

1 A Structural Theory of Cognitive
2 Representation and Problem Solving
3 Contexts, Invariance, and the Knowledge Space

4 Antal Jakovac^{1,2} and András Telcs^{1,3}

5 ¹Department of Computational Sciences, HUN-REN Wigner Research Centre for
6 Physics, H-1121 Budapest, Hungary

7 ²Department of Statistics, Institute of Data Analytics and Information Systems,
8 Corvinus University of Budapest, 8 Fővám Square, H-1093 Budapest, Hungary

9 October 6, 2026

10 **Abstract**

11 Learning and problem solving depend critically on the structure
12 of internal representations. While many modern data-driven artificial
13 systems achieve strong predictive performance, their learned represen-
14 tations often lack explicit structure for expressing abstraction, invari-
15 ance, and task-relevant regularities. We propose a minimal structural
16 framework in which representational operations relevant to problem
17 solving, such as context formation, invariance recognition, represen-
18 tative selection, abstraction, and procedural reuse, are made explicit.
19 The central notion is that of a *context*, formalized as a partition of a
20 subset of an underlying state space, which fixes the distinctions, granu-
21 larity, and form in which a problem can be posed. Within this setting,
22 invariance recognition and representative selection are treated as fun-
23 damental representational operations. The framework is realized as a
24 Knowledge Space composed of two coupled graph structures: a Con-
25 cept Graph that hosts constructed and refined concepts, and a Proce-
26 dure Graph that encodes typed operations over representations. To-
27 gether, these structures provide a minimal cognitive-representational
28 algebra for operating on representations without assuming sophisti-
29 cated inference, learning, control, perception, or motor mechanisms.

Using simple illustrative examples and a finite weak-solver demonstration, we show that appropriate representational organization can simplify the form and scope of admissible regularities, even when problem solving is carried out by a fixed and limited solver. The contribution of the paper is structural rather than algorithmic: it identifies representational prerequisites for abstraction, invariance, and procedural reuse in problem solving, and states explicit success and failure conditions for the weak-solver setting.

1 Introduction

Learning and problem solving depend critically on how information is represented. While many modern data-driven artificial systems achieve strong predictive performance, their learned representations often leave implicit the structures needed for abstraction, invariance, compositionality, and task-relevant regularities [1, 2, 3, 4]. As a consequence, basic cognitive operations such as equivalence, invariance, abstraction, and relational reasoning are not explicitly available at the representational level.

In the earlier work [5], we introduced a representation framework based on explicit coordinate systems, distinguishing law-relevant, descriptive, and task-specific components. That framework provides a flexible interface between observations and decision procedures. However, it leaves open a foundational question: what structural organization must representations have in order to support basic cognitive primitives independently of specific learning algorithms?

The present paper addresses this question by introducing a minimal structural framework centered on the notion of *context*. Contexts are formalized as finite partitions of subsets of the state space. They specify which distinctions are meaningful, at what granularity a task is posed, and in which representational space admissible solutions can be expressed. In this sense, contexts define the structural conditions under which problem solving is possible.

We treat invariance and representative selection as representational operations acting on contexts. They are not introduced as learning objectives, nor as statistical effects of data or loss functions. Instead, they are structural mechanisms that suppress irrelevant distinctions and expose simpler task-relevant laws. We add further cognitive primitives to the framework to obtain a basic cognitive algebra.

65 To make these ideas explicit, we introduce the *Knowledge Space* as a
66 minimal representational scaffold composed of two coupled graph structures:
67 a Concept Graph and a Procedure Graph (2G). The Concept Graph serves as
68 a host structure for concepts that are introduced, refined, and reused during
69 problem solving, rather than as a fixed ontology. Procedures are assumed
70 to be weak and fixed; representational structure is the primary source of
71 problem-solving power.

72 The contribution of this work is structural rather than algorithmic. We
73 formalize contexts as the foundational unit of distinction, specify how con-
74 texts and invariance organize a knowledge space, and demonstrate how this
75 structure supports simple forms of problem solving with weak solvers. The
76 paper does not propose a full learning architecture, a control mechanism, or
77 a theory of causality. Instead, it gives a self-contained representational core:
78 context formation, invariance recognition, representative selection, procedure
79 storage, and a finite weak-solver demonstration with explicit success and fail-
80 ure conditions. Thus the paper does not ask the reader to accept the general
81 claim that representation helps; it specifies conditions under which represen-
82 tational structure reduces the burden placed on a fixed weak solver.

83 2 Related Work and Positioning

84 This work is related to several lines of research in cognitive science, theoretical
85 computer science, and artificial intelligence. We briefly position it relative to
86 these directions, emphasizing points of contact and distinction.

87 The role of representation in problem solving has long been emphasized
88 in cognitive science. Classical work by Newell and Simon highlights how
89 representational choice determines the tractability of problem solving, inde-
90 pendent of search strategy [6]. Marr’s levels of analysis further distinguish
91 representational questions from algorithmic and implementational concerns
92 [7]. The present work operates explicitly at the representational level. Its
93 aim is to identify the structural organization of representations that is needed
94 before particular learning algorithms or control architectures are specified.

2.1 Cognitive architectures, schemas, and procedural memory

The present framework is related to, but different from, cognitive architectures such as ACT-R, Soar, CLARION, LIDA, and the Common Model of Cognition [8, 9, 10, 11, 12, 13, 14, 15]. These architectures aim to specify broad computational organizations of cognition, usually including working memory, long-term declarative memory, procedural memory, control mechanisms, learning mechanisms, and, in some cases, perceptual-motor interfaces. ACT-R and Soar, for example, have been compared in terms of working memory, procedural memory, and long-term declarative memory [14]. Soar provides an explicit decision cycle, impasses, subgoals, semantic and episodic memory, and chunking [11, 15]. ACT-R organizes cognition around declarative chunks, procedural rules, buffers, and production firing [9, 10, 16]. CLARION emphasizes the interaction between implicit and explicit representations and processes [17, 12]. The Common Model of Cognition has also been proposed as a shared architectural framework across artificial intelligence, cognitive science, neuroscience, and robotics [13].

Our proposal is not a cognitive architecture in this sense. It does not define a complete control cycle, a perception-action loop, motivational or metacognitive subsystems, timing assumptions, or quantitative behavioral predictions. Its narrower aim is to formalize a representational substrate on which tasks, contexts, concepts, and procedures can be stated. The Knowledge Space should therefore be read as a structural layer that may support architectural mechanisms, not as a replacement for ACT-R, Soar, CLARION, LIDA, or the Common Model. In this respect the closest point of contact is not the full architecture of these systems, but their distinction between transient task states, long-term conceptual or declarative structures, and reusable procedural structures.

The framework is also related to classical schema theory. Bartlett introduced schemas as organized structures that shape remembering and interpretation [18]. Minsky’s frames represented knowledge through structured descriptions with slots, defaults, and procedural attachments [19]. Schank and Abelson’s scripts represented stereotyped event structures and action sequences [20]. In the present framework, contexts play a related but more formal role: a context fixes the distinctions and granularity within which a task is posed. A schema is then not taken as a primitive mental object, but as a stable pattern of representational and procedural organization that can

132 be reconstructed from contexts, features, relations, procedures, and stored
133 solution traces.

134 This distinction is important for the term “cognitive primitive.” We do
135 not intend graph traversal, set union, or Boolean operations, by themselves,
136 to carry cognitive content. They are implementation-level supports. The cog-
137 nitive content lies in their representational role: reading a segmented task
138 code, forming distinctions, comparing feature coordinates, projecting away
139 irrelevant variation, recognizing equality or inequality, retrieving a stored pro-
140 cedure, composing procedures, and consolidating repeated successful traces
141 into stable concepts or procedures. Thus the primitives are cognitive only
142 when they operate on the representational structures supplied by contexts
143 and the Knowledge Space.

144 The Procedure Graph is introduced for the same reason. It is not meant
145 to be a production system with a complete interpreter, conflict-resolution
146 mechanism, or control cycle. Production systems represent action or infer-
147 ence rules in an if-then format and are used centrally in architectures such
148 as ACT-R and Soar. By contrast, a node of the Procedure Graph represents
149 a typed operation over representations: for example, a projection, a com-
150 parison, a conjunction of tests, a representative-selection step, or a stored
151 solution procedure. During problem solving, successful solution traces create
152 new procedure nodes. Direct reuse corresponds to applying a stored node.
153 Bounded composition creates a new procedure node linked to the procedures
154 from which it was composed. Repeatedly successful procedures may induce
155 equivalence edges, while procedures that discard distinctions and still pre-
156 serve success may induce generalization edges.

157 In this way the Procedure Graph is not decorative. It records which
158 operations are available, how they are typed, when they are reusable, and how
159 new procedures arise from successful traces. The constructive weak-solver
160 demonstration uses this structure explicitly: the solver begins with primitive
161 procedures, solves tasks when possible, stores successful procedures, reuses
162 them on later tasks, and forms new procedure nodes by bounded composition
163 or bounded generation. The Concept Graph and Procedure Graph together
164 therefore play the role of a minimal long-term representational store, while
165 the current task code and its active trace play the role of a transient working
166 structure. This analogy to working and long-term memory is structural only;
167 no claim is made that the present model is a psychologically complete memory
168 architecture.

169 Formal notions of abstraction and simplification have also been studied in

170 logic and symbolic AI. Research on rule systems, theory revision, and sym-
 171 bolic reasoning typically assumes that laws are already available in symbolic
 172 form and focuses on their manipulation [21]. In contrast, our work addresses
 173 an earlier stage by defining basic cognitive-representational primitives and
 174 the structural conditions under which they form a cognitive algebra.

175 **Category-theoretic and invariance-based approaches.** Category-theoretic
 176 approaches to cognition provide another structural point of comparison. In
 177 particular, Phillips and Wilson used category-theoretic notions to address
 178 systematicity, compositionality, and the preservation of relational structure
 179 in human cognition [22, 23]. The present paper is adjacent to this line of work
 180 only at the level of motivation: it also treats cognitive representation as a
 181 structured object rather than as an unorganized set of features. However, we
 182 do not use categorical machinery, functors, universal constructions, or cate-
 183 gorical semantics. Our formal level is deliberately more elementary: contexts
 184 are finite partitions, and the relevant operations are refinement, projection,
 185 representative selection, and typed procedures over representational objects.

186 The role of invariance in representation has also been studied exten-
 187 sively in representation learning and structured neural architectures. Work
 188 on equivariance and equivalence of image representations, group-equivariant
 189 convolutional networks, and geometric deep learning shows how known sym-
 190 metries and structural regularities can be incorporated into or measured in
 191 learned representations [24, 25, 26]. Our use of invariance is different. We
 192 do not build an architecture that is equivariant to a prescribed transforma-
 193 tion group. Instead, invariance is introduced as a task-relative equivalence
 194 relation over a context, and representative selection is treated as a represen-
 195 tational operation that may simplify the form of the task for a weak solver.

196 Modern representation learning emphasizes invariance and disentangle-
 197 ment as prerequisites for generalization [27]. Architectural approaches such
 198 as equivariant neural networks encode known symmetries directly into mod-
 199 els. By contrast, we treat invariance as a cognitive and structural primitive,
 200 expressed as equivalence relations over contexts rather than as architectural
 201 constraints.

202 Work on invariant risk minimization and out-of-distribution generaliza-
 203 tion studies predictors that remain stable across multiple environments [28].
 204 These approaches operate at the level of predictors and losses. Our focus
 205 is instead on the representational substrate: we analyze how contexts and

206 invariances define the space in which laws can be stated, prior to any notion
207 of risk or optimization.

208 Causal modeling further illustrates the importance of representational
209 structure. Structural approaches to causality distinguish between correlation
210 and causal relations by imposing constraints on admissible models [29]. In
211 the present paper, causality is not modeled explicitly. We only introduce
212 minimal structural primitives, such as successor and conditional dependence,
213 that are necessary prerequisites for later causal reasoning.

214 Within the Cognitive Systems Research community, conceptual and struc-
215 tural approaches to representation have emphasized geometry, similarity, and
216 conceptual spaces [30]. Our framework shares the emphasis on structure but
217 replaces metric similarity with context partitions, evolving concept graphs
218 structure and refinement, allowing abstraction to be defined independently
219 of geometry.

220 **Rough sets and context partitions.** Because the present framework for-
221 malizes a context as a finite partition of a subset of the state space, it is im-
222 portant to position it explicitly relative to rough set theory. Pawlak’s rough
223 sets start from indiscernibility relations and the partitions that they induce,
224 and the rough-set literature develops approximation, reducts, and discernibil-
225 ity tools for reasoning over such partitions [31, 32, 33]. Our use of contexts is
226 not intended to claim priority over this partition-based formal core. Rather,
227 the distinction is that contexts are used here as task-relative representational
228 units inside a cognitive knowledge space, coupled to concept and procedure
229 graphs and to operations such as invariance recognition and representative
230 selection. Skowron and Rauszer introduced the discernibility matrix and the
231 discernibility function as central tools for the analysis of Pawlak information
232 systems [33]. In that setting, objects are described by attributes, and the dis-
233 cernibility matrix records, for pairs of objects, which attributes distinguish
234 them. Reducts are then obtained as minimal attribute subsets preserving
235 the relevant discernibility, and the same formalism supports the computa-
236 tion of cores, dependencies, rough definability, and decision rules. This gives
237 a precise algebraic and algorithmic treatment of information reduction in
238 an already specified information system. The present framework is related
239 to this line of work at the level of partitions and equivalence classes, but
240 the direction is different. We do not start from an information table whose
241 attributes have to be reduced while preserving discernibility. Rather, parti-

242 tions, representatives, and relations are introduced as task-relative cognitive
 243 units. A representative is not selected in order to preserve all discernibility
 244 encoded by an information system, but in order to support the formulation
 245 of a task, the creation of a concept, or the execution of a solution procedure.
 246 Thus the present construction should be seen as complementary to reduct
 247 theory: reducts optimize a given formal information description, whereas
 248 our representatives participate in the construction and testing of cognitive
 249 representations through use.

250 Formal Concept Analysis provides a central reference point for any formal
 251 treatment of concepts and conceptual hierarchy. In FCA, a formal context
 252 is a triple (G, M, I) consisting of objects, attributes, and an incidence rela-
 253 tion. Formal concepts are extent–intent pairs obtained by closure from this
 254 incidence relation, and the set of formal concepts forms a complete concept
 255 lattice ordered by generality [34, 35]. Thus FCA develops a mathematically
 256 controlled concept system: concepts and their relations are correct by con-
 257 struction relative to the given formal context. The present framework uses
 258 the word context in a different sense, namely as a task-relative representa-
 259 tional context. We do not assume an object–attribute incidence relation, do
 260 not construct extent–intent pairs, and do not require the resulting network
 261 of representational units to be a concept lattice. Concepts, partitions, and
 262 relations may be introduced provisionally in the course of representing and
 263 transforming a task. Their soundness is not guaranteed by a closure oper-
 264 ation at the moment of construction, but is tested through repeated use in
 265 task formulation, task transformation, and task solution. To avoid termino-
 266 logical confusion, we shall use the expression task context or cognitive task
 267 context when referring to our notion of context.

268 The terminology of concepts, contexts, and concept graphs has several
 269 important predecessors. Sowa’s conceptual graphs provide a graph-based
 270 formalism for representing conceptual structures and reasoning [36]. For-
 271 mal Concept Analysis constructs formal concepts from an object–attribute
 272 incidence context and orders them in a complete concept lattice [34, 35].
 273 Pattern structures extend FCA to more complex object descriptions, such
 274 as graphs or other structured data, by replacing simple attribute sets with
 275 descriptions equipped with a semilattice operation [37]. Granular comput-
 276 ing, developed in particular by Yao and others, studies information granules
 277 as a general method of abstraction, approximation, and problem solving,
 278 including rough-set and fuzzy-set based models [38]. The present paper is
 279 adjacent to these traditions, but it does not propose another formal concept

lattice, another reduct algorithm, or another general theory of information granules. Its aim is more limited and more cognitive: to describe how a task is represented inside a finite cognitive knowledge space by means of partitions, representatives, laws, and admissible transformations. In this setting, representational units may be provisional. They are not necessarily correct by construction, as in FCA closure, and they are not selected to preserve discernibility in a given information system, as in reduct theory. Their adequacy is tested through repeated use in task formulation, task transformation, and task solution.

In summary, the present work differs from prior approaches by treating context as the foundational unit of task formulation and representation. Rather than proposing new learning algorithms or inference mechanisms, we provide a minimal structural framework. This positioning is intended to support later work on task specification, cognitive schema and learning, evaluation, causality, and control, while remaining self-contained at the representational level.

Scope and falsifiability. The present paper is intentionally restricted to the representational level, but the restriction is not merely promissory. The claims made here concern when a task is well posed relative to a context, when a context transformation preserves the task-relevant relation, and when a fixed weak solver can solve a task after representational simplification. These claims can fail: they fail if the proposed invariance is not task-preserving, if representative selection changes the relevant relation, if projection does not preserve the target structure, or if the reduced representation still lies outside the admissible class of the weak solver. Appendix A gives a finite weak-solver demonstration with explicit success and failure conditions.

3 Foundations

This section introduces the basic definitions that serve as the foundational layer for the development of the cognitive algebra. We present a representational scheme on which elementary constructs such as sets, axioms, and primitive manipulations can be defined. The goal is not to impose a specific domain interpretation, but to establish a minimal and self-consistent structural basis for subsequent developments.

3.1 Universe, States and Time

We adopt a deliberately physical viewpoint, centered on the states of the world and their evolution in time. As will become clear later, this choice is not restrictive: the resulting formalism is sufficiently general to support a wide range of cognitive and representational interpretations.

The ultimate arena in which all actions take place is the Universe. Time serves as a label for different spatial instances, parameterized by a real number. It is assumed to possess an additive structure, meaning that for any three spatial instances a , b and c , the corresponding time differences satisfy

$$t_{ab} + t_{bc} = t_{ac}.$$

At each spatial instance, the Universe occupies one of its possible states. The set of all such states is denoted by Ω . A state is assumed to be complete, in the sense that it uniquely determines all future states of the Universe. Under this assumption, the time evolution of the Universe can be described as a deterministic, memoryless process.

We emphasize that the Universe, its states, and the precise form of its time evolution are theoretical constructs. Their role is to support the assumption that the evolution of the Universe is deterministic, given a single actual state, even though such determinism may not be directly observable or computationally accessible.

3.2 Object classes

We now shift to a description from the point of view of an agent – an intelligent entity that may be a living being, in particular a human, or an artificial system. The agent observes the Universe, understood as the ultimate underlying reality. However, the vast majority of the Universe is irrelevant for the agent’s description of the world. This is partly because the agent cannot observe most of it, partly because it lacks the computational capacity to account for all microscopic details, and most importantly because many details play no role in the fulfillment of the agent’s purpose. In biological agents this purpose is typically the maintenance of the agent within its environment, whereas in artificial agents it may be defined by externally imposed goals.

This line of reasoning implies that the agent’s relevant concepts do not correspond to individual states of the Universe, but rather to collections of

states. For example, the concept of a book refers not to a single microscopic configuration of the Universe, but to a large set of states in which the book exists, irrespective of irrelevant details such as the precise position of Mars.

Consequently, for an agent, the fundamental constituents of the world are not the elementary states themselves, but subsets of the state space Ω . These subsets are typically very large (cf. cylinder sets in probability theory). We refer to such subsets as object classes, or simply classes.

(Object) class: subsets of the Universe, collecting states that differ only in irrelevant details.

For example:

- a *pen* means the same object, irrespective to the exact shape, matter content, position in the space and time
- living being is a collection of states where the singled out object is a living being, while all other details are irrelevant
- in an image the concept that a given pixel is red means that all other pixels have arbitrary color

3.3 Context

The object classes are the simplest elements of the description of the world. Often we want to speak about finer distinctions relative to a given concept (e.g. dog breeds within the class of dogs). A *context* formalizes such a distinction as a partition of the underlying class.

Context (partition). Let $C \subseteq \Omega$ be an object class. A **context** on C is a finite partition $\mathcal{C}$ of C , i.e. a finite family of nonempty subsets of C such that

$$\bigcup_{c \in \mathcal{C}} c = C, \quad c \cap c' = \emptyset \text{ for } c \neq c', \quad c \neq \emptyset \text{ for all } c \in \mathcal{C}.$$

The elements $c \in \mathcal{C}$ are called the **cells** (or **context cells**) of $\mathcal{C}$. The **underling class** of the context $\mathcal{C}$ is $\text{base}(\mathcal{C}) := C$.

Note that an object class can be represented as a context with a single element set.

370 3.3.1 Contexts instead of states

371 States are a semantic abstraction. In practice, an agent accesses the world
 372 exclusively through *contexts*, whose cells are object classes. There is no
 373 ultimate description of the world; contexts may be refined as needed. A
 374 context determines what can be distinguished and thus sets the framework
 375 of a task to be performed or a problem to be solved.

376 From this point on, we treat contexts (and their cells) as the primary
 377 objects, using states only as semantic scaffolding when needed.

378 3.3.2 Refinement and induced maps

379 **Refinement.** Let $\mathcal{C}$ and $\mathcal{C}'$ be contexts on the same underlying class C .
 380 We say that $\mathcal{C}'$ is a **refinement** of $\mathcal{C}$, written

$$\mathcal{C}' \preceq \mathcal{C},$$

381 if every cell of $\mathcal{C}'$ is contained in a (unique) cell of $\mathcal{C}$:

$$\forall c' \in \mathcal{C}' \exists! c \in \mathcal{C} \text{ such that } c' \subseteq c.$$

382 Equivalently, $\mathcal{C}$ is a coarsening of $\mathcal{C}'$.

383 Example: dog breeds is a refinement of the dogs.

384 **Projection between contexts.** If $\mathcal{C}' \preceq \mathcal{C}$, define the projection Π

$$\Pi_{\mathcal{C} \leftarrow \mathcal{C}'} : \mathcal{C}' \rightarrow \mathcal{C}, \quad \Pi_{\mathcal{C} \leftarrow \mathcal{C}'}(c') := c \text{ where } c' \subseteq c.$$

385 This map expresses how a finer description (cells of $\mathcal{C}'$) collapses to a coarser
 386 description (cells of $\mathcal{C}$).

387 **Local refinement.** A context $\mathcal{C}'$ is a **local refinement** of $\mathcal{C}$ if there exists
 388 a cell $c \in \mathcal{C}$ such that $\text{base}(\mathcal{C}') = c$ (i.e. $\mathcal{C}'$ is a partition of one cell of $\mathcal{C}$).

389 **Global refinement update.** If $\mathcal{C}'$ is a local refinement of $\mathcal{C}$ refining a cell
 390 $c \in \mathcal{C}$, the updated (globally refined) context is

$$\mathcal{C}^+ := (\mathcal{C} \setminus \{c\}) \cup \mathcal{C}'.$$

391 Then $\mathcal{C}^+$ is a context on $\text{base}(\mathcal{C})$ and satisfies $\mathcal{C}^+ \preceq \mathcal{C}$.

392 **Common refinement.** Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be contexts on the same underlying
 393 class C . Their **common refinement** is the context

$$\mathcal{C}_1 \wedge \mathcal{C}_2 := \{c_1 \cap c_2 \mid c_1 \in \mathcal{C}_1, c_2 \in \mathcal{C}_2, c_1 \cap c_2 \neq \emptyset\}.$$

394 It satisfies $\mathcal{C}_1 \wedge \mathcal{C}_2 \preceq \mathcal{C}_1$ and $\mathcal{C}_1 \wedge \mathcal{C}_2 \preceq \mathcal{C}_2$. Moreover, for any context $\mathcal{D}$ on C
 395 such that $\mathcal{D} \preceq \mathcal{C}_1$ and $\mathcal{D} \preceq \mathcal{C}_2$, we have $\mathcal{D} \preceq \mathcal{C}_1 \wedge \mathcal{C}_2$.

396 3.4 Characterization of contexts

397 We briefly describe three basic ways of representing a context. They are not
 398 mutually exclusive. They serve as reference points for later constructions.

399 3.4.1 Direct representation

400 The simplest representation treats each cell of the context as an atomic ob-
 401 ject. This is formally complete, but infeasible in high dimensions. For exam-
 402 ple, a black-and-white image with one million pixels induces $2^{1,000,000}$ possible
 403 configurations, hence a direct indexing of all cells is not practical.

404 Typical instances where this representation is applied, include:

- 405 • image classification, where a context cell corresponds to a class label;
- 406 • one-hot encoding of tokens, where each token corresponds to a distinct
 407 cell.

408 3.4.2 Global coordinates

409 A fine context can often be represented through several coarser contexts
 410 whose common refinement reproduces it.

411 **Coordination.** A family of contexts $\mathcal{C}_1, \dots, \mathcal{C}_n$ is a *coordination* of a con-
 412 text $\mathcal{C}$ if

$$\mathcal{C}_1 \wedge \dots \wedge \mathcal{C}_n = \mathcal{C}.$$

413 This approach is common in mathematics and in coordinate-based repre-
 414 sentations. For example:

- 415 • A d -dimensional real vector space can be coordinated by families of
 416 parallel hyperplanes. Their common intersections identify points.

- Embedding spaces in language models may be seen as global-coordinate representations, where different directions correspond to different contextual distinctions.

3.4.3 Local coordinates

A context can also be built by successive local refinements. This yields a refinement tree and, consequently, a prefix description of cells.

Refinement tree. A *refinement tree* for a context $\mathcal{C}$ is a rooted tree whose nodes are labeled by subsets of $\mathcal{C}$ such that:

- the root is labeled by $\mathcal{C}$;
- the children of any node form a partition of the parent label;
- the leaves are singletons $\{c\}$ with $c \in \mathcal{C}$.

The path from the root to a leaf defines a prefix code for the corresponding cell.

The refinement tree is not unique, even for a fixed context. If leaf probabilities are known, one may choose a tree that minimizes expected code length (e.g. Huffman coding).

Examples include:

- taxonomies (e.g. biological classification);
- attribute-based product categories (e.g. display size, fat content);
- the “twenty questions” game.

To represent local coordinates one typically needs not only a value but also the tested property. This motivates triplet-style descriptions (name:property:value).

3.5 Features

A *feature* is a function defined on the cells of a context.

441 **Feature.** Let $\mathcal{C}$ be a context on $C \subseteq \Omega$. A *feature* is a map that is constant
 442 on the cells of $\mathcal{C}$

$$f : C \rightarrow B, \quad f(x) = f(y) \quad \forall x, y, c : x, y \in c \in \mathcal{C}$$

443 where B is a value set (e.g. numbers or symbols).

444 A feature can be considered as a function whose domain is the context
 445 $f : \mathcal{C} \rightarrow B$, if it does not cause misunderstanding.

446 In measure and probability theory, a feature is called a measurable func-
 447 tion with respect to a measurable space $(C, \sigma(\mathcal{C}))$, where $\sigma(\mathcal{C})$ is the canonical
 448 sigma-algebra σ -algebra generated by the context $\mathcal{C}$.

449 **Induced context.** A feature induces a context (partition) of C by grouping
 450 states according to the feature value:

$$f \mapsto \{ f^{-1}(y) \mid y \in \text{Range}(f), f^{-1}(y) \neq \emptyset \}.$$

451 **Extension across refinements.** If $\mathcal{C}' \preceq \mathcal{C}$, then a feature $f : \mathcal{C} \rightarrow B$
 452 extends to $\mathcal{C}'$ by constancy on sub-cells: if $c' \subseteq c$ then $f(c') := f(c)$.

453 **Relation to rough-set consistency.** A feature is called *relevant* for a
 454 context if it distinguishes cells in a way compatible with the task being stud-
 455 ied. Otherwise it is *irrelevant*. The earlier notion of a relevant feature can
 456 also be expressed in the language of rough set theory. If features are treated
 457 as attributes of an information system and the task output is treated as a
 458 target or decision partition, then relevance is closely related to the consis-
 459 tency of an attribute-induced partition with the target partition, and hence
 460 to the background of reduct theory. The present paper does not claim this
 461 consistency condition as a new formal notion. Its aim is to place such condi-
 462 tions inside a cognitive task framework in which representational units may
 463 be features, partitions, representatives, laws, or admissible transformations.

464 4 The Knowledge Space as Two Coupled Graphs 465 (2G)

466 We model the Knowledge Space (KS) as two coupled but distinct graph-
 467 structured layers, collectively denoted **2G**: (i) a *Concept Graph* encoding

conceptual entities and their relations, (ii) a *Procedure Graph* encoding operations as typed functions linked to their conceptual domains and ranges. The purpose of this section is to specify a minimal structural substrate on which basic cognitive primitives can be defined.

4.1 Concept Graph (CG)

Definition. A *Concept Graph* is a directed, typed graph

$$CG = (V_C, E_C, \tau_C),$$

where V_C is a set of concept nodes, $E_C \subseteq V_C \times \Sigma_C \times V_C$ is a set of labeled edges, and $\tau_C : V_C \rightarrow \mathcal{T}_C$ assigns each node a type from a finite set $\mathcal{T}_C$.

Concept nodes. Nodes $v \in V_C$ represent concepts, including:

- objects and attributes,
- sets and grouping constructs,
- relations (binary and higher-arity predicates),
- derived concepts (including invariances and representative choices).

Contexts and concept nodes. Contexts are not treated as graph objects. They provide the semantic scaffolding for introducing and refining concepts. Concept nodes are named distinctions that may arise at different levels of granularity. In particular, a concept node may represent: (i) an underlying class $C \subseteq \Omega$, (ii) a cell of a context defined on C , or (iii) a derived construct obtained from contexts by operations such as invariance recognition and representative selection. In this way, contexts explain how distinctions are formed, while the Concept Graph records which distinctions are available for reuse.

Edge labels. Edge labels encode basic structural relations between concepts. In the permanent Concept Graph, we restrict labels to a minimal vocabulary: *equivalence*, represented by unoriented edges, and *generalization*, represented by oriented edges from specific to more general concepts. During problem solving, richer semantic relations and alternative labelings may

be introduced on task-specific copies of Concept Graph fragments. When a problem is successfully solved, the resulting task-solution structure is stored in a separate layer of permanent memory, together with explicit mappings to and from the permanent Concept Graph. These mappings link solution-specific constructs to stable concepts, and may give rise to the introduction of new permanent concept nodes. The mechanisms governing consolidation across memory layers are beyond the scope of the present paper.

4.2 Procedure Graph (PG)

Definition. A *Procedure Graph* is a directed, typed graph

$$\text{PG} = (V_F, E_F, \text{dom}, \text{rng}, \tau_F),$$

where V_F is a set of procedure nodes, $E_F \subseteq V_F \times \Sigma_F \times V_F$ is a set of labeled edges, dom and rng (domain and range) map each procedure to its input and output types, and τ_F assigns each procedure node a procedure type.

Procedures as typed functions. Each node $p \in V_F$ represents a (possibly partial) function

$$p : \text{dom}(p) \rightarrow \text{rng}(p),$$

where input and output types refer to concept-structured objects (e.g. sets, relations, graphs, coordinate tuples).

Edge labels in the Procedure Graph. Edges in the Procedure Graph encode structural relations between procedures. We distinguish two fundamental types of edges.

Equivalence. An unoriented edge between two procedures represents equivalence. Two procedures $p : D_p \rightarrow R_p$ and $q : D_q \rightarrow R_q$ are equivalent if there exist bijections $a : D_p \rightarrow D_q$ and $b : R_p \rightarrow R_q$ such that the diagram commutes 1:

$$q \circ a = b \circ p.$$

Equivalently, the two procedures differ only by a relabeling of inputs and outputs and induce isomorphic partitions of their domains. Equivalence edges identify procedures that implement the same structural transformation up to representation.

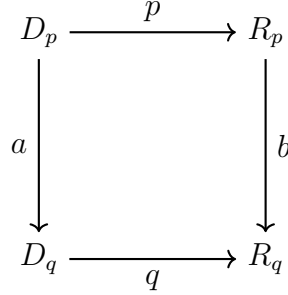


Figure 1: Equivalence of procedures. Procedures $p : D_p \rightarrow R_p$ and $q : D_q \rightarrow R_q$ are equivalent if there exist bijections a and b such that $q \circ a = b \circ p$.

522 *Generalization.* An oriented edge $p \rightarrow q$ represents procedural generaliza-
 523 tion. It indicates that q is a more general procedure than p , in the sense that
 524 q collapses distinctions preserved by p and induces a coarser partition of the
 525 domain. Thus, procedural generalization mirrors abstraction in the Concept
 526 Graph and reflects a loss of representational detail rather than execution
 527 order or control flow.

528 These two edge types constitute the complete labeling vocabulary of the
 529 Procedure Graph in the present paper. Procedural composition, when well-
 530 typed, is defined independently of these edges and follows from the underlying
 531 algebra.

532 **Examples.** The Procedure Graph contains nodes representing primitive
 533 procedures operating on structured representations. Typical examples in-
 534 clude:

- 535 • *Representative selection* operators, which choose a canonical represen-
 536 tative within an equivalence class induced by invariance;
- 537 • *Abstraction* operators, such as projection or quotienting, which elimi-
 538 nate nuisance distinctions and induce coarser partitions;
- 539 • *Derived-relation* constructors, which form new relations from existing
 540 ones within a fixed context;
- 541 • *Simple inference or transformation* operators used by a solver to ma-
 542 nipulate structured representations.

543 **Remarks.** In the present paper, the Procedure Graph is not used to model
 544 planning, control, or execution order. Its role is to organize primitive pro-
 545 cedures by equivalence and generalization, providing a structural account
 546 of how invariance and representational simplification are expressed at the
 547 procedural level.

548 The purpose of 2G in this paper is to define a minimal structural substrate
 549 for representations, not to propose a complete cognitive architecture. Explicit
 550 goal representations, executive control, and long-horizon planning are not
 551 introduced. Instead, we use 2G to localize invariance and representative
 552 selection as representation-level operations on structured knowledge, and we
 553 evaluate their effect using simple solvers.

554 During problem solving, task-specific copies of graph fragments are in-
 555 stantiated in transient memory; successful solutions are transferred to per-
 556 manent memory together with mappings to the underlying conceptual struc-
 557 tures. The internal organization of memory is not modeled in detail here.

558 5 A Primitive Cognitive Algebra

559 We introduce a minimal formal substrate together with genuinely cognitive
 560 principles. It is useful to separate (i) background formal assumptions required
 561 for well-formed reasoning from (ii) cognitive principles governing structural
 562 simplification.

563 5.1 Formal assumptions

564 **Assumption A1 (Typing and compatibility).** We assume a set of ab-
 565 stract type labels $\mathcal{T}$. Types are assigned only to *basic entities*, namely con-
 566 cept nodes and procedure interfaces (domains and ranges). Typing serves
 567 solely to express compatibility of entities under operations; types do not
 568 denote sets, collections, or semantic domains.

569 **Assumption A2 (Structural set formation).** For each type label $\tau \in$
 570 $\mathcal{T}$, there exists a collection type $\mathbf{Set}_\tau$. An expression $x : \tau$ means that x is
 571 compatible with type τ . Elements of $\mathbf{Set}_\tau$ are finite collections of expressions
 572 x such that $x : \tau$ holds for each element.

573 The empty collection $\emptyset_\tau : \mathbf{Set}_\tau$ and singleton collections $\{x\} : \mathbf{Set}_\tau$ are
 574 admissible whenever $x : \tau$. No semantic interpretation of τ as a universe

of elements is assumed; types function solely as structural labels governing
admissibility.

Assumption A3 (Extensionality). For $A, B \in \text{Set}_\tau$,

$$A = B \iff \forall x : \tau, (x \in A \iff x \in B).$$

Assumption A4 (Functions and composition). If $f : \tau \rightarrow \sigma$ and
 $g : \sigma \rightarrow \rho$, then $g \circ f : \tau \rightarrow \rho$ is well-defined. For each τ there exists
 $id_\tau : \tau \rightarrow \tau$.

Assumption A5 (Relations). Relations are predicates returning truth
values on well-typed arguments.

Assumption A6 (Equality and compatibility). For each type τ , equal-
ity $=_\tau$ is an equivalence relation on entities of type τ . A procedure $f : \tau \rightarrow \sigma$
is said to be *type compatible* if

$$x =_\tau y \Rightarrow f(x) =_\sigma f(y)$$

whenever both sides are defined. No assumption is made that all proce-
dures are equality-compatible. This assumption is particularly important for
representative selection (see in the next section).

5.2 Core Cognitive Schemas and Algorithmic Support

The cognitive algebra introduced in this paper provides a small set of repre-
sentational primitives and foundational schemas. From these, a wide range
of more complex cognitive schemas can be constructed. In this section, we
briefly revisit a few core schemas introduced in earlier work and clarify how
they are supported within the present framework. The list is not intended
to be exhaustive; rather, it identifies essential schemas that recur across
problem-solving tasks.

In earlier work, we introduced the schemas of *generalization*, *abstraction*,
and *extension* in a coordination-based setting. In the present paper, we do
not redevelop these schemas in detail. Instead, we indicate how they arise
naturally from the primitives and foundational schemas introduced above.

601 Abstraction is realized by combining invariance-based equivalence with
 602 representative selection. By selecting a representative for each equivalence
 603 class, irrelevant distinctions are suppressed while task-relevant structure is
 604 preserved. Abstraction therefore operates at the level of representation rather
 605 than at the level of inference or decision-making.

606 Generalization concerns the construction of a joint description that covers
 607 multiple classes (cf. [5]). Suppose we are given a collection of subsets $\Omega_j \subseteq \Omega$,
 608 $j = 1, \dots, J$, each corresponding to a class of world states. Within the present
 609 framework, each subset Ω_j induces a family of distinctions, represented by
 610 those coordinates or attributes that take fixed values across all states in Ω_j .

611 The joint generalization is obtained by identifying the distinctions that
 612 are shared by all observed subsets. Formally, let S_j denote the set of indices
 613 of coordinates whose values are constant over Ω_j , and define

$$S^* = \bigcap_{j=1}^J S_j.$$

614 The generalized class Ω^* is then defined as the set of states that satisfy all
 615 shared distinctions specified by S^* . In this construction, S^* represents the
 616 joint abstraction of the observed classes, while Ω^* is the generalized class
 617 determined by the shared distinctions.

618 Extension corresponds to enriching the active conceptual structure when
 619 existing distinctions are insufficient. Within the 2G framework, this is real-
 620 ized by introducing new concept nodes or relations in the Concept Graph,
 621 together with contexts that support the new distinctions. Extension increases
 622 representational capacity and is therefore treated conservatively.

623 A particularly important higher-level operation is *change of representa-*
 624 *tion*. This refers to restructuring the representational space itself, for example
 625 by introducing new coordinates, redefining contexts, or shifting between al-
 626 ternative invariant representations. A full account of change of representation
 627 is not developed here. In the present paper it appears only in its structural
 628 role: as a higher-level operation built from context refinement, projection,
 629 representative selection, and extension.

630 **Algorithmic primitives.** To isolate the structural contribution of the
 631 framework, we assume access to a small collection of standard algorithmic
 632 primitives. These primitives are not learned and are not domain-specific;

633 they serve only to operate on the representational structures provided by the
634 Knowledge Space. The assumed classes of primitives include:

- 635 • *Graph primitives*: traversal, reachability queries, and bounded search
636 on the Concept and Procedure Graphs;
- 637 • *Representative selection primitives*: procedures that select a concrete
638 representative for equivalence classes induced by recognized invariances;
- 639 • *Abstraction primitives*: projection and elimination operations that sup-
640 press distinctions or concept nodes that are irrelevant within a given
641 context;
- 642 • *Clustering primitives*: fixed, parametrized grouping procedures used to
643 expose repeated or similar structure, without adaptive learning;
- 644 • *Set manipulation primitives*: union, intersection, and set difference,
645 applied to sets of concepts, cells, or relations;
- 646 • *Set decision primitives*: tests such as emptiness, membership, and sub-
647 set inclusion;
- 648 • *Boolean primitives*: conjunction, disjunction, and negation, applied to
649 the outputs of predicates or decision primitives.

650 The collection of primitives listed above is intentionally minimal. It suf-
651 fices for the structural constructions used in the present paper and for the
652 finite weak-solver demonstration in Appendix A. The primitives should not
653 be read as general programming operations alone. Their cognitive role is
654 fixed by the structures on which they act: contexts, feature coordinates, con-
655 cept nodes, procedure nodes, and solution traces. In this restricted setting,
656 they support three operations that are used explicitly below: direct applica-
657 tion of available procedures, bounded composition of stored procedures, and
658 bounded generation followed by testing.

659 **6 Problem Solving with Simple Solvers on** 660 **2G**

661 We describe how basic problem-solving behavior can be expressed within
662 the 2G framework using only simple solvers and the primitives above. In

663 this paper, a *problem* is the task of identifying a valid and compact rule or
 664 transformation consistent with given constraints. We do not address long-
 665 horizon planning or planning-and-control.

666 The informal examples in this section are complemented by the finite
 667 demonstration in Appendix A. There the task format, initial Knowledge
 668 Space, solver stages, update rules, worked trace, and failure conditions are
 669 specified explicitly. The examples below therefore serve to motivate the rep-
 670 resentational operations, while the appendix gives the corresponding testable
 671 weak-solver mechanism.

672 Across the examples, problem solving follows the same abstract pattern:

- 673 1. Input is mapped into a structured representation, creating a transient
 674 subgraph of the Concept Graph within the Knowledge Space.
- 675 2. Cognitive primitives and schema operate on this structure, producing
 676 alternative representations through invariance recognition, representa-
 677 tive selection, or contextual projection.
- 678 3. A fixed and deliberately weak solver evaluates admissible representa-
 679 tions under explicit constraints and instantiates a valid solution when
 680 one exists.

681 The solver itself does not change across tasks. Any reduction in complex-
 682 ity or increase in generality is attributed to the structure imposed by the
 683 representation and the context in which the task is formulated.

684 6.1 Example 1: Permutation invariance

685 **Task.** Let $x = (x_1, \dots, x_n)$ be a finite sequence of symbols drawn from a
 686 finite alphabet. The task is to decide whether the sequence contains at least
 687 two identical elements:

$$f(x) = \mathbf{1}\{\exists i \neq j : x_i = x_j\}.$$

688 Observation - The order of elements is irrelevant for this task. A raw
 689 representation treats different permutations of x as distinct inputs, which
 690 leads to unnecessarily complex descriptions.

691 Invariance recognition. - Permutation invariance induces an equivalence
 692 relation on sequences: $x \sim y$ if y is obtained from x by a permutation.

693 Representative selection. - Once this invariance is recognized, a represen-
694 tative may be selected for each equivalence class.

695 In this example, sorting the sequence provides a simple choice of proper
696 representation, but alternative selections (e.g. multiset encodings) would
697 serve the same role.

698 This example serves only to illustrate how representational refinement, via
699 invariance recognition and representative selection, changes which solutions
700 are expressible in a given context, without altering the solver itself.

701 6.2 Example 2: Invariance under nuisance coordinates

702

703 **Task.** Let $x = (x_1, \dots, x_d) \in \{0, 1\}^d$. There exists an unknown subset
704 $S \subset \{1, \dots, d\}$ such that

$$f(x) = \bigoplus_{i \in S} x_i,$$

705 where $\oplus$ denotes XOR. The task is to recover or evaluate the governing law
706 from data.

707 In this task, only the coordinates indexed by a subset S influence the out-
708 put $f(x)$. All remaining coordinates are irrelevant to the task and introduce
709 variation that does not participate in any dependency between inputs and
710 outputs.

711 This situation gives rise to an invariance: changing coordinates outside
712 S does not affect the value of $f(x)$. Accordingly, an equivalence relation is
713 induced on the task context, where two inputs are equivalent if they agree
714 on all coordinates in S .

715 Representative selection fixes a concrete encoding for each equivalence
716 class. A natural choice is to project inputs onto the coordinates in S , discard-
717 ing all others. This operation constitutes a choice of proper representation;
718 its cognitive content lies in the restriction of admissible distinctions, not in
719 the specific projection mechanism.

720 Within the reduced representation, admissible solutions can be expressed
721 as relations over $|S|$ variables. The same solver applied to the unreduced
722 context must account for irrelevant variation, whereas in the refined context
723 this variation is structurally excluded.

724 As in the previous example, the task specification itself does not identify
725 relevant coordinates. The reduction in representational complexity follows
726 entirely from the application of cognitive primitives at the level of contexts,
727 independently of how the relevant subset S is discovered. This example can
728 also be encoded as a Boolean decision table. In that encoding, the task is
729 closely related to the classical rough-set reduct problem: one seeks a subset
730 of attributes that preserves the distinctions needed for the decision output.
731 The present example is not intended to introduce a new reduct algorithm. Its
732 role is to show the same structural issue in task-context language: nuisance
733 coordinates create distinctions that are irrelevant for the task, and a suitable
734 task-relative representation suppresses them.

735 7 Discussion and Conclusion

736 This work proposes a minimal structural framework for organizing represen-
737 tations so that basic cognitive primitives become explicit and usable. The
738 contribution is structural rather than algorithmic.

739 Our first claim is that the cognitive algebra and the associated struc-
740 tural principles introduced here form a sufficient basis for a broad class
741 of representation-level operations relevant to problem solving. Invariance
742 recognition and representative selection provide a minimal mechanism for
743 suppressing irrelevant distinctions and exposing simpler task-relevant regu-
744 larities. Within 2G, these operations are localized as procedures acting on
745 structured conceptual content.

746 In the present paper this claim is developed at the structural level: the
747 weak-solver demonstration shows how stored procedures, composition, and
748 bounded generation provide a minimal instance of such construction.

749 Our third claim concerns the growth of the Knowledge Space under a
750 weak solver. The paper does not assume a strong learning mechanism. In-
751 stead, Appendix A gives a finite demonstration in which the solver starts
752 from an explicit K_0 , applies primitive procedures, composes stored proce-
753 dures, generates bounded candidate extensions, and stores only successful
754 procedures and traces. This gives a minimal in-paper account of procedural
755 reuse and representational extension, while keeping the solver weak and the
756 success conditions explicit.

757 The examples are intentionally limited. They are meant to illustrate
758 structural effects. The solver is fixed and weak throughout, emphasizing that

759 gains arise from the representation structure rather than from algorithmic
760 sophistication.

761 **Contributions and scope.**

762 The main contribution of this paper is the formalization of *context* as
763 the foundational unit of distinction underlying representation, abstraction,
764 and problem formulation. Contexts are defined as partitions of subsets of
765 the state space and provide a uniform mechanism for expressing relevance,
766 refinement, and task-specific structure.

767 Building on this foundation, we introduce a minimal structural frame-
768 work—the Knowledge Space with two coupled graphs (2G)—that situates
769 concepts and procedures within a single representational scaffold. In this
770 framework, the Concept Graph serves as a *host structure* for concepts that
771 are constructed, refined, and reused during learning and problem solving,
772 rather than as a fixed ontology.

773 Classical cognitive notions such as invariance, representative choice, ab-
774 straction, and generalization are not proposed as novel algorithms. Instead,
775 they are placed within a unified structural setting that makes their represen-
776 tational roles and interactions explicit. In the present paper these operations
777 are used in two ways. First, they organize the examples of representational
778 simplification. Second, they provide the primitives and schema used in the
779 finite weak-solver demonstration. The focus is therefore not on a complete
780 architecture for learning or control, but on a testable representational mech-
781 anism by which simple problem solving becomes possible with weak solvers.

782 The issue of context–task mismatch becomes central for complex problem
783 solving. As problem complexity increases, failures often arise not from the
784 absence of a solution, but from posing the task in a context whose distinctions
785 are incompatible with the available data or with the question being asked.
786 In the present framework, such situations correspond to ill-posed tasks: no
787 admissible solution can be expressed within the current context or at its cho-
788 sen granularity. Recognizing ill-posedness and revising contexts—by chang-
789 ing granularity, reformulating the task, or introducing new distinctions—are
790 therefore essential higher-level operations.

791 For classical perspectives on problem simplification, see Polya.[39] In the
792 present framework, simplification corresponds to representational operations
793 such as fixing coordinates, restricting attention to a context cell, or moving
794 to a coarser partition.

795 8 Additional Contextual Examples

796 Additional illustrative examples highlighting context choice and representa-
797 tional form are provided in the appendix.

798 8.1 Example 3: Context choice and representational 799 form

800 Consider finite sequences of symbols drawn from a fixed alphabet. We com-
801 pare two contexts defined on the same underlying class.

802 In the context $\mathcal{C}_{\text{pos}}$, sequences are distinguished by symbol identity at
803 each position. Two sequences belong to different cells whenever they differ
804 at any position. In the context $\mathcal{C}_{\text{mult}}$, sequences are distinguished only by
805 symbol multiplicities. Sequences that differ by permutation belong to the
806 same cell. Thus, $\mathcal{C}_{\text{mult}}$ is a coarsening of $\mathcal{C}_{\text{pos}}$.

807 The two contexts admit different forms of representation. In $\mathcal{C}_{\text{pos}}$, struc-
808 ture is expressed in terms of positions and ordering. In $\mathcal{C}_{\text{mult}}$, structure is
809 expressed in terms of counts or set membership.

810 This example shows that context choice fixes which distinctions are ad-
811 missible, independently of the underlying objects.

812 8.2 Example 4: Local versus uniform context construc- 813 tion

814 Consider a class of objects described by multiple attributes. One context
815 assigns values in a fixed coordinate system shared across all objects. Each
816 coordinate induces a partition of the underlying class, and the common re-
817 finement of these coordinate partitions yields the full context.

818 An alternative context is constructed by successive local refinements.
819 Starting from a coarse partition, a cell is refined only when a finer distinc-
820 tion is required for the current purpose. The resulting context is naturally
821 represented by a refinement tree: internal nodes correspond to intermediate
822 cells, and leaves correspond to the final cells of the induced partition.

823 Both constructions define valid contexts over the same underlying class,
824 but they differ in how distinctions are introduced and organized. A uni-
825 form coordinate system commits to a fixed representational interface from

826 the start. Local refinement introduces distinctions selectively and keeps the
827 refinement history explicit.

828 This example isolates a structural choice: representing a context as a
829 single uniform coordinate product versus representing it as a refinement pro-
830 cess. Even when the resulting partitions coincide, the induced organization
831 of distinctions (and the associated procedures that traverse or update them)
832 need not be the same.

833 **8.3 Finite weak solver and explicit demonstration.**

834 To make the constructive mechanism explicit, Appendix A gives a finite weak
835 solver and a fixed curriculum of tasks. The solver does not receive images
836 and does not perform visual segmentation. It receives segmented task codes,
837 where objects are already represented as finite tuples of feature values. The
838 relevance of the coordinates is not given to the solver. Starting from a small
839 initial Knowledge Space containing only primitive operations, the solver pro-
840 ceeds in three stages: direct application of primitives or stored procedures,
841 bounded composition of known procedures, and bounded generation and test-
842 ing of new admissible items. The appendix gives the corresponding update
843 rules for the Knowledge Space and includes worked traces showing how suc-
844 cessful procedures and solution traces are stored. It also gives propositions
845 and proofs for direct separation, reuse, bounded compositional solution, fail-
846 ure under bounded search, and under-determination. These results do not
847 claim general problem-solving power. Their role is to make the weak solver
848 exact: its success conditions and failure conditions are both stated explicitly.

849 **Declaration of Interest**

850 The authors declare that they have no known competing financial interests
851 or personal relationships that could have appeared to influence the work
852 reported in this paper.

853 Appendix

854 A Constructive demonstration with a finite 855 weak solver

856 We present a finite weak solver and a fixed curriculum of tasks. The purpose
857 is to show, by explicit pseudocode and worked traces, how the Knowledge
858 Space evolves when only the declared primitives are available. The demon-
859 stration is not intended to provide a strong solver. Its role is to make the
860 mechanism testable: the solver succeeds under stated conditions and fails
861 under stated conditions.

862 A.1 Task code and feature coordinates

863 The input to the solver is a segmented task code. The solver does not re-
864 ceive an image and does not learn segmentation. It receives objects already
865 represented as finite tuples of feature values. (c.f. [40].)

866 An object code is written as

$$x = (x_1, \dots, x_d),$$

867 where each index $i \in \{1, \dots, d\}$ denotes a feature coordinate and x_i is the
868 value observed at that coordinate. The solver may read and compare coor-
869 dinate values, but it does not know their external semantic interpretation.
870 For example, a coordinate may externally correspond to shape, colour, size,
871 or position, but these names are not available to the solver.

872 For a set of coordinates $I \subseteq \{1, \dots, d\}$, let

$$x|_I$$

873 denote the projection of x to the coordinates in I .

874 For example,

875 one-feature object: (a)
876 two-feature object: (a,b)
877 three-feature object: (a,b,c)

878 A task is written as

$$T = (P, N, C),$$

879 where P is a finite set of positive examples, N is a finite set of negative
880 examples, and C is a finite set of candidates. The task source may choose
881 which feature coordinates are relevant, but this information is not given to
882 the solver. Thus the main constraint is

segmentation is given, but relevance of coordinates is not given.

883 **A.2 Initial Knowledge Space**

884 The initial Knowledge Space K_0 contains only primitive operations. In the
885 present demonstration these are:

- 886 • read a segmented task code;
- 887 • read a feature coordinate;
- 888 • compare two coordinate values;
- 889 • test equality;
- 890 • test inequality;
- 891 • project an object to selected coordinates;
- 892 • combine tests by conjunction;
- 893 • enumerate bounded coordinate subsets;
- 894 • apply a test to examples and candidates;
- 895 • store a successful procedure in the Procedure Graph;
- 896 • insert a corresponding class distinction in the Concept Graph when
897 needed;
- 898 • retrieve a stored procedure;
- 899 • apply a stored procedure;
- 900 • store a solution trace.

901 The solver has no hidden rule labels and no oracle access. It does not
902 know names such as shape, fill, circle, class, operation, or rule. These are only
903 external interpretations. All procedures created during the demonstration
904 are anonymous executable tests over feature coordinates.

905 **A.3 Separating coordinate tests**

906 Let I be a subset of feature coordinates. A coordinate subset I is acceptable
907 for a task $T = (P, N, C)$ if all positive examples agree on I , and every
908 negative example differs from this positive pattern on at least one coordinate
909 in I . That is,

$$x|_I = x'|_I \quad \text{for all } x, x' \in P,$$

910 and

$$y|_I \neq x|_I \quad \text{for all } y \in N \text{ and } x \in P.$$

911 An acceptable coordinate subset defines a classifier: an object is accepted if
912 its projection to I agrees with the positive pattern.

913 The solver searches only a bounded family of coordinate subsets. For the
914 demonstration with at most three coordinates this family is

$$\{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}.$$

915 If several acceptable subsets exist, the solver chooses the first one according
916 to a fixed order, for example increasing cardinality and then lexicographic
917 order. If this choice leads to ambiguity among candidates and no feedback
918 resolves it, the task is treated as underdetermined.

919 **A.4 Weak-solver algorithm**

920 Input:

921 segmented task $T = (P, N, C)$
922 current Knowledge Space K_t

923

924 Stage 1:

925 Try one-step solutions:

926 primitives

927 stored procedures

928

929 If one succeeds:

930 store the solution trace

931 stop.

932

933 Stage 2:

934 Try bounded compositions of known items:

```

935         p_i AND p_j
936         p_i after p_j
937         p_i after p_j after p_k
938
939     If one succeeds:
940         store the composed procedure
941         store the solution trace
942         stop.
943
944 Stage 3:
945     Generate a bounded list of admissible new items:
946         new projection
947         new abstraction
948         new class distinction
949         new procedure candidate
950
951     Test each generated item in combination with existing items.
952
953     If one succeeds:
954         insert the generated item into the Knowledge Space
955         store the new procedure
956         store the solution trace
957         stop.
958
959 Failure:
960     If all three stages fail, the task is unsolved in  $K_t$ 
961     under the present bounds.

```

962 This order separates three mechanisms. In Stage 1 the task is already
 963 expressible in K_t . In Stage 2 the task is solvable by recombining known
 964 items. In Stage 3 the task requires a bounded extension of K_t .

Level	Mechanism	Meaning
1	direct application	the task is already expressible in K_t
2	bounded composition	the task is solvable by recombining known items
3	bounded generation	the task requires extension of K_t

966 The solver remains weak throughout. It does not search over all possible
 967 rules. It follows the fixed order

apply $\longrightarrow$ compose $\longrightarrow$ generate bounded candidates and test.

968 **A.5 Knowledge Space update**

969 If a stored or primitive procedure p solves T , then only the trace is added:

$$K_{t+1} = K_t \cup \{\text{trace}(T, p)\}.$$

970 If a bounded composition succeeds, the composed procedure is stored:

$$K_{t+1} = K_t \cup \{p_{\text{new}}, \text{trace}(T, p_{\text{new}})\}.$$

971 Here p_{new} is not a new primitive. It is a stored composition of already avail-
 972 able procedures.

973 If Stage 3 succeeds by using a generated item g_ℓ , the update has the form

$$K_{t+1} = K_t \cup \{g_\ell, p_{\text{new}}, \text{trace}(T, p_{\text{new}})\}.$$

974 Only successful generated items are inserted into the Knowledge Space. Rep-
 975 etition may then stabilize a solution pattern into a concept node or a reusable
 976 procedure node:

repeated successful traces $\longrightarrow$ stable concept or procedure.

977 **A.6 Curriculum**

978 The curriculum begins with feature equality and inequality and then moves
 979 to classification. The early tasks require only direct application of primitives.
 980 Later tasks require composition and finally bounded generation.

Step	Input form	What the weak solver learns
1	one coordinate	equality and inequality of one feature value
2	one-coordinate classification	a procedure separating positive and negative examples
3	two coordinates, one relevant	projection: one coordinate matters, one coordinate is ignored
4	two coordinates, both relevant	conjunction of two coordinate tests
5	three coordinates, one relevant	abstraction over two irrelevant coordinates
6	three coordinates, two relevant	two-coordinate classification procedure
7	three coordinates, all relevant	full conjunction procedure
8	reuse task	application of an already stored procedure
9	combination task	bounded combination of stored procedures
10	extension task	bounded generation and test of a new item

982 A.7 Worked trace

983 A typical trace has the following form.

```

984 K_0:
985     read_code
986     read_coordinate
987     equal
988     not_equal
989     project
990     AND
991     store
992     retrieve
993     apply
994
995 Task T_1:

```

```

996     one-coordinate classification.
997     WS creates proc_001.
998
999 K_1 = K_0 + proc_001 + trace(T_1,proc_001)
1000
1001 Task T_2:
1002     two-coordinate objects, one relevant coordinate.
1003     WS reuses equality and creates proc_002,
1004     which ignores the irrelevant coordinate.
1005
1006 K_2 = K_1 + proc_002 + trace(T_2,proc_002)
1007
1008 Task T_3:
1009     two-coordinate objects, both coordinates relevant.
1010     WS creates proc_003 by conjunction.
1011
1012 K_3 = K_2 + proc_003 + trace(T_3,proc_003)
1013
1014 Task T_4:
1015     three-coordinate objects, two relevant coordinates.
1016     WS creates proc_004 by projection and conjunction.
1017
1018 K_4 = K_3 + proc_004 + trace(T_4,proc_004)
1019
1020 Task T_5:
1021     reuse task.
1022     WS applies proc_004 directly.
1023
1024 K_5 = K_4 + trace(T_5,proc_004)
1025
1026 Task T_6:
1027     combination task.
1028     WS tries bounded combinations of stored procedures
1029     and stores the successful combination as proc_005.
1030
1031 K_6 = K_5 + proc_005 + trace(T_6,proc_005)
1032
1033 Task T_7:

```

1034 no stored procedure or bounded composition succeeds.
 1035 WS generates a bounded list of admissible candidates.
 1036 One generated abstraction g_1 succeeds in combination
 1037 with stored procedures.
 1038
 1039 $K_7 = K_6 + g_1 + \text{proc_006} + \text{trace}(T_7, \text{proc_006})$

1040 This trace shows the intended mechanism. Nothing hidden is given to the
 1041 weak solver. No oracle provides the relevant feature coordinate. Segmenta-
 1042 tion is given, but relevance is not given. The solver uses only primitives in
 1043 K_0 , stored procedures, bounded compositions, and bounded generation. The
 1044 Knowledge Space grows from K_0 to K_t through successful solution traces.

1045 A.8 Success and failure conditions

1046 **Proposition 1** (Direct separation). *Let $T = (P, N, C)$ be a task whose ob-*
 1047 *jects have at most three feature coordinates. Suppose there exists an accept-*
 1048 *able coordinate subset I in the solver's bounded search family, and suppose*
 1049 *that the corresponding classifier selects a unique correct candidate from C .*
 1050 *Then the weak solver finds an acceptable subset, solves the task, stores the*
 1051 *corresponding anonymous procedure, and stores the solution trace.*

1052 *Proof.* The bounded search family is finite and is enumerated by the solver.
 1053 Since I belongs to this family, the solver eventually tests it. By assumption,
 1054 I separates the positive examples from the negative examples and selects
 1055 a unique correct candidate. Therefore the solver accepts the test, applies
 1056 it to the candidates, obtains the correct solution, and stores the resulting
 1057 procedure and trace by the update rule. $\square$

1058 **Proposition 2** (Reuse). *Suppose that a procedure p_I corresponding to a*
 1059 *coordinate subset I has been stored in K_t . If a later task $T' = (P', N', C')$ is*
 1060 *separated by the same coordinate subset I , and if p_I selects a unique correct*
 1061 *candidate from C' , then the weak solver solves T' at Stage 1.*

1062 *Proof.* At Stage 1 the solver tries stored procedures from K_t . Since $p_I \in K_t$,
 1063 it is applied to T' . By assumption, it separates the examples and selects a
 1064 unique correct candidate. Hence the task is solved before the solver enters
 1065 the compositional or generation stages. $\square$

1066 **Proposition 3** (Bounded compositional solution). *Suppose that no single*
1067 *primitive or stored procedure solves T , but there exists a bounded composition*
1068 *of stored procedures that separates the examples and selects a unique correct*
1069 *candidate. Then the weak solver solves T at Stage 2 and stores the successful*
1070 *composition as a new procedure.*

1071 *Proof.* Stage 2 enumerates the permitted finite set of bounded compositions.
1072 Since a successful composition belongs to this set, it is eventually tested. By
1073 assumption, it solves the task. The solver therefore stores the composition
1074 as a new procedure and records the corresponding solution trace. $\square$

1075 **Proposition 4** (Failure under bounded search). *Let $T = (P, N, C)$ be a task.*
1076 *If no primitive, no stored procedure, no permitted bounded composition, and*
1077 *no generated candidate within the prescribed generation bound solves T , then*
1078 *the weak solver fails on T in K_t under the present bounds.*

1079 *Proof.* The solver tests exactly the alternatives allowed in Stages 1–3. By
1080 assumption none of these alternatives solves the task. Hence no successful
1081 update rule is triggered, and the solver returns failure under the present
1082 bounds. $\square$

1083 **Proposition 5** (Underdetermination). *If two admissible procedures are equally*
1084 *compatible with the examples but lead to different candidate choices, and no*
1085 *feedback or additional constraint distinguishes them, then the weak solver*
1086 *cannot determine a unique solution from the given task code.*

1087 *Proof.* The available information does not distinguish between the two admis-
1088 sible procedures. Since they lead to different candidate choices, any selection
1089 between them would require information not present in the task code or in
1090 K_t . Therefore the task is underdetermined for the weak solver. $\square$

1091 These propositions make the demonstration falsifiable. The solver is not
1092 claimed to solve arbitrary tasks. It succeeds when an admissible separating
1093 procedure, composition, or bounded generated extension exists and is tested.
1094 It fails when no such item exists within the declared bounds, or when the
1095 task is underdetermined by the given information.

References

- [1] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2013.
- [2] Francesco Locatello, Stefan Bauer, Mario Lucic, Gunnar Raetsch, Sylvain Gelly, Bernhard Schölkopf, and Olivier Bachem. Challenging common assumptions in the unsupervised learning of disentangled representations. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 4114–4124. PMLR, 2019.
- [3] Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A. Wichmann. Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2(11):665–673, 2020.
- [4] Brenden M. Lake, Tomer D. Ullman, Joshua B. Tenenbaum, and Samuel J. Gershman. Building machines that learn and think like people. *Behavioral and Brain Sciences*, 40:e253, 2017.
- [5] Antal Jakovác and András Telcs. Representation and abstraction. *Mathematics*, 13(10), 2025.
- [6] Allen Newell and Herbert A. Simon. *Human Problem Solving*. Prentice-Hall, Englewood Cliffs, NJ, 1972.
- [7] David Marr. *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. W. H. Freeman, San Francisco, 1982.
- [8] Allen Newell. *Unified Theories of Cognition*. Harvard University Press, 1990.
- [9] John R. Anderson and Christian Lebiere. *The Atomic Components of Thought*. Lawrence Erlbaum Associates, 1998.
- [10] John R. Anderson, Daniel Bothell, Michael D. Byrne, Scott Douglass, Christian Lebiere, and Yulin Qin. An integrated theory of the mind. *Psychological Review*, 111(4):1036–1060, 2004.

- 1127 [11] John E. Laird. *The Soar Cognitive Architecture*. MIT Press, 2012.
- 1128 [12] Ron Sun. *Anatomy of the Mind: Exploring Psychological Mechanisms*
1129 *and Processes with the Clarion Cognitive Architecture*. Oxford Univer-
1130 sity Press, 2016.
- 1131 [13] John E. Laird, Christian Lebiere, and Paul S. Rosenbloom. A standard
1132 model of the mind: Toward a common computational framework across
1133 artificial intelligence, cognitive science, neuroscience, and robotics. *AI*
1134 *Magazine*, 38(4):13–26, 2017.
- 1135 [14] John E. Laird. An analysis and comparison of ACT-R and Soar, 2022.
- 1136 [15] John E. Laird. Introduction to Soar, 2022.
- 1137 [16] Daniel Gall and Thom Frühwirth. An operational semantics for the
1138 cognitive architecture ACT-R and its translation to constraint handling
1139 rules, 2017.
- 1140 [17] Ron Sun, Edward Merrill, and Todd Peterson. From implicit skills to
1141 explicit knowledge: A bottom-up model of skill learning. *Cognitive Sci-*
1142 *ence*, 25(2):203–244, 2001.
- 1143 [18] Frederic C. Bartlett. *Remembering: A Study in Experimental and Social*
1144 *Psychology*. Cambridge University Press, 1932.
- 1145 [19] Marvin Minsky. A framework for representing knowledge. In Patrick H.
1146 Winston, editor, *The Psychology of Computer Vision*, pages 211–277.
1147 McGraw-Hill, 1975.
- 1148 [20] Roger C. Schank and Robert P. Abelson. *Scripts, Plans, Goals and Un-*
1149 *derstanding: An Inquiry into Human Knowledge Structures*. Lawrence
1150 Erlbaum Associates, 1977.
- 1151 [21] Dov M. Gabbay, Odinaldo Rodrigues, and Alessandra Russo. Belief
1152 revision in non-classical logics. *The Review of Symbolic Logic*, 1(3):267–
1153 304, 2008.
- 1154 [22] Steven Phillips and William H. Wilson. Categorical compositionality: A
1155 category theory explanation for the systematicity of human cognition.
1156 *PLoS Computational Biology*, 6(7):e1000858, 2010.

- 1157 [23] Steven Phillips and William H. Wilson. Categorical compositionality ii:
 1158 Universal constructions and a general theory of (quasi-)systematicity in
 1159 human cognition. *PLoS Computational Biology*, 7(8):e1002102, 2011.
- 1160 [24] Karel Lenc and Andrea Vedaldi. Understanding image representations
 1161 by measuring their equivariance and equivalence. *International Journal*
 1162 *of Computer Vision*, 127:456–476, 2019.
- 1163 [25] Taco Cohen and Max Welling. Group equivariant convolutional net-
 1164 works. In *Proceedings of the 33rd International Conference on Machine*
 1165 *Learning*, pages 2990–2999, 2016.
- 1166 [26] Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković.
 1167 Geometric deep learning: Grids, groups, graphs, geodesics, and gauges.
 1168 *arXiv preprint arXiv:2104.13478*, 2021.
- 1169 [27] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation
 1170 learning: A review and new perspectives. *IEEE Transactions on Pattern*
 1171 *Analysis and Machine Intelligence*, 35(8):1798–1828, 2013.
- 1172 [28] Martin Arjovsky, Léon Bottou, Ishaan Gulrajani, and David Lopez-Paz.
 1173 Invariant risk minimization. *arXiv preprint arXiv:1907.02893*, 2019.
- 1174 [29] Judea Pearl. *Causality: Models, Reasoning, and Inference*. Cambridge
 1175 University Press, Cambridge, 2 edition, 2009.
- 1176 [30] Peter Gärdenfors. *Conceptual Spaces: The Geometry of Thought*. MIT
 1177 Press, Cambridge, MA, 2000.
- 1178 [31] Zdzisław Pawlak. Rough sets. *International Journal of Computer and*
 1179 *Information Sciences*, 11(5):341–356, 1982.
- 1180 [32] Zdzisław Pawlak. *Rough Sets: Theoretical Aspects of Reasoning about*
 1181 *Data*, volume 9 of *Theory and Decision Library, Series D*. Kluwer Aca-
 1182 demic Publishers, Dordrecht, 1991.
- 1183 [33] Andrzej Skowron and Cecylia Rauszer. The discernibility matrices and
 1184 functions in information systems. In Roman Słowiński, editor, *Intelligent*
 1185 *Decision Support: Handbook of Applications and Advances of the Rough*
 1186 *Sets Theory*, volume 11 of *Theory and Decision Library*, pages 331–362.
 1187 Kluwer Academic Publishers, Dordrecht, 1992.

- 1188 [34] Rudolf Wille. Restructuring lattice theory: An approach based on hi-
 1189 erarchies of concepts. In Ivan Rival, editor, *Ordered Sets*, volume 83
 1190 of *NATO Advanced Study Institutes Series C: Mathematical and Physi-
 1191 cal Sciences*, pages 445–470. D. Reidel Publishing Company, Dordrecht,
 1192 1982.
- 1193 [35] Bernhard Ganter and Rudolf Wille. *Formal Concept Analysis: Mathe-
 1194 matical Foundations*. Springer, Berlin, 1999.
- 1195 [36] John F. Sowa. *Conceptual Structures: Information Processing in Mind
 1196 and Machine*. Addison-Wesley, Reading, MA, 1984.
- 1197 [37] Bernhard Ganter and Sergei O. Kuznetsov. Pattern structures and their
 1198 projections. In *Conceptual Structures: Broadening the Base*, volume
 1199 2120 of *Lecture Notes in Computer Science*, pages 129–142. Springer,
 1200 Berlin, 2001.
- 1201 [38] Yiyu Yao. Granular computing: Past, present and future. In *2008
 1202 IEEE International Conference on Granular Computing*, pages 80–85,
 1203 Hangzhou, China, 2008. IEEE.
- 1204 [39] George Pólya. *How to Solve It: A New Aspect of Mathematical Method*.
 1205 Princeton University Press, Princeton, NJ, 1945.
- 1206 [40] Chi Zhang, Feng Gao, Baoxiong Jia, Yixin Zhu, and Song-Chun Zhu.
 1207 RAVEN: A dataset for relational and analogical visual reasoning. *arXiv
 1208 preprint arXiv:1903.02741*, 2019.