

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

Jingyi Xu^{1*} Tushar Vaidya^{2◦*} Yufei Wu^{2◦*} Saket Chandra¹ Zhangsheng Lai^{3◦} Kai Fong Ernest Chong^{1†}

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

Outline. As part of supplementary materials for our main paper titled “Abstract Visual Reasoning: An Algebraic Approach for Solving Raven’s Progressive Matrices”, we provide further details, organized into the following three appendices:

- Appendix **A** gives further algebraic details on algebraic machine reasoning.
- Appendix **B** gives further experiment details for how we solved RPMs.
- Appendix **C** gives further discussion on algebraic machine reasoning, including its potential societal impact and its differences from logic-based reasoning.

A. Further Algebraic Details

This appendix serves as an in-depth elaboration of the algebraic ideas presented in Section 3. For definitions and relevant terminology, we have kept this appendix as self-contained as possible. For proofs of “standard” algebraic results, we provide references to explicit proofs in textbooks wherever possible, and provide full proof details otherwise. For ease of reading, it would be helpful if the reader has at least some prior exposure to the basics of commutative algebra, e.g. at the level of undergraduate abstract algebra. For a gentle introduction to commutative algebra and computational algebra, we recommend [7]. For a more detailed treatment of the subject, see [9].

Throughout the appendix, let $R := \mathbb{k}[x_1, \dots, x_n]$ be a polynomial ring on n variables x_1, \dots, x_n , with coefficients in a field \mathbb{k} . In the main paper, we assumed for simplicity that $\mathbb{k} = \mathbb{R}$; in fact, \mathbb{k} can be chosen to be any field. In particular, \mathbb{k} need not be an algebraically closed field.¹

^{*}Equal contributions. [†] Corresponding author.

[◦] This work was done when the author was previously at SUTD.

¹ When dealing with a system of polynomial equations in n variables, where the coefficients of the polynomial equations are treated as values in \mathbb{k} , any solution to this system of polynomial equations is a point in \mathbb{k}^n . It is frequently assumed that \mathbb{k} is algebraically closed, so that the analysis of the solution space is easier. Similarly, when considering the algebraic variety

(Note that \mathbb{C} is an algebraically closed field, while \mathbb{R} is not.) This means that we allow $\mathbb{k} = \mathbb{R}$. We also allow \mathbb{k} to be a field with non-zero characteristic (e.g. a finite field $\mathbb{Z}/p\mathbb{Z}$ for some prime number p). The specific choice of field \mathbb{k} will be irrelevant for the details discussed in this appendix, since none of the algebraic computations required for reasoning involve numerical computations on the coefficients of the terms of polynomials. For concreteness, the reader may choose to assume henceforth that $\mathbb{k} = \mathbb{R}$, without any loss of generality.

For our algebraic notation, we use “ $\langle g_1, \dots, g_k \rangle$ ” to denote an ideal generated by g_1, \dots, g_k , in contrast to some other authors who use the notation “ (g_1, \dots, g_k) ” instead to refer to ideals. For us, we reserve the use of round brackets to denote tuples, e.g. “ (g_1, \dots, g_k) ” is a k -tuple. All other algebraic notation we use is consistent with the present-day notation used in the commutative algebra literature.

The rest of Appendix **A** is organized as follows:

- Appendix **A.1** gives further details and relevant results concerning concepts (i.e. monomial ideals), including algebraic operations on ideals, and with a focus on the special case of monomial ideals.
- Appendix **A.2** gives a detailed treatment of Gröbner basis theory, the key “workhorse” underlying most of the algorithms in computational algebra and commutative algebra.
- Appendix **A.3** gives more details on primary ideals and primary decompositions, as well as a discussion on *how primary decompositions relate to the inductive bias for algebraic machine reasoning*.
- Appendix **A.4** completes the proof of Theorem 3.1 from the main paper, and gives details on *why defining concepts as monomial ideals captures the expressive-*

associated to such a system of polynomial equations, the analysis of the algebraic variety is easier when the base field \mathbb{k} is algebraically closed. In our paper, we do not deal with or solve systems of polynomial equations, and we will not work with algebraic varieties. Hence, we will not assume that \mathbb{k} is algebraically closed.

A.1. Concepts, monomial ideals, ideal operations

Let \mathbb{N} be the set of non-negative integers. For each $\alpha = (\alpha_1, \dots, \alpha_n) \in \mathbb{N}^n$, let \mathbf{x}^α denote the monomial $x_1^{\alpha_1} x_2^{\alpha_2} \dots x_n^{\alpha_n}$ in R . By definition, a *monomial* in R is a polynomial of the form \mathbf{x}^α , i.e. the leading coefficient of a monomial is always 1, the multiplicative identity of the field \mathbb{k} . The *degree* of \mathbf{x}^α , which we denote by $\deg(\mathbf{x}^\alpha)$, is the sum $\alpha_1 + \dots + \alpha_n$. For example, $\deg(x_1^2 x_2^3 x_3^7) = 12$.

A.1.1 Algebraic properties of concepts

Let \mathcal{M}_R be the set of all monomials in R . We begin with the observation that the polynomial ring R is a \mathbb{k} -vector space with basis \mathcal{M}_R . This means that every polynomial p in R can be written as $p = \sum_{m \in \mathcal{M}_R} a_m \cdot m$, where each a_m is a uniquely determined scalar in \mathbb{k} , called the *coefficient* of the monomial m in p . A monomial $m \in \mathcal{M}_R$ is called a *monomial of p* if its coefficient a_m is non-zero.

Proposition A.1. [7, Chap. 2.4, Lemma 3] *Let p be a polynomial of R , and let J be a monomial ideal of R . Then p is an element of J if and only if every monomial m of p is an element of J .*

Although \mathcal{M}_R is infinite (since it contains monomials of all degrees), it is a well-known algebraic fact that every ideal I of R has a *finite* generating set; this is famously known as Hilbert’s basis theorem (see [7, Chap. 2.5, Thm. 4] or [9, Thm. 1.2] for a proof). For monomial ideals, we have the following stronger result:

Proposition A.2. [32, Lemma 1.2] *Every monomial ideal of R has a **unique** minimal generating set consisting of finitely many monomial generators.*

To make sense of Proposition A.2, note that a generating set \mathcal{G} for an ideal J of R is called *minimal* if no strictly smaller subset $\mathcal{G}' \subsetneq \mathcal{G}$ generates J . This means if we are given a monomial ideal J of R , and any subset $\mathcal{M} \subseteq \mathcal{M}_R$ of monomials that generates J , i.e. $J = \langle \mathcal{M} \rangle$, then Proposition A.2 tells us that we can always find a minimal subset $\mathcal{M}' \subseteq \mathcal{M}$ (with respect to set inclusion) that generates the same ideal J , such that this minimal subset \mathcal{M}' is uniquely determined, independent of our choice of \mathcal{M} . This unique minimal generating set of monomials shall be denoted by $\text{mingen}(J)$.

Given any monomial ideal J of R , a monomial contained in $\text{mingen}(J)$ is called a *minimal monomial generator* of J . Since a concept is defined to be a monomial ideal of R , we can analogously define a *minimal instance* of J to be a minimal monomial generator of J . Thus, a concept is generated by all of its minimal instances.

Recall that there are three basic operations on ideals: sums, products, and intersections. Let $J_1 = \langle g_1, \dots, g_k \rangle$ and $J_2 = \langle h_1, \dots, h_\ell \rangle$ be any two ideals of R . Then the sum $J_1 + J_2$, product $J_1 J_2$ and intersection $J_1 \cap J_2$ are ideals defined as follows:

$$\begin{aligned} J_1 + J_2 &:= \langle g_1, \dots, g_k, h_1, \dots, h_\ell \rangle; \\ J_1 J_2 &:= \langle \{g_i h_j \mid 1 \leq i \leq k, 1 \leq j \leq \ell\} \rangle; \\ J_1 \cap J_2 &:= \{r \in R : r \in J_1 \text{ and } r \in J_2\}. \end{aligned}$$

In particular, $J_1 \cap J_2$ is an ideal, since for any polynomials r_1, \dots, r_s in $J_1 \cap J_2$, the polynomial combination $r_1 p_1 + \dots + r_s q_s$ (for any polynomials p_1, \dots, p_s in R) is by definition contained in both J_1 and J_2 .

A useful property of these basis operations on ideals is that monomial ideals remain as monomial ideals under such operations:

Proposition A.3. *If J_1 and J_2 are concepts of R , then $J_1 + J_2$, $J_1 \cap J_2$ and $J_1 J_2$ are also concepts of R . Furthermore, we always have $J_1 J_2 \subseteq J_1 \cap J_2$.*

Proof. Clearly, $J_1 + J_2$ and $J_1 J_2$ are monomial ideals, since the generators specified in their definitions are monomials. Also, for each $1 \leq i \leq k$, $1 \leq j \leq \ell$, the element $g_i h_j$ is by definition contained in both J_1 and J_2 , which means that every minimal monomial generator of $J_1 J_2$ is an element of $J_1 \cap J_2$, thus we get $J_1 J_2 \subseteq J_1 \cap J_2$.

Finally, we shall prove that $J_1 \cap J_2$ is a monomial ideal. Let $\mathcal{M}' := \mathcal{M}_R \cap J_1 \cap J_2$, and define the monomial ideal $J' := \langle \mathcal{M}' \rangle \subseteq R$. By definition, $\mathcal{M}' \subseteq J_1 \cap J_2$, hence $J' \subseteq J_1 \cap J_2$. To show that the converse $J' \supseteq J_1 \cap J_2$ holds, consider an arbitrary element $p \in J_1 \cap J_2$, and write $p = \sum_{m \in \mathcal{M}_R} a_m \cdot m$, where each $a_m \in \mathbb{k}$ is the coefficient of the monomial m . Note that $p \in J_1$ and $p \in J_2$. Since J_1 and J_2 are both monomial ideals, Proposition A.1 implies that every monomial m of p (i.e. with coefficient $a_m \neq 0$) is an element of $J_1 \cap J_2$, and so must also be contained in $\mathcal{M}' = \mathcal{M} \cap J_1 \cap J_2$. Thus, $p \in \langle \mathcal{M}' \rangle = J'$. \square

A.1.2 A hierarchy of different kinds of concepts

Recall that the first stage of algebraic machine reasoning is algebraic representation. For the RPM task, we extracted attribute information from raw RPM images, and encoded each RPM panel algebraically as a concept. These concepts that we have used for our algebraic representation of RPM panels are particularly nice: Their minimal monomial generators are squarefree. Recall that a monomial \mathbf{x}^α in R (for some $\alpha = (\alpha_1, \dots, \alpha_n) \in \mathbb{N}^n$) is called *squarefree* if every exponent α_i is either 0 or 1, or equivalently, if $\mathbf{x}^\alpha \notin \langle x_1^2, \dots, x_n^2 \rangle$. (Informally, a monomial m is squarefree if the variables appearing in m do not have “higher powers”, i.e. none of the variables have exponents

≥ 2 .) Monomial ideals generated by squarefree monomials are called *squarefree monomial ideals*, and such ideals are heavily studied in commutative algebra for their rich properties [32, 40]. Hence, we are motivated to define the following notion of “basic” concepts.

Definition A.4. A concept J in R is called *basic* if $J \neq R$, and if all of its minimal instances (i.e. minimal monomial generators) are squarefree.

We shall later prove in Theorem A.13 that basic concepts have a particularly nice structure for extracting patterns.

We shall also introduce the notion of “simple” concepts. Consider the ideal $J_1 := \langle x_{\text{black}}x_{\text{square}}, x_{\text{white}}x_{\text{circle}} \rangle$, which is a basic concept that represents “black square or white circle”. Such a concept is contained in the intuitively simpler concept $J_2 := \langle x_{\text{black}}, x_{\text{white}} \rangle$ representing “black or white”. Note that J_2 is generated by two primitive instances x_{black} and x_{white} . To capture this intuition that “black or white” is simpler than “black square or white circle”, we define a concept J in R to be *simple* if J is generated by a non-empty set of primitive instances in R , i.e. a non-empty subset of $\{x_1, \dots, x_n\}$. Equivalently, a concept is *simple* if it is the sum of primitive concepts. Notice that all simple concepts are basic, since a primitive instance is squarefree. As we shall see later in Appendix A.3.2, every basic concept can be “decomposed” into multiple simple concepts.

To summarize, we have the following hierarchy of different kinds of concepts:

$$\{\text{concepts}\} \supseteq \{\text{basic concepts}\} \supseteq \{\text{simple concepts}\} \supseteq \{\text{primitive concepts}\}.$$

A.2. Gröbner basis theory

In this subsection, we review basic terminology and several elementary results in Gröbner basis theory. For a quick introduction to what a Gröbner basis of an ideal is, see [42].

Why is Gröbner basis theory important? In essence, if we want to *compute* the answer to a computational problem involving ideals, we typically need, as a key initial step, to compute the Gröbner bases of the input ideals. Subsequent computational steps would usually involve working directly with the computed Gröbner bases, rather than the originally given generating sets for the input ideals. To avoid the reader thinking that Gröbner basis computations are for “more advanced” operations on ideals, we highlight here that computing the intersections of ideals already relies on Gröbner basis computations: Given ideals J_1, J_2 of R with generating sets $\mathcal{G}_1, \mathcal{G}_2$ respectively, it is already not trivial to compute a generating set for the intersection $J_1 \cap J_2$.

A.2.1 What is a Gröbner basis?

Recall that R is a \mathbb{k} -vector space with basis \mathcal{M}_R , where \mathcal{M}_R denotes the set of monomials in R . The starting point in Gröbner basis theory is the sorting of the set \mathcal{M}_R .

Definition A.5. A *monomial order* on R is a well-order on \mathcal{M}_R , denoted by $<$, such that for all monomials m_1, m_2, m in \mathcal{M}_R satisfying $m \neq 1$, the following condition holds:

$$\text{If } m_1 > m_2, \text{ then } mm_1 > mm_2 > m_2.$$

Given a monomial order $<$, and any polynomial p in R , we can always write $p = c_1m_1 + \dots + c_km_k$ for some non-zero scalars c_1, \dots, c_k in \mathbb{k} , and some monomials m_1, \dots, m_k in \mathcal{M}_R that are sorted in descending order $m_1 > \dots > m_k$ with respect to the monomial order $<$. These scalars and monomials are uniquely determined given p and $<$. Recall that the monomials m_1, \dots, m_k are precisely all the monomials of p , and each scalar c_i is the coefficient of m_i in p . The *initial monomial* of p (with respect to $<$), denoted by $\text{in}_<(p)$, is the largest monomial m_1 . By default, we define $\text{in}_<(0) = 0$.

Suppose $\mathcal{A} = \{p_1, \dots, p_k\}$ is a finite set of polynomials in R . Write $p = (\sum_{i=1}^k f_i p_i) + q$ where f_1, \dots, f_k, q are polynomials in R , such that $\text{in}_<(p) \geq \text{in}_<(f_i p_i)$ for all $1 \leq i \leq k$, and such that none of the monomials of q are divisible by any monomial contained in $\{\text{in}_<(p_1), \dots, \text{in}_<(p_k)\}$. Such an expression always exists and is called a *standard expression* of p with respect to \mathcal{A} . Any such polynomial q is called a *remainder* of p with respect to \mathcal{A} . If 0 is a remainder of p with respect to \mathcal{A} , then we say that p *reduces to zero* with respect to \mathcal{A} . Note that standard expressions and remainders of p (with respect to \mathcal{A}) are in general not unique.

Let $f, g \in R$ be polynomials that are not both the zero polynomial. The *S-pair* of f and g (with respect to $<$) is the polynomial

$$\mathcal{S}(f, g) := \frac{c_g \text{in}_<(g)}{\text{gcd}(\text{in}_<(f), \text{in}_<(g))} f - \frac{c_f \text{in}_<(f)}{\text{gcd}(\text{in}_<(f), \text{in}_<(g))} g,$$

where $\text{gcd}(\text{in}_<(f), \text{in}_<(g))$ denotes the greatest common divisor of $\text{in}_<(f)$ and $\text{in}_<(g)$, and c_f (resp. c_g) is the coefficient of $\text{in}_<(f)$ (resp. $\text{in}_<(g)$) in f (resp. g). We use the convention that $\text{gcd}(f, 0) = f$ for all non-zero polynomials $f \in R$. By default, define $\mathcal{S}(0, 0) = 0$. It is straightforward to check that if $\text{gcd}(\text{in}_<(f), \text{in}_<(g)) = 1$, then $\mathcal{S}(f, g)$ always reduces to zero with respect to $\{f, g\}$.

For any ideal I of R , the monomial ideal $\text{in}_<(I) := \langle \{\text{in}_<(f) \mid f \in I\} \rangle$, generated by the initial monomials of all elements in I , is called the *initial ideal* of I (with respect to $<$). A *Gröbner basis* for I (with respect to $<$) is a **finite** set \mathcal{G} of elements in I , such that $\text{in}_<(I)$ is generated by the set $\{\text{in}_<(f) \mid f \in \mathcal{G}\}$. Gröbner bases for I (with respect to any monomial order) always exist, and every Gröbner basis for I must generate I . If \mathcal{G} is a Gröbner basis for I , then the remainder of p with respect to \mathcal{G} is unique. In particular, $p \in I$ if and only if p reduces to zero with respect to \mathcal{G} .

Consider any finite generating set \mathcal{A} for the ideal I . *Buchberger’s criterion* says that \mathcal{A} is a Gröbner basis for

I (with respect to some given monomial order $<$) if and only if $S(f, g)$ reduces to zero for every $f, g \in \mathcal{A}$. This criterion yields *Buchberger's algorithm*, which is an algorithm to compute a Gröbner basis \mathcal{G} for any ideal I of S , given a finite generating set \mathcal{A} for I as input. Buchberger's algorithm is described as follows: Starting with the input generating set \mathcal{A} , compute the remainders of the S -pairs $S(f, g)$ with respect to \mathcal{A} , for all generators $f, g \in \mathcal{A}$; if any such remainder q is non-zero, then insert the polynomial q into \mathcal{A} and repeat the process. This process must terminate after finitely many steps, and the final \mathcal{A} obtained after the process terminates is a Gröbner basis for I , which contains the initially given generating set \mathcal{A} . Note that by definition, all S -pairs of pairs of elements in the final Gröbner basis \mathcal{A} must reduce to zero with respect to \mathcal{A} .

A.2.2 Computational subroutines in algebra

Computational problems in algebra can be solved using computer algebra systems. Well-known computer algebra systems (e.g. Maple, Magma, Mathematica, MATLAB; cf. [7, Appendix C]) are frequently used to solve systems of polynomial equations, while more specialized systems (e.g. CoCoA [1], GAP [11], Macaulay2 [16], SageMath [44], Singular [8]) are more commonly used by algebraists to solve computational problems where ideals are the “actual objects of study”, i.e. not involving the assignment of numerical values to variables.

Gröbner bases are always defined with respect to some given monomial order $<$. In many computer algebra systems (e.g. CoCoA [1], Macaulay2 [16], SageMath [44], Singular [8]), the default monomial order for Gröbner basis computations is the graded reverse-lexicographic order, also known as the degree reverse-lexicographic order.

Definition A.6. The *graded reverse-lexicographic order* (grevlex order) on R , which we denote by $<_{r\ell}$, is a monomial order on R , such that for any distinct monomials $\mathbf{x}^\alpha, \mathbf{x}^\beta$ in R , where $\alpha = (\alpha_1, \dots, \alpha_n)$ and $\beta = (\beta_1, \dots, \beta_n)$, we have $\mathbf{x}^\alpha >_{r\ell} \mathbf{x}^\beta$ if and only if either $\deg(\mathbf{x}^\alpha) > \deg(\mathbf{x}^\beta)$; or $\deg(\mathbf{x}^\alpha) = \deg(\mathbf{x}^\beta)$, and $\alpha_i < \beta_i$ for the largest $1 \leq i \leq n$ such that $\alpha_i \neq \beta_i$.

For example, if $n = 3$ (i.e. $R = \mathbb{k}[x_1, x_2, x_3]$), then the degree 3 monomials in R , arranged in the grevlex order, is given as follows:

$$\begin{aligned} x_1^3 &>_{r\ell} x_1^2 x_2 >_{r\ell} x_1 x_2^2 >_{r\ell} x_2^3 >_{r\ell} x_1^2 x_3 >_{r\ell} x_1 x_2 x_3 \\ &>_{r\ell} x_2^2 x_3 >_{r\ell} x_1 x_3^2 >_{r\ell} x_2 x_3^2 >_{r\ell} x_3^3. \end{aligned}$$

Notice that in our definition for the grevlex order, the degree 1 monomials (which are precisely the n variables in R) are arranged as $x_1 >_{r\ell} x_2 >_{r\ell} \dots >_{r\ell} x_n$. Hence, by permuting these variables, there are $n!$ distinct grevlex orders that can be defined on a polynomial ring with n variables.

In general, we would need to specify a linear order on the n variables in R , then define the grevlex order on top of the specified linear order for these degree 1 monomials. For this paper, we assume implicitly and without loss of generality that $x_1 > x_2 > \dots > x_n$ for all monomial orders $<$ on R .

Ideal membership. Let $p \in R$ be a polynomial, let I be an ideal of R , and suppose that \mathcal{G} is a Gröbner basis for I . How do we decide if p is contained in I ? We can compute the remainder of p with respect to \mathcal{G} . This remainder is zero if and only if p is contained in I ; see [7, Chap. 2.8].

Ideal containment. Let J_1, J_2 be ideals of R , and suppose $\mathcal{G}_1, \mathcal{G}_2$ are Gröbner bases for J_1, J_2 , respectively. How do we decide if $J_1 \subseteq J_2$? Suppose $\mathcal{G}_1 = \{g_1, \dots, g_k\}$. Then $J_1 \subseteq J_2$ if and only if g_i reduces to zero with respect to \mathcal{G}_2 for all $1 \leq i \leq k$.

Intersection of ideals. Let J_1, J_2 be ideals of R , and suppose that $\mathcal{G}_1 = \{g_1, \dots, g_k\}, \mathcal{G}_2 = \{h_1, \dots, h_\ell\}$ are Gröbner bases for J_1, J_2 , respectively. What is a possible generating set \mathcal{G} for $J_1 \cap J_2$? We can compute \mathcal{G} algorithmically as follows: First, construct an “extended” polynomial ring $R' := \mathbb{k}[x_1, \dots, x_n, t]$ that has an additional variable t , so that R is a subring of R' , and consider any monomial order $<$ such that any monomial divisible by t is larger than all monomials not divisible by t . Construct a new ideal $K := \langle tg_1, \dots, tg_k, (1-t)h_1, \dots, (1-t)h_\ell \rangle$ contained in the new ring R' , compute a Gröbner basis \mathcal{G}' for K with respect to the described monomial order $<$, then compute the subset $\mathcal{G} \subseteq \mathcal{G}'$ comprising all those elements $g' \in \mathcal{G}'$ such that $\text{in}_<(g')$ is not divisible by t . This final set \mathcal{G} is not just a generating set for $J_1 \cap J_2$; it is in fact a Gröbner basis for $J_1 \cap J_2$. For proof details on why this algorithm works, see [7, Chap. 4.3].

For a more detailed treatment of Gröbner basis theory, including its rich connections to algorithms in computational algebra and the numerous computational problems in commutative algebra, see [9, Chap. 15].

A.3. Primary ideals and primary decompositions

The notion of primary decompositions arises as a far-reaching generalization of the idea of prime factorization in integers, and has deep implications in number theory, group theory, and algebraic geometry. For example, the prime factorization of integers, the classification of finite abelian groups, and the decomposition of algebraic varieties into their irreducible components, can be interpreted as three seemingly different decomposition theorems that are three special cases of the celebrated Lasker–Noether theorem; see [9, Thm. 3.10] for a statement of this theorem in its full generality.

In our context, the Lasker–Noether theorem tells us that every ideal of R can be decomposed into an intersection of finitely many “primary” ideals. Such a decomposition is called a “primary decomposition”, and there are several

known algorithms for computing the primary decompositions of ideals in polynomial rings; see [10, 13, 39] for state-of-the-art algorithms. In our paper, the computation of primary decompositions of concepts (i.e. monomial ideals) is a key ingredient for our algebraic machine reasoning framework.

A.3.1 What exactly is a primary decomposition?

To rigorously define primary decompositions and related notions, we first need to introduce more technical algebraic terminology.

Prime ideals and primary ideals. Let J be an ideal of R . We say that J is *proper* if J is not the entire polynomial ring R , i.e. if $J \neq \langle 1 \rangle$. We say that J is a *prime* ideal if J is proper ideal that satisfies the following condition:

If J_1, J_2 are ideals of R such that $J_1 J_2 \subseteq J$, then either $J_1 \subseteq J$ or $J_2 \subseteq J$ (or both).

A prime ideal P of R is called an *associated prime* of J , if there exists an element $r \in R$ satisfying $r \notin J$, such that $P = \{p \in R \mid pr \in J\}$. We say the ideal J is *primary* if J has exactly one associated prime; if this unique associated prime is P , then we say that J is a *P -primary* ideal. There is another useful equivalent definition for primary ideals; cf. [9, Prop. 3.9]. An ideal J of R is *primary* if it is a proper ideal that satisfies the following condition:

If x, y are elements of R such that $xy \in J$, then either $x \in J$, or $y^k \in J$ for some $k \in \mathbb{Z}^+$.

Primary decompositions and associated primes. We now state (a special case of) the Lasker–Noether theorem:

Theorem A.7 ([9, Thm. 3.10]). *Every proper ideal J of R can be written as an intersection $J = J_1 \cap \dots \cap J_k$ for some finitely many primary ideals J_1, \dots, J_k of R . Furthermore, if each J_i is a P_i -primary ideal for a prime ideal P_i of R , then every associated prime of J must be one of the prime ideals P_1, \dots, P_k .*

A *primary decomposition* of $J \subseteq R$ is a decomposition of J as an intersection of finitely many primary ideals of R . Theorem A.7 says that primary decompositions for proper ideals of R always exist. Note that in general, an ideal J could have multiple primary decompositions.

Consider a primary decomposition $J = J_1 \cap \dots \cap J_k$ of a proper ideal J . The primary ideals J_1, \dots, J_k are called the *primary components* of this primary decomposition.² If

²When dealing with primary components, it is implicitly assumed that any primary component is referenced with respect to some given primary decomposition. Since primary decompositions are not unique, merely mentioning that J_i is a primary component of J without having any context of the primary decomposition that J_i is a primary component of, would be ambiguous.

each J_i is a P_i -primary ideal for some prime ideal P_i , then we say that J_i is a P_i -primary component (of this primary decomposition). If the number k of primary components in the primary decomposition $J = J_1 \cap \dots \cap J_k$ is the smallest among all possible primary decompositions of J , then we say that $J = J_1 \cap \dots \cap J_k$ is a *minimal* primary decomposition. In this case, the associated primes P_1, \dots, P_k must all be distinct (see [9, Thm. 3.10c]). Note that minimal primary decompositions are in general not unique.

The *radical* of an ideal $J \subseteq R$ is the ideal defined by

$$\sqrt{J} := \{r \in R \mid r^k \in J \text{ for some } k \in \mathbb{Z}^+\}.$$

The next proposition relates radicals to the associated primes of primary components in a primary decomposition.

Proposition A.8. *Let J be a proper ideal of R , and let $J = J_1 \cap \dots \cap J_k$ be a minimal primary decomposition of J . Then for each $1 \leq i \leq k$, the radical $\sqrt{J_i}$ is a prime ideal, and the primary component J_i is a $\sqrt{J_i}$ -primary ideal. Furthermore, $\sqrt{J_1}, \dots, \sqrt{J_k}$ are all the distinct associated primes of J .*

Proof. Let J'_1, J'_2 be ideals of $\sqrt{J_i}$ such that $J'_1 J'_2 \subseteq \sqrt{J_i}$. Let $j_1 \in J'_1, j_2 \in J'_2$ be arbitrarily chosen non-zero elements. By the definition of a radical, we have $(j_1 j_2)^t \in J$ for some $t \in \mathbb{Z}^+$, so since J_i is primary, we infer that either $j_1^t \in J_i$, or $j_2^{tk} \in J_i$ for some $k \in \mathbb{Z}^+$; this implies that either $j_1 \in \sqrt{J_i}$ or $j_2 \in \sqrt{J_i}$, i.e. $\sqrt{J_i}$ is a prime ideal. Thus, by [9, Cor. 2.12], $\sqrt{J_i}$ is the smallest prime ideal containing J_i , hence [9, Prop. 3.9] implies that $\sqrt{J_i}$ is the unique associated prime of J_i . Finally, [9, Thm. 3.9c] completes the proof. \square

Remark A.9. A useful consequence of Proposition A.8 is that although an ideal J of R could have multiple different minimal primary decompositions $J = J_1 \cap \dots \cap J_k$, the set $\{\sqrt{J_1}, \dots, \sqrt{J_k}\}$ of prime ideals corresponding to any minimal primary decomposition is always the same; this set is called the *set of associated primes* of J .

Remark A.10. In computer algebra systems, a primary decomposition is typically represented as a list of primary ideals. Although there are known algorithms for computing a minimal primary decomposition³ of an arbitrary proper ideal of a polynomial ring (see, e.g., [41] for several implementations in Macaulay2), we caution the reader that the computation of primary decompositions may not be supported in some computer algebra systems.

Unique minimal primary decompositions. In the special case that J is a proper monomial ideal of R , the minimal primary decompositions of J are particularly nice:

³Existing implementations of algorithms to compute primary decompositions always return minimal primary decompositions. For a fixed ideal J , it is possible to compute more than one minimal primary decomposition, e.g. in Macaulay2; see [17, 41].

Theorem A.11 ([4]; cf. [9, Ex. 3.11]). *Let J be a proper monomial ideal of R . Then there exists a minimal primary decomposition $J = J_1 \cap \dots \cap J_k$ satisfying the following:*

- *Every primary component J_i is a monomial ideal.*
- *Each primary component J_i is a P_i -primary ideal, where the prime ideal $P_i = \sqrt{J_i}$ is generated by a subset of $\{x_1, \dots, x_n\}$.*
- *Each primary component J_i is maximal among all possible monomial P_i -primary components.*

Furthermore, any minimal primary decomposition of J satisfying these properties is uniquely determined up to re-orderings of the primary components.

Succinctly, every proper monomial ideal J of R has a **unique** minimal primary decomposition with maximal monomial primary components, whose corresponding uniquely determined set $\{J_1, \dots, J_k\}$ of primary components is denoted by (J) .

In our language of algebraic machine reasoning (see, in particular, Appendix A.1.2), Theorem A.11, combined with Proposition A.8, yields the following corollary.

Corollary A.12. *Let J be a concept of R such that $J \neq \langle 1 \rangle$, i.e. J is not the “conceivable” concept. Then there exist finitely many distinct simple concepts P_1, \dots, P_k of R that are uniquely determined by J , and there exists a primary decomposition $J = J_1 \cap \dots \cap J_k$ of the concept J as an intersection of k concepts J_1, \dots, J_k , such that every radical $\sqrt{J_i}$ is the simple concept P_i . Furthermore, any such decomposition for which J_i is maximal over all possible concepts with radical $\sqrt{J_i} = P_i$, is uniquely determined.*

Primary decompositions for basic concepts. As we have mentioned in Appendix A.1.2, we encoded RPM panels algebraically as basic concepts, i.e. whose minimal instances are squarefree. The following theorem tells us that basic concepts are particularly nice, because they have unique minimal primary decompositions whose primary components are simple concepts.

Theorem A.13. *Let J be a basic concept of R . Then there exists a primary decomposition $J = J_1 \cap \dots \cap J_k$ of the concept J as an intersection of finitely many distinct simple concepts J_1, \dots, J_k . Furthermore, such a decomposition is unique up to re-orderings of the simple concepts J_1, \dots, J_k .*

Proof. Suppose $\text{mingen}(J) = \{m_1, \dots, m_k\}$. Since $J \neq \langle 1 \rangle$, none of the monomial generators m_1, \dots, m_k can be 1, so each of them has degree ≥ 1 . If all of m_1, \dots, m_k have degrees exactly 1, then J is by definition a simple concept, and we are trivially done. Thus, we can assume henceforth that there is some minimal monomial generator m_j such that $\deg(m_j) \geq 2$. This means we can write $m_j = m' m''$ for some monomials m', m'' in \mathcal{M}_R , each with degree ≥ 1 . Since J is a basic concept, all minimal monomial generators

of J must be squarefree, so m_j is a squarefree monomial, which implies that $\gcd(m', m'') = 1$. Hence, by defining $\hat{G} := \text{mingen}(J) \setminus \{m_j\}$, we can then decompose J as an intersection

$$J = \langle \hat{G} \cup \{m'\} \rangle \cap \langle \hat{G} \cup \{m''\} \rangle, \quad (1)$$

where each of the two ideals $\langle \hat{G} \cup \{m'\} \rangle$ and $\langle \hat{G} \cup \{m''\} \rangle$ in this decomposition is a squarefree monomial ideal.

Consequently, for every squarefree monomial ideal J' in any decomposition of J as an intersection of squarefree monomial ideals, we can iteratively decompose J' further as an intersection of two strictly larger squarefree monomial ideals whenever J' contains a minimal monomial generator m with degree ≥ 2 . By iteratively repeating this process, we would then get a decomposition $J = J_1 \cap \dots \cap J_k$, where J_1, \dots, J_k are squarefree monomial ideals whose minimal monomial generators all have degree 1. By definition, each such J_i is precisely a simple concept, and $\sqrt{J_i} = P_i$, therefore our assertion follows from Corollary A.12. \square

A.3.2 Inductive bias for algebraic machine reasoning

Theorem A.13 has an important consequence. It tells us that any basic concept J of R , no matter how “complicated” it may seem, can always be decomposed as an intersection of simple concepts. Simple concepts are easy to interpret, since by definition they are sums of primitive concepts.

Among all possible simple concepts (i.e. whose generating sets range over all possible non-empty subsets of $\{x_1, \dots, x_n\}$), there are certain “distinguished” simple concepts that are important for reasoning. Such “distinguished” simple concepts are task-specific, and we call them *attribute concepts*. Informally, an attribute concept is a concept representing a certain attribute of interest.

In the RPM task, recall that we have attribute concepts representing “position”, “number”, “type”, “size”, and “color”; these are simple concepts that categorize the primitive instances according to their semantics, into what humans would call attributes. For example, the “type” attribute concept $J_{\text{type}} = \langle x_{\text{triangle}}, x_{\text{square}}, x_{\text{pentagon}}, x_{\text{hexagon}}, x_{\text{circle}} \rangle$ is a simple concept generated by $x_{\text{triangle}}, x_{\text{square}}, x_{\text{pentagon}}, x_{\text{hexagon}}, x_{\text{circle}}$, which represent all possible instances of the attribute “type” in the RPM task.

The minimal instances of an attribute concept can be interpreted as “attribute values”. This means that a simple concept contained in an attribute concept would be generated by “attribute values” of a single “attribute”, and hence would inherit the same semantics as the corresponding attribute concept. For example, the simple concept $J := \langle x_{\text{triangle}}, x_{\text{pentagon}}, x_{\text{circle}} \rangle$ is contained in J_{type} and inherits the semantics of the attribute “type”. Notice that J_{type} is a

concept that is meaningful to humans (representing possible types of geometric entities), hence J would also be a concept that is similarly meaningful to humans (representing some possible types of geometric entities). After all, the minimal instances of J are all instances of the same attribute “type”.

Note that the identification of such attribute concepts is task-specific, and the resulting reasoning performance (e.g. in terms of the extracted patterns) would depend heavily on these identified attribute concepts. In particular, our choice of which simple concepts are attribute concepts would determine the *inductive bias* of our reasoning framework: After decomposing a basic concept into multiple simple concepts, only those simple concepts contained in attribute concepts are deemed “meaningful”, and we extract such simple concepts as concepts representing patterns that are meaningful to humans. Roughly speaking, when extracting common patterns with this inductive bias, our reasoning framework is “biased” towards only those “simpler” concepts that are contained in attribute concepts.

What about primitive concepts? Typically, they are self-evident and easy to define. This is because the precise formulation of any reasoning task would first require an identification of (the semantics of) the possible data attribute values, which then naturally informs us what our primitive concepts should be. For the RPM task, we had 4 perception modules for the 4 attributes “color”, “size”, “type”, “position”, so all the possible attribute values across these 4 attributes would naturally form primitive concepts.

In general, for visual reasoning tasks, the possible object classes that can be predicted by perception models would naturally take the role of primitive concepts for our reasoning framework.

A.4. Algebraic intuition for concepts

A.4.1 Proof of Theorem 3.1

For convenience to the reader, we re-state Theorem 3.1:

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 $\text{mingen}(J)$.*
- (iii) *If $J \neq \langle 1 \rangle$, then J has a unique set of associated concepts $\{P_1, \dots, P_k\}$, together with a unique minimal primary decomposition $J = J_1 \cap \dots \cap J_k$, 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.*

Proof. Since $R = \mathbb{K}[x_1, \dots, x_n]$, there are a total of n primitive concepts in R , i.e. $\langle x_1 \rangle, \dots, \langle x_n \rangle$. Since the set \mathcal{M}_R of all monomials in R is infinite, and since each $m \in \mathcal{M}_R$

generates a concept $\langle m \rangle$, there are infinitely many concepts in R . To show part (i), notice that if $J \neq \langle 0 \rangle$, then J must contain at least one monomial $m \in \mathcal{M}_R$, thus by the definition of an ideal, J contains mx_1^k for all $k \in \mathbb{N}$ and so has infinitely many instances. For part (ii), see Proposition A.2. Finally, part (iii) follows directly from Corollary A.12 and Proposition A.8. \square

A.4.2 Compatibility with human reasoning and human understanding of concepts

In cognitive science, *concepts* are the most fundamental constructs for understanding human cognition [30, 35]. There are numerous competing theories [5, 14, 21, 28, 29, 31, 37] on what “concepts” are (or should be). Even the seemingly innocuous question of whether the underlying principles (upon which these different theories are based on) are compatible or not, has been fiercely debated, and still remains unresolved today [43].

Faced with such circumstances as our backdrop, we do not seek to add to the debate by presenting a new theory on concepts (in the sense of constructs for human cognition). Rather, we seek to explain how our definition of concepts as monomial ideals adequately captures some of the richness of various aspects of concepts in human cognition. To us, our algebraic approach is grounded in the intuition that concepts are *computable*, and they represent abstractions that derive *computable meanings* from its relations to other concepts.

Compositionality. Concepts are compositional [21, 23]. A concept could be composed of multiple “simpler” concepts. Distinct concepts could share a common concept. In philosophy and in linguistics, this compositional structure for concepts is widely accepted as an important aspect of the human experience in learning new concepts [15]. For the RPM task, the algebraic representation we used to encode RPM panels is compositional. We can compose a concept in R by means of sums, products and intersections of concepts in R . We can decompose a concept as an intersection of multiple concepts (via primary decompositions). As we have discussed in the main paper, and elaborated in Appendix A.3.2, the discovery of abstract patterns is based on primary decompositions; for this algebraic approach to work, we inherently need the “compositional” property of ideals; cf. Appendix C.1.

Essence of concepts. In some theories of concepts [14, 21, 28], there is an imprecise notion of the “essence” of a concept. We could view this “essence” as the definition of the concept (e.g. consider the famous example that the concept “bachelor” is synonymous with “unmarried man”). However, more recent theories of concepts have espoused the belief that many concepts that are meaningful to humans (e.g. “knowledge”, “truth”) do not lend themselves

easily to precise definitions in terms of other concepts [12]. This lack of definitional structure in some concepts has led to the idea that concepts can be defined in terms of its constituent *features*. For example, “feathers” and “wings” can be viewed as features of the concept “bird”.

Careful distinction has to be made between *defining features* and *irrelevant features*. For example, “singing and chirping” could be a feature of the concept “bird”, but it is not a defining feature: A bird does not need to be able to sing or chirp to be considered a bird. Only the defining features of the concept would be considered part of the “essence” of that concept. However, it remains a challenge as to how the defining features of concepts can be extracted in a principled manner; see [25].

Is there an analogous notion of “essence” for concepts defined as monomial ideals? By definition, we can construct a concept in R by specifying its generating set; this is very much analogous to specifying features of the concept. Given a generating set \mathcal{G} for a concept J in R , we can compute the set $\text{mingen}(J)$ of minimal monomial generators of J . By the uniqueness of $\text{mingen}(J)$ (see Proposition A.2), this means we can interpret the generators in $\text{mingen}(J)$ as being analogous to the defining features of the concept. Generators in \mathcal{G} that are not in $\text{mingen}(J)$ are redundant for generating J , and could be interpreted as irrelevant features. Consequently, we can view $\text{mingen}(J)$ as the “essence” of concept J : Effectively, $\text{mingen}(J)$ is a “distinguished” finite set of instances that determines all the infinitely many possible instances in J ; cf. Theorem 3.1 of the main paper.

In contrast, there is no analogous notion of $\text{mingen}(J)$ in a logic-based or set-theoretic setting: For a given logical expression, there is *a priori* no “distinguished” subset of the Boolean-type variables (appearing in the logical expression), or a subset of the clauses, that would represent the entire logical expression. For any given infinite set, if we do not have any further information about its elements, then there is no natural way to assign to it a “distinguished” finite subset that still represents the entire infinite set.

Concepts with partial definitions. In the classical theory of concepts, a concept is characterized in terms of necessary and sufficient conditions for an instance to be a member of that concept [5]. A key criticism of this classical theory is that humans are still able to reason with concepts that have *partial definitions*; this has been the starting point of neoclassical theories of concepts since the 1980s [30].

In algebraic machine reasoning, our notion of concepts is well-equipped to handle partial definitions. Consider the attribute value “ x_{square} ” from the RPM task. Note that the concept $\langle x_{\text{square}} \rangle$ represents “square”, which can be defined mathematically as a geometric entity in a plane with four equal sides and four right angles. Yet, we do not require this full definition to solve the RPM task. All we need is a *partial* definition based on the condition that a

square has four sides; we do not explicitly use other necessary conditions such as “equals sides” and “equal angles”. The necessary condition “four sides” is encoded as follows: (i) the relation $f_{\text{next}}(\langle x_{\text{triangle}} \rangle) = \langle x_{\text{square}} \rangle$; (ii) the relation $f_{\text{next}}(\langle x_{\text{square}} \rangle) = \langle x_{\text{pentagon}} \rangle$; and implicitly, (iii) a perception model that distinguishes the object class associated to x_{square} from other object classes associated to the remaining variables in $\mathcal{A}_{\text{type}}$. For example, even if we do not know the defining feature that a square should have four right angles, and allow a rhombus to be an instance of concept $\langle x_{\text{square}} \rangle$, we could still solve the RPM task, provided that interior angles are not a relevant feature for solving (which is the case for both the RAVEN and I-RAVEN datasets).

At the heart of the versatility of our algebraic approach, especially in dealing with partial definitions, is the fact that we are free to impose any semantics we want on the primitive concepts. In particular, the primitive concepts in R do not necessarily need to match the traditional notion of “primitive concepts”. In the classical theory of concepts (in cognitive science), primitive concepts are primarily based on human sensory perception [28]. In contrast, what we define as primitive concepts in R is up to us, and would effectively depend on the specific reasoning task we wish to solve, as well as the perception models we have; cf. Appendix A.3.2.

Furthermore, in contrast to logic-based approaches, we do not need to “assign” truth values to all primitive concepts. For example, to determine if an instance m is contained in the concept $J = \langle x_{\text{white}}x_{\text{square}} \rangle$, we do not need to know if the entity represented by m is small or large. The “position” and “size” attribute values could be unknown. All we need for determining that $m \in J$ is to verify the *necessary* conditions, that the entity represented by m is white and is a square. We could still reason about all instances in J , even if some of these instances have certain unknown attribute values. Consequently, our reasoning framework is able to deal with *partial information*, and still make inferences about instances for which some of the truth values (true or false), on whether they are contained in each primitive concept, could be unknown. Intuitively, we do not have to assign truth values to all variables (representing statements about instance membership in primitive concepts), to be able to reason using algebraic machine reasoning.

B. Further experiment details

In this section, we provide further implementation details of our algebraic machine reasoning framework and more experimental results, organized as follows:

- Appendix B.1 gives further details on the algebraic representation stage, including the design of the attribute concepts and the performance of our object detection modules.
- Appendix B.2 describes the details on how to “in-

Attribute value set	Variable labels
$\mathcal{A}_{\text{num}} := \{x_0, \dots, x_8\}$	$x_{\text{one}}, x_{\text{two}}, \dots, x_{\text{nine}}$
$\mathcal{A}_{\text{pos}} := \{x_9, \dots, x_{31}\}$	$x(0.16, 0.16, 0.33), x(0.16, 0.5, 0.33), x(0.16, 0.83, 0.33), x(0.5, 0.16, 0.33), x(0.5, 0.5, 0.33),$ $x(0.5, 0.83, 0.33), x(0.83, 0.16, 0.33), x(0.83, 0.5, 0.33), x(0.83, 0.83, 0.33),$ $x(0.25, 0.25, 0.5), x(0.25, 0.75, 0.5), x(0.75, 0.25, 0.5), x(0.75, 0.75, 0.5),$ $x(0.42, 0.42, 0.15), x(0.42, 0.58, 0.15), x(0.58, 0.42, 0.15), x(0.58, 0.58, 0.15),$ $x(0.5, 0.25, 0.5), x(0.5, 0.75, 0.5), x(0.25, 0.5, 0.5), x(0.75, 0.5, 0.5), x(0.5, 0.5, 1.0), x_{\text{dummy}}$
$\mathcal{A}_{\text{type}} := \{x_{32}, \dots, x_{36}\}$	$x_{\text{triangle}}, x_{\text{square}}, x_{\text{pentagon}}, x_{\text{hexagon}}, x_{\text{circle}}$
$\mathcal{A}_{\text{color}} := \{x_{37}, \dots, x_{46}\}$	$x_{\#255}, x_{\#224}, x_{\#196}, x_{\#168}, x_{\#140}, x_{\#112}, x_{\#84}, x_{\#56}, x_{\#28}, x_{\#0}$
$\mathcal{A}_{\text{size}} := \{x_{47}, \dots, x_{68}\}$	$x(0.6, 0.15), x(0.7, 0.15), x(0.8, 0.15), x(0.9, 0.15),$ $x(0.4, 0.33), x(0.5, 0.33), x(0.6, 0.33), x(0.7, 0.33), x(0.8, 0.33), x(0.9, 0.33),$ $x(0.4, 0.5), x(0.5, 0.5), x(0.6, 0.5), x(0.7, 0.5), x(0.8, 0.5), x(0.9, 0.5),$ $x(0.4, 1), x(0.5, 1), x(0.6, 1), x(0.7, 1), x(0.8, 1), x(0.9, 1),$

Table 1. Five sets of variables that correspond to the attribute values of five attributes.

Attribute	Sub-sequences	Remarks
position	$\left[\begin{array}{l} x(0.16, 0.16, 0.33), x(0.16, 0.5, 0.33), x(0.16, 0.83, 0.33), x(0.5, 0.16, 0.33), x(0.5, 0.5, 0.33), \\ x(0.5, 0.83, 0.33), x(0.83, 0.16, 0.33), x(0.83, 0.5, 0.33), x(0.83, 0.83, 0.33) \end{array} \right]$ $[x(0.25, 0.25, 0.5), x(0.25, 0.75, 0.5), x(0.75, 0.25, 0.5), x(0.75, 0.75, 0.5)]$ $[x(0.42, 0.42, 0.15), x(0.42, 0.58, 0.15), x(0.58, 0.42, 0.15), x(0.58, 0.58, 0.15)]$ $x(0.5, 0.25, 0.5), x(0.5, 0.75, 0.5), x(0.25, 0.5, 0.5), x(0.75, 0.5, 0.5), x(0.5, 0.5, 1.0), x_{\text{dummy}}$	$\left. \begin{array}{l} \text{Splitting based on} \\ \text{width (the last entry)} \end{array} \right\}$ $f_{\text{next}}(\langle x \rangle, \Delta) = \langle x_{\text{dummy}} \rangle$
size	$[x(0.6, 0.15), x(0.7, 0.15), x(0.8, 0.15), x(0.9, 0.15)]$ $[x(0.4, 0.33), x(0.5, 0.33), x(0.6, 0.33), x(0.7, 0.33), x(0.8, 0.33), x(0.9, 0.33)]$ $[x(0.4, 0.5), x(0.5, 0.5), x(0.6, 0.5), x(0.7, 0.5), x(0.8, 0.5), x(0.9, 0.5)]$ $[x(0.4, 1), x(0.5, 1), x(0.6, 1), x(0.7, 1), x(0.8, 1), x(0.9, 1)]$	Splitting based on width (the last entry)

Table 2. The sub-sequences for attributes “position” and “size”, within which we cyclically order the variables to capture the sequential information of the attribute values.

versely” apply the 4 invariance modules to generate answers for RPM instances.

- Appendix B.3 gives examples of full computational details of the 4 invariance modules and the answer generation process.
- Appendix B.4 gives details on an ablation study of our algebraic machine reasoning framework to analyze the effectiveness of each invariance module in the answer selection task.
- Appendix B.5 provides examples of generated answers for each RPM configuration in I-RAVEN [18].

B.1. Algebraic representation for RPM task

B.1.1 Semantics of algebraic objects

Attribute concepts. An instance of the RAVEN [47] (or I-RAVEN [18]) dataset is given as a pair of files: an NPZ file that stores the pixel values for a raw image, and an XML file that stores the given answer and information about the attribute values of entities in each of the 16 panels. As described in Section 3.2.1 of the main paper, we first need to define five disjoint sets of variables (representing the five attributes) to encode the RPM instances algebraically. In the main paper, we only considered those attribute values that are involved in our running example for ease of explanation. However, in order to use the same algebraic representation set-up across all RPM instances, we shall define the sets of

variables that are composed of all possible attribute values.

In Table 1, we provide the list of all 69 variables (denoted by x_0, \dots, x_{68}), grouped into our five attribute sets. We shall also assign labels to these 69 variables; these labels are character strings meant for easier human interpretation of the meaning of each variable. We shall use the indices and variable labels inter-changeably, e.g. $x_0 = x_{\text{one}}$. Note that the last variable x_{dummy} in \mathcal{A}_{pos} is introduced for the purpose of subsequent use (in conjunction with the function f_{next}) in the compositional invariance module, which will be explained later. For the rest of the primitive instances, we briefly describe below how they should be interpreted:

- Instances in \mathcal{A}_{num} represent the number of geometric entities in a given panel.
- Instances in \mathcal{A}_{pos} represent the positions of sub-panels corresponding to geometric entities, in the form of 3-tuples $(x_{\text{center}}, y_{\text{center}}, \text{width})$. (The position values provided in the XML files are in the form of 4-tuples, i.e. $(x_{\text{center}}, y_{\text{center}}, \text{width}, \text{height})$. We omit the 4th entry since the width and height of each sub-panel are always equal.)
- Instances in $\mathcal{A}_{\text{type}}$ represent the types (or “shapes”) of geometric entities.
- Instances in $\mathcal{A}_{\text{color}}$ represent the grayscale pixel values of geometric entities, where $x_{\#255}$ represents “white” and $x_{\#0}$ represents “black”.
- Instances in $\mathcal{A}_{\text{size}}$ represent the absolute sizes of sub-panels, in the form of 2-tuples (relative size, width). (The size values provided in the XML files are the relative size of entities (i.e. $0.4, \dots, 0.9$) with respect to the size of the sub-panel.)

Let $\mathcal{L} := \{\text{num}, \text{pos}, \text{type}, \text{color}, \text{size}\}$ denote the set of attribute labels. Then for each attribute label $\ell \in \mathcal{L}$, we can define the attribute concept $J_\ell = \langle \mathcal{A}_\ell \rangle$.

Function f_{next} in the compositional invariance module. We have described in Section 3.3.1 of the main paper that we cyclically order the values in \mathcal{A}_ℓ for each attribute label $\ell \in \mathcal{L}$, and define a function $f_{\text{next}}(J|\Delta)$ to encode the idea of “next”. Here Δ represents the step size to map any variable $x \in \mathcal{A}_\ell$ that appears in a generator of concept J to the Δ -th variable after x ($|\Delta|$ -th variable before x for negative Δ), w.r.t. to the cyclic order on \mathcal{A}_ℓ . The cyclic order within each \mathcal{A}_ℓ is used to guarantee that $f_{\text{next}}(\langle x \rangle|\Delta)$ always lies in the same attribute concept as $\langle x \rangle$.

For attributes “number”, “type” and “color”, the variables can be naturally ordered following the sequential order as indicated in Table 1. For attributes “position” and “size”, we split the sequences of variables into multiple sub-sequences. Within each sub-sequence, we define a cyclic order that follows human intuition for the sequential information that is inherent in the semantics of the variable labels; see Table 2. For example, the “next” position of $x_{(0.75, 0.75, 0.5)}$ (representing the bottom-right sub-

panel) should cycle back to $x_{(0.25, 0.25, 0.5)}$ (representing the top-left subpanel), the “next” size of $x_{(0.9, 0.5)}$ should cycle back to $x_{(0.4, 0.5)}$, etc. Note that the last few variables $x_{(0.5, 0.25, 0.5)}, \dots, x_{(0.5, 0.5, 1.0)}$ of “position” are not in any sub-sequence, as there is no obvious sequential information on these position variables. Hence we introduce a new variable x_{dummy} so that for any position variable $x \in \{x_{(0.5, 0.25, 0.5)}, \dots, x_{(0.5, 0.5, 1.0)}\}$,

$$\begin{aligned} f_{\text{next}}(\langle x \rangle | \text{any step } \Delta) &= \langle x_{\text{dummy}} \rangle, \\ f_{\text{next}}(\langle x_{\text{dummy}} \rangle | \text{any step } \Delta) &= \langle x_{\text{dummy}} \rangle. \end{aligned}$$

Function g in the binary-operator invariance module. In Section 3.3.3 of the main paper, we introduced the binary-operator module to extract numerical patterns, based on a given real-valued function g on concepts, and a given set Λ of binary operators. In our experiments on RAVEN and I-RAVEN, we only considered attributes “number”, “color” and “size” for this invariance module, as the remaining two attributes “type” and “position” do not have obvious numerical meanings. For attribute “number”, we define the function g_{num} that maps each concept J to the number of generators in the unique minimal generating set of J :

$$g_{\text{num}}(J) := |\text{mingen}(J)|.$$

For attribute “color”, we define the function g_{color} to extract the variable index within the color value sequence in Table 1. Note that we only apply g_{color} to those concepts J for which the color variables are invariant across all generators in $\text{mingen}(J)$. Similarly for attribute “size”, we define the function g_{size} to extract the variable index within the size value sub-sequences in Table 2.

For example, consider the concept below:

$$J = \left\langle \begin{array}{l} x_{\text{two}} x_{(0.5, 0.25, 0.5)} x_{\text{square}} x_{\#255} x_{(0.6, 0.5)}, \\ x_{\text{two}} x_{(0.5, 0.75, 0.5)} x_{\text{circle}} x_{\#255} x_{(0.8, 0.5)} \end{array} \right\rangle.$$

We can compute

- $g_{\text{num}}(J) = |\text{mingen}(J)| = 2$, which indicates that J has two minimal monomial generators;
- $g_{\text{color}}(J) = 0$, which indicates that the color variable (invariant across all generators in $\text{mingen}(J)$) is the first element in the color value sequence;
- $g_{\text{size}}(J) = \text{None}$, as the size variables are different across the two generators in $\text{mingen}(J)$.

B.1.2 Object detection modules

In this subsection, we explain how we trained the object detection modules to extract the attribute values from the raw RPM images. As mentioned in the main paper, we trained 4 standard RetinaNet models (each with a ResNet-50 backbone) separately for all attributes except attribute

Class	GTs	Dets	Recall	AP
triangle	120978	120984	1.000	1.000
square	95967	95967	1.000	1.000
pentagon	121722	121722	1.000	1.000
hexagon	95017	95017	1.000	1.000
circle	118578	118578	1.000	1.000

Table 3. Object detection results for attribute “type”, evaluated on the validation set of I-RAVEN. “GTs” refers to the number of ground truth objects, “Dets” refers to the number of detected objects, and “AP” refers to the average precision score.

Class	GTs	Dets	Recall	AP
#255	114701	114743	1.000	1.000
#224	48561	48570	1.000	1.000
#196	49626	49627	1.000	1.000
#168	50013	50016	1.000	1.000
#140	52634	52639	1.000	1.000
#112	51144	51144	1.000	1.000
#84	48390	48396	1.000	1.000
#56	48566	48568	1.000	1.000
#28	44464	44468	1.000	1.000
#0	44163	44167	1.000	1.000

Table 4. Object detection results for attribute “color”, evaluated on the validation set of I-RAVEN.

“number”, which can be directly inferred by counting the number of predicted bounding boxes within a panel.

The RAVEN/I-RAVEN dataset typically contains 70,000 RPM instances that are evenly distributed among 7 configurations. Within each configuration, the 10,000 RPM instances are randomly split into 6,000 training instances, 2,000 validation instances, and 2,000 test instances. For our experiments, we used only 10% of the training set, i.e. we trained our object detection modules on 4,200 RPM instances (600 from each configuration). Every RPM instance can be split into 16 grayscale images of size 160×160 , corresponding to 8 panels in the question matrix and 8 answer options. Each image has one or more geometric entities, which we shall detect. The ground truth labels and bounding box of each geometric entity can be obtained from the given XML files.

We used the `MMDetection` package [6] for standard training, with their implementation of RetinaNet models with a ResNet-50 backbone. All hyper-parameters are set to the default values, with the following exceptions: (i) during training, the initial learning rate is 0.003 for all attributes; and (ii) during the inference stage, we use a confidence threshold of 0.99 for “type”, and 0.95 for the other 3 attributes. All 4 modules are trained over 12 epochs.

Class	GTs	Dets	Recall	AP
(0.16, 0.16, 0.33)	16646	16646	1.000	1.000
(0.16, 0.5, 0.33)	16803	16803	1.000	1.000
(0.16, 0.83, 0.33)	16531	16559	1.000	1.000
(0.5, 0.16, 0.33)	16652	16652	1.000	1.000
(0.5, 0.5, 0.33)	18455	18445	0.999	0.999
(0.5, 0.83, 0.33)	18354	18348	1.000	1.000
(0.83, 0.16, 0.33)	32000	32000	1.000	1.000
(0.83, 0.5, 0.33)	32000	32000	1.000	1.000
(0.83, 0.83, 0.33)	32000	32000	1.000	1.000
(0.25, 0.25, 0.5)	16761	16761	1.000	1.000
(0.25, 0.75, 0.5)	16851	16851	1.000	1.000
(0.75, 0.25, 0.5)	18420	18404	0.999	0.999
(0.75, 0.75, 0.5)	96000	96000	1.000	1.000
(0.42, 0.42, 0.15)	16753	16753	1.000	1.000
(0.42, 0.58, 0.15)	16611	16611	1.000	1.000
(0.58, 0.42, 0.15)	18128	18128	1.000	1.000
(0.58, 0.58, 0.15)	17992	17992	1.000	1.000
(0.5, 0.25, 0.5)	18081	18042	0.998	0.998
(0.5, 0.75, 0.5)	18135	18104	0.998	0.998
(0.25, 0.5, 0.5)	48648	48688	1.000	1.000
(0.75, 0.5, 0.5)	32000	32000	1.000	1.000
(0.5, 0.5, 1.0)	18441	18417	0.999	0.999

Table 5. Object detection results for attribute “position”, evaluated on the validation set of I-RAVEN.

The evaluation results on the validation set with 14,000 instances (2,000 for each configuration) are shown in Tables 3-6.

According to the tables above, the object detection module for attribute “type” has only 6 errors out of 552,262 objects, which significantly outperforms the object detection modules for the other 3 attributes. Hence the module for “type” is used as an “anchor” to combine the detection results obtained from all 4 modules. In particular, we first obtain the list of detected objects, with a confidence score above 0.99, from the “type” detection module. The prediction result of each object can be reformulated as a pair (type label, bounding box). For every detected “type” object, we then obtain the lists of detection results from the other 3 detection modules with a confidence score above 0.8; within each list, we select the (label, bounding box) pair with the highest Intersection Over Union (IoU) score w.r.t. the anchor bounding box from “type” module. These 3 selected pairs are assumed to be related to the entity corresponding to the pair generated by the “type” module. Thereafter, we can collect 4 closely related (label, bounding box) pairs, each representing a detected object returned by one detection module, and we can combine the labels from these pairs as the attribute values extracted from a single entity. Finally, we applied the 4 object detection modules trained

Class	GTs	Dets	Recall	AP
(0.6, 0.15)	23552	19372	0.823	0.823
(0.7, 0.15)	14447	11715	0.811	0.811
(0.8, 0.15)	14811	14805	1.000	1.000
(0.9, 0.15)	20084	20084	1.000	1.000
(0.4, 0.33)	32152	32152	1.000	1.000
(0.5, 0.33)	30635	30635	1.000	1.000
(0.6, 0.33)	30070	31111	1.000	1.000
(0.7, 0.33)	29539	29539	1.000	1.000
(0.8, 0.33)	30528	30528	1.000	1.000
(0.9, 0.33)	29332	29361	1.000	1.000
(0.4, 0.5)	35748	35748	1.000	1.000
(0.5, 0.5)	34319	34319	1.000	1.000
(0.6, 0.5)	32247	32247	1.000	1.000
(0.7, 0.5)	33439	33439	1.000	1.000
(0.8, 0.5)	33002	33002	1.000	1.000
(0.9, 0.5)	32357	32365	1.000	1.000
(0.4, 1.0)	5455	5455	1.000	1.000
(0.5, 1.0)	5829	5829	1.000	1.000
(0.6, 1.0)	5336	5336	1.000	1.000
(0.7, 1.0)	26793	26793	1.000	1.000
(0.8, 1.0)	25638	25636	1.000	1.000
(0.9, 1.0)	26949	26949	1.000	1.000

Table 6. Object detection results for attribute “size”, evaluated on the validation set of I-RAVEN.

Configuration	RAVEN	I-RAVEN
Center	32000	32000
2×2Grid	32000	32000
3×3Grid	31996	31986
O-IC	23290	23617
O-IG	31988	31987
L-R	31999	31998
U-D	32000	32000
Avg. proportion	96.10%	96.24%

Table 7. The perceptual results of attribute values extracted via object detection modules, evaluated on RAVEN and I-RAVEN, in terms of the number of correctly detected panels (out of $2,000 \times 16$ panels) for each configuration and the overall proportions of correctly detected panels.

on I-RAVEN to both RAVEN and I-RAVEN datasets, to extract the attribute values from the raw RPM test images. The evaluation results are given in Table 7.

B.2. Full algorithmic details for “inverse” invariance modules and answer generation

In this subsection, we provide the details on how to generate answers for RPM instances based on the common pat-

terns extracted from the first two rows and the first two panels in the 3rd row. Informally, for every non-conflicting pattern pair $(K, \tilde{\mathbf{J}}) \in \mathcal{P}_{1,2}(\mathbf{J}) := \mathcal{P}_1^{(\text{all})}(\mathbf{J}) \cap \mathcal{P}_2^{(\text{all})}(\mathbf{J})$, we shall “inversely” apply the corresponding invariance module to the 3rd row $\tilde{\mathbf{J}}_3 := [\tilde{J}_{3,1}, \tilde{J}_{3,2}]$. Recall the following:

- $\mathcal{P}_i^{(\text{all})}(\mathbf{J})$ represents the set of pattern pairs extracted from the i -th row of concept matrix \mathbf{J} ;
- $\tilde{\mathbf{J}}$ represents a concept matrix from the extended list $[\mathbf{J}, \tilde{\mathbf{J}}^{(p_1)}, \hat{\mathbf{J}}^{(p_1)}, \dots, \tilde{\mathbf{J}}^{(p_k)}, \hat{\mathbf{J}}^{(p_k)}]$;
- K represents a common pattern specific to one attribute extracted from the first two rows of $\tilde{\mathbf{J}}$.

We begin by introducing the algorithmic details for each “inverse” invariance module.

B.2.1 “Inverse” intra-invariance module

Algorithm 1 “Inverse” intra-invariance module.

Inputs: A pattern ℓ and the 3rd row $\tilde{\mathbf{J}}_3 := [\tilde{J}_{3,1}, \tilde{J}_{3,2}]$.

- 1: $I_{\text{out}} \leftarrow \langle 0 \rangle$.
 - 2: $\mathcal{I} \leftarrow \text{pd}(\tilde{J}_{3,1} + \tilde{J}_{3,2}) \cap \text{pd}(\tilde{J}_{3,1} \cap \tilde{J}_{3,2})$.
 - 3: **for** $I \in \mathcal{I}$ **do**
 - 4: **if** $I \subseteq \langle \mathcal{A}_\ell \rangle$ **then**
 - 5: $I_{\text{out}} \leftarrow I$.
 - 6: **return** I_{out} .
-

Given a sequence of concepts J_1, \dots, J_k , we have denoted the sum and intersection of these concepts by $J_+ := J_1 + \dots + J_k$ and $J_\cap := J_1 \cap \dots \cap J_k$. Recall that this intra-invariance module extracts patterns where for some attribute, the corresponding set of attribute values for entities in J_i remains invariant over all i . The set of such extracted patterns is given by:

$$\mathcal{P}_{\text{intra}}([J_1 \dots J_k]) := \{\text{attr} \in \mathcal{L} \mid \exists I \in \text{pd}(J_+) \cap \text{pd}(J_\cap), I \subseteq \langle \mathcal{A}_{\text{attr}} \rangle\}.$$

Hence for a pattern pair $(K, \tilde{\mathbf{J}})$ extracted via this invariance module, the common pattern K is by definition an attribute label $\ell \in \mathcal{L}$. The set of values for attribute ℓ should remain invariant across $\tilde{J}_{3,1}$, $\tilde{J}_{3,2}$ and $\tilde{J}_{3,3}$. See Algorithm 1 for the pseudo-code to compute the ideal generated by the corresponding attribute variable for $\tilde{J}_{3,3}$.

Note that Algorithm 1 can automatically filter out those patterns that conflict with the 3rd row $\tilde{\mathbf{J}}_3$, without needing any manual checking. Specifically, the output of Algorithm 1 would be a zero ideal ($I_{\text{out}} = \langle 0 \rangle$) if the input pattern ℓ conflicts with the input 3rd row $\tilde{\mathbf{J}}_3$. For the remaining 3 “inverse” invariance modules, we have the same mechanism to automatically filter the conflicting patterns, i.e. the presence of conflicting patterns would yield an output $I_{\text{out}} = \langle 0 \rangle$.

B.2.2 “Inverse” inter-invariance module

Algorithm 2 “Inverse” inter-invariance module.

Inputs: A pattern (ℓ, \mathcal{I}) and the 3rd row $\check{\mathbf{J}}_3 := [\check{J}_{3,1}, \check{J}_{3,2}]$.

- 1: $I_{\text{out}} \leftarrow \langle 0 \rangle$.
 - 2: Compute \mathcal{I}' in (2).
 - 3: **if** $\mathcal{I} - \mathcal{I}' \neq \emptyset$ and $\mathcal{I} \cap \mathcal{I}' \neq \emptyset$ **then**
 - 4: $I_{\text{out}} \leftarrow \sum_{I \in \mathcal{I} - \mathcal{I}'} I$.
 - 5: **return** I_{out} .
-

Recall that the set of patterns extracted via inter-invariance module is given by:

$$\mathcal{P}_{\text{inter}}([J_1, \dots, J_k]) := \left\{ (\text{attr}, \mathcal{I}) \left| \begin{array}{l} \mathcal{I} \subseteq \text{pd}(J_{\cap}) - \text{pd}(J_{+}), \\ \text{attr} \in \mathcal{L}, I \subseteq \langle \mathcal{A}_{\text{attr}} \rangle \forall I \in \mathcal{I} \end{array} \right. \right\},$$

For a pattern pair $(K, \check{\mathbf{J}})$ extracted via this invariance module, the common pattern $K = (\ell, \mathcal{I})$ is a pair, whose first entry ℓ is an attribute label, and whose second entry \mathcal{I} is a set of ideals contained in the attribute concept $\langle \mathcal{A}_{\ell} \rangle$. Naturally, we shall infer the attribute values for $\check{J}_{3,3}$ based on the set difference $\mathcal{I} - \mathcal{I}'$, where \mathcal{I}' is defined by:

$$\mathcal{I}' = \left\{ I \mid I \in \left(\text{pd}(\check{J}_{3,1} \cap \check{J}_{3,2}) - \text{pd}(\check{J}_{3,1} + \check{J}_{3,2}) \right), I \subseteq \langle \mathcal{A}_{\ell} \rangle \right\}. \quad (2)$$

Note that this \mathcal{I}' is defined in the manner that is analogous to our definition for $\mathcal{P}_{\text{inter}}$ (restricted to the 3rd row of $\check{\mathbf{J}}$ and to attribute ℓ). See Algorithm 2 for a summary.

B.2.3 “Inverse” compositional invariance module

Algorithm 3 “Inverse” compositional invariance module.

Inputs: A pattern $(\ell, f_{\text{next}}|_{\Delta=\Delta^*})$ and the 3rd row $\check{\mathbf{J}}_3 := [\check{J}_{3,1}, \check{J}_{3,2}]$.

- 1: $I_{\text{out}} \leftarrow \langle 0 \rangle$.
 - 2: $\mathcal{I} \leftarrow \text{pd}(f_{\text{next}}^2(\check{J}_{3,1}|\Delta^*)) \cap \text{pd}(f_{\text{next}}(\check{J}_{3,2}|\Delta^*))$.
 - 3: **for** $I \in \mathcal{I}$ **do**
 - 4: **if** $I \in \langle \mathcal{A}_{\ell} \rangle$ **then**
 - 5: $I_{\text{out}} \leftarrow I$.
 - 6: **return** I_{out} .
-

Given a function f that maps concepts to concepts, the compositional invariance module extracts the patterns arising from invariant attribute values in the new sequence of concepts:

$$[J'_1, \dots, J'_k] = [f^{k-1}(J_1), f^{k-2}(J_2), \dots, f(J_{k-1}), J_k].$$

The set of such patterns is given by:

$$\mathcal{P}_{\text{comp}}([J_1, \dots, J_k]) := \left\{ (\text{attr}, f) \left| \begin{array}{l} \exists I \in \bigcap_{i=1}^k \text{pd}(f^{k-i}(J_i)), \\ \text{attr} \in \mathcal{L}, I \subseteq \langle \mathcal{A}_{\text{attr}} \rangle \end{array} \right. \right\}.$$

Each pattern $K = (\ell, f)$ can be interpreted as follows: The values of attribute ℓ are invariant across the new sequence of concepts. In other words, for any pair of successive concepts $[J_i, J_{i+1}]$, $i = 1, \dots, k-1$, in the original sequence, the values of attribute ℓ should be invariant in $[f(J_i), J_{i+1}]$. This property provides a natural way to compute the attribute values of a concept J_{i+1} , given the attribute values of the previous concept J_i in the sequence; see Algorithm 3. Note that for the RPM task, we introduce the $f_{\text{next}}|_{\Delta}$ function for our compositional invariance module to extract the sequential information within a row of the concept matrix.

B.2.4 “Inverse” binary-operator invariance module

Algorithm 4 “Inverse” binary-operator invariance module.

Inputs: A pattern $(\overline{\mathcal{O}}, g_{\ell}, \Lambda)$, and the 3rd row $\check{\mathbf{J}}_3 := [\check{J}_{3,1}, \check{J}_{3,2}]$.

- 1: $I_{\text{out}} \leftarrow \langle 0 \rangle$.
 - 2: $t \leftarrow g_{\ell}(\check{J}_{3,1}) \odot_1 g_{\ell}(\check{J}_{3,2})$. // expected $g_{\ell}(\check{J}_{3,3})$ value
 - 3: $x \leftarrow t$ -th variable in the corresponding attribute sequence (or subsequence) for attribute ℓ .
 - 4: $I_{\text{out}} \leftarrow \langle x \rangle$.
 - 5: **return** I_{out} .
-

Given a real-valued function g on concepts and a set Λ of binary operators, the binary-operator invariance module extracts patterns arising from the sequence of numbers $g(J_1), \dots, g(J_k)$, described as follows:

$$\mathcal{P}_{\text{binary}}(\mathbf{J}_i) := \left\{ (\overline{\mathcal{O}}, g, \Lambda) \left| \begin{array}{l} \overline{\mathcal{O}} = [\odot_1, \dots, \odot_{k-2}], \odot_i \in \Lambda, \\ g(J_1) \odot_1 \dots \odot_{k-2} g(J_{k-1}) = g(J_k) \end{array} \right. \right\}.$$

In other words, we can represent each pattern as an equation, obtained by inserting binary operators chosen from Λ between the numbers in the sequence.

For a new given sequence of concepts J'_1, \dots, J'_{k-1} and a pattern $K = ([\odot_1, \dots, \odot_{k-2}], g, \Lambda)$, the value of $g(J'_k)$ we expect from this pattern can be directly computed as $g(J'_k) = g(J'_1) \odot_1 \dots \odot_{k-2} g(J'_{k-1})$. Then we can infer the information about concept J'_k based on the meaning of function g .

As described in Section B.1, we introduced the function $g_{\text{num}}(J)$ to extract the number of minimal monomial generators of J (which also represents the number of entities in the corresponding panel), and introduced the functions g_{color} (resp. g_{size}) to extract the variable index within the “color” value sequence (resp. within the “size” value sub-sequence). Hence we can directly infer the corresponding attribute variables for J'_k from the values of $g_{\ell}(J'_k)$, for $\ell \in [\text{“num”}, \text{“color”}, \text{“size”}]$. See Algorithm 4 for a summary.

Methods	Avg. Acc.	Center	2×2G	3×3G	O-IC	O-IG	L-R	U-D
With all 4 invariance modules	93.2	99.5	89.6	89.7	99.6	74.7	99.7	99.5
W/o intra-invariance module	63.7	63.5	71.6	73.4	60.2	49.8	63.2	64.0
W/o inter-invariance module	59.0	64.3	56.8	59.2	59.2	43.5	64.5	65.8
W/o compositional invariance module	69.5	73.0	65.9	64.3	78.1	61.5	72.9	71.2
W/o binary operator invariance module	74.6	75.8	69.3	67.1	86.2	70.2	77.3	76.7

Table 8. Detailed ablation study results of our algebraic machine reasoning framework, evaluated on the I-RAVEN dataset, in terms of the overall answer-selection accuracy and the individual accuracies for all seven RPM configurations.

Algorithm 5 Detailed answer generation process.

Inputs: A concept matrix $\mathbf{J} = [J_{1,1}, \dots, J_{3,2}]$.

- 1: Compute $\text{comPos}(\mathbf{J}) = [p_1, \dots, p_k]$.
- 2: $I_{\text{out}} \leftarrow \langle 0 \rangle$.
- 3: **for** $p_i \in [p_1, \dots, p_k]$ **do**
// initialize with fixed “number” and “position”
- 4: $I^{(i)} \leftarrow \langle x_{k-1} x_{p_i} \rangle$.
- 5: **for** $K \in \mathcal{P}(\bar{\mathbf{J}}_1^{(p_i)}) \cap \mathcal{P}(\bar{\mathbf{J}}_2^{(p_i)})$ **do**
- 6: $I^* \leftarrow$ Output ideal generated based on one of the Algorithms 1-4, given inputs K and $\bar{\mathbf{J}}_3^{(p_i)}$.
- 7: **if** $I^* \neq \langle 0 \rangle$ **then**
- 8: $I^{(i)} \leftarrow I^{(i)} \cap I^*$
- 9: $m \leftarrow$ generator randomly selected from $\text{mingen}(I^{(i)})$.
- 10: $I_{\text{out}} \leftarrow I_{\text{out}} + \langle m \rangle$.
- 11: $\mathcal{G} \leftarrow \emptyset$
- 12: **for** $m \in \text{mingen}(I_{\text{out}})$ **do**
- 13: $m' \leftarrow m$
- 14: **while** $\exists \ell \in \mathcal{L}$ s.t. x does not divide $m \forall x \in \mathcal{A}_\ell$ **do**
- 15: $x' \leftarrow$ a variable randomly selected from \mathcal{A}_ℓ .
- 16: $m' \leftarrow m' x'$.
- 17: $\mathcal{G} \leftarrow \mathcal{G} \cup \{m'\}$.
- 18: $I_{\text{out}} \leftarrow \langle \mathcal{G} \rangle$.
- 19: **return** I_{out} .

B.2.5 Overall process of answer generation

In the RAVEN/I-RAVEN dataset, the values for some attribute within a panel may contain randomness by design. For example, there do not exist rules for both “number” and “position” to avoid conflicts, thus the entity positions in a panel from some configurations (such as 2×2Grid, 3×3Grid, and Out-InGrid) could be totally random once the number of entities is fixed. The values of other attributes could also be random; see Figure 1 for example. It is difficult to extract patterns from those concept matrices with such random attribute values, and effectively impossible to generate an answer that matches the given correct answer. Hence, for any given RPM instance, we shall simplify the answer generation process by focusing on each single position.

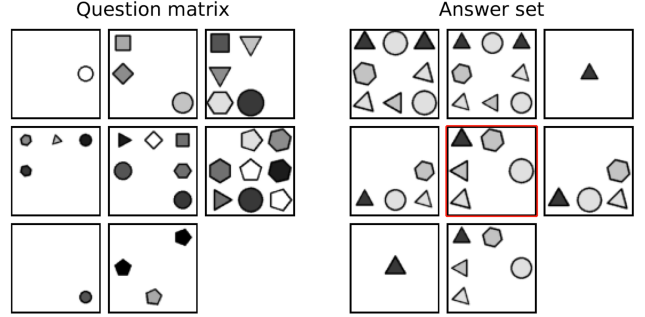


Figure 1. An example of a RPM instance with random values for attributes “position”, “type” and “color”, within each panel. The correct answer is marked with a red box.

Recall that for a concept matrix \mathbf{J} representing a given RPM instance, we extract the common patterns from each concept matrix $\bar{\mathbf{J}}$ in an extended list of concept matrices, i.e. $[\mathbf{J}, \bar{\mathbf{J}}^{(p_1)}, \hat{\mathbf{J}}^{(p_1)}, \dots, \bar{\mathbf{J}}^{(p_k)}, \hat{\mathbf{J}}^{(p_k)}]$; see Section 3.3.4 in the main paper. Here p_1, \dots, p_k are the common positions we extracted from the 8 panels of the question matrix corresponding to \mathbf{J} , and all $\bar{\mathbf{J}}^{(p_t)}, \hat{\mathbf{J}}^{(p_t)}$ are defined as follows:

- Each concept $\bar{\mathbf{J}}_{i,j}^{(p_t)}$ in $\bar{\mathbf{J}}^{(p_t)}$ is generated by the unique generator in $J_{i,j}$ that is divisible by p_t ;
- Each concept $\hat{\mathbf{J}}_{i,j}^{(p_t)}$ in $\hat{\mathbf{J}}^{(p_t)}$ is generated by all generators in $J_{i,j}$ that are not divisible by p_t .

In other words, for a common position p_t , each panel in the original question matrix is split into two panels, with one panel containing only the entity in position p_t (represented by the corresponding concept in $\bar{\mathbf{J}}^{(p_t)}$), and the other panel containing all the remaining entities (represented by the corresponding concept in $\hat{\mathbf{J}}^{(p_t)}$). In the answer generation task, we shall consider only all $\bar{\mathbf{J}}^{(p_i)}$ as discussed before. Intuitively, we can separately extract the patterns restricted to each single common position, across the panels in the original question matrix. Since the attributes “number” and “position” are inherently fixed within each $\bar{\mathbf{J}}^{(p_i)}$, we only need to extract patterns for the remaining 3 attributes. See Algorithm 5 for details. Note that in Line 6 of Algorithm 5, when generating the output ideal I^* , the exact algorithm (from Algorithm 1-4) to apply to the inputs K and $\bar{\mathbf{J}}_3^{(p_i)}$ is uniquely determined by the syntax of K .

B.3. Examples for invariance modules

Consider the RPM instance depicted in Figure 2. (This is the same as the running example in the main paper.) In this subsection, we provide specific computational details on this RPM instance to show how the invariance modules are used to extract patterns, and generate the correct answer.

Example 1 (Intra-invariance module). Consider the first row in Fig. 2. By extracting attribute values for entities in each panel via the object detection modules, we can obtain the algebraic representation given as follows:

$$\mathbf{J}_1 := [J_{1,1}, J_{1,2}, J_{1,3}],$$

$$J_{1,1} = \langle x_{\text{two}}x_{\text{left}}x_{\text{square}}x_{\text{black}}x_{\text{avg}}, x_{\text{two}}x_{\text{right}}x_{\text{triangle}}x_{\text{gray}}x_{\text{avg}} \rangle,$$

$$J_{1,2} = \langle x_{\text{two}}x_{\text{left}}x_{\text{pentagon}}x_{\text{gray}}x_{\text{avg}}, x_{\text{two}}x_{\text{right}}x_{\text{square}}x_{\text{gray}}x_{\text{avg}} \rangle,$$

$$J_{1,3} = \langle x_{\text{two}}x_{\text{left}}x_{\text{circle}}x_{\text{white}}x_{\text{avg}}, x_{\text{two}}x_{\text{right}}x_{\text{pentagon}}x_{\text{gray}}x_{\text{avg}} \rangle.$$

We first compute $\text{pd}(J_+)$ and $\text{pd}(J_\cap)$.

$$\begin{aligned} \text{pd}(J_+) = \{ & \langle x_{\text{two}} \rangle, \langle x_{\text{avg}} \rangle, \langle x_{\text{left}}, x_{\text{right}} \rangle, \langle x_{\text{white}}, x_{\text{gray}}, x_{\text{black}} \rangle, \\ & \langle x_{\text{triangle}}, x_{\text{square}}, x_{\text{pentagon}}, x_{\text{circle}} \rangle, \langle x_{\text{left}}, x_{\text{gray}} \rangle, \\ & \langle x_{\text{square}}, x_{\text{circle}}, x_{\text{gray}} \rangle, \langle x_{\text{square}}, x_{\text{white}}, x_{\text{gray}} \rangle \\ & \langle x_{\text{circle}}, x_{\text{gray}}, x_{\text{black}} \rangle, \\ & \langle x_{\text{left}}, x_{\text{triangle}}, x_{\text{square}}, x_{\text{pentagon}} \rangle \\ & \langle x_{\text{right}}, x_{\text{square}}, x_{\text{pentagon}}, x_{\text{circle}} \rangle \\ & \langle x_{\text{right}}, x_{\text{square}}, x_{\text{pentagon}}, x_{\text{white}} \rangle \\ & \langle x_{\text{right}}, x_{\text{pentagon}}, x_{\text{circle}}, x_{\text{black}} \rangle \\ & \langle x_{\text{right}}, x_{\text{pentagon}}, x_{\text{white}}, x_{\text{black}} \rangle \\ & \langle x_{\text{triangle}}, x_{\text{square}}, x_{\text{pentagon}}, x_{\text{white}} \rangle \}, \\ \text{pd}(J_\cap) = \{ & \langle x_{\text{two}} \rangle, \langle x_{\text{avg}} \rangle, \langle x_{\text{left}}, x_{\text{right}} \rangle, \langle x_{\text{gray}} \rangle, \\ & \langle x_{\text{triangle}}, x_{\text{square}} \rangle, \langle x_{\text{square}}, x_{\text{pentagon}} \rangle, \\ & \langle x_{\text{pentagon}}, x_{\text{circle}} \rangle, \langle x_{\text{left}}, x_{\text{triangle}} \rangle, \langle x_{\text{left}}, x_{\text{square}} \rangle, \\ & \langle x_{\text{left}}, x_{\text{pentagon}} \rangle, \langle x_{\text{right}}, x_{\text{square}} \rangle, \langle x_{\text{right}}, x_{\text{pentagon}} \rangle, \\ & \langle x_{\text{right}}, x_{\text{circle}} \rangle, \langle x_{\text{right}}, x_{\text{white}} \rangle, \langle x_{\text{right}}, x_{\text{black}} \rangle, \\ & \langle x_{\text{triangle}}, x_{\text{black}} \rangle, \langle x_{\text{pentagon}}, x_{\text{white}} \rangle \}. \end{aligned}$$

For those primary components that are contained in attribute concepts, we have highlighted them in blue. These are the concepts that are deemed “meaningful” for the RPM task. Note that most of the primary components are not contained in any attribute concept.

Next, we compute the set intersection $\text{pd}(J_+) \cap \text{pd}(J_\cap) = \{ \langle x_{\text{two}} \rangle, \langle x_{\text{avg}} \rangle, \langle x_{\text{left}}, x_{\text{right}} \rangle \}$, which can be interpreted to mean that all panels have two average-sized entities, one on the left side and one on the right side.

Recall that the output of the intra-invariance module is:

$$\mathcal{P}_{\text{intra}}([J_1 \dots J_k]) := \{ \text{attr} \in \mathcal{L} \mid \exists I \in \text{pd}(J_+) \cap \text{pd}(J_\cap), I \subseteq \langle \mathcal{A}_{\text{attr}} \rangle \}.$$

Hence, we have

$$\mathcal{P}_{\text{intra}}(\mathbf{J}_1) = \{ \text{“size”, “number”, “position”} \}.$$

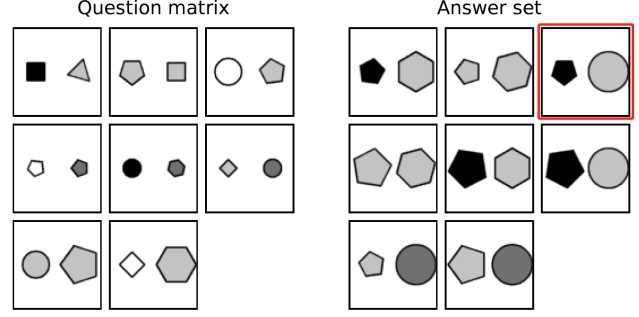


Figure 2. An example of RPM instance from the I-RAVEN dataset. The correct answer is marked with a red box.

Example 2 (Inter-invariance module). Consider the second row \mathbf{J}_2 in Fig. 2. Similarly, we first obtain the algebraic representation of \mathbf{J}_2 :

$$\mathbf{J}_2 := [J_{2,1}, J_{2,2}, J_{2,3}],$$

$$J_{2,1} = \langle x_{\text{two}}x_{\text{left}}x_{\text{pentagon}}x_{\text{white}}x_{\text{small}}, x_{\text{two}}x_{\text{right}}x_{\text{pentagon}}x_{\text{dgray}}x_{\text{small}} \rangle,$$

$$J_{2,2} = \langle x_{\text{two}}x_{\text{left}}x_{\text{circle}}x_{\text{black}}x_{\text{small}}, x_{\text{two}}x_{\text{right}}x_{\text{hexagon}}x_{\text{dgray}}x_{\text{small}} \rangle,$$

$$J_{2,3} = \langle x_{\text{two}}x_{\text{left}}x_{\text{square}}x_{\text{gray}}x_{\text{small}}, x_{\text{two}}x_{\text{right}}x_{\text{circle}}x_{\text{dgray}}x_{\text{small}} \rangle.$$

Next, we compute $\text{pd}(J_+)$, $\text{pd}(J_\cap)$ as follows:

$$\begin{aligned} \text{pd}(J_+) = \{ & \langle x_{\text{two}} \rangle, \langle x_{\text{small}} \rangle, \langle x_{\text{left}}, x_{\text{right}} \rangle, \\ & \langle x_{\text{white}}, x_{\text{gray}}, x_{\text{dgray}}, x_{\text{black}} \rangle, \\ & \langle x_{\text{square}}, x_{\text{pentagon}}, x_{\text{hexagon}}, x_{\text{circle}} \rangle, \\ & \langle x_{\text{left}}, x_{\text{dgray}} \rangle, \langle x_{\text{left}}, x_{\text{pentagon}}, x_{\text{hexagon}}, x_{\text{circle}} \rangle, \\ & \langle x_{\text{right}}, x_{\text{square}}, x_{\text{pentagon}}, x_{\text{circle}} \rangle, \\ & \langle x_{\text{right}}, x_{\text{square}}, x_{\text{pentagon}}, x_{\text{black}} \rangle, \\ & \langle x_{\text{right}}, x_{\text{square}}, x_{\text{circle}}, x_{\text{white}} \rangle, \\ & \langle x_{\text{right}}, x_{\text{square}}, x_{\text{white}}, x_{\text{black}} \rangle, \\ & \langle x_{\text{right}}, x_{\text{pentagon}}, x_{\text{circle}}, x_{\text{gray}} \rangle, \\ & \langle x_{\text{right}}, x_{\text{pentagon}}, x_{\text{gray}}, x_{\text{black}} \rangle, \\ & \langle x_{\text{right}}, x_{\text{circle}}, x_{\text{white}}, x_{\text{gray}} \rangle, \\ & \langle x_{\text{right}}, x_{\text{white}}, x_{\text{gray}}, x_{\text{black}} \rangle, \\ & \langle x_{\text{square}}, x_{\text{pentagon}}, x_{\text{circle}}, x_{\text{dgray}} \rangle, \\ & \langle x_{\text{square}}, x_{\text{pentagon}}, x_{\text{dgray}}, x_{\text{black}} \rangle, \\ & \langle x_{\text{square}}, x_{\text{circle}}, x_{\text{white}}, x_{\text{dgray}} \rangle, \\ & \langle x_{\text{square}}, x_{\text{white}}, x_{\text{dgray}}, x_{\text{black}} \rangle, \\ & \langle x_{\text{pentagon}}, x_{\text{hexagon}}, x_{\text{circle}}, x_{\text{gray}} \rangle, \\ & \langle x_{\text{pentagon}}, x_{\text{circle}}, x_{\text{gray}}, x_{\text{dgray}} \rangle, \\ & \langle x_{\text{pentagon}}, x_{\text{gray}}, x_{\text{dgray}}, x_{\text{black}} \rangle, \\ & \langle x_{\text{circle}}, x_{\text{white}}, x_{\text{gray}}, x_{\text{dgray}} \rangle \}, \end{aligned}$$

$$\begin{aligned} \text{pd}(J_\cap) = \{ & \langle x_{\text{two}}, \rangle, \langle x_{\text{small}}, \rangle, \langle x_{\text{left}}, x_{\text{right}} \rangle, \langle x_{\text{pentagon}}, \rangle, \\ & \langle x_{\text{square}}, x_{\text{circle}} \rangle, \langle x_{\text{hexagon}}, x_{\text{circle}} \rangle, \\ & \langle x_{\text{white}}, x_{\text{dgray}} \rangle, \langle x_{\text{gray}}, x_{\text{dgray}} \rangle, \langle x_{\text{dgray}}, x_{\text{black}} \rangle, \\ & \langle x_{\text{left}}, x_{\text{hexagon}} \rangle, \langle x_{\text{left}}, x_{\text{circle}} \rangle, \langle x_{\text{left}}, x_{\text{dgray}} \rangle, \\ & \langle x_{\text{right}}, x_{\text{square}} \rangle, \langle x_{\text{right}}, x_{\text{circle}} \rangle, \langle x_{\text{right}}, x_{\text{white}} \rangle, \\ & \langle x_{\text{right}}, x_{\text{gray}} \rangle, \langle x_{\text{right}}, x_{\text{black}} \rangle, \langle x_{\text{square}}, x_{\text{dgray}} \rangle, \\ & \langle x_{\text{hexagon}}, x_{\text{black}} \rangle, \langle x_{\text{circle}}, x_{\text{gray}} \rangle, \langle x_{\text{circle}}, x_{\text{dgray}} \rangle \}. \end{aligned}$$

Similarly, we have indicated in blue those primary components that are contained in attribute concepts.

Recall that the output of the inter-invariance module is:

$$\mathcal{P}_{\text{inter}}([J_1, \dots, J_k]) := \left\{ (\text{attr}, \mathcal{I}) \left| \begin{array}{l} \mathcal{I} \subseteq \text{pd}(J_\cap) - \text{pd}(J_+), \\ \text{attr} \in \mathcal{L}, \mathcal{I} \subseteq \langle \mathcal{A}_{\text{attr}} \rangle \forall \mathcal{I} \in \mathcal{I} \end{array} \right. \right\},$$

Hence $\mathcal{P}_{\text{inter}}(\mathbf{J}_2)$ is the following set with two pairs:

$$\left\{ \begin{array}{l} \left(\text{"color"}, \{ \langle x_{\text{white}}, x_{\text{dgray}} \rangle, \langle x_{\text{gray}}, x_{\text{dgray}} \rangle, \langle x_{\text{black}}, x_{\text{dgray}} \rangle \} \right), \\ \left(\text{"type"}, \{ \langle x_{\text{pentagon}} \rangle, \langle x_{\text{square}}, x_{\text{circle}} \rangle, \langle x_{\text{hexagon}}, x_{\text{circle}} \rangle \} \right) \end{array} \right\}.$$

The first pair encodes the information that the three extracted “color” concepts correspond to “white or dark gray”, “gray or dark gray”, and “black or dark gray”. The second pair encodes the information that the three extracted “type” concepts correspond to “pentagon”, “square or circle”, and “hexagon or circle”.

Example 3 (Compositional invariance module). Consider the right subpanels of the first row $\bar{\mathbf{J}}_1^{(\text{right})}$ in Fig. 2, and $\Delta = 1$. We can generate the new row:

$$\begin{aligned} \bar{J}'_{1,1} &= f_{\text{next}}^2(\bar{J}_{1,1}^{(\text{right})}|1) = \langle x_{\text{right}}x_{\text{one}}x_{\text{pentagon}}x_{\text{black}}x_{\text{small}} \rangle, \\ \bar{J}'_{1,2} &= f_{\text{next}}(\bar{J}_{1,2}^{(\text{right})}|1) = \langle x_{\text{left}}x_{\text{two}}x_{\text{pentagon}}x_{\text{dgray}}x_{\text{large}} \rangle, \\ \bar{J}'_{1,3} &= \bar{J}_{1,3}^{(\text{right})} = \langle x_{\text{right}}x_{\text{one}}x_{\text{pentagon}}x_{\text{gray}}x_{\text{avg}} \rangle. \end{aligned}$$

Recall that the output of the compositional invariance module is:

$$\mathcal{P}_{\text{comp}}([J_1, \dots, J_k]) := \left\{ (\text{attr}, f) \left| \begin{array}{l} \exists I \in \bigcap_{i=1}^k \text{pd}(f^{k-i}(J_i)), \\ \text{attr} \in \mathcal{L}, \mathcal{I} \subseteq \langle \mathcal{A}_{\text{attr}} \rangle \end{array} \right. \right\}.$$

Hence, we have $\mathcal{P}_{\text{comp}}(\bar{\mathbf{J}}_1^{(\text{right})}) = \{(\text{"type"}, f_{\text{next}}|_{\Delta=1})\}$, which means that the “type” value (i.e. x_{pentagon}) remains invariant in the new row generated by f_{next} with step $\Delta = 1$.

Example 4 (Binary-operator module). Recall that the output of the binary-operator module is:

$$\mathcal{P}_{\text{binary}}(\mathbf{J}_i) := \left\{ \overline{\mathcal{O}} \left| \begin{array}{l} \overline{\mathcal{O}} = [\mathcal{O}_1, \dots, \mathcal{O}_{k-2}], \mathcal{O}_i \in \Lambda, \\ g(J_1) \mathcal{O}_1 \dots \mathcal{O}_{k-2} g(J_{k-1}) = g(J_k) \end{array} \right. \right\}.$$

Let g be the real-valued function on concepts given by $J \mapsto |\text{mingen}(J)|$. Note that for both rows \mathbf{J}_1 and \mathbf{J}_2 in Fig. 2, every panel $J_{i,j}$ has two entities, which can be computed as $g(J_{i,j}) = 2$. Hence, if $\Lambda = \{+, -\}$, then $\mathcal{P}_{\text{binary}}(\mathbf{J}_1)$ and $\mathcal{P}_{\text{binary}}(\mathbf{J}_2)$ are both empty.

Example 5 (Answer generation). For the example depicted in Fig. 2, we have $\text{comPos}(\mathbf{J}) = \{x_{\text{left}}, x_{\text{right}}\}$.

Note that $\bar{\mathbf{J}}^{(x_{\text{left}})} = \hat{\mathbf{J}}^{(x_{\text{right}})}$. (The panel with only the left entity is identical to the panel without the right entity.) Hence we shall consider only the list of concept matrices $[\mathbf{J}, \bar{\mathbf{J}}^{(x_{\text{left}})}, \bar{\mathbf{J}}^{(x_{\text{right}})}]$.

By applying the invariance modules iteratively, we can then compute $\mathcal{P}_{1,2}(\mathbf{J}) := \mathcal{P}_1^{(\text{all})}(\mathbf{J}) \cap \mathcal{P}_2^{(\text{all})}(\mathbf{J})$, whose 13 elements are labeled as $\mathfrak{P}_1, \dots, \mathfrak{P}_{13}$:

$$\begin{aligned} \mathfrak{P}_1 &= (\text{"num"}, \mathbf{J}) & \mathfrak{P}_2 &= (\text{"num"}, \bar{\mathbf{J}}^{(x_{\text{left}})}) & \mathfrak{P}_3 &= (\text{"num"}, \bar{\mathbf{J}}^{(x_{\text{right}})}) \\ \mathfrak{P}_4 &= (\text{"pos"}, \mathbf{J}) & \mathfrak{P}_5 &= (\text{"pos"}, \bar{\mathbf{J}}^{(x_{\text{left}})}) & \mathfrak{P}_6 &= (\text{"pos"}, \bar{\mathbf{J}}^{(x_{\text{right}})}) \\ \mathfrak{P}_7 &= ((\text{"type"}, \{ \langle x_{\text{square}} \rangle, \langle x_{\text{pentagon}} \rangle, \langle x_{\text{circle}} \rangle \}), \bar{\mathbf{J}}^{(x_{\text{left}})}) \\ \mathfrak{P}_8 &= ((\text{"type"}, f_{\text{next}}|_{\Delta=1}), \bar{\mathbf{J}}^{(x_{\text{right}})}) & \mathfrak{P}_9 &= (\text{"color"}, \bar{\mathbf{J}}^{(x_{\text{right}})}) \\ \mathfrak{P}_{10} &= ((\text{"color"}, \{ \langle x_{\text{white}} \rangle, \langle x_{\text{gray}} \rangle, \langle x_{\text{black}} \rangle \}), \bar{\mathbf{J}}^{(x_{\text{left}})}) \\ \mathfrak{P}_{11} &= (\text{"size"}, \mathbf{J}) & \mathfrak{P}_{12} &= (\text{"size"}, \bar{\mathbf{J}}^{(x_{\text{left}})}) & \mathfrak{P}_{13} &= (\text{"size"}, \bar{\mathbf{J}}^{(x_{\text{right}})}) \end{aligned}$$

We can also generate the algebraic representations of the first two panels in the 3rd row $\mathbf{J}_3 := [J_{3,1}, J_{3,2}]$:

$$\begin{aligned} J_{3,1} &= \langle x_{\text{two}}x_{\text{left}}x_{\text{circle}}x_{\text{avg}}x_{\text{gray}}, x_{\text{two}}x_{\text{right}}x_{\text{pentagon}}x_{\text{gray}}x_{\text{large}} \rangle; \\ J_{3,2} &= \langle x_{\text{two}}x_{\text{left}}x_{\text{square}}x_{\text{avg}}x_{\text{white}}, x_{\text{two}}x_{\text{right}}x_{\text{hexagon}}x_{\text{gray}}x_{\text{large}} \rangle. \end{aligned}$$

Among these 13 patterns, only \mathfrak{P}_{11} (the “size” values of entities are invariant across each row in \mathbf{J}) *conflicts* with \mathbf{J}_3 , as the “size” values of the two entities in $J_{3,1}$ (or $J_{3,2}$) are different. Each remaining non-conflicting pattern gives information about the corresponding attribute value for $J_{3,3}$. \mathfrak{P}_1 to \mathfrak{P}_6 : two entities in $J_{3,3}$, with one entity on the left side and the other entity on the right side; $\mathfrak{P}_7 \Rightarrow x_{\text{pentagon}}$ for the left entity; $\mathfrak{P}_8 \Rightarrow x_{\text{circle}}$ for the right entity; $\mathfrak{P}_9 \Rightarrow x_{\text{gray}}$ for the right entity; $\mathfrak{P}_{10} \Rightarrow x_{\text{black}}$ for the left entity; $\mathfrak{P}_{12} \Rightarrow x_{\text{avg}}$ for the left entity; and $\mathfrak{P}_{13} \Rightarrow x_{\text{large}}$ for the right entity. Consequently, we can collect all the attribute values to form our generated answer concept

$$J'_{3,3} = \langle x_{\text{two}}x_{\text{left}}x_{\text{pentagon}}x_{\text{avg}}x_{\text{black}}, x_{\text{two}}x_{\text{right}}x_{\text{circle}}x_{\text{large}}x_{\text{gray}} \rangle.$$

B.4. Ablation study

In order to study the effect of each invariance module, we conducted an ablation study and evaluated the performance of our algebraic machine reasoning framework on the answer selection task; see Table 8.

For any given RPM instance, there could be a tie in the `comPattern` list (defined in Algorithm 1 of the main paper), i.e. there exist multiple entries in `comPattern` with the same highest score. Recall that the i -th entry in `comPattern` represents the number of extracted common patterns across 3 rows, with the i -th answer option inserted in the question matrix. Hence we report the weighted accuracies of our framework, defined as follows:

$$\frac{1}{N} \sum_{i=1}^N \frac{1}{n_i},$$

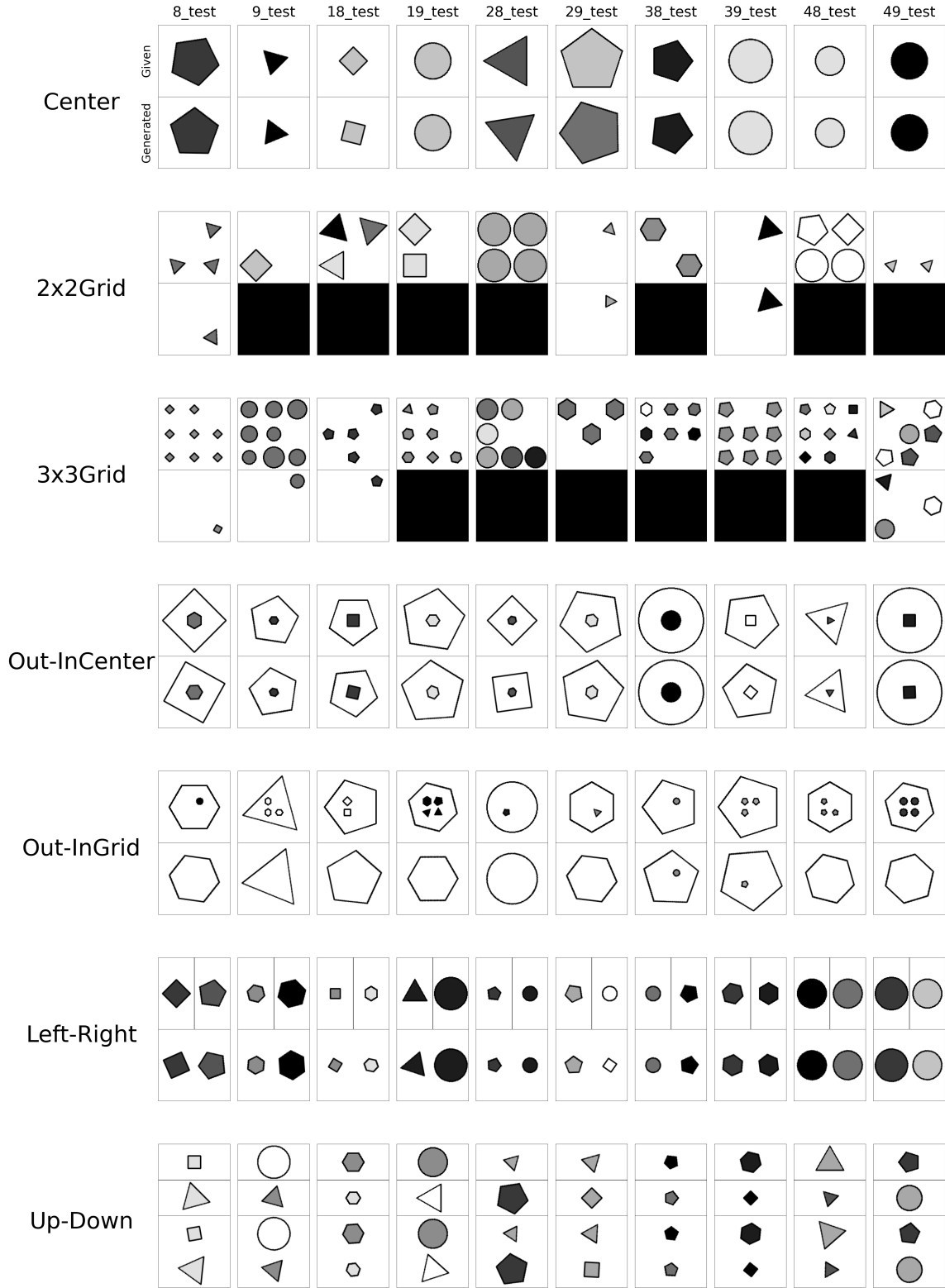


Figure 3. A collection of the given answer images (top) and the corresponding generated images (bottom) for the first 10 I-RAVEN test instances in each configuration. For those test instances for which our reasoning framework fails to extract non-conflicting patterns, the output of Algorithm 5 would be $I_{\text{out}} = \langle 0 \rangle$; we have indicated such generated results as completely black panels.

where N denotes the total number of RPM test instances, and n_i represents the number of answer options with the same highest score for the i -th test instance.

As shown in Table 8, the inter-invariance module has the most significant contribution towards the final average accuracy. Interestingly, this very same module allows us to discover an unexpected new pattern as depicted in Figure 4 in the main paper, which is a rather natural pattern that human could conceivably think of, but which is not one of the designed rules for I-RAVEN.

B.5. Examples of generated answers

To further evaluate the performance of our algebraic machine reasoning framework on the answer generation task, we provide in Figure 3 the comparison between the given correct answer images and our generated answer images for the I-RAVEN dataset. For each configuration, we choose the first 10 test files to show the comparison between the given answer images and the generated images. (In the I-RAVEN dataset, every 10 instances are split into 6 training instances, 2 validation instances, and 2 test instances. Hence every test instance has a filename that ends with “8_test” or “9_test”.)

From Figure 3, we can see that for the configurations Center, Out-InCenter, Left-Right, and Up-Down, our generated answers capture almost all salient image attribute features, except for the angles of entities (i.e. the orientation with respect to the panels) which is not involved in any of the rules of RAVEN/I-RAVEN. However, for the remaining configurations, our reasoning framework is only able to generate a part of the answer. This is because we simplified the answer generation process by separately extracting patterns restricted to every single common position, across the panels in the question matrix. Hence, for those configurations where the RPM panels always contain a fixed set of positions, our reasoning framework is able to effectively extract the patterns and generate the correct answer. On the other hand, for those configurations where the RPM panels may contain an arbitrary set of positions, our reasoning framework does not perform as well.

C. Further Discussion

C.1. Why is algebraic machine reasoning different from logic-based reasoning?

Logic is the foundation of reasoning. Logic programming [3, 20, 27] is a logic-based programming paradigm that serves as the foundational computational framework for logic-based reasoning methods. At the heart of these methods in logic-based reasoning is the idea that reasoning can be realized very concretely as the resolution (or inverse resolution) of logical expressions. Inherent in this idea is the notion of *satisfiability*; cf. [20]. Intuitively, we have a log-

ical expression, typically expressed in some canonical normal form, and we want to assign truth values (true or false) to literals in the logical expression, so that the entire logical expression is satisfied (i.e. with truth value “true”). In fact, much of today’s very exciting progress in automated theorem proving [2, 19, 24, 26, 36, 45, 46, 48] is based on logic-based reasoning.

In contrast, algebraic machine reasoning builds upon computational algebra and computer algebra systems; cf. Appendix A.2.2. As described in the main paper, at the heart of our algebraic approach is the idea that reasoning can be realized very 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 $R = \mathbb{k}[x_1, \dots, x_n]$. In particular, although primitive concepts $\langle x_1 \rangle, \dots, \langle x_n \rangle$ in R correspond to the variables x_1, \dots, x_n , we do not assign values to primitive concepts. Instead, as we have emphasized in the main paper, ideals are treated as the “actual objects of study”, and we reduce “solving a reasoning task” to “solving non-numerical computational problems involving ideals”.

This contrast is perhaps even more evident when comparing algebraic machine reasoning to inductive logic programming [33, 34]. Both seek to solve reasoning tasks involving the discovery of new patterns, but they are fundamentally different approaches.

In inductive logic programming, we are given (as our starting point) background knowledge B and examples E , both in the form of logical formulas. For example, B could be a conjunction of implications ($p_1 \wedge \dots \wedge p_k \rightarrow q$), and E could be a formula in conjunctive normal form. The goal is to discover a hypothesis H , which is a logical formula that must satisfy $B \wedge H \models E$ (i.e. $B \wedge H$ entails E). Informally, we seek to discover a “good” hypothesis H that is consistent with both background knowledge and examples. The key technical challenge is to find a hypothesis H that is “good” in a precise computable sense, from among the numerous (potentially exponentially many) possible hypotheses. In essence, inductive logic programming tackles this technical challenge as a search-and-selection problem [22, 34, 38]: In the space of all candidate hypotheses, search for a subset of valid hypotheses; then within this subset, select a valid hypothesis that maximizes or minimizes a pre-defined objective function (e.g. select H that minimizes Kolmogorov complexity).

In contrast, algebraic machine reasoning is intrinsically not search-based. When solving the RPM task, the new patterns discovered are computed via algebraic computations. For the RPM answer generation task, we generate answers to RPM instances not by searching for valid answers in the (huge) space of all candidate answers, but by directly computing our answer. Pattern discovery becomes computing: We are computing various primary decomposi-

tions, and computing which of the primary components are contained in attribute concepts. For the RPM answer selection task, we solve it as a “compute-and-select” problem. For the RPM answer generation task, we compute attribute values from the extracted patterns wherever possible, and randomly select attribute values for those attributes of entities not involved in the extracted patterns.

When solving RPMs, if we try to frame the reasoning process of our algebraic approach in a language similar to inductive logic programming, then have the following: Given an RPM instance represented as a concept matrix \mathbf{J} , we extract patterns $\mathcal{P}_{1,2}(\mathbf{J})$ from the first two rows of \mathbf{J} ; here $\mathcal{P}_{1,2}(\mathbf{J})$ takes the role of “background knowledge”. We then systematically check if the extracted patterns conflict with concepts $J_{3,1}, J_{3,2}$ (representing the two panels in the 3rd row of the question matrix); the patterns that can be extracted from $[J_{3,1}, J_{3,2}]$ take the role of “examples”. Finally, we wish to discover a suitable concept $J_{3,3}$ for the missing 9-th panel; here $J_{3,3}$ takes the role of “hypothesis”.

However, a closer look would reveal several disparities in our attempted analogy. We seem to have analogous notions for “background knowledge”, “examples” and “hypothesis” from the set-up of inductive logic programming, yet what we call “background knowledge” (i.e. the extracted patterns $\mathcal{P}_{1,2}(\mathbf{J})$) is not given prior knowledge, but must be computed. Similarly, the “examples” are not given prior knowledge, but are extracted patterns from $[J_{3,1}, J_{3,2}]$ that must be computed. To discover a “hypothesis”, we are not searching for it, but computing it. Effectively, the common missing ingredient for making this “analogy” proper is algebraic computations, which is the essence of algebraic machine reasoning.

C.2. Potential societal impact of algebraic machine reasoning

Algebraic machine reasoning could potentially help to automate “easier” reasoning tasks currently performed by humans. In education, algebraic machine reasoning could help with the design of better intelligence tests. In finance, our framework could help in processing personal bank loan applications or detecting fraud, based on invariant features of such cases. A negative implication, however, could also be present. Our reasoning process is based on extracting patterns from a few examples (as in the RPM task) and then generalizing. If the cases of bank loan applications or fraud have already been unfairly associated with certain socio-economic groups, then that inequity would be propagated even in our reasoning framework.

Another potential downstream application of our framework is in medical diagnostics. Diagnostic decision making, based on a patient’s reported symptoms and past medical history, could be modeled as reasoning tasks. Specific symptoms and medical conditions could be encoded as

concepts and a final medical diagnosis could potentially be computed. If properly implemented, an algebraic machine reasoning framework for medical diagnosis would significantly speed up the diagnostic decision-making process, as well as reduce the cost of medical diagnostics. Unfortunately, such an application, while bringing clear benefits, also comes with ethical concerns. If a wrong medical diagnosis is made, when a doctor uses algebraic machine reasoning to aid in diagnostic decision-making, who gets the blame? Will the creators of the reasoning framework be liable for legal action?

Biasness could also be inherent in the *human* encoding process of a machine reasoning framework. If concepts encoded reflect extant human bias, then reasoning output could be flawed, yet seemingly “reasonable”. For any future application of algebraic machine reasoning to legal or criminal cases, care must be taken in assessing and processing information (concepts) associated with each counter-party. Though fairness is outside the scope of our framework, our approach could highlight potential misuses.

Lastly, since algebraic machine reasoning is able to surpass human performance on the RPM task (an intelligence test originally designed to evaluate general human intelligence and abstract reasoning), our work is effectively cracking open one of the biggest bastions of human cognition. This may inadvertently contribute towards the fear by the general public, of AI outcompeting humans. One way to mitigate this is through education and outreach to demystify algebraic machine reasoning.

References

- [1] J. Abbott, A. M. Bigatti, and L. Robbiano. CoCoA: a system for doing Computations in Commutative Algebra. Available at <http://cocoa.dima.unige.it>. 4
- [2] Ibrahim Abdelaziz, Maxwell Crouse, Bassem Makni, Vernon Austel, Cristina Cornelio, Shajith Ikbal, Pavan Kapani-pathi, Ndivhuwo Makondo, Kavitha Srinivas, Michael Witbrock, et al. Learning to guide a saturation-based theorem prover. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022. 18
- [3] Krzysztof R Apt. Logic programming. *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics (B)*, 1990:493–574, 1990. 18
- [4] Dave Bayer, André Galligo, and Mike Stillman. Gröbner bases and extension of scalars. In *Computational algebraic geometry and commutative algebra (Cortona, 1991)*, Sympos. Math., XXXIV, pages 198–215. Cambridge Univ. Press, Cambridge, 1993. 6
- [5] Susan Carey. The origin of concepts. *Journal of Cognition and Development*, 1(1):37–41, 2000. 7, 8
- [6] Kai Chen, Jiaqi Wang, Jiangmiao Pang, Yuhang Cao, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jiarui Xu, et al. Mmdetection: Open mmlab detection toolbox and benchmark. *arXiv preprint arXiv:1906.07155*, 2019. 11

- [7] David A Cox, John Little, and Donal O'Shea. *Ideals, varieties, and algorithms: an introduction to computational algebraic geometry and commutative algebra*. Springer, 2015. 1, 2, 4
- [8] Wolfram Decker, Gert-Martin Greuel, Gerhard Pfister, and Hans Schönemann. SINGULAR 4-3-0 — A computer algebra system for polynomial computations. <http://www.singular.uni-kl.de>, 2022. 4
- [9] David Eisenbud. *Commutative algebra*, volume 150 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 1995. With a view toward algebraic geometry. 1, 2, 4, 5, 6
- [10] David Eisenbud, Craig Huneke, and Wolmer Vasconcelos. Direct methods for primary decomposition. *Invent. Math.*, 110(2):207–235, 1992. 5
- [11] The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.12.1*, 2022. <https://www.gap-system.org>. 4
- [12] Edmund Gettier. Is justified true belief knowledge? In *Arguing about knowledge*, pages 14–15. Routledge, 2020. 8
- [13] Patrizia Gianni, Barry Trager, and Gail Zacharias. Gröbner bases and primary decomposition of polynomial ideals. *J. Symbolic Comput.*, 6(2-3):149–167, 1988. Computational aspects of commutative algebra. 5
- [14] Alison Gopnik and Henry M Wellman. The theory theory. In *An earlier version of this chapter was presented at the Society for Research in Child Development Meeting, 1991*. Cambridge University Press, 1994. 7
- [15] Richard E. Grandy. Understanding and the principle of compositionality. *Philosophical Perspectives*, 4:557–572, 1990. 7
- [16] Daniel R. Grayson and Michael E. Stillman. Macaulay2, a software system for research in algebraic geometry. Available at <http://www.math.uiuc.edu/Macaulay2/>. 4
- [17] Serkan Hoşten and Gregory G Smith. Monomial ideals. In *Computations in algebraic geometry with Macaulay 2*, pages 73–100. Springer, 2002. 5
- [18] Sheng Hu, Yuqing Ma, Xianglong Liu, Yanlu Wei, and Shihao Bai. Stratified rule-aware network for abstract visual reasoning. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*, 2021. 9
- [19] Geoffrey Irving, Christian Szegedy, Alexander A Alemi, Niklas Eén, François Chollet, and Josef Urban. Deepmath-deep sequence models for premise selection. In *Advances in Neural Information Processing Systems*, pages 2235–2243, 2016. 18
- [20] Joxan Jaffar and J-L Lassez. Constraint logic programming. In *Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 111–119, 1987. 18
- [21] Hans Kamp and Barbara Partee. Prototype theory and compositionality. *Cognition*, 57(2):129–191, 1995. 7
- [22] Tim Kimber, Krysia Broda, and Alessandra Russo. Induction on failure: Learning connected horn theories. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 169–181. Springer, 2009. 18
- [23] Brenden M Lake, Tomer D Ullman, Joshua B Tenenbaum, and Samuel J Gershman. Ingredients of intelligence: From classic debates to an engineering roadmap. *Behavioral and Brain Sciences*, 40, 2017. 7
- [24] Guillaume Lample and François Charton. Deep learning for symbolic mathematics. In *International Conference on Learning Representations*, 2020. 18
- [25] Barbara Landau. Will the real grandmother please stand up? the psychological reality of dual meaning representations. *Journal of Psycholinguistic Research*, 11(1):47–62, 1982. 8
- [26] Wenda Li, Lei Yu, Yuhuai Wu, and Lawrence C Paulson. Isarstep: a benchmark for high-level mathematical reasoning. In *International Conference on Learning Representations*, 2021. 18
- [27] John W Lloyd. *Foundations of logic programming*. Springer Science & Business Media, 2012. 18
- [28] John Locke. An essay concerning human understanding, 1690. 1948. 7, 8
- [29] Barbara C Malt and Edward E Smith. Correlated properties in natural categories. *Journal of verbal learning and verbal behavior*, 23(2):250–269, 1984. 7
- [30] Eric Ed Margolis and Stephen Ed Laurence. *Concepts: Core Readings*. The MIT Press, 1999. 7, 8
- [31] Douglas L Medin, Edward E Smith, et al. Concepts and concept formation. *Annual review of psychology*, 35(1):113–138, 1984. 7
- [32] Ezra Miller and Bernd Sturmfels. *Combinatorial commutative algebra*, volume 227 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 2005. 2, 3
- [33] Stephen Muggleton. Inductive logic programming. *New generation computing*, 8(4):295–318, 1991. 18
- [34] Stephen Muggleton. Inverse entailment and progol. *New generation computing*, 13(3):245–286, 1995. 18
- [35] Gregory Murphy. *The big book of concepts*. MIT press, 2004. 7
- [36] Aditya Paliwal, Sarah Loos, Markus Rabe, Kshitij Bansal, and Christian Szegedy. Graph representations for higher-order logic and theorem proving. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(03):2967–2974, Apr. 2020. 18
- [37] Steven Pinker and Ray Jackendoff. The faculty of language: what's special about it? *Cognition*, 95(2):201–236, 2005. 7
- [38] Oliver Ray, Krysia Broda, and Alessandra Russo. Hybrid abductive inductive learning: A generalisation of progol. In *International Conference on Inductive Logic Programming*, pages 311–328. Springer, 2003. 18
- [39] Takeshi Shimoyama and Kazuhiro Yokoyama. Localization and primary decomposition of polynomial ideals. *J. Symbolic Comput.*, 22(3):247–277, 1996. 5
- [40] Richard P. Stanley. *Combinatorics and commutative algebra*, volume 41 of *Progress in Mathematics*. Birkhäuser Boston, Inc., Boston, MA, second edition, 1996. 3
- [41] Mike Stillman, Carolyn Yackel, Justin Chen, and Mahrud Sayrafi. PrimaryDecomposition: primary decomposition and associated primes routines. Version 2.0. A Macaulay2 package available at <https://github.com/Macaulay2/M2/tree/master/M2/Macaulay2/packages>. 5
- [42] Bernd Sturmfels. What is ... a Gröbner basis? *Notices Amer. Math. Soc.*, 52(10):1199–1200, 2005. 3

- [43] Henry Taylor and Peter Vickers. Conceptual fragmentation and the rise of eliminativism. *European Journal for Philosophy of Science*, 7(1):17–40, 2017. 7
- [44] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 9.6)*, 2022. <https://www.sagemath.org>. 4
- [45] Mingzhe Wang and Jia Deng. Learning to prove theorems by learning to generate theorems. In *Advances in Neural Information Processing Systems*, volume 33, pages 18146–18157, 2020. 18
- [46] Yuhuai Wu, Albert Jiang, Jimmy Ba, and Roger Baker Grosse. Int: An inequality benchmark for evaluating generalization in theorem proving. In *International Conference on Learning Representations*, 2021. 18
- [47] Chi Zhang, Feng Gao, Baoxiong Jia, Yixin Zhu, and Song-Chun Zhu. RAVEN: A dataset for relational and analogical visual reasoning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5317–5327, 2019. 9
- [48] Zsolt Zombori, Josef Urban, and Chad E Brown. Prolog technology reinforcement learning prover. In *International Joint Conference on Automated Reasoning*, pages 489–507. Springer, 2020. 18