A Program-Synthesis Challenge for ARC-like Tasks

Aditya Challa[1], Ashwin Srinivasan[1⋆], Michael Bain[2], and Gautam Shroff[3]

[1] Dept. of CSIS & APPCAIR, BITS Pilani, Goa Campus, India
[2] School of CSE, University of New South Wales, Sydney, Australia
[3] TCS Research, New Delhi, India

Abstract We propose a program synthesis challenge inspired by the Abstraction and Reasoning Corpus (ARC) [3]. The ARC is intended as a touchstone for human intelligence. It consists of 400 tasks, each with very small numbers (3–5) of 'input-output' image pairs. It is known that the tasks are 'human-solvable' in the sense that, for any of the tasks, there exists a human-authored description that transforms input images in the task to the corresponding output images. Besides the 'small data problem', other features of ARC make it hard to use as a yardstick for machine learning. The solutions are not provided, nor is it known if they are unique. The use of some basic prior knowledge is acknowledged, but no definitions are available. The solutions are known also to apply to images that may be significantly different to those provided, but those images are not described. Inspired by ARC, but motivated to address some of these issues, in this paper we propose the Inductive Program Synthesis Challenge for ARC-like tasks (IPARC). The IPARC challenge is much more controlled, focusing on the inductive synthesis of structured programs. We specify for the challenge a set of 'ARC-like' tasks characterised by: training and test example sets drawn from a clearly-defined set of 'ARC-like' input-output image pairs; a set of image transformation functions from the image-processing field of Mathematical Morphology (MM); and target programs known to solve the tasks by transforming input to output images. The IPARC tasks rely on a result known as the 'Structured Program Theorem' that identifies a small set of rules as sufficient for construction of a wide class of programs. Tasks in the IPARC challenge are intended for machine learning methods of program synthesis able to address instances of these rules. In principle, Inductive Logic Programming (ILP) has the techniques needed to identify the constructs implied by the Structured Program Theorem. But, in practice, is there an ILP implementation that can achieve this? The purpose of the IPARC challenge is to determine if this is the case.

Keywords: Program synthesis · IPARC · ILP

⋆ AS is currently visiting the Centre for Health Informatics, Macquarie University and the School of CSE, UNSW.

1 Introduction

Perhaps the earliest demonstration of inductive program synthesis using images was by Evans [4]. ANALOGY solved tasks of the kind shown in Fig. 1. Computation in ANALOGY proceeded in two stages: a representation stage, that detected objects and extracted represented them using pre-defined basic functions and relations. The second stage then proceeded to find an appropriate program. The first step in the program-synthesis process was to find automatically one or more rules–in ANALOGY, LISP programs–that describe (in Evans' words): "how the objects of figure A are removed, added to, or altered in their properties and their relations to other objects to generate B". Generalisations of the rules are then constructed to ensure that C is transformed to exactly one of the alternatives 1–5. If multiple programs are possible, then ANALOGY used a notion of *generalisation strength* to select one[4].

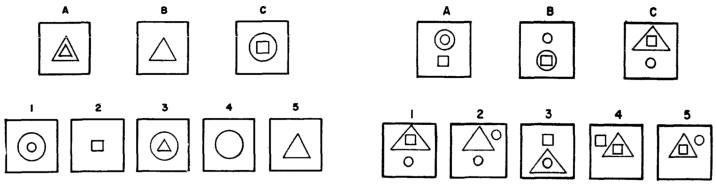


Figure 1. Examples for program-synthesis from [4]. Images are all line-drawings, and the tasks considered were all of the kind "if A:B then which one of the following is true: (a) C:1; (b) C:2; $\cdots$; (e) C:5".

Now, nearly six decades later, a significantly more complex and wider variety of visual tasks, with some overlapping motivations to those of Evans' paper, has been proposed in the Abstraction and Reasoning Corpus (ARC [3]). The ARC consists of 400 tasks of the kind shown in Fig. 2.
The tasks in the ARC are not especially envisaged as being exclusively for program synthesis. What is known is the the tasks are 'human-solvable' in the sense that, for any of the tasks, there exists a human-authored procedure that transforms input images in the task to the corresponding output images. But, as noted by Francois Chollet in his description of the ARC:

> Crucially, to the best of our knowledge, ARC does not appear to be approachable by any existing machine learning technique (including Deep Learning), due to its focus on broad generalization and few-shot learning, as well as the fact that the evaluation set only features tasks that

[4] in general, such analogical tasks can be 'ill-posed', in that multiple solutions might exist for all training examples but only one of these is correct for unseen test cases, with the criteria as to which to choose being impossible to 'logically' deduce from the training examples alone.

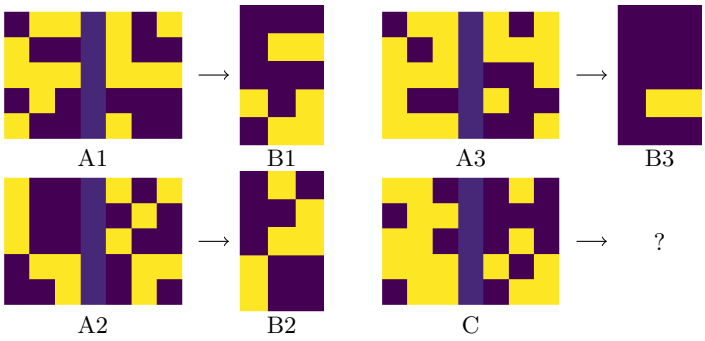


Figure 2. An example task from the ARC (taken from https://samacquaviva.com/LARC/explore/). The task to be solved is of the form "if A1:B1 and A2:B2 and A3:B3 then C: ?".

> do not appear in the training set. For a researcher setting out to solve it, ARC is perhaps best understood as a program synthesis benchmark.

However, there are several aspects that pose a challenge when considering the ARC as a benchmark for program synthesis. The human-authored solutions are not provided, nor is it known if they are unique. The use of some basic prior knowledge is acknowledged, but no definitions are available. The solutions are known also to apply to images that may be significantly different to those provided, but those images are not described. Two different routes have been sought to address these limitations. The authors of [1] re-interpret the ARC tasks in terms of language. Solutions are 'natural programs' in the form of instructions in a natural language. A natural program for a task is first obtained from a human participant. This natural language description is then tested by a second participant. A natural program is considered successful if it can be used by the second participant to solve the task. The intent is that a collection of successful natural programs will allow the identification of prior knowledge that can subsequently be used as prior knowledge for inductive program synthesis by machine. Separately, a Kaggle competition[5] focussed on automated techniques for solving the ARC without reference to any prior knowledge, but providing the usual machine-learning artefacts of training- and test-data; and allowing the possibility that solutions may not be unique.

Inspired by ARC, but in a more controlled setting, we propose the Inductive Program Synthesis Challenge for ARC-like tasks (IPARC). We are particularly interested in the identification of structured programs. For the challenge, we define a set of 'ARC-like' tasks characterised by: (a) *Data*, consisting of training- and test-sets for constructing and testing programs, with images drawn from a clearly defined set (which we call 'ARC-like' images); (b) *Basic functions*, drawn from four different categories of image-transformation operations (mathematical morphology operations, re-colouring, re-sizing, and endomorphisms), that we

[5] https://www.kaggle.com/c/abstraction-and-reasoning-challenge

show are sufficient for constructing a transformation between any single pair of input-output ARC-like images; and (c) *Target programs*, known to solve the tasks in the sense of being able to transform input images to output image in both training- and test-sets.

The tasks in the IPARC rely on a result from structured program-construction that identifies three rules of grammar (encoding sequence, selection and and iteration) as being sufficient for representing a wide class of programs. It is possible to view any instance of these rules within a program as constituting a 'subprogram', with a meaning given by a mathematical (sub-)function defined using some pre-defined basic functions. The overall computation of the program is then a (mathematical) composition of (sub-)functions. The tasks in the IPARC challenge are intended for machine learning methods of program-synthesis that can address instances of these three rules. It is useful to clarify immediately what is not being claimed in this paper. Given the open-ended nature of ARC, we do not claim solving tasks in the IPARC is necessary or sufficient to solve tasks in the ARC. However, it seems plausible that a machine-learning approach equipped with techniques for addressing the IPARC would be able to re-use those techniques to good effect to address tasks in the ARC.

There are good reasons to consider ILP as being well-suited to address tasks in the IPARC: (a) Sequences, selection and iteration can all be represented as logic- (or functional-)programs; (b) Basic functions can be provided as background knowledge to an ILP system; (3) Identification of sub-functions corresponds to the invention of new predicates by an ILP system; and (4) Identification of iteration can be achieved through the construction of recursive statements by an ILP system. In principle therefore, the tasks in the IPARC challenge should be solvable by an ILP system. But, in practice is there a single ILP system or approach that can effectively construct structured programs? The purpose of the IPARC challenge is to determine if this is the case.

The rest of the paper is organised as follows. Section 2 provides an introduction to the area of morphological filters that form a significant component of the background knowledge for the tasks we propose later. Section 3.3 describes a set basic functions for ARC-like tasks. Section 4 describes the 'IPARC' challenge that contains 3 categories of tasks aimed at testing the capabilities of current ILP systems to both invent new functions and synthesise complete programs. Section 5 concludes the paper.

2 Mathematical Morphology (MM)

Mathematical Morphology (MM) is field developed during the 1960s [9] for the purpose of analysis of petrographic samples. However, it has evolved into a strong image processing framework based on non-linear operators. In this section, we review three important operators from MM for binary images relevant to this paper. For details regarding other operators and theory of MM refer to [7,8]. Before describing the operators we make some remarks:

– All MM operators in this paper operate with *two* images — one is considered to be the input image and the other, referred to as the *structuring element* (s.e.), is used to "probe" the input to generate the output;

- Note that a binary image can be written as a function $f : \mathbb{Z}^2 \to \{0, 1\}$. Hence, each pixel within the image can be "addressed" using pairs of integers. The set of pixels which take the value 1 are referred to as *foreground* and the set of pixels which take the value 0 are referred to as *background*;
- The structuring element (s.e.) is defined with an origin $(0, 0)$, usually identified with a $\bigcirc$ at the pixel. The other pixels are accordingly addressed using integers. For instance, the pixel to the left of the origin is given the address of $(-1, 0)$, and so on.
- Also, observe that each binary image can be equivalently represented as a set by considering the set of pixels with value 1. We will use binary images and sets interchangeably in this paper. Thus, union of two images refers to the union of sets equivalent to the image, and intersection of two images refers to the intersection of sets equivalent to the image. Note that these are only valid when the images are of same size;
- It is useful to think of MM operators as belonging to the class of functions $\mathcal{I} \times \mathcal{F} \times \mathcal{B} \to \mathcal{I}$. Here $\mathcal{I}$ is a set of images, $\mathcal{F}$ is a set of Boolean functions, and $\mathcal{B}$ is a set of s.e's. Any particular MM operator usually involves a specific Boolean function $F \in \mathcal{F}$ and a specific s.e. $B \in \mathcal{B}$. The resulting operator is then $\mu_{F,B} : \mathcal{I} \to \mathcal{I}$. When one or the other or both of F, B is obvious from the context, it is dropped from the suffix of the operator.

For reasons of space, we direct the reader to the IPARC website for informal descriptions of some commonly used MM operators.

3 Program Synthesis for ARC-like Tasks

3.1 ARC-like Images and Tasks

We first develop an abstract characterisation of what we will call ARC-like' images (the corresponding tasks being ARC-like tasks). Let $I^k_{m,n}$ denote a particular image, where (m, n) denotes the size of the grid and k denotes the set of possible colours each pixel can take. This image can be decomposed as follows :

$$I^k_{m,n} = \langle I^{(1)}_{m,n}, I^{(2)}_{m,n}, \cdots, I^{(k)}_{m,n} \rangle \tag{1}$$

where $I^{(i)}_{m,n}$ denotes a binary image (only possible values are $0, 1$) of colour i. Let

$$\mathcal{L}^k_{m,n} = \{I^k_{m,n} \mid m, n \text{ fixed and each pixel has value 1 in at most one of } \langle I^{(j)}_{m,n} \rangle^k_{j=1}\} \tag{2}$$

We distinguish $\mathcal{L}^k_{m,n}$ from:

$$\tilde{\mathcal{L}}^k_{m,n} = \{I^k_{m,n} \mid m, n \text{ fixed where each of } \langle I^{(j)}_{m,n} \rangle^k_{j=1} \text{ can be any binary image.}\} \tag{3}$$

That is, the set $\tilde{\mathcal{L}}$ contains images which can have multiple colours associated with any pixel. We will denote the set of 'ARC-like' images as $\mathcal{L} = \bigcup_{m,n} \mathcal{L}^k_{m,n}$.

An 'ARC-like' task T is a finite set $\{(i, o) : i, o \in \mathcal{L}\}$. Usually, $|T|$ will be some small number, like three. In this paper, program-synthesis for an ARC-like task T will mean constructing a sequence of functional compositions that identifies a many-to-one mapping $f : \mathcal{I} \to \mathcal{I}$ s.t. for every $(i, o) \in T$, $f(i) = o$. Additionally, we will require the function f to be "general enough" to be correct on additional sets of input images *not* used in the construction of f.

3.2 Functional Categories for ARC-like Tasks

Given colours $1 \dots k$ recall that all the input/output images in an ARC-like task belong to the space $\mathcal{L} = \bigcup_{m,n} \mathcal{L}^k_{m,n}$, where $\mathcal{L}^k_{m,n}$ is the set of all colour-images of a (fixed) size m, n. Also, for a fixed m, n let $\mathcal{I}^{(j)}_{m,n}$ denote the set of $m \times n$ binary images of colour j. In this paper, we propose functions in the following basic categories for solving ARC-like tasks:

Structure. Functions in this category are concerned with changing the structure of the image. We provide definitions for the following kinds of functions:

1. For each colour $j = 1 \dots k$, colour-specific MM-operators are functions of the form $\phi^{(j)} : \mathcal{I}^{(j)}_{m,n} \to \mathcal{I}^{(j)}_{m,n}$ [6]
2. An overall structuring operation is a function $\phi : \mathcal{L}^k_{m,n} \to \tilde{\mathcal{L}}^k_{m,n}$ defined as follows:

$$\phi(\langle I^{(1)}_{m,n}, \cdots, I^{(k)}_{m,n} \rangle) = \langle \tilde{I}^{(1)}_{m,n}, \cdots, \tilde{I}^{(k)}_{m,n} \rangle$$

where $\tilde{I}^{(j)}_{m,n} = \phi^{(j)}(I^{(j)}_{m,n})$. Note that the l.h.s. is the same as $\phi(I^k_{m,n})$ and is an element of $\tilde{\mathcal{L}}^k_{m,n}$. That is, a ϕ operation can assign more than one colour to a pixel.

Colour. Functions in this category either resolve conflicting assignments of colours to the same pixel, or perform consistent re-colourings. For some m, n this category contains functions of the form $\gamma : \tilde{\mathcal{L}}^k_{m,n} \to \mathcal{L}^k_{m,n}$.

Size. One or more re-sizing functions of the form $\psi : \mathcal{L}^k_{m_1,n_1} \to \mathcal{L}^k_{m_2,n_2}$. These change the size of the grids of an image $I^k_{m_1,n_1}$ to $I^k_{m_2,n_2}$ where $(m_1, n_1) \neq (m_2, n_2)$ [7].

It is useful also to distinguish an overall structure-and-colour operation $\Phi : \mathcal{L}^k_{m,n} \to \mathcal{L}^k_{m,n}$ defined as:

$$\Phi(I^k_{m,n}) = \gamma(\phi(I^k_{m,n}))$$

$\Phi(\cdot)$ transforms an image in the manner shown in Fig. 3.

We define one further category of operations based on $\Phi(\cdot)$:

[6] Correctly, the function is $\phi^{(j)}_{F_j,B_j}$, where F_j is the Boolean function used by the MM operator, and B_j is the set of s.e's used by the operator.

[7] That is, either $m_1 \neq m_2$ or $n_1 \neq n_2$

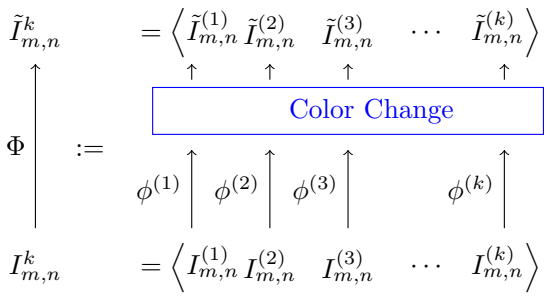


Figure 3. Illustrating the Φ operator.

Endomorphisms. Two specific endomorphic functions are defined on $\mathcal{L}^k_{m,n}$:

$$\oplus(I^k_{m,n}) = I^k_{m,n} \cup \Phi(I^k_{m,n})$$

and

$$\otimes(I^k_{m,n}) = I^k_{m,n} \cap \Phi(I^k_{m,n})$$

where,

$$\begin{aligned} I^k_{m,n} \cup \phi(I^k_{m,n}) &= \langle I^{(1)}_{m,n} \cup \tilde{I}^{(1)}_{m,n}, I^{(2)}_{m,n} \cup \tilde{I}^{(2)}_{m,n}, \cdots, I^{(k)}_{m,n} \cup \tilde{I}^{(k)}_{m,n} \rangle \\ I^k_{m,n} \cap \phi(I^k_{m,n}) &= \langle I^{(1)}_{m,n} \cap \tilde{I}^{(1)}_{m,n}, I^{(2)}_{m,n} \cap \tilde{I}^{(2)}_{m,n}, \cdots, I^{(k)}_{m,n} \cap \tilde{I}^{(k)}_{m,n} \ \rangle \end{aligned} \tag{4}$$

Although not explicitly defined as such, we will take each of Structure, Colour, Size and Endomorphisms to be sets of functions, and denote $\Sigma = \mathit{Structure} \cup \mathit{Colour} \cup \mathit{Size} \cup \mathit{Endomorphisms}$. We claim that these sets are *sufficient* for a restricted form of ARC-like tasks. Specifically, we have the following proposition.

Proposition 1. *If $I^k_{m_1,n_1}, \tilde{I}^k_{m_2,n_2}$ are from $\bigcup \mathcal{L}^k_{m,n}$ and $I^k_{m_1,n_1}$ is non-empty, then there exists a sequence of functions $\langle f_1, f_2, \ldots, f_N \rangle$ s.t. each $f_i \in \Sigma$ and $\tilde{I}^k_{m_2,n_2} = f_N(f_{N-1} \cdots (f_1(I^k_{m_1,n_1})) \cdots)$.*

Proof. See Appendix A.

We will use $\tilde{I} =_\Sigma f(I)$ to denote that given an an input-output pair $(I, \tilde{I})$, $\tilde{I}$ is obtained by the composition f of some functions from Σ applied to I.

The result in Proposition 1 is useful in that it identifies the kinds of functions needed for ARC-like tasks. But the result does not immediately tell us what the $\langle f_i \rangle_1^N$ are, because: (a) Other than Φ, $\oplus$, and $\otimes$, all other functions remain unspecified. These are the $\phi^{(j)}_{m,n}$-functions in Structure, the γ-functions in Colour and the ψ-functions in Size; (b) Even if all the functions were specified, the proof does not provide a constructive method to identify the sequence; and (c) Even if an appropriate sequence for an image-pair was found, this may still not be sufficient for solving an ARC-like task T. We would like a function sequence to be 'general enough' to apply to all image-pairs in T. We will have more to say on this shortly.

We have minimally to address each of these problems when identifying a solution for ARC-like tasks.

3.3 ILP-based Program Synthesis for ARC-like Tasks

Before we consider the use of ILP for solving ARC-like tasks it is useful to clarify what exactly we will mean by a program and the computation it performs. In this paper, we will adopt the position taken in the Structured Program Theorem (or the Böhm-Jacopini Theorem [2]), according to which structured programs can be constructed solely by combining subprograms containing sequence, selection, or iteration. We will further assume each subprogram can be mathematically represented by a function, which we will call a sub-function, and that a program's computation given some input is equivalent to the application of the composition of the sub-functions to the input.

We turn now to an ILP formulation of identifying such compositions. A first-cut at an ILP formulation is just this:

Given: the set of basic functions B as background knowledge, and examples E of input-output images,
Find: a program H s.t. $B \wedge H \models E$.

For ARC-like tasks, we will assume the basic functions will include well understood image-manipulation functions from the sets Structure, Colour, Size, and Endomorphisms categories. Also, we restrict ourselves to the standard MM operators of *Dilation* and *Erosion* with different structuring elements to model the Structure operators. The structuring elements to be used are those shown in Fig. 4. The set of functions Σ_1 consists of: (1) *Dilation* and *Erosion* with eight different structuring elements as shown in Figure 4; (2) *Colour* and *Size* operations as described above; and (3) *Endomorphisms* as described above. The functions in Σ_1 are available at: `https://ac20.github.io/IPARC/Challenge/`.

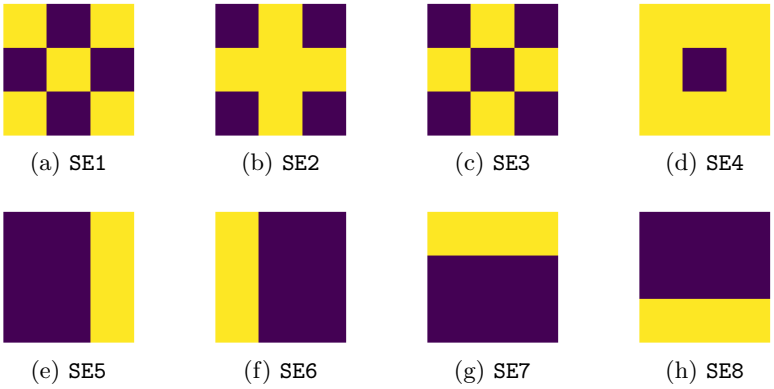


Figure 4. The eight structuring elements (s.e.'s) used in this paper, where yellow pixels denote value 1 and purple denotes 0. Each s.e. is a 3×3 grid. Functions in Σ_1 comprise the `Dilation` and `Erosion` operators with any of these structuring elements, along with Color, Size and Endomorphism operators (see text for details).

Even though Σ_1 is a restriction of Σ, it remains challenging as we note with the following proposition.

Proposition 2. *Given the set of functions Σ_1 there exists an ARC-like task T that contains elements $(I_1, \tilde{I}_1)$, $(I_2, \tilde{I}_2)$, s.t. $\tilde{I}_1 =_{\Sigma_1} f(I_1)$, and $\tilde{I}_2 =_{\Sigma_1} g(I_2)$ but there is no h s.t. $\tilde{I}_1 =_{\Sigma_1} h(I_1)$ and $\tilde{I}_2 =_{\Sigma_1} h(I_2)$.*

Proof. See Appendix A.

That is, there may not be any single composition sequence of functions from Σ_1 that can solve all examples in an ARC-like task.[8] It follows that solving a given set of ARC-like tasks may require enlargement of the set of functions Σ_1. Here we will restrict ourselves to enlarging the existing set of functions Σ_1 by constructing new functions that re-use elements of Σ_1 within the constructs described by the Structured Program Theorem, namely sequence, selection and iteration. Specifically, we will require that the new functions of these kinds should be constructed automatically by the program-synthesis engine (in ILP terminology, we require Σ_1 to be augmented by predicate-invention only). In turn, this means that the background knowledge B in the ILP formulation of the program-synthesis task will need to be augmented with definitions for sequence, selection and iteration to allow inference; and perhaps also meta-information that can help learn these control structures from data. The former is straightforward, the latter is less obvious.

Remark: Note that all problems in IPARC can be solved using Σ_1^+ (Σ with predicate-invention), even though not all (i,o) pairs from $\mathcal{L}$ would have a solution in Σ_1^+. (From proposition 1, we have Σ is sufficient to obtain a solution to an arbitrary (i,o) pair.)

4 IPARC

The Inductive Program Synthesis Challenge for ARC-like Tasks, or IPARC, is intended to assess the current state of readiness of ILP systems to address challenges posed by the synthesis of structured programs. Specifically, we structure tasks in the IPARC in the following way: (A) Tasks that require simple sequence of steps; (B) Tasks that augment techniques in (A) with techniques for synthesis of sub-programs; and (C) Tasks that augment techniques in (B) with techniques for constructing full programs from sub-programs.

Below we describe the categories of tasks in the IPARC and provide examples in each category. Within each category, we distinguish between "easy" and "hard" variants.[9] In all cases, programs that solve the tasks are generated automatically. The programs serve two purposes: they serve as potential 'target programs' for the program-synthesis engines; and they can be used as generators

[8] This is consistent with the Böhm-Jacopini result, which says that more than just sequence will be needed.

[9] For reasons of space, a complete listing of tasks in each category is not provided. Instead, we describe the requirements in each category, and provide some illustrative examples. A complete listing will be available at the IPARC website.

of data. The latter may prove useful for approaches that require more data than are provided.

Remark 1 *Assuming that the length of sequence of operators is l and number of options for each operator is d, the size of this set is $\mathcal{O}(d^l)$. So, the complexity increases exponentially with the length. For IPARC tasks, we seek to have the length between $4-8$. For instance, the Structure functions we have chosen to strictly alternate between* `Dilation` *and* `Erosion`*, starting with* `Dilation`*, and each operator has a choice of eight structuring elements (illustrated in Figure 4). Hence d in this case is 8. So, the size of the search space is of the order 2^{24}.*

4.1 Category A: Warm-Up

Category A tasks consist simply of learning generalisations by functional compositions of relations in Σ_1. The tasks are in two sub-categories: (a) **Easy.** Solutions of tasks consist of programs that require relations from the categories of Structure and Endomorphisms; and (b) **Hard.** Solutions of tasks consist of programs that require relations from the entire set in Σ_1.

In principle, all programs in Category A can be obtained by a generate-and-test approach, with some simple generalisation over parameters used by the background relations. However the size of the search-space (see Remark 1) may preclude such a brute-force approach. Figure 5 illustrates a simple program from category A.

(a) (b) (c) (d) (e)

Figure 5. Example images from a program used to generate data in Category A, where (a) shows a sample input image, (b) shows the image obtained after operating on (a) with `Dilation_SE4`, (c) shows the image after operating on (b) with `Erosion_SE5`, (d) shows the image after operating on (c) with `Dilation_SE1`, and (e) shows the image after operating on (d) with `Erosion_SE3`.

4.2 Category B: Learning Sub-Programs

Proposition 2 suggests that in general, an ability to invent new functions may be needed to construct programs that explain multiple instances in an ARC-like task. Here we consider a simpler but nevertheless still useful variant of the general predicate-invention problem. All the tasks we consider here are solvable as in Category A, but inventing one or more new functions defined in terms of functions already in Σ_1 will allow a more compact definitions across tasks,

due to reuse of definitions. Within this restricted setting, we propose tasks that require the invention of predicates corresponding to subsets of functions in Σ_1.

For all tasks in this category, systems are given a set of ARC-like tasks $\{T_1, T_2, \ldots, T_N\}$. Let $H_1, H_2, \ldots, H_N$ be individual programs obtained independently for each task using background relations in Σ_1. The tasks involve automatically augmenting Σ_1 to $\Sigma_1^+ \supset \Sigma_1$ and obtaining programs $H'_1, H'_2, \ldots, H'_N$ such that $(|\Sigma_1 \cup H_i \cup H_2 \cdots \cup H_N|) > (|\Sigma_1^+ \cup H'_1 \cup H'_2 \cdots \cup H'_N|)$, where $|\cdot|$ denotes some measure of size.

Augmentations correspond to sub-programs, and in an ILP setting corresponds to invention of the new predicates. We propose tasks that benefit from the following kinds of predicate-invention:

(a) Invention of Sequence. The predicate to be invented can be seen as to be of the kind used in the inter-construction operation of inverse resolution [6]. It may be helpful to think of this form of predicate-invention as 'slot-filling' in the value of NewP in the following higher-order statement (the syntax is Prolog-ish; `X`, `Y` are images):

```
newp(X,Y,NewP):- NewP(X,Y).
```

New predicate definitions are therefore similar to the programs obtained in Category A, except they may only represent some sub-computation. As with Category A, we will distinguish between Easy and Hard variants, where specifications of these sub-categories will be available on the IPARC website.

(b) Invention of Selection. Here predicate-invention is used to represent conditionals. As above, it may be helpful to see this to mean slot-filling the values of `Q`, `NewP1`, and `NewP2` in the following Prolog-like higher-order statement:[10]

```
if_else(X,Y,Q,NewP1,NewP2):- (Q(X) -> NewP1(X,Y); NewP2(X,Y)).
```

In all tasks, we will assume `Q` to be restricted to one of the Hit-or-Miss relations in Σ_1. We again propose to distinguish between Easy and Hard variants; we refer the reader to the IPARC website for details.

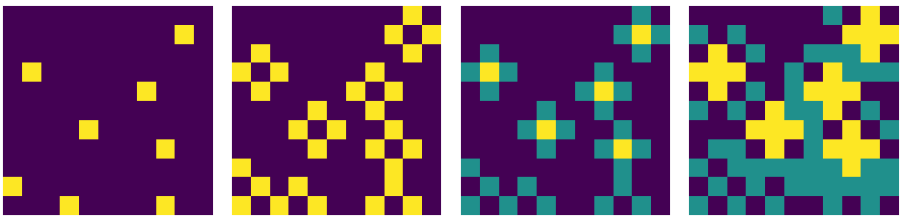
(c) Invention of Iteration. We will restrict iteration to bounded repetition. corresponding to slot-filling `K` and `NewP` in:

```
repeat_k(X,Y,K,NewP):-
  (K > 0 -> (NewP(X,Y1), K1 is K - 1, repeat_k(Y1,Y,K1,NewP)); Y=X).
```

Again, some tasks will involve the invention of more complex iterators than others.

In each case, the benefit of predicate-invention has to be decided automatically, based on: (a) the repeated occurrence on some subset of tasks provided; and (b) the trade-off of increasing the size of Σ_1 with the addition of new functions and the decrease in overall size of the programs for all tasks. Figure 6 shows example of **Selection** predicate-invention.

[10] `A->B;C` is to be read as "if `A` then `B` else `C`"; `X`, `Y` are images, `Q` is a variable, standing for any Boolean function of `X` from Σ_1, `NewP1` and `NewP2` are invented predicates.



(a) (b) (c) (d)

Figure 6. Example of Selection predicate-invention in Category B. (a) indicates the input image. (b) is obtained using the `Dilation_SE3` operator on (a). (c) indicates the pixels selected using `Hit_Or_Miss_SE3`. Recall that `Hit_Or_Miss` selects those pixels which fit a particular pattern and is used to simulate the Conditional `Q`. (d) shows the final image where pixels identified by the conditional are transformed using `Dilation_SE1` and the rest of the pixels are transformed using `Dilation_SE2`. That is, we have `NewP1` = `Dilation_SE1` and `NewP2` = `Dilation_SE2`.

4.3 Category C: Learning Programs from Traces

This category is concerned with the connecting sub-programs using a sequence of examples describing a computation trace. This requires us to generalise the definition of ARC-like tasks to include data in a manner akin to inductive program synthesis by demonstration (PBD [5]). Instead of a task containing a set pairs of input-output images, it is now a set of sequences of images. Each sequence represents a computation by the program from an input-image to an output-image.[11] Given a set of tasks $\{T_1, T_2, \ldots, T_N\}$, the requirement is to find a program that correctly derives all the intermediate input-output pairs of images. Suppose the computation for task T_j is represented by the sequence of images $\langle I_{j,1}, I_{j,2}, \ldots, I_{j,n_k}\rangle$. Then each pair $(I_{j,k}, I_{j,k+1})$ in this sequence can be seen as a specification for a Category B task. Several (sub-)programs may synthesised could solve this task. In principle, a complete solution for a set of tasks in Category C can be found by identifying and combining alternative sub-programs for all such image-pairs, possibly with some generalisation. Again, the combinations possible may preclude such a brute-force approach. Tasks within the category C would provide snapshots after each sub-program (belonging to Category A or B) similar to the one provided in figures 5 and 6.

Website for the dataset

The dataset and the generating procedures will be hosted on https://github.com/ac20/IPARC_ChallengeV2.

5 Concluding Remarks

Inductive program synthesis, especially using small data sets exemplifying visual analogy problems has a surprisingly long history in Computer Science. The Ab-

[11] This is clearly more information than provided in in Categories A and B, and, in that sense, Category C tasks can be seen as providing more constraints on the program-synthesis problem.

straction and Reasoning Corpus (ARC) continues this long tradition. The ARC presents a formidable challenge to the use of machine learning techniques for program synthesis: it is abstract, dealing with apparently arbitrary visual patterns; the data are extremely limited; it requires some core background knowledge, which may be necessary but not sufficient. Finally, the programs are required to be applicable on data which may look substantially different. No significant details are available on either the background knowledge or the new data, and all that is known is that all tasks in the ARC have been solved manually (but the solutions are not known). While the ARC represents a laudable goal to achieve, we have used it as inspiration for the Inductive Program Synthesis Challenge for ARC-like Tasks (IPARC). The IPARC presents a controlled setting directly concerned with the automated synthesis of programs. We propose a definition of 'ARC-like' images, identify a set of 'ARC-like' tasks, and provide background knowledge, with the following properties: (a) All images in the ARC are also in the set of ARC-like images; (b) The tasks are concerned with identifying constructs known to be sufficient for a large class of structured programs; and (c) The background knowledge is provably sufficient for any specific pair of images in the data provided for each ARC-like task. We provide training- and test-data sets for all tasks, and also present programs that correctly transform input-images to output-images (although these may not be the only solutions).

We note here that the background knowledge proposed for solving IPARC tasks may not immediately useful for ARC tasks. For example, consider the case that there exists a unique program in Σ_1^+, (that is, functions in Σ_1 and augmented with predicates invented for specific compositions, selection and iteration) that solves every training example of an ARC task. An IPARC solver will find this program, but this program may still fail for the test example(s) in the ARC task, with the 'true' solution program being outside Σ_1^+, which uses the hitherto unspecified and possibly unknown set of 'core knowledge' primitives that ARC tasks are based on. This does not of course invalidate the utility of an ILP engine capable of starting with some set background primitives and enlarging that set to construct programs automatically from examples.

Despite its restricted setting, tasks in the IPARC are still non-trivial. They require the identification of sub-programs consisting of sequences of functional transformations, conditionals and iteration. In principle, techniques have been proposed and demonstrated within ILP systems for addressing each of these requirements. But these have been over nearly 4 decades of conceptual development in the use of domain-knowledge, predicate-invention, and learning recursive programs. What about in practice? That is, are there any current ILP systems that can address the type of problems posed in the IPARC? We invite ILP practitioners to participate in the IPARC and showcase the program-synthesis capabilities of their ILP system.

Acknowledgements

AS is a Visiting Professor at Macquarie University, Sydney. He is also the Class of 1981 Chair Professor at BITS Pilani, Goa, the Head of the Anuradha and Prashant Palak-

urthi Centre for AI Research (APPCAIR) at BITS Pilani, and a Research Associate at TCS Research.

References

1. Acquaviva, S., Pu, Y., Kryven, M., Wong, C., Ecanow, G.E., Nye, M., Sechopoulos, T., Tessler, M.H., Tenenbaum, J.B.: Communicating natural programs to humans and machines. arXiv preprint arXiv:2106.07824 (2021)
2. Böhm, C., Jacopini, G.: Flow Diagrams, Turing Machines and Languages with Only Two Formation Rules. Communications of the ACM **9(5)**, 366–371 (1966)
3. Chollet, F.: On the measure of intelligence. CoRR **abs/1911.01547** (2019), http://arxiv.org/abs/1911.01547
4. Evans, T.G.: A heuristic program to solve geometric-analogy problems. In: Fischler, M.A., Firschein, O. (eds.) Readings in Computer Vision, pp. 444–455. Morgan Kaufmann, San Francisco (CA) (1987). https://doi.org/https://doi.org/10.1016/B978-0-08-051581-6.50046-5, https://www.sciencedirect.com/science/article/pii/B9780080515816500465
5. Lau, T.A., Domingos, P.M., Weld, D.S.: Learning programs from traces using version space algebra. In: Gennari, J.H., Porter, B.W., Gil, Y. (eds.) Proceedings of the 2nd International Conference on Knowledge Capture (K-CAP 2003), October 23-25, 2003, Sanibel Island, FL, USA. pp. 36–43. ACM (2003). https://doi.org/10.1145/945645.945654, https://doi.org/10.1145/945645.945654
6. Muggleton, S.H., Raedt, L.D.: Inductive logic programming: Theory and methods. J. Log. Program. **19/20**, 629–679 (1994). https://doi.org/10.1016/0743-1066(94)90035-3, https://doi.org/10.1016/0743-1066(94)90035-3
7. Najman, L., Talbot, H.: Mathematical Morphology: from theory to applications. ISTE-Wiley (Jun 2010). https://doi.org/10.1002/9781118600788, https://hal-upec-upem.archives-ouvertes.fr/hal-00622479, iSBN: 9781848212152 (520 pp.)
8. Soille, P.: Morphological Image Analysis. Springer Berlin Heidelberg (2004). https://doi.org/10.1007/978-3-662-05088-0, http://dx.doi.org/10.1007/978-3-662-05088-0
9. Soille, P., Serra, J. (eds.): Mathematical Morphology and Its Applications to Image Processing. Springer Netherlands (1994). https://doi.org/10.1007/978-94-011-1040-2, http://dx.doi.org/10.1007/978-94-011-1040-2

A Proofs

We prove Proposition 1 by using the following two lemmas.

Lemma 1. *Given any $m_1 \neq m_2$ and $n_1 \neq n_2$, there exists a sequence of ψ transitions which take an image from $\mathcal{L}_{m_1,n_1}$ and returns an image from $\mathcal{L}_{m_2,n_2}$.*

Lemma 2. *Given any two images $I^k_{m,n}, \tilde{I}^k_{m,n} \in \mathcal{L}^k_{m,n}$, one can find a sequence of transitions within ϕ operators which takes in as input $I^k_{m,n}$ and returns $\tilde{I}^k_{m,n}$.*

From lemmas 1 and 2, it is clear that there exists a sequence of transitions which takes in as input $I^k_{m_1,n_1}$ and returns $\tilde{I}^k_{m_2,n_2}$. We first prove lemma 1

Proof (Proof of lemma 1). The proof follows from the observation that one can find an integer p such that $pm_1 \geq m_2$ and $pn_1 \geq n_2$. Thus, consider the resizing operator $\psi^{1,p}$ which copies the image $I^k_{m_1,n_1}$ on a grid of size $p \times p$, followed by $\psi_{(0,0),(m_2,n_2)}(.)$ which crops the image with corners $(0,0)$ and (m_2,n_2). This achieves the desired result.

Proof (Proof of lemma 2). Observe the following - Given any two binary images (of same shape) I_1, I_2, with I_1 as non-zero, we have that,

$$\delta_{I_2}\epsilon_{I_1}(I_1) = I_2 \tag{5}$$

Case A: If all of $I^{(i)}_{m,n}$ (refer figure 3.2) are non-empty, we have that there exists a transition $\phi^{(i)}$ which takes $I^{(i)}_{m,n}$ to output $\tilde{I}^{(i)}_{m,n}$.

Case B: Let $I^{(i_0)}_{m,n}$ be empty for some colour i_0. Since $I^{k}_{m,n}$ is non-empty, there exists an i' such that $I^{(i')}_{m,n}$ is non-empty. We define the operator ϕ as follows - $\phi^{(i)} = id$, where id denotes the identity map, and the colour change c maps colour i' to i_0. Then the proof is as in Case A.

Proof (Proof of proposition 2). The proof of proposition 2 follows from the following observation - Any given transition in Σ_1 either

- Takes an element from $\mathcal{L}^k_{m,n}$ to itself, or
- Takes an element from $\mathcal{L}^k_{m,n}$ to $\tilde{\mathcal{L}}^k_{m,n}$, or
- Takes an element from $\mathcal{L}^k_{m,n}$ to $\mathcal{L}^k_{m',n'}$ where $m' \neq m$ and $n' \neq n$.

Thus, given any I_1, I_2 and f we have that the shape of $(\tilde{I}_1) =_\Sigma f(I_1)$ should be the same as $(\tilde{I}_2) =_\Sigma f(I_2)$.

However, it is possible for an ARC-like task to have different shapes for $(\tilde{I}_1)$ and $(\tilde{I}_2)$. An example of this is shown below.

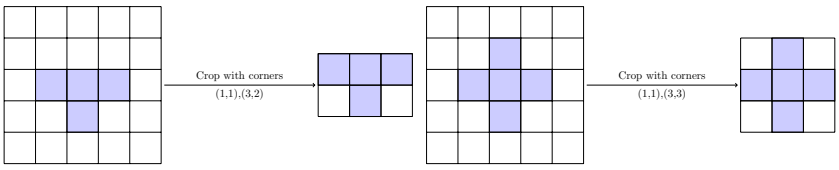


Figure 7. Example of ARC-like task with different output shapes.

B Description of basic mathematical morphological operators

Dilation. The dilation operator uses the Boolean function OR. For any s.e. B, the dilation operation $(\delta_B(\cdot))$ is shown as an example in Fig. 8. Here X is the input. Simply put, the dilation operator places the s.e at each 1 pixel and takes the union of all the resulting images (equivalently, the disjunction of the pixel values within the "window" defined by the s.e.)
Mathematically, we have

$$\begin{aligned}\delta_B(X) = X \oplus B &= \bigcup_{b \in B} X_b \\ &= \{x \mid B_x \cap X \neq \emptyset\}\end{aligned} \tag{6}$$

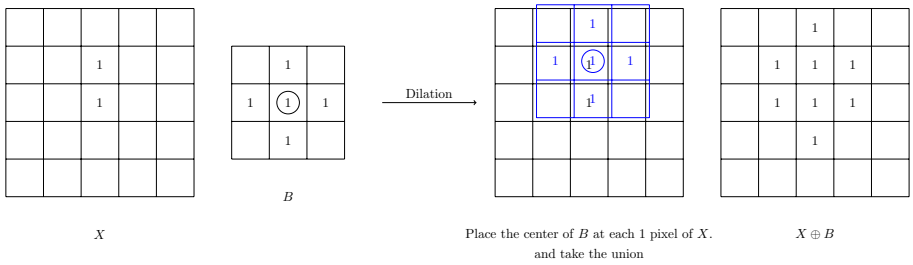


Figure 8. Illustrating the Dilation operator.

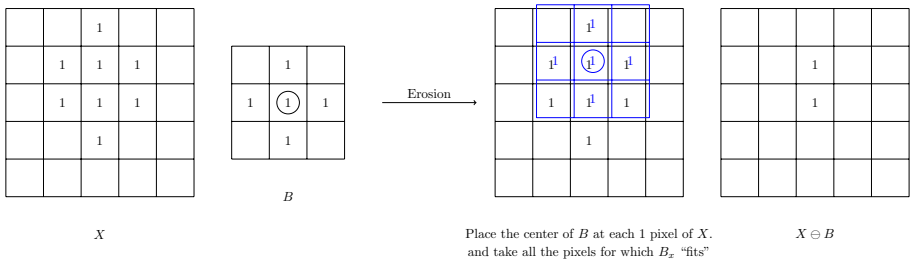


Figure 9. Illustrating the Erosion operator.

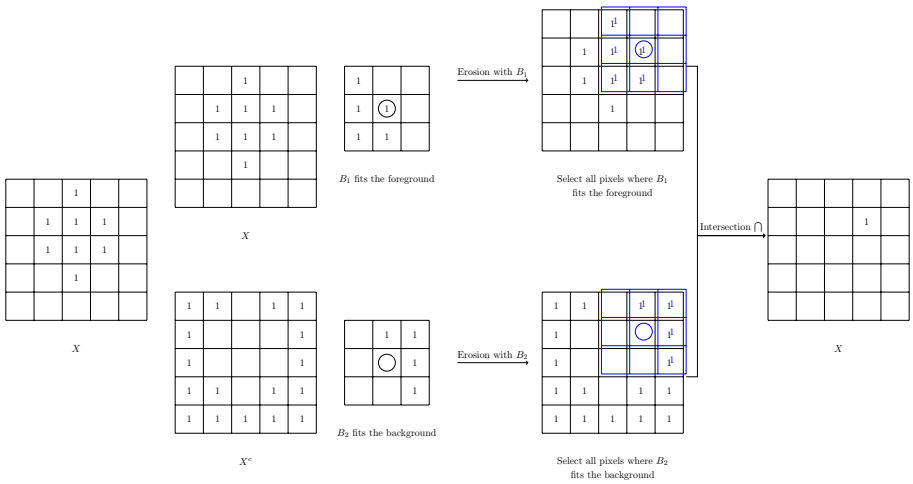


Figure 10. Illustrating the Hit-or-Miss operator.

where X_b denotes the set X translates by b, B_x denotes the set B translated by x.

Erosion. The erosion operator ($\epsilon_B(\cdot)$) uses the Boolean function `AND`. It can be thought of as an opposite of the dilation operator. Fig. 9 shows a simple illustration. As before, X is considered the input and B denotes the s.e. used to probe the input. That is, the erosion operator places the s.e at each pixel and takes those pixels for which the s.e "fits" within the foreground (pixels with value 1).
Mathematically, we have

$$\begin{aligned} \epsilon_B(X) = X \ominus B &= \bigcap_{b \in B} X_{-b} \\ &= \{x \mid B_x \subseteq X\} \end{aligned} \tag{7}$$

where X_b denotes the set X translates by b, B_x denotes the set B translated by x.

Hit-or-Miss. The erosion operator can also be extended to match patterns within the neighbourhood. Using two s.e's B_1 and B_2 (with shared origin) we identify all pixels such that $B_{1,x}$ matches the foreground (pixels with 1) and $B_{2,x}$ matches the background. This is referred to as the *Hit-or-Miss* transform. An illustrative example is in Fig. 10.