-standing fundamental questions in machine reasoning? It was only much more recently in 2014 that Léon Bottou [4] suggested that humans should “build reasoning capabilities from the ground up”, and he speculated that the missing ingredient could be an algebraic approach.

Why use ideals to represent concepts? Why not use sets? Why not use symbolic expressions, e.g. polynomials? Intuitively, we think of a concept as an “umbrella term” consisting of multiple (potentially infinitely many) instances of the concept. Treating concepts as merely sets of instances is inadequate in capturing the expressiveness of human reasoning. A set-theoretic representation system with finitely many “primitive sets” can only have finitely many possible sets in total. In contrast, we proved that we can construct infinitely many concepts from only finitely many primitive concepts (Theorem 3.1). This agrees with our intuition that humans are able to express infinitely many concepts from only finitely many primitive concepts. The main reason is that the “richer” algebraic structure of ideals allows for significantly more operations on ideals, beyond set-theoretic operations. See Appendix A.4 for further discussion.

Why is our algebraic method fundamentally different from logic-based methods, e.g. those based on logic programming? At the heart of logic-based reasoning is the idea that reasoning can be realized concretely as the resolution (or inverse resolution) of logical expressions. Inherent in this idea is the notion of satisfiability; cf. [14]. Intuitively, we have a logical expression, usually expressed in a canonical normal form, and we want to assign truth values (true or false) to literals in the logical expression, so that the entire expression is satisfied (i.e. truth value is “true”); see Appendix C.1 for more discussion. In fact, much of the exciting progress in automated theorem proving [1, 13, 19, 21, 44, 55] is based on logic-based reasoning.

In contrast, algebraic machine reasoning builds upon computational algebra and computer algebra systems. At the heart of our algebraic approach is the idea that reasoning can be realized concretely as solving computational problems in algebra. Crucially, there is no notion of satisfiability. We do not assign truth values (or numerical values) to concepts in $R = k[x_1, \ldots, x_n]$. In particular, although primitive concepts $\langle x_1 \rangle, \ldots, \langle x_n \rangle$ in $R$ correspond to the variables $x_1, \ldots, x_n$, we do not assign values to primitive concepts. Instead, ideals are treated as the “actual objects of study”, and we reduce “solving a reasoning task” to “solving non-numerical computational problems involving ideals”. Moreover, our framework can discover new patterns beyond the actual rules of the RPM task; see Section 5.2.

In the RPM task, we have attribute concepts representing “position”, “number”, “type”, “size”, and “color”; these are concepts that categorize the primitive instances according...
to their semantics, into what humans would call attributes. Intuitively, an attribute concept combines certain primitive concepts together in a manner that is “meaningful” to the task. For example, \( (x_{\text{white}}, x_{\text{gray}}, x_{\text{black}}) \) is “more meaningful” than \( (x_{\text{white}}, x_{\text{circle}}, x_{\text{large}}) \) as a “simpler” or “generalized” concept, since we would treat \( x_{\text{white}}, x_{\text{gray}}, x_{\text{black}} \) as instances of a single broader “color” concept.

Notice that the primitive concepts correspond precisely to the prediction classes of our object detection models. Such prediction classes are already implicitly identified by the available data. Consequently, our method is limited by what our perception modules can perceive. For other tasks, e.g. where text data is available, entity extraction methods can be used to identify primitive concepts. Note also that our method requires prior knowledge, since there is no training step for the reasoning module. This limitation can be mitigated if we replace user-defined functions on concepts with trainable functions optimized via deep learning.

In general, the identification of attribute concepts is task-specific, and the resulting reasoning performance would depend heavily on these identified attribute concepts. Effectively, our choice of attribute concepts would determine the inductive bias of our reasoning framework: As we decompose a concept \( J \) into “simpler” concepts (i.e. primary components in \( \text{pd}(J) \)), only those “simpler” concepts contained in attribute concepts are deemed “meaningful”. Concretely, let \( J, J' \subseteq R \) be concepts such that \( \text{pd}(J) = \{J_1, \ldots, J_k\} \) and \( \text{pd}(J') = \{J'_1, \ldots, J'_l\} \), i.e. \( J, J' \) have minimal primary decompositions \( J = J_1 \cap \cdots \cap J_k \) and \( J' = J'_1 \cap \cdots \cap J'_l \), respectively. We can examine their primary components and extract out those primary components (between the two primary decompositions) that are contained in some common attribute concept. For example, if \( A \) is an attribute concept of \( R \) such that \( J_1 \subseteq A \) and \( J'_1 \subseteq A \), then \( J \) and \( J' \) share a “common pattern”, represented by the attribute concept \( A \).

### 5. Experiment results

To show the effectiveness of our framework, we conducted experiments on the RAVEN [46] and I-RAVEN datasets. In both datasets, RPMs are generated according to 7 configurations. We trained our perception modules on 4200 images from I-RAVEN [12] (600 from each configuration), and used them to predict attribute values of entities. The average accuracy of our perception modules is 96.24%. For both datasets, we tested on 2000 instances for each configuration. Overall, our reasoning framework is fast (7 hours for 14000 instances on a 16-core Gen11 Intel i7 CPU processor). See Appendix B for full experiment details.

#### 5.1. Comparison with other baselines

Table 1 compares the performance of our method with 10 other baseline methods. We use the accuracies on I-RAVEN reported in [12, 52] for methods 1-7, and the accuracies on RAVEN reported in [46, 52] for methods 1-5. All the other accuracies are obtained from the original papers. As a reference, we also include the human performance on the RAVEN dataset (i.e. not I-RAVEN) as reported in [46].

#### 5.2. Ambiguous instances and new patterns

Although our method outperforms all baselines, some instances have multiple answer options that are assigned equal top scores by our framework. Most of these cases occur due to the discovery of (i) “accidental” unintended rules (e.g. Fig. 3); or (ii) new patterns beyond the actual rules in the dataset (e.g. Fig. 4). Case (i) occurs because in the design of I-RAVEN, at most one rule is assigned to each attribute. Interestingly, case (ii) reveals that our framework is able to discover completely new patterns that are not originally designed as rules for I-RAVEN. In Fig. 4, the new pattern discovered is arguably very natural to humans.

---

Table 1. Performance on I-RAVEN/RAVEN. We report mean accuracy, and the accuracies for all configurations: Center, 2x2Grid, 3x3Grid, Out-InCenter, Out-InGrid, Left-Right, and Up-Down.
5.3. Evaluation of answer generation

Every RPM instance is assumed to have a single correct answer from the given answer set. However, there are multiple other possible images that are also acceptable as correct answers. For example, images modified from the given correct answer, via random perturbations of those attributes that are not involved in any of the rules (e.g. entity angles in the I-RAVEN dataset), are also correct. All these distinct correct answers (images) can be encoded algebraically as the same concept, based on prior knowledge of which raw perceptual attributes are relevant for the RPM task. Hence, to evaluate the answer generation process proposed in Section 3.4.2, we will directly evaluate the generated concepts.

Let \( J = \langle e_1, \ldots, e_k \rangle \) and \( J' = \langle e'_1, \ldots, e'_\ell \rangle \) be concepts representing the ground truth answer and our generated answer, respectively. Here, each \( e_i \) (or \( e'_i \)) is a monomial of the form \( x_i^{\mathrm{pos}} y_i^{\mathrm{type}} z_i^{\mathrm{color}} u_i^{\mathrm{size}} \), and represents an entity described by 4 attributes. Motivated by the well-known idea of Intersection over Union (IoU), we propose a new similarity measure between \( J \) and \( J' \). In order to define analogous notions of “intersection” and “union”, we first pair each \( e_i \) with \( e'_j \) if \( x_i^{\mathrm{pos}} = x'_j^{\mathrm{pos}} \) (i.e. same “position” values). This pairing is well-defined, since the “position” values of the entities in any panel are uniquely determined. Hence we can group all entities in \( J \) and \( J' \) into 3 sets:

\[
S_1 := \{ (e_i, e'_j) \mid e_i \in J, e'_j \in J', x_i^{\mathrm{pos}} = x'_j^{\mathrm{pos}} \};
\]

\[
S_2 := \{ e_i \in J \mid \nexists e'_j \in J' \text{ such that } (e_i, e'_j) \in S_1 \};
\]

\[
S_3 := \{ e'_j \in J' \mid \nexists e_i \in J \text{ such that } (e_i, e'_j) \in S_1 \}.
\]

We can interpret \( S_1 \) and \( S_2 \cup S_3 \) as analogous notions of the “intersection” and “union” of \( J \) and \( J' \), respectively. Thus, we define our similarity measure as follows:

\[
\phi(J, J') := \frac{\sum_{(e_i, e'_j) \in S_1} \phi(e_i, e'_j)}{|S_1| + |S_2| + |S_3|}; \tag{2}
\]

\[
\phi(e_i, e'_j) := \frac{1}{4} \sum_a \mathbb{I}(x_i^{(a)} = x_j'^{(a)}); \tag{3}
\]

where in (3), \( a \) ranges over the 4 attributes in \{pos, type, color, size\}. Here, \( \phi(e_i, e'_j) \) is the similarity score between \( e_i \) and \( e'_j \), measured by the proportion of common variables.

The overall average similarity score of the generated answers is 67.7\%. Note that within a panel, some attribute values such as “size”, “color” and “position”, may be totally random for 2x2Grid, 3x3Grid, Out-InGrid (e.g. as shown in Fig. 3). Hence, achieving high similarity scores for such cases would inherently require task-specific optimization and knowledge of how the data is generated. We assume neither. This could explain why our overall similarity score is lower than our answer selection accuracy.

For examples of generated images, see Appendix B.5.

6. Conclusion

Algebraic machine reasoning is a reasoning framework that is well-suited for abstract reasoning. In its current form, we have used primary decompositions as a key algebraic operation to discover abstract patterns in the RPM task, via the invariance modules that we have specially designed to mimic human reasoning. The idea that “discovering common patterns” can be realized concretely as “computing primary decompositions” is rather broad, and could potentially be applied to other inferential reasoning tasks.

More generally, our algebraic approach opens up new possibilities of tapping into the vast literature of commutative algebra and computational algebra. There are numerous algebraic operations on ideals (ideal quotients, radicals, saturation, etc.) and algebraic invariants (depth, height, etc.) that have not been explored in machine reasoning (or even in AI). Can we use them to tackle other reasoning tasks?
References


