# Partial Functions and Preference for ARC: Domains, Interpolation, and Adaptation (Research Note)

Cristiano Calcagno

2025-10-31

**Abstract**

We formalize Abstraction and Reasoning Corpus (ARC) tasks using partial functions over grids and a numeric preference function on programs. The central concept is the *behavioural domain* that encodes the programmer's intended coverage. We give a short calculus for behavioural definedness under boolean connectives, characterize interpolation, generalisation, and adaptation to novelty, and work through a representative example where generalisation succeeds precisely by extending the behavioural domain to include a previously unhandled color.

## 1 Model

**Definition 1** (Objects). *Let $\mathcal{G}$ be the set of grids. Candidate programs are partial functions $f : \mathcal{G} \rightharpoonup \mathcal{G}$; write $\mathrm{Dom}_b(f) \subseteq \mathcal{G}$ for the (behavioural) domain of $f$, i.e. the set of inputs on which $f$* specifies *behaviour. Let $\mathcal{F}$ denote a chosen hypothesis class of such programs.*

**Definition 2** (Preference). *A numeric scoring function $P : \mathcal{F} \to \mathbb{R}$ induces a strict preference: $f_1$ is preferred to $f_2$ iff $P(f_1) > P(f_2)$.*

**Definition 3** (ARC puzzles). *An ARC puzzle (instance) consists of training pairs $\{(i_k, o_k)\}_{k=1}^n \subseteq \mathcal{G} \times \mathcal{G}$ and a test input $i \in \mathcal{G}$.*

**Definition 4** (Interpolants). *For training pairs $\{(i_k, o_k)\}_{k=1}^n$, a program $f$* interpolates *the training set if for all $k$, $i_k \in \mathrm{Dom}_b(f)$ and $f(i_k) = o_k$. Let $\mathcal{I} = \{f \in \mathcal{F} \mid f \text{ interpolates } \{(i_k, o_k)\}_{k=1}^n\}$ denote the set of all interpolants.*

**Definition 5** (Tentative solution, solution, ambiguity/void). *A* tentative solution *is any interpolant $f \in \mathcal{I}$ such that $i \in \mathrm{Dom}_b(f)$. Let*

$$\mathcal{S} \;=\; \big\{ f \in \mathcal{I} \,\big|\, i \in \mathrm{Dom}_b(f) \big\}.$$

*The* intended solution *is any $f^\star \in \arg\max_{f \in \mathcal{S}} P(f)$ (i.e., the element of $\mathcal{S}$ that maximizes $P$). The instance is* void *if $\mathcal{S} = \varnothing$ and* ambiguous *if $|\arg\max_{f \in \mathcal{S}} P(f)| > 1$.*

**Definition 6** (Generalisation and adaptation to novelty). Generalisation *is moving between interpolants $f, g \in \mathcal{I}$ with $P(g) > P(f)$. The top-scoring interpolant $f^\dagger \in \arg\max_{f \in \mathcal{I}} P(f)$ (the interpolant maximizing $P$) may fail to cover the test input ($i \notin \mathrm{Dom}_b(f^\dagger)$).* Adaptation to novelty *selects a tentative solution $g^\star \in \arg\max_{f \in \mathcal{S}} P(f)$; since $\mathcal{S} \subseteq \mathcal{I}$, necessarily $P(g^\star) \le P(f^\dagger)$.*

This formulation clarifies the tension in ARC: generalisation seeks higher-scoring interpolants (simpler, more elegant programs), while adaptation prioritizes coverage of the test input. When the test introduces novel features absent in training (e.g., a new color), the top interpolant $f^\dagger$ may exclude that case from $\mathrm{Dom}_b$, forcing a trade-off: accept a lower-preference tentative solution $g^\star$ that extends the behavioural domain, or stick with $f^\dagger$ and risk failure.

## 2 Behavioural domain

We focus on the **behavioural domain** $\mathrm{Dom}_b(e)$: the inputs on which the specification intends to define behaviour (coverage). We assume standard short-circuit evaluation for expressions; runtime definedness is incidental here and not our emphasis.

**Definedness calculus (examples).** For boolean expressions $E, F$ and comparisons $x{=}c$,

$$\mathrm{Dom}_b(\neg E) = \mathrm{Dom}_b(E), \tag{1}$$

$$\mathrm{Dom}_b((x{=}c \wedge E) \vee F) = (x{=}c \wedge \mathrm{Dom}_b(E)) \vee \mathrm{Dom}_b(F). \tag{2}$$

For a DSL primitive $g(\cdot)$, we assume a given predicate $\mathrm{Dom}_b(g)$.

## 3 Worked example

We illustrate the role of the behavioural domain using ARC-AGI-2 task dfadab01. The puzzle is available at `https://arcprize.org/play?task=dfadab01`; the full solver analysis at `https://gist.github.com/cristianoc/e86779cee94658567ecba429133d6667`.

Consider the following (corrected) DSL function:

```
def valid_anchor(grid: Grid, anchor: Anchor) -> bool:
    row, col, color = anchor
    return not (
        (color == 2 and guard_nw_eq(grid, (row, col), 4))
        or (color == 5 and guard_nw_eq(grid, (row, col), 6))
    )
```

**Example 1** (Behavioural domain). *If the intended specification handles only colors* 2 *(red) and* 5 *(gray), the behavioural domain must be:*

$$\begin{aligned}\mathrm{Dom}_b(\texttt{valid\_anchor}) = {} & (color{=}2 \wedge \mathrm{Dom}_b(\texttt{guard\_nw\_eq}(grid, (row, col), 4))) \\ & \vee (color{=}5 \wedge \mathrm{Dom}_b(\texttt{guard\_nw\_eq}(grid, (row, col), 6))).\end{aligned}$$

*This captures the explicit color-specific guards: only when color is* 2 *do we check the guard for color* 4*, and only when color is* 5 *do we check the guard for color* 6*. Any other color falls outside the intended behavioural domain.*

**Remark 1** (Novelty at test time). *If the test input has* `color`$=8$ *(sky blue), then the expression* computationally *evaluates (no guard is forced), but* $i \notin \mathrm{Dom}_b(\texttt{valid\_anchor})$ *by the behavioural domain example above. Generalisation must therefore extend the behavioural domain (e.g. add a new branch or a principled fallback) so that* 8 *becomes covered.*

## 4 Practical Implications

The model suggests a concrete operating procedure and design tips:

- **Selection rule.** Among interpolants, pick the tentative solutions that cover the test ($\mathcal{S}$) and choose the $P$-maximizer; if $\mathcal{S} = \varnothing$ the instance is *void*, and if multiple maximizers remain it is *ambiguous*.
- **Novelty handling.** If the top-scoring interpolant $f^\dagger$ does not cover the test ($i \notin \mathrm{Dom}_b(f^\dagger)$), extend the behavioural domain with the minimal, principled change that includes $i$, accepting a possible decrease in $P$.

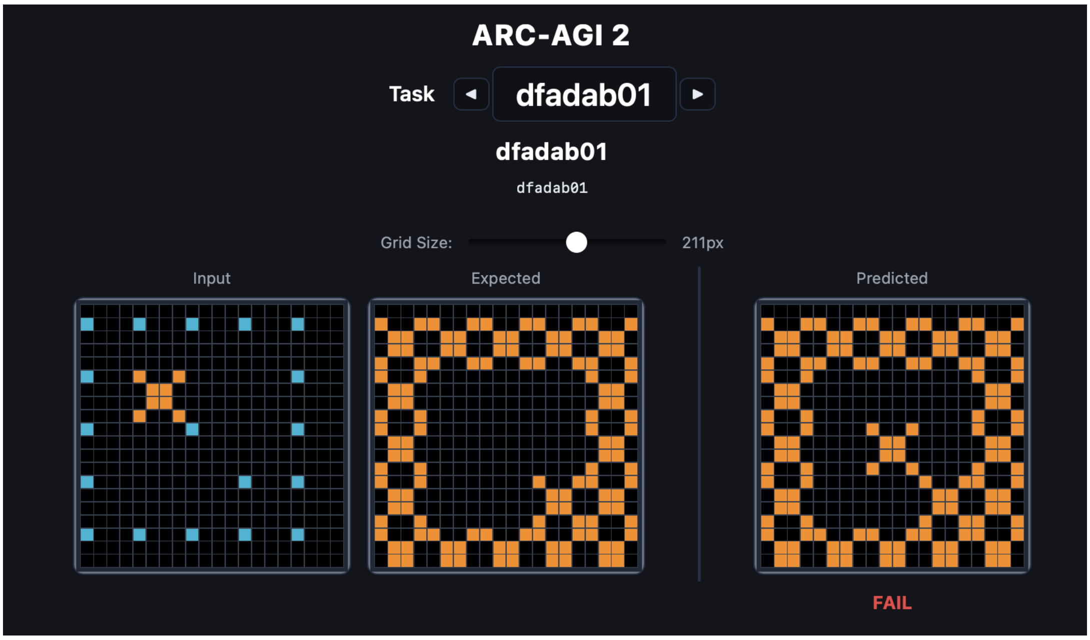


Figure 1: Before/after comparison showing domain extension failure and recovery: the initial solver explicitly handles only red (2) and gray (5). When sky blue (8) appears in the test input (corresponding to an `X`-pattern motif with orange (7) borders), the solver incorrectly stamps the motif because $8 \notin \mathrm{Dom}_b(\texttt{valid\_anchor})$. The simplified version introduces `guard_color` (deriving the guard color 7 from the motif's top-left cell) to uniformly treat all motifs, extending $\mathrm{Dom}_b$ to include color 8 and preventing spurious stamping.

- **Make coverage explicit.** Encode intended coverage via guard-style case-splits with no default `else`; this makes $\mathrm{Dom}_b$ visible and simplifies reasoning about what inputs are intentionally handled.
- **Debugging checklist.** On failures, first distinguish evaluation errors from coverage gaps: compute or approximate $\mathrm{Dom}_b(e)$ for the failing expression and check whether the test input lies outside it.

## 5 Summary

Modeling ARC programs as partial functions with a numeric score yields a compact account of interpolation, generalisation, and adaptation. The *behavioural domain* is essential: many failures are not evaluation errors but *coverage* gaps. In the example, the initial solver fails on the test because color 8 is outside $\mathrm{Dom}_b$; the generalised solution succeeds precisely by extending $\mathrm{Dom}_b$ to include that case.